Improved Congestion Control for Packet Switched Data Networks and the Internet

Aun Haider

A thesis presented for the degree of
Doctor of Philosophy
in
Electrical and Electronic Engineering
at the
University of Canterbury,
Christchurch, New Zealand.

1 March 2004

ABSTRACT

Congestion control is one of the fundamental issues in computer networks. Without proper congestion control mechanisms there is the possibility of inefficient utilization of resources, ultimately leading to network collapse. Hence congestion control is an effort to adapt the performance of a network to changes in the traffic load without adversely affecting users perceived utilities. This thesis is a step in the direction of improved network congestion control.

Traditionally the Internet has adopted a best effort policy while relying on an end-to-end mechanism. Complex functions are implemented by end users, keeping the core routers of network simple and scalable. This policy also helps in updating the software at the users' end. Thus, currently most of the functionality of the current Internet lie within the end users' protocols, particularly within Transmission Control Protocol (TCP). This strategy has worked fine to date, but networks have evolved and the traffic volume has increased many fold; hence routers need to be involved in controlling traffic, particularly during periods of congestion. Other benefits of using routers to control the flow of traffic would be facilitating the introduction of differentiated services or offering different qualities of service to different users. Any real congestion episode due to demand of greater than available bandwidth, or congestion created on a particular target host by computer viruses, will hamper the smooth execution of the offered network services. Thus, the role of congestion control mechanisms in modern computer networks is very crucial.

In order to find effective solutions to congestion control, in this thesis we use feedback control system models of computer networks. The closed loop formed by TCP/IP between the end hosts, through intermediate routers, relies on implicit feedback of congestion information through returning acknowledgements. This feedback information about the congestion state of the network can be in the form of lost packets, changes in round trip time and rate of arrival of acknowledements. Thus, end hosts can either execute reactive or proactive congestion control mechanisms. The former approach uses duplicate acknowledgements and timeouts as congestion signals, as done in TCP Reno, whereas the latter approach depends on changes in the round trip time, as in TCP Vegas. The protocols employing the second approach are still in their infancy as they cannot co-exist safely with protocols employing the first approach. Whereas TCP Reno and its mutations, such as TCP Sack, are presently widely used in computer networks, including the current Internet. These protocols require packet losses to happen before they can detect congestion, thus inherently leading to wastage of time and network bandwidth.

Active Queue Management (AQM) is an alternative approach which provides congestion feedback from routers to end users. It makes a network to behave as a sensitive closed loop

feedback control system, with a response time of one round trip time, congestion information being delivered to the end host to reduce data sending rates before actual packets losses happen. From this congestion information, end hosts can reduce their congestion window size, thus pumping fewer packets into a congested network until the congestion period is over and routers stop sending congestion signals.

Keeping both approaches in view, we have adopted a two-pronged strategy to address the problem of congestion control. They are to adapt the network at its edges as well as its core routers.

We begin by introducing TCP/IP based computer networks and defining the congestion control problem. Next we look at different proactive end-to-end protocols, including TCP Vegas due to its better fairness properties. We address the incompatibility problem between TCP Vegas and TCP Reno by using ECN based on Random Early Detection (RED) algorithm to adjust parameters of TCP Vegas. Further, we develop two alternative algorithms, namely optimal minimum variance and generalized optimal minimum variance, for fair end-to-end protocols. The relationship between $(p,\ 1)$ proportionally fair algorithm and the generalized algorithm is investigated along with conditions for its stable operation. Noteworthy is a novel treatment of the issue of transient fairness. This represents the work done on congestion control at the edges of network.

Next, we focus on router-based congestion control algorithms and start with a survey of previous work done in that direction. We select the RED algorithm for further work due to it being recommended for the implementation of AQM. First we devise a new Hybrid RED algorithm which employs instantaneous queue size along with an exponential weighted moving average queue size for making decisions about packet marking/dropping, and adjusts the average value during periods of low traffic. This algorithm improves the link utilization and packet loss rate as compared to basic RED. We further propose a control theory based Auto-tuning RED algorithm that adapts to changing traffic load. This algorithm can clamp the average queue size to a desired reference value which can be used to estimate queueing delays for Quality of Service purposes.

As an alternative approach to router-based congestion control, we investigate Proportional, Proportional-Integral (PI) and Proportional-Integral-Derivative (PID) principles based control algorithms for AQM. New control-theoretic RED and frequency response based PI and PID control algorithms are developed and their performance is compared with that of existing algorithms. Later we transform the RED and PI principle based algorithms into their adaptive versions using the well known square root of p formula. The performance of these load adaptive algorithms is compared with that of the previously developed fixed parameter algorithms.

Apart from some recent research, most of the previous efforts on the design of congestion control algorithms have been heuristic. This thesis provides an effective use of control theory principles in the design of congestion control algorithms. We develop fixed-parameter-type feedback congestion control algorithms as well as their adaptive versions. All of the newly

proposed algorithms are evaluated by using ns-based simulations.

The thesis concludes with a number of research proposals emanating from the work reported.

ACKNOWLEDGEMENTS

I am very grateful to my research supervisors, Assoc. Professor Dr. Harsha Sirisena and Assoc. Professor Dr. Krzysztof Pawlikowski for their guidance and encouragement to carry out this research endeavour. Their brilliant ideas and suggestions helped me to improve the presentation of this thesis.

Many thanks to all members of Networking and Simulation research groups of Electrical and Computer Engineering (ECE) and Computer Science (COSC) Departments for their discussions during weekly meetings of the groups. My special thanks are for Dr. Greg Ewing for his kind help and suggestions whenever glitches in simulations software and scripts seemed to me impossible to compile. Many thanks are to Assoc. Professor Dr. Tim Bell, Head of COSC, for allowing me to use all of COSC facilities so generously. I wish to thank all computer programmers in ECE and COSC for their help during installing ns simulator and solving other problems related with operating systems.

I would like to thank my father Assoc. Professor Munir Hussain Bukhari and my mother Umay Kalsoom, for all good things they gave to me. I am also thankful to my younger brothers Engr. Mohsan Mehdi, Dr. Aqueel Imran and my sisters Qurat-ul-Anne, Narjis and Fizza for their encouragement and support.

I would like to thank David Rankine and Sven Ostring for their help during the first year of my Ph.D studies, Paul Johnstone for many trips around Christchurch and Regan Anderson for his hospitality. I am greatly indebted to the Government of New Zealand for providing me a great opportunity to do Ph.D in Electrical and Electronic Engineering at University of Canterbury Christchurch New Zealand by giving me full financial support.

I am also thankful to the Royal Society of New Zealand, IEEE Chapter of the University of Canterbury and Department of Computer Science for providing me travel grants to attend the IEEE Globecom 2002 conference at Taipei Taiwan. I want to express my acknowledgements to ACM SIGCOMM Student Travel Awards Committee who provided me full financial support for attending SIGCOMM 2002 in Pittsburgh, Pennsylvania, U.S.A.

In the end, I would like to say my heartiest words of thanks to all of my friends at Ilam Village. They made my life so easy and enjoyable in flat 39, 12, 15 and 20, during the last three and half years. Furthermore, I am thankful to every one who helped me to achieve this milestone in my academic career.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

ABR	Available Bit Rate
ACK	Acknowledgement
ACKs	Acknowledgements
AIAD	Additive Increase Additive Decrease
AIMD	Additive Increase Multiplicative Decrease
ATM	Asynchronous Transfer Mode
ARIMA	Autoregressive Integrated Moving Average
BSD	Berkeley Software Distribution
B, BW	Bandwidth
CA	Congestion Avoidance
CBR	Constant Bit Rate
W, cwnd	Congestion Window
ECN	Explicit Congestion Notification
EWMA	Exponential Weighted Moving Average
FACK	Forward Acknowledgement
FIFO	First In First Out
FTP	File Transfer Protocol
FR	Fast Retransmit
GM	Gain Margin
http	Hyper Text Transfer Protocol
IP	Internet Protocol
ISO	International Standards Organization
LMS	Least Mean Square
Mbps	Mega bits per second
MIMD	Multiplicative Increase Multiplicative Decrease
M_s, MSS	Maximum Segment Size
NS	Network Simulator
OSI	Open Systems Interconnection
p	Loss Probability
PI	Proportional Integral
PD	Proportional Derivative
PID	Proportional Integral Derivative
PM	Phase Margin

QoS	Quality of Service
RED	Random Early Detection
REM	Random Exponential Marking
RFC	Request For Comments
RIO	Random Early Drop with in/out bit
R, RTT	Round Trip Time
R_s	Smoothed RTT
R_{to}, RTO	Retransmission Time Out
SACK	Selective Acknowledgement
SDE	Stochastic Differential Equation
SMSS	Sender Maximum Segment Size
SS	Slow Start
TCP	Transmission Control Protocol
TTL	Time to live
UDP	User Datagram Protocol
Var	Variance

Chapter 1

INTRODUCTION

Today's computer networks such as the Internet form an essential component of modern civilization. They are responsible for a variety of services in our society and their importance is unquestioned and unparalleled. In order to get maximum benefit at the least cost, these networks should be well designed and operated in an optimized manner. Though most computer networks work well under light load, congestion problems start to occur under heavy load. Therefore avoiding and controlling congestion is one of the most critical issues in present and future networks. This dissertation presents some novel approaches to the design of protocols and router algorithms for congestion avoidance and control.

1.1 HISTORICAL PERSPECTIVE

The launching of Sputnik by USSR in 1957 prompted the Advanced Research Projects Agency (ARPA) under the Department of Defence (DoD), USA, to start work on the ARPANET project in the mid 1960's, with the aim to provide a command and control computer network which could survive a nuclear attack, [Tanenbaum 1996]. ARPA decided to adopt the concept of packet-switched networks consisting of subnets and host computers. These efforts eventually connected hundreds of Universities and Government installations using leased telephone lines. Under these circumstances the Transmission Control Protocol (TCP) and Internet Protocol (IP) were first designed in the early 70's to connect the multiple networks in a seamless way.

These two protocols were defined in [Cerf and Kahn 1974] and in [Leiner *et al.* 1985], as a central part of the TCP/IP architecture to connect different networks through packet switching technology. TCP and IP lie in the Transport and Internet layers of TCP/IP model, respectively, as shown in Figure 1.1, [Tanenbaum 1996]. The Open System Interconnection (OSI) model, shown in Figure 1.1, was proposed in 1983 by the International Standards Organization (ISO) for the standardization of computer network protocols. However, the OSI protocols have not become popular despite their usefulness.

On the other hand, TCP/IP protocols were widely used initially and the model was practically nonexistent in the marketplace [Tanenbaum 1996]. Later this architecture became known as the DoD reference model, [Spragins *et al.* 1991]. It is also widely known as the TCP/IP

Reference Model, [Tanenbaum 1996]. Thus, in TCP/IP the protocols came first and reference model came later.

In the TCP/IP model, the transport layer contains TCP and User Datagram Protocol (UDP) as end-to-end protocols. The major aims of TCP are to enhance the best-effort service provided by IP and to control the flow of packets so as not to overload either the network or the receiving host. TCP implements end-to-end window based flow control [Postel 1981] which is in contrast to rate-based flow control where sources transmit data continuously at suitable rate e.g. the Available Bit Rate (ABR) service in ATM networks. UDP is an unreliable and connectionless protocol, mainly used for one-shot client server type request-reply queries and applications where prompt delivery is more important than accurate delivery, such as transmitting speech and video [Tanenbaum 1996]. The Internet layer is a linchpin which holds together the whole

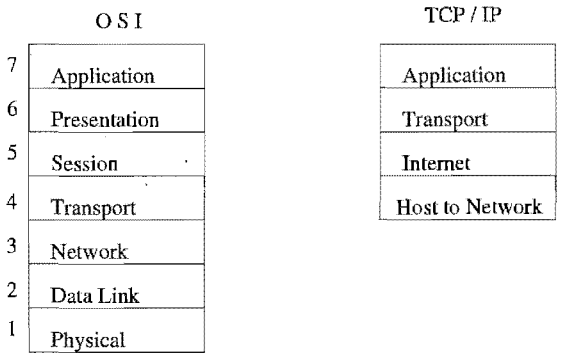

Figure 1.1 OSI and TCP/IP architecture.

architecture of TCP/IP by defining the packet format, IP protocol and implementing a guaranteeless simple datagram delivery service; its major design issues are packet routing and congestion avoidance, [Spragins *et al.* 1991] and [Tanenbaum 1996].

At the time of birth of TCP, receivers were the main bottleneck in the network and senders were required not to send more than the capacity of the receiver buffer. A window was used for sending data whose value was advertised at the time of connection set-up [Postel 1981]. The sender was not allowed to transmit more than the window size of data before the receipt of an acknowledgement. Computer networks used this type of TCP for many years to get a reliable, connection-oriented and in-order service at the rate estimated by the sender and receiver buffer size. In this set-up the capacity of the network was not taken into account. The sender always tried to put more data into the network whenever it found an empty space in the receiver buffer. With continuous growth of the Internet in its size and utilization, such activities lead to network saturation. This resulted in congestion collapse of the Internet in October 1986, which was fixed by Van Jacobson et al. by introducing a new state variable called *congestion window* which interacts with the receiver advertised window (minimum of two) to send the right amount of data into the network, [Jacobson 1988].

To avoid future congestion problems, network researchers came up with a set of seven new algorithms for TCP (version 4BSD) which were: round-trip variance estimation, exponential re-

transmit timer backoff, slow start, more aggressive receiver acknowledgement policy, dynamic window sizing on congestion, Karn's clamped retransmit backoff and fast retransmit, [Jacobson 1988]. The first five of them were based on the fact that the integral of packet density around the sender-receiver-sender loop is constant, which is also known as the packet conservation principle, [Jacobson 1988]. Karn's clamped retransmit algorithm recommends that acknowledgements received for retransmitted packets should not be used for computation of the round trip time [Karn and Partridge 1987]. The last algorithm, i.e. Fast Retransmit, will be explained in detail in the proceeding Chapter 2. Presently all of these seven algorithms are part and parcel of the most commonly used Reno version of TCP.

The variants of these algorithms have resulted in the evolution of different other versions of TCP such as New Reno, Selective Acknowledgement (SACK) and TCP Vegas. A lot of research had been done on congestion control algorithms and performance modelling of TCP by using analytical, simulations and test bed measurements techniques since the first collapse of the Internet in 1986. In the light of these results appropriate changes have been made in TCP which will be briefly discussed in next Chapter. Although, the congestion control problem had been historically addressed mainly through the end host protocols but in the future router mechanisms may be required to complement them. One of the first effort to involve routers in the congestion control is the Random Early Detection (RED) algorithm as proposed in [Floyd and Jacobson 1993]. It tries to alleviate the congestion by randomly dropping or marking (if Explicit Congestion Notification (ECN) is being employed) the packets at router. Recently, a number of researchers have proposed different other schemes for router-based congestion control which will be overviewed in Chapter 5.

Due to the fact that presently the Internet users do not belong to a small closely knit community which follows TCP congestion control principles honestly, we can not rely solely on end nodes and developers to incorporate end-to-end congestion control in their Internet applications. Furthermore, with the emergence of new applications in the Internet, there is a large amount of data flows which does not follow the TCP congestion avoidance and control principles; this poses a potential danger to the stability of the Internet. Thus, it can be justified that in future the routers must participate in congestion control for optimizing the overall performance and resource utilization in the networks, [Floyd and Fall 1999]. Therefore new algorithms are required to be developed, both in the end host protocols and in the routers, to cope with future problems and challenges of congestion control in computer networks.

1.2 LOOKING FORWARD

According to an estimate, the world wide number of the Internet users in September 2002 was about 600 million [Nua 2002]. The total number of Internet users in the world swells up every day. This phenomenal growth of use of computer networks in general and the Internet in particular has posed a lot of theoretical and practical problems to be solved.

These huge figures transform into immensely challenging tasks to avoid any future meltdown of the Internet due to congestion in backbone routers. There are always bandwidth hungry applications which will consume all of the available bandwidth. Some of these difficulties in simulation of Internet are pointed out in [Floyd and Paxon 2001].

At present the Internet is a best effort network with no provision of Quality of Service, but in future it might provide different classes of service for different users which will require the design of new and modification of the existing protocols. The emergence of new technologies such as fiber optics, which offer high bandwidth delay product transmission links, also demand radical changes in protocol suite and router design. Efforts have already been started in this direction such as design of HighSpeed TCP in [Floyd 2003] and research on high performance Internet as given in [Internet2 2003]. These new technologies promise to provide full multimedia services to users at very nominal price and are under extensive research presently.

1.3 PROBLEM STATEMENT

This thesis deals with the problem of congestion avoidance and control by considering it as a feedback control system problem for dynamic management of bandwidth in packet switched networks and the Internet. Congestion can be defined as a network state in which the total demand for resources, e.g. bandwidth, among the competing users exceeds the available capacity, leading to packet or information loss and resultantly requiring packet retransmissions. At the time of congestion in a computer network there will be a simultaneous increase in queuing delay, packet loss and number of packet re-transmissions. Thus, congestion will ultimately lead to a drop in a network's throughput which is highly undesirable.

The classic problem of congestion control has been described in [Chiu and Jain 1989] and [Jain 1991], where a computer network is required to be designed to have the operating point at *knee* (associated with number of users n_k) for congestion avoidance, between *knee* and *cliff* (associated with number of users n_n) for congestion control, as shown in Figure 1.2. Where the total load (bis/s) handled by network at knee and cliff points is represented by $l(n_k)$ and $l(n_n)$, respectively, with $l(n_k) < l(n_n)$.

When the quantity of data received at a router's input port exceeds the nominal buffer capacity then congestion occurs and throughput or efficiency of the network decreases drastically. It will require quick remedial measures both at the end hosts and at the routers. Thus, at the time of congestion the end hosts need to decrease their data sending rates (or congestion windows) and routers need to drop packets until the congestion state is relieved. It will approximately take one round trip time for end hosts to reduce their sending rates and thus its effect on a congested router. Dropping of packets at routers will cause a wastage of network resources and thus not very good remedy for congestion control.

A superficial approach to address the congestion control problem through router mechanism might be to simply increase the queue buffer capacity. But it has been proved in [Nagle 1987]

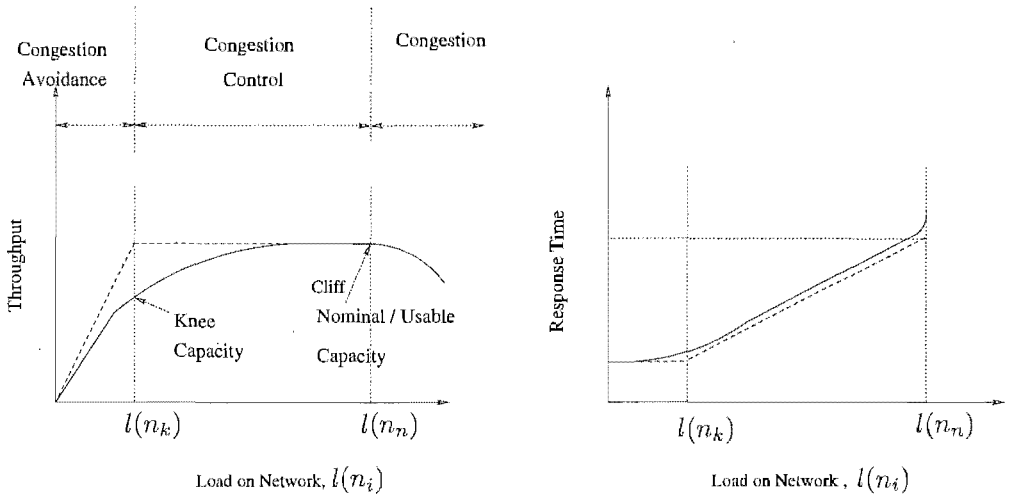

Figure 1.2 Illustration of Congestion Avoidance and Control problem, where $l(n_i)$ is the load (bits/s) generated by n_i number of users.

that the problem of packet loss in congested routers will continue to exist even with the buffers having infinite size. These hypothetical routers with infinite size of buffers will lose packets during congestion due to expiry of the Time To Live (TTL) field and not due to the lack of space.

Thus, better techniques such as Active Queue Management (AQM) have been proposed to solve these problems. In AQM the congested router starts to mark packets as soon as the amount of data in buffer of router's queue exceeds a threshold that is smaller than the maximum buffer size. These marked packets are congestion signals which prompt TCP compliant data sources to reduce their data sending rates (half of the current value of their congestion windows). In practice the AQM based on packet marking is implemented in conjunction with Explicit Congestion Notification (ECN) which defines the logic and format of congestion signals to be transmitted in the headers of data and acknowledgement packets.

In order that AQM based on packet marking should work properly, both sender and receiver of data should be ECN capable otherwise the explicit feedback control loop to be formed by the marked packets will remain open and thus router would be required to use more drastic measures (forming the implicit feedback control loop through packet dropping) to force the reduction in sending rates of data sources. In case of receiving a sudden burst of packets (which causes the total amount of data to exceed the maximum buffer capacity) the router will drop all of the extra packets. Thus, we can control the congestion adequately by providing the sufficient amount of buffer capacity to absorb transient bursts of data and selecting the suitable threshold for packet marking in the router. Alternatively, we can also control congestion by providing some reference or target value of queue size which would be required to be maintained by router. Thus, we need to have algorithms in routers which can start marking packets before congestion actually happens, therefore operating network in the congestion control region shown in Figure 1.2.

Closely related to the problem of congestion control and maintaining high throughput of

network is the issue of fairness. A transport layer protocol which have good congestion control properties but poor fairness is not very useful in practice because it will unduly punish some users of network. Hence, the congestion control schemes are required to be efficient (ensuring high throughput), fair, able to deal with heterogenous environments (scalable), able to deal with mis-behaving flows, as well as be stable and simple to implement.

1.4 FOCUS OF THIS THESIS

This thesis investigates congestion related issues in the present and future wired computer networks using control-theoretic modelling and simulation techniques. It addresses congestion avoidance and control from both end-to-end and router-centric methods, thereby leading to efficient design of so called Active Queue Management. It emphasizes the development of fair end-to-end host protocols to optimize the network performance. Also it analyzes presently used congestion control algorithms in routers and presents a spectrum of new algorithms, thus supporting the notion of router participation in end-to-end congestion control. The newly developed fair end-to-end host protocols and the router based congestion control algorithms are tested by performing simulation experiments using variety of traffic mixtures and network scenarios.

A vast body of literature was studied before starting the research work reported in this thesis. It was soon realized that it is very difficult to encompass the entire area of congestion avoidance and control in the limited period of time. Thus, we restricted the scope our research work to congestion control in unicast wired networks only. However, the concepts developed for unicast and end-to-end wired environments can be further extended to multicast and wireless environments after introducing appropriate modifications.

1.5 SUMMARY OF ASSUMPTIONS

We have made the following assumptions in this thesis:

1. Routers are assumed to be under administrative control but not the end users.
2. Both end users and routers have responsibility for congestion control.
3. The minimum time needed to carry out any control action is one round trip time.
4. Congestion control schemes operate in the region where the bandwidth delay product is larger than the current window size.
5. Congestion control schemes are window based and not rate based.
6. Congestion information is conveyed to sources by packet dropping or marking.
7. Traffic consist of File Transfer Protocol (FTP) over TCP and Exponential ON/OFF or Pareto ON/OFF over UDP.

1.6 CONTRIBUTIONS OF THIS THESIS

This thesis makes the following specific contributions:

1. Development of RED and ECN based fair TCP Vegas algorithm for compatibility with TCP Reno (Chapter 3).

2. Development, analysis and design of optimal minimum variance window congestion control algorithm for TCP (Chapter 4).

3. Development, analysis and design of generalized optimal minimum variance window congestion control algorithm for TCP (Chapter 4).

4. Development of Hybrid RED algorithm (Chapter 5).

5. Development, analysis and design of feedback control theory based Auto-Tuning RED algorithm (Chapter 6).

6. Development, analysis and design of fixed parameters feedback congestion controllers based on RED, PI and PID principles (Chapter 7).

7. Development, analysis and design of adaptive feedback congestion controllers based on RED and PI principles (Chapter 8).

1.7 STRUCTURE OF THESIS

This thesis has nine Chapters and it can be divided into two parts. The first part, Chapter 2 to Chapter 4, deals with changes to be made at the edges of the network or more precisely modifications to TCP for improved congestion control. The second part, Chapter 5 to Chapter 8, presents new router algorithms for congestion control. A brief overview of each Chapter is presented below.

Chapter 1 introduces TCP/IP based computer networks and briefly presents their history and future perspectives. The classical problem of congestion control is stated and explained in general terms. A summary of assumptions and a list of major contributions of the thesis are also presented. Chapter 2 gives a concise and in-depth overview of all major types of TCP. The major algorithms are explained and their strengths and weaknesses are discussed. Then it presents an up-to-date survey of mathematical models of TCP. Next, it describes ECN, Internet 2 and protocols for high bandwidth-delay product networks. The commonly used metrics for TCP performance evaluation are also discussed.

Chapter 3 deals with the design and analysis of fair end-to-end congestion control protocols. A detailed survey of existing fair end-to-end algorithms is presented and TCP Vegas is selected for further development and design of a fair protocol. Some major problems associated with

TCP Vegas are pointed. Among these problems, one of the most crucial is selected (incompatibility of TCP Vegas with TCP Reno for practical deployment in the Internet) and investigated both analytically and by using ns-based simulations. Then, based on the results of simulation experiments, we develop a new algorithm which uses ECN signals from the RED algorithm.

Chapter 4 develops an end-to-end Optimized Minimum Variance window control algorithm. The effect of changing the reference or target queue level on the bandwidth allocation among different TCP connections is investigated. In the second half of this Chapter we develop, analyze and design the Generalized Optimal Minimum Variance window control algorithm. Next its relation to another $(p, 1)$ proportionally fair algorithm is investigated. Furthermore, this Chapter introduces the concept of short term or transient fairness among the competing TCP flows and determines an expression for the system time constant. The effects of different tuning parameters of generalized algorithm and their mutual inter-dependence on its performance are throughly investigated by using ns based simulations.

Chapter 5 begins the second part of thesis in which we employ routers to control congestion. It gives an introduction to AQM and then a survey of algorithms proposed for implementation of AQM in a router. A new Hybrid RED algorithm is developed which employs both Instantaneous and EWMA queue sizes for packet mark/drop decisions during congestion.

Chapter 6 deals with RED-type algorithms which can adapt their tuning parameters with changes in the traffic load. Firstly we review an existing adaptive RED algorithm which is based on heuristic principles. Next, a control theoretic model of AQM based on the RED algorithm is described. Using this model a new Auto-Tuning RED algorithm is developed whose performance is compared with the existing adaptive RED algorithm by using ns-based simulations. The guidelines for selecting the fixed tuning parameters for Auto-Tuning RED algorithm are also presented.

Chapter 7 attacks the problem of router-based congestion control from the closed-loop feedback control system viewpoint. It develops feedback congestion control algorithms for AQM routers supporting TCP flows. A range of control-theoretic designs for RED, Proportional Integral (PI) and Proportional Integral and Derivative (PID) control algorithms are presented and simulated. The convergence of queue level to a setpoint (target level), response time and stability are compared with other existing algorithms.

Chapter 8 extends the work done in Chapter 7 by developing adaptive feedback congestion control algorithms. These adaptive algorithms can adjust their tuning parameters with changes in traffic load. This Chapter develops and analyzes adaptive versions of RED and PI principle based congestion control algorithms. Moreover the ns based simulation results showing the performance of these adaptive algorithms are presented.

Finally, Chapter 9 presents the overall conclusions and indicates future directions of research.

1.8 PUBLICATIONS LIST

The following papers were prepared during the course of this work:

[1] AUN HAIDER, HARSHA SIRISENA, KRZYSZTOF PAWLIKOWSKI AND MICHAEL J. FERGUSON, 'Congestion Control Algorithms in High Speed Telecommunication Networks', *Proceedings of 36th Annual ORSNZ Conference*, pp. 88-97, November 29-December 1, 2001, Christchurch, New Zealand.

[2] HARSHA SIRISENA, MAHBUB HASSAN AND AUN HAIDER, 'Optimal TCP Congestion Control', *Proceedings of the* 9^{th} *International Conference on Telecommunications ICT'2002*, Vol. 1, pp. 732-736, June 23-26, 2002, Beijing, China.

[3] HARSHA SIRISENA, AUN HAIDER AND KRZYSZTOF PAWLIKOWSKI, 'Auto-Tuning RED for Accurate Queue Control', *Proceedings of IEEE Globecom'02*, Vol. 2, pp. 2010-2015, November 17-21, 2002, Taipei, Taiwan.

[4] AUN HAIDER, HARSHA SIRISENA AND KRZYSZTOF PAWLIKOWSKI, 'Improved Congestion Control with Hybrid RED', *Proceedings of the* 10^{th} *IEEE International Conference on Telecommunications ICT'2003*, Vol. 2, pp. 923-928, February 23-March 1, 2003, Tahiti, Papeete, French Polynesia.

[5] AUN HAIDER, HARSHA SIRISENA AND KRZYSZTOF PAWLIKOWSKI, 'A Control-Theoretic Design of RED for Improved Congestion Control in TCP/IP Routers', *Proceedings of the Third International Dynamics of Continuous, Discrete and Impulsive Systems, (DCDIS), Conference on Engineering Applications and Computational Algorithms*, pp. 459-464, May 15-18, 2003, Guelph, Ontario, Canada.

[6] HARSHA SIRISENA, AUN HAIDER, MAHBUB HASSAN AND KRZYSZTOF PAWLIKOWSKI, 'Transient Fairness of Optimized End-to-End Window Control', *Proceedings of IEEE Globecom'03* , vol. 7, pp. 3979-3983, Dec 1-5, 2003, San Francisco, USA.

[7] AUN HAIDER, HARSHA SIRISENA AND KRZYSZTOF PAWLIKOWSKI, 'PID based Congestion Control Algorithm for AQM Routers supporting TCP/IP flows', *Accepted in IEICE Transactions on Communications*, March, 2004.

[8] AUN HAIDER, HARSHA SIRISENA AND KRZYSZTOF PAWLIKOWSKI, 'Fair End-to-End Optimal Minimum Variance Window Control', *Accepted in IEICE Transactions on Communications*, March, 2004.

[9] AUN HAIDER, HARSHA SIRISENA AND KRZYSZTOF PAWLIKOWSKI, 'Interaction between TCP Reno and TCP Vegas in End-to-End Congestion Control', *Proceedings of The International Conference on Information Networking (ICOIN) 2004*, Vol. 1, pp. 55-64, February 18-20, 2004, Busan, Korea.

Chapter 2

TCP AND CONGESTION CONTROL

Congestion control is a major performance/design issue in computer networks and end-to-end architecture of the Internet. Without proper congestion control mechanisms there can be a congestion collapse, which is highly resource wasteful and thus undesirable state of network operation. In a network collapsed by congestion, the amount of data transfer across the bottleneck routers will be nill or very less. In [Floyd and Fall 1999], different types of past and potential future congestion collapse are categorized as: *classical congestion collapse* as given in [Jacobson 1988] and [Nagle 1984], *congestion collapse from undelivered packets, fragmentation based congestion collapse, congestion collapse from increased control traffic* and *congestion collapse due to stale packets.* Further, the wide spread use of the Internet has caused proliferation of substandard versions of TCP such as TCP's having poor implementation, [Paxson *et al.* 1999], and TCP's opening multiple connections aggressively, which can cause congestion. Parallelly, there is also a significant number of non-responsive and TCP non-compatible bandwidth hungry traffic flows in the Internet which can also be source of congestion episodes. Thus, all of these rogue traffic sources are another future threat of chronic congestion in the Internet, [Floyd 2000b]. In this thesis we will be only dealing with potential congestion caused by increased traffic flow or in other words avoiding the potential classical congestion collapse in present and future high speed networks.

The successful performance and robustness of the present Internet is mainly due to TCP which has undergone number of changes in its primitive design, in an evolutionary process, over the last 3 decades. Presently, couple of basic congestion control algorithms, namely slow start, congestion avoidance, fast retransmission and fast recovery, are essential part and parcel of almost every existing and newly proposed versions of TCP's. These congestion control algorithms of TCP are end-to-end based and do not require any support of routers for their functioning.

The basic mechanism which controls the flow of data from most of the TCP source to receiver is additive increase of window of packets during periods of no congestion and multiplicative decrease of that window during time of congestion in network routers. This mechanism is commonly called Additive Increase and Multiplicative Decrease (AIMD) as given in [Chiu and Jain 1989]. In implementations of TCP's, using AIMD principle, the universal additive increase factor is chosen as 1 and decrease factor is chosen as 0.5 , as proposed by means of heuristic analysis in [Jacobson 1988].

When there is congestion or onset of congestion in a bottleneck router then packets will start to drop and no ACK with new sequence number will reach TCP sender which will signal it to decrease its current window size. In general, in the present end-to-end architecture, TCP data sender detects congestion by two mechanisms which are timeout and reception of duplicate ACKs. Detection of packet loss by TCP source will decrease its congestion window by half which will decrease the data flow to router thus controlling congestion.

A dropped packet means that all network resources, including router processing power and bandwidth, for that particular packet are wasted. Therefore detecting congestion at the edges of network by means of actual packet drops in router queues is not very resource-efficient method. Recently a new mechanism called Explicit Congestion Notification (ECN) has been proposed to mark packets instead of dropping them at congested router queues. Thus by using ECN, we can avoid the wastage of network resources as well as wastage of time, due to packet retransmissions and timeouts caused by dropped packets.

This Chapter presents a brief and concise overview of major TCP types, with special emphasis on their congestion avoidance/control techniques and packet loss recovery algorithms, in Section 2.1. This survey provides us necessary background material to design new congestion avoidance/control algorithms, in later Chapters. In Section 2.2, we analyze the biasing problem of most commonly used TCP version, i.e. TCP Reno, against connections having longer round trip times. This issue of round trip time bias will be taken into account while designing new algorithms in the proceeding Chapters. Packet re-ordering and TCP AIMD mechanisms are briefly given in Sections 2.3 and 2.4, respectively. An overview of short TCP connections, such as occurring in web transfers, with a simple mathematical model is presented in Section 2.5. Next we will briefly describe the concept of self-similarity, in TCP traffic, in Section 2.6.

Then we present major mathematical models of TCP in Section 2.7. The use of packet marking technique (Explicit Congestion Notification) instead of packet dropping (as a congestion signal from router), along with its weaknesses, is briefly discussed in Section 2.8. Internet 2 is briefly outlined in Section 2.9. Transport protocols for future high bandwidth-delay product networks are briefly described in Section 2.10. After designing of a new congestion control algorithm the next logical step is its performance evaluation by computer simulations before evaluating it in testbed environment and subsequently implementing it in real networks. In this context, we present different types of performance evaluation metrics for TCP in the Section 2.11. Finally, we present our conclusion in Section 2.12.

2.1 EVOLUTION OF TCP

Early TCP implementations followed a go-back-n model using the cumulative positive acknowledgements and they required a retransmit timer to expire for resending of lost data, [Fall and Floyd 1996]. A lot of research work has been carried out on the refinement and evolution of TCP, e.g. see: [Stevens 1994], [Stevens 1995]. In this Section we will present an overall view

of important types and modifications made to TCP.

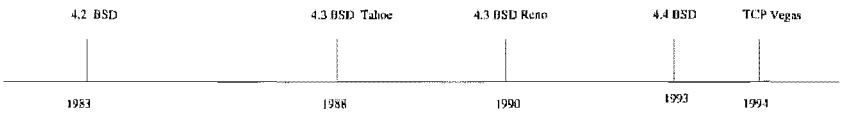

Figure 2.1 Time line for important types of TCP.

2.1.1 TCP Tahoe

This version of TCP, originated from the seminal work in [Jacobson 1988], has Slow-Start (SS), Congestion Avoidance (CA) and Fast Retransmit (FR) algorithms built in it, and was first implemented in 1988 as 4.3 BSD Tahoe TCP and was later improved in [Jacobson 1990].

In code of TCP a new variable called congestion window, ($cwnd$), was added per-connection state which limits the amount of data that can be send, [Jacobson 1988]. The variable $cwnd$ operates in conjunction with another variable called slow start threshold, ($ssthresh$), and switches TCP Tahoe from slow start to congestion avoidance phase. It is related to the corresponding window of data at sender side by $\frac{cwnd}{M_s}$, where M_s is sender maximum segment size in bytes or in packets, see: [Mathis *et al.* 1997] and [Allman *et al.* 1999]. Further, we use congestion window and window synonymously and represent them by W (in packets), as in [Lakshman and Madhow 1997] and [Mathis *et al.* 1997].

At any given time a TCP must not send data with sequence number higher than sum of highest ACKed sequence number and minimum of W and receiver advertised window, $rwnd$, for details see: [Postel 1981], [Jacobson 1988] and [Allman *et al.* 1999]. The SS, CA and FR algorithms are briefly described in the following Subsections.

2.1.1.1 SS Algorithm

A TCP connection starts in SS algorithm with W set equal to 1 packet and $ssthresh$ set to arbitrary high value, such as equal to 65535 bytes as in [Stevens 1997] or $rwnd$ as in [Allman *et al.* 1999], and for every ACK of new packet increment W by 1 packet, i.e. ($W \leftarrow W + 1$) until $W < ssthresh$.

2.1.1.2 CA Algorithm

The CA algorithm will be triggered when $W > ssthresh$, in which W is increased by $1/W$ for every new ACK, i.e. ($W \leftarrow W + \frac{1}{W}$). In this algorithm the congestion window is incremented by 1 packet per round trip time. In case of timeout due to packet loss, set $ssthresh = W/2$ and $W = 1$ and on receiving a new ACK increase W by 1 as in SS algorithm, [Allman *et al.* 1999].

2.1.1.3 FR Algorithm

In FR algorithm, TCP Tahoe retransmit the lost packet upon receiving three duplicate ACKs (with same sequence numbers) without waiting for retransmit timer to expire. After retransmission of missing segment it sets $W = 1$ and $ssthresh = W/2$ and performs SS.

Latter it was found that this algorithm works well for single packet drop but fails in case of multiple packet drop from same window of data, [Fall and Floyd 1996]. Each retransmission of packet will force TCP Tahoe to go to SS mode thus resulting in serious loss of performance, see: [Floyd 1995]. This weakness of TCP Tahoe is partially improved in TCP Reno using Fast Recovery algorithm as explained in next Section.

2.1.2 TCP Reno

TCP Reno is a most widely used version of TCP, [Paxson 1997a] and [Padhye 2000]. It was first implemented in 1990 as an improved form of TCP Tahoe in 4.3 BSD, [Jacobson 1990]. In addition to other algorithms of TCP Tahoe as discussed in last Subsection, another algorithm called Fast Recovery algorithm was implemented along with Fast Retransmit algorithm in TCP Reno.

TCP Reno after retransmission of missing sequence number by fast retransmission algorithm will enter the Fast Recovery algorithm until the receipt of non duplicate ACK. Its Fast Recovery algorithm reduces the congestion window by half after Fast Retransmission instead of decreasing it to one segment and starting again in slow start phase, as is done in TCP Tahoe.

Though TCP Reno significantly improved the behavior of TCP Tahoe when a single packet is dropped from a window of data, but it also suffers from recovery problems when multiple packets are lost in a single flight of data window [Fall and Floyd 1996]. The Fast Retransmit and Fast Recovery algorithms of TCP Reno, [Allman *et al.* 1999], are implemented together as given in the next Subsection.

2.1.2.1 Fast Retransmit and Fast Recovery Algorithms

A receiver of TCP Reno will generate duplicate ACKs when packets with out of order sequence numbers are received due to packet loss, reordering or replication during transmission. Hence, sender of TCP Reno will detect out of order sequence, enqueued at the receiver side, by the reception of 3 duplicate ACKs. Then it calls Fast Retransmit algorithm, which retransmits the missing sequence number without waiting for the expiry of retransmission timeout interval.

Thus, the purpose of 3 duplicate ACKs is to inform the data sender that an out of order packet has been received and the sequence number actually expected by receiving side is indicated. Presently TCP Reno sender implements "Fast Retransmit and Fast Recovery Algorithms" together in the following procedure, [Allman *et al.* 1999]:

[1] After receiving third duplicate ACK, set $ssthresh$ to no more then $max\left(\frac{Flightsize}{2}, 2 \cdot M_s\right)$, where $Flightsize$ is amount of unacknowledged sent data and M_s is sender maximum segment size which is 1 packet usually.

[2] Retransmit the lost sequence number and set $W = ssthresh + 3 \cdot M_s$ to inflate the congestion window artificially by three which are buffered at receiver side.

[3] Increment W by M_s for each additional duplicate ACK, which will inflate the W to reflect the additional packets which have left the network.

[4] Transmit new sequence number if allowed by new W and receiver's advertised window.

[5] After the arrival of new ACK of retransmitted packet in step 1 (one round trip time or earlier), deflate W by setting it to value in step 1.

It has been shown in [Floyd 1995] and [Fall and Floyd 1996], that Fast Recovery in TCP Reno is generally not efficient in the case of multiple packet losses in same window, for which expiry of retransmission timeout interval and corresponding slow start phase will be needed.

2.1.3 TCP New Reno

The New Reno TCP [Floyd and Henderson 1999], is a modification in the Fast Recovery Algorithm of TCP Reno [Jacobson 1988], [Jacobson 1990], [Allman *et al.* 1999].

Along with other proposals (which were not considered in [Floyd and Henderson 1999]), it was suggested in [Hoe 1995] and [Hoe 1996] that during the Fast Recovery the TCP data sender should respond to partial ACK by inferring that the indicated packet has been lost, and retransmitting that packet.

Thus, in contrast to TCP Reno, the partial ACKs do not take TCP New Reno out of Fast Recovery. Therefore, without undergoing retransmission timeout, TCP New Reno can recover from multiple packet loss in a single window of data, by retransmitting one packet per round trip time. Next, we describe the modifications made by TCP New Reno in the Fast Recovery algorithm of TCP Reno as described in previous Subsection.

2.1.3.1 TCP New Reno and Fast Recovery Algorithm

The fix to the problem of multiple packet loss in TCP Reno is provided by TCP New Reno [Floyd and Henderson 1999] by making the following modifications in its Fast Recovery algorithm.

[1] Initialize a new variable $send\_high$ to initial send sequence number. When third duplicate ACK is received and sender is not already in Fast Recovery mode then check that the duplicate ACKs cover more than variable $send\_high$. If they do, then set $ssthresh$ to no more then $max(\frac{Flightsize}{2}, 2 \cdot M_s)$ as in TCP Reno and record the highest sequence number in another new variable $recover$.

[2] Do steps 2, 3, 4 and 5 of TCP Reno as given in Subsection 2.1.2.1.

[3] If ACK in step 5 of Subsection 2.1.2.1, acknowledges all of the sequence numbers up to and including $recover$ (known as Full ACK), then set W either to $min(ssthresh,\ Flightsize + M_s)$ or $ssthresh$ as in step 1 and exit Fast Recovery, otherwise do not exit and perform the following steps.

[4] For the ACKs which do not acknowledge all of the sequence numbers up to and including $recover$ (known as partial ACK), retransmit the first unacknowledged packet and deflate W by amount of new data acknowledged then add one M_s and afterwards try to send new segment if permitted by new W and $rwnd$. Reset the retransmit timer with the arrival of first partial ACK. Do not exit Fast Recovery until full ACK is received.

[5] If a retransmit timeout happens then record the highest sequence number transmitted so far in the variable $send\_high$ and exit fast recovery if applicable.

Further, it had also been pointed out in [Floyd and Henderson 1999], that there exist different possible variations of New Reno which differ slightly from each other but significantly from TCP Reno.

2.1.4 TCP Reno with Selective Acknowledgement

The Selective Acknowledgement option (SACK) was first proposed in [Jacobson and Braden 1988], as an extension to TCP Reno to get an efficient operation over high bandwidth-delay product paths. Latter, after some slight modifications, it became an Internet standards track protocol in [Mathis *et al.* 1996].

SACK option is used by TCP data receiver to inform the TCP data sender that a non contiguous segment of data has been received and it is queued. To use the SACK option in the TCP data transmission the TCP receiver must be SACK capable otherwise sender will default to TCP Reno. Thus adding the SACK option does not change the basic underlying congestion control algorithms of TCP Tahoe and TCP Reno and they will still uses the retransmit timeout method for packet loss recovery as a last resort [Fall and Floyd 1996]. The TCP receiver will wait for the missing sequence numbers until all of them are received and then it issues the cumulative ACK as usual. The information about missing sequence numbers is transmitted to TCP sender by using three SACK blocks with each ACK, using the rules described in [Mathis *et al.* 1996].

A simulation based comparison of TCP Tahoe, TCP Reno and TCP SACK showed that TCP SACK recover quickly and smoothly without the expiry of retransmission timeout interval in the event of multiple packets losses, [Fall and Floyd 1996]. These simulation based studies were further strengthened by test bed emulation results and the real Internet experiments in [Bruyeron *et al.* 1998]. These experiments showed that in the real Internet, TCP SACK has 15% to 45% higher throughput then TCP Reno. Whereas on test bed the improvement in throughput

of TCP SACK over TCP Reno ranges from 10% to 120% depending on the congestion pattern being employed. The fairness between TCP Reno and TCP SACK has been investigated in [Floyd 1996] by using simulations and some practical implementational issues has been discussed in [Rizzo 1996].

Hence in summary, SACK is a strategy to improve the throughput degradation (due to loss of ACK-clock which leads to subsequent retransmission timeout interval expiry and slow start), during the period of multiple packet loss in a single window of data. It corrects the problem of unnecessary retransmission of lost packets and try to maintain the ACK-based clock in case of multiple packets loss. Thus, it tries to optimize the retransmissions by providing additional information to data sender. However, a complete mathematical model of TCP SACK is still an open problem, see: [Padhye 2000].

2.1.5 TCP FACK

The Forward Acknowledgement (FACK) algorithm proposed in [Mathis and Mahdavi 1996], is designed to be used in conjunction with SACK option.

It decouples congestion control from other algorithms such as data recovery algorithm by introducing two new variables *snd.fack* and *retran_data*, [Mathis and Mahdavi 1996]. Thus, the sender can estimate the actual quantity of data outstanding in network and can inject new data if allowed by receiver's window. With the variables $snd.una$ and $snd.nxt$, as defined in [Postel 1981], this algorithm can be briefly as outlined as follows:

During the non-recovery state, $snd.fack$ is updated from ACK and is same as $snd.una$, but in the recovery state the sender updates $snd.una$ from ACK number, but $snd.fack$ is updated from SACK. When a SACK block is received which acknowledges data with a higher sequence number than the current value of $snd.fack$, than it is updated to reflects the highest sequence number plus 1. Each time a sequence number is retransmitted the $retran\_data$ is incremented by segment size and decreased by same amount when a retransmitted segment left network. Thus, TCP FACK estimates the amount of data outstanding in network by $awnd = snd.nxt - snd.fack + retran\_data$. Then TCP FACK send data $while\ \ (awnd < W)$.

However, TCP FACK behaves poorly when *awnd* falls below the W and a loss is experienced, for details see: [Mathis and Mahdavi 1997].

2.1.6 TCP SACK with Rate-Halving

The TCP Rate-Halving Algorithm was first presented in [Mathis and Mahdavi 1997], later refined in [Mathis *et al.* 1999], and is based on modifications in the TCP Reno's Fast Recovery algorithm as proposed in [Hoe 1995] and [Hoe 1996].

When TCP Reno enters Fast Recovery after Fast Retransmission, it has to wait for a half round trip time for the arrival of additional duplicate ACKs before new data can be send. After

half round trip time, a new window of data is transmitted which cause burstiness in network. In contrast to TCP Reno, Rate-Halving transmit data over entire recovery R for each arriving ACK. Although, theoretically Rate-Halving algorithm can be used with TCP Reno, TCP New Reno, TCP SACK with and without ECN, but it has not been investigated much and needs to be explored further by simulations and practical test bed experiments.

2.1.7 TCP Vegas

TCP Vegas was first proposed in [Brakmo *et al.* 1994a], as an algorithm in which congestion window is increased and decreased additively as compared to TCP Reno in which increase is additive and decrease is multiplicative.

While in the congestion avoidance phase, it adds or subtracts one packet from the congestion window every round trip time, depending upon the difference between expected and actual bandwidths. On the other hand TCP Reno in CA phase, adds one packet to the congestion window every round trip time and decreases congestion window by half on detection of packet loss. Further, as compared to TCP Reno, the slow start and packet retransmission algorithms in TCP Vegas had also been modified.

The main advantages of TCP Vegas are its fairness and high throughput. Wherein contrast to TCP Reno, it does not have bias against connections with longer round trip times. Its major draw back is that it cannot compete with TCP Reno in a heterogenous environment, where it relinquishes its fair share of bandwidth to TCP Reno. Furthermore, TCP Vegas will be investigated in detail in forthcoming Chapter.

2.2 TCP RENO'S BIAS AGAINST LONGER ROUND TRIP TIME CONNECTIONS

In this Section we analyze the fairness or bandwidth sharing among N_c number of TCP Reno connections, each having different round trip time. Let us consider a single TCP connection $i \in N_c$, which has propagation delay of τ_{p_i} and congestion window size of $W_{r_i}(t)$ at time t. It is sending data to a Droptail router (FIFO) of buffer capacity B (packets). The router has service rate μ (packets/s) on a bottleneck link of capacity C, where $C \leq \mu$. The round trip time, R_i, is obtained by the sum of propagation delay and queuing delay, which is given by the following expression:

$$R_i = \tau_{p_i} + \left(\frac{q(t)+1}{\mu}\right), \tag{2.1}$$

where $q(t)$ is instantaneous queue size at time t. The growth rate of congestion window W_{r_i} with time and with the number of arriving ACKs, N_a, are represented by $\frac{dW_{r_i}}{dt}$ and $\frac{d\,W_{r_i}}{dN_a}$, respectively. The rate at which the ACKs arrive at the source side is denoted by $\frac{dN_a}{dt}$. Then, during

congestion avoidance phase of TCP Reno the following identity holds, [Shenker *et al.* 1990]:

$$\frac{dW_{r_i}}{dN_a} = \frac{1}{W_{r_i}}. \tag{2.2}$$

The ACKs arrive at TCP source at rate equal to the instantaneous throughput, λ_i, thus we can have following simple relation :

$$\lambda_i \equiv \frac{dN_a}{dt} = min\left(\frac{W_{r_i}}{R}, \mu\right). \tag{2.3}$$

For the case of $\frac{W_{r_i}}{R} < \mu$ in equation (2.3), we can have the following relation:

$$\lambda_i \equiv \frac{dN_a}{dt} = \frac{W_{r_i}}{R_i}. \tag{2.4}$$

Multiplying equations (2.2) and (2.4), we can get

$$\frac{dW_{r_i}}{dt} = \frac{1}{R_i}. \tag{2.5}$$

The above equations (2.4) and (2.5) shows us that both the throughput λ_i and congestion window growth rate $\frac{d\,W_{r_i}}{dt}$ are inversely proportional to the R_i, i.e.

$$W_{r_i} \propto \frac{1}{R_i}. \tag{2.6}$$

Hence we have:

$$\lambda_i \propto \frac{1}{R_i^2}. \tag{2.7}$$

The empirical observations in [Lakshman and Madhow 1997] suggest that in general throughput, λ_i, is actually governed by the following relation:

$$\lambda_i \propto \frac{1}{{R_i}^{a_r}}, \tag{2.8}$$

where a_r in equation (2.8) is a constant whose value varies from 1 to 2, i.e. $1 \leq a_r < 2$, according to given network. The analysis given above suggests that TCP Reno's bias against connections with larger round trip times is a fundamental consequence of its congestion window dynamics, see: [Floyd 1991a] and also its references.

One solution to this round trip time bias problem of TCP Reno is to use buffer size which are much larger than the bandwidth-delay product then round trip time will be dominated by the queuing delays, which at the high utilizations would be same for all connections, for details see: [Lakshman and Madhow 1997]. However, using buffers of large capacity will increase the

response time and thus consequent slower system. Also, increased queuing delays will harm Quality of Service as they are its major component, [Floyd *et al.* 2001].

2.3 TCP AND PACKET REORDERING

Packet reordering is a phenomenon in which out of order packets are transmitted to data receiver due to multiple routes having different time delays. It can happen on forward path (packet reordering) as well as on reverse path (ACK reordering), [Bennett *et al.* 1999]. Its major effects are the unnecessary retransmission of packets, incorrect estimation of round trip times, less data sending rate (low value of W and $ssthresh$), reduction in efficiency of receiving TCP and late retransmission of actually lost packets, [Bennett *et al.* 1999].

The reliability of TCP, maintained by cumulative ACKs policy at receiver side, forces the transmitter to retransmit the reordered packets. For fast retransmission algorithm of TCP, see Subsection 2.1.2.1 or [Stevens 1997] and [Allman *et al.* 1999] for details.

In [Blanton and Allman 2002] several alternatives have been proposed to make TCP robust to packet reordering. However, the simulations results presented there need validation by real network. Also, improvement in TCP's performance under reordering has been reported in [Zhang *et al.* 2002] by using DSACK, [Floyd *et al.* 2000a]. Whereas, these exhaustive ns-based results need to be further verified by implementing the proposed enhancements in the TCP stack of operating systems; such as FreeBSD or Linux.

Further, in addition to packet losses and reordering there are also other pathologies in real networks such as ACK loss, ACK duplication and packet corruption etc which need further research, for details see: [Paxson 1997a].

2.4 TCP AND AIMD PRINCIPLE

The design and operation of present TCP implementations depends on Additive Increase and Multiplicative Decrease (AIMD) algorithm for controlling of their congestion window, [Jain *et al.* 1987], [Jacobson 1988] and [Chiu and Jain 1989].

It has been proved, [Chiu and Jain 1989], that in a network with all competing connections having same additive increase rate and multiplicative decrease rate under synchronous control actions, AIMD can provide fair allocation of bandwidth in a distributed manner.

Whereas in actual TCP the decrease factor is same for all connections, but the additive increase factor is roughly one segment per round trip time which does not provide a uniform rate of increase for connections with different round trip times. Thus, in a mixture of short R connections and long R connections the short one will grab the bandwidth before longer ones and cause unfairness. Also the current Internet is not a synchronous network and frequency of congestion feedback can be different for different flows.

Thus, in the design of future protocols for high speed networks the strict model of AIMD currently being employed in TCP, [Jacobson 1988], may be abandoned, as has been suggested in [Gorinsky and Vin 2002] and [Floyd 2003].

2.5 SHORT TCP CONNECTIONS

In [Cardwell *et al.* 1998], and references therein, it has been mentioned that most of the TCP connections are used for short data transfer. They carry data using hyper text transfer protocol (http) with vast majority of them having transfer size of 8-12 Kilo Bytes. In generic HTTP 1.0, [Berners Lee *et al.* 1996], a web browser opens up a new TCP connection for each object embedded in a web page [Stevens 1996]. In HTTP 1.1 and in some implementations HTTP 1.0 there is a support for persistent connections which allows a single TCP connection to download several objects, [Fielding *et al.* 1997].

For short TCP connections the steady models of TCP such as given in [Floyd 1991a], [Mathis *et al.* 1997] and [Padhye *et al.* 2000] are unproven and thus simple models are required. One such model has been derived in [Cardwell *et al.* 1998] where it has been proved that time, t_d, required to open k TCP connections and transfer D_b bytes of data without packet loss, incorporating the time for handshake, delayed acknowledgement and slow start from initial congestion window of W is given by:

$$t_d = \log_r \left(\frac{D_b \cdot (r-1)}{W \cdot M_s \cdot k} \right) \cdot R + R + t_{del}, \tag{2.9}$$

where in equation (2.9) r is ratio of geometric series formed by congestion window, M_s is a maximum segment size, R is round trip time and t_{del} is time for delayed acknowledgements. Further, equation (2.9) has also been modified for random packet losses in [Cardwell *et al.* 1998]. Latter simulations results and analytical results from equation (2.9) and from random loss model are compared with each other and with the real world TCP implementations in the Internet. It was found that simulations results agree with real world Internet measurements.

The steady state model of TCP in [Padhye *et al.* 2000] has been further extended for short TCP transfers in [Cardwell *et al.* 2000]. Using simulations, controlled measurements of TCP transfers and live world wide web measurements it is shown that contrary to [Padhye *et al.* 2000] the new model in [Cardwell *et al.* 2000] describes connection establishment and data transfer latency under range of packet losses and no losses.

2.6 TCP AND SELF-SIMILARITY IN TRAFFIC

In networking literature the term "self similarity" is used as a feature of a class of processes whose stochastic characteristics unvary with changes in time scale. This is in contrast to Poisson processes which loose their burstiness and flatten when scaled appropriately. Mathematically, a

process $X(T)$ is self similar with self-similarity parameter H, also known as known as Hurst parameter, if

$$X(\alpha t) \stackrel{\mathrm{d}}{=} \alpha^H X(t), \tag{2.10}$$

where $\alpha > 0$, $H \in [0, 1.0)$ and $\stackrel{\mathrm{d}}{=}$ is used to show equivalence in distribution, [Ostring 2001]. Self similar processes, known as long range dependent, that occur in telecommunication networks are characterized by $H \in [0.5, 1.0)$.

Propagation of self similarity in the Internet by TCP congestion control algorithms has been analyzed in [Veres *et al.* 2000]. It has been found that this property of TCP is due to adaptation of congestion control algorithms to self-similar traffic fluctuations on several time scales and if a TCP connection shares bottleneck with a self-similar background traffic flow then it propagates the correlation structure of background traffic to other parts of network.

It has been also shown that even short TCP connections can effectively propagate the long range correlations. Thus, self similarity of one TCP stream can be passed on to other TCP streams which are multiplexed with it, [Veres *et al.* 2000].

A comprehensive bibliographical guide for self similarity in data networks is given in [Willinger *et al.* 1996]. Recently, in depth study, modelling and analysis of self-similar teletraffic (for simulation purposes) has been carried out in [Jeong 2002]. Hence, the phenomenon of self similarity has a definite impact on end-to-end dynamics of TCP and it must be investigated further in the development of congestion control algorithms for future networks.

2.7 MATHEMATICAL MODELLING OF TCP

Modeling of TCP is crucial for efficient utilization of network resources and improvement in services provided to the Internet users. Its main objective is to derive explicit formulae for throughput, fairness and stability etc under different network states and traffic conditions. Also it provides us an analytical insight into the mutual working of different algorithms being used in TCP, [Padhye 2000].

Much work had been done in this direction, see e.g. : [Ott *et al.* 1996], [Lakshman and Madhow 1997], [Mathis *et al.* 1997], [Misra *et al.* 1999], [Padhye *et al.* 2000] and [Altman *et al.* 2000].

In [Mathis *et al.* 1997], an expression has been derived for bandwidth B of a TCP connection in terms of random packet loss at constant probability p , maximum segment size M_s and round trip time R, which is given by:

$$B = \frac{M_s}{R} \cdot \frac{K}{\sqrt{p}}. \tag{2.11}$$

From equation (2.11) above, window size W can be written as:

$$W = \frac{M_s \cdot K}{\sqrt{p}}. \tag{2.12}$$

The values of constant K are given in Table 2.1 which vary from 0.87 to 1.22 for periodic loss and from 0.93 to 1.31 for random loss, depending upon the policy of ACK. The number

Derivation	ACK Strategy	K
Periodic Loss	Every Packet	$\sqrt{3/2} = 1.22$
	Delayed	$\sqrt{3/4} = 0.87$
Random Loss	Every Packet	1.31
	Delayed	0.93

Table 2.1 Values of constant K

of packets N_p required to reach the equilibrium of equation (2.11) is given by the following relation:

$$N_p = \frac{1}{p} \cdot \log_2 \left(\frac{1}{K \cdot \sqrt{p}} \right). \tag{2.13}$$

In this model the retransmission timeouts are neglected. Thus, it can be used as a upper bound to calculate the bandwidth, B, [Mathis *et al.* 1997]:

$$B \leq \left(\frac{M_s}{R} \right) \cdot \frac{1}{\sqrt{p}}. \tag{2.14}$$

In order to include the effects of retransmission timeouts, [Padhye *et al.* 2000] proposed a new model and drove the following equation for bandwidth B:

$$\begin{aligned} B &= \frac{\frac{1-p}{p} + \frac{2+b_{na}}{3b} + \sqrt{\frac{8(1-p}{3bp} + \left(\frac{2+b_{na}}{3b_{na}}\right)^2}}{R \cdot \left(\frac{2+b_{na}}{6} + \sqrt{\frac{2b_{na}(1-p)}{3p} + \left(\frac{2+b_{na}}{6}\right)^2} + 1 \right)}, \\ &= \frac{1}{R}\sqrt{\frac{3}{2b_{na}p}} + o(1/\sqrt{p}). \end{aligned} \tag{2.15}$$

where b_{na} is the number of packets acknowledged by each ACK received by the source. The equation (2.15) reduces to inverse square-root p model of TCP, [Mathis *et al.* 1997], if the loss probability p is small, $b_{na} = 1$ and no timeouts occur, [Padhye *et al.* 2000]. If timeouts are included, the sending rate can be approximated by:

$$B \approx \frac{1}{R\sqrt{\frac{2b_{na}p}{3}} + T_0 \min\left(1, 3\sqrt{\frac{3b_{na}p}{8}}p(1+32p^2)\right)}, \tag{2.16}$$

where T_0 is the value for the first timeout period. However, this model does not captures the subtleties of Fast Recovery algorithm and also it assumes that time spent in slow start is negligible.

By using techniques of stochastic processes, an expression for TCP throughput is computed in [Altman *et al.* 2000]. It has been shown that for a TCP connection having transmission rate of $X(t)$ at time t, multiplicative decrease factor of ν, linear increase factor of α, loss events intensity λ, and a correlation structure $\mathbb{R}(k)$ then throughput $\bar{X}$ is given by:

$$\bar{X} = \lambda\alpha \left[\frac{1}{2}\mathbb{R}(0) + \sum_{k=1}^{\infty} \nu^k \cdot \mathbb{R}(k) \right], \tag{2.17}$$

$$\bar{X} = \frac{1}{R\sqrt{b_{na}p}} \sqrt{\frac{1}{2}\hat{\mathbb{R}}(0) + \sum_{k=1}^{\infty} \nu^k \cdot \hat{\mathbb{R}}(k)}. \tag{2.18}$$

where b_{na} is the number of packets acknowledged by each ACK, as in the previous model, and $\hat{\mathbb{R}}(k)$ is a normalized correlation function, for details see: [Altman *et al.* 2000]. Also it has been shown that the equation (2.17) confirms the TCP model presented in [Mathis *et al.* 1997].

2.8 EXPLICIT CONGESTION NOTIFICATION

The proposal to add a two-bit ECN field in the IP header and two-bit Flag in TCP header is first given in [Ramakrishnan 1999] and latter modified in [Ramakrishnan *et al.* 2001]. The bits in the IP header are called the ECN-Capable Transport (ECT) and the Congestion Experienced (CE) bits. The flags used by TCP header are called ECN-Echo and Congestion Window Reduced (CWR).

If ECT is 0, then it indicates that TCP will ignore the CE bit. The ECT bit set to "1" means that TCP can participate in ECN. In the case of congestion, the router sets the CE bit to 1 to indicate congestion to end nodes. Any router should never change the CE bit from 1 to 0. The default values of both ECT and CE are 0. In order that TCP carry out ECN successfully, it requires the following three changes:

- ECN negotiation during the connection set up phase,
- When a data receiver gets a packet with the CE bit high from the router, then it signals this congestion information to the data sender through the setting of ECN-Echo bit of the corresponding ACK packet to 1.
- The data receiver on getting an ACK packet with ECN-Echo set to 1, will reduce its current congestion window and $ssthresh$ by half and sets the CWR flag to 1 in the next packet.

- The data receiver will keep on sending ACK's with ECN-Echo bit equal to 1 until it receives a new packet through the router which has the CWR bit equal to 1 and then it stop setting ECN-Echo bit to 1 until it get another packet with CE equal to 1. TCP data sender should not react to congestion indication more then once in one window of data or one round trip time.

2.8.1 Problems with ECN

The purpose of ECN can be defeated by non-complying end nodes or by the faulty or rogue router. The ECN mechanism will not work in the following cases:

- End nodes set the ECT bit of transmitted packets but do not respond to CE packets.
- Router setting the ECT bit in a packet of a non-ECN capable transport.
- Router erases the CE bit in arriving packets. This would prevent the congestion indication to downstream receivers.
- Router sets the CE bit with unnecessarily high frequency. This situation is analogous to excessive packet dropping by a router.

For details about the problems in ECN see: [Floyd 1994] and [Ramakrishnan *et al.* 2001].

Further, in [Misra and Ott 2003] the performance sensitivity and fairness of ECN-aware TCP has been studied. A generalized congestion control and ECN-mod protocol has been introduced, whose performance is latter compared with conventional ECN-aware and ECN-unaware TCP New Reno, by performing simulations.

Also, it has been pointed out that the optimal shape of packet marking function and its dependence on the precise choice of the ECN-mod coefficients is an open problem.

2.9 TCP AND INTERNET 2

Internet 2 is an experimental high performance network being led by more than 200 universities in partnership with Industry and Government agencies, [Internet2 2003]. Its main features are all-optical networks, Internet Protocol version 6 [Deering and Hinden 1995] and [IPv6 2003], multicast technology and having very high capacity backbone (OC-48) links, [Yeung 1997].

Present TCP implementations are optimized for low bandwidth and does not provide adequate buffer space for high performance data transfers. One possible solution would be manually setting of buffer space in an application or in an operating system, which is not automatic method. A well tuned TCP can carry up to 1 Gbps and non tuned connection gets only up to 3 Mbps, [Mathis 2001]. In order to overcome these problems, different tools and software are being developed and employed in the Web100 project, so that new and well tuned versions of TCP are being experimented with the Internet 2, [Web100 2002].

Thus, new protocols as well as modifications in present TCP are needed for the future high speed Internet, one such proposal is briefly explained in the next Section.

2.10 TRANSPORT PROTOCOLS FOR HIGH BANDWIDTH-DELAY PRODUCT NETWORKS

Present TCP based flows show poor performance in high bandwidth product environments, [Floyd 2003]. A standard TCP connection with 1500 bytes packet size and a round trip time of 100 ms, achieving a steady state throughput of 10 Gbps would require an average congestion window of 83,333 segments, [Floyd 2003]. It would require a packet drop rate of at most one in every 5×10^9 packets or one congestion event every 1 2/3 hours, which is widely acknowledged as an unrealistic constraint, [Floyd 2003].

Currently Internet users wishing to achieve throughputs of 1 Gbps or more typically employ multiple TCP connections in parallel or use MulTCP, [Crowcroft and Oechslin 1998], which is basically an aggregate of N TCP connections, [Floyd 2003]. This approach does not work well in periods of moderate or high congestion. Therefore, we need to find stable, flexible and scalable solutions which can coexist with the standard TCP in congested environments. Further in this Section we will briefly review some recently proposed protocols for the future high speed networks.

2.10.1 Explicit Congestion Notification

Recently a new proposal Explicit Congestion Notification (XCP), [Katabi *et al.* 2002], has been proposed to address the problem of low performance by TCP networks. It achieves high link utilization and reduces wastage of bandwidth by separating the mechanisms for efficiency and fairness. A XCP based router has two parts namely: efficiency controller and fairness controller, [Katabi *et al.* 2002]. The purpose of the former is to maximize the link utilization and the latter is to ensure fairness.

In [Katabi *et al.* 2002] XCP is widely tested through ns-based simulations and its performance is shown to be better for high bandwidth-delay product networks when compared to existing TCP based protocols. Further its stability has been investigated by employing feedback control theory as well as by ns-based simulations.

XCP is a good fit for high performance computing, including petabyte-size file transfers and networks with very expensive bandwidth, as it grabs the idle bandwidth of expensive links very quickly, see [Falk *et al.* 2003].

However, XCP requires the participation of all routers along the path of a traffic flow which limits the scope of its deployment in real networks. Therefore XCP can be deployed on a cloud-by-cloud basis and also within specialized networks, [Falk *et al.* 2003]. A number of

research groups including MIT, ISI at University of Southern California and Cisco systems, [Falk *et al.* 2003], are working to evaluate the performance of XCP in real networks.

2.10.2 HighSpeed TCP

In [Floyd 2003], a new version of TCP, called HighSpeed TCP for large congestion windows, has been proposed for high speed Gigabit networks by using modifications in equation (2.12).

HighSpeed TCP defines four parameters which are low congestion window W_l with low packet loss probability p_l and high congestion window W_h with high packet loss probability p_h. For a given window size of W with packet loss probability p, if $W < W_l$ then follow standard TCP response function as given in equation (2.12), otherwise for $W > W_l$ modify W by following relation:

$$W = \left(\frac{p}{p_l}\right)^S \cdot W_l, \tag{2.19}$$

where S in above equation is given by:

$$S = \frac{\log W_h - \log W_l}{\log p_h - \log p_l}. \tag{2.20}$$

It is also suggested that for each ACK during non-congestion state increase W by a factor of $a_i(W)$ segments per round trip time and during congestion state decrease W by $1 - b_d(W)$, where for the standard TCP we have $a_i(W) = 1$ and $b_d(W) = 0.5$, [Jacobson 1988]. When $W \leq W_l$, HighSpeed TCP uses values of increase and decrease factors $a(W)$ and $b(W)$ same as standard TCP and for $W = W_h$ we have:

$$a_i(W) = W_h^2 \cdot p_h \cdot \left\{\frac{2 \cdot b_d(W)}{2 - b_d(W)}\right\}. \tag{2.21}$$

Presently, HighSpeed TCP has been implemented in Linux 2.4.16 Web100 kernel, [Web100 2002], and is under experimentation, [Floyd 2003].

Further extending the concept of HighSpeed TCP, [Kelly 2002] presents Scalable TCP. It has been implemented in the Linux 2.4.19 operating system's kernel. Its performance was determined by constructing a test bed of high performance PC's (having 2.4 GHz Xeon processors) at CERN, Geneva and StarLight, Chicago. However, experiments reported in [Kelly 2002] were performed only with standard TCP and Scalable TCP. Better conclusions can been drawn by conducting further experiments with other proposals as well, such as XCP described in the previous Subsection.

2.10.3 Fast AQM Scalable TCP

Recently, in [Jin *et al.* 2004], Fast AQM Scalable TCP (FAST) has been introduced for high-speed long latency networks. It separates the congestion control algorithm of TCP into four components, which are data control, window control, burstiness control and estimation.

The data control determines which packet has to be transmitted, window control determines how many packets have to be transmitted and burstiness control determines when to transmit these packets. These decisions are made on the basis on information provided by the estimation component, [Jin *et al.* 2004].

TCP FAST has the same equlibrium properties as that of TCP Vegas described before in Subsection 2.1.7. In the testbed environments FAST TCP is shown to perform better than High-Speed TCP, already described in Subsection 2.10.2. However, the stability of the FAST TCP algorithm has to be explored further before its deployment in real high bandwidth-delay product networks.

2.11 TCP PERFORMANCE EVALUATION

Traditionally performance of TCP type protocols is mainly determined by throughput and fairness [Jain and Ramakrishnan 1988] and [Allman and Falk 1999]. However, other metrics such as link utilization and loss rate are also important in effective evaluation of TCP. These metrics are briefly described in the following Subsections:

2.11.1 Throughput

Throughput is generally obtained by measuring the number of packets, bytes or bits transferred from source to destination in a given time interval or simulation time after eliminating the protocol overhead, retransmissions and duplicate packets in calculations.

In Droptail routers, to obtain high value of throughput we need to maintain large buffers which will increase the queuing delay, resulting in an increase in the response time. On the other hand to get shorter response time, smaller buffers are required which will reduce throughput. Thus, in the case of Droptail routers there is tradeoff between high throughput and low response time, [Floyd *et al.* 2001].

Internet applications like FTP requires high throughput while applications like remote login or telnet require low response time, [Jain and Ramakrishnan 1988]. Random Early Detection routers, discussed in later Chapters, can provide high throughput and low queuing delay at the same time.

2.11.2 Fairness

Fairness is a another most important metric for performance evaluation of computer networks protocols. and is a major design objective of congestion control algorithms in the current Internet.

Its most intuitive and obvious definition is that the all sessions of data flow should be entitled to use the network resources including the bandwidth equally without any bias. Thus a fair TCP connection does not deprive other connections their fair share of bandwidth, [Bertsekas and Gallager 1996].

Fairness is quantified by fairness metrics which try to determine the mutual effects of TCP connections on bandwidth sharing by each other. One such metric is called Jain's Fairness Index (JFI) which is explained in next Subsection.

2.11.2.1 Jain's Fairness Index

Following fairness function, commonly called Jain's Fairness Index (JFI), is defined in [Jain *et al.* 1984] and [Jain 1991], to quantify the fairness among n independent users of a computer network who share a common resource (such as bottleneck link bandwidth B) and have throughputs $(x_1, x_2, ..., x_n)$:

$$JFI \equiv f(x_1, x_2, ..., x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} x_i^2}. \tag{2.22}$$

For all nonnegative values of x_i's the JFI always lies between 0 and 1. In the case that all users receive an equal throughput the JFI become equal to 1. If only k out of the n users receive equal throughput and the remaining $n - k$ users receive zero throughput then JFI has the value k/n, [Jain 1991].

2.11.2.2 Proportional Fairness

The concept of proportional fairness for a generalized closed fluid network was first proposed in [Kelly 1997]. With detailed notations as given in [Kelly 1997], a vector of rates x is said to be $proportionally\ fair$ if it is feasible for a link of capacity C, number of sources S and connection matrix A, i.e. $x \geq 0, Ax \leq C$, and for any other feasible vector x^*, the aggregate of proportional changes is zero or negative, i.e. :

$$\sum_{s \in S} \frac{x_s^* - x_s}{x_s} \leq 0. \tag{2.23}$$

An algorithm was proposed in [Kelly *et al.* 1998] to achieve a proportionally fair rate vector as defined in equation (2.23).

2.11.3 Loss Rate

The loss rate δ is defined as ratio of packets lost at queue buffer due to overflow or random dropping to the total arrived packets, see e.g [Feng *et al.* 1999a] and [Allman and Falk 1999]. It can be expressed as follows:

$$Loss\ \ Rate, \delta = \frac{Number\ of\ packets\ dropped\ at\ queue\ buffer}{Total\ number\ of\ packets\ arrived\ at\ queue\ buffer}. \tag{2.24}$$

2.11.4 Link Utilization

Closely related to the fairness is link utilization (η) metric which determines the effective use of scarce network bandwidth. High fairness without proper link utilization usually leads to under-utilization of link capacity.

It is defined as number of original packets/bits (not counting the retransmissions and duplicate packets) successfully transferred over a link during some given time interval divided by maximum capacity of the link, [Feng *et al.* 1999a].

$$Link\ Utilization, \eta = \frac{Total\ number\ of\ packets\ sent\ over\ link}{Maximum\ number\ of\ packets\ which\ could\ be\ send\ over\ the\ link}. \tag{2.25}$$

2.12 CONCLUSIONS

In this Chapter we have described the major algorithms being used in the main flavours of TCP with an emphasis on congestion control and packet loss recovery algorithms. We first described the universally used slow start and congestion avoidance algorithms and then in sequel we have also described the fast retransmission and fast recovery algorithms of TCP Reno and TCP New Reno.

Further, we analyzed the bias of Reno derived TCPs against connections with longer round trip times. It forms a basis for the development of fair end-to-end congestion control protocols/algorithms in latter Chapters. After studying TCP Reno and its variants, we also looked briefly at TCP Vegas, which will be studied in detail in next Chapter.

We have also briefly discussed the packet reordering phenomenon, in which we do not loose packets but receive them in an incorrect sequence, which causes unnecessary retransmissions. Also we noticed that less work has been done in this area, thus it needs to be further explored by mathematical analysis, by simulation studies and also with the live Internet measurements.

Then we overviewed AIMD principle used in TCP Reno and its variants. It is worth to mention that TCP Reno is an AIMD algorithm whereas in contrast TCP Vegas is an AIAD algorithm which will be investigated further in forthcoming Chapter. Next we briefly described self similarity in TCP/IP networks. Also we gave a succinct description of the Internet 2. Then

we summarily looked at the three major mathematical models of TCP. Whereas one of the models, [Mathis *et al.* 1997], has been employed in the latter Chapters to develop new router based congestion control algorithms.

Next, a newly proposed TCP for future high speed Gbps networks is explained concisely. Further, we also describe a model for short TCP transfers, such as used in www. Next we presented ECN in lieu of packet drops as a congestion notification signal to the end hosts. Also the weaknesses of ECN were briefly described. In next Chapter, ECN will be used in conjunction with TCP Vegas to improve its performance. In the last part of this Chapter we presented metrics for evaluating TCP. These metrics will be employed in the proceeding Chapters to evaluate new fair end-to-end congestion control algorithms.

Chapter 3

FAIR END-TO-END CONGESTION CONTROL PROTOCOLS

As discussed in Chapter 2, the most widely used version of TCP i.e. TCP Reno and its derivatives, have bias against connections with longer round trip times (R). TCP connections with longer R's open their congestion windows more slowly then connections with shorter R's. Thus, if there is a mixture of short and longer R connections sharing the same bottleneck link then shorter R connections will grab more bandwidth than connections with longer R. This inherent problem of fairness in TCP Reno has been well observed by many other researchers such as [Hashem 1989], [Mankin 1990], [Floyd 1991a] and [Lakshman and Madhow 1997].

In order to improve the fairness of TCP Reno in environments of different (long and short) round trip times, [Floyd 1991a] has proposed a Constant-Rate (CR) algorithm for window increase. In CR algorithm, each connection increases its window size by $a \cdot r^2$ packets each round trip time, where a is some fixed constant and r is the calculated average round trip time. Using CR algorithm each TCP connection increases its window size by a packets per second. CR algorithm has been simulated in [Floyd 1991a] using $a = 4$ for multiple congested Random Early Detection (RED) routers, [Floyd and Jacobson 1993], and it was found to reduce bias for longer round trip time connections. An important issue related to CR algorithm is proper selection of the constant a, otherwise its performance is not optimized.

In [Henderson *et al.* 1998], CR algorithm is further studied using ns simulations and effects of constant a on TCP fairness has been investigated. It is found that improper selection of parameter a will lead to very bursty packet send patterns which will lead to increased packet losses. In order to eliminate the bursty packet send patterns, it is suggested that the increase in window size should be bounded by 1 segment per ACK. After several simulations Jain's fairness index is computed and plotted for different values of a for three different network topologies. Since it is difficult for a connection to determine the number and type of other connections against which it is competing, thus it is concluded that a good value of constant a cannot be determined with high accuracy on an operational basis, see: [Henderson *et al.* 1998]. Thus, CR policy is difficult to be managed in a distributed network and it is detrimental to TCP connections operating under the standard policy of updating congestion windows i.e. increase by one and decrease by half.

Another algorithm, called increase-by-K (IBK), is also presented in [Henderson *et al.* 1998], for improving the fairness of TCP Reno. In this policy, TCP additive increase is selectively

modified to K packets rather than the standard increase by one and the maximum increase is again limited to one packet per ACK. It was found that if every connection adopts IBK then fairness will stay as original, hence there must be some round trip time threshold beyond which connections can become more aggressive. Determining the optimal value of K as a function of round trip time and the required accuracy/resolution of TCP's round trip time estimates are still open questions.

Thus, while maintaining network stability and avoiding packet bursts and consequent losses, remedying the unfair behavior of TCP Reno for connections having different round trip times is difficult because of its basic design limitation. TCP Reno keeps on increasing the size of its congestion window until packet loss is detected. Thus, it first creates congestion itself and then adjust the size of its congestion window. Hence, the design of TCP Reno can termed as reactive rather then proactive, see: [Brakmo *et al.* 1994a]. Thus, it is desired to find an alternative proactive policy in which congestion can be detected before it happens and necessary remedial measures may be taken by end hosts before congestion actually occurs. Therefore, it is required to develop proactive congestion control protocols which do not create congestion by themselves as in the case of TCP Reno.

The two most important goals of any stable network protocol are high values of throughput and fairness, see e.g: [Jain and Ramakrishnan 1988]. High value of throughput ensures that as much data as possible is transfered across a network without much packet loss and fairness ensures that each connection gets an equal share of the bandwidth available at the bottleneck links. TCP Reno in combination with router based congestion control algorithms, such as RED, can provide good overall throughput even under the congested state of network, see [Floyd 1991a], [Floyd 1991b] and [Floyd and Jacobson 1993]. However, it fails to provide the necessary fairness among the competing connections having different (long and short) round trip times [Henderson *et al.* 1998].

In this Chapter our objective is to develop protocols which can provide a high throughput and good fairness simultaneously and which also avoids unnecessary packet losses. First we study in detail the core design of different protocols presented in the literature. After studying the evolution of proactive congestion control, we concentrate on TCP Vegas and improve its performance in a heterogenous environments in which it competes with TCP Reno for getting the available bandwidth at the bottleneck link. We employ the RED algorithm, [Floyd and Jacobson 1993], with Explicit Congestion Notification (ECN), [Ramakrishnan *et al.* 2001], to develop a new algorithm which boosts up the aggressiveness of TCP Vegas so that it can get the fair share of bandwidth.

The remainder of this Chapter is organized as follows. We begin with a brief overview of the evolution of proactive congestion control protocols (CARD, DUAL, Tri S and TCP Vegas) in Section 3.1. After reviewing the core concepts of these protocols we build further on TCP Vegas as it is suitable to the existing end-to-end paradigm of TCP/IP networks. The new algorithms introduced in TCP Vegas are described in Section 3.2. A review of previous work done on

TCP Vegas to analyze its fairness and its interaction with TCP Reno is presented in Section 3.3. Next, in Section 3.4 we give a mathematical analysis of TCP Vegas and TCP Reno interaction by determining expressions for their throughputs and Jain's Fairness Index (JFI).

In Section 3.5 we perform ns-based simulation experiments. We investigate the effects of having same and different round trip times connections on the performance of TCP Reno and TCP Vegas both in the homogenous (either TCP Reno or TCP Vegas) and in the heterogenous (both TCP Reno and TCP Vegas) environments. These simulation results obtained in heterogenous experiments are further used to develop the new RED and ECN based variation of TCP Vegas. The new version of TCP Vegas is presented in Section 3.6 together with its simulations results. Finally, conclusions are presented in Section 3.7.

3.1 EVOLUTION OF PROACTIVE CONGESTION DETECTION

Before the introduction of fair end-to-end congestion control protocols such as TCP Vegas, [Brakmo *et al.* 1994a], several proposals for proactive congestion detection were put forward. Most of these schemes are based on the fact that network conditions changes when it approaches the state of congestion, see e.g: [Jain and Ramakrishnan 1988].

The concept of proactive congestion control is in contrast to reactive congestion control used in TCP Reno. TCP Reno first creates congestion by continuously increasing the size of its congestion window, thus sending more and more packets, until it detects packet loss by three duplicate ACK's or timeout and after which it reduces the size of its congestion window, for details see : [Allman *et al.* 1999]. This reactive cycle keeps on going with TCP Reno connections. Whereas in case of proactive congestion control we detect and control congestion before it actually causes the packet loss.

In this Section we will briefly look at some of the proactive active congestion control protocols before describing the TCP Vegas.

3.1.1 CARD Algorithm

The Congestion Avoidance using Round trip Delay (CARD) algorithm was presented in [Jain 1989] and it is based on fact, that as load on network increases router queue will build up resulting in an increased round trip time R.

It is assumed that only implicit congestion information, such as increase in queuing delay and thus R, decrease in throughput and timeouts, is available at end hosts of a network which is being treated as a black box, [Jain 1989]. The round trip time (or delay) R and throughput T are functions of (congestion) window W, i.e. $R = f(W), T = f(W)$, and corresponding network power P, is defined as:

$$P = \frac{T^{\alpha_c}}{R}, \tag{3.1}$$

where α_c is a positive real number whose value is usually set equal to 1 for maximizing the power, see [Jain and Ramakrishnan 1988] and [Ramakrishnan and Jain 1990] and references cited in them. We can rewrite equation (3.1) as:

$$\log P = \alpha_c \cdot \log(T) - \log(R). \tag{3.2}$$

At knee point of Figure 1.2, we have

$$\frac{dP}{P} = \alpha_c \frac{dT}{T} - \frac{dR}{R}. \tag{3.3}$$

By equating equation (3.2) to 0, we can have $\alpha_c \frac{dT}{T} = \frac{dR}{R}$. Thus, before the knee point the inequality $\frac{dT}{T} > \frac{dR}{R}$ holds and beyond it $\frac{dT}{T} < \frac{dR}{R}$ is true, [Jain 1989]. By substituting $T = \frac{W}{R}$ into equation (3.3), we can obtain following equation for the optimal (congestion) window:

$$\hat{W}_{opt} = \frac{\alpha_c}{1 + \alpha_c} \cdot \left(\frac{R}{\frac{d\,R}{dW}} \right). \tag{3.4}$$

The above results are valid for all types of networks and no assumption is made about the internal elements or variables of black box, such as deterministic or probabilistic distribution of service times etc, [Jain 1989]. Also, the author defines the normalized delay gradient (NDG) as $NDG = \frac{d\,R/dW}{R/W}$ which is used to determine the correct direction of window adjustment. For total number of n users the social optimum operating point for the $i-th$ user is obtained as follows:

$$\hat{W}_{soc_i} = \frac{\alpha_c}{\alpha_c + 1} \cdot \left(\frac{R}{\frac{\partial\, R}{\partial W_i}} \right) - \sum_{j \neq i}^{n} \hat{W}_j. \tag{3.5}$$

Similarly, the selfish optimum operating point is given by following equation:

$$\hat{W}_{sel_i} = \frac{\alpha_c}{\alpha_c + 1} \cdot \left(\frac{R}{\frac{\partial\, R}{\partial W_i}} \right). \tag{3.6}$$

For present (congestion) window W with round trip time R and old (congestion) window W_{old} with round trip time R_{old}, the overall CARD algorithm for maximum window size of W_{max} and minimum window of W_{min}, can be written as follows:

Every two R's

$$NDG \leftarrow \left(\frac{R-R_{old}}{R+R_{old}}\right) \cdot \left(\frac{W+W_{old}}{W-W_{old}}\right)$$

if $(NDG > 0)$ or $(W = W_{max})$;
Then Decrease (W);
elseif $(NDG \leq 0)$ or $(W = W_{min})$
Then Increase W;

This scheme uses the Additive Increase and Multiplicative Decrease (AIMD) algorithm, [Chiu and Jain 1989], with an increase factor of 1 and decrease factor of $1 - 2^{-3} = 0.875$, see: [Jain *et al.* 1987]. This value of decrease factor is chosen so that decrement in window can be done without floating point operation hardware and it can be carried out by simple logical shift instructions, [Jain *et al.* 1987], whereas in present Reno TCP the decrease factor is 0.5 due to reason of being more conservative as given in [Jacobson 1988].

The main drawback in this scheme is that in order to attain socially optimum (congestion) window size in real probabilistic networks, each user must know (congestion) window size of other users which is not realistic and acceptable design, [Jain 1989].

3.1.2 Tri-S Algorithm

Tri-S (Slow Start and Search) algorithm, first proposed in [Wang and Crowcroft 1991], does not perform repeated AIMD operations and quickly establishes fair operating point whenever there are major traffic changes. Once sharing of resources has been settled the operating point will remain unchanged until traffic change occur again. The operation of this algorithm depends on fact that sending rate flattens as network approaches the congestion state. Briefly, this algorithm can outlined as follows:

- The (congestion) window is set equal to one Basic Adjustment Unit (BAU), when a connection is initiated by a user. The source increases the (congestion) window by one BAU, each time an ACK is received until (congestion) window reaches maximum allowed by end user (receive window).

- Tri-S algorithm enters the decrease mode after timeout and the corresponding retransmission of lost packet. The (congestion) window is set equal to one BAU, and it checks normalized throughput gradient (NTG) upon receiving an ACK. The size of (congestion) window is increased by one BAU if NTG is above the threshold value of NTG_d, else it enters the increase mode.

  Throughput Gradient (TG), between previous and present (congestion) windows W_{n-1} and W_n, with corresponding throughputs $T(W_n)$ and $T(W_{n-1})$, is defined as:

$$TG(W_n) = \frac{T(W_n) - T(W_{n-1})}{W_n - W_{n-1}}, \tag{3.7}$$

whereas the corresponding NTG is obtained by $NTG(W_n) = \frac{TG(W_n)}{TG(W_1)}$. The value of NTG is close to 0 during the onset of congestion and close to 1 when network is lightly loaded, thus during normal operation $NTG \in [0, 1]$.

- During the increase mode, (congestion) window is increased by BAU/W_n for each received ACK. If the total increase is greater then the packet size, then NTG is checked and if NTG is less then a threshold NTG_i then decrement (congestion) window by one packet, else do nothing.

It is not a well designed algorithm and needs further research work. For instance, the precise relationship between operating point and two thresholds, NTG_i and NTG_d, is not well defined and poor synchronization of resource adjustment occurs after a change in traffic demand, [Wang and Crowcroft 1991].

3.1.3 DUAL Algorithm

This algorithm was first proposed in [Wang and Crowcroft 1992], and is based on changes in round trip time with changes in the level of congestion in a network. To explain it, let us consider a router having a Droptail queue with buffer size of B packets, service rate of μ packets/s and a bottleneck link of capacity C packets/s with $C \leq \mu$. For a packet entering the router's queue the round trip time, R, is a sum of propagation delay τ_p and queuing delay τ_d. Thus, we can write the following expression for round trip time R:

$$R = \tau_p + \tau_d. \tag{3.8}$$

At any time instant t the instantaneous queue size of router is $q(t)$ which gives the following expression for queuing delay τ_d:

$$\tau_d = \left\{ \frac{q(t) + 1}{\mu} \right\}. \tag{3.9}$$

Thus, as instantaneous queue size $q(t)$ fluctuates between minimum and maximum values of $q_{min}(t) = 0$ and $q_{max}(t) = B$, the value of round trip time, R, will also vary between minimum value of R_{min} and maximum value of R_{max}. Where R_{min} and R_{max} are given by the following expressions:

$$R_{min} = \left(\tau_p + \frac{1}{\mu} \right), \tag{3.10}$$

$$R_{max} = \tau_p + \left(\frac{B + 1}{\mu} \right). \tag{3.11}$$

The minimum round trip time, R_{min}, is also called based round trip time, R_b, in literature, c.f. [Brakmo *et al.* 1994b]. In this algorithm it is assumed that computation of retransmission timeout, (R_{to}), is accurate and packets are lost only when router buffer overflows, hence timeout can be viewed as a queue length signal threshold $q_{max}(t)$. In general the queue length signal

threshold, $q_i(t)$, and the corresponding round trip time signal threshold, R_i, of this algorithm are defined as:

$$q_i(t) = \alpha_d \cdot q_{max}(t), \tag{3.12}$$

$$R_i = R_{min} + \left\{ \frac{\alpha_d \cdot q_{max}(t)}{\mu} \right\}. \tag{3.13}$$

Therefore, by using equations (3.10) and (3.11), we can have the following relation for round trip time signal threshold R_i:

$$R_i = (1 - \alpha_d) \cdot R_{min} + \alpha_d \cdot R_{max}. \tag{3.14}$$

In equation (3.14) the parameter α_d determines the operating point, around which actual values of $q(t)$ and R will oscillate, and its value is chosen as 0.5 in [Wang and Crowcroft 1992]. At $\alpha_d = 0.5$ we will have $q_i(t) = 0.5 \cdot q_{max}(t)$ and $R_i = 0.5 \cdot (R_{min} + R_{max})$ as an operating point, giving maximum amplitude of queue size fluctuations and maintain R in middle of range between R_{max} and R_{min}. In [Wang and Crowcroft 1992], both queue length threshold (timeout) and round trip delay thresholds are used as congestion signals and a dual strategy has been devised for traffic adjustments.

Queue length threshold of Dual algorithm is based on timeout occurring due to dropping of packets from buffer overflow which is not very practicable for routers having large buffers. In such routers there will be larger queuing delays and response of DUAL will be very slow. Also the delay threshold based adjustment of DUAL algorithm needs accurate measurement of round trip time, which might not be possible in many cases e.g. re-routing of packets. Though this algorithm has drawbacks and limitations as given above, yet it is a good attempt to relate changes in measured round trip time at end hosts and state of congestion in bottleneck router. It also suggested that in end-to-end architecture of the Internet, the changes in round trip time, R, can be effectively used as a congestion indicator.

3.1.4 TCP Vegas

TCP Vegas, first proposed in [Brakmo *et al.* 1994a], is an end-to-end protocol for proactive congestion control in computer networks and its operation is partly similar to Tri-S algorithm explained in the Subsection 3.1.2.

Although both Tri-S scheme and TCP Vegas both look at changes in throughput, however they differ from each other in its calculation, [Brakmo *et al.* 1994a]. Tri-S computes throughput by dividing the number of bytes outstanding in the network by round trip time, whereas TCP Vegas computes actual and expected throughputs. The actual throughput of TCP Vegas is obtained by dividing the number of bytes sent in a round trip time obtained by distinguished segment (packet) and its expected throughput is obtained by dividing current congestion window size by the minimum round trip time. Also TCP Vegas does not look at changes in the slope of

throughput but alternatively it compares actual (measured) throughput with expected throughput to determine change in the size of its congestion window, [Brakmo *et al.* 1994a]. Thus, the existing literature suggests that TCP Vegas is an evolved form of previously known proactive congestion control attempts such as discussed in previous three Subsections.

Using x-kernel based, [Hutchinson and Peterson 1991], simulations it was proved in [Brakmo and Peterson 1995] that the TCP Vegas achieves 37 to 71 % more throughput then the most widely used TCP Reno. In [Ahn *et al.* 1995], TCP Vegas was re-evaluated on a wide area emulator and these experiments corroborated that it show higher efficiency than TCP Reno and also has much lesser packet retransmissions.

In the previous three Subsections we have discussed the major existing proactive congestion control algorithms with their drawbacks and limitations. We conclude that, as compared to CARD, DUAL and Tri S the TCP Vegas is the best proactive congestion control protocol and thus we further investigate its constituent algorithms in the next Section.

3.2 DESCRIPTION OF TCP VEGAS ALGORITHM

The operation of TCP Vegas depends on the following three algorithms:

- New Retransmission Mechanism.
- Modified Congestion Avoidance.
- Modified Slow Start.

These algorithms are briefly explained in the proceeding Subsections.

3.2.1 New Retransmission Mechanism

A procedure for the computation of retransmission timeout interval (R_{to}) is presented in [Postel 1981]. It consists of first determining the smoothed round trip time (R_s) from measured round trip time R by using the following Exponentially Weighted Moving Average (EWMA), [Young 1984], process with $0 \leq \alpha_r \leq 1$ as a smoothing factor:

$$R_s = \alpha_r \cdot R_s + (1 - \alpha_r) \cdot R. \tag{3.15}$$

Then, R_{to} is determined as follows:

$$R_{to} = \beta_r \cdot R_s. \tag{3.16}$$

In equation (3.16) the constant β_r accounts for variations in round trip time, R, and is known as delay variance factor. In [Postel 1981] it was suggested that α_r should have values between 0.8

to 0.9 and β_r should have value ranging from 1.3 to 2. It has been given in [Jacobson 1988] that for $\beta_r = 2$, R_{to} can adapt at most to a 30% of load variation which is not very good as above this point a TCP connection will respond to load increases by retransmitting packets that have been only delayed in transit. Thus, in order to avoid wasting bandwidth, the following alternative method of computation of R_{to} is presented in [Jacobson 1988]:

$$R_{to} = R_s + 2 \cdot D, \tag{3.17}$$

$$D = (1 - h) \cdot D + h * |R - R_s|, \tag{3.18}$$

where R_s is computed as given in equation (3.15) with $\alpha_r = 0.125$ and D is mean deviation with factor $h = 0.25$. Latter, in [Jacobson and Karels 1992], (revised version of [Jacobson 1988]), the equation (3.17) was slightly modified as $R_{to} = R_s + 4 \cdot D$ which has been used widely by different TCP implementations including Reno to compute its R_{to}, [Allman and Paxon 1999]. Also to compute round trip time and for performing timing decisions, a coarse grained timer (around 500 ms) is used in TCP Reno [Stevens 1995] and [Allman and Paxon 1999]. While maintaining a single timer is easier, but it is not a very accurate method which affects both Fast Retransmit and Fast Recovery algorithms used for packet retransmission. For further details about computation of TCP's retransmission timer and related issues see [Paxon and Allman 2000].

The retransmission mechanism of TCP Vegas is an extension of TCP Reno which is more accurate because it maintains the record of the system clock for each packet sent away from source. When an ACK arrives at a TCP Vegas source, it performs R calculation using present time and time stamp recorded for corresponding sequence number of the already sent packet. Then it uses this more accurate R to decide retransmission as in the following two situations, [Brakmo *et al.* 1994a]:

- In case of duplicate ACK arrival, TCP Vegas computes the difference between the current time and the time stamp recorded for respective sent packet. If this difference is greater then the R_{to} then that packet is retransmitted without waiting for triple duplicate ACK's.
- When a non duplicate ACK is received and if it is first or second after a retransmission then TCP Vegas checks whether the time interval since corresponding packet was sent is greater then R_{to}; if so then TCP Vegas retransmits that packet without waiting for duplicate ACK.

TCP Vegas still contains the coarse grained mechanism of TCP Reno, which can be employed if these new mechanisms do not work properly.

3.2.2 Congestion Avoidance Mechanism of TCP Vegas

In contrast to TCP Reno, TCP Vegas anticipates the onset of congestion state in advance by its modified congestion avoidance mechanism which is explained in this Section.

TCP Vegas sender computes the minimum R, called base round trip time R_b, for a given connection during the period of no congestion, which is usually R of the first sent packet before the router queue builds up. The congestion window W_v of TCP Vegas is updated by following Congestion Avoidance algorithm, [Brakmo *et al.* 1994a]:

- Compute the expected throughput, T_{exp}, by using R_b in following relation

$$T_{exp} = \frac{W_v}{R_b}. \tag{3.19}$$

- Then calculate actual throughput, T_{act}, by using actual round trip time R:

$$T_{act} = \frac{W_v}{R}. \tag{3.20}$$

- Next estimate backlog, D, in router buffers by following relation:

$$D = (T_{exp} - T_{act}) \cdot R_b. \tag{3.21}$$

- Finally using two defined constants $\alpha_v = 1$ and $\beta_v = 3$, it updates congestion window by following set of rules:

$$W_v = \begin{cases} W_v + 1, & \text{if } D < \alpha_v; \\ W_v, & \text{if } \alpha_v \leq D \leq \beta_v; \\ W_v - 1, & \text{if } D > \beta_v. \end{cases} \tag{3.22}$$

In equation (3.22), the constants α_v and β_v control the Additive increase and Additive decrease (AIAD) operation in W_v of TCP Vegas. Further we will be concentrating only on the congestion avoidance phase of TCP Vegas, due to its fundamental role and importance.

3.2.3 Modified Slow Start

In order to avoid packet loss at slow start, TCP Vegas has incorporated the congestion detection mechanism with slight modifications into Slow Start (SS) phase as well. During SS the exponential growth of W_v occurs only every other R. In between period the W_v stays fixed so a valid comparison of expected and actual rates can be made. When the actual rate falls below the expected rate by a certain amount, called threshold γ, the TCP Vegas switches from slow start to linear increase/decrease mode.

During SS, two packets are send for every received ACK, thus if there is already congestion in network then this modified slow start mechanism will not work properly. The packets send will be lost and there will be no mechanism for getting feedback so TCP Vegas sender can slow

down the data sending rate by decreasing its W_v, [Brakmo *et al.* 1994a]. Further in this Chapter we do not consider the SS and retransmission mechanisms and focus only on the congestion avoidance phase as has been done in [Mathis *et al.* 1997] for TCP Reno and in [Bonald 1998] for TCP Vegas. In the next Section we will briefly review the previous work done on analyzing the fairness of TCP Vegas in literature.

3.3 REVIEW OF PREVIOUS WORK DONE ON TCP VEGAS AS A FAIR END-TO-END PROTOCOL

A fluid model of congestion avoidance phase of TCP Vegas in Droptail router environments is presented in [Bonald 1998], wherein flow control efficiency of both TCP Reno and TCP Vegas is compared while neglecting slow start phase, loss detection and re-transmission mechanisms. It is shown that the queuing delay will be decreased and whole of the available bandwidth will be fully utilized if all of the connections are employing TCP Vegas only. Further, it has been proved analytically that TCP Vegas connections having different round trip times will share the available bandwidth fairly among themselves. Thus, TCP Vegas will not show bias against connections with larger round trip times. Hence, if only TCP Vegas is used then there would not be problem of unfairness among the competing connections. It is also suggested that TCP Vegas is suitable for large bandwidth delay product links such as satellite links. The main weakness of the model presented in [Bonald 1998] is that it does not consider the situation in which both TCP Reno and TCP Vegas share a bottleneck link. Similar results have also been reported in [Mo *et al.* 1999], where a heterogenous scenario of TCP Vegas and TCP Reno is also analyzed.

In [Ait Hellal and Altman 1997] expressions for congestion window size and round trip time have been derived for slow start and congestion avoidance phases of both TCP Reno and TCP Vegas. However, analytical results obtained are not validated by extensive simulations. Fairness of TCP Vegas has also been studied in [Low *et al.* 2001], where it is shown that TCP Vegas can be configured to have proportional and weighted proportional fairness. The proportionally fair behavior of TCP Vegas has also been re-confirmed in [Biro and Luong 2001].

A duality model of TCP Vegas has been proposed in [Low *et al.* 2001] and it is shown that objective of TCP Vegas is to maximize aggregate resource utility subject to capacity constraints of network. It is also shown that TCP Vegas is a dual method to solve the maximization problem, [Low and Lapsley 1999]. In order to avoid problem of unfairness and persistent congestion it is recommended to use the Random Exponential Marking (REM) algorithm with TCP Vegas. However, the main problem with REM algorithm is proper selection of a variable ϕ which must be known globally, see: [Low and Lapsley 1999], [Athuraliya and Low 2000] and [Athuraliya *et al.* 2001].

A single link model of TCP Vegas has been presented in [Boutremans and Boudec 2000] and effects of overestimation of propagation delay on fairness are determined both for $\alpha_v = \beta_v$ and $\alpha_v < \beta_v$ cases. It is concluded that for achieving high fairness and stability, TCP Vegas

should use $\alpha_v = \beta_v$ configuration along with correct estimation of propagation delay. However, no solution is proposed to reduce the effects of errors in estimation of propagation delay.

Hence, due to its high throughput and good fairness properties in homogenous environments, TCP Vegas satisfies the main objectives of network protocols as given in [Allman and Falk 1999]. Whereas the major problems associated with TCP Vegas are re-routing of packets, persistent congestion and in-compatibility with TCP Reno. The former two problems (re-routing of packets and persistent congestion) are related to the proper estimation of propagation delay, τ_p, or base round trip time and they mostly affect the throughput of TCP Vegas connections. They have been partially addressed in [Richard *et al.* 1998] and [Low *et al.* 2001], respectively. All of the above mentioned problems with TCP Vegas are inter-related and can occur at the same time in practical situations. However, further in this Chapter we will focus only on the incompatibility issue of TCP Vegas with TCP Reno as it directly affects the fair allocation of bandwidth among competing flows. Therefore, in the next Subsection we present previous work done to address this issue.

3.3.1 Incompatibility of TCP Vegas and TCP Reno

The incompatibility issue of TCP Vegas and TCP Reno is most important for future deployment of TCP Vegas, because dominant version of TCP in the current Internet and other computer networks such as computational grids, is TCP Reno as has been investigated in [Paxson 1997b], [Paxson 1999] and [Weigle and chun Feng 2001].

In [Padhye and Floyd 2002] and [Floyd and Paxon 2001], it has been found that recently there has been a significant deployment TCP New Reno and TCP SACK in the current Internet. These new protocols retain the congestion control algorithms of TCP Reno, hence their interaction with TCP Vegas would be almost similar as that of TCP Reno. Although in heterogenous environments TCP Reno user gets higher bandwidth then TCP Vegas users when buffer sizes are large, the packets of TCP Reno will experience long queuing delays and jitter due to larger queue size, [Mo *et al.* 1999]. TCP Vegas on other hand maintains a very small backlog of packets in queue, which results in shorter queuing delay and once TCP Vegas reaches its equilibrium state then delay jitter also becomes very small, see: [Mo *et al.* 1999].

In [Ahn *et al.* 1995], emulation techniques are used to study the interaction of TCP Vegas and TCP Reno in heterogenous environments. These experiments showed that in head-to-head transfers TCP Reno will get 50 % more throughput then TCP Vegas.

Incompatibility issue has been investigated in [Mo *et al.* 1999], [Lai and Yao 2000], [Low 2000], [Bonald 1998] and [Boutremans and Boudec 2000] using analytical/simulations techniques and it has been found that TCP Vegas shows better fairness and higher throughput than TCP Reno in homogenous simulation scenarios but it cannot perform well in heterogenous scenarios with TCP Reno.

To improve the performance of of TCP Vegas in a heterogenous environments comprising

TCP Reno and TCP Vegas, use of a RED router, [Floyd and Jacobson 1993], is suggested in [Mo *et al.* 1999]. It is observed through simulations that relative performance of TCP Vegas against TCP Reno depends upon values of max_{th} and min_{th} of RED and throughput of TCP Reno increases and that of TCP Vegas decreases at higher values of thresholds. The idea of using RED algorithm for improving the fairness between TCP Reno and TCP Vegas was also studied in [Lai 2001], in which authors varied max_{th} parameter of RED, to get more bandwidth for TCP Vegas. The main flaw in their scheme is that, as the max_{th} of RED is lowered the link utilization decreases, because in this case the RED will mark/drop more packets, see: [Floyd and Jacobson 1993]. It will reduce the overall throughput and thus not a very efficient scheme.

In [Hasegawa *et al.* 2000], two solutions are presented to make TCP Vegas compatible with TCP Reno namely: using TCP Vegas with Stabilized RED (SRED) algorithm routers, [Ott *et al.* 1999], and modifying the basic TCP Vegas algorithm. It is suggested that TCP Vegas should behave similarly to TCP Reno, when the former is not getting fair share of bandwidth; thus it divides the operation of TCP Vegas into two modes of different aggressiveness, which are moderate and aggressive modes. Moderate mode is similar to normal TCP Vegas and aggressive mode is similar to TCP Reno. The main weakness of these algorithms is arbitrary choice of different tuning parameters, see [Hasegawa *et al.* 2000].

In order to overcome unfairness and low throughput characteristics of TCP Vegas, while competing with TCP Reno in heterogenous environments, we propose a RED, [Floyd and Jacobson 1993], based ECN, [Ramakrishnan *et al.* 2001], solution in Section 3.6. Whereas in the next Section we first analyze the interaction between TCP Vegas and TCP Reno.

3.4 ANALYSIS OF TCP VEGAS AND TCP RENO INTERACTION

In this Section we analyze a heterogenous environment where TCP Vegas and TCP Reno are sharing a common bottleneck link of capacity C packets/s and having a two way propagation delay of τ_p seconds. Both TCP sources are operating in the congestion avoidance phase. Droptail router having service rate of $\mu \leq C$ packets/s is being used with the bottleneck link. Next, in the following two Subsections we derive the analytical expressions for throughputs and Jain's Fairness Index (JFI) for the heterogenous scenario of TCP Vegas and TCP Reno.

3.4.1 Throughputs of TCP Reno and TCP Vegas

Let the queue size is $q(t)$ packets and congestion windows of TCP Vegas source and TCP Reno are $W_v(t)$ and $W_r(t)$ packets, respectively. We can write equations for minimum round trip time called based round trip time, R_b, and round trip time, $R(t)$, as follows:

$$R_b = \tau_p + \frac{1}{\mu}, \tag{3.23}$$

$$R(t) = \tau_p + \left(\frac{q(t)+1}{\mu}\right). \tag{3.24}$$

Further, as derived in [Mathis *et al.* 1997], during congestion avoidance phase, for random packet loss at constant probability p, the congestion window, W, of TCP Reno is given by:

$$W = \sqrt{\frac{8}{3p}}. \tag{3.25}$$

In order to determine congestion window of TCP Reno, W_r, for a given fixed value of loss probability p, we have plotted equation (3.25) in Figure 3.1. Further, we can generalize equation

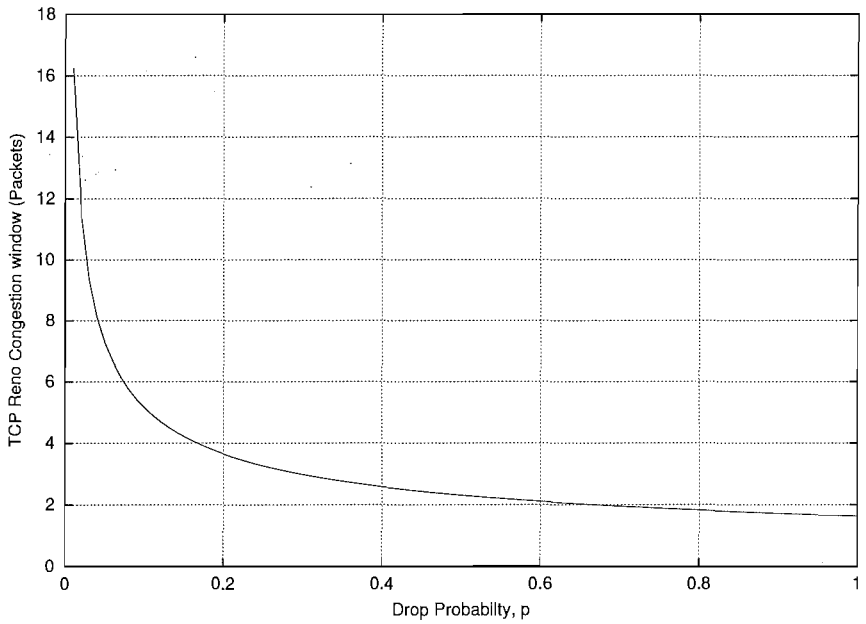

Figure 3.1 Variations in congestion window of TCP Reno, W_r (packets), for different values of loss probability, p.

(3.25) as follows:

$$W_r(t) = \frac{K_r}{\sqrt{p(t)}}, \tag{3.26}$$

where K_r is a constant. The expression for throughput of TCP Reno, $T_r(t)$, is given by:

$$T_r(t) \equiv \frac{W_r(t)}{R(t)} = \frac{K_r}{\sqrt{p(t)}} \cdot \frac{1}{R(t)}. \tag{3.27}$$

Substituting equation (3.24) into (3.27), we get:

$$T_r(t) = \frac{K_r}{\sqrt{p(t)}} \cdot \frac{\mu}{\mu \cdot \tau_p + q(t) + 1}. \tag{3.28}$$

To further analyze the throughput generated by TCP Vegas, we can write D given in equation (3.21) as:

$$D = \left(\frac{W_v(t)}{R_b} - \frac{W_v(t)}{R(t)} \right). \tag{3.29}$$

We can substitute (3.23) and (3.24) into (3.29) to get following expression

$$D = W_v(t) \cdot \left(\frac{q(t)}{q(t) + C\tau_p + 1} \right) \cdot R_b. \tag{3.30}$$

Thus, we can write congestion avoidance phase of TCP Vegas as:

$$W_v(t+1) = \begin{cases} W_v(t) + 1, & \text{if } \alpha_v \cdot \left(\frac{q(t)+\mu\tau_p+1}{q(t)} \right) > W_v(t); \\ W_v(t), & \text{if } \alpha_v \leq D \leq \beta_v; \\ W_v(t) - 1, & \text{if } \beta_v \cdot \left(\frac{q(t)+\mu\tau_p+1}{q(t)} \right) < W_v(t). \end{cases} \tag{3.31}$$

We can obtain throughput expression of TCP Vegas, $T_v(t)$, using equations (3.24) and equation (3.31) for each case given in (3.31), by the following relation:

$$T_v(t) = \frac{W_v(t)}{R(t)}. \tag{3.32}$$

Hence, as we increase the values of α_v and β_v in TCP Vegas, the product $\alpha \cdot \left(\frac{q(t)+C\tau_p+1}{q(t)} \right)$ will become larger in magnitude which will keep $W_v(t)$ in an increasing phase, which will cause an increase in throughput. Thus, in a heterogenous environment TCP Vegas can also get higher throughput as TCP Reno, by increasing the values of its α_v and β_v parameters. This fact has been also proved in latter Sections by using ns simulations and further it has been used to design a new ECN based algorithm for TCP Vegas, which eliminate the incompatibility of TCP Vegas with TCP Reno.

3.4.2 Fairness between TCP Reno and TCP Vegas

In heterogenous scenarios packet loss will occur if the overall input traffic rate exceeds the bottleneck link capacity C. We can write the loss free condition as:

$$\frac{W_v(t)}{R(t)} + \frac{W_r(t)}{R(t)} \leq C. \tag{3.33}$$

At the equilibrium point, we can get following expression for $W_v(t)$ from equation (3.33):

$$W_v(t) = R(t) \cdot C - W_r(t). \tag{3.34}$$

Substituting equations (3.26) and (3.31) into (3.34) we get:

$$W_v(t) = R(t) \cdot C - \frac{K_r}{\sqrt{p(t)}}. \tag{3.35}$$

After substituting equation (3.24) in above equation (3.35) we get:

$$W_v(t) = q(t) + \tau_p \cdot C + 1 - \frac{K_r}{\sqrt{p(t)}}. \tag{3.36}$$

Thus, we have expressions for the congestion windows of TCP Vegas and TCP Reno in the heterogenous environment. For TCP Reno's throughput, $T_r(t)$, and TCP Vegas's throughput, $T_v(t)$, the Jain's Fairness Index (JFI) is given by:

$$JFI = \frac{\{T_r(t) + T_v(t)\}^2}{2 \cdot \{T_r(t)^2 + T_v(t)^2\}}. \tag{3.37}$$

After substituting the values of $T_r(t)$ and $T_v(t)$ in equation (3.37) we get

$$JFI = 0.5 \cdot \left[\frac{\{W_r(t) + W_v(t)\}^2}{\{W_r(t)^2 + W_v(t)^2\}} \right]. \tag{3.38}$$

Further simplification of equation (3.38) will yield:

$$JFI = 0.5 + \left\{ \frac{W_r(t) \cdot W_v(t)}{W_r(t)^2 + W_v(t)^2} \right\}. \tag{3.39}$$

From equation (3.39) we can see that $JFI = 1$ when $W_v = W_r \equiv \frac{K_r}{\sqrt{p(t)}}$, which on substitution into equation (3.36) gives $RC = \frac{2K_r}{p(t)}$ as the condition of maximum fairness. Further, substituting equations (3.26) and (3.36) into equation (3.39) gives:

$$JFI = 0.5 + \left[\frac{\left(\frac{K_r}{\sqrt{p(t)}}\right) \cdot \left(R(t) \cdot C - \frac{K_r}{\sqrt{p(t)}}\right)}{\left(\frac{K_r}{\sqrt{p(t)}}\right)^2 + \left(R(t) \cdot C - \frac{K_r}{\sqrt{p(t)}}\right)^2} \right]. \tag{3.40}$$

In order to plot variations of JFI with bandwidth-delay product, we substitute $K_r = \sqrt{\frac{8}{3}}$, $p(t) = p$ and $R(t) = R$ (i.e. assume that packet dropping probability, p, and round trip time,R, are time invariant quantities as in [Mathis *et al.* 1997]) into equation (3.40) to get the following expression:

$$JFI = 0.5 + \frac{1.632RCp^{-0.5} - 2.666p^{-1}}{5.333p^{-1} - 3.265RCp^{-0.5} + (RC)^2}. \tag{3.41}$$

Equation (3.41) is plotted in Figure 3.2 for different values of packet dropping probability, p, and bandwidth delay product RC. We vary p from 0.0001 to 0.1, corresponding to TCP Reno's congestion window of 163 to 5 packets, and bandwidth-delay product RC from 1 to 40 packets. Figure 3.2 shows that for a fixed value of p, the JFI increases with an increase in RC until its maximum value of 1 (when $W_r = W_v$) and then it starts to decrease again. In the JFI plot shown in Figure 3.2, TCP Vegas is receiving less than its fair bandwidth share for (p, RC) combinations to the left of the ridge while TCP Reno suffers from unfairness to the right of the ridge.

We can also have the following alternative expression for JFI in terms of TCP Reno congestion window, W_r, by simplifying equation (3.39):

$$JFI = 0.5 + \left\{ \frac{W_r \cdot RC - W_r^2}{2W_r^2 - 2W_r \cdot RC + (RC)^2} \right\}. \tag{3.42}$$

Equation (3.42) is plotted in Figure 3.3, which again shows that the balance of fairness between TCP Reno and TCP Vegas shifts in the favour of the latter at high bandwidth- delay products.

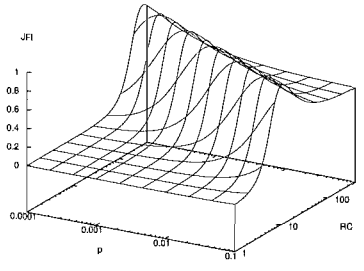

Figure 3.2 JFI variation with drop probability p and bandwidth delay product RC (packets).

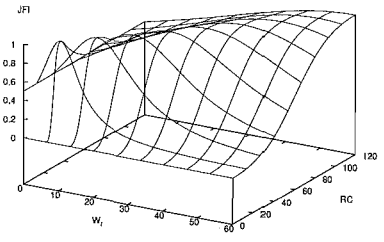

Figure 3.3 JFI variation with W_r (packets) and bandwidth delay product RC (packets).

The expression for JFI given in equation (3.40) is for single TCP Vegas and TCP Reno connections, which can be further generalized to a scenario when there are N_v TCP Vegas connections and N_r TCP Reno connections, competing against each other for bandwidth over the same bottleneck link of capacity C. The round trip time of all TCP Vegas and TCP Reno connections will be same if the two way propagation delay τ_p is the same for all connections, as each connection will experience the same queuing delay $q(t)/C$. Thus we will have the following expression at equilibrium:

$$\frac{N_v \cdot W_v(t)}{R(t)} + \frac{N_r \cdot W_r(t)}{R(t)} = C. \tag{3.43}$$

After a similar derivation procedure the following expression for JFI can be obtained from

equations (3.37) and (3.43):

$$JFI = 0.5 + \frac{\frac{R(t)\cdot C\cdot K_r}{N_v\cdot\sqrt{p(t)}} - \frac{N_r\cdot K_r^2}{N_v\cdot p(t)}}{\left(\frac{R(t)\cdot C}{N_v} - \frac{K_r\cdot N_r}{\sqrt{p(t)}\cdot N_v}\right)^2 + \frac{K_r^2}{p(t)}}. \quad (3.44)$$

The above equation (3.44) can be further solved for any number of TCP Vegas and TCP Reno connections.

Remark

If the round trip times of different TCP flows are not equal then we will have following equilibrium condition:

$$\left(\frac{N_{v1}\cdot W_{v1}(t)}{R_{v1}(t)} + \frac{N_{v2}\cdot W_{v2}(t)}{R_{v2}(t)} + \cdot\cdot\right) + \left(\frac{N_{r1}\cdot W_{r1}(t)}{R_{r1}(t)} + \frac{N_{r2}\cdot W_{r1}(t)}{R_{r2}(t)} + \cdot\cdot\right) = C. \quad (3.45)$$

The exact solution of equation (3.45) is non-trivial as the round trip time given by equation (3.24) is a function of queue size $q(t)$, i.e. $R(t) = f(q(t))$, and an expression of $q(t)$ does not exist in the most general case of incoming traffic.

In the next Section we perform ns-based simulations to show the fairness characteristics of TCP Reno and TCP Vegas both in homogenous environments with different round trip times and also in heterogenous environments.

3.5 SIMULATIONS

3.5.1 Network Topology

We simulate a network consisting of a TCP Reno source and a TCP Vegas source sharing a bottleneck link as shown in Figure 3.4. We use Droptail and RED routers, [Floyd and Jacobson 1993], at the bottleneck link and ECN is not employed unless explicitly stated. The links between nodes n0 to n2, nodes n1 to n2, nodes n3 to n4 and nodes n3 to n5 are having a capacity of 10 Mbps with propagation delay of 10 ms. TCP Reno sends data between nodes n0 and n4 and TCP Vegas sends data between nodes n1 to n5. The bottleneck link between nodes n2 to n3 has capacity, C, of 5 Mbps with propagation delay of 10 ms. In order to analyze the performance of two TCP's we use congestion window variations and throughput as metrics and plot their graphs after simulation experiments as given in the following Subsections.

3.5.2 Experiment 1

In the first part of this experiment we have two TCP Reno connections and in the second part we have two TCP Vegas connections, each forming a homogenous scenario in the simulation topology shown in Figure 3.4. In each part of this experiment, both TCP's have different round trip

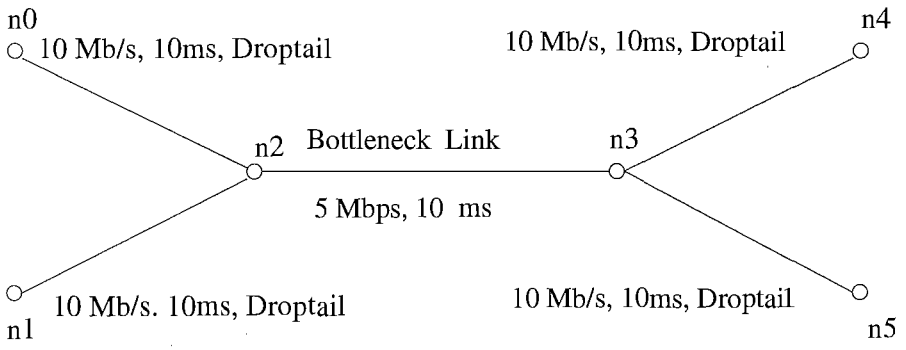

Figure 3.4 Network topology for simulations.

times. We modify the round trip time of one of the TCP connections by altering the propagation delay on links n1-n2 from 10ms to 1ms, while all other parameters of the network are kept the same.

For two TCP Reno connections we plot congestion windows, received bytes, throughputs and JFI in Figures 3.5, 3.7, 3.9, and 3.11, respectively. We observe that TCP Reno is unfair for the connection having longer round trip time as found by other researchers c.f. [Henderson *et al.* 1998]. The congestion window of the TCP Reno connection having the longer round trip time is smaller than of the connection having the shorter round trip time. The same is true for the throughput. The JFI plot further shows the unfairness characteristics of TCP Reno for connections with different round trip times. Hence, these simulations suggest the need to find new protocols for end hosts which can avoid such bias against connections having longer round trip times, thus providing better fairness.

For two TCP Vegas connections we plot congestion windows, received bytes, throughputs and JFI in Figures 3.6, 3.8, 3.10 and 3.12, respectively. For better fairness between both connections of TCP Vegas we make $\alpha_v = \beta_v$, as in [Hasegawa *et al.* 1999], and set their value equal to 2. We observe that the congestion windows of the two connections are close to each other. The throughputs of the two TCP Vegas connections are also close to each other and their sum is higher than in previous case of two TCP Reno connections. The JFI of the two connections is close to 1, which shows the good fairness characteristics of TCP Vegas in homogenous environments with different round trip times. Next we perform another experiment to determine the mutual interaction between the two types of TCP, Reno and Vegas, in a heterogenous simulation environment.

3.5.3 Experiment 2

In this experiment we employ a TCP Reno and a TCP Vegas source having the same propagation delays, and thus round trip times, in the simulation topology shown in Figure 3.4. First we perform simulations with TCP Reno and default TCP Vegas ($\alpha_v = 1$, $\beta_v = 3$), [Brakmo *et al.* 1994a]. We find that at the end of the simulation time (100 s), TCP Reno and TCP Vegas

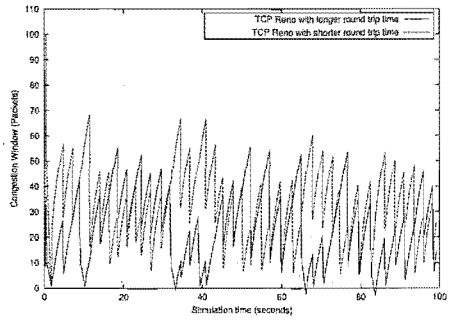

Figure 3.5 Congestion windows of two TCP Reno connections with different round trip times.

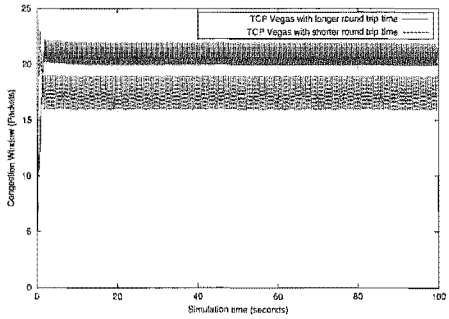

Figure 3.6 Congestion windows of two TCP Vegas connections with different round trip times.

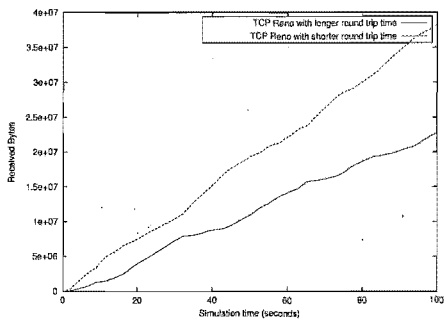

Figure 3.7 Received bytes of two TCP Reno connections with different round trip times.

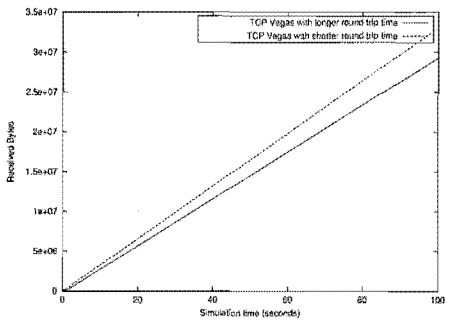

Figure 3.8 Received bytes of two TCP Vegas connections with different round trip times.

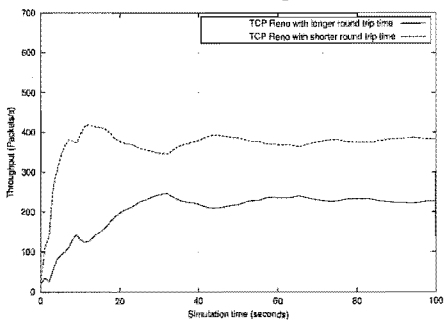

Figure 3.9 Throughput of two TCP Reno connections with different round trip times.

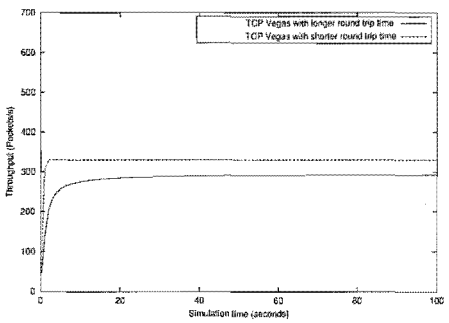

Figure 3.10 Throughput of two TCP Vegas connections with different round trip times.

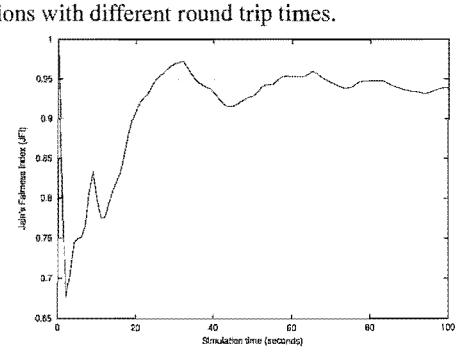

Figure 3.11 JFI of two TCP Reno connections with different round trip times.

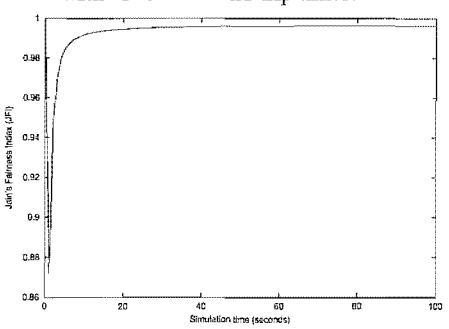

Figure 3.12 JFI of two TCP Vegas connections with different round trip times.

get throughputs of 553 and 62 packets/s, respectively. Thus giving $JFI = 0.61$, which is quite low, hence the TCP Vegas needs to be improved further.

Next, in order to get better fairness properties we make $\alpha_v = \beta_v$ for the TCP Vegas connection, [Hasegawa *et al.* 1999] and [Boutremans and Boudec 2000], and gradually increase their values from 2 to 32. The resulting congestion windows, received bytes, throughputs and JFI for the $\alpha_v = \beta_v = 2$ case are plotted in Figures 3.13, 3.14, 3.15 and 3.16, for the $\alpha_v = \beta_v = 8$ case they are plotted in Figures 3.17, 3.18, 3.19 and 3.20, for the $\alpha_v = \beta_v = 16$ case they are plotted in Figures 3.21, 3.22, 3.23 and 3.24, for the $\alpha_v = \beta_v = 20$ case they are plotted in Figures 3.25, 3.26, 3.27 and 3.28, for the $\alpha_v = \beta_v = 25$ case they are plotted in Figures 3.29, 3.30, 3.31 and 3.32, and for the $\alpha_v = \beta_v = 32$ case they are plotted in Figures 3.33, 3.34, 3.35 and 3.36, respectively.

For the $\alpha_v = \beta_v = 2$ case there is gross unfairness in bandwidth sharing between TCP Reno and TCP Vegas. The JFI is at a low value i.e. around 0.60. In the next simulation for $\alpha_v = \beta_v = 8$ the TCP Vegas performance showed a small improvement by getting slightly more bandwidth share and JFI also improves to 0.82. In the case of $\alpha_v = \beta_v = 16$, TCP Vegas performance improves considerably and its congestion window is close to that of TCP Reno. The JFI also climbs to values ranging from 0.98 to 1. Similar results can be seen for the $\alpha_v = \beta_v = 20$ case. For the cases of $\alpha_v = \beta_v = 25$ and $\alpha_v = \beta_v = 32$, the TCP Vegas connection becomes more aggressive than the TCP Reno connection, grabbing more bandwidth and thus causing a decrease in the value of JFI.

This experiment shows that the throughput of TCP Vegas can be improved as desired in a heterogenous environment of TCP Reno by increasing/decreasing the α_v and β_v parameters of TCP Vegas. In Figure 3.37 we plot the variations in throughput of TCP Reno and TCP Vegas with an increase in $\alpha_v = \beta_v$ parameter of TCP Vegas. We find that close to $\alpha_v = \beta_v = 20$ the two throughputs would be exactly equal for the network topology under consideration and JFI will reach the maximum value of unity. Below these values of α_v and β_v TCP Reno will get more throughput than TCP Vegas and the value of JFI will be lower than unity. Also, above these values of α_v and β_v the throughput of TCP Vegas will become greater than TCP Reno and JFI will drop again.

We also determine the number of bytes received by both TCP receivers at the end of each simulation and present these values in Table 3.1. These values of received bytes show similar results to those discussed above. There are oscillations in the congestion windows of both TCP Vegas connections due to equal values of their α_v and β_v parameters, [Hasegawa *et al.* 1999]. This phenomenon occurs in TCP Vegas because for the case of $\alpha_v = \beta_v$ the condition of unchanging congestion window size in equations (3.22) and (3.31) is eliminated.

Next, we summarize the results of the two simulation experiments. The homogenous environment of two TCP Reno and two TCP Vegas connections with different round trip times, simulated in Subsection 3.5.2, leads to the following observations:

- TCP Reno connections having longer round trip times will get less bandwidth as compared

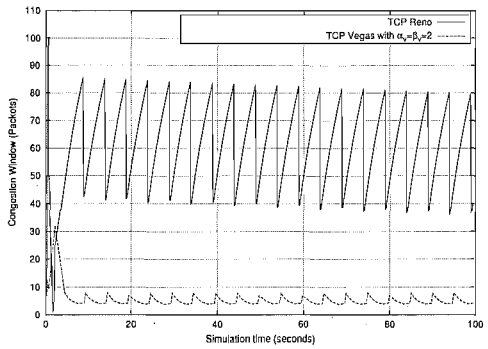

Figure 3.13 Congestion windows of TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 2$) connections.

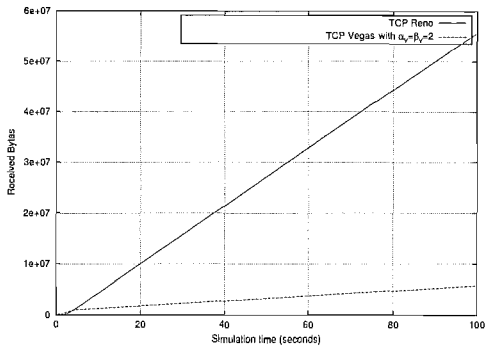

Figure 3.14 Received bytes of TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 2$) connections.

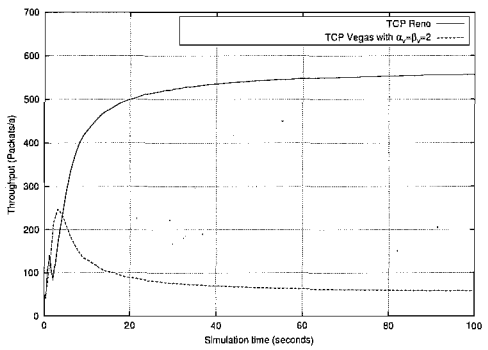

Figure 3.15 Throughputs of TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 2$) connections.

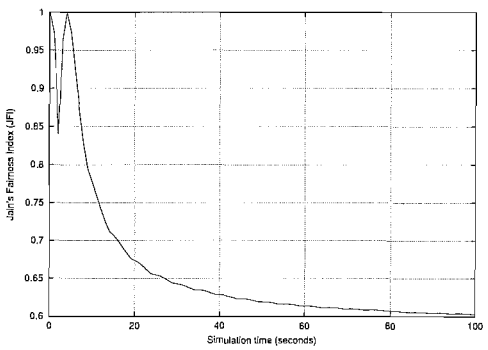

Figure 3.16 JFI, TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 2$).

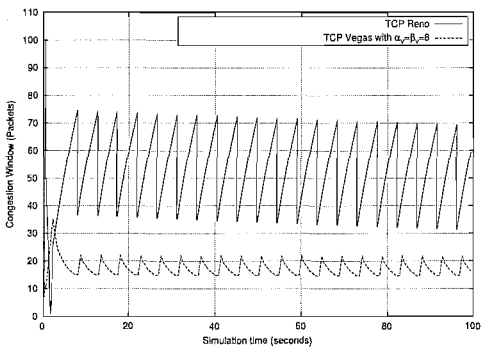

Figure 3.17 Congestion windows for TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 8$) connections.

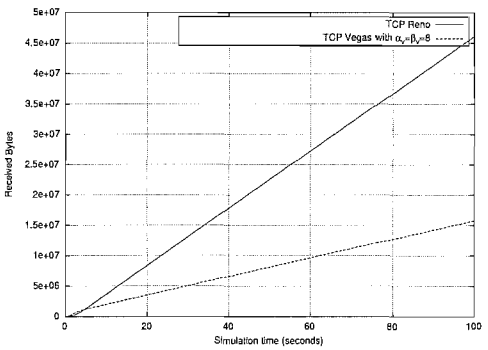

Figure 3.18 Received bytes for TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 8$) connections.

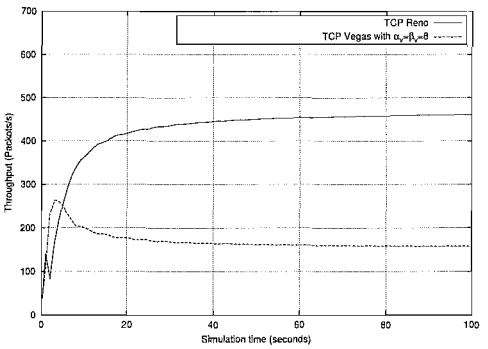

Figure 3.19 Throughputs for TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 8$) connections.

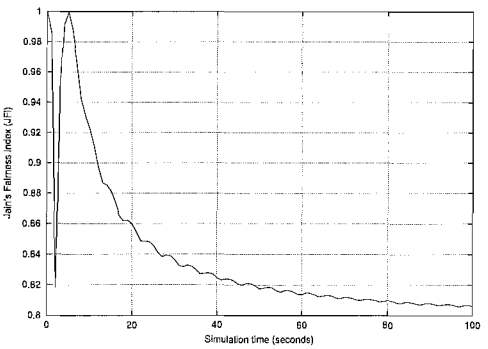

Figure 3.20 JFI for TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 8$) connections.

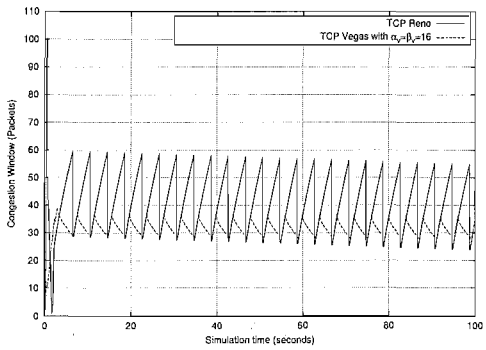

Figure 3.21 Congestion windows for TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 16$) connections.

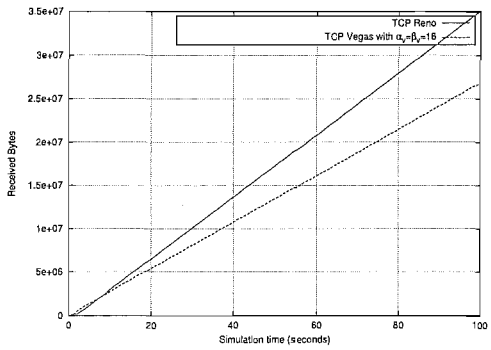

Figure 3.22 Received bytes for TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 16$) connections.

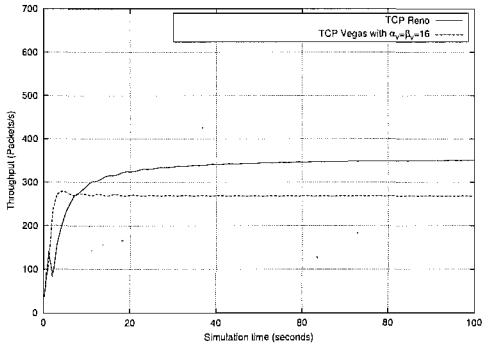

Figure 3.23 Throughputs for TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 16$) connections.

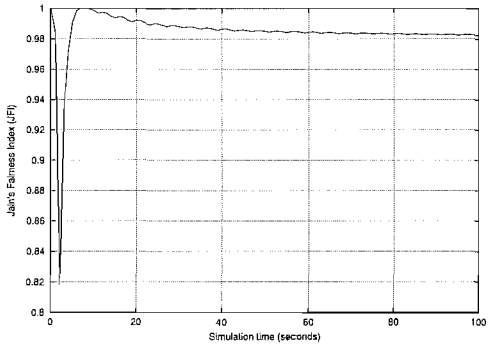

Figure 3.24 JFI for TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 16$) connections.

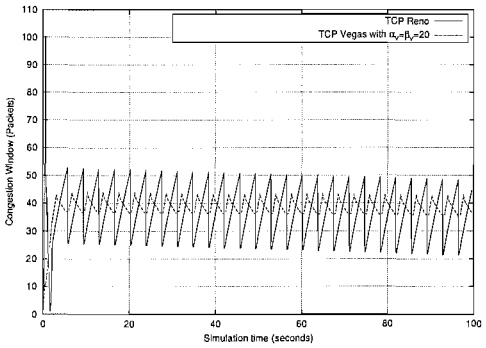

Figure 3.25 Congestion windows for TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 20$) connections.

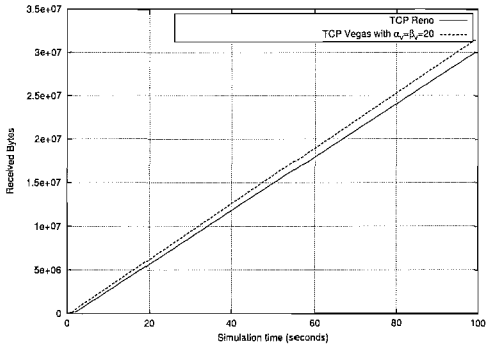

Figure 3.26 Received bytes for TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 20$) connections.

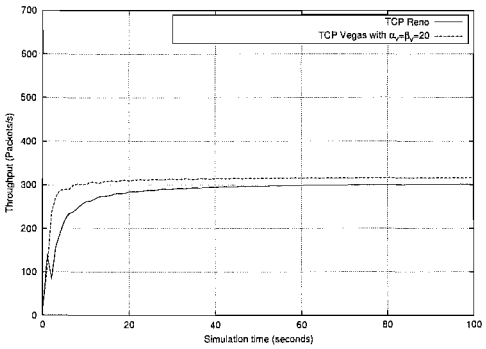

Figure 3.27 Throughputs for TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 20$) connections.

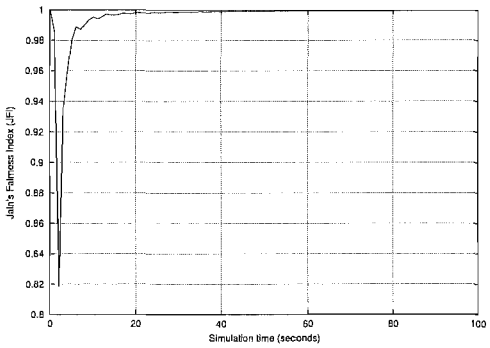

Figure 3.28 JFI for TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 20$) connections.

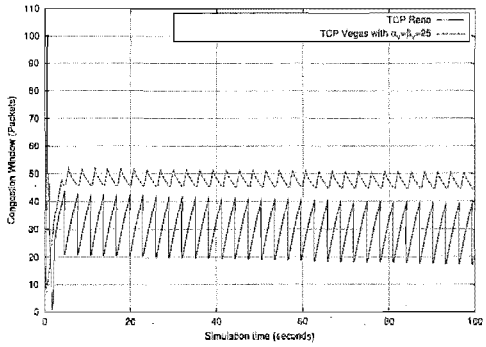

Figure 3.29 Congestion window for TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 25$) connection.

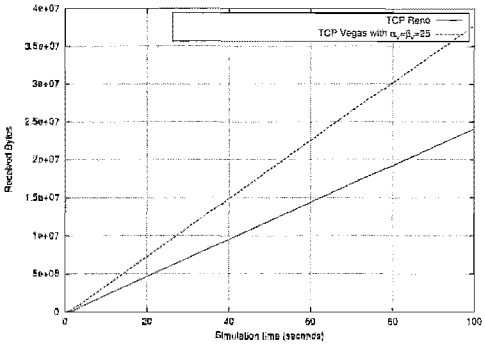

Figure 3.30 Received bytes for TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 25$) connections.

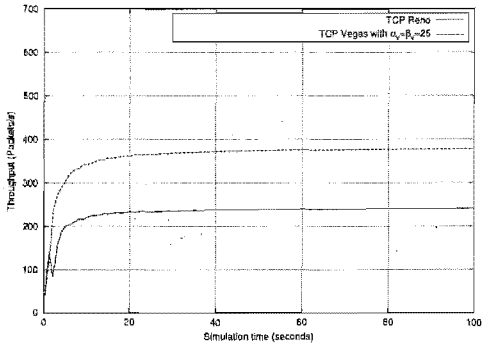

Figure 3.31 Throughputs for TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 25$) connections.

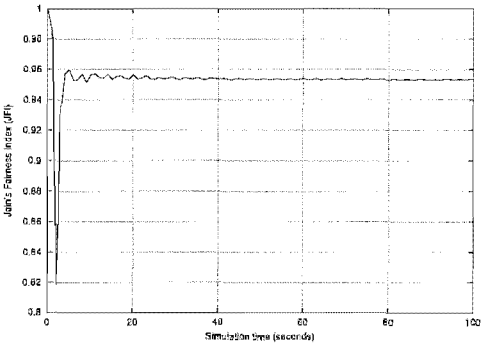

Figure 3.32 JFI for TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 25$) connections.

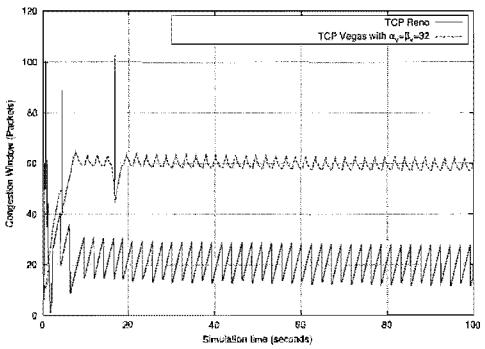

Figure 3.33 Congestion windows for TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 32$) connections.

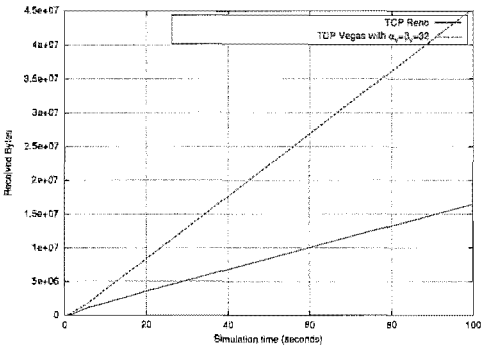

Figure 3.34 Received bytes for TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 32$) connections.

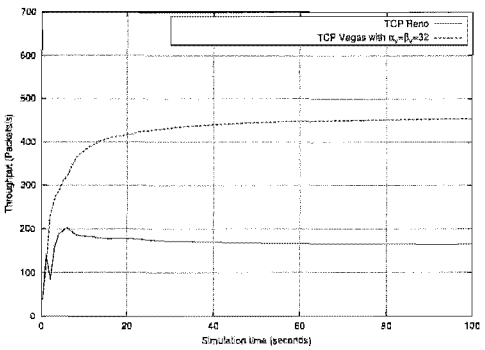

Figure 3.35 Throughputs for TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 32$) connections.

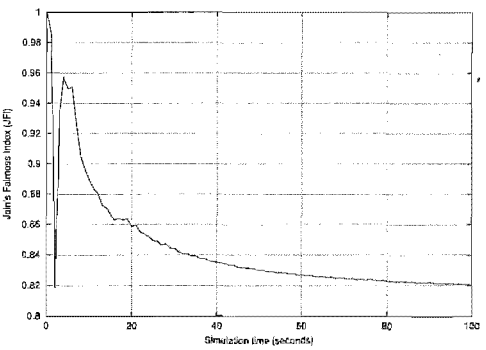

Figure 3.36 JFI for TCP Reno and TCP Vegas ($\alpha_v = \beta_v = 32$) connections.

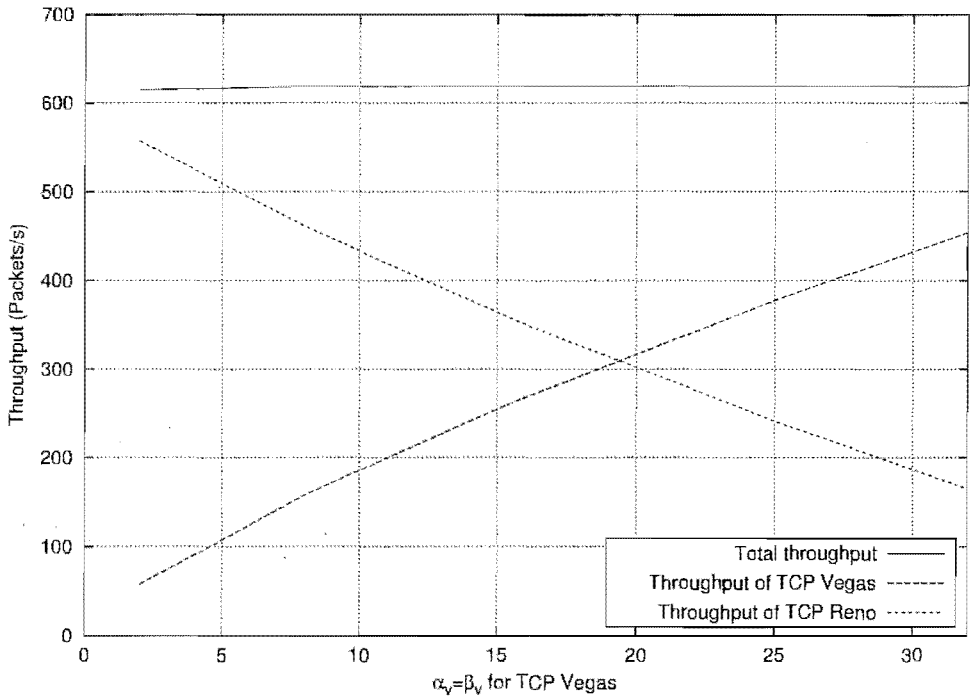

Figure 3.37 Variations in throughput of TCP Reno and TCP Vegas for different values of $\alpha_v = \beta_v$.

$\alpha = \beta$	Received Bytes		Ratio, $\frac{T_{B_R}}{T_{B_V}}$
	TCP Reno, T_{B_R}	TCP Vegas, T_{B_V}	
2	55746040	5784000	9.63
8	46169040	15784000	2.92
16	35105040	26848000	1.30
20	30251040	31702000	0.95
25	24137040	37816000	0.63
32	16492040	45461000	0.36

Table 3.1 Comparison between bytes received (at 100 s) by TCP Reno and TCP Vegas for different values of $\alpha = \beta$ parameters of TCP Vegas.

to TCP Reno connections having shorter round trip times.

- TCP Vegas connections will get nearly equal bandwidth irrespective to their round trip times.
- The sum of throughputs of all connections is greater for TCP Vegas as compared to TCP Reno case.

The simulation results in a heterogenous environment of a TCP Vegas and a TCP Reno, having same round trip times, were presented in this Subsection. They showed the incompatibility of the original TCP Vegas and TCP Reno. Later we increased the values of the α_v and β_v parameters of TCP Vegas connections and observed that for higher values, TCP Vegas can also get high values of throughput compared to TCP Reno. Thus by controlling the values of α_v and β_v appropriately, we can overcome the bandwidth loss by TCP Vegas in heterogenous environments. In the next Section we employ this characteristic of TCP Vegas in conjunction with ECN, [Ramakrishnan 1999] and [Ramakrishnan *et al.* 2001], based on Random Early Detection (RED) routers, [Floyd and Jacobson 1993], to vary the α_v and β_v parameters at sending ends to get fair distribution of bandwidth among TCP Vegas and TCP Reno connections. Where the RED algorithm will be explained in detail in the forthcoming Chapters 5 and 6.

3.6 IMPROVING TCP VEGAS / RENO COMPATIBILITY WITH RED BASED ECN

In this Section we propose a new strategy to increase or decrease the α_v and β_v parameters of TCP Vegas, to make it compatible with TCP Reno, by using ECN based packet marking by the RED algorithm. We have proved analytically in Section 3.4.1, as well as by simulations in Subsection 3.5.3, that the throughput of TCP Vegas increases with an increase in its α_v and β_v parameters.

In the case of homogenous environments of only TCP Vegas connections, after the period of initial transient, the round trip times of all connections will attain a steady state value and afterwards it will not change much, [Bonald 1998]. It occurs because TCP Vegas keeps a fixed number of packets in queue buffer i.e. equal to $\alpha_v = \beta_v$ or between α_v and β_v in case of $\alpha_v < \beta_v$, see: [Low *et al.* 2001]. However, in the case of $\alpha_v = \beta_v$ there will be small oscillations in the congestion windows of TCP Vegas connection which might lead to corresponding small variations in measured round trip time by TCP Vegas. Hence, if the round trip time of a TCP Vegas connection does not change much then the end host will infer that all other connections are of TCP Vegas type and there is no need to be more aggressive. Thus, we will not have any problem of unfairness among different connections in a homogenous environment of only TCP Vegas, [Hasegawa *et al.* 1999].

On the other hand, in heterogenous environments of both TCP Vegas and TCP Reno, the round trip time will keep on changing as TCP Reno increases and decreases its congestion

window. Thus, when the round trip time of TCP Vegas connections varies significantly during the steady state operation, it can be inferred that TCP Vegas is competing with TCP Reno in a heterogenous environment. Therefore, in order to avoid an unfair distribution of bottleneck bandwidth it is necessary to increase the aggressiveness of TCP Vegas by increasing its α_v and β_v parameters.

When there is no congestion at a bottleneck link router in such a heterogenous scenario, TCP Reno will keep on increasing the size of its congestion window (until packet drop occurs) and thus sending more and more packets into buffer of router queue. It will increase queue size, thus queuing delay, resultantly causing an increase in round trip time as measured by TCP Vegas. This increase in the round trip time of TCP Vegas will switch it to the window reduction state although there is no congestion in reality, [Mo *et al.* 1999], thus, losing bandwidth to TCP Reno.

Further, at the onset of the congestion state, due to increased traffic load, the RED router at the bottleneck link will start marking/dropping packets randomly before starting to drop them due to buffer overflow, see: [Floyd and Jacobson 1993] and [Braden *et al.* 1998]. All of the incoming packets will be dropped by the RED router, called forced drops, only at the time of severe congestion, for details see: [Floyd *et al.* 1998]. Whereas in ECN the probabilistic marking of packets is done according to the guidelines given in [Ramakrishnan 1999] and [Ramakrishnan *et al.* 2001].Thus in ECN, at the time of slight or moderate congestion packets are marked and at extreme congestion the packets are dropped.

Upon receipt of congestion experienced (CE) bit from RED router, set as high, an ECN echo bit is set high by receiver in returning ACK to data sender. These marked ACK packets act as congestion signals to all data sending sources. After receiving marked ACK packets traffic sources are required to decrease their data sending rates or congestion windows. The TCP Reno sources should reduce their congestion window by half as recommended in [Ramakrishnan 1999] and [Ramakrishnan *et al.* 2001]. We suggest that TCP Vegas sources should set α_v and β_v to their default values of 1 and 3 as in [Brakmo *et al.* 1994a] or to some other designed values (e.g. set $\alpha_v = \beta_v = 2$ in our simulations) in case of receiving a congestion signal in an ACK. Keeping the above facts in view, the pseudocode of RED based ECN algorithm for TCP Vegas, termed as NVegas, is given by the following steps:

- Compute the new round trip time, R_n, after the arrival of each ACK.
- Compare R_n with previous round trip time R_o.
- If $R_n > R_o$ then increase the values α_v and β_v by 1.

- Keep on increasing α_v and β_v until $R_n \leq R_o$.
- Keep the values of α_v and β_v fixed when $R_n \leq R_o$.
- If ACK has ECN echo bit high, set $\alpha_v = 1$ and $\beta_v = 3$ or other designed values such as $\alpha_v = \beta_v = 2$ and set the congestion window of TCP Reno connections to half the existing value.

In the next Subsection we implement the algorithm discussed above and compare its results with TCP Vegas operating with a RED router without ECN.

3.6.1 Simulations

In these simulation experiments we consider heterogenous scenarios of TCP Reno/TCP Vegas ($\alpha_v = \beta_v = 2$) and TCP Reno/TCP NVegas in the network topology shown in Figure 3.4. In each experiment both TCP's have the same round trip times. We concentrate on congestion window, received bytes, throughput and JFI as in previous simulation experiments.

First, we simulate TCP Reno/TCP Vegas using RED algorithm (without ECN) at the bottleneck link n2-n3. The resulting congestion windows, received bytes, throughputs and JFI are plotted in Figures 3.38, 3.40, 3.42 and 3.44, respectively.

Neglecting the starting transient, the congestion window of TCP Vegas varies between 10 and 22 packets with an average value of 16 packets, whereas the congestion window of TCP Reno varies between 20 to 40 packets giving an average value of 28 packets (approximately). Also the ratio of received bytes of two TCP's, $\frac{T_{B_R}}{T_{B_V}}$, is about 1.61 which is more than $\alpha_v = \beta_v = 16$ case in Table 3.1, given in the Subsection 3.5.3. At the end of this simulation the throughputs achieved by TCP Reno and TCP Vegas are 366 and and 226 packets/s, respectively, which gives $JFI = 0.94$.

Next, we simulate TCP Reno/TCP NVegas using RED algorithm (with ECN) and the resulting congestion windows, received bytes, throughputs and JFI are plotted in Figures 3.39, 3.41, 3.43 and 3.45, respectively. These graphs show that as compared to TCP Reno/TCP Vegas (employing RED without ECN) the fairness as well as throughput has improved in case of TCP Reno/TCP NVegas (employing RED with ECN).

The congestion windows of both TCP NVegas and TCP Reno varies between 8 to 40 packets, thus giving an average congestion window of 24 packets (approximately). The ratio $\frac{T_{B_R}}{T_{B_V}}$ in this case is about 1.06 which is less than previous case. Hence, the two TCP's are getting almost equal bandwidth. At the end of this simulation the throughputs achieved by TCP Reno and TCP NVegas are 301 and and 284 packets/s, respectively, which gives $JFI = 0.99$. Therefore, TCP Reno/TCP NVegas employing RED with ECN works better than TCP Reno/TCP Vegas employing Droptail (Subsection 3.5.3) and TCP Reno/TCP Vegas employing RED without ECN.

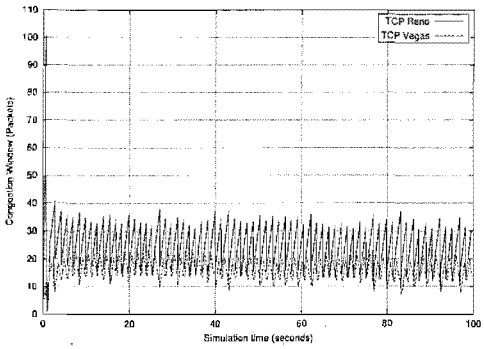

Figure 3.38 Congestion windows of TCP Reno and TCP Vegas with RED router at bottleneck link.

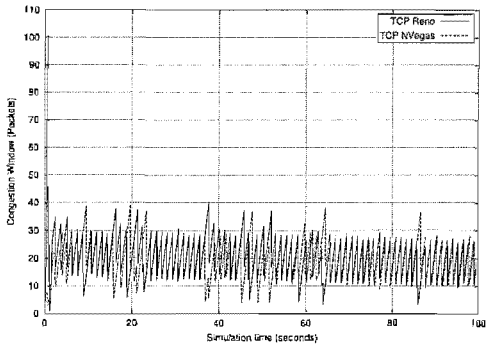

Figure 3.39 Congestion windows of TCP Reno and TCP NVegas with RED router based ECN at the bottleneck link.

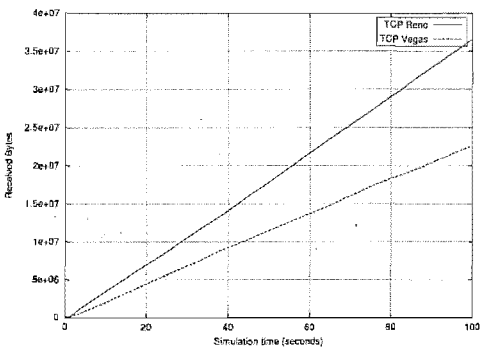

Figure 3.40 Received bytes for TCP Reno and TCP Vegas with RED router at bottleneck link.

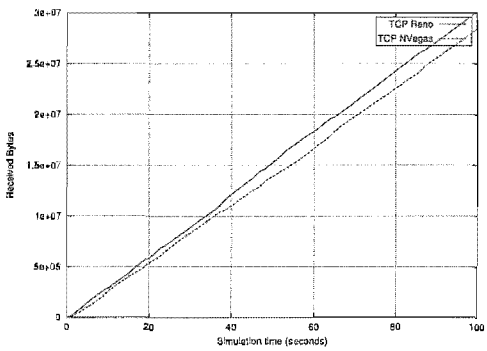

Figure 3.41 Received bytes for TCP Reno and TCP NVegas with RED router based ECN at the bottleneck link.

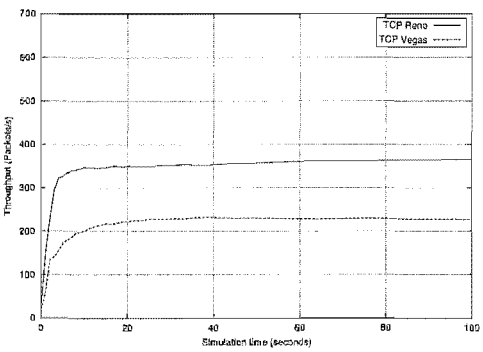

Figure 3.42 Throughput of TCP Reno and TCP Vegas with RED router at bottleneck link.

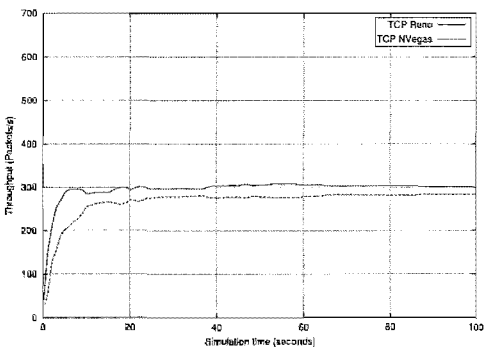

Figure 3.43 Throughput of TCP Reno and TCP NVegas with RED router based ECN at the bottleneck link.

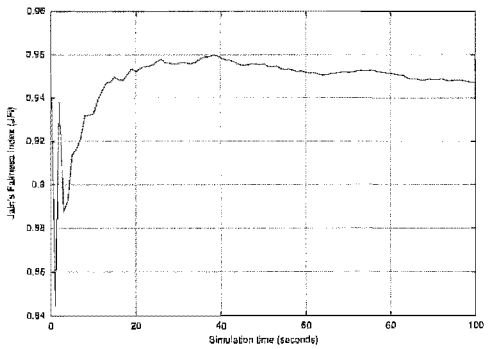

Figure 3.44 JFI with RED router.

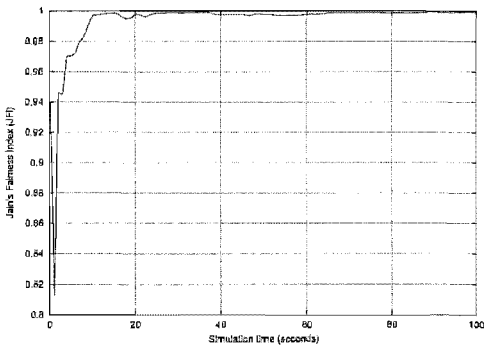

Figure 3.45 JFI with RED router based ECN.

3.7 CONCLUSIONS

In this Chapter first we recounted the evolution of proactive congestion control algorithms. We briefly described the CARD, DUAL, Tri S and TCP Vegas algorithms. Next we presented the details of three major algorithms of TCP Vegas (New Retransmission Mechanism, Modified Congestion Avoidance and Modified Slow Start). Then we outlined various drawbacks and shortcomings in TCP Vegas.

Next we concentrated on one of the main problems of TCP Vegas, i.e. incompatibility of TCP Vegas with TCP Reno, and outlined different existing approaches in the literature to solve this problem. A heteregenous case of single TCP Reno and TCP Vegas was analyzed and expressions for the congestion windows and JFI were derived under the ideal condition of the same round trip time for both TCP's. Latter these expressions were generalized to the case of larger numbers of TCP Reno and TCP Vegas connections. An open problem regarding most general case of different round trip times for all of the TCP connections was also outlined.

We performed ns-based simulation studies to investigate the behavior of TCP Reno and TCP Vegas in a homogenous environment with different round trip times. Our simulation results reconfirmed that TCP Reno has a bias against connections with longer round trip times whereas TCP Vegas does not have such a bias. Next we simulated a heterogenous scenario of TCP Reno and TCP Vegas with the same round trip times and found that TCP Vegas can be made compatible with TCP Reno by increasing its α_v and β_v parameters. We did detailed simulations and found that the throughput of TCP Vegas can be improved and even can be made greater than for TCP Reno by increasing its α_v and β_v parameters. Moreover, it was found that JFI improve markedly with an increase in α_v and β_v.

Based on this finding, we presented a new algorithm, TCP NVegas, employing RED based ECN for changing the α_v and β_v parameters of TCP Vegas. This algorithm was simulated in the last part of this Chapter and its performance was compared to TCP Vegas employing RED under similar conditions. The performance of TCP NVegas was found to be better than TCP Vegas. The JFI and throughput plots of TCP NVegas suggest that it should be experimented further in a testbed environments. Thus, TCP NVegas is a solution to make TCP Vegas compatible with TCP Reno in a heterogenous environment. However, it has all the limitations of ECN as briefly described in Section 2.8 of Chapter 2 and discussed at length in the literature, e.g. see: [Floyd 1994], [Ramakrishnan 1999] and [Ramakrishnan *et al.* 2001].

One direction of future work might be to investigate TCP NVegas further and derive limits and frequency of change of α_v and β_v with variations in measured round trip time. Also, one could tie variations in α_v and β_v with exponential weighted moving average, [Young 1984], of the measured round trip time with some suitable weighting factor.

Chapter 4

OPTIMIZATION OF TCP CONGESTION CONTROL

Due to the pivotal role being played by host TCP protocols in implementing end-to-end congestion control, optimization and proper design of their congestion control algorithms is essential for maximizing throughput while maintaining high fairness among end users. In Chapter 3 we have investigated fairness issues of end-to-end protocols currently being used in the current Internet (TCP Reno) and a newly proposed experimental protocol (TCP Vegas). The results of simulations showed that TCP Vegas is fairer than TCP Reno. We also presented enhancements to the TCP Vegas algorithm so that it can get its fair share of bandwidth in a heterogenous environment.

In this Chapter we propose a new optimized congestion control algorithm for TCP type protocols that minimizes variance of queue length. First we derive a formula for the minimum-variance window control algorithm. Then we generalize it by introducing a weighting factor for trading off window size fluctuations against queue variance, for current implementations of TCP that require window update of at most one packet per round trip time. This generalized optimal window control algorithm requires two-step-ahead estimation of available bandwidth. In order to implement this control algorithm practically, we also perform parameter estimation for bandwidth prediction using a normalized least mean square algorithm. Next we investigate the choice of the two window control parameters, target queue size and window weighting factor, on the transient fairness of the generalized control algorithm.

We also show that a similar control law with one step ahead bandwidth estimation can be derived from the scheme proposed in [Mo and Walrand 2000]. TCP Vegas can be viewed as an approximate implementation of one step ahead bandwidth estimation algorithm.

The generalized minimum variance congestion control algorithm is implemented in *ns* and the simulation results are analyzed and compared with TCP Vegas simulated under similar conditions. The rest of this Chapter is organized as follows. We review previous related work in Section 4.1. The abstracted model for steady state network scenario is given in Section 4.2.The optimal congestion control problem is formulated in Section 4.3. The optimal minimum variance window control algorithm is presented in Section 4.4. The results of simulation experiments with optimal minimum variance window control are given in Section 4.5. We generalize the minimum variance congestion control algorithm in Section 4.6 and explore the similarity between generalized algorithm and the $(p, 1)$ proportionally fair algorithm [Mo and Walrand 2000].The

stability, steady state and transient fairness issues of the generalized optimum algorithm are analyzed in Section 4.7 and 4.8. The simulations for generalized algorithm are presented in Section 4.9.Finally the conclusions are presented in Section 4.10.

4.1 REVIEW OF RELATED WORK

The cornerstone for achieving fairness in TCP and its derived protocols including TCP friendly TFRC protocols, is the additive increase and multiplicative decrease (AIMD) policy for window size variations, [Jacobson 1988]. It is proved in [Chiu and Jain 1989] that AIMD will result in classical min-max fairness among the end users. The window increase and decrease factors have been chosen as 1 and 7/8, respectively; see [Jain *et al.* 1987], and optimized as 1 and 0.5 in [Jacobson 1988]. As given in Chapter 2 the definition of proportional fairness is presented in [Kelly 1997] and the existence of a rate control scheme for achieving proportional fairness is proved in [Kelly *et al.* 1998]. This algorithm requires feedback of residual link capacity from routers and thus does not follow the end-to-end principle which does not require such feedback.

The fundamental problem of the existence of decentralized, window based and proportionally fair end-to-end protocols is addressed in [Mo and Walrand 2000]. Using a multiclass fluid model it is shown that TCP Vegas behaves like $(p, 1)$ fair window-based algorithm and its convergence is proved using Lyapunov function. Further, a generalized (p, α) proportionally fair end-to-end window based algorithm was developed for theoretical studies. These proportionally fair protocols require that an end user should know the exact value of propagation delay which is not possible; thus implementation of these algorithms is an open question.

The work in [Mo and Walrand 2000] may be viewed, in part, as a more rigorous treatment of a problem originally solved by the TCP Vegas algorithm [Brakmo *et al.* 1994b]. In essence, Mo and Walrand proved the heuristic idea behind Vegas that proportional fairness obtains (in the long run) when every connection has the same number of backlogged packets. They used (somewhat nonlinear) integral control to achieve the state of fairness whereas Vegas uses additive increase additive decrease (AIAD) which is essentially floating integral control [Astrom and Wittenmark 1990])

Recently [Low *et al.* 2001], presented results related to those of [Mo and Walrand 2000] while giving a theoretical justification for TCP Vegas. Both [Low *et al.* 2001] and [Mo and Walrand 2000] expose long-term proportional fairness, introduced by [Kelly *et al.* 1998], as being equivalent to the (static) optimization with an arbitrary concave utility function. However shortcomings of [Mo and Walrand 2000] include:

[1] Although a (constructive) existence proof was given of a window control that yields long-term proportional fairness, its optimization, e.g. minimization of time to return to the fair condition after a disturbance, was not addressed.

[2] The round trip time was neglected in the stability proof, suggesting that the system remains

stable for any value of the control gain. An upper bound on round trip time for stability has been recently given in [La 2001].

[3] Fairness over shorter time scales of the order of few round trip times was not considered. Fairness in the long run is of limited practical use because a large portion of the Internet traffic on World Wide Web (WWW) exists for shorter periods of time.

[4] The effects due to non-TCP traffic sharing the network were not considered.

4.2 ABSTRACT MODEL FOR THE STEADY STATE NETWORK SCENARIO

Let us consider a general closed multiclass fluid network as shown in Figure 4.1. It has M links where each link has its own buffer and N connections transporting small and infinitely divisible packets, [Mo and Walrand 2000]. Each connection has a (constant in the long run) window of packets in flight. It is shown in [Mo and Walrand 2000] that for a given network, the flow rates $x_i, i = 1, ..., N$ of the connections are uniquely determined by the window sizes w_i. The

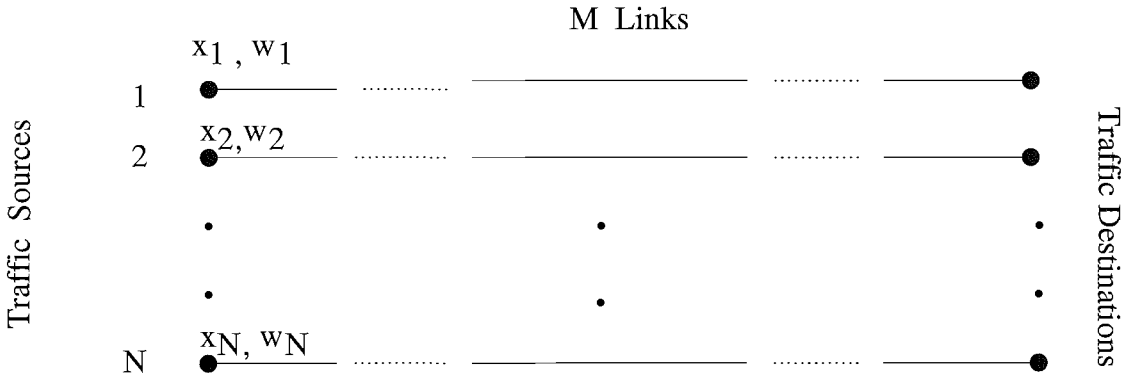

Figure 4.1 Abstract Model of a general Network.

links associated with non-empty buffers are the network bottlenecks. The queue sizes in the individual buffers are not unique, in general. However, the sum of packets belonging to any one connection in all the buffers along its path, i.e. the connection's backlog, is a constant equal to the difference between the connection's window size and its bandwidth-delay product. If the backlogs are equal (and nonzero) for all the connections, then proportional fairness is obtained [Mo and Walrand 2000]. Furthermore, if the backlogs are made unequal, then we have weighted proportional fairness, the weights being equal to the corresponding backlogs.

Focusing on a single connection we see that, in general, it would have one or more nonempty buffers along its path. Proportional fairness precludes the case of no bottlenecks because the rate of such a connection can be increased without having to decrease the rate of any other flow. For purpose of analyzing the flow of such a single connection, all the bottleneck buffers can be

consolidated into a single buffer as shown in Figure 4.2 with an occupancy equal to the sum of the occupancies of the individual bottleneck buffers.

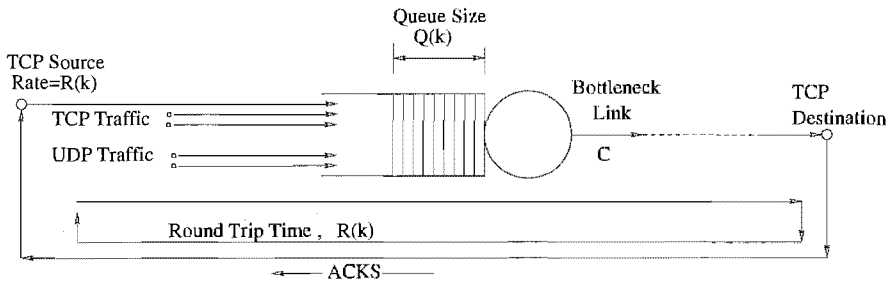

Figure 4.2 Network model for a single TCP connection.

4.2.1 Dynamic Bandwidth Model

The bandwidth available to the TCP connection at hand is the difference between the bottlenecked link's capacity and the aggregate bandwidth of the cross traffic. Taking the round trip time as the sampling interval, we may model the available bandwidth $B(k)$ at sampling instant k by a discrete-time stochastic model. For ease of exposition and simplicity, we adopt a first-order Auto-Regressive (AR) model, as in [Hassan and Sirisena 2001], but the extension to any other model can be done. Thus, for AR model we get the following equation for bandwidth at k:

$$B(k) = a_{ar} \cdot B(k-1) + (1 - a_{ar}) \cdot \bar{B} + b_{ar} \cdot w(k), \tag{4.1}$$

whereas $w(k)$ denotes the unit white noise and $\bar{B}$ denotes the mean value of $B(k)$. The parameter a_{ar} captures the time correlation of the $B(k)$. The parameter b_{ar} depends on the variance of $B(k)$ and its expression is derived in equation (A.8) and is given by:

$$Var\{B(k)\} = \frac{b_{ar}^2}{1 - a_{ar}^2}. \tag{4.2}$$

The equation (4.1) models the short range dependency and is exact for ON-OFF sources with exponentially distributed ON and OFF times. We can rewrite equation (4.1) as:

$$B(k+1) = a_{ar} \cdot B(k) + (1 - a_{ar}) \cdot \bar{B} + b_{ar} \cdot w(k+1). \tag{4.3}$$

Thus, substituting equation (4.1) into equation (4.3) we get:

$$B(k+1) = a_{ar}^2 \cdot B(k) + (1 - a_{ar}^2) \cdot \bar{B} + a_{ar} \cdot b_{ar} \cdot w(k) + b_{ar} \cdot w(k+1). \tag{4.4}$$

Equation (4.4) will be used in Section 4.4 to derive an expression for two-step-ahead bandwidth prediction.

4.2.2 Queue Buffer Dynamics

Let $Q(k)$ denote the number of packets in the buffer belonging to the connection at hand at the k-th instant. Then we have the following difference equation.

$$Q(k+1) = Q(k) + R \cdot \{\varrho(k-1) - B(k)\}, \tag{4.5}$$

where R denotes the round trip time which is also the sampling time and $\varrho(k-1)$ is the TCP source rate during the previous round trip time, thus allowing for the round trip delay in the control loop. For a queuing delay of τ_d, we have the following equation for round trip time, R:

$$R = R_b + \tau_d, \tag{4.6}$$

where R_b is the minimum R as given in equation (3.23). Now under the reasonable assumption, [Mo and Walrand 2000], that the bandwidth $B(k)$ of a connection is proportional to the number of packets $Q(k)$ it has in the buffer, we have that:

$$\tau_d = \left\{ \frac{Q(k)}{B(k)} \right\}. \tag{4.7}$$

Substituting equation (4.7) into equation (4.6) we get following expression for R:

$$R = R_b + \left\{ \frac{Q(k)}{B(k)} \right\}. \tag{4.8}$$

Further substitution of equation (4.8) into equation (4.5) gives following relation:

$$Q(k+1) = R \cdot \varrho(k-1) - R_b \cdot B(k). \tag{4.9}$$

The term $R \cdot \varrho(k-1)$ is equal to the window $W(k-1)$ of packets sent during the $(k-1)$th round trip time, thus we get following:

$$Q(k+1) = W(k-1) - R_b \cdot B(k). \tag{4.10}$$

In order to find the optimal window size $W(k)$ for the current R, we advance the time index in equation (4.10) to get the following relation:

$$Q(k+2) = W(k) - R_b \cdot B(k+1). \tag{4.11}$$

The queue buffer dynamics as depicted by equation (4.11) is employed in Subsection 4.4 to derive the optimal minimum variance congestion window control algorithm.

4.3 WINDOW BASED OPTIMAL CONGESTION CONTROL ALGORITHMS

4.3.1 Problem formulation

We aim to find the window control algorithm that achieves the following goals:

- Proportional fairness in the long run.
- Return the network to the proportionally fair condition in an optimal manner following a disturbance, e.g. a sudden change in load.

From the results of [Mo and Walrand 2000], the first goal may be achieved by designing the control to keep the queue length fixed at some reference level q_{ref}. Proportional fairness is obtained if q_{ref} is chosen to be the same for all the connections. Weighted proportional fairness is obtained by setting the q_{ref} of a connection proportional to its desired weighting.

The control algorithm for achieving the second goal depends on the chosen optimality criterion. Our aim is to minimize fluctuations from the fair steady-state condition, and hence we choose the variance of the queue length as the metric to be minimized.That is, we find the minimum variance control algorithm [Astrom and Wittenmark 1990]. One problem with minimum variance control is that it might require large increases or decreases in the window size from one round trip interval to the next. Hence, we also do an optimization with a metric that imposes a cost on the change in the window size.

4.4 OPTIMAL MINIMUM VARIANCE WINDOW CONTROL

We first find the window control algorithm that minimizes the queue variance, remembering that in order to be physically realizable it has to be based on information actually available at the instant optimal window $W(k)$ is calculated. Now the deviation of $Q(k+2)$ from its reference value q_{ref} is given by:

$$\Delta Q(k+2) = q_{ref} - W(k) - R_b \cdot B(k+1). \tag{4.12}$$

At the $(k-1)$ th sampling instant, the sender's most recent bandwidth measurement is the rate at which the previous window of packets was delivered, i.e.

$$B(k-1) = \left\{ \frac{W(k-1)}{R} \right\}. \tag{4.13}$$

Hence, we can rewrite equation (4.12) in terms of the estimate of $B(k+1)$ based on information available up to the sampling instant $(k-1)$.

$$\Delta Q(k+1) = q_{ref} - W(k) - R_b \cdot \left\{ \hat{B}(k+1|k-1) + \varepsilon_{\hat{B}}(k+1|k-1) \right\}, \tag{4.14}$$

where $\hat{B}$ denotes the estimate and $\varepsilon_{\hat{B}}$ denotes the estimation error. From equation (4.14) we get the following,

$$Var\left\{Q(k+1)\right\} = \left\{q_{ref} - W(k) + R_b \cdot \varepsilon_{\hat{B}}(k+1|k-1)\right\}^2 + Var\left\{R_b \cdot B(k+1|k-1)\right\}. \tag{4.15}$$

Clearly, the second term in equation (4.15) is independent of $W(k)$ and the first term is minimized by making it vanish, i.e. by choosing it to be

$$W(k) = q_{ref} + R_b \cdot \hat{B}(k+1|k-1). \tag{4.16}$$

The above equation (4.16) is the optimal window control algorithm and it can be given a simple intuitive interpretation: the optimal window size is the sum of the desired number of packets in the buffer plus the product of the round trip delay and the two-steps-ahead prediction of the available bandwidth. If rate control is employed, then the optimal sending rate is

$$\varrho(k) = \left\{ \frac{W(k)}{R} \right\}. \tag{4.17}$$

For the simple AR model given in equation (4.1), the required two step ahead prediction can be obtained as by the following equation:

$$\hat{B}(k+1|k-1) = a_{ar}^2 \cdot B(k-1) + (1 - a_{ar}^2) \cdot \bar{B}(k-1). \tag{4.18}$$

If another stochastic model is known to be appropriate, e.g. Fractional Brownian Motion or Fractional ARIMA models, [Ostring and Sirisena 2001], when the cross traffic is self-similar, then the corresponding two step ahead predictor has to be substituted in equation (4.16).

4.4.1 Implementation of Optimal Window Control Algorithm

In order to implement the optimal window control as given by the equations (4.16) and (4.18), we need to determine the values of R_b, a_{ar} and $\bar{B}$. The value of R_b can be estimated by the shortest round trip time measurement as in TCP Vegas, [Brakmo *et al.* 1994b], and $\bar{B}$ can be obtained by applying the following EWMA process:

$$\bar{B}(k) = \gamma \cdot \bar{B}(k-1) + (1-\gamma) \cdot B(k). \tag{4.19}$$

We used γ=0.95 in equation (4.19) for our simulation experiments. The value of a_{ar} can be estimated by applying the normalized Least Mean Square (LMS) algorithm as given below, adapted from [Haykin 1998] and [Astrom and Wittenmark 1995]. The value of the estimate, $\hat{a}_{ar}$, of the parameter, a_{ar}, at sampling instant k is given by:

$$\hat{a}_{ar}(k) = \hat{a}_{ar}(k-1) + \left[\frac{\delta}{Var\{B(k)\}}\right] \cdot \left[B(k) - \left\{a_{ar}(k-1) \cdot B(k-1) + \{1 - a_{ar}(k-1)\} \cdot \bar{B}(k-1)\right\}\right], \tag{4.20}$$

where $Var\{B(k)\}$ is estimated by using the following EWMA process:

$$Var\{B(k)\} = 0.95 \cdot Var\{B(k-1)\} + 0.05 \cdot \left\{B(k) - \bar{B}(k)\right\}^2. \tag{4.21}$$

In the next Section we present the results of simulation experiments after implementing the optimal minimum variance control algorithm as derived in this Section.

4.5 SIMULATION OF OPTIMAL MINIMUM VARIANCE WINDOW CONTROL ALGORITHM

The model and topology of simulated network are shown in Figures 4.2 and 4.4, respectively. We use 5 TCP sources which are implementing the optimal minimum variance window control algorithm, described in Section 4.4 and equation (4.16), and 10 UDP sources as the background traffic. All of the traffic sources start sending data at the same time i.e. 0 s. All of the access links have a capacity of 10 Mbps with propagation delays varying gradually from 25 ms to 125 ms. The bottleneck link has capacity of 16 Mbps and a 25 ms propagation delay as shown in Figure 4.4. Droptail routers are used on all links. The buffer of the bottleneck link router is large enough, allowing us to study the queue dynamics without packet losses due to buffer overflow.

The five TCP connections have different base round trip times, R_b's, as given in Table 4.1 and their congestion windows are updated after every round trip time which consists of the propagation delay plus the queuing delay at the congested router as given by equation (4.6). We use traffic generated by File Transfer Protocol (FTP) on 5 TCP connections and Exponential/Pareto ON/OF traffic on 10 UDP connections. An ON-OFF source transmits traffic at a constant rate during the ON period and no traffic during the OFF period. For exponential background traffic, the ON and OFF periods are exponentially distributed and for self-similar background traffic, the lengths of ON and OFF periods are drawn from a Pareto distribution as given in [Leland *et al.* 1994].

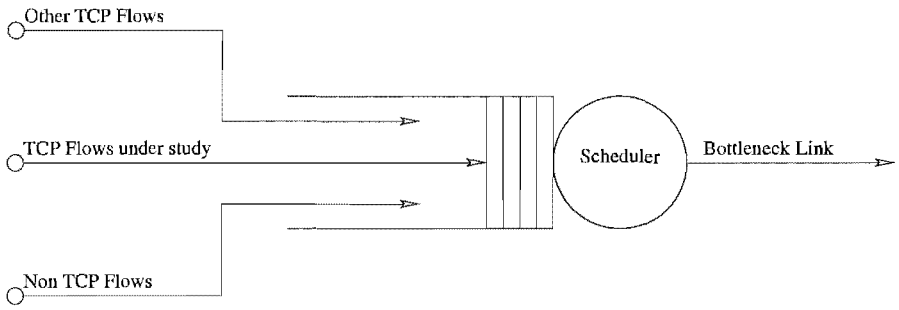

Figure 4.3 Network model for discrete event simulations of Optimal minimum variance window control algorithm.

4.5.1 Experiment 1

In this experiment all of the 5 TCP sources have same value of q_{ref}=16 packets and different values of base round trip time R_b. The quantitative results of this experiments are summarized in Table 4.1 and instantaneous queue variations are shown in Figure 4.5. It is found that the mean value of instantaneous queue is 109 packets, with a variance of 1702, which is close to the sum of the q_{ref}'s of the five TCP flows namely 80 packets. The mean sending rates or mean bandwidth of TCP sources are also close to each other despite the difference in their round trip times due to equal setting of q_{ref}. The sum of sending rates of all TCP sources is 1503 packets/s which is close to difference in bottleneck link capacity and aggregate mean of ON/OFF sources which is 1800 packets/s. Thus, the simulation results are close to the results predicted by theory of optimal minimum variance window control algorithm.

Source	R_b (ms)	R (ms)	q_{ref} (Packets)	Mean W (Packets)	Mean Bandwidth (Packets/s)
TCP 1	100	154.41	16	48	311
TCP 2	150	204.41	16	63	308
TCP 3	200	254.41	16	77	302
TCP 4	250	304.41	16	90	297
TCP 5	300	354.41	16	100	285

Table 4.1 Results for experiment 1 with Optimal Minimum Variance window control algorithm.

4.5.2 Experiment 2

In this experiment we use different values of q_{ref} for the five TCP connections as shown in Table 4.2. The mean of the instantaneous queue size is 76 packets whereas the sum of q_{ref}'s of all the TCP sources is 62 packets. These two values are quite close to each other. The variations in instantaneous queue size at the bottleneck router are shown in Figure 4.6. We observed that the mean sending rates of TCP's are increasing with increase in value of q_{ref}. The proportion of increase of mean bandwidth with q_{ref} can be written as $2:4:8:16:32::136:183:264:429:675$ or in simplified form $1:2:4:8:16::1:1.34:1.94:3.15:4.96$ which show

that mean bandwidth increases almost in proportion to increase in q_{ref}. The sum of allocated bandwidths is 1687 packets/s which is close to the available bandwidth.

Source	R_b (ms)	R (ms)	q_{ref} (Packets)	Mean W (Packets)	Mean Bandwidth (Packets/s)
TCP 1	100	138.29	2	18	136
TCP 2	150	188.29	4	34	183
TCP 3	200	238.29	8	63	264
TCP 4	250	288.29	16	123	429
TCP 5	300	338.29	32	222	675

Table 4.2 Results for experiment 2 with Optimal Minimum Variance window control algorithm.

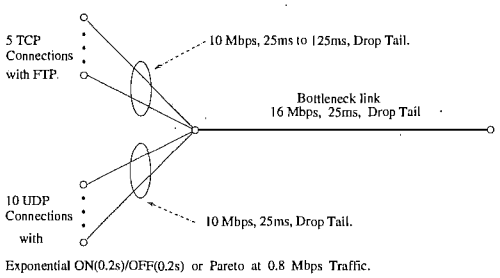

Figure 4.4 Network topology for simulations of optimal minimum variance window control algorithm.

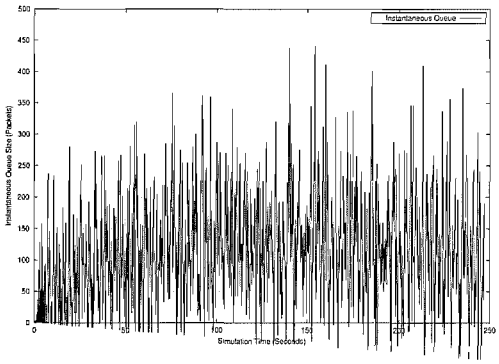

Figure 4.5 Instantaneous queue variations of bottleneck link router for Experiment 1.

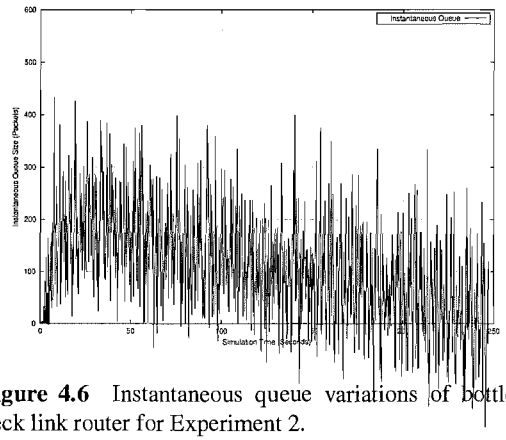

Figure 4.6 Instantaneous queue variations of bottleneck link router for Experiment 2.

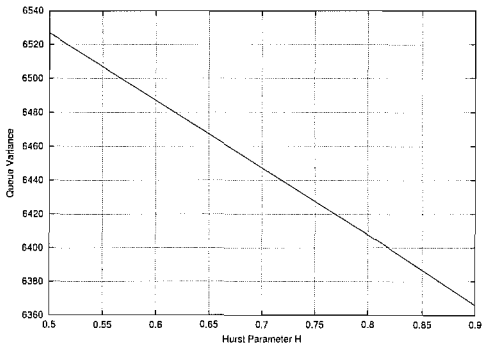

Figure 4.7 Queue variance for self similar background traffic as a function of the Hurst parameter.

4.5.3 Experiment 3

In this experiment the conditions are the same as in Experiment 1, described in Section 4.5.1, except that the ON/OFF traffic sources are Pareto distributed with varying Hurst parameter, H. We use the Pareto traffic generator available in ns and varied the shape parameter α of the Pareto distribution to get the traffic of desired self-similarity. We vary α from 1.2 to 2 which results in

variation of H in the range of 0.5 to 0.9. Where, for a given value of α, the corresponding value of H is computed by using relation $\alpha = 3 - 2 \cdot H$, [Leland *et al.* 1994].

It is observed that the average queue variance decreases with an increase in Hurst parameter as shown in Figure 4.7. This result agrees with that obtained previously in [Hassan and Sirisena 2001] for centralized ABR rate control. An explanation for this behavior is that long range dependence makes the traffic more predictable and hence the predicted bandwidth in window equation (4.16) is more accurate.

4.6 GENERALIZED OPTIMAL MINIMUM VARIANCE WINDOW CONTROL ALGORITHM

The window control algorithm given in equation (4.16) could conceivably require changes in the window size of considerably more than one packet from one round trip interval to the next and this could be a problem in practical implementations. In contrast, during the congestion avoidance phase of TCP Reno, the window may increase by a maximum of one packet during one round trip time interval. Similarly in TCP Vegas, both window size increases and decreases are limited to one packet per round trip time.

Hence let us consider a metric that weights the change in the window size from one round trip interval to the next. Specifically, we minimize the factor $\theta \cdot Var\{Q(k+1)\} + (1-\theta) \cdot \{W(k) - W(k-1)\}^2$ to obtain following optimal window control algorithm:

$$W(k) = (1-\theta) \cdot W(k-1) + \theta \cdot \left\{ q_{ref} + R_b \cdot \hat{B}(k+1|k-1) \right\}. \tag{4.22}$$

This is a weighted average of the current window size and that for minimum variance control and may be viewed as a generalization given by equation (4.16). Note that the optimal minimum variance control algorithm, as given in equation (4.16), can be obtained by substituting $\theta = 1$ in equation (4.22).

In the next Subsection we compare the generalized optimal window control algorithm with the $(p, 1)$ proportionally fair algorithm proposed in [Mo and Walrand 2000].

4.6.1 Comparison of Generalized Optimal Window Control Algorithm with (p, 1) Proportionally Fair Algorithm

Following the notation of this Chapter, the algorithm given in [Mo and Walrand 2000] can be written as:

$$\frac{dW}{dt} = \left(\frac{-\kappa \cdot R_b \cdot S}{R \cdot W} \right), \tag{4.23}$$

where S in equation (4.23) can be written as:

$$S = W - B \cdot R_b - q_{ref}. \tag{4.24}$$

Discretising equation (4.23) with the sampling interval equal to round trip time, R, we get the equation for congestion window:

$$W(k) = W(k-1) - \kappa \cdot R_b \cdot \left\{ \frac{S(k-1)}{W} \right\}. \tag{4.25}$$

Now let us define a modified gain factor as follows:

$$\theta = \kappa \cdot \left(\frac{R_b}{W} \right), \tag{4.26}$$

which in the long run tends to a constant as the window size W stabilizes. Substituting equation (4.24) into equation (4.25) and solving for $W(k)$ gives the following discretized form of $(p, 1)$ proportionally fair window control algorithm.

$$W(k) = (1-\theta) \cdot W(k-1) + \theta \cdot \{q_{ref} + B(k-1) \cdot R_b\}. \tag{4.27}$$

Comparing equation (4.27) with the optimal window control algorithm in equation (4.22), we observe that essential difference between two is that equation (4.27) contains the most recently observed bandwidth $B(k-1)$, whereas equation (4.22) uses the two-step prediction of $B(k+1)$. Finally we note that proof of stability of the $(p, 1)$ proportionally fair algorithm for any positive gain κ, however large, given in [Mo and Walrand 2000] was based on neglecting the round trip delay in the feedback loop. Therefore in next Section we address the stability issue in more details.

4.7 STABILITY OF GENERALIZED OPTIMAL WINDOW CONTROL ALGORITHM

Substituting the bandwidth equation (4.13) into the generalized optimal minimum variance window equation (4.27) we get:

$$W(k) = (1-\theta) \cdot W(k-1) + \theta \cdot \left\{ q_{ref} + \frac{W(k-1)}{R} \cdot R_b \right\}. \tag{4.28}$$

Taking z-transforms, [Hostetter 1988], on both sides of equation (4.28), we get:

$$W(z) = \theta \cdot q_{ref} \cdot \left[\frac{z^2}{z-1} \cdot \frac{1}{z - \left\{1 - \theta \cdot \left(1 - \frac{R_b}{R}\right)\right\}} \right]. \tag{4.29}$$

Thus, the z-transform of the generalized window in equation (4.29) has two poles, one at $z = 1$ and the other given by:

$$z = (1 - \theta) + \theta \cdot \left(\frac{R_b}{R}\right). \tag{4.30}$$

To study the dynamics of system we substitute equations (3.23) and (3.24) into equation (4.30) to get the following expression:

$$z = (1 - \theta) + \theta \cdot \left(\frac{\mu \cdot \tau_p + 1}{\mu \cdot \tau_p + q(t) + 1}\right). \tag{4.31}$$

We can rewrite equation (4.31) as follows:

$$z = (1 - \theta) + \theta \cdot \left(\frac{1}{1 + \frac{q(t)}{\mu \cdot \tau_p + 1}}\right). \tag{4.32}$$

The pole location, as given by equation (4.32), has two parts, one fixed which is $1 - \theta$ and other variable which depends upon current value of queue size. Further simplifying equation (4.32) we can have:

$$z = 1 - \theta \cdot \left(\frac{1}{1 + \frac{1 + \mu \cdot \tau_p}{q(t)}}\right). \tag{4.33}$$

For $\theta = 0$ the pole will be at $z = 1$ and for higher values of θ, its location depends upon the instantaneous queue size $q(t)$. For a fixed value of $\theta > 0$, the number of connections N, and the queue maximum buffer size B_q, the pole will move away from $z = 1$ along the horizontal axis as $q(t)$ changes from 0 to its maximum size of B_q/N packets. Where B_q/N is a fair share of queue buffer space for each competing connection. At a certain value of queue size, the pole will move out of unit circle causing instability of algorithm. We can show that for a fixed value of θ, the condition of stability is:

$$\frac{B_q}{N} \cdot (\theta - 2) \geq 2 \cdot (\mu \cdot \tau_p + 1). \tag{4.34}$$

Thus, system will become unstable if queue buffer size B_q is small i.e.

$$B_q < 2N \cdot \left(\frac{\mu \cdot \tau_p + 1}{\theta - 2}\right). \tag{4.35}$$

Hence, for stable operation with fixed value of θ we should have buffer size B_q such that:

$$B_q > 2N \cdot \left(\frac{\mu \cdot \tau_p + 1}{\theta - 2} \right). \tag{4.36}$$

Alternatively, for a fixed value of buffer size B_q the expression for θ for stable operation is given by:

$$\theta \geq 2 \left\{ 1 + \frac{N}{B_q} (1 + \mu \cdot \tau_p) \right\}. \tag{4.37}$$

The above equation serves as a lower limit for stable operation of algorithm given in equation (4.22). Where as upper limit is determined by other system parameters as in proceeding Section.

4.8 FAIRNESS OF GENERALIZED OPTIMAL WINDOW CONTROL ALGORITHM

In the following two subsections we investigate the steady state and transient fairness of the generalized optimal minimum variance window control algorithm.

4.8.1 Steady State Fairness

The window dynamics of Generalized Optimal Control algorithm in equation (4.22), are determined by equations (4.6), (4.7) and (4.13). In order to calculate the time constant of the dynamics, the stochastic term $\hat{B}(k+1|k-1)$ in equation (4.22) is replaced by its expected value given in equation (4.13) to get the following expression:

$$W(k) = W(k-1) \cdot \left\{ (1-\theta) + \theta \cdot \left(\frac{R_b}{R} \right) \right\} + \theta \cdot q_{ref}. \tag{4.38}$$

Above equation (4.38) shows that, as $k \rightarrow \infty$, the expected window size becomes:

$$W(k) = q_{ref} \cdot \left(\frac{R}{R - R_b} \right). \tag{4.39}$$

Now, the denominator $(R - R_b)$ is merely the queuing delay. If there are N connections, each having the same q_{ref} setting, the total number of packets in the queue would on average be $N \cdot q_{ref}$, thus queuing delay, τ_d, for the bottleneck link having capacity C is given by following relation:

$$\tau_d = R - R_b = N \cdot \left(\frac{q_{ref}}{C} \right). \tag{4.40}$$

Substituting equation (4.40) into equation (4.39), we get the following expression:

$$W(k) = R \cdot \left(\frac{C}{N} \right), \tag{4.41}$$

which by equation (4.13) confirms that, in the long run, each connection will obtain the same fair share C/N of the available bandwidth, irrespective of the individual round trip times.

4.8.2 Transient Fairness

Let us now consider the issue of transient fairness, that is how quickly the network converges to the long-term fair condition (4.41), or returns to it after a change in traffic load. This is determined by the pole of the discrete time system (4.38) as given in equation (4.30), i.e.

$$z = (1 - \theta) + \theta \cdot \left(\frac{R_b}{R}\right). \tag{4.42}$$

Next we derive the time constant τ of system by using the equivalence relation, $z = e^{\frac{-R}{\tau}}$. We have the following equation for the time constant τ:

$$\tau = -\frac{R}{\ln z}. \tag{4.43}$$

Substituting equations (4.40) and (4.42) into equation (4.43), we get

$$\tau = \frac{-R}{\ln\left(1 - \frac{\theta \cdot N \cdot q_{ref}}{C \cdot R}\right)}. \tag{4.44}$$

We can write equation (4.44) in terms of congestion window size by substituting equation (4.13), i.e.

$$\tau = \frac{-R}{\ln\left(1 - \frac{\theta \cdot q_{ref}}{W}\right)}. \tag{4.45}$$

Solving equation (4.45) for $\theta \cdot q_{ref}$ we get:

$$\theta \cdot q_{ref} = W \cdot \left(1 - e^{\frac{-R}{\tau}}\right). \tag{4.46}$$

The equation (4.46) is plotted in Figure 4.8. It can be used to tune the TCP based on generalized minimum variance algorithm by selection of τ/R and finding corresponding value of $\theta \cdot q_{ref}/W$. Using the approximation $\ln(1-x) = -x$, for small values of x, the expression for τ in equation (4.44) can be further simplified as:

$$\tau = \frac{C \cdot R^2}{N \cdot \theta \cdot q_{ref}}. \tag{4.47}$$

We can also rewrite equation (4.47) in window form as follows:

$$\tau = \frac{W \cdot R}{\theta \cdot q_{ref}}. \tag{4.48}$$

Thus, the theoretical time constant τ for generalized minimum variance window control algorithm, varies inversely with θ and q_{ref} for fixed values of C, N and R. Also the time constant will remain unchanged if θ and q_{ref} are varied such that their product remains unchanged. We can rewrite equation (4.47) as

$$\tau = \left(\frac{R \cdot C}{N \cdot \theta \cdot q_{ref}} \right) \cdot R. \tag{4.49}$$

We further investigate the characteristics of the system time constant τ in Subsection 4.9.2.

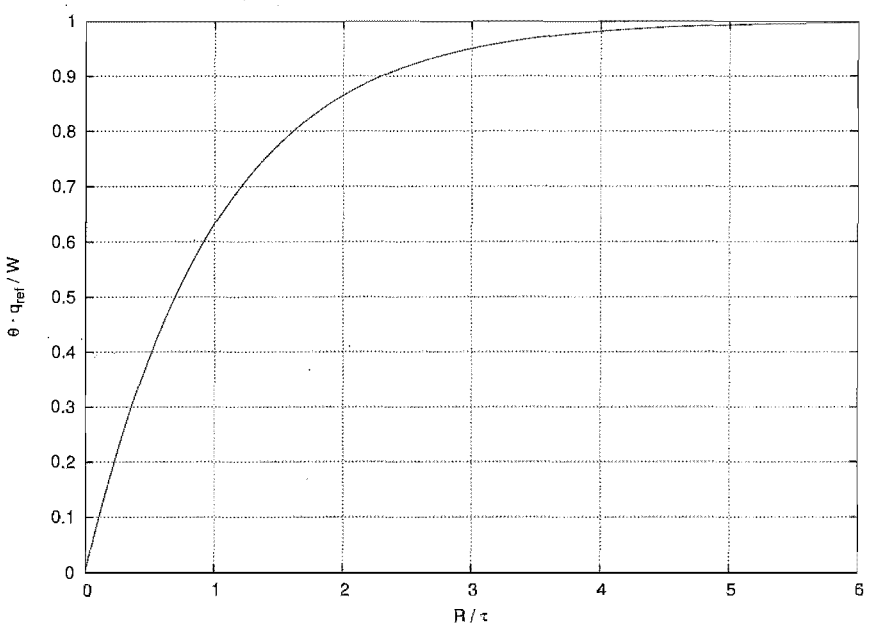

Figure 4.8 Variations of $\theta \cdot q_{ref}/W$ vs R/τ.

4.8.3 Metrics for Measurement of Fairness

For N number of connections sharing a bottleneck link and having bandwidths $B_1, B_2, ..., B_N$ Jain's Fairness Index (JFI) is defined in equation (2.22) as:

$$JFI = \frac{(B_1 + B_2 + \cdots + B_N)^2}{N \cdot (B_1^2 + B_2^2 + \cdots + B_N^2)}. \tag{4.50}$$

For bandwidths allocated in a short time interval T, we can modify equation (4.50) to:

$$JFI(T) = \frac{\{B_1(T) + B_2(T) + \cdots + B_N(T)\}^2}{N \cdot \{B_1(T)^2 + B_2(T)^2 + \cdots + B_N(T)^2\}}, \tag{4.51}$$

where in equation (4.51), $B_1(T), B_2(T), \cdots, B_N(T)$ are the bandwidths attained by the N flows in the time interval T. The fairness index as defined in equation (4.50) is independent of population size, scale, bounded and is continuous. However, it does not explicitly describe the

difference between the maximum and minimum bandwidth allocations among the competing flows.

Thus, in order to determine the transient fairness over a time interval T, we define an alternate index of fairness, called Short Term Fairness Index ($STFI$), which is similar to max-min ratio defined in [Jain *et al.* 1984], as:

$$STFI(T) = \frac{\min\limits_{1 \le i \le N} B_i(T)}{\max\limits_{1 \le j \le N} B_j(T)}, \quad i, j \in [1, N], \tag{4.52}$$

where in equation (4.52), B_i is minimum allocated bandwidth to i-th user and B_j is maximum allocated bandwidth to j-th user among N number of total users. Although $STFI$ as defined in equation (4.52) does not have continuity as in case of JFI in equation (4.51), but it is a straight forward indicator of fairness in bandwidth allocation among competing flows. In our simulations studies of generalized optimal minimum variance window control algorithm in the next Section we vary the value of time interval T from 0.25 s to 1 s to determine the average values of JFI and $STFI$ for different settings of θ and q_{ref} parameters.

4.9 SIMULATION OF GENERALIZED OPTIMAL MINIMUM VARIANCE WINDOW ALGORITHM

In this Section, we present the network topology used in the simulation experiments, study the characteristics of the system time constant τ and finally explore the fairness characteristics of generalized optimum minimum variance window control algorithm implemented in TCP end hosts.

4.9.1 Simulation Topology

The simulation scenario shown in Figure 4.9 consists of 5 TCP sources which are implementing the generalized minimum variance window control algorithm and 10 UDP sources. All traffic sources are sharing a common bottleneck link of capacity of 16 Mbps and 25 ms propagation delay through a Droptail router. All other links in the simulation topology are also using Droptail routers and have capacity of 10 Mbps, with propagation delays varying from 25 ms to 125 ms for TCP sources and up to 250 ms for UDP sources.

We use FTP traffic sources over the TCP sources, which start sending data gradually from 0 s to 140 s one after the other with an interval of 10 s. The base round trip time of the these five TCP's also vary gradually from 100 ms to 300 ms with an interval of 50 ms. On all UDP we run exponential traffic with sending rate of 0.8 Mbps having ON time of 0.2 s and OFF time of 0.2 s. There is also a CBR traffic source which start sending data at time of 100 s and keep

on pumping packets into the network until the end of the simulation. Its sending rate is 4 Mbps and its link speed is 10 Mbps with a propagation delay of 25 ms.

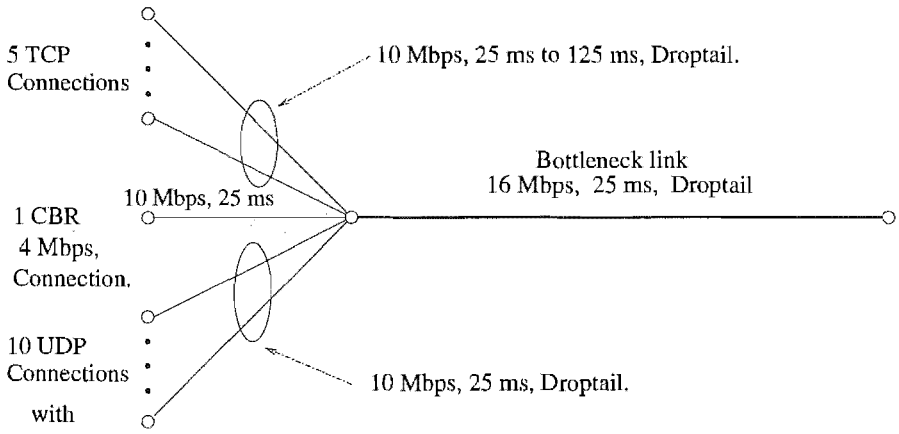

Figure 4.9 Network topology for simulations of generalized optimal minimum variance window control algorithm.

4.9.2 Time Constant

In this Subsection we investigate the behavior of time constant equation (4.44) by changing values of round trip time, R, and parameter θ while maintaining N, C and q_{ref} as constant.

For the simulation topology of Figure 4.9 we have number of TCP connections $N = 5$, bottleneck link capacity $C = 16$ Mbps or 2000 packets/s (for packet size of 1000 bytes). The reference value of queue size, q_{ref}, is chosen to be 16 packets, hence we can write equation (4.44) as:

$$\tau = \frac{-R}{\ln\left\{1 - 0.04\left(\frac{\theta}{R}\right)\right\}}. \tag{4.53}$$

We plot equation (4.53) in Figure 4.10, which shows that for a given value of round trip time the time constant τ is almost constant for different values of θ. Also, for a fixed value of θ, the system time constant increases rapidly with an increase in round trip time. Thus, on shorter time scales the TCP connections having larger round trip times will take more time to converge then TCP connections having smaller round trip times. Hence this algorithm has a bias in transient fairness against connections with larger round trip times, despite being fair in the steady state as proved in Section 4.8.1.

4.9.3 Fairness characteristics

In order to investigate the fairness characteristics of generalized optimal minimum variance algorithm presented in Section 4.6, we perform ns based simulations and plot JFI and $STFI$, as defined in Section 4.8.3. We simulate different combinations of θ and q_{ref} with bandwidth sampling intervals of 0.25 s and 1 s. The log scale plots of JFI and $STFI$ for $T = 0.25$ s

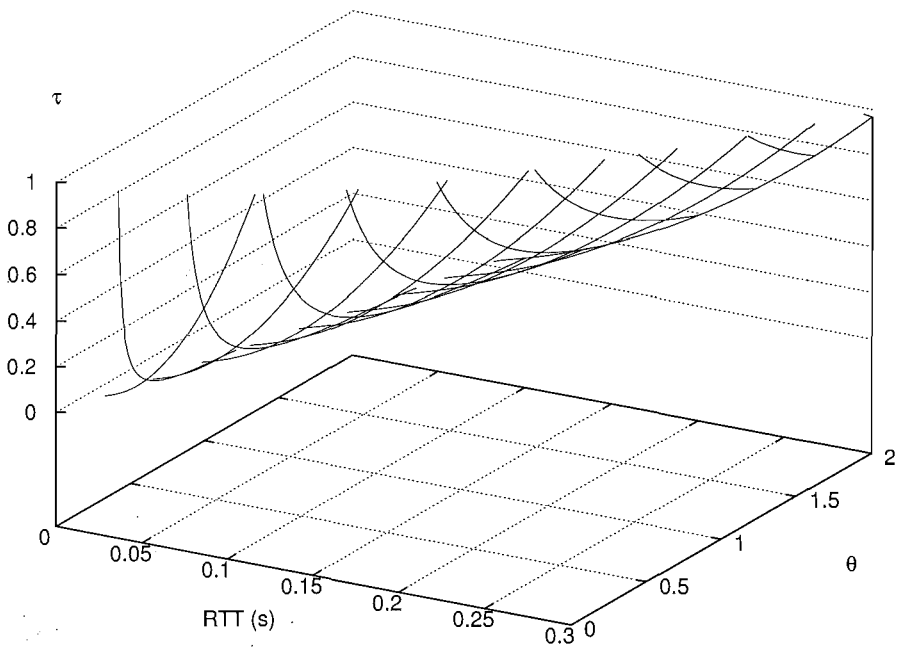

Figure 4.10 Time constant, τ, variations with round trip time, R, and θ on linear scale.

are shown in Figures 4.11 and 4.12 and for $T = 1$ s in Figures 4.13 and 4.14, respectively. For the shorter sampling time intervals, i.e. 0.25 s, JFI has higher value (closer to 1) when θ is small but for larger values of θ the JFI decreases slightly to values between 0.9 and 0.95. The value of $STFI$ is also close to 0.9 until $\theta = 1$ when it decreases slightly. It shows that generalized optimal minimum variance algorithm has good fairness at shorter time scales. To determine the fairness performance at larger bandwidth sampling interval we select $T = 1$ s and plot JFI and $STFI$ as in previous case. In this case JFI is very close to 1 for $\theta < 1$ and drops slightly for $\theta > 0.5$ for a given value of q_{ref}. The closeness of JFI to 1 indicates that all TCP flows are getting almost the same bandwidth and thus the generalized optimal minimum variance algorithm is fair for TCP connections having different round trip times as proved in Subsection 4.8.1. The reason for decreasing fairness for higher values of q_{ref} can be explained by an increase in queuing delay, given in equation (4.40), and thus round trip time R leading to lower fairness characteristics with droptail.

In the expression of generalized minimum variance control algorithm given in equation (4.22), the term $(1 - \theta) \cdot W(k - 1)$ will become negative for values of $\theta > 1$, thus decreasing the value of $W(k)$ which may cause an undesired increase in resulting congestion window thus leading to lower fairness. Therefore, in Figures 4.13 and 4.14 the values of JFI and $STFI$ drops for higher values of q_{ref} and θ.

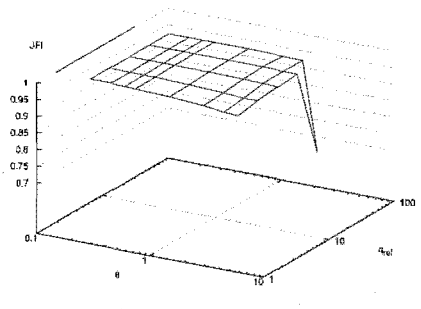

Figure 4.11 Log plot of JFI with sampling time $T = 0.25s$.

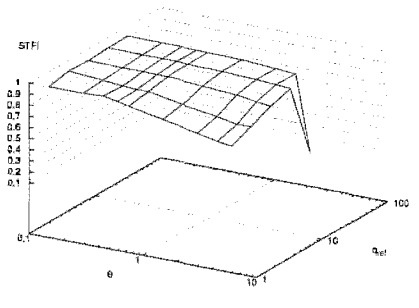

Figure 4.12 Log plot of STFI with sampling time $T = 0.25s$

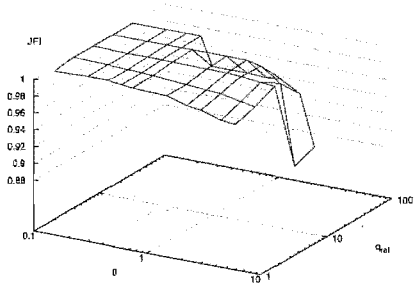

Figure 4.13 Log plot of JFI with sampling time $T = 1s$

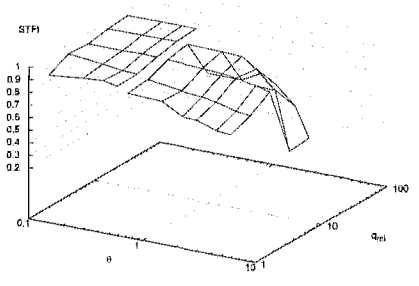

Figure 4.14 Log plot of STFI with sampling time $T = 1s$

4.9.4 Congestion window characteristics

For visualizing the effects of change in θ and q_{ref} on performance of generalized optimal minimum variance control protocol, we plot congestion window graphs for different combinations of (θ, q_{ref}), where $\theta \in \{0.1, 0.25, 0.5, 1.0\}$ and $q_{ref} \in \{2, 4, 8, 16\}$. The congestion window's for $\{(\theta, q_{ref}) = (0.1, 2), (0.25, 2), (0.50, 2), (1.0, 2)\}$ are shown in Figures 4.17, 4.18, 4.19, 4.20, for $\{(\theta, q_{ref}) = (0.1, 4), (0.25, 4), (0.50, 4), (1.0, 4)$ are shown in Figures 4.21, 4.22, 4.23, 4.24, for $\{(\theta, q_{ref}) = (0.1, 8), (0.25, 8), (0.50, 8), (1.0, 8)\}$ are shown in Figures 4.25, 4.26, 4.27, 4.28 and for $\{(\theta, q_{ref}) = (0.1, 16), (0.25, 16), (0.50, 16), (1.0, 16)\}$ are shown in Figures 4.29, 4.30, 4.31, 4.32, respectively.

It is observed that for lower values of q_{ref} the congestion windows of different TCP connections are not distinct in region from 0 s to 100 s. The most probable reason is that at the lower values of q_{ref}, the number of packets of each TCP connection does not have enough space in buffer of queue, thus cannot rise to its initial transient value. Where as for higher values of q_{ref}, the congestion window plots are very well separated in the said period. For a given value of q_{ref} the time to reach steady state value (or time constant τ) decreases with increase of θ as suggested by equation (4.49). It can be seen that for $q_{ref} = 16$ there is a sudden spike and congestion window of all TCP's drop to very low value. This phenomenon can be explained by fact that CBR source will pump packets at higher rate causing TCP's to decrease their congestion

windows as queuing delay will be increased. It can be treated like a global synchronization of all TCP sources. Thus, all TCP's will back off at same time and restart their normal operation after short time.

4.9.5 Congestion window variance characteristics

Also we plot the variance of congestion window of all 5 TCP connections in Figures 4.49, 4.50, 4.51, 4.52 and 4.53. The variance in congestion window of any particular TCP for a given value of q_{ref} increases slightly with an increase in θ up to value of 1. After $\theta > 1$ the variance increases abruptly due to $(1-\theta)\cdot W(k-1)$ term in equation (4.22) which also agrees with the stability condition defined in equation (4.37). It can be observed from the variance plots that for lower values of q_{ref} the variations in θ has less effect on variance which is in accordance with theory developed in Section 4.6.

We present the quantitative results for arithmetic mean and standard deviation of TCP 1, TCP 2, TCP 3, TCP 4 and TCP 5 in Tables 4.3, 4.4, 4.5, 4.6 and 4.7, respectively. These results show that for a fixed value of q_{ref} the arithmetic mean of congestion window does not change in large amount with increasing values of θ which is in accordance to theory developed in Section 4.6. However for a fixed value of q_{ref} the variance and standard deviation of congestion window change significantly from lower to higher values of θ. Thus, showing that the fluctuations in congestion window increase with θ. The reason for increased variance or standard deviation of congestion window is that the second term (involving bandwidth prediction) in the algorithm given in equation (4.22) increases with an increase in θ, thereby causing increase in window growth leading to higher variance. Also the algorithm become unstable beyond the stability limits defined in equations (4.36) and (4.37). Thus, for higher values of q_{ref} and θ the congestion window become unstable hence we do not compute the values of mean and standard deviation at these values.

q_{ref}	Arithmetic Mean/Standard Deviation of congestion window of TCP 1					
	$\theta = 0.10$	$\theta = 0.20$	$\theta = 0.50$	$\theta = 1.0$	$\theta = 2.0$	$\theta = 4.0$
2	30.45/1.84	32.22/2.16	35.15/2.62	36.98/2.83	41.14/4.03	44.60/5.37
4	33.81/2.13	35.53/2.52	38.27/3.42	41.20/4.43	44.51/6.07	47.83/7.81
8	38.73/2.13	39.94/2.71	41.93/4.06	44.01/5.48	46.97/8.18	50.64/11.32
16	46.22/1.90	46.99/2.34	48.46/3.56	49.28/4.37	50.71/5.87	51.95/8.44
32	61.76/1.38	62.45/1.81	63.18/2.52	64.00/2.95	64.73/4.03	-

Table 4.3 Arithmetic mean and standard deviation of congestion window of TCP connection with base round trip time of 100 ms and implementing Generalized optimum minimum variance algorithm in Figure 4.9.

q_{ref}	Arithmetic Mean/Standard Deviation of congestion window of TCP 2					
	$\theta = 0.10$	$\theta = 0.20$	$\theta = 0.50$	$\theta = 1.0$	$\theta = 2.0$	$\theta = 4.0$
2	44.02/1.50	45.48/1.70	47.83/1.90	50.18/2.21	51.77/3.35	56.65/4.19
4	47.31/1.77	49.06/1.96	51.37/2.68	54.33/3.26	56.51/4.47	59.35/5.99
8	52.62/1.74	54.09/2.22	56.16/3.01	58.27/4.09	61.32/5.95	65.01/8.29
16	60.39/1.64	61.54/2.18	63.57/2.97	65.10/3.82	66.63/5.13	69.48/7.95
32	76.28/1.40	77.32/1.75	78.86/2.42	79.98/3.14	81.02/4.04	-

Table 4.4 Arithmetic mean and standard deviation of congestion window of TCP connection with base round trip time of 150 ms and implementing Generalized optimum minimum variance algorithm in Figure 4.9.

q_{ref}	Arithmetic Mean/Standard Deviation of congestion window of TCP 3					
	$\theta = 0.10$	$\theta = 0.20$	$\theta = 0.50$	$\theta = 1.0$	$\theta = 2.0$	$\theta = 4.0$
2	58.03/1.14	59.68/1.64	60.10/1.64	61.04/1.95	61.32/2.73	61.43/3.88
4	60.92/1.29	62.51/1.84	64.39/1.93	66.75/2.73	69.01/3.38	69.88/5.16
8	66.57/1.56	67.47/1.84	70.14/2.38	71.60/3.24	74.14/4.57	75.47/5.50
16	74.75/1.46	76.36/1.81	78.34/2.47	80.27/3.22	82.11/4.57	84.61/6.86
32	91.20/1.39	92.09/1.64	94.43/2.29	95.84/3.02	96.86/4.91	-

Table 4.5 Arithmetic mean and standard deviation of congestion window of TCP connection with base round trip time of 200 ms and implementing Generalized optimum minimum variance algorithm in Figure 4.9.

q_{ref}	Arithmetic Mean/Standard Deviation of congestion window of TCP 4					
	$\theta = 0.10$	$\theta = 0.20$	$\theta = 0.50$	$\theta = 1.0$	$\theta = 2.0$	$\theta = 4.0$
2	73.54/0.91	70.48/1.18	70.80/1.69	72.36/1.96	73.64/3.31	71.64/3.83
4	75.58/1.57	76.15/1.38	76.56/1.88	77.08/2.23	76.76/3.01	79.85/4.61
8	80.21/1.57	81.53/1.81	83.68/2.07	85.37/2.80	86.44/4.21	88.22/4.97
16	89.71/1.58	90.97/1.62	92.83/2.19	94.97/2.78	96.83/3.76	99.25/6.18
32	106.04/1.39	107.53/1.63	109.59/2.14	111.17/2.96	112.30/5.12	-

Table 4.6 Arithmetic mean and standard deviation of congestion window of TCP connection with base round trip time of 250 ms and implementing Generalized optimum minimum variance algorithm in Figure 4.9.

q_{ref}	Arithmetic Mean/Standard Deviation of congestion window of TCP 5					
	$\theta = 0.10$	$\theta = 0.20$	$\theta = 0.50$	$\theta = 1.0$	$\theta = 2.0$	$\theta = 4.0$
2	78.54/2.88	81.24/1.22	82.10/1.60	78.75/2.36	78.56/2.39	76.46/3.72
4	89.08/1.27	88.54/2.28	89.84/1.67	86.46/2.29	85.66/3.66	85.62/3.61
8	94.25/1.40	95.70/1.62	97.14/1.84	97.01/2.34	99.04/3.81	98.50/5.34
16	104.13/1.45	105.58/1.72	107.86/1.89	109.63/2.57	111.40/3.74	112.42/5.06
32	120.59/1.41	122.08/1.48	124.54/2.06	126.61/2.58	128.25/3.88	-

Table 4.7 Arithmetic mean and standard deviation of congestion window of TCP connection with base round trip time of 300 ms and implementing Generalized optimum minimum variance algorithm in Figure 4.9.

4.9.6 Bottleneck link's queue variations characteristics

Associated with congestion window we have also shown the plots of EWMA queue variations at bottleneck Droptail router in the Figures 4.33 to 4.48. Although these EWMA plots does not show the actual queue size but they depict the convergence characteristics of instantaneous queue size. We employ the relation $\bar{q} \leftarrow (1 - w_q) \cdot \bar{q} + w_q \cdot q$ to compute the EWMA of instantaneous queue size as in [Floyd and Jacobson 1993]. This EWMA of instantaneous queue is computed with a low weight, w_q of 2×10^{-5}, so that sudden variations in instantaneous queue size are filtered out properly.

The EWMA queue plots, for $\{(\theta, q_{ref}) = (0.1, 2), (0.25, 2), (0.5, 2), (1.0, 2)\}$ are shown in Figures 4.33, 4.34, 4.35, 4.36, for $\{(\theta, q_{ref}) = (0.1, 4), (0.25, 4), (0.5, 4), (1.0, 4)\}$ are shown in Figures 4.37, 4.38, 4.39, 4.40, for $\{(\theta, q_{ref}) = (0.1, 8), (0.25, 8), (0.5, 8), (1.0, 8)\}$ are shown in Figures 4.41, 4.42, 4.43, 4.44, and for $\{(\theta, q_{ref}) = (0.1, 16), (0.25, 16), (0.5, 16), (1.0, 16)\}$ are shown in Figures 4.45, 4.46, 4.47, 4.48, respectively. These graphs show that as value of q_{ref} is increased the level of steady state queue increases as TCP sources will try to achieve higher reference queue size. It can be observed that for $q_{ref} = 2, 4, 8, 16$ the steady state EWMA is close to 100, 150, 250 and 500 packets, respectively and it is not effected by change in values of θ. There is a sudden rise in queue size at 100s due to introduction of CBR source with sending rate of 4 Mbps. This transient die out in about 25 s for q_{ref}=2, 4 and 8 but for q_{ref}=16, it dies earlier because the congestion window of TCP will drop to lower values.

In order to determine the quantitative performance of queue behavior we computed the arithmetic mean and variance of arithmetic mean of instantaneous queue size in Tables 4.8 and 4.9, respectively. These results show that for a fixed value of q_{ref}, the arithmetic mean does not change significantly with increase in values of θ. The similar behavior has been observed in case of EWMA plots as mentioned before. The arithmetic mean increases with increase in value of q_{ref}. The variance of instantaneous queue size decreases with increase in θ from 0.1 to 1.0 and start to increase for $\theta > 1$. Thus, we achieve minimum variance in instantaneous queue size for $\theta = 1$ which is according to theory of generalized minimum variance window control algorithm. In order to show changes in mean value instantaneous queue size and its variance with changes in θ and q_{ref}, we plot the results given in Tables 4.8 and 4.9 as Figures 4.15 and 4.16, respectively.

q_{ref}	Arithmetic Mean of Instantaneous Queue Size (Packets)					
	$\theta = 0.10$	$\theta = 0.25$	$\theta = 0.50$	$\theta = 1.0$	$\theta = 2.0$	$\theta = 4.0$
2	81	82	82	79	79	78
4	126	125	125	124	123	123
8	215	214	214	212	210	208
16	373	378	359	333	341	334
32	367	371	350	293	291	306

Table 4.8 Effect of change in (θ, q_{ref}) on arithmetic mean of instantaneous queue for generalized optimal minimum variance window control algorithm.

q_{ref}	Variance of Instantaneous Queue Size					
	$\theta = 0.10$	$\theta = 0.25$	$\theta = 0.50$	$\theta = 1.0$	$\theta = 2.0$	$\theta = 4.0$
2	8463	8663	8815	7775	7871	8117
4	12302	12502	12363	12226	12389	12472
8	20712	20491	20318	20207	20301	20317
16	37336	36408	38506	30743	39780	33577
32	37213	35520	37293	29066	33228	24976

Table 4.9 Effect of change in (θ, q_{ref}) on variance of instantaneous queue for generalized optimal minimum variance window control algorithm.

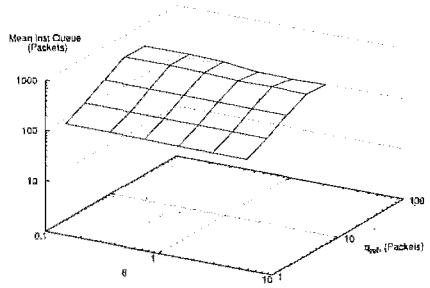

Figure 4.15 Arithmetic mean of Instantaneous queue at the bottleneck link router for the Generalized optimum minimum variance algorithm simulated in Figure 4.9.

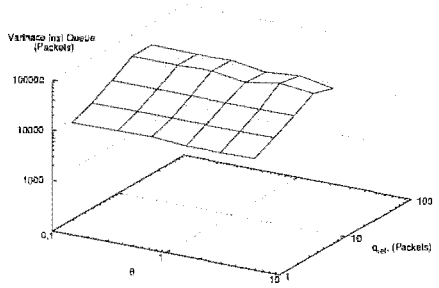

Figure 4.16 Variance of Instantaneous queue at the bottleneck link router for the Generalized optimum minimum variance algorithm simulated in Figure 4.9.

4.10 CONCLUSION

In this Chapter we have presented two new algorithms, namely optimal minimum variance window control and generalized optimal minimum variance window control, for fair end-to-end congestion control. The window form of both algorithms was implemented in TCP and performance was evaluated by using ns based simulations.

We performed three different experiments with optimal minimum variance window control algorithm. In first experiment we found that in case of same value of q_{ref} for all TCP flows, the mean value of instantaneous queue was close to the sum of individual q_{ref} 's. Also we found that sum of sending rates of TCP's is equal to available bandwidth at bottleneck link. Thus, for same value of target queue q_{ref} each TCP will maintain same number of packets in router buffer which is independent of the round trip times of TCP's. Hence we get fair and optimal protocol for same value of q_{ref}.

In second experiment we used different values of q_{ref} for different TCP's. The results are similar to first experiment for mean value of instantaneous queue. The bandwidth allocations among TCP connections were expected to be in exact proportion to q_{ref}. We set q_{ref}'s for different TCP connections in ratio of 1:2:4:8:16 and found that bandwidth allocations were in ratio of

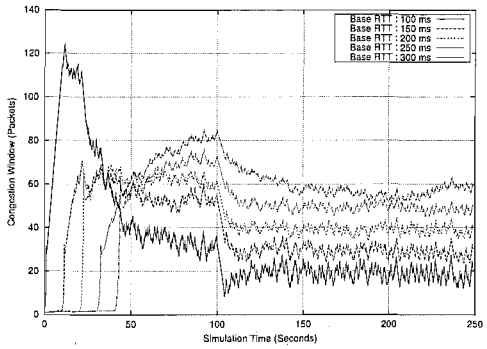

Figure 4.17 Congestion windows of five TCP connections implementing Generalized optimum minimum variance algorithm with q_{ref}=2 and θ=0.10 in Figure 4.9.

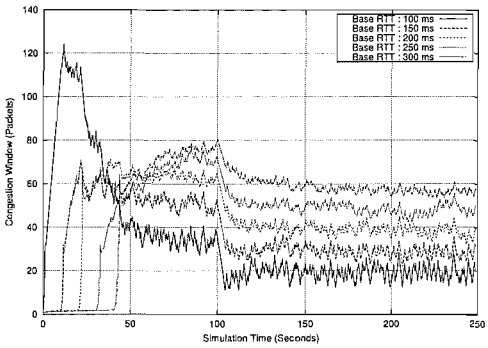

Figure 4.18 Congestion windows of five TCP connections implementing Generalized optimum minimum variance algorithm with q_{ref}=2 and θ=0.25 in Figure 4.9.

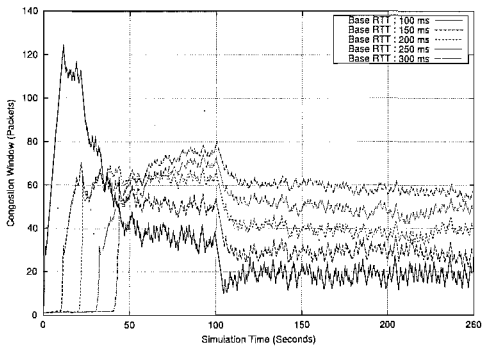

Figure 4.19 Congestion windows of five TCP connections implementing Generalized optimum minimum variance algorithm with q_{ref}=2 and θ=0.50 in Figure 4.9.

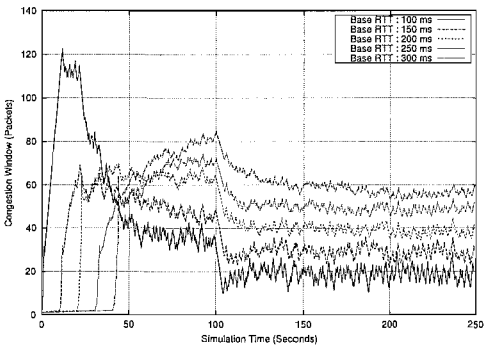

Figure 4.20 Congestion windows of five TCP connections implementing Generalized optimum minimum variance algorithm with q_{ref}=2 and θ=1.0 in Figure 4.9.

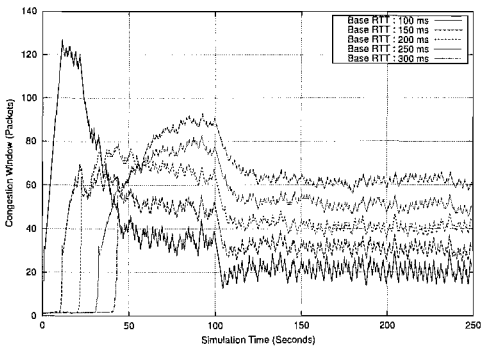

Figure 4.21 Congestion windows of five TCP connections implementing Generalized optimum minimum variance algorithm with q_{ref}=4 and θ=0.10 in Figure 4.9.

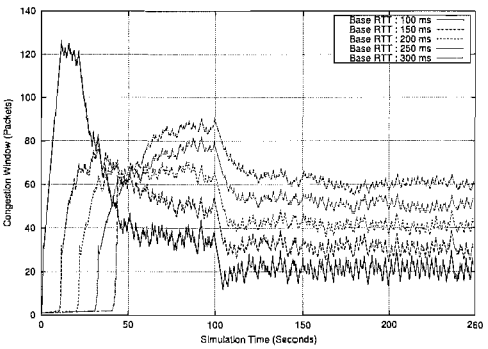

Figure 4.22 Congestion windows of five TCP connections implementing Generalized optimum minimum variance algorithm with q_{ref}=4 and θ=0.25 in Figure 4.9.

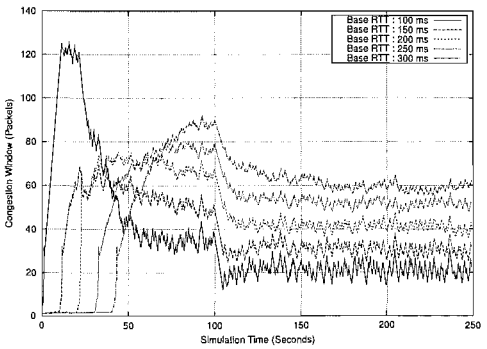

Figure 4.23 Congestion windows of five TCP connections implementing Generalized optimum minimum variance algorithm with q_{ref}=4 and θ=0.50 in Figure 4.9.

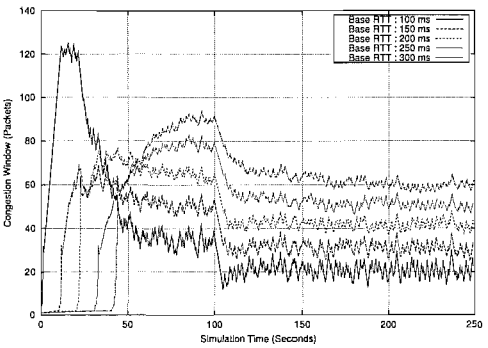

Figure 4.24 Congestion windows of five TCP connections implementing Generalized optimum minimum variance algorithm with q_{ref}=4 and θ=1.0 in Figure 4.9.

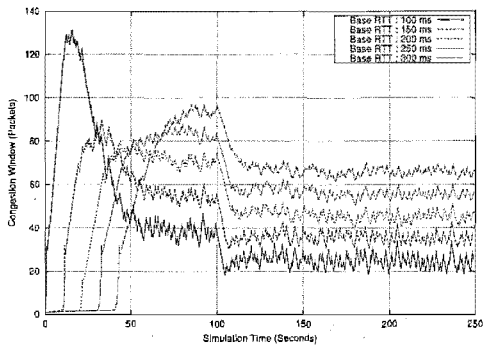

Figure 4.25 Congestion windows of five TCP connections implementing Generalized optimum minimum variance algorithm with q_{ref}=8 and θ=0.10 in Figure 4.9.

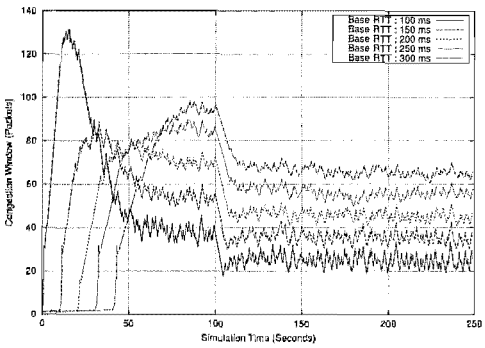

Figure 4.26 Congestion windows of five TCP connections implementing Generalized optimum minimum variance algorithm with q_{ref}=8 and θ=0.25 in Figure 4.9.

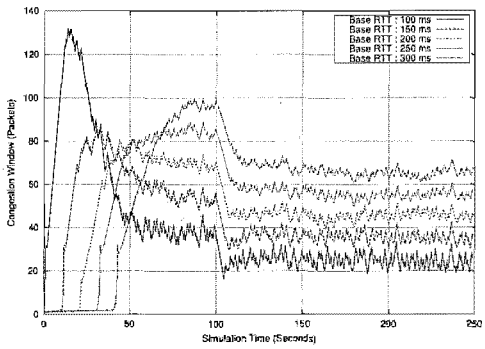

Figure 4.27 Congestion windows of five TCP connections implementing Generalized optimum minimum variance algorithm with q_{ref}=8 and θ=0.50 in Figure 4.9.

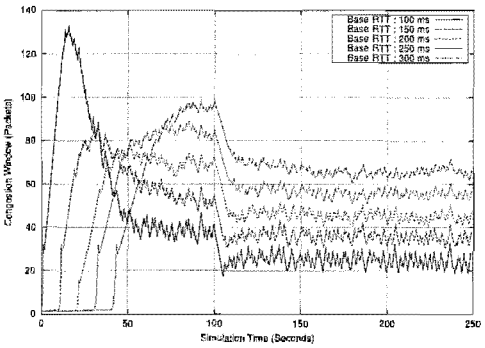

Figure 4.28 Congestion windows of five TCP connections implementing Generalized optimum minimum variance algorithm with q_{ref}=8 and θ=1.0 in Figure 4.9.

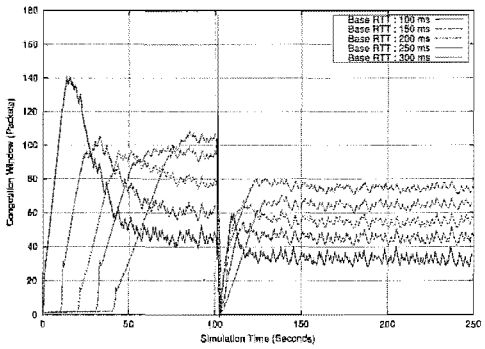

Figure 4.29 Congestion windows of five TCP connections implementing Generalized optimum minimum variance algorithm with q_{ref}=16 and θ=0.10 in Figure 4.9.

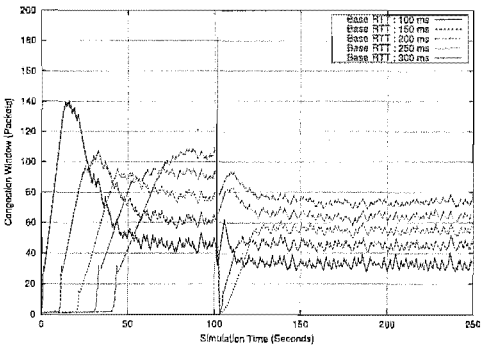

Figure 4.30 Congestion windows of five TCP connections implementing Generalized optimum minimum variance algorithm with q_{ref}=16 and θ=0.25 in Figure 4.9.

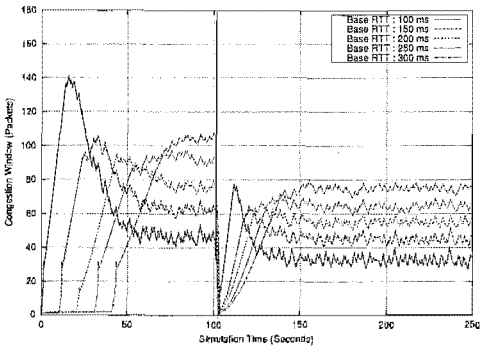

Figure 4.31 Congestion windows of five TCP connections implementing Generalized optimum minimum variance algorithm with q_{ref}=16 and θ=0.50 in Figure 4.9.

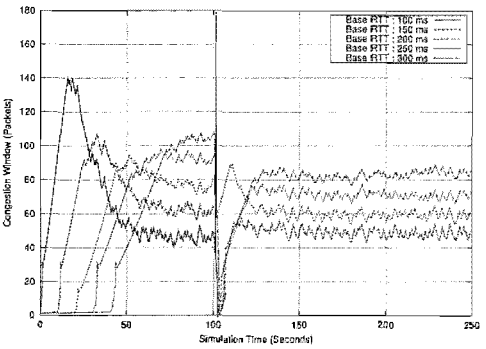

Figure 4.32 Congestion windows of five TCP connections implementing Generalized optimum minimum variance algorithm with q_{ref}=16 and θ=1.0 in Figure 4.9.

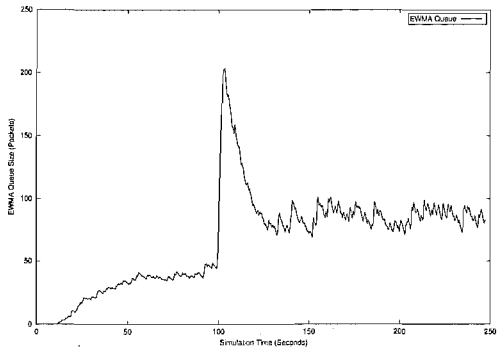

Figure 4.33 The EWMA queue at bottleneck link router for Generalized optimum minimum variance algorithm with q_{ref}=2 and θ=0.10 in Figure 4.9.

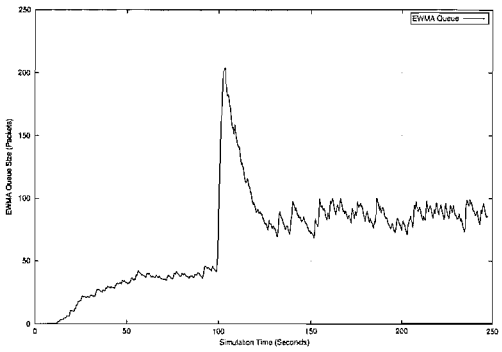

Figure 4.34 The EWMA queue at bottleneck link router for Generalized optimum minimum variance algorithm with q_{ref}=2 and θ=0.25 in Figure 4.9.

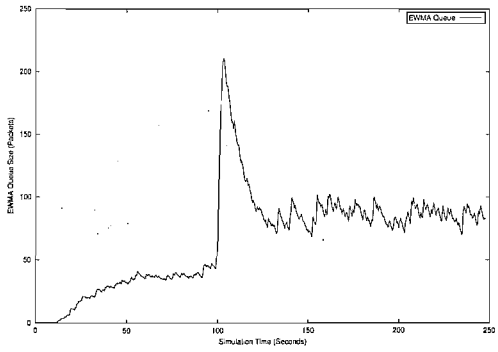

Figure 4.35 The EWMA queue at bottleneck link router for Generalized optimum minimum variance algorithm with q_{ref}=2 and θ=0.5 in Figure 4.9.

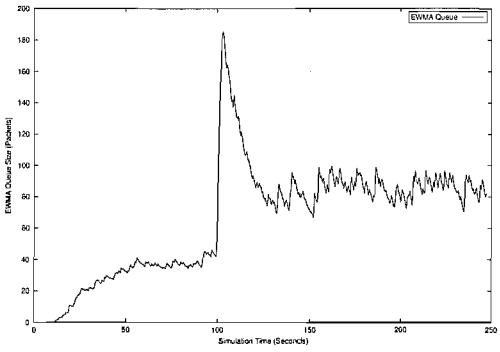

Figure 4.36 The EWMA queue at bottleneck link router for Generalized optimum minimum variance algorithm with q_{ref}=2 and θ=1.0 in Figure 4.9.

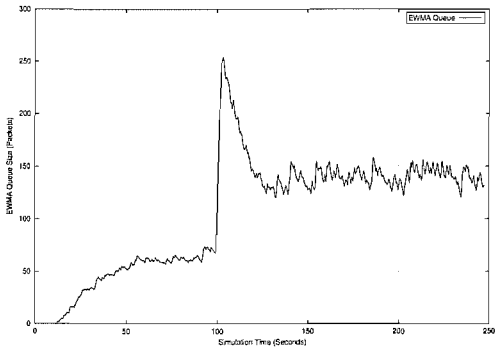

Figure 4.37 The EWMA queue at bottleneck link router for Generalized optimum minimum variance algorithm with q_{ref}=4 and θ=0.10 in Figure 4.9.

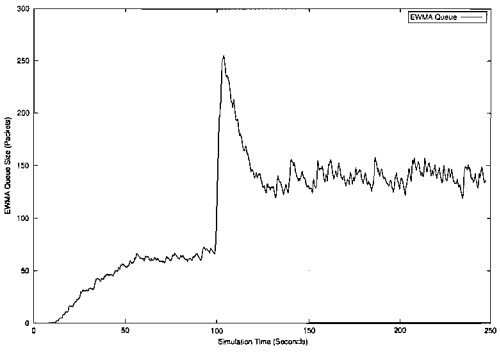

Figure 4.38 The EWMA queue at bottleneck link router for Generalized optimum minimum variance algorithm with q_{ref}=4 and θ=0.25 in Figure 4.9.

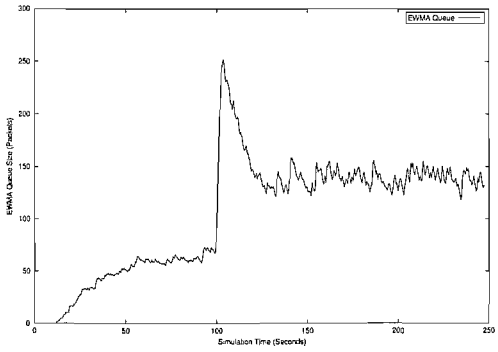

Figure 4.39 The EWMA queue at bottleneck link router for Generalized optimum minimum variance algorithm with q_{ref}=4 and θ=0.5 in Figure 4.9.

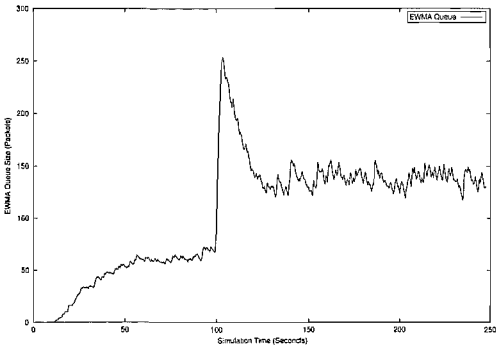

Figure 4.40 The EWMA queue at bottleneck link router for Generalized optimum minimum variance algorithm with q_{ref}=4 and θ=1.0 in Figure 4.9.

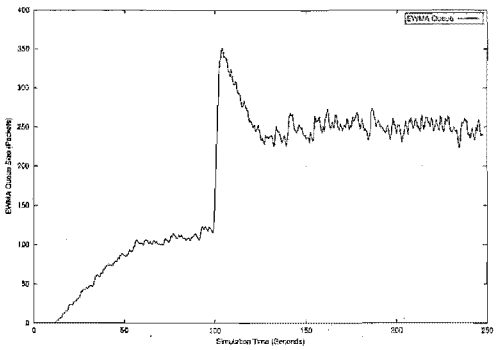

Figure 4.41 The EWMA queue at bottleneck link router for Generalized optimum minimum variance algorithm with q_{ref}=8 and θ=0.10 in Figure 4.9.

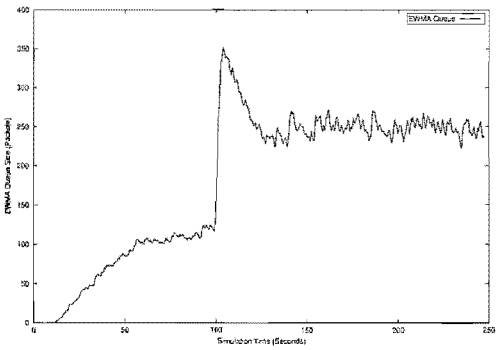

Figure 4.42 The EWMA queue at bottleneck link router for Generalized optimum minimum variance algorithm with q_{ref}=8 and θ=0.25 in Figure 4.9.

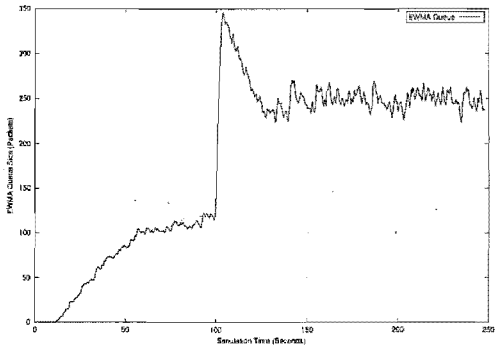

Figure 4.43 The EWMA queue at bottleneck link router for Generalized optimum minimum variance algorithm with q_{ref}=8 and θ=0.5 in Figure 4.9.

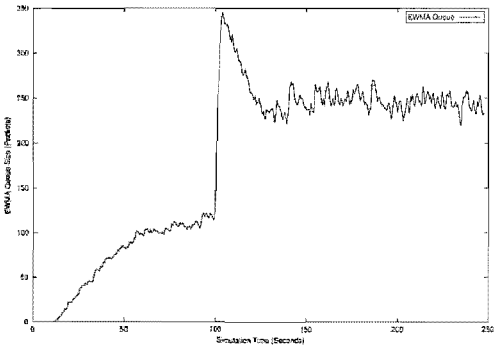

Figure 4.44 The EWMA queue at bottleneck link router for Generalized optimum minimum variance algorithm with q_{ref}=8 and θ=1.0 in Figure 4.9.

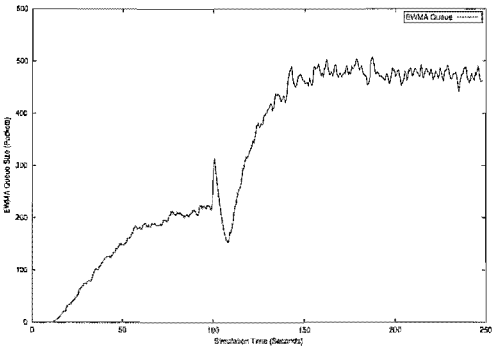

Figure 4.45 The EWMA queue at bottleneck link router for Generalized optimum minimum variance algorithm with q_{ref}=16 and θ=0.10 in Figure 4.9.

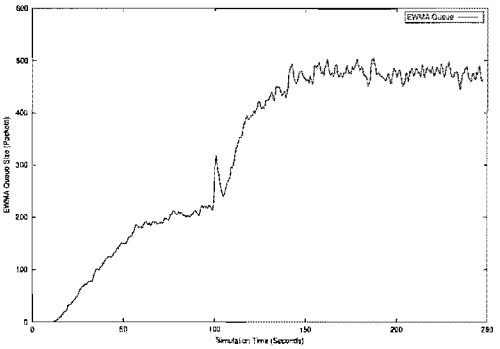

Figure 4.46 The EWMA queue at bottleneck link router for Generalized optimum minimum variance algorithm with q_{ref}=16 and θ=0.25 in Figure 4.9.

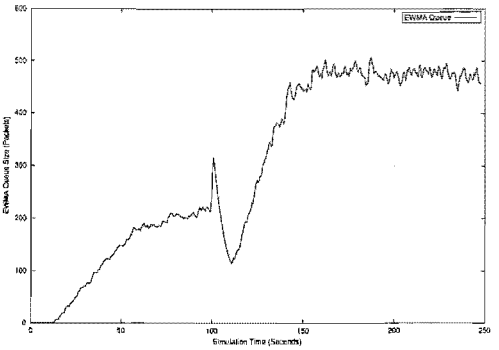

Figure 4.47 The EWMA queue at bottleneck link router for Generalized optimum minimum variance algorithm with q_{ref}=16 and θ=0.5 in Figure 4.9.

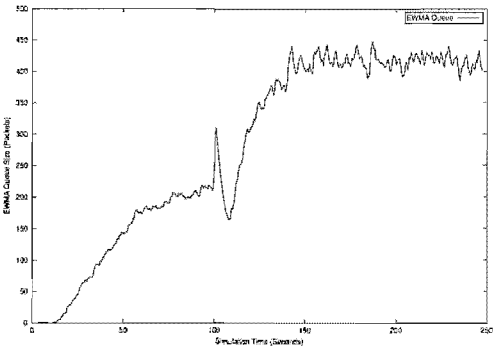

Figure 4.48 The EWMA queue at bottleneck link router for Generalized optimum minimum variance algorithm with q_{ref}=16 and θ=1.0 in Figure 4.9.

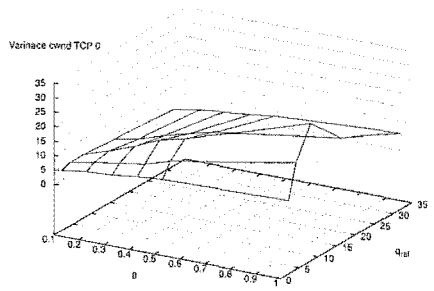

Figure 4.49 Variance of congestion window of TCP with base round trip time of 100 ms and implementing Generalized optimum minimum variance algorithm in Figure 4.9.

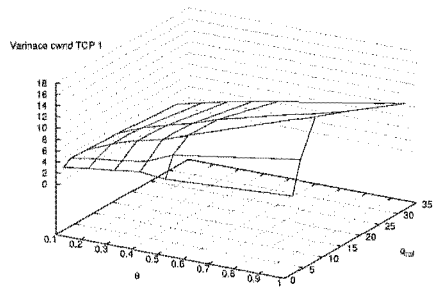

Figure 4.50 Variance of congestion window of TCP with base round trip time of 150 ms and implementing Generalized optimum minimum variance algorithm in Figure 4.9.

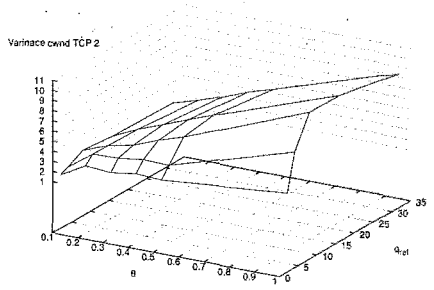

Figure 4.51 Variance of congestion window of TCP with base round trip time of 200 ms and implementing Generalized optimum minimum variance algorithm in Figure 4.9.

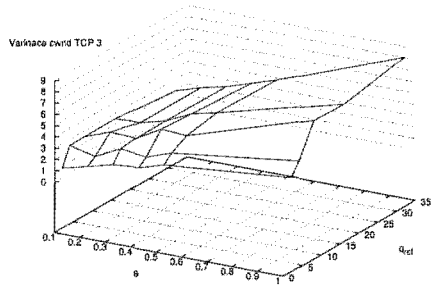

Figure 4.52 Variance of congestion window of TCP with base round trip time of 250 ms and implementing Generalized optimum minimum variance algorithm in Figure 4.9.

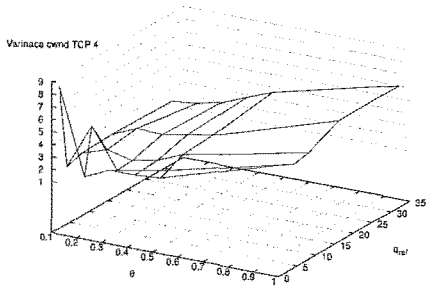

Figure 4.53 Variance of congestion window of TCP with base round trip time of 300 ms and implementing Generalized optimum minimum variance algorithm in Figure 4.9.

1:1.34:1.94:3.15:4.83. One of the probable reason of bandwidth allocations not following q_{ref} is due to following the default algorithms of TCP in conjunction with optimal minimum variance control algorithm in congestion avoidance phase. Though we get different ratio of bandwidth's but it is still increasing with increase in q_{ref}, as required for proportional fairness. Thus, for different values of q_{ref}'s the optimal minimum variance algorithm achieves approximate proportional fairness in our implementation in ns simulator.

We also experimented with self similar traffic in optimal minimum variance algorithm, which shows the usual behavior that as Hurst parameter is increased bandwidth prediction is more accurate. Hence our optimal minimum variance algorithm will show even better performance in self similar traffic environments. Similar performance characteristics can be predicted for TCP employing generalized optimal minimum variance control algorithm.

Since the optimal minimum variance window control algorithm can be obtained by substituting $\theta = 1$ in the expression of generalized optimal minimum variance control algorithm, therefore we concentrated more on general algorithm. We determined a similarity in generalized minimum variance control algorithm and $(p, 1)$ proportionally fair algorithm given in literature. The $(p, 1)$ algorithm utilizes the recently observed bandwidth $B(k-1)$ in contrast to generalized optimal minimum variance algorithm which uses two step ahead prediction of $B(k+1)$. Thus, we found that the implementation of minimum variance algorithms in TCP will lead to proportionally fair allocation of bandwidth.

In order to determine the stability conditions for system based on general algorithm, we performed a detailed analysis using theory of z-transforms for discrete control systems and derived the guidelines for choosing θ and maximum buffer size. The theoretical fairness analysis of generalized algorithm shows that it is possible to achieve fair state by proper choice of tuning parameters both for transient and steady state conditions.

The theoretical analysis of fairness characteristics of generalized optimal minimum variance algorithm was corroborated by simulation experiments. The plots of congestion windows of TCP flows showed that good fairness characteristics can be achieved by using generalized algorithm with suitable choice of tuning parameters. The level of steady state queue size at the bottleneck link router depends upon the choice of q_{ref}, thus buffers of large size are required for routers operating with higher values of q_{ref}. The variance in instantaneous queue size of router implementing generalized algorithm is found to be minimum for $\theta = 1$, which suggests that values of θ close to unity are required for less queue fluctuations.

The variance characteristics of congestion windows of TCP's suggested that for higher values of q_{ref} it was desirable to have smaller values of θ for less variance in congestion windows of TCP connections.

Thus, we concluded that the generalized minimum variance algorithm can be tuned to get high throughput and low variance in congestion window by increasing q_{ref} and decreasing the value of parameter θ. On the other hand, in order to get smaller fluctuations in queue size we need θ to be having higher values closer to unity. Thus, in order to have smaller variance in

congestion windows as well as smaller variance in queue size, we would be required to have θ close to unity and smaller values q_{ref}. However, the smaller values of q_{ref} will give less throughput at the end users side. Coupled with variance characteristics of congestion window and queue size are the fairness index plots which show that the fairness of generalized minimum variance algorithm slightly drops for higher values of θ and q_{ref}. Thus, the generalized algorithm can be operated with desired set of characteristics and its works well both in transient and steady state regions of operation of TCP.

Some of the issues investigated in this Chapter can be explored further, such as accurate estimation of one step ahead bandwidth prediction and errors introduced by incorrect bandwidth estimation. Also the working of TCP's employing the minimum variance algorithms (such as presented in this Chapter) with RED routers has not been investigated and it can be a further direction of work.

In the proceeding Chapters we will investigate and propose the congestion control mechanisms involving the co-ordination among end hosts (using TCP) and routers also known as Active Queue Management policy. The congestion control algorithms presented in this Chapter were implemented in end host, whereas the congestion control algorithms presented in further Chapters are required to be implemented in the routers.

Chapter 5

ROUTER CONGESTION CONTROL ALGORITHMS

The most important and critical factor in robustness of the current Internet is implementation of end-to-end congestion control in TCP end users. Its rationale is scalability and placing complex functions in hosts rather than inside the network. For instance the upgrading of existing functions or addition of new functions to TCP software can be done easily in end hosts and the configurations of network routers need not to be altered with the end-to-end implementation of the TCP paradigm. This policy worked well in the past but in the future routers also need to participate actively in efficient control of congestion due to the heterogeneity of applications in the Internet [Braden *et al.* 1998] and [Floyd and Fall 1999].

As discussed before our approach to the problem of congestion control is also two-fold i.e. at both TCP hosts and routers. The first methodology was applied in the previous two chapters where we presented new fair congestion control algorithms. These algorithms can ensure proper congestion control in scenarios of TCP flows or TCP friendly flows only which is not exactly true in the case of real traffic. Thus, if there are some aggressive flows that do not follow congestion control principles then the end hosts cannot do any thing except suffering from decrease in available bandwidth and thus reducing their sending rate, resulting in gross unfairness. In such a scenario, the router can effectively monitor the aggressive flows and can penalize the non-compliant traffic by either dropping their packets more or pricing them more in the case of quality of service scenarios. One such mechanism, which can drop the packets of aggressive flows more frequently, is the Random Early Detection (RED) algorithm that is explained in this Chapter; its adaptive version will be given in Chapter 6. Other mechanisms which can also control non-compliant flows are also briefly discussed in this Chapter.

We summarize the major congestion control algorithms being employed or proposed for IP routers. We focus on those algorithms which have been recommended for AQM such as RED. The fluid-based feedback control model of the AQM mechanism reviewed in this Chapter will latter become a basis for further development of new congestion control algorithms for AQM routers supporting TCP/IP flows.

This Chapter begins with an outline of the different types of traffic flows through a congested router which are given in Section 5.1. Next we briefly discuss the drawbacks of Droptail routers and explain remedial measures using random packet dropping or marking by RED algorithm in Section 5.2. In Section 5.3 two existing models of AQM are presented. The generalized

Stochastic Differential Equation model of AQM is linearized and after simplifications the s domain transfer functions of TCP and router queue dynamics are presented. Further a block diagram representation of AQM feedback control mechanism is also obtained which will be used as reference in rest of thesis. A brief overview of major congestion control algorithms proposed for TCP/IP supporting routers is presented in Sections 5.4 to 5.13, where some of these algorithms has been proposed to be employed in AQM. After comparison of different algorithms proposed AQM we selected RED for further investigation due to its better qualities. Thus, we present a detailed description of RED algorithm in Section 5.14. In order to introduce a new idea of coupling instantaneous queue size with EWMA queue size in RED type algorithms, we design a new Hybrid RED algorithm in the latter parts of this Chapter. Its major aim is to improve the packet loss rate and link utilization. A literature review of major research work already been done in employing instantaneous queue size along with EWMA queue size for mark/drop decisions has been presented in Section 5.15. Next in Section 5.16 we present Hybrid RED algorithm which uses both Instantaneous queue size and EWMA queue size for packets marking/dropping decisions. The results of Hybrid RED algorithm after multiple runs of simulations are also presented. Finally conclusions are drawn in Section 5.17.

5.1 TYPES OF TRAFFIC FLOWS

The huge amount of traffic through gateway routers, including the Internet, contains a mixture of flows including TCP/IP traffic and different other types of traffic. These different traffic flows through gateway routers can be broadly classified into the following types as suggested in [Braden *et al.* 1998] and [Lin and Morris 1997]:

- **TCP Flows**
  These flows include data traffic originating from sources using TCP/IP as the transport protocol, and follow a standard set of congestion control techniques and algorithms as discussed in Chapter 2.

- **TCP-compatible flows**
  TCP compatible flows behave like TCP flows under congestion and are responsive to congestion notification e.g. traffic generated by TCP Friendly Rate Control protocol [Floyd *et al.* 2000b]. In steady state conditions they use no more bandwidth than a conformant TCP flow running under comparable conditions. They can be further subdivided into the following two types:

  - **Robust Flows**
    These retransmission capable flows always have data to send and take as much bandwidth as the network allows them fairly. They will consume any spare available bandwidth in the network quickly but do not grab bandwidth of other competing flows at times of congestion.

- **Fragile Flows**
  These traffic flows are sensitive to packet losses and adapt slowly to available spare bandwidth, e.g. interactive terminal applications such as *telnet*. They usually send small clusters of packets from time to time and do not cause unfairness problems during their life time.

- **Non TCP-Compatible Flows**
  These flows are responsive to congestion signals from routers but are not compatible with TCP congestion control policies. They may originate due to accidentally or deliberately faulty implementations of TCP. Such applications can grab a grossly unfair share of network bandwidth.

- **Non-Responsive Flows**
  Non-responsive traffic flows do not decrease their packet sending rate even during congestion epochs. An example of such flows is UDP based streaming multimedia voice/video applications. Congestion avoidance algorithms of some of these applications are inadequate or nonexistent. These traffic sources can lead to a congestion in routers. Recently it has been suggested to incorporate congestion avoidance mechanisms such as Receiver Driven Layered Multicast in UDP-based streaming applications [McCanne *et al.* 1996] and [Bolot *et al.* 1994].

5.2 ACTIVE QUEUE MANAGEMENT

The connectionless, flexible, robust and end-to-end architecture of the present Internet requires careful design of the TCP/IP protocol to provide a good service to users under heavy load conditions. Lack of a proper design can result in "congestion collapse" [Jacobson 1988].

The current TCP congestion avoidance mechanisms, while necessary and powerful, are not sufficient to provide good service under all traffic circumstances in routers [Stevens 1997]. It is because there is a limit to the degree of control that can be accomplished from the edges of a network while using an end-to-end architecture with the traditional Droptail technique for managing router queue length.

In Droptail routers there is a maximum set queue length after which packets are dropped until some already enqueued packets are transmitted. The packets drop probability of a typical Droptail router is shown in Figure 5.2. Although such routers have served the Internet well in the past, they may be less useful in the future due to the following major drawbacks:

- **Lock-Out**:
  In some circumstances Droptail routers allows a single connection or a few flows to monopolize the buffer space thus preventing other connections from getting room in the queue buffer. This phenomenon is called "lock-out" and it is often the result of synchronisation or other timing effects [Floyd and Jacobson 1992].

- **Full Queues**:
  The Droptail routers allows queue buffer to maintain a full (or, almost full) status for long periods of time, since Droptail mechanism signals congestion (via packet drop) only when the queue buffer has become completely full. In modern high-speed networks one of the goals of queue management is to reduce the steady-state queue size, thus reducing buffering requirements.

Hence, some new mechanisms are needed in the routers to complement the end-to-end congestion avoidance mechanisms in the TCP protocol [Braden *et al.* 1998] and [Floyd and Fall 1999]. One such approach is called Active Queue Management (AQM) in which a congested router will start marking packets rather then dropping them at times of congestion. These marked packets carry congestion information to traffic sources which will then reduce their sending rate. Such feedback of congestion information from routers to end sources has been standardized in the Explicit Congestion Notification (ECN) policy by introducing four new bits in the packet header [Ramakrishnan 1999]. Random Early Detection (RED) is one of the most commonly used router algorithms for AQM implementation; it will be discussed in detail in Section 5.14.

5.2.1 Objectives of AQM

As given in [Hollot *et al.* 2002] the major objectives of implementing AQM are as follows:

- **Efficient Queue Utilization**:
  It is required that the buffer of queue should be neither empty or underutilized nor completely full during normal operation. If all of the space in buffer of a router's queue is already committed to steady state traffic, or if the buffer space is inadequate, then there will be no space to absorb packet bursts. By keeping the average queue size small, AQM will provide a greater capacity to absorb naturally occurring bursts without dropping packets.

- **Provide lower queuing delay**:
  AQM should be able to reduce queuing delays seen by different traffic flows by keeping a small average queue size. This will be particularly useful for interactive applications such as short world wide web (www) transfers, telnet traffic or interactive audio-video sessions whose performance is better when the end-to-end delay is low.

- **Avoid lock-out behavior**:
  AQM should prevent lock-out behavior by ensuring that there is always some buffer space available for incoming packets. Thus, it should be able prevent router bias against low bandwidth but highly bursty flows.

- **Robustness**
  Since AQM is a closed-loop feedback technique, it should be able to maintain its perfor-

mance characteristics despite changing network conditions such as a change in the number of connections, change in the round trip time or the bottleneck link capacity.

5.3 EXISTING MODELS OF AQM

5.3.1 Firoiu and Borden's Model of AQM Based on RED Algorithm

In [Firoiu and Borden 2000] the authors have investigated the feedback control system comprising TCP and the RED algorithm using the TCP model proposed in [Padhye *et al.* 2000]. They assumed homogenous round trip times in their model and derived a set of guidelines and recommendations to reach the desired equilibrium point. The operation of the system in steady state is considered as a long-term process and quantitative results are presented along with simulation results.

To overcome the shortcomings of the steady state model a transient state quantitative study of congestion control mechanism of RED algorithm is presented in [Firoiu and Borden 2000]. Further a qualitative model to select the queue weight, w_q, for computing the EWMA of instantaneous queue size is also presented in [Firoiu and Borden 2000]. Where for the sampling interval δ, moving average interval I and constant $0 < a < 1$, the expression of w_q is $w_q = 1 - a^{\delta/I}$. It has been observed that a good balance is obtained between EWMA following instantaneous queue size too closely and maintaining EWMA close to long term average by making the I equal to increase-decrease period of TCP congestion window.

5.3.2 Stochastic Differential Equations Based Fluid Model of AQM

The dynamics of a single TCP connection having window size $W(t)$ (packets), round trip time $R(t)$ (seconds), propagation delay τ_p (seconds), $R(t) = \frac{q(t)}{C} + \tau_p$, bottleneck link capacity C (packets/s), number of TCP sessions $N(t)$, probability of packet loss $p(t)$, and queue length of $q(t)$ (packets), can be modelled by the following Poisson Counter Driven Stochastic coupled differential equations [Misra *et al.* 2000] and [Hollot *et al.* 2001a]:

$$\frac{dW(t)}{dt} = \frac{1}{R(t)} - \frac{W(t) \cdot W(t - R(t))}{2 \cdot R(t)} \cdot p(t - R(t)), \tag{5.1}$$

$$\frac{dq(t)}{dt} = \frac{N(t) \cdot W(t)}{R(t)} - C. \tag{5.2}$$

Using equations (5.1) and (5.2) the time domain model of a TCP connection through an AQM router has been constructed in [Chait *et al.* 2002] which is presented in Figure 5.1. After linearization of equations (5.1) and (5.2) the following transfer functions of round trip time delay,

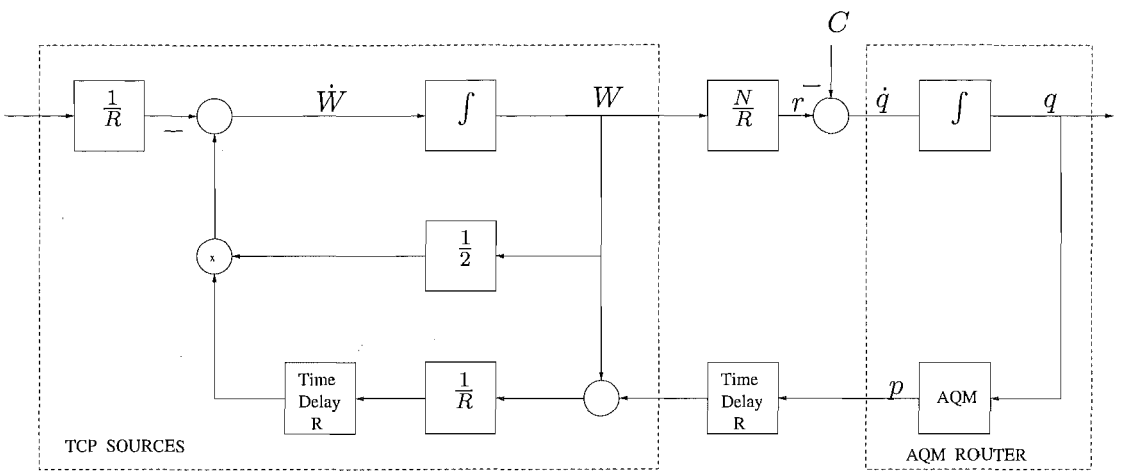

Figure 5.1 The time domain model of a TCP connection through an AQM router.

TCP and queue size variations have been obtained in [Hollot *et al.* 2001a]:

$$P_{td}(s) = e^{-sR}, \tag{5.3}$$

$$P_{tcp}(s) = \frac{\frac{R \cdot C^2}{2 \cdot N^2}}{s + \frac{2 \cdot N}{R^2 \cdot C}}, \tag{5.4}$$

$$P_{queue}(s) = \frac{\frac{N}{R}}{s + \frac{1}{R}}. \tag{5.5}$$

Hence, the overall plant transfer function is given by:

$$P(s) = P_{td}(s) \cdot P_{tcp}(s) \cdot P_{queue}(s) = \frac{\left(\frac{C^2}{2N}\right) e^{-sR}}{(s + \frac{2N}{R^2 C})(s + \frac{1}{R})}. \tag{5.6}$$

A transfer function model of the RED algorithm has also been developed in [Misra *et al.* 2000] which will be given in Section 6.2 of Chapter 6. In further sections (from 5.4 to 5.13) we give a survey of the major congestion control algorithms which are being used in routers and in implementation of AQM.

5.4 DROPTAIL ALGORITHM

Droptail is the simplest and most commonly used algorithm in the current Internet routers which drops packets from the tail of a completely full buffer of queue. Its packet drop probability for maximum buffer size B_q is given in Figure 5.2.

Its main advantages are simplicity, suitability to heterogeneity and its decentralized nature. However, this approach has some serious disadvantages, such as lack of fairness, no protection against the misbehaving or non-responsive flows (i.e. flows which do not reduce their sending

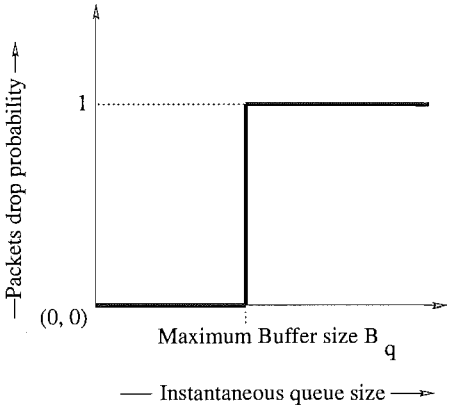

Figure 5.2 Packets dropping probability of Droptail algorithm

rate after receiving congestion signals from congested router, Section 5.1) and no relative Quality of Service (QoS). Also its major drawbacks in the implementation of AQM are already given before in Section 5.2 of this Chapter. QoS is a new concept (which is in contrast to the traditional "best effort" Internet) in which we have some guarantees of transmission rates, error rates and other characteristics in advance [Crawley *et al.* 1998]. It is of particular concern for the continuous transmission of high-bandwidth video and multimedia information. Transmitting this kind of content is difficult in the present Internet with Droptail routers. Generally the Droptail algorithm is used as a baseline case for assessing the performance of the newly proposed router algorithms for implementing of AQM.

5.5 DECBIT ALGORITHM

The earliest example of congestion detection at gateways is provided by the DECbit congestion avoidance scheme [Ramakrishnan and Jain 1990]. In this scheme the congested gateway uses a *congestion-indication* bit in packet headers to provide feedback about congestion. When the average queue length exceeds one the gateway sets a *congestion-indication* bit in the header of the arriving packet.

The sources uses a window-based flow control mechanism. They update their windows of data packets once every two round trip times. If at least half of the packets in the last window had the *congestion-indication* bit set then the window size is decreased exponentially, otherwise it is increased linearly.

The main disadvantages of this scheme are averaging queue size for fairly short periods of time and no difference between congestion detection and indication. Solutions to these problems are attempted in the RED algorithm which is given in the next Section.

5.6 RANDOM EARLY DETECTION ALGORITHM

The Random Early Detection algorithm (RED) was first proposed in [Floyd and Jacobson 1993] and it is recommended for implementation of AQM in [Braden *et al.* 1998]. In the RED algorithm average queue size, $\bar{q}$, is calculated for each packet arrival by using an Exponential Weighted Moving Average (EWMA) as given in [Floyd and Jacobson 1993] and [Young 1984]. The EWMA queue size is compared with a minimum threshold min_{th} and a maximum threshold max_{th} to determine the next action of the router.

The core of the RED algorithm can be summarized as follows: If $\bar{q} \leq min_{th}$ then no incoming packet is marked/dropped, and if $min_{th} \leq \bar{q} \leq max_{th}$ then the arriving packet is marked/dropped with probability p_b which is given by: $p_b \leftarrow max_p(\bar{q} - min_{th})/(max_{th} - min_{th})$. Finally, if we have $\bar{q} > max_{th}$ then all incoming packets are marked/dropped. To make the inter-packet drop probability uniform instead of geometric one can use $p_a \leftarrow p_b/(1 - count \cdot p_b)$ as the marking/dropping probability, where $count$ indicates the number of packets forwarded since the last mark/drop [Floyd and Jacobson 1993]. RED algorithm effectively controls the average queue size while still accommodating bursts of packets without loss and its randomness breaks up synchronized processes that lead to lock-out phenomena. The main disadvantage of RED algorithm is that its performance is very sensitive to the parameters settings and a badly configured RED router will not do better than a Droptail router.

5.7 PROPORTIONAL INTEGRAL CONTROLLER ALGORITHM

In order to overcome the limitations of response speed, stability, coupling between queue length and loss probability of RED algorithm, the Proportional Integral controller (PI) algorithm was designed in [Hollot *et al.* 2001b]. It can be implemented in router by following equation:

$$p := a * (q - q_{ref} - b * (q_{old} - q_{ref}) + p_{old}, \tag{5.7}$$

where p is packet mark/drop probability, q is present instantaneous queue length, q_{ref} is a desired queue length, a and b are constants with $p_{old} := p$ $q_{old} := q$. The operation of PI controller algorithm depends upon proper choice of the constants a and b.

5.8 CHOKE ALGORITHM

In "CHOose and Keep for responsive flows, CHOose and Kill for un-responsive flows" or CHOKe algorithm, [Pan *et al.* 2000], whenever a new packet arrives at the congested gateway router a packet is drawn at random from the FIFO buffer and compared with the arriving packet. If both belong to the same flow then both are dropped else the randomly chosen packet is kept intact and the new incoming packet is admitted into the buffer with a probability that depends on the level of congestion. This probability is computed exactly the same as in RED

algorithm. It is truly a simple and stateless algorithm which does not require any special data structure. However this algorithm is not likely to perform well when the number of flows is large compared to the buffer space. The Block diagram representation of CHOKe algorithm as given in [Pan *et al.* 2000] is shown in Figure 5.3 below:

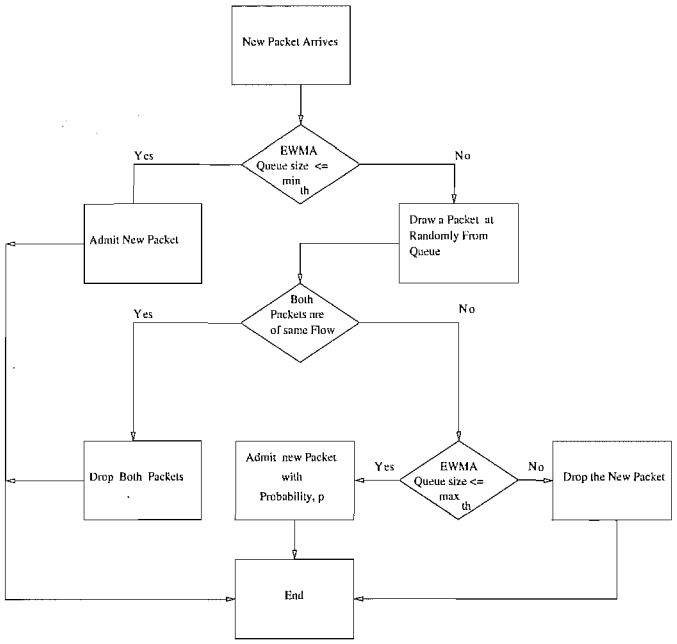

Figure 5.3 Block diagram representation of CHOKe algorithm.

5.9 BLUE ALGORITHMS

The basic idea behind RED algorithm is to detect incipient congestion and to feed back the congestion notification to the end hosts, allowing them to reduce their sending rates accordingly. However, the RED queue length gives very little information about the number of competing connections in a shared link.

BLUE and Stochastic Fair Blue Algorithms (SFB) were designed to overcome these problems by using packet loss and link idle events for protecting TCP flows against non-responsive flows. SFB is highly scalable and enforces fairness using an extremely small amount of state information and a small amount of buffer space. It is a FIFO queuing algorithm that identifies and limits the non-responsive flows based on an accounting procedure similar to BLUE. In [Feng *et al.* 1999b] the authors show by simulation that both BLUE and SFB perform much better than RED algorithm. The main limitation of these algorithms is a possible misclassification of responsive TCP flows as non-responsive.

5.10 RANDOM EXPONENTIAL MARKING ALGORITHM

The Random Exponential Marking (REM) algorithm, given in [Lapsley and Low 1999a], [Lapsley and Low 1999b], [Low and Lapsley 1999], [Athuraliya and Low 2000] and [Athuraliya *et al.* 2001], is a technique for congestion control whose main aim is to achieve a high utilization of link capacity, scalability, negligible loss and delay. The REM algorithm maintains a variable called *price* $p_l(.)$ which is a congestion measure that is updated as follows:

$$p_l(k+1) = [p_l(k) + \gamma(\alpha_l(b - l(k) - b_l^*) + x_l(k) - c_l(k))]^+, \tag{5.8}$$

where $\gamma > 0$ and $\alpha_l > 0$ are small constants and $[z]^+ = max\{z, 0\}$. Here $b_l(k)$ is the aggregate buffer occupancy, b_l^* is a target queue length, $x_l(k)$ is the aggregate input rate to queue and $c_l(k)$ is the available bandwidth. The constant α_l trades off between utilization and queuing delay during the transient period. The constant γ controls the responsiveness of REM to changes in network conditions. At the equilibrium point we have $\alpha_l(b_l(k) - b_l^*) + x_l(k) - c_l(k) = 0$ and $p_l(k+1) = [p_l(k)+\gamma(b_l(k+1)-(1-\alpha_l)b_l(K)-\alpha_l b^*)]^+$ as an update rule. If a packet traverses links $l = 1, 2, ..., L$ that have prices $p_l(k)$ at a sampling instant k then the marking probability $m_l(k)$ at queue l is $m_l(k) = 1 - \phi^{-p_l(k)}$, where ϕ is a constant. The end-to-end marking probability for a packet is $1 - \prod_{l=1}^{L}(1 - m_l(k)) = 1 - \phi^{-\sum_l p_l(k)}$ which can be approximated as $(log_e\phi)\sum_l p_l(k)$.

The main limitations of this algorithm are firstly it gives no incentive to cooperating sources and secondly a properly calculated and fixed value of the constant ϕ must be known globally to all users, [Athuraliya and Low 2000].

5.11 GREEN ALGORITHM

Recently a new algorithm, named GREEN, was proposed in [Wydrowski and Zukerman 2002] for the implementation of AQM in the Internet. It controls the rate of congestion notification to the end hosts by estimating (exponential averaging) the arrival rate of data at the router queue.

A nonlinear model of GREEN has been developed in [KONG *et al.* 2003], which investigates its properties on the basis of feedback control theory. It was shown that when the round trip time and the link bandwidth is relatively small, GREEN has a good performance and is not affected much by changes in network parameters.

However, when the link rate is relatively high or the round trip time is large, GREEN can cause huge oscillations in the router's queue, thereby decreasing the efficiency of AQM, [KONG *et al.* 2003]. Furthermore, to get the optimum performance the parameters of GREEN need to be selected carefully.

No.	Algorithm	Main strengths	Main weaknesses
1.	Droptail	simplicity; no State information required	lacks QoS; no fairness; global synchronisation problems; biased against bursty traffic
2.	DECbit	simple; distributed; optimized; low overhead; congestion feedback by marking packets; dynamic; provides good fairness	use simple averaging; biased against bursty traffic
3.	RED and Variants	simple; fair; QoS; EWMA; AQM; unbiased against bursty traffic	sensitive to parameters settings
4.	PI	simple; fast; robust; AQM; less queue oscillations	difficulty in estimation and setting of constants
5.	CHOKe	simple; stateless; protects against non-responsive flows; easy to implement	difficult to analyze; scalability problems for multiple number of non-responsive flows
6.	BLUE & SFB	low packet loss rate; less buffer needed	not scalable
7.	REM	low packet loss; high link utilization; scalable; low delay	based on globally known parameter ϕ; lacks QoS
8.	GREEN	high link utilization; low delay and packet loss	huge oscillations in queue length for high R
9.	VQ	high link utilization.	Droptail type Virtual queue having fixed buffer capacity
10.	AVQ	adaptive to traffic changes	Droptail type Virtual queue

Table 5.1 Comparison of major congestion control algorithms in the current Internet.

5.12 VIRTUAL QUEUE ALGORITHM

The Virtual Queue algorithm (VQ) is a radical technique reported by Gibben and Kelly [Gibben and Kelly 1999]. In this scheme the link maintains a virtual queue with the same arrival rate as the real queue. However the capacity of the virtual queue is smaller then the capacity of a real queue. When the virtual queue drops a packet then all packets already enqueued in the real queue as well as all of the new incoming packets are marked until the virtual queue becomes empty again. The Virtual queue is useful when feedback delays are short compared to the congestion intervals in a Droptail router as packets will be marked before buffer overflow occurs and dropping of packets start. The fixed size of FIFO based virtual queue is a main weakness of this algorithm. The implementation of a virtual queue in the RED type algorithms is still an open area for research, [Floyd *et al.* 2001].

5.13 ADAPTIVE VIRTUAL QUEUE ALGORITHM

A recent algorithm for AQM is Adaptive Virtual Queue algorithm (AVQ) as presented in [Kunniyur and Srikant 2001]. If C is link capacity and the desired link utilization for packet arrival rate of λ is γ, where $\gamma \leq 1$, then the router maintains a virtual queue of capacity $\widetilde{C}$ where $\widetilde{C} \leq C$. For buffer size of virtual queue same as that of real queue the virtual queue capacity is updated at each packet arrival according to the differential equation $\dot{\widetilde{C}} = \alpha(\gamma C - \lambda)$ where $\alpha > 0$ is the damping factor. Both α and γ determine the stability of the AVQ algorithm. The main limitations of this algorithm is that adaptation of the virtual queue might not correctly follow the changing traffic pattern at the router and presently it is based on Droptail algorithm.

The main strengths and weaknesses of major congestion control algorithms used in routers are summarized in the Table 5.1. After having a survey of important algorithms proposed for AQM we have selected RED algorithm based AQM for further research due to its superior characteristics. In the following Section we give more details of RED algorithm which has been introduced previously in Section 5.6.

5.14 DETAILED DESCRIPTION OF RED ALGORITHM

As already stated in Section 5.6 the RED algorithm was first proposed to alleviate the problems of Droptail algorithm and recently it has also been recommended for implementation of AQM. The performance of AQM based on RED algorithm depends upon four main tuning parameters which are minimum threshold min_{th}, maximum threshold max_{th}, maximum probability max_p and queue weight w_q.

The typical recommended values of four main parameters of RED algorithm i.e. $\{min_{th}, max_{th}, max_p, w_q\}$ are given in Table 5.2. The issue of parameters tuning of the RED algorithm had been discussed in [Floyd 1997] and more recently in [Floyd *et al.* 2001]. In [Floyd 1997] it has been suggested min_{th} = 3 packets, max_{th} = 15 packets, $max_p = 0.1$ and $w_q = 0.002$ with $max_{th} = 3 \cdot min_{th}$ as working guidelines of parameters tuning of RED algorithm. The optimal setting for min_{th} depends upon link speed, propagation delay, maximum buffer size and desired tradeoff between low average delay and high link utilization at router. The difference $(max_{th} - min_{th})$ is suggested to be larger then typical increase in calculated average queue size in one round trip time of a TCP connection [Floyd and Jacobson 1993]. Furthermore, based on the fact that the steady state end-to-end packet drop in a router can not be more then 5% of total enqueued packets, the value of max_p was proposed to be 0.1 and value of queue weight w_q is chosen as 0.002. A set of guidelines for choosing RED parameters is also given in [Sirisena *et al.* 2002]. For proper operation under all of the network conditions the four tuning parameters of RED algorithm, $\{min_{th}, max_{th}, max_p, w_q\}$, are required to be adjusted according to three network scenario parameters, i.e. number of TCP connection N, round trip time R of TCP connections and bottleneck link capacity C, [T. Ziegler and Brandauer 2000]. The issue of automatic tuning of parameters of RED algorithm has been

Parameter Name	Symbol	Role Played in RED algorithm	Default Values proposed in [Floyd 1997]
Minimum Threshold	min_{th}	It is a threshold through which the EWMA queue size must increase before any dropping or marking of packet starts.	5 Packets
Maximum Threshold	max_{th}	It is a threshold through which the EWMA queue size must increase before all packets are dropped/marked	15 Packets $max_{th} = 3 \cdot min_{th}$
Maximum Mark/Drop Prob	max_p	It determines that how aggressively the RED queue marks/drops the packets	0.1
Queue weight	w_q	Time constant for computing EWMA queue size	0.002

Table 5.2 Parameters of RED algorithm.

addressed in the next Chapter where a new Auto-Tuning RED algorithms is presented. In this Chapter we employ the typical values of RED algorithm parameters as given in Table 5.2. The pseudocode of RED algorithm as given in [Floyd and Jacobson 1993] is as follows:

Initialize $\bar{q}$ and count =0;
For each packet arrival:
calculate the new EWMA queue size :
if queue is non-empty
$\bar{q} \leftarrow \bar{q} + w_q \cdot (q - \bar{q})$;
else
$\bar{q} \leftarrow (1 - w_q)^m \cdot \bar{q}$;
if $(min_{th} \leq \bar{q} < max_{th})$
increment count;
Compute $p_a = \frac{p_b}{(1 - count \cdot p_b)}$;
Mark/Drop the arriving packet if $(p_a < R \in [0, 1])$
count=0;
else if $(max_{th} \leq \bar{q})$
Mark/Drop the arriving packet;

Where $m = \left(\frac{t_c - t_i}{t_t}\right)$ and
t_c: is current time,
t_i: is start of idle time,
t_t: is typical transmission time of packet.

5.14.1 Gentle RED Algorithm

The original RED algorithm described above has a discontinuity in its marking/dropping probability profile at $\bar{q} \geq max_{th}$ as shown in Figure 5.4. The probability suddenly jumps to 1 when $\bar{q} > max_{th}$, which could cause violent oscillations in the queue size. This is because all packets would be lost after $\bar{q}$ exceeds max_{th} and the instantaneous queue size will drop until this condition no longer holds.

In order to overcome the discontinuous behavior of the RED algorithm for $\bar{q} > max_{th}$, an optimum sigmoid function of the form $f(x) = \frac{1}{1+e^{-(x+a)}}$ can be used for marking/dropping of packets. Whereas in the gentle version of RED algorithm, as proposed in [Floyd 2000a], a piecewise-linear function is used as a marking/dropping probability profile. Thus, gentle RED algorithm is exactly same as RED algorithm except for the marking/dropping probability function which is linear and continuous for the whole range of router operation. The marking/dropping probability functions of both RED algorithm and Gentle RED algorithm are given in next Subsection which are diagrammatically shown in in Figure 5.4 and 5.5 respectively.

5.14.2 Linear Marking/Dropping Probability Functions of RED and Gentle RED Algorithms.

The packets marking/dropping probability function of the RED algorithm can be written as follows:

$$p_b = \begin{cases} 0, & \text{if } \bar{q} < min_{th}; \\ 1, & \text{if } \bar{q} \geq max_{th}; \\ \frac{\bar{q}-min_{th}}{max_{th}-min_{th}} \cdot max_p, & \text{if } \bar{q} \in [min_{th}, max_{th}]. \end{cases} \tag{5.9}$$

Whereas for gentle RED algorithm the equation (5.9) is modified as follows:

$$p_b = \begin{cases} 0, & \text{if } \bar{q} < min_{th}; \\ 1, & \text{if } \bar{q} \geq 2 \cdot max_{th}; \\ \frac{\bar{q}-min_{th}}{max_{th}-min_{th}} \cdot max_p, & \text{if } \bar{q} \in [min_{th}, max_{th}]; \\ \frac{\bar{q}-max_{th}}{max_{th}} \cdot (1 - max_p) + max_p, & \text{otherwise.} \end{cases} \tag{5.10}$$

The marking/dropping probability function of the RED and gentle RED algorithms as given in equations (5.9) and (5.10) can be plotted as follows:

5.14.3 Instantaneous Queue Size of Router Based on RED Algorithm

The instantaneous queue size, q, of RED routers varies rapidly between zero and the maximum buffer size B_q. When q hits zero value a period of underutilisation of the bottleneck link bandwidth begins. On the other hand when it reaches the upper limit B_q all of the incoming packets

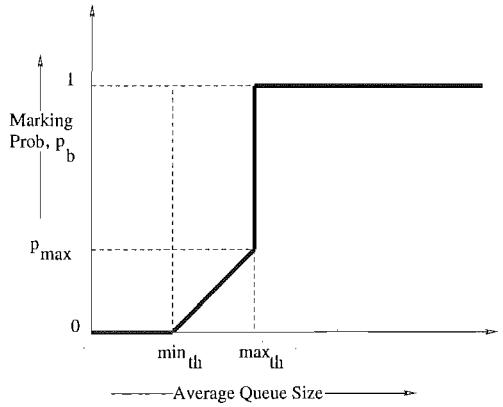

Figure 5.4 Mark/Drop probability profile of RED algorithm.

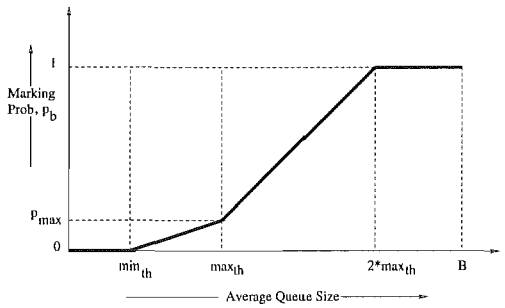

Figure 5.5 Mark/Drop probability profile of Gentle RED algorithm.

will be dropped (called forced drops) which will require their retransmissions. This process will lead to occurrence of congestion. However, sometimes transient periods of bursty traffic can cause wide fluctuations in q for which EWMA queue size would show better results.

5.14.4 EWMA Queue Size of Router Based on RED Algorithm

The RED algorithm given in [Floyd and Jacobson 1993] uses a low pass filter with time constant of w_q to calculate the average queue size $\bar{q}$, which is an exponential weighted moving average or an *Exponential Memory* of instantaneous queue size q [Misra *et al.* 2001]. As given in Section 5.14 the EWMA queue size equations of RED algorithm can be written as follows:

$$\bar{q} = \begin{cases} (1 - w_q) \cdot \bar{q} + w_q \cdot q, & \text{if buffer of RED queue is not empty;} \\ (1 - w_q)^m \cdot \bar{q}, & \text{if buffer of RED queue is empty.} \end{cases} \tag{5.11}$$

where m is a ratio of difference between current and idle time to the typical packet transmission time. We usually assume that RED queue is never empty thus only the first condition in equation (5.11) will be used in the further analysis and design of congestion control algorithms. The performance of the EWMA filter in equation (5.11) is dependent on a proper choice of queue weight w_q. If w_q is too large then transient congestion will not be filtered out and if it is too small then $\bar{q}$ will respond too slowly to changes in q. A number of simulation studies by researchers have suggested that larger values of w_q encourages oscillations in queue size.

In [Ott 1999] and [Misra *et al.* 2001] the EWMA queue size of RED algorithm has been modelled as a delayed Ornstein-Uhlenbeck process, and it is shown that for much smaller values of w_q the memory of process increases but can drive stable occupancy process into oscillatory behavior. Further in [Misra *et al.* 2001] it is suggested that for well designed drop-biasing techniques w_q should be in range of 0.5-1 which is in contradiction to well known simulation results of RED. This discrepancy might be due to poor modelling or approximations.

For the arrival of burst of L packets at a RED router, an alternate expression for $\bar{q}$ is derived which is given as below and also its upper limit is determined in [Floyd and Jacobson 1993].

$$\bar{q} = \left\{ L + 1 + \frac{(1 - w_q)^{L+1} - 1}{w_q} \right\}. \tag{5.12}$$

Upper limit on w_q is obtained by choosing $\bar{q} < min_{th}$ i.e.

$$\left\{ L + 1 + \frac{(1 - w_q)^{L+1} - 1}{w_q} \right\} < min_{th}. \tag{5.13}$$

The lower limit to w_q is computed by the simple observation that it takes $\frac{-1}{\ln(1-w_q)}$ packet arrivals to change $\bar{q}$ from 0 to 0.63 while q stays at 1 packet. In practice for RED type algorithms, the value of w_q is usually chosen small in the order of 0.002 in order to avoid a bias against bursts of packets.

5.14.5 Analysis of Effects of Using EWMA Queue Size in the Operation of RED Algorithm

Suppose a bottleneck link of designed capacity C is being served by a RED router at a rate $< C$. Thus, queue q will not build up and $\bar{q}$ will be small in value. Let another traffic flow joins the session which makes the aggregate packet arrival rate at router ports greater than C. Also suppose that the queue weight w_q of RED has a small value. Now q is large and $\bar{q}$ is computed on packet arrivals which makes $\bar{q}$ to grow much larger than max_{th} after which all of the new incoming packets will be dropped. This dropping of packets will continue for a long time even after the queue has become empty because $\bar{q}$ will take a long time to fall below max_{th}. During this epoch of time, the packet transmission rate from the queue will reach a very low value as almost all of the incoming traffic will be dropped.

This situation will be alleviated after the value of $\bar{q}$ drops below max_{th}. Then we will reach a state where $\bar{q}$ oscillates around max_{th} and packet will be again dropped forcibly [Floyd and Fall 1997]. Thus, depending upon the rate of incoming packets, bottleneck link rate and responsiveness of flows, there might be epochs when all of the packets will be discarded while the link is idle. Two such cases are illustrated in Figures 5.6 and 5.7 where q is well below min_{th} but $\bar{q} > max_th$ and $min_{th} < \bar{q} < max_{th}$ respectively, thus causing unwanted packet drops. This indicates that EWMA queue size by itself does not provide the true picture of backlog in buffers of RED queues. There can be instants when EWMA queue size gives false indications of buffer occupancy which will lead to wastage of precious network bandwidth.

5.14.6 Limitations in Use of EWMA Queue Size in RED Algorithm

There are also following major limitations in use of EWMA queue size in RED type algorithms:

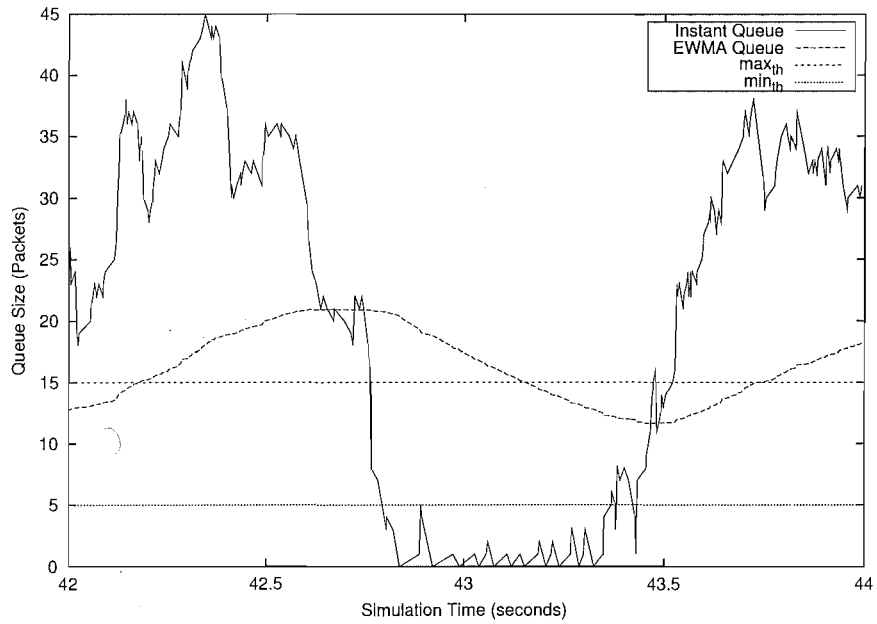

Figure 5.6 A snapshot of instantaneous and EWMA queue sizes when $q < min_{th}$ and $\bar{q} > max_{th}$.

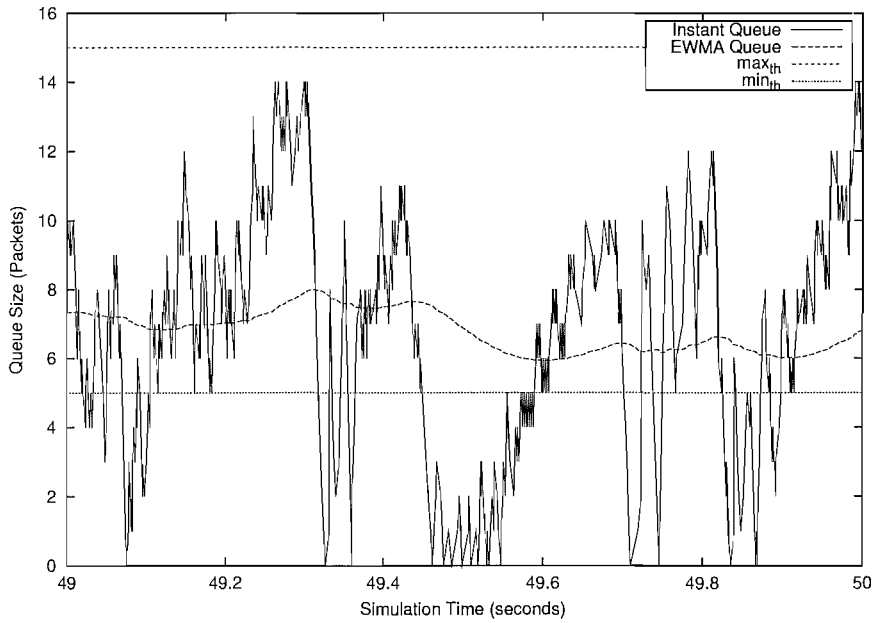

Figure 5.7 A snapshot of instantaneous and EWMA queue sizes when $q < min_{th}$ and $min_{th} < \bar{q} < max_{th}$.

[1] Since EWMA queue size is computed at each packet arrival, it can grow at any rate depending upon the arrival rate. On the other hand, the maximum decrease rate in EWMA queue size is limited by bottleneck bandwidth (or even less since recomputation is only triggered by packet arrivals not departures). Thus, EWMA queue size represents an averaging of the packet arrival rate rather then the true buffer occupancy.

[2] Usually the packet arrival rate is greater than the packet departure rate at the bottleneck

routers. Due to this difference in two rates the equation (5.11) resembles more like a peak detector rather than an EWMA filter (in case the queue weight, w_q, is not selected properly). Thus, in case of high buffer occupancy a sudden surge of traffic will cause EWMA to exceed more than max_{th} which will cause loss of further packets. Thus, it shows bias against transient bursty traffic at larger values of steady state queue buffer occupancy.

Such problems in EWMA queue size are also considered in [Lin and Morris 1997] where a new algorithm, named Flow Random Early Detection (FRED) has been proposed. However, FRED has several performance limitations as given in [Stoica *et al.* 1998] e.g. it maintains state variables for all active flows which makes it less scalable.

In the following Section we present a new algorithm, called Hybrid RED, aimed at overcoming the above mentioned problems by utilizing both instantaneous queue q and its EWMA value $\bar{q}$.

5.15 RELATED WORK ON PACKET MARKING/DROPPING ON USING BOTH EWMA AND INSTANTANEOUS QUEUE SIZE

The use of EWMA queue size for packet marking/dropping in conjunction with a simple linear function requires an optimal value of "queue weight" [Floyd and Jacobson 1993] and [Young 1984]. Furthermore the optimal values for other RED parameters are discussed in [Floyd 1997].

In [May *et al.* 2000] the use of both instantaneous and EWMA queue size for packet marking decisions was investigated on a testbed. In one of the tests the instantaneous queue size was used in the gentle version of RED algorithm [Floyd 2000a]. These experiments showed that EWMA queue size does not help to anticipate congestion and there must be better congestion indicators (like instantaneous queue) size to detect incipient congestion in a network.

In [Jacobson *et al.* 1999] the behavior of the RED algorithm is analyzed using simulations with a small number of long lived TCP flows. A new algorithm was proposed based on the simple models developed therein. The paper suggests that the gain of the filter or w_q should be different for building queues and draining queues. For building queues the response of RED to a unit step function is investigated with w_q chosen to be equal to the inverse of the number of queue samples in a round trip time such that filter will average over a round trip time and approximates the the mean over the round trip time. The analysis shows that RED algorithm takes two round trip times to reach 90% of the persistent queue level. For draining queues it suggests to use w_q equal to one i.e. EWMA queue size should be equal to the instantaneous queue size which might be unrealistic in real world. It is also recommended that min_{th} should be set to 0.3 times and the max_{th} should be 1 to 1.5 times the bandwidth-delay product of the network scenario.

In [Rizzo 1997] it has been pointed out that the RED algorithm needs some type of fine tuning by using both q and $\bar{q}$ in packets mark/drop decisions. Also, the following two mechanisms are proposed to avoid the bias against large values of instantaneous queue q:

- Compute EWMA queue size at departures of packets rather than at arrivals.
- Compute EWMA queue size by periodic sampling of instantaneous queue size.

The above two methods might lead to bias against small values of q because they would not respond properly to bursty traffic due to the coarse sampling being done in both. The LPF/ODA algorithm in [Zheng and Atiquzzaman 2001] aims to improve link utilization when EWMA queue size is at higher values ($min_{th} < \bar{q} < max_{th}$ *or* $\bar{q} > max_{th}$) but instantaneous queue size is at value lower than the min_{th}. The algorithm can be summarized as follows:

- During normal operation the EWMA queue size is computed as in default RED algorithm. This phase is named as *Active Drop Phase*, during which the router will behave as normal RED algorithm.
- If the instantaneous queue size stays below min_{th} for three consecutive packets then it indicates the onset of the *Over Drop Avoidance Phase* during which the EWMA queue size is reduced to half of its present value.
- After this reduction keep on updating the EWMA queue size as in normal RED algorithm.

The working of this algorithm has not been tested throughly yet. The use of instantaneous queue size is also suggested indirectly in [Misra *et al.* 2001] by the recommending w_q to be in range of 0.5 to 1.

5.16 HYBRID RED ALGORITHM

The new Hybrid RED algorithm adjusts $\bar{q}$ during $q < min_{th}$ periods and also takes q into account during enqueue operation of incoming packets. It is derived in two steps as given in the following subsections:

5.16.1 Decreasing EWMA Queue Size

Discretizing equation (5.11) at time instant k and converting the assignment operation to equality yields the following equation:

$$\bar{q}(k) = (1 - w_q) \cdot \bar{q}(k-1) + w_q \cdot q(k). \tag{5.14}$$

By re-arranging terms in equation (5.14) the expression of $q(k)$ is given by:

$$q(k) = \left(\frac{1 - w_q}{w_q}\right) \cdot \bar{q}(k-1) + \frac{\bar{q}(k)}{w_q}. \tag{5.15}$$

Above equations (5.14) and (5.15) indicate the interdependency of q and $\bar{q}$ created through the use w_q in EWMA process as given in equation (5.11). As $q(k)$ varies between 0 and B it also traverses the regions between 0 to min_{th} and between min_{th} and max_{th} of Figure 5.4. If $q(k) \leq min_{th}$ for time intervals greater then a single packet arrival time (or say for $\theta \geq 1$ consecutive packet arrivals) then we need to decrease the first term at RHS of equation (5.14) by factor ξ so that $\bar{q}$ is either less then min_{th} or close to it. In discrete terms we can write this sub-algorithm as:

$$\bar{q}(k) = \begin{cases} (1 - w_q) \cdot \bar{q}(k-1) + w_q \cdot q(k), & \text{if } q > min_{th}; \\ \left(\frac{1-w_q}{\xi}\right) \cdot \bar{q}(k-1) + w_q \cdot q(k), & \text{if } q < min_{th} \;\&\&\; \theta > 1. \end{cases} \tag{5.16}$$

The usefulness of equation (5.16) depends upon the proper choice of the pair $\{\theta, \xi\}$ otherwise it can even degrade the performance of Hybrid RED and even cause stability problems by making wide variations in $\bar{q}$. Thus, there must be a balance between the values of ξ and θ. For larger values of θ we should choose larger values of ξ and for smaller values of θ the ξ must be smaller. If ξ is chosen such that $min_{th} < \bar{q} < max_{th}$ while $q(k) \leq min_{th}$ then we will still have some probabilistic packet drops as governed by equation (5.9) or (5.10). These drops will be fewer in number and thus will cause less bandwidth wastage than in the previous state of $\bar{q} > max_{th}$.

5.16.2 Instantaneous and EWMA Queue Size Based Packets Marking/Dropping

The enqueuing and marking/dropping instances of Hybrid RED as a function of both q and $\bar{q}$ are shown in Figure 5.8 by regions labelled from 1 to 11. Regions 1 and 11 correspond to forced packet dropping and 2, 3, 4, 6, 7 correspond to probabilistic or random packet marking/dropping [Floyd and Fall 1997]. The RED type algorithms either derived from basic or gentle versions do forced mark/drop of packets in all regions for which $\bar{q} > max_{th}$ (4,7 and 10). By contrast in Hybrid RED we can enqueue packets in region 7 and 10 which are ($min_{th} < q < max_{th}$) and ($q < min_{th}$) respectively. A more conservative approach would be to use region 10 only, and not use region 7, for enqueue operations while $\bar{q} > max_{th}$. Also skipping of the region 10 will turn this sub-algorithm OFF and we will default to RED regions only.

5.16.3 Pseudocode of Hybrid RED Algorithm

The sub-algorithms described in subsections 5.16.1 and 5.16.2 can be combined together into the following coherent Hybrid RED algorithm:

```
From the instantaneous queue size, q, compute the corresponding
```

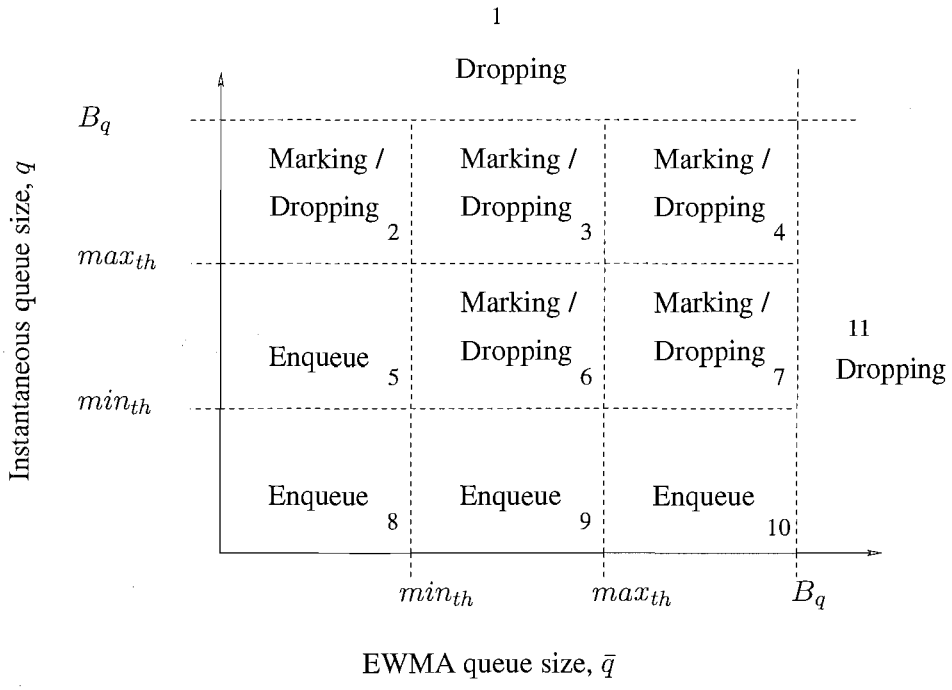

Figure 5.8 Packet enqueuing and marking/dropping logic for Hybrid RED algorithm.

EWMA queue size, $\bar{q}$, at each packet arrival and following the equations (5.9) and (5.10) of RED algorithm under all conditions:

if($q < min_{th}$ for θ consecutive packet arrivals) {
Reduce the existing EWMA by a factor ξ i.e. $\bar{q}/\xi$
Afterwards proceed as in normal RED
}
else if {
if ($\bar{q} > max_{th}$ && $q < min_{th}$)
enqueue packet
}
if ($\bar{q} > max_{th}$ && $min_{th} < q < max_{th}$) enqueue packet otherwise mark/drop as in normal RED.
Where $\{min_{th}, max_{th}, max_p, w_q\} = \{5,\ 15,\ 0.1,\ 0.002\}$, and $max_{th} = 3 \cdot min_{th}$ (packets) as in [Floyd 1997].

In the proceeding Subsections firstly we describe the simulation set up and performance metrics and secondly simulation results for the Hybrid RED algorithm described in this Subsection.

5.16.4 Simulation Setup

The network simulator ns-2.1b8 [DARPA/VINT *et al.* 2001] is used in these simulation experiments. The simulation scenario is illustrated in Figure 5.9 which consists of a dumbbell topology consisting of three links, each having a capacity of 100 Mbps and propagation delays of 1 ms, 3 ms and 5 ms. The bottleneck link has a capacity of 10 Mbps with propagation delay of 5 ms. Each of the three nodes A, B and C in Figure 5.9 is sending FTP traffic on standard TCP Sack protocol [Floyd *et al.* 2000a] for a simulation period of 50 s. The packet size is 1000 bytes. The bottleneck link is using the Hybrid RED router and all other links are using Droptail routers. The buffer size of all routers is set to 50 packets. We use $min_{th} = 5$ and $max_{th} = 15$ packets and $w_q = 0.002$ for Hybrid RED which are also the default values of RED in our ns simulator. We use Hybrid RED in dropping mode rather in packet marking mode. The performance metrices used for the evaluation of Hybrid RED algorithm are plots of EWMA queue size variations, loss rate (δ) and link utilization (η). The values of δ and η are computed according to equations (2.24) and (2.25) respectively.

5.16.5 Simulation Results

We performed simulations for different combinations of $\theta = \{1, 2, 3, 4, 5\}$ and $\xi = \{1.1, 1.25, 1.5, 1.75, 2.0, 2.5, 3.0\}$. The resulting bottleneck link loss rate, δ, and link utilization, η, for single run of simulation are plotted in Figures 5.10 and 5.11 respectively. Also we perform multiple runs (100 times with different sequences of random numbers) of each simulation and computed minimum, maximum and mean values of δ and η.

The results for these multiple run simulations for δ values are shown in Figures 5.12 to 5.17 with associated spread of data between maximum and minimum values. A comparison of δ for RED and Hybrid RED algorithms is presented in Figure 5.18. Similarly the results of multiple run simulations for η values are shown in Figures 5.19 to 5.24 with associated spread of data. Also a comparison of η for RED and Hybrid RED is presented in Figure 5.25.

We show the $\bar{q}$ plots for RED in Figure 5.26 and for Hybrid RED with $\theta = \{3\}$ and $\xi = \{1.1, 1.25, 1.5\}$ in Figures 5.29, 5.30 and 5.31. In Figures 5.11 and 5.25, η attains maximum value in the interval $\xi = 1.1$ to 1.25 for almost all values of θ except when $\theta = 1$, for which the maximum occurs around $\xi = 2.5$. To compare EWMA queue size at different values of θ for a single optimum ξ we also plot Figures 5.27, 5.28, 5.32 and 5.33 for $\{\theta, \xi\}$ pairs $\{1, 1.25\}$, $\{2, 1.25\}$, $\{4, 1.25\}$ and $\{5, 1.25\}$. For the higher values of ξ, EWMA queue size becomes noisy and fluctuates rapidly which might cause queue stability problems. Hence, we recommend the value of $\xi = 1.1$ for most values of θ which we have simulated in this Chapter. On the other hand the higher values of ξ improve bottleneck link loss rate as seen in Figures 5.10 and 5.18. There seems to be a trade off between stable queue behavior and better link utilization as well lower link loss rate. In general the overall performance of Hybrid RED algorithm is better than that of the plain RED algorithm described in [Floyd and Jacobson 1993].

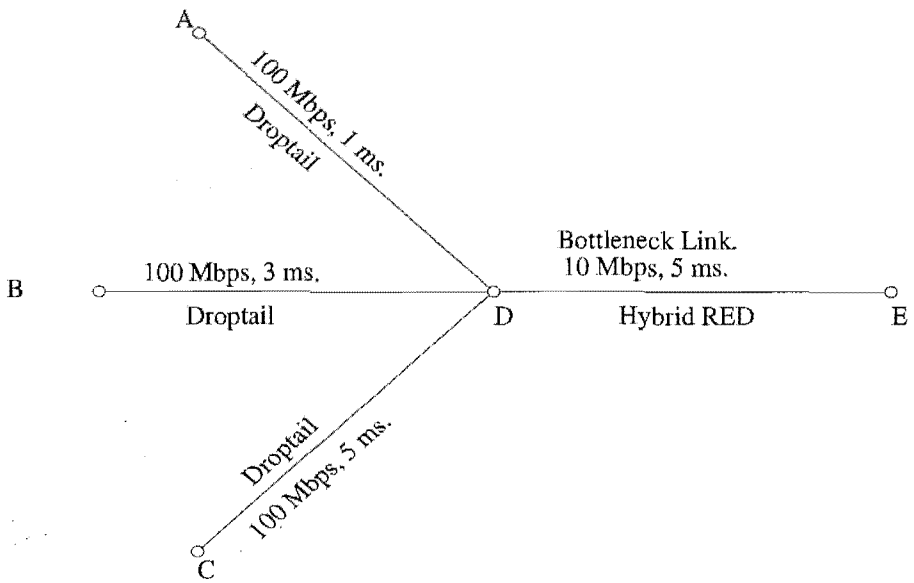

Figure 5.9 Network topology for simulation of Hybrid RED algorithm.

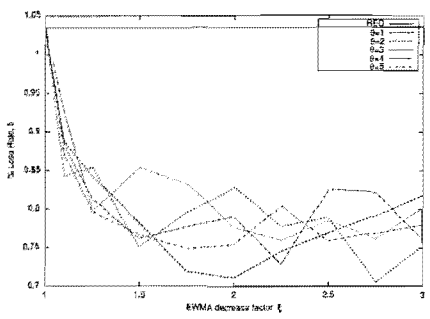

Figure 5.10 Bottleneck link loss rate, δ, with RED and Hybrid RED algorithms after single run of ns simulation.

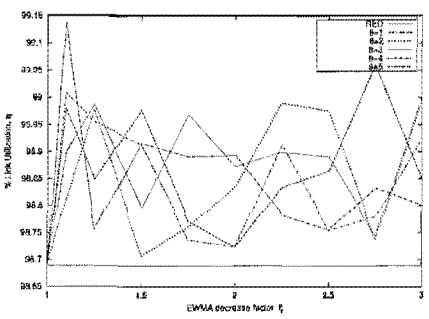

Figure 5.11 Bottleneck link utilization, η, with RED and Hybrid RED algorithms after single run of ns simulation.

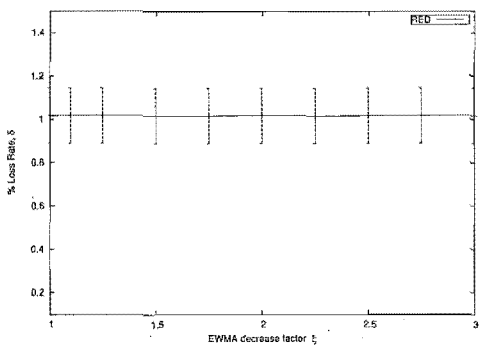

Figure 5.12 Bottleneck link loss rate, δ, with RED in the simulation topology shown in Figure 5.9.

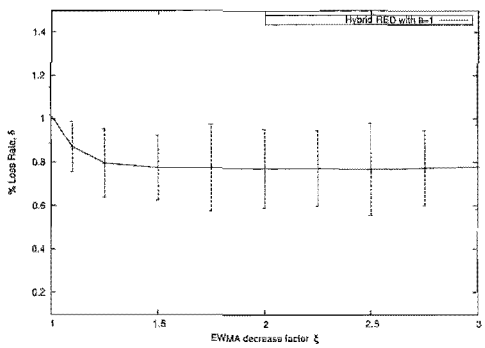

Figure 5.13 Bottleneck link loss rate, δ, with Hybrid RED for $\theta = 1$ in the simulation topology shown in Figure 5.9.

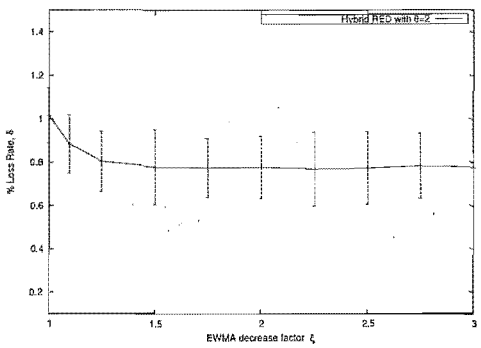

Figure 5.14 Bottleneck link loss rate, δ, with Hybrid RED for $\theta = 2$ in the simulation topology shown in Figure 5.9.

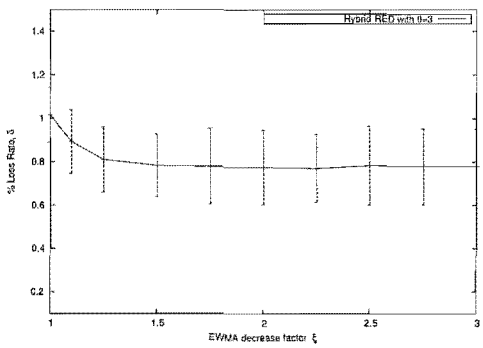

Figure 5.15 Bottleneck link loss rate, δ, with Hybrid RED for $\theta = 3$ in the simulation topology shown in Figure 5.9.

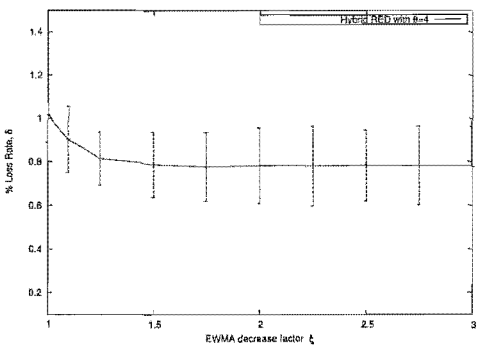

Figure 5.16 Bottleneck link loss rate, δ, with Hybrid RED for $\theta = 4$ in the simulation topology shown in Figure 5.9.

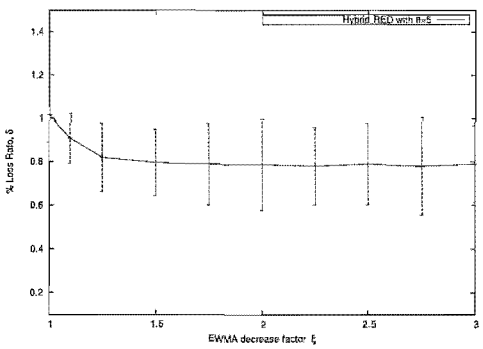

Figure 5.17 Bottleneck link loss rate, δ, with Hybrid RED for $\theta = 5$ in the simulation topology shown in Figure 5.9.

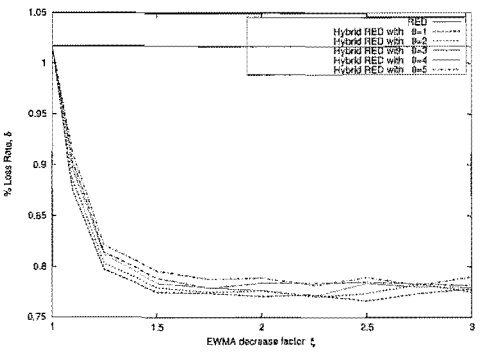

Figure 5.18 Comparison of bottleneck link loss rate, δ, with RED and Hybrid RED algorithms.

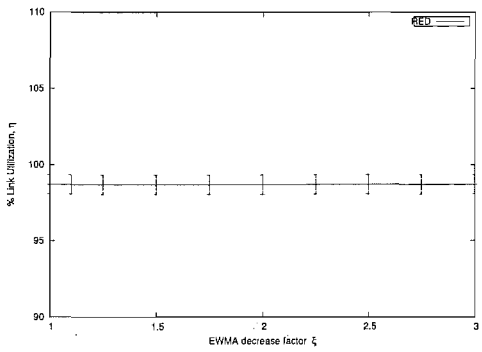

Figure 5.19 Link utilization, η, with RED in the simulation topology shown in Figure 5.9.

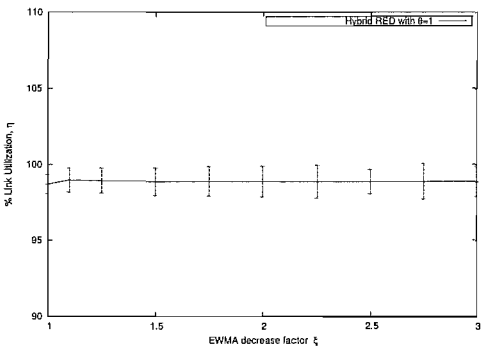

Figure 5.20 Link utilization, η, with Hybrid RED for $\theta = 1$ in the simulation topology shown in Figure 5.9.

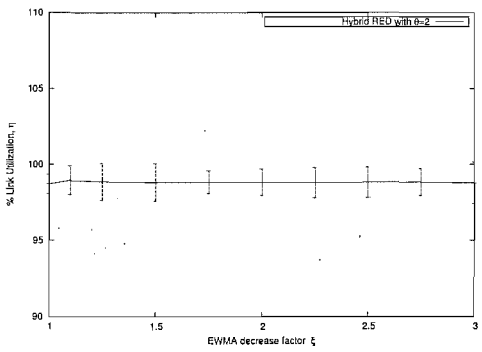

Figure 5.21 Link utilization, η, with Hybrid RED for $\theta = 2$ in the simulation topology shown in Figure 5.9.

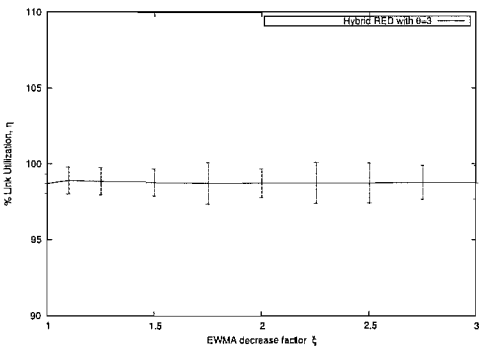

Figure 5.22 Link utilization, η, with Hybrid RED for $\theta = 3$ in the simulation topology shown in Figure 5.9.

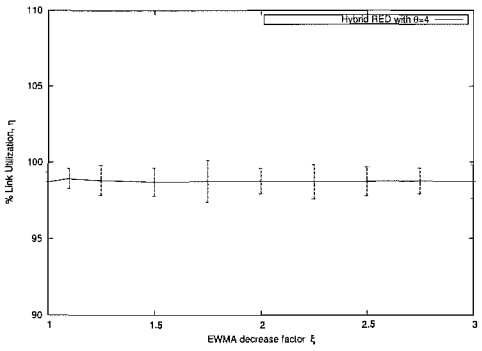

Figure 5.23 Link utilization, η, with Hybrid RED for $\theta = 4$ in the simulation topology shown in Figure 5.9.

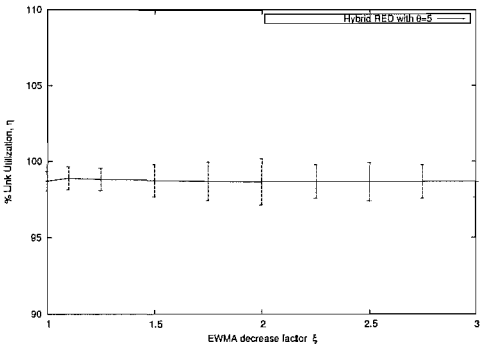

Figure 5.24 Link utilization, η, with Hybrid RED for $\theta = 5$ in the simulation topology shown in Figure 5.9.

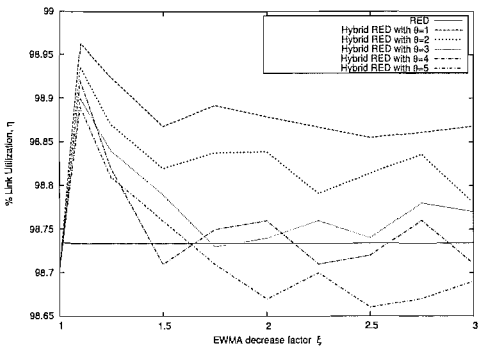

Figure 5.25 Comparison of link utilization, η, with RED and Hybrid RED algorithms.

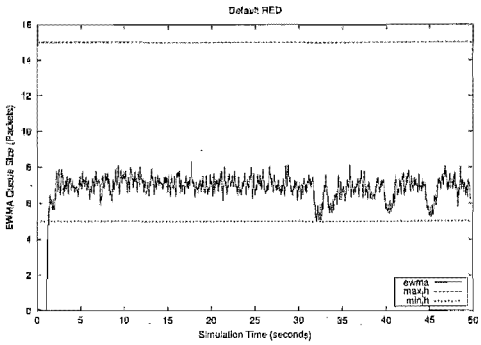

Figure 5.26 EWMA queue size of RED algorithm.

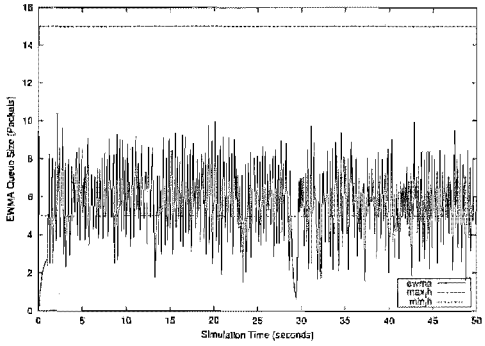

Figure 5.27 EWMA queue size of Hybrid RED algorithm with $\{\theta, \xi\} = \{1, 1.25\}$.

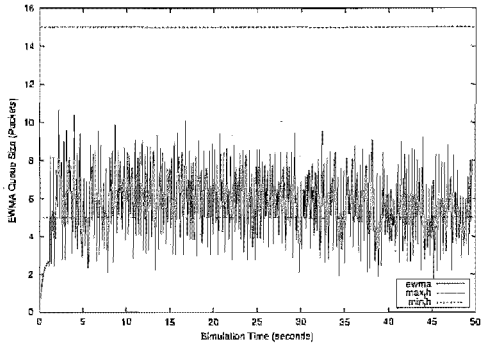

Figure 5.28 EWMA queue size of Hybrid RED algorithm with $\{\theta, \xi\} = \{2, 1.25\}$.

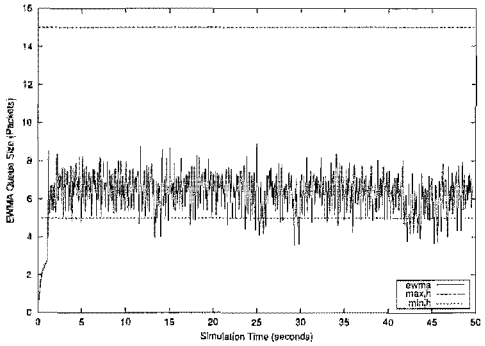

Figure 5.29 EWMA queue size of Hybrid RED algorithm with $\{\theta, \xi\} = \{3, 1.10\}$.

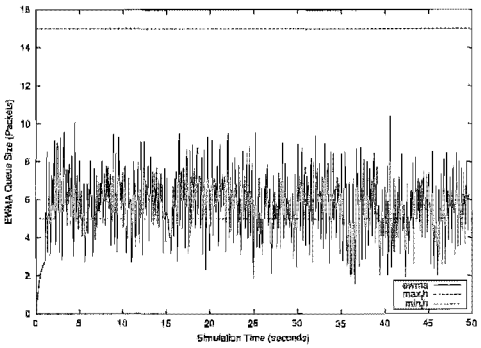

Figure 5.30 EWMA queue size of Hybrid RED algorithm with $\{\theta, \xi\} = \{3, 1.25\}$.

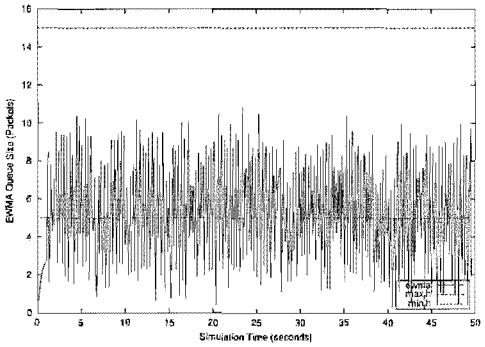

Figure 5.31 EWMA queue size of Hybrid RED algorithm with $\{\theta, \xi\} = \{3, 1.50\}$.

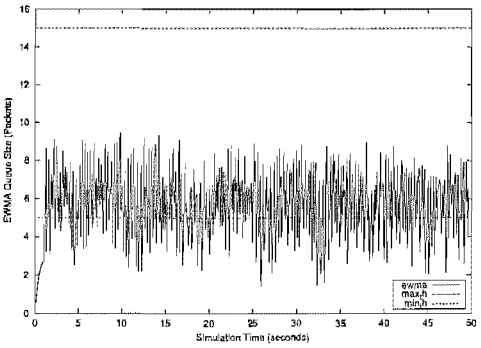

Figure 5.32 EWMA queue size of Hybrid RED algorithm with $\{\theta, \xi\} = \{4, 1.25\}$.

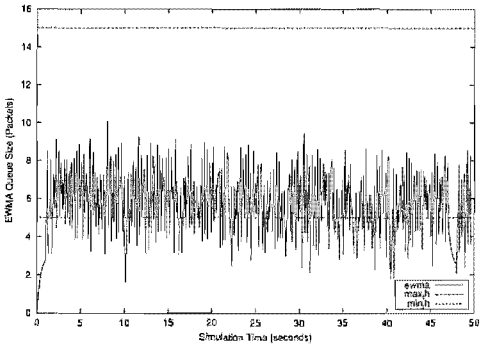

Figure 5.33 EWMA queue size of Hybrid RED algorithm with $\{\theta, \xi\} = \{5, 1.25\}$.

5.17 CONCLUSIONS

This chapter introduced router based congestion control mechanisms which have already been proposed in the literature. It began by describing the types of traffic flows and shortcomings of the Droptail algorithm. This qualitative explanation of drawbacks of the Droptail algorithm motivated the use of RED algorithm which has random mark/drop characteristics. The RED algorithm can be effectively used in breaking the synchronization of flows and overcoming the essential trade off between high throughput and low queuing delay in Droptail routers. Then AQM was discussed along with its main objectives and its existing models. In order to have an overview of major congestion control algorithms being proposed for the gateway routers we present a concise review of core concepts of these algorithms as in existing literature. We have presented a broad range of router algorithms for congestion control in TCP/IP networks and discussed their strengths and weakness. In order to improve the link utilization and loss rate performance of basic RED algorithm as presented in the earlier parts of this chapter we have devised a new algorithm, named Hybrid RED, which combines both instantaneous and EWMA queue size for its operation. The simulations results of the Hybrid RED algorithm further strengthened the concept of combining both instantaneous and EWMA queue size for packets mark/drop decisions in RED algorithm. In order to ensure the accuracy of our results we ran our simulations multiple times and found that the results are more reliable in case of multiple runs because of calculating the mean value and the estimation of associated range between maximum and minimum values. In subsequent Chapters we will not incorporate the Hybrid RED technique in the basic RED algorithm because the performance improvement (queue convergence, response time etc) of other enhancements can be determined separately and systematically. In the next Chapter we will further analyze and design new congestion control algorithms which are based on principle of basic RED algorithm.

Chapter 6

RED BASED AQM AND CONGESTION CONTROL

As described in the last Chapter, traditional Droptail routers will loose data when buffers are full, causing wastage of network resources, and can also cause problems of TCP flow synchronization and correlation of drop events. In order to absorb transient bursts of network traffic such routers needs to have large buffers. Large buffers will increase queuing delay and thus jitter but also the throughput. On the other hand, smaller buffers will decrease both throughput and delay. Thus there exists a trade off between high throughput and low queuing delay in Droptail routers.

This trade off can be overcome by using the Random Early Detection (RED) algorithm, which can simultaneously give low queuing delay and high throughput at the same time. Routers using the RED algorithm and its variants are usually used in conjunction with Explicit Congestion Notification (ECN) to implement the AQM techniques. It forms a proactive congestion control mechanism in which TCP sources have to decrease their congestion windows on receiving congestion signals and thus avoiding further severity in congestion. It has been proved in [Mathis *et al.* 1997] that the average congestion window size and hence the throughput of a TCP connection roughly varies inversely with the square root of the packet dropping probability. Whereas in the case of RED routers the packet dropping probability increases with the number of TCP connections so that the average queue size remains within the threshold limits. Thus, intuitively, in RED AQM the mark/drop probability will increase with the TCP traffic load which will cause TCP sources to decrease their congestion windows, hence avoiding congestion.

Using feedback control theory techniques we develop a new fast-acting and stable version of RED algorithm named as Auto-Tuning RED. This algorithm adapts itself better to changing traffic load. In this algorithm we determine suitable values for RED parameters as well as the adaptation frequency, based on a stability analysis that employs the linearized TCP model presented in [Misra *et al.* 2000].

The convergence of EWMA queue to a desired level after a change in traffic load is important for the estimation of queuing delay which is needed for providing quality of service to end users. The Auto-Tuning RED algorithm aims to achieve convergence of EWMA queue size to a desired target level during sudden variations in traffic load.

We organize this Chapter as follows. The scalability issue of RED algorithm and description of existing adaptive RED algorithm is presented in Section 6.1. The feedback control model of RED algorithm in frequency domain is given in Section 6.2. The expressions for steady state

RED queue length are given in Section 6.3 and a normalized model is presented in Section 6.3.1. The guidelines for choice of fixed and adaptive parameters of RED are given in Section 6.4 and the Auto-Tuning RED algorithm is derived in Section 6.5. Simulation results are given in Section 6.6. This Chapter ends with conclusions and an outline of some further research problems relating to AQM in Section 6.7.

6.1 SCALABILITY OF RED ALGORITHM

The term scalability refers to the scaling up or scaling down of characteristics of router algorithm, such as congestion control algorithms, with an increase or decrease in the number of connections at input ports. The number of connections in a router is always changing and the router needs to be able to adjust itself to these changes. In this Section we investigate the scaling of the tunable parameters of RED algorithm and derive an upper limit for marking/dropping probability as given in [Feng *et al.* 1999a].

Let us consider a RED router supporting N TCP connections, each having maximum segment size of M_s bytes. The bandwidth B of single TCP connection, having round trip time R is derived in [Mathis *et al.* 1997] as:

$$B = \frac{M_s}{R} \cdot \frac{K}{\sqrt{p}}, \tag{6.1}$$

where K is a constant whose value depends upon whether packet losses are periodic or random and on whether there is a per packet ACK policy or delayed ACK policy. An upper bound on the bandwidth of TCP connection is given in [Mathis *et al.* 1997] as:

$$B < \frac{M_s}{R} \cdot \frac{1}{\sqrt{p}}. \tag{6.2}$$

Using equation (6.2), we can approximate the packet loss rate over a single bottleneck link of capacity C for a given number N of TCP connections sharing the bottleneck with absolute min-max fairness. The bandwidth share B_i of each TCP connection would be:

$$B_i = \frac{C}{N} \quad \forall\, i = 1, 2, 3, \ldots, N. \tag{6.3}$$

From equations (6.2) and (6.3) we get the following expression for p:

$$p < \left(\frac{N}{C} \cdot \frac{M_s}{R}\right)^2. \tag{6.4}$$

For $M_s = 1$ packet, we have following inequality:

$$p < \left(\frac{N}{C} \cdot \frac{1}{R}\right)^2. \tag{6.5}$$

Thus, for TCP connections, the upper bound on p will roughly increase quadratically with increasing N as given in equation (6.4), but due to packet retransmission timeouts it will actually vary between linear and quadratic. Hence for a particular flow, fewer packets will be marked/dropped with increasing N, which might cause congestion in the router.

Also with the changes in traffic load at input port of RED router, an adaptation in max_{th} can be made so that required buffer size, $B \geq (max_{th} - min_{th})$, can be dynamically maintained at an optimum value. Without dynamic buffer management, large buffer sizes will be required for even very light loads, thus underutilizing router resources. One such load adaptive RED variation has been proposed in [Aweya *et al.* 2001] where min_{th} and max_{th} are changed while keeping max_p and w_q constant. In this scheme max_{th} is tied to the bandwidth-delay product of link at the start of the algorithm and varied later for a fixed value of maximum loss probability max_p. In this adaptive scheme the focus is to maintain max_p at some fixed level and not on maintaining the queue level at a reference value. Hence, it can provide desired link utilization but no guarantee of convergence to a desired queue level. Therefore, it cannot be used for a rough *a priori* estimate of queue length, and thus queuing delay, which is a major component of quality of service to be delivered to customers. On the other hand, an alternative technique would be to adapt max_p while keeping min_{th} and max_{th} fixed to achieve some target level q_{ref} for queue size as is done in [Floyd *et al.* 2001].

Thus, in general, it is required to adjust tuning parameters, i.e. $\{min_{th}, max_{th}, max_p, w_q\}$, of RED type algorithms with changes in traffic load and configurations of link capacity i.e. with $\{N, R, C\}$, so that optimized router performance level can be achieved under different traffic loads. In this Chapter we adopt the latter approach and perform adaptations in maximum mark/drop probability, max_p, with changes in incoming traffic at the router's input port rather than making changes in min_{th} and max_{th}. In the next Subsection we review a variant of the RED algorithm, [Floyd *et al.* 2001], that adapts the maximum mark/drop probability, max_p, parameter with changes in traffic load.

6.1.1 Adaptive RED Algorithm

As seen in the previous Section, the packets mark/drop probability profile of RED algorithm needs to be adaptive with variations in number of TCP connections N. An adaptive version of RED has been presented in [Feng *et al.* 1999a] where max_p is increased and decreased multiplicatively by arbitrary factors of 2 and 3, respectively. Adaptive RED algorithm presented in [Feng *et al.* 1999a] has been improved in [Floyd *et al.* 2001] by using additive increase and multiplicative decrease policy in max_p with change in traffic load at the router's input port. The main design objectives of adaptive-probability RED algorithms are to decrease queuing delays to minimum possible while maintaining high throughput and to keep EWMA queue within a specified target band, thus avoiding congestion in present networks and to provide QoS in future high speed networks. These conflicting objectives are achieved partially in the improved adaptive RED algorithm, [Floyd *et al.* 2001], which can be summarized as follows:

Every *interval* seconds:
    If ($\bar{q} > target$ && $max_p \leq 0.5$)
    increase max_p additively as:
    $max_p = max_p + \alpha$;
elseif ($\bar{q} < target$ && $max_p \geq 0.01$
    decrease max_p multiplicatively as:
    $max_p = max_p * \beta$;
Variables parameters:
$\bar{q}$: EWMA queue size
Fixed parameters:
time *interval*: 0.5 seconds
target for $\bar{q}$: $[min_{th} + 0.4 * (max_th - min_{th}), min_{th} + 0.6 * (max_{th} - min_{th})]$
α: increment; min $(0.01, max_p/4)$
β: decrease factor; 0.9

The characteristics of convergence to the target level queue of the above given algorithm will be used as a benchmark for assessing performance of the new control-theoretic auto-tuning RED algorithm designed later in Subsection 6.5.3 of this Chapter. In next Section we review a transfer function model of RED algorithm and a feedback control based closed loop model of AQM based on RED.

6.2 TRANSFER FUNCTION MODEL OF RED ALGORITHM

From EWMA dynamics of full RED queue case, as given in equation (5.11), and from linear mark/drop probability functions, as given in equation (5.9), the following transfer function has been derived in [Misra *et al.* 2000]:

$$C_{red}(s) = \left(\frac{L_{red}}{\frac{s}{K_{red}} + 1} \right). \tag{6.6}$$

In equation (6.6) L_{red} is a gain factor and it can be expressed in terms of RED parameters, $\{max_p, max_{th}, min_{th}\}$, as follows:

$$L_{red} = \frac{max_p}{max_{th} - min_{th}}. \tag{6.7}$$

In equation (6.6) the factor K_{red} can be interpreted as the RED filter's bandwidth and the inverse of K_{red} as the filter's "averaging time". For a link of capacity C packets/s used with RED router having queue weight of w_q, the following expression for K_{red} is given in [Misra *et al.* 2000]:

$$K_{red} = C \cdot \log_e(1 - w_q). \tag{6.8}$$

Now with equation (6.6) and transfer functions of TCP and Queue, as given in Chapter 5 Subsection 5.3.2 by equations (5.4) and (5.5) respectively, we have a complete model of AQM which can be used for analysis and performance evaluation and optimization. Thus, when using equation (6.6) as AQM control law, the overall closed loop feedback control model of AQM in Chapter 5 Subsection 5.3.2 can be expressed diagrammatically as in Figure 6.1. The transfer function of RED algorithm, as given in equation (6.6), and normalized form of closed loop model block diagram, shown in Figure 6.1, will be used in this Chapter to design a new control theoretic auto-tuning RED algorithm.

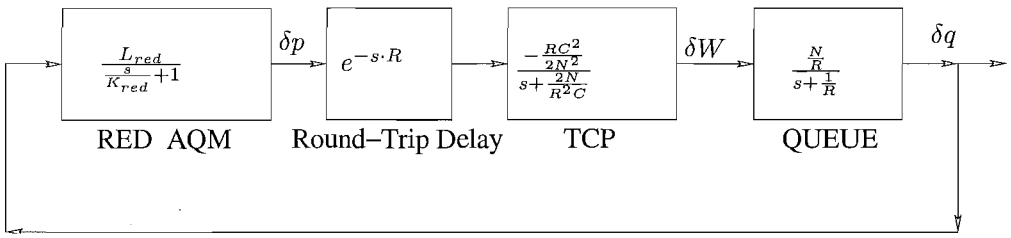

Figure 6.1 Feedback control based closed loop model of AQM based on RED algorithm.

6.3 ANALYSIS OF STEADY STATE QUEUE LENGTH OF RED ALGORITHM

Let C packet/s be the link bandwidth available at a RED router to be shared by N TCP connections after the bandwidth needs of non-responsive connections, including TCP connections that are bottlenecked elsewhere, have been met. We assume that all the connections experience the same round trip time R. It has been shown in [Bonald and Massoulie 2001] that, when the round trip times are different than an equivalent round trip time, equal to the harmonic mean of the individual round trip times, correctly accounts for the steady-state throughput of TCP. Hence, under our assumption, each connection receives its fair share C/N of the total available bandwidth in the steady state. The window size of each connection is thus given by:

$$W = \left(\frac{C}{N} \cdot R\right). \tag{6.9}$$

Expression (6.9) can also be interpreted as the available bandwidth-delay product per connection. In [Mathis *et al.* 1997], the steady state window size W for packet dropping probability p is given by:

$$W = \left(\sqrt{\frac{8}{3}} \cdot \frac{1}{\sqrt{p}}\right). \tag{6.10}$$

As given in equation (5.9), we can have following expression for mark/drop probability p when $min_{th} \leq \bar{q} < max_{th}$:

$$p = max_p \cdot \left(\frac{\bar{q} - min_{th}}{max_{th} - min_{th}}\right). \tag{6.11}$$

We wish to keep $\bar{q}$ close to a reference value, q_{ref}. An appropriate choice is to choose q_{ref}, midway between min_{th} and max_{th}, giving following identity:

$$q_{ref} = \left(\frac{min_{th} + max_{th}}{2}\right). \tag{6.12}$$

When $\bar{q} = q_{ref}$, after substitution of equation (6.12) into (6.11) we get $p = \frac{max_p}{2}$, which on further substitution into equation (6.10) will give the following expression of max_p:

$$max_p = \frac{16}{3} \cdot \frac{1}{W^2}. \tag{6.13}$$

Although the behavior of real networks deviates from equation (6.13), this equation does indicates the trend of the relationship between max_p and W. It also explains why the fixed parameter RED algorithm does not perform well under varying traffic loads. As traffic load N will change then W will also change and equation (6.13) will not remain true for a fixed value of max_p. Thus, the problem is to design an algorithm for adapting max_p to keep $\bar{q}$ close to the q_{ref} despite changes in N, and thus W. Choosing appropriate fixed values for the remaining RED parameters, i.e. $\{max_{th}, min_{th}, w_q\}$, as well as the parameters of the adaptation scheme itself, to ensure stable operation of the scheme, is a complementary part of the problem.

6.3.1 Normalized Model of Queue Length Dynamics of RED

In this Section we normalize the closed loop feedback control model of RED AQM presented in Section 6.2. Analysis given in this Section leads to an important conclusion that for adaptation in maximum mark/drop probability, max_p, it is possible to find fixed values for the remaining RED parameters i.e. $\{min_{th}, max_{th}, w_q\}$, that are appropriate over a wide range of values of N for a given network configuration.

In order to normalize the RED AQM model given in Figure 6.1, let us express the difference between the RED parameters max_{th} and min_{th} as a fraction v of the aggregate bandwidth-delay product:

$$max_{th} - min_{th} = v \cdot C \cdot R. \tag{6.14}$$

Hence, the RED gain, L_{red}, as given in equation (6.7) can be expressed as:

$$L_{red} \equiv \left(\frac{max_p}{max_{th} - min_{th}}\right) = \left(\frac{max_p}{v \cdot C \cdot R}\right). \tag{6.15}$$

Expressing the RED filter's averaging time T_{red}, defined in Section 6.2 as a multiple n of the round trip time R we get:

$$T_{red} = \frac{-1}{C \cdot ln(1 - w_q)} = n \cdot R. \tag{6.16}$$

We can rewrite equation (6.16) as follows:

$$n = \frac{-1}{C \cdot R \cdot ln(1 - w_q)} \approx \frac{1}{(C \cdot R \cdot w_q)}. \tag{6.17}$$

Next we express the TCP model's gain and time constant in equations (6.15) and (6.16) in terms of the window size W of an individual connection using the relationship in equation (6.9). Finally we time-scale the model by choosing the round trip time R as the unit of time, thus transforming the model's Laplace Transform operator s to s/R. The resulting normalized closed loop transfer function, which determines the system's stability and speed of response is given by:

$$G(s) = \frac{\left(\frac{W \cdot max_p}{2 \cdot v \cdot n}\right) \cdot e^{-s}}{(s + \frac{2}{W})(s + 1)(s + \frac{1}{n})}. \tag{6.18}$$

The normalized model of RED algorithm based AQM, as shown in Figure 6.2, is applicable

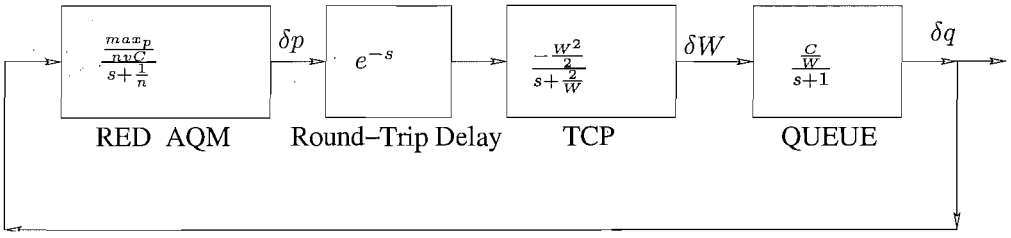

Figure 6.2 Feedback control based normalized closed loop model of AQM based on RED algorithm.

to a wide variety of links having different capacities. In the next Section we present guidelines to select the fixed parameters of RED algorithm which will be used with the new Auto-Tuning RED algorithm.

6.4 CHOICE OF FIXED PARAMETERS OF RED ALGORITHMS

6.4.1 Model Parameter Ranges

We consider the closed loop dynamics in the vicinity of the desired operating condition, i.e. once max_p has adapted to the correct value as given by equation (6.13). The loop transfer function in equation (6.18) then simplifies to

$$G(s) = \frac{\left(\frac{3}{2 \cdot v \cdot W \cdot n}\right) \cdot e^{-s}}{(s + \frac{2}{W})(s + 1)(s + \frac{1}{n})}. \tag{6.19}$$

Inspection of equation (6.19) reveals that the system's dynamics are dependent on three quantities which are W, v and n. There is one invariant open loop pole at $s = -1$ (which was $s = -1/R$ before normalization) produced by the buffer dynamics. Note that the buffer deviates from being a pure integrator because the round trip time R varies with buffer occupancy.

Let us consider the locations of the other two open loop poles. The pole at $s = -1/n$ belongs to the RED filter. The results in [Floyd *et al.* 2001] indicate that $n = 10$ is a good value; the filter then averages the queue length over 10 round trip times. We explore the choice of n more carefully in the next Section and confirm that the RED filter pole should be an order of magnitude closer to the origin than the fixed pole at -1. Second pole (inherent in transfer function of TCP) is at $-2/W$ and its location depends on the (average) window size W of an individual connection. A lower bound on W under normal network conditions would be 4 (making triple duplicate acknowledgments possible in the event of a packet loss). This implies that the "pole of TCP" is to right of the fixed pole at $s = -1$. A practical upper bound on W would be 64, arising from the default TCP window size of 64 Kbytes, for a packet size of 1 Kbyte. When $W > 2n$, which occurs under a light network loading, the pole of TCP is closest to origin of the three open loop poles.

6.4.2 Choice of Filter Averaging Time

Inspection of the transfer function in equation (6.19) shows that, for a given (average) window size W, the stability of the feedback system will be determined by the values of n, the RED filter averaging time expressed as a multiple of the round trip time R, and v, the difference between max_{th} and min_{th} expressed as a fraction of the aggregate bandwidth-delay product CR. Basically, the system becomes more stable as v increases as then the gain (slope) of the RED characteristic decreases.

The choice of $n = 10$ is recommended in [Floyd *et al.* 2001]. We will consider this value as well as two values on either side: $n = 3$ and $n = 30$. In each case, we find the values of v that give a phase margin of 45 deg ($\frac{\pi}{4}$ rad) for different values of congestion window, W: 2, 6, 20 and 64. The system's gain crossover frequency ω_c occurs when the transfer function's phase angle becomes $-\frac{3\pi}{4}$ with a gain magnitude of 1. This gives us the following pair of equations:

$$\tan^{-1}(\omega_c) + \tan^{-1}(W \cdot \frac{\omega_c}{2}) + \tan^{-1}(n \cdot \omega_c) + \omega_c = \frac{3\pi}{4}, \tag{6.20}$$

$$\frac{2vWn}{3} \cdot \sqrt{1+\omega_c^2} \cdot \sqrt{\frac{4}{W^2} + \omega_c^2} \cdot \sqrt{\frac{1}{n^2} + \omega_c^2} = 1. \tag{6.21}$$

These equations can be solved numerically for the required values of v and the corresponding crossover frequency ω_c. The results are given in Table 6.1, for the different combinations of n and W. The closed loop system's gain varies inversely as v, hence the phase margin increases with v thus the values of v given in the table are the minimum satisfactory value in each case. Observe that when $n = 3$, the value of v must be greater than 0.427 for satisfactory performance over the full range of window sizes. This means that $max_{th} - min_{th}$ must be over 0.42 of the whole bandwidth-delay product, which perhaps implies too large a buffer. For $n = 10$, this reduces to 0.21 times the product which would be more acceptable. An even smaller buffer size is possible with the choice $n = 30$, with $max_{th} - min_{th}$ having to be just 0.11 times the product.

W (Packets)	$n = 3$		$n = 10$		$n = 30$	
	v	ω_c (rad/s)	v	ω_c (rad/s)	v	ω_c (rad/s)
2	0.409	0.485	0.177	0.362	0.072	0.304
6	0.427	0.361	0.211	0.255	0.093	0.203
20	0.322	0.255	0.205	0.162	0.109	0.113
64	0.169	0.201	0.135	0.111	0.090	0.066

Table 6.1 Design of averaging time for RED algorithm with phase margin of 45 deg.

But the lower values of the crossover frequency ω_c show that a price would have to be paid by way of a lower speed of response.

Another risk of making n too large is that it would delay the response to a change in traffic conditions. Hence we conclude that $n = 10$ is a good choice for the filter averaging time which by equation (6.16) implies the parameter setting given by:

$$w_q = 1 - e^{\frac{-1}{10 \cdot C \cdot R}} \approx \frac{1}{10 \cdot C \cdot R}. \tag{6.22}$$

The analysis given in this Section has theoretically proved the simulations based recommendations for RED parameters selection in [Floyd *et al.* 2001].

6.4.3 Choice of Thresholds

With the choice of $n = 10$, from Table 6.1, we have $v = 0.21$ to get a phase margin of 45 degrees. Then, by equation (6.14), the buffer thresholds should satisfy:

$$max_{th} - min_{th} \geq 0.21 \cdot C \cdot R. \tag{6.23}$$

if we require a phase margin of at least 45 deg (Phase Margin can be shown to increase as v decreases). A common guideline for the ratio of thresholds is

$$max_{th} = 3 \cdot min_{th}. \tag{6.24}$$

Substituting equation (6.24) in (6.23), we get:

$$min_{th} \geq 0.13 \cdot C \cdot R. \tag{6.25}$$

The guideline for selecting min_{th} as given in equation (6.25) is in contrast to that given in [Floyd *et al.* 2001], i.e. $min_{th} = Max\left[5, \frac{delay_{target}C}{2}\right]$, where it is arbitrarily specified that $delay_{target} = 5ms$. As this value is not related to the round trip time R, it would not cover all the scenarios of network topologies.

After presenting the guidelines for selecting the fixed RED parameters i.e. maximum threshold max_{th}, minimum threshold min_{th} and queue weight w_q, in subsections 6.4.3 and 6.4.2 respectively, we design a new Auto-Tuning RED algorithm in the next Section which adapts the maximum probability, max_p, parameter with the changes in traffic load offered to RED router.

6.5 THE AUTO-TUNING ALGORITHM

6.5.1 Derivation

The key insight leading to our new approach is that the correct value of max_p for the existing traffic conditions can be determined from the current value of the (averaged) queue size $\bar{q}$ and the, in general incorrect, current max_p setting of a RED router. The marking/dropping probability corresponding to $\bar{q}$, in the normal operating region between min_{th} and max_{th}, is given by :

$$p \leftarrow \left(\frac{\bar{q} - min_{th}}{max_{th} - min_{th}} \right) \cdot max_p. \tag{6.26}$$

Now for the correct max_p, this value should occur when the queue size is q_{ref}. Also, under our standing assumption in equation (6.12) the correct value of max_p is obviously double that given by equation (6.26). Thus we have:

$$\Delta max_p \leftarrow 2 \cdot \left(\frac{\bar{q} - min_{th}}{max_{th} - min_{th}} \right) \cdot max_p - max_p. \tag{6.27}$$

The equation (6.27) may be interpreted as an increment in max_p, which after simplification is given by:

$$\Delta max_p \leftarrow 2 \cdot \left(\frac{\bar{q} - q_{ref}}{max_{th} - min_{th}} \right) \cdot max_p. \tag{6.28}$$

The equation (6.28) implies that the correct max_p can be obtained in a single iteration by a multiplicative change in the existing value, assuming that $\bar{q}$ has settled down to its steady-state value. However, unlike in [Feng *et al.* 1999a] and [Floyd *et al.* 2001], the multiplicative factor is not an arbitrary constant but is proportional to the error between the $\bar{q}$ and q_{ref} which is intuitively satisfying. Thus the change factor is large when the error is large but tends to zero as $\bar{q}$ approaches q_{ref}. Although equation (6.28) was derived on the basis that $\bar{q}$ fell between min_{th} and max_{th}, its form makes sense even when $\bar{q}$ falls outside this interval and so will use it globally subject to the limits specified later arising from practical considerations.

6.5.2 Frequency of Adaptation

We now consider the frequency at which max_p should be updated. An obvious choice is that the time between updates should be sufficient for $\bar{q}$ to settle down after a change in traffic conditions.

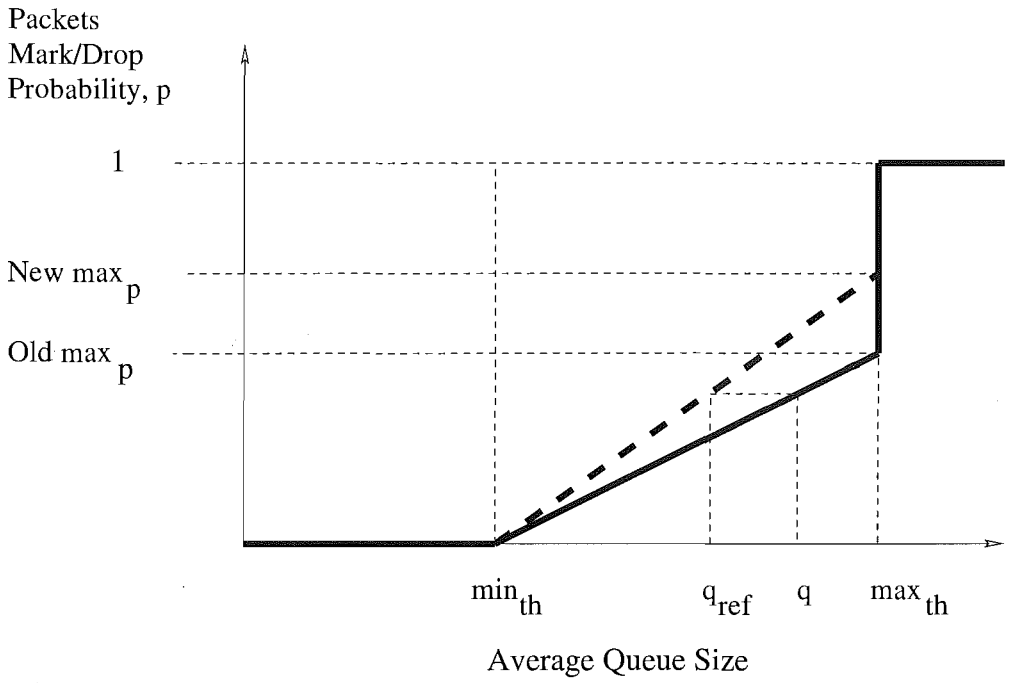

Figure 6.3 The working principle of Auto-Tuning RED algorithm.

For instance, it would settle to within 5% of its final value after three (dominant) time constants.

To estimate the system's dominant time constant, we use the "rule of thumb" that the damping ratio of a system's dominant closed loop poles is equal to PM/100. Hence, the real part of the pair of dominant closed loop poles can be approximated by the crossover frequency times $PM/100$. Examining the values of w in Table 6.1, clearly the worst case (smallest real part and hence largest time constant) occurs when $W = 64$. Taking $W = 20$ as being a more typical worst case, the estimate of the system's time constant, assuming $n = 10$, is $\frac{1}{(0.45)(0.162)} = 14$ round trip times. By this line of reasoning, the update interval should be three times as large, i.e. about 40 round trip times. This might seen excessive, especially in comparison to the update interval of 5 round trip times employed in [Floyd *et al.* 2001]. Indeed, despite the promise of near one step convergence to the correct max_p, a strong argument against such a long update interval is that the traffic level could change significantly during that time. Hence, in the following algorithm, we choose to make smaller increments more frequently.

6.5.3 Incremental Algorithm

The possibility of varying traffic level can be handled by taking a fraction ζ of the correction step, given in equation (6.28), of theoretical settling time of 40 round trip times. That is,

$$T_{update} = \zeta \cdot 40 \cdot R, \tag{6.29}$$

where $0 < \zeta < 1$. To speed up convergence, we multiply the increment in max_p by an acceleration factor $\epsilon \geq 1$. Hence the update formula is given by:

$$max_p \leftarrow max_p + \epsilon \cdot \zeta \cdot \Delta max_p, \tag{6.30}$$

where Δmax_p, is given by equation (6.28). Finally, to prevent max_p becoming too large or too small, as in [Floyd *et al.* 2001], we upper bound it to 0.5 and lower bound it to 0.01, which by equation (6.13) will permit a maximum congestion window size of about 23 packets. Thus we implement the following algorithm:

$$max_p \leftarrow Min\left[0.5, Max\left(max_p + \Delta max_p,\ 0.01\right)\right]. \tag{6.31}$$

The New adaptive RED algorithm can be summarized as follows:

Every *interval* seconds:
    If $(\bar{q} > q_{ref})$ and $max_p < 0.5$
    increase max_p:
    $max_p \leftarrow Min\,(0.5, max_p + \epsilon \cdot \zeta \cdot \Delta max_p)$;
elseif $(\bar{q} < q_{ref})$ and $max_p > 0.01$
    decrease max_p:
    $max_p \leftarrow Max\,(0.01, max_p + \epsilon \cdot \zeta \cdot \Delta max_p)$;
Variable parameters:
$\bar{q}$: EWMA queue size
Fixed parameters:
$interval = 4R$; $\epsilon = 1.5$; and $\zeta = 0.1$;

The above algorithm has been implemented in ns and EWMA queue plots are drawn to determine its convergence to the desired value of q_{ref}. From simulation graphs we can determine the most optimum values of ϵ and ζ. The recommended values of ϵ and ζ as 1.5/2.0 and 0.1 respectively. These simulation results of Auto-Tuning RED algorithm are presented in next Section for network scenario shown in Figure 6.9.

6.6 SIMULATION RESULTS

We performed simulations on a dumbbell topology shown in Figure 6.9, with one bottleneck link of capacity 1.5 Mbps, propagation delay of 20 ms and having a buffer size of 35 packets. The individual packet size is 1500 bytes, queue averaging weight w_q is chosen to be 0.0027, $min_{th} = 5$ packets, $max_{th} = 15$ packets. Simulations are run for 80 s until the steady state queue level is obtained.

The traffic on the bottleneck link consists of two long-lived TCP flows (which keep on sending data for whole of simulated time) and the reverse traffic consists of one long lived

TCP flow. The presence of data traffic on the reverse path introduces the acknowledgement compression and loss of acknowledgements, which introduces burstiness in the forward path. At time equal to 25 s, we introduce 20 new TCP flows which start one after other at the interval of 0.1 s. Each flow has a maximum window size of 20 packets. This is intended to model a change in the congestion level. In Figure 6.4 we obtain the same graphs as in [Floyd *et al.* 2001] for EWMA queue size and it serves as a bench mark for our simulation experiments. We plot the EWMA queue sizes only so that the queue convergence properties can be seen observed. To set the fixed parameters of our new algorithm, we first estimate the effective round trip time R and bandwidth-delay product $C \cdot R$ of the bottleneck link. Assuming $min_{th} = 5$ packets as employed in [Floyd *et al.* 2001] satisfies our guideline given in equation (6.25), q_{ref} is set to 10 packets and hence the average queuing delay at the bottleneck RED router is 80 ms. Adding the round trip propagation delay, we get R = 120 ms. We fix $\zeta = 0.1$, the default value for the new algorithm, and so $T_{update} = \zeta \cdot 4 \cdot R = 0.48$ s, which is very close to the value of 0.5 s employed in [Floyd *et al.* 2001]. We also find that the bandwidth-delay product $C \cdot R = 15$ packets. Hence the guideline in equation (6.25) gives $min_{th} \geq 2$ packets, so the value of 5 packets is suitable for our algorithm, too.

We ran simulations for different values of the acceleration factor ϵ. The EWMA queue plots for ϵ = 1, 1.5 and 2 are shown in Figures 6.5, 6.6 and 6.7 respectively. The variations in EWMA queue are given in Table 6.3 for entire simulated period (0 to 80 s) and in Table 6.4 for steady state period (40 s to 80 s). The variance of EWMA queue size is minimum for ϵ = 1.5 for entire simulated time and for ϵ = 2.0 for steady state period. The adaptation of max_p over entire period of simulated time to the change in traffic level is shown in Figure 6.8. The best trade-off between the speed of adaptation and magnitude of the fluctuations of max_p from its correct value appears to be between ϵ = 1.5 and ϵ = 2.0

The transient period in the EWMA queue size from 25 s to 40 s is shown in Figure 6.10 for adaptive RED algorithm in [Floyd *et al.* 2001], whereas for Auto-Tuning RED algorithm having ϵ = 1 , 1.5 and 2 it is shown in Figures 6.11, 6.12 and 6.13 respectively. The time, Δt, required by EWMA queue size during transient period (25 s to 40 s) to hit reference value (q_{ref} = 10 packets) for the first time by adaptive RED algorithm and Auto-Tuning RED algorithms is given in Table 6.2. The Auto-Tuning RED algorithm with ϵ = 2.0 takes minimum time of 10.41 s to hit q_{ref} for first time. Thus for entire length of simulated time the Auto-Tuning RED algorithm with ϵ = 1.5 shows better results where as for the the steady state period ϵ = 2.0 is better. Thus appropriate value of ϵ for Auto-Tuning RED algorithm can be selected according to requirements.

The Auto-Tuning RED algorithm as given in this Chapter has good convergence characteristics to reference value of queue, q_{ref}, and it can be employed in real RED routers due to its simplicity and robustness. The one shortcoming in this algorithm is that the q_{ref} depends upon the minimum and maximum thresholds, min_{th} and max_{th}, of RED algorithm as given in equation (6.12). Thus to change the value of q_{ref} we need to change min_{th} and max_{th}. Also the purpose of q_{ref} might be defeated in the case of adaptation of these thresholds with

changes in load as done in load-adaptive algorithm given in [Aweya *et al.* 2001]. Thus in the tuning-parameter space of RED algorithm, i.e. $\{min_{th}, max_{th}, max_p, w_q\}$, we have to keep the thresholds constant to employ the Auto-tuning principle based algorithms. In order to overcome the parameter dependence of Auto-Tuning RED algorithm, in the next Chapter we investigate the other types of congestion controllers, such as based on the Proportional Integral (PI) principle.

Type of RED	ϵ	Δt s
Adaptive [Floyd *et al.* 2001]	-	11.76
Auto-Tuning	1.0	14.10
Auto-Tuning	1.5	11.13
Auto-Tuning	2.0	10.41

Table 6.2 Comparison of transient period, Δt, from 25 s to 40 s for Adaptive RED and Auto-Tuning RED algorithms.

Type of RED	ϵ	Mean of EWMA	Variance of EWMA
Adaptive [Floyd *et al.* 2001]	-	11.21	9.09
Auto-Tuning	1.0	10.74	7.26
Auto-Tuning	1.5	10.60	6.37
Auto-Tuning	2.0	10.55	7.04

Table 6.3 Performance comparison of Adaptive RED and Auto-Tuning RED algorithms over the entire simulated time from 0 s to 80 s.

Type of RED	ϵ	Mean of EWMA	Variance of EWMA
Adaptive [Floyd *et al.* 2001]	-	11.59	5.78
Auto-Tuning	1.0	10.25	1.55
Auto-Tuning	1.5	10.02	1.24
Auto-Tuning	2.0	9.96	1.06

Table 6.4 Performance comparison of Adaptive RED and Auto-Tuning RED algorithms over steady state period from 40 s to 80 s.

6.7 CONCLUSIONS

The use of proper congestion control algorithm in a router is crucial for the overall performance of AQM. In the current literature, RED based AQM is most widely studied after Droptail based networks.

Although the work of most researchers has favoured the use of RED in implementing AQM but a few studies have also shown that RED does not much help to improve congestion control, see: [Bonald *et al.* 2000]. Also some weaknesses of RED have been pointed out in [Feng

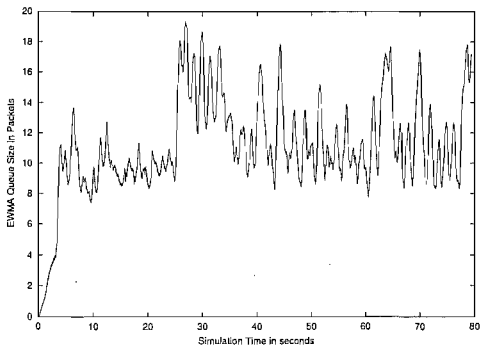

Figure 6.4 EWMA queue size of default adaptive RED.

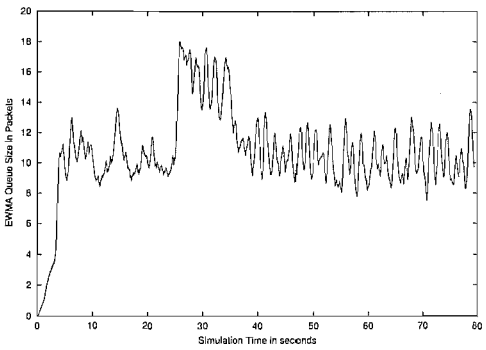

Figure 6.5 EWMA queue of Auto-Tuning RED with $\epsilon = 1.0$

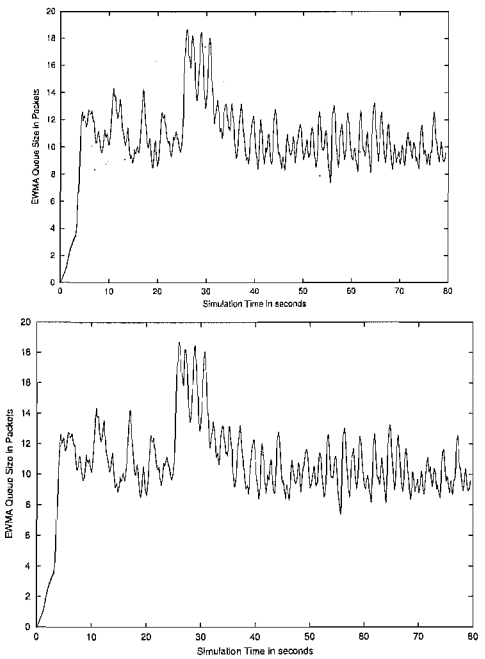

Figure 6.6 EWMA queue size of Auto-Tuning RED with $\epsilon = 1.5$

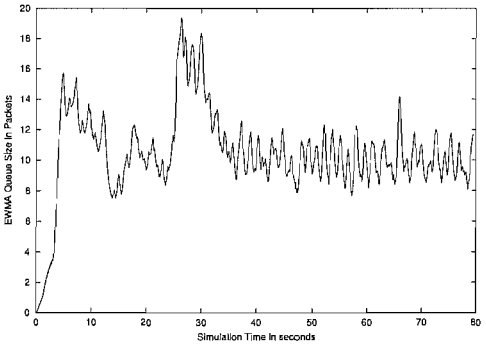

Figure 6.7 EWMA queue size of Auto-Tuning RED with $\epsilon = 2.0$

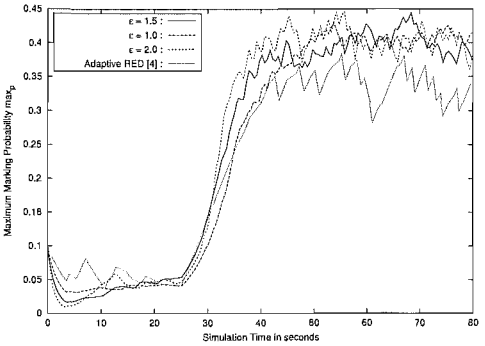

Figure 6.8 Variations of max_p for Adaptive RED and Auto-Tuning RED with different values of ϵ.

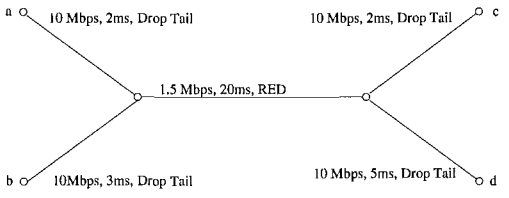

Figure 6.9 Network topology for simulation of Auto-Tuning RED.

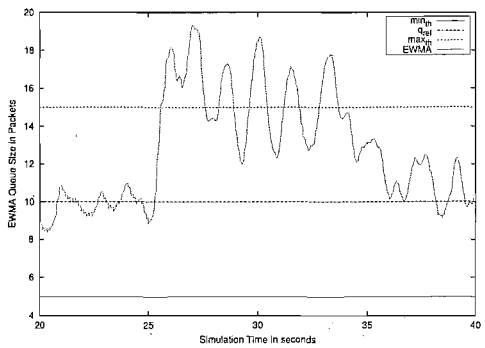

Figure 6.10 Transient in EWMA queue of default adaptive RED algorithm from 20 s to 40 s.

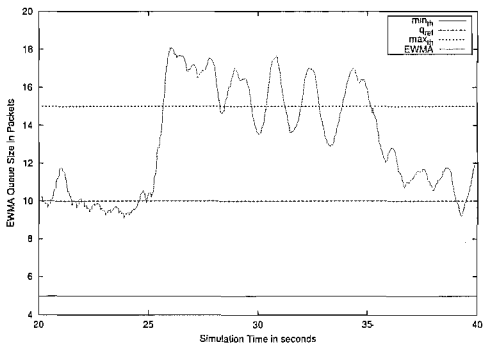

Figure 6.11 Transient in EWMA queue of Auto-Tuning RED ($\epsilon = 1.0$) from 20 s to 40 s.

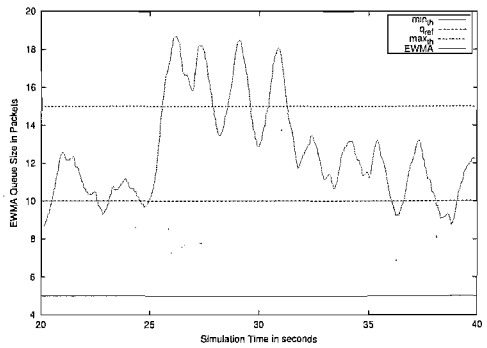

Figure 6.12 Transient in EWMA queue of Auto-Tuning RED ($\epsilon = 1.5$) from 20 s to 40 s.

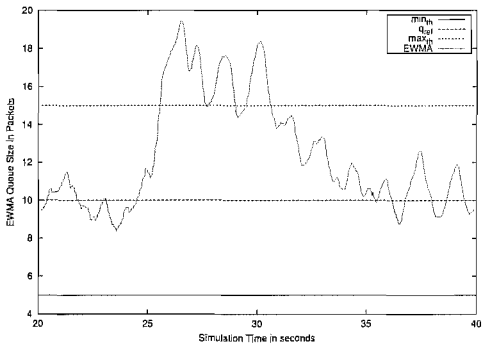

Figure 6.13 Transient in EWMA queue of Auto-Tuning RED ($\epsilon = 2.0$) from 20 s to 40 s.

et al. 1997], [Feng *et al.* 1999a] and [May *et al.* 1999]. Whereas the simulation experiments results and practical industrial experience, such as [Cisco 1998], of most researchers recommend the use of RED in implementing AQM. It appears that in order to get the full benefits of RED based AQM, proper tuning of all its parameters is very important.

Thus, in this Chapter after reviewing the strong and weak points of RED AQM in the available research literature, we have designed a new Auto-Tuning RED algorithm which is based on control-theoretic principles. It auto-tunes the key RED parameter max_p to adapt to the prevailing traffic conditions such as to keep the buffer queue length close to a reference value.

The setpoint target value for EWMA queue size, q_{ref}, in Auto-Tuning RED algorithm was, for convenience, chosen to be the arithmetic mean of the min_{th} and max_{th} parameters, for maintaining the operating point of router in the middle of dynamic range of queue size i.e. (max_{th}, min_{th}). Thus, in order to change q_{ref} we need to change thresholds which will affect the operating range of queue variations and will also require changes to the buffer size. However, the Auto-Tuning RED algorithm can be easily modified to decouple q_{ref} from the arithmetic mean of min_{th} and max_{th}. Thus, any limitation in Auto-Tuning RED arising due to changes in q_{ref} can be easily avoided.

Guidelines were given for choosing the algorithm's parameters, such as the thresholds max_{th} and min_{th} and the time between updates, so as to achieve a stable adaptive system with a phase margin of at least 45 deg. The new algorithm was tested using simulations by ns.

The results showed that it performed better than a recently proposed adaptive RED algorithm. Specifically, the new algorithm responded faster to traffic load changes, and it produced smaller deviations of the averaged queue length from the reference value.

More work is required in the direction of tying up the all parameters of RED to form a unified adaptive algorithm for AQM, which can give optimized performance over wide range of traffic variations. In the next Chapter we develop alternative algorithms for AQM, based on principles of feedback controllers, which keep queue length of router close to the desired target value.

Chapter 7

AQM AND FEEDBACK CONGESTION CONTROL ALGORITHMS

In the previous two chapters we introduced two new algorithms, Hybrid RED and Auto-Tuning RED, derived from basic RED [Floyd and Jacobson 1993], for improved congestion control in TCP/IP networks. Hybrid RED was proposed to improve the link utilization/ packet loss rate and Auto-Tuning RED was proposed for better queue control in scenarios of changing traffic load.

In this Chapter first we present a new design of the basic RED algorithm, called N-RED, based on control theoretical analysis, and its performance is compared with that of another control-theoretic design, called HO-RED, given in [Hollot *et al.* 2001a]. We generalise the Ziegler-Nichols principle [Ziegler and Nichols 1942] of designing feedback proportional controllers and thereby develop a new class of RED algorithm, whose performance is evaluated using control theory and ns simulations. We show that the new control-theoretic RED designs have better queue convergence characteristics and are more responsive to changes in traffic loads when compared to existing RED type algorithms.

Router algorithms based on the RED principle have certain limitations such as parameters sensitivity, dependance of EWMA on congestion level and slow response to changing traffic loads. The main reason for the slow response of RED AQM is the use of a low pass EWMA filter in packet marking/dropping decisions. Analytical study of RED in [Bonald *et al.* 2000] suggests that RED decreases the loss bias for bursty traffic at the expense of non-bursty traffic and RED has more consecutive packet marking/dropping than Droptail which might lead to global synchronization [Zhang and Clark 1990], [Floyd and Jacobson 1994]. Further it is expected that though RED decreases the mean delay of packets, [Floyd *et al.* 2001], but may also cause large delay variance, [May *et al.* 2000], resulting in increased jitter for interactive applications such as IP telephony [Zheng *et al.* 2001]. Thus, to get the desired benefits of RED based algorithms, we need very precise and proper tuning of all four RED parameters $\{min_{th}, max_{th}, max_p, w_q\}$, otherwise RED performance may deteriorate to Droptail or might be even worse. Hence, when looking at RED based algorithms, one will conclude that we need to find some other alternative algorithms which avoids the shortcomings of RED.

Recently Proportional (P) and Proportional-Integral (PI) principles based control algorithms have been proposed in lieu of RED-based algorithms for congestion control in computer net-

works and implementation of AQM [Hollot *et al.* 2001b]. However, the performance of P control algorithm necessitates large buffers in routers which is not practical, [Hollot *et al.* 2001b] and [Hollot *et al.* 2002]. The PI control algorithm in [Hollot *et al.* 2001b], termed HO-PI in this Chapter, is designed on the basis of heuristic techniques of classical feedback control theory, which involves placing a zero of PI transfer function on the TCP pole. However, it is found that the recommended settings of HO-PI lead to a poorly tuned router algorithm.

After developing N-RED algorithms in the first part of this Chapter, we present a new methodology for the design of PI-based feedback congestion control algorithms, by generalising the Ziegler-Nichols tuning rules, [Ziegler and Nichols 1942]. In order to improve the stability of the closed loop AQM system, we also develop a new Proportional-Integral-Derivative (PID) based feedback congestion control algorithm. It is tuned by Ziegler-Nichols tuning rules for PID controllers as given in [Ziegler and Nichols 1942] and its performance is evaluated through control theoretic techniques and ns simulations. This algorithm can be further generalized using the same methods as developed for PI-based feedback congestion control algorithms. Thus in this Chapter we give broad and generalized guidelines for development and designing of feedback control theory principles based congestion control algorithms for AQM routers supporting TCP/IP traffic in computer networks and the Internet.

The structure of this Chapter is as follows. This Chapter commences with a brief review in Section 7.1 of previous work done in the area. The feedback control model of AQM, used in this Chapter, is given in Section 7.2. The control theoretic design of an existing HO-RED algorithm is presented in Section 7.3. The simulation setup and network topology model/analysis, used in this Chapter, as well in the next Chapter 8, is presented in Section 7.3.2. The Ziegler-Nichols frequency response method and tuning rules are given in Section 7.4. In Section 7.5 we develop a new RED principle based control theoretic algorithm, N-RED. The PI principle based feedback congestion control algorithms are presented in Section 7.7. The development of PID principle based feedback congestion control algorithm and its Ziegler-Nicholas tuning rules based design is given in Section 7.8. The performance of feedback congestion control algorithms in the presence of non-responsive traffic flows is presented in Section 7.9. Finally conlusions are drawn in Section 7.10

7.1 REVIEW OF RELATED WORK

As described in Chapter 1, congestion is the result of a mismatch between network resources and the amount of admitted traffic. The problem of congestion avoidance in packet switched data networks has been recognized as a feedback control problem of matching the inputs to output of dynamical systems in [Jacobson 1988] and [Hass and Winters 1991]. The major goals of feedback-based congestion control algorithms in routers are to stabilize queue sizes by reducing oscillations and increasing the speed of response to sudden surges of network traffic.

A feedback based congestion control scheme for ATM based networks, to cope with traffic

surges that are shorter than network round trip delay, was proposed in [Hass and Winters 1991]. This scheme was not simulated throughly and does not seems to be suitable for end-to-end congestion control in IP based networks. In [Benmohamed and Meerkov 1993] an adaptive and a robust feedback-based congestion controller has been developed for the single congested node case. The congestion controller associated with each link computes the admission rate periodically and supplies this information to traffic sources which adjust their rates accordingly. These type of controllers are also not suitable for the present window based end-to-end setup using TCP/IP. Rate-based feed-back congestion controllers with H^{∞} design are developed in [Ozbay *et al.* 1998] for ATM networks. These feedback controllers are also not suitable for TCP/IP networks which follow an end-to-end congestion control paradigm. Using Smith's principle of classical control theory, a congestion control algorithm is developed in [Mascolo 1999] which is transformed into discrete time for both ATM and TCP/IP networks. It models the congestion control algorithm of TCP/IP by a Smith predictor, [Astrom and Wittenmark 1990], and shows that overflow of TCP receiver is avoided by this built in mechanism. However, this analysis lacks detailed modelling of TCP/IP networks.

Without using the Smith predictor term, a detailed and in-depth model of TCP/IP networks is developed in [Misra *et al.* 2000]. It uses Poisson counter-driven stochastic differential equations to model the sample path description of TCP/IP with a fluid approximation to data traffic. These stochastic differential equations are transformed into ordinary differential equations and a RED-based AQM is analyzed, simulated and also solved numerically. The transfer function of RED based AQM developed in [Misra *et al.* 2000] and [Hollot *et al.* 2001a] is used in this Chapter.

In [La 2001] the stability of a $(p, 1)$ proportionally fair and global optimization algorithm is studied and cascaded feedback-based controllers are designed to improve their transient and steady state behavior. These feedback-based congestion controllers (proportional, proportional-integral and proportional-integral-derivative) are implemented within the TCP protocol at edges of network i.e. at end user's sides. These controllers require end hosts to measure round trip time and queuing delay to compute proportional, integral and derivative based control actions in order to update the congestion window size of TCP. Thus, they require changes to the existing end user's TCP protocols which is quite difficult due to the huge size of the present Internet.

In [Hollot *et al.* 2000] and [Hollot *et al.* 2001a] the RED algorithm has been analyzed using feedback control theory and new guidelines have been developed for its tuning. Weaknesses of this algorithm are discussed in [Hollot *et al.* 2001b] and two new (Proportional and Proportional-Integral controllers) router-based feed-back control algorithms are developed which follow end-to-end congestion control principles. Later similar results were also presented in [Hollot *et al.* 2002] by carrying out analysis and ns-based simulations.

PD-RED algorithm has been proposed in [Sun *et al.* 2003a] and [Sun *et al.* 2003b]. In this algorithm the max_p parameter of RED algorithm, as given in Table 5.2, is varied by a Proportional-Derivative control law, [Franklin *et al.* 1994]. Its perfomance was investigated by

using ns-based simulations. The queue length behavior of PD-RED was also compared with that of Dynamic RED [Aweya *et al.* 2001], Adaptive RED [Floyd *et al.* 2001] and PI Controller [Hollot *et al.* 2001b]. It was shown that PD-RED can maintain its queue length around the target value under different traffic conditions and different round trip times. Also it was shown that PD-RED works well with bursty http connections and non-responsive UDP flows. However, the performance of this algorithm depends upon the proper choice of sampling time, proportional gain, derivative gain, filter gain and no-drop threshold.

Recently, a Proportional-Integral-Derivative based congestion control algorithm for AQM routers has been presented in [Ren and Lin 2003]. In this design of PID based congestion control algorithm, the number of unknown variables exceeds the number of equations which requires an arbitrary selection of plant frequency, ω_p (i.e. for constant $\alpha \in [0.5, 2]$ and phase cross over frequency ω_c the ω_p is defined as $\omega_p = \alpha \cdot \omega_c$). Also the algorithm has not been simulated thoroughly. Later in this Chapter we will present an improved PID based congestion control algorithm for AQM.

7.2 AQM AS A FEEDBACK CONTROL SYSTEM

In Chapter 6 we employed the fluid based model of AQM [Hollot *et al.* 2001a] to design an Auto-tuning RED algorithm. In order to develop new feedback congestion control algorithms we generalize the block diagram shown in Figure 6.1 by introducing a general control algorithm modelled by transfer function $C(s)$. Hence, the overall feedback control loop of AQM can be expressed as in Figure 7.1 below:

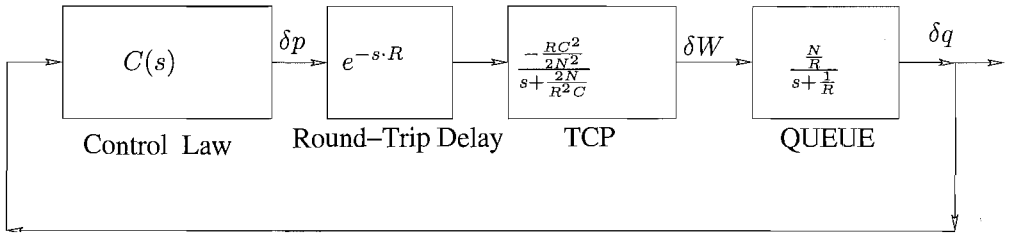

Figure 7.1 The feedback control based closed loop model of AQM.

The above Figure cascades the linearised transfer functions of round trip time delay in sending of a packet and receiving the corresponding ACK, TCP dynamics (excluding packets retransmissions due timeouts) and the associated router's queue size behavior, as already given in equations 5.3, 5.4 and 5.5 respectively. It effectively captures the dynamics of the implicit feedback mechanism built in TCP (by sequential ACKs) [Postel 1981] as well as the additive increase and multiplicative decrease policy of TCP [Chiu and Jain 1989] during congestion avoidance and control phases. The problem addressed in this Chapter is to develop congestion control algorithm, $C(s)$, to meet the objectives of AQM as stated earlier in Subsection 5.2.1 and further elaborated in later sections of this Chapter. In the next Section we present and evaluate

an existing control-theoretic design of RED algorithm, Section 5.14.1, as developed in [Hollot *et al.* 2001a], herein named as HO-RED algorithm.

7.3 HO-RED CONGESTION CONTROL ALGORITHM

7.3.1 Design of HO-RED Algorithm

The expression for EWMA queue size of RED algorithm given in equation (5.11) and linear mark/drop probability functions given in equation (5.9) has been transformed into the following frequency domain transfer function in [Misra *et al.* 2000]:

$$C_{red}(s) = L_{red} \cdot \frac{K_{red}}{s + K_{red}}. \tag{7.1}$$

The transfer function of RED, $C_{red}(s)$, is decomposed into two parts i.e. EWMA filter and gain as shown below in Figure 7.2. Using $C_{red}(s)$ and AQM transfer function as given by equation

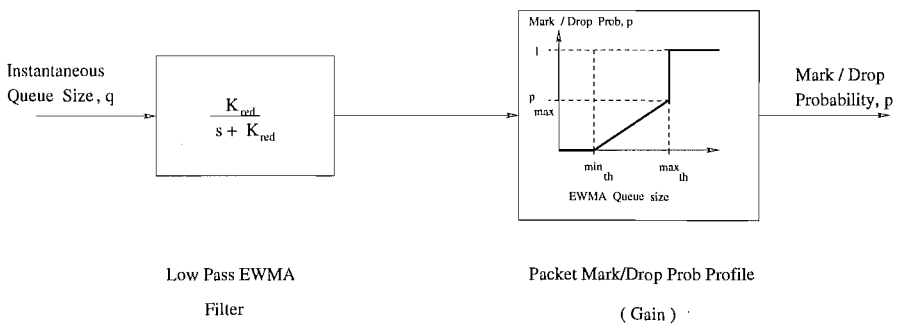

Figure 7.2 Transfer function of RED algorithm.

(5.6) in Subsection 5.3.2 the following design equation has been developed in [Hollot *et al.* 2000] and [Hollot *et al.* 2001a]:

$$\frac{L_{red} \cdot (RC)^3}{2N^2} \leq \sqrt{\frac{\omega_g^2}{K_{red}^2} + 1}, \tag{7.2}$$

where ω_g, is given by:

$$\omega_g = 0.1min\left(\frac{2N}{R^2C}, \frac{1}{R}\right). \tag{7.3}$$

HO-RED designed by equation (7.2) has a Gain Margin (GM) $\geq 5\pi \approx 16$ and a Phase Margin (PM) $\geq$ 85 degrees [Hollot *et al.* 2001a]. The closed loop response of response of HO-RED AQM is commensurately slow due to the low value of ω_g, as given in equation (7.3), compared to the bandwidth of either the queue or the TCP dynamics as given by equations (5.5) and (5.4) respectively. It can be attributed to the choice of excessive gain and phase margins (typically the values of GM and PM should be between 2 to 5 and 30 to 60 degrees, respectively [Astrom and Hagglund 1995]). The excessive GM and PM leads to overdamped transient response of

system, [Franklin *et al.* 1994]. This slow response can be improved by using higher values of ω_g which would reduce the stability margins to more practical values. Thus, yielding a less damped transient response.

In [Hollot *et al.* 2001a] the performance of HO-RED has been compared with the default RED algorithm described in Section 5.14 with parameters as given in Table 5.2 of Chapter 5. It has been found that the default RED algorithm creates underutilization of link capacity and produce large oscillations in instantaneous queue size leading to considerable jitter in round-trip times of packets. These drawbacks in the default RED algorithm have been partly overcome by the HO-RED algorithm. In the next Subsection we present the simulation setup and control theoretic analysis of network topology which will be used to simulate the HO-RED algorithm. Also, for the purpose of uniformity in performance comparison of different feedback congestion control algorithms for AQM, developed in this Chapter and in the next, we use the same simulation setup.

7.3.2 Simulation Setup

We perform ns-based simulations on the network shown in Figure 7.3, which consists of 60 ftp traffic sources (node A to node C) and 180 http traffic sources (node B to node C), sending data on links of 15 Mbps capacity with propagation delays varying from 0 to 63 ms and 0 to 90 ms respectively. All simulations were done for period of 250 s. A set of 40 ftp sources start at 0 s and keep on sending data uptill end of simulation. Another set of 20 ftp sources start at 0 s, stop at 100 s and then restart at 140 s and send data until 250 s.

The data senders from node A to node C are using TCP Reno. The links from A to C and B to C are using Droptail queue management, while the link from C to D employs the newly designed feedback based congestion control algorithms. The capacity of the bottleneck link is 15 Mbps. TCP packet size is assumed to be 1000 bytes or 8000 bits and the bottleneck link capacity is 3750 packets/s. The maximum round trip time of the 60 TCP flows is 246 ms.

The buffer sizes of access links are sufficiently large to avoid packet loss due to buffer overflow. After simulations we plot the instantaneous queue size, EWMA queue size, marking/dropping probability and its EWMA value with different weights. Hence, in these plots the units on x-axis is simulation time in seconds and units on y-axis are queue size in packets or marking/dropping probability of packets. The queue size is measured in packets unless otherwise stated. In the next Subsection we determine the transfer function of the simulated network, and determine its stability margins using both Bode plots and Nyquist plots. This transfer function model will be used later to develop new congestion control algorithms for the implementation of active queue management.

7.3.2.1 Network Model / Analysis

For the simulated network topology in Section 7.3.2 we have link capacity $C = 3750$ packets/s, round trip time $R = 0.246$ s, and number of connections $N = 60$. Substituting values of C, R and N in equation (5.6) gives the following transfer function model of the network:

$$P(s) = \frac{117187.5\ e^{-0.246s}}{(s + 0.528)(s + 4.065)}. \tag{7.4}$$

Applying a second-order Pade approximation [Franklin *et al.* 1994] to the exponential term in equation (7.4) we get:

$$e^{-0.246s} = \frac{s^2 - 24.39s + 198.29}{s^2 + 24.39s + 198.29}. \tag{7.5}$$

Thus, the overall transfer function of the system is approximated by:

$$P(s) = \left(\frac{117187.5}{(s + 0.528)(s + 4.065)}\right) \cdot \left(\frac{s^2 - 24.39s + 198.29}{s^2 + 24.39s + 198.29}\right). \tag{7.6}$$

We can also write equation (7.6) as:

$$P(s) = \left(\frac{117187.5}{(s + 0.528)(s + 4.065)}\right) \cdot \left(\frac{\{s + (-12.19 - j7.04)\}\{s + (-12.19 + j1.04)\}}{\{s + (12.19 - j7.04)\}\{s + (12.19 + j1.04)\}}\right). \tag{7.7}$$

The Bode plots [Bode 1940], [Franklin *et al.* 1994] of equation (7.7) are shown in Figure 7.4

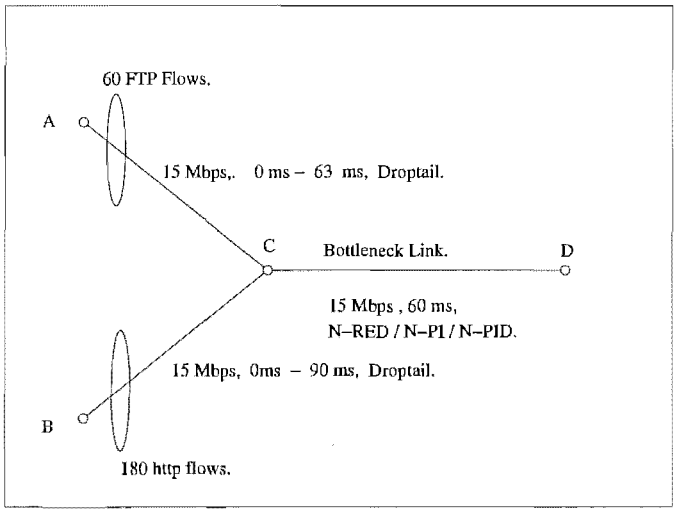

Figure 7.3 Network topology to simulate the feedback congestion control algorithms.

and the Nyquist plot [Nyquist 1932], [Franklin *et al.* 1994] is shown in Figure 7.5. For the uncompensated plant in equation (7.7) the Gain Margin (GM) is -74.06 db at 3.85 rad/s and Phase Margin (PM) is -351.07 degrees at 342.31 rad/s. The GM is related to ultimate gain K_u, [Astrom and Hagglund 1995], by $20 \log K_u = GM$, which gives us $K_u = 1.862 \times 10^{-4}$. The

frequency of GM is the ultimate frequency ω_u, which gives us the ultimate time $T_u = \frac{2\pi}{\omega_u}$. For the network under consideration $\omega_u = 3.856$ rad/s which gives $T_u = 2\pi/3.856 = 1.629$ s.

The ultimate point of the network is $(-1/K_u,\ 0) = (-5370.56,\ 0)$, which can be shifted at an arbitrary position in the complex s-plane (thus introducing phase lag or lead), [Astrom and Hagglund 1995], by different congestion control algorithms developed through the frequency response method given in Subsection 7.4.1. In Figure 7.5 we show the ultimate point on the Nyquist plot for the simulated network topology shown in Figure 7.3. For the general design of PI and PID controllers and design based on Ziegler-Nichols guidelines as given in Table 7.1 the shifting of ultimate point is derived in Appendix B. In the next Subsection first we use MATLAB

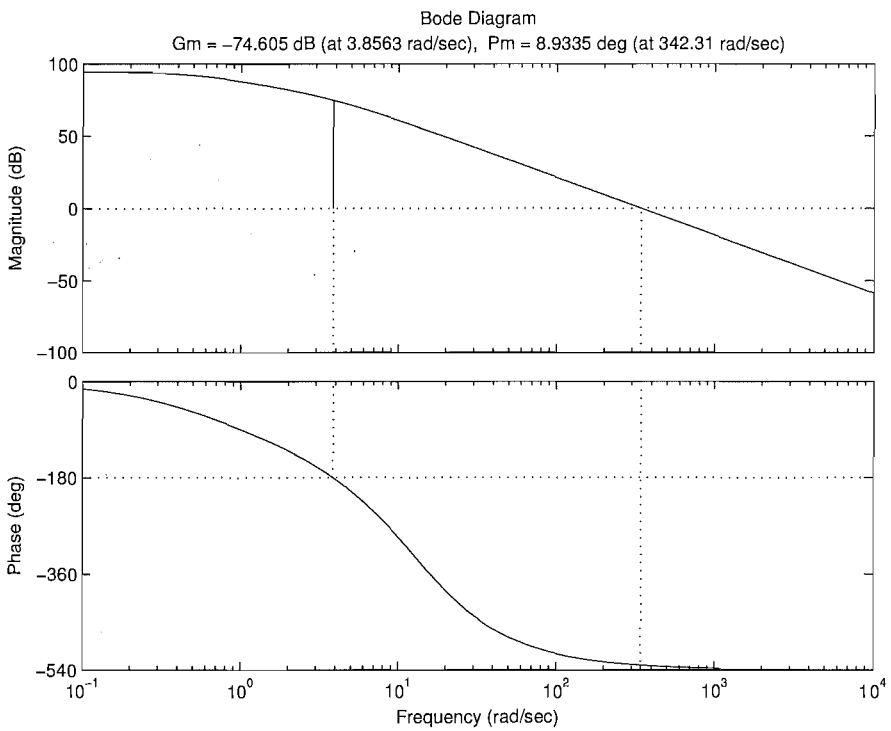

Figure 7.4 Gain and Phase margins of transfer function model of network topology used for simulation of feedback congestion control algorithms.

to determine the step response and stability margins (through the Bode plots) of closed loop control system formed by HO-RED AQM. Later we perform ns-based simulations with HO-RED algorithm used with the bottleneck link (node C to node D) in the network topology shown in Figure 7.3.

7.3.3 Simulations of HO-RED Algorithm

For the network in Figure 7.3 of Section 7.3.2 we have $C = 3750$ packets/second, $N = 60$ and $R = 0.246$ s. Substitution of these values in equation (7.2) and equation (7.3) gives $L_{red} = 1.86 \times 10^{-4}$ and $\omega_g = 0.053$ rad/s respectively and for $K = 0.005$ we get queue weight $w_q = 1.33 \times 10^{-6}$ from equation (6.8). Substituting values in equation (7.1) we get the following

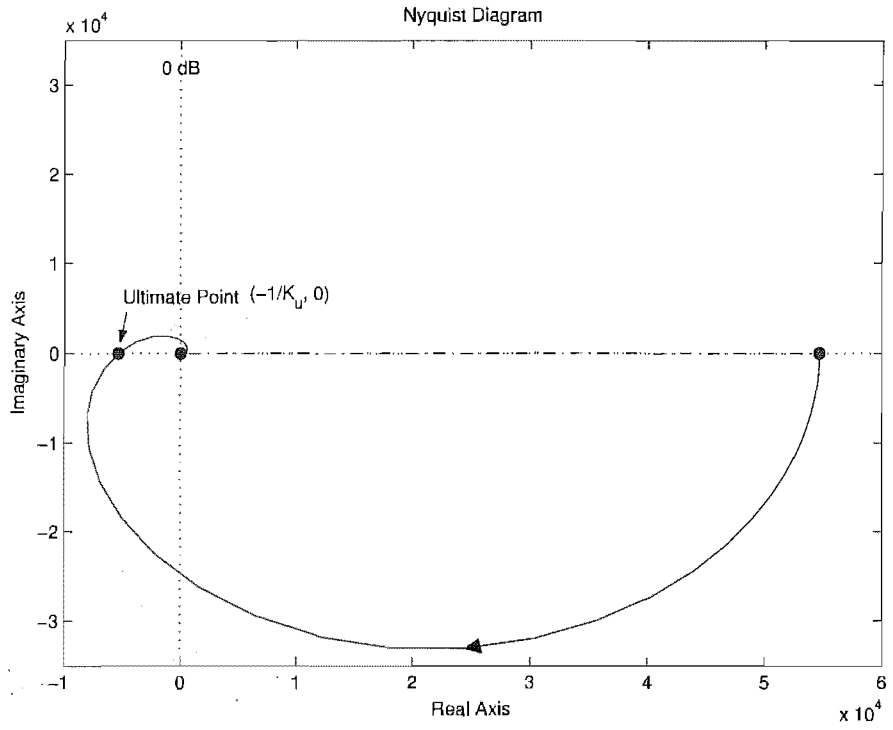

Figure 7.5 Nyquist plot of transfer function model of network topology used for simulation of feedback congestion control algorithms.

transfer function of Hollot's RED design [Hollot *et al.* 2000]:

$$C_{HO-RED}(s) = \frac{9.3 \times 10^{-7}}{s + 0.005}. \tag{7.8}$$

Using $C_{HO-RED}(s)$ as controller in Figure 7.1 gives following loop gain:

$$L(s) \equiv P(s) \cdot C_{HO-RED}(s) = \left(\frac{117187.5e^{-0.246s}}{(s + 0.528)(s + 4.065)}\right) \cdot \left(\frac{9.3 \times 10^{-7}}{s + 0.005}\right). \tag{7.9}$$

Bode plots of equation (7.9) gives GM of 32.82 db at 1.004 rad/s and PM of 88.81 deg at 0.0502 rad/s as shown in Figure 7.7. The step response of closed loop AQM feedback loop in Figure 7.1 is plotted in Figure 7.6, which shows that $C_{HO-RED}(s)$ control algorithm is overdamped and quite sluggish with settling time of 63.3 s and rise time of 34.5. It was predicted by the control theoretic arguments given in Subsection 7.3.1. The slow resposne of HO-RED is a very critical limitation during periods of changing traffic. In real traffic many TCP and http connections have short lifetimes, less then settling time of $C_{HO-RED}(s)$ control algorithm. Thus, in the case of varying and bursty traffic, AQM system with HO-RED will respond too slowly causing an increase in queuing delay and leading to congestion.

The same effects can be seen both in the instantaneous and EWMA queue size plots as shown in Figure 7.8. The packet marking/dropping probability plot is also given in Figure 7.9

which shows the low value at higher instantaneous queue size due to smaller value of EWMA queue size for $w_q = 1.33 \times 10^{-6}$. Thus, these simulation results suggests the need to have

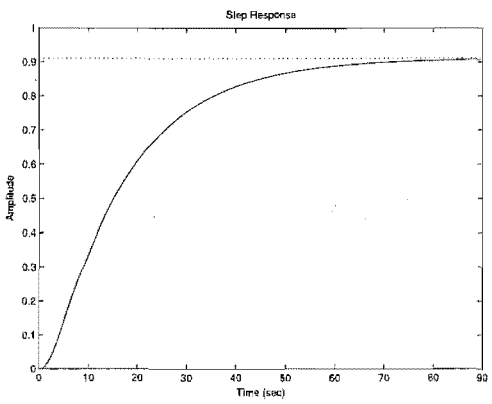

Figure 7.6 Step response of HO-RED algorithm

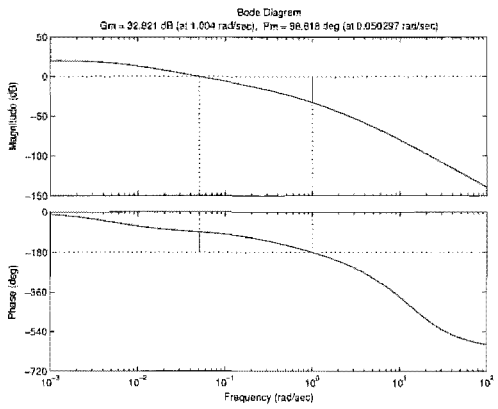

Figure 7.7 Stability margins of HO-RED algorithm

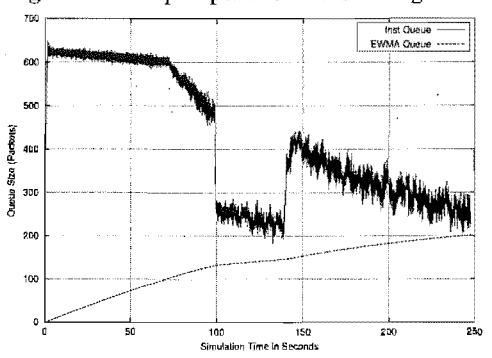

Figure 7.8 Instantaneous/EWMA queue of HO-RED.

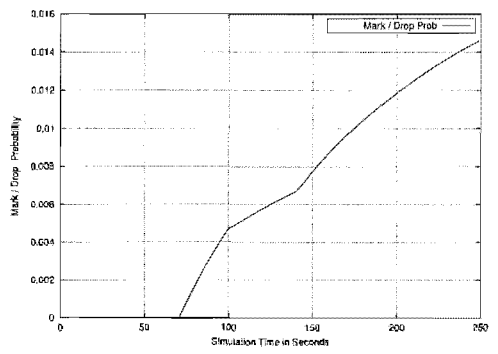

Figure 7.9 Mark/Drop probability of HO-RED algorithm.

an alternative and better control-theoretic design of RED algorithm which can overcome the weakness's of HO-RED algorithm. Therefore in the proceeding Subsection we present a new design of RED algorithm, called N-RED algorithm, using the frequency response method given in Subsection 7.4.1 of this Chapter.

7.4 ZIEGLER-NICHOLS DESIGN METHODS

Two classical methods for determining the parameters of Proportional (P), Proportional Integral (PI) and Proportional Integral Derivative controllers are given in [Ziegler and Nichols 1942]. These are the *step response method* and the *frequency response method* or *continuous cycling method.*

In the step response method the decay ratio (ratio between two consecutive maxima of error for a step change in setpoint) is made equal to about 0.25, making transients to decay to a quarter of their magnitude after one period of oscillation. It requires the determination of the step response (or process reaction curve) of the control system from either transient response experimental data or dynamic simulations [Franklin *et al.* 1994] and [Astrom and Hagglund 1995]. The

transient response experiment requires that the system be at rest before the step input is applied and there are no measurement errors [Astrom and Hagglund 1995]. Both of these conditions are difficult to be satisfied in practice, hence the utility of this method is limited. Similarly, in practice, it is difficult to perform dynamic simulations for complex plants such as that produced by computer networks.

In the alternative frequency response or continuous cycling method, the parameters are evaluated at the limit of stability, which is the point on s-plane where Nyquist curve cuts the negative real axis, i.e. $(\frac{-1}{K_u}, 0)$. The gain K_u, which brings system to the stability limit, is known as the ultimate gain. The corresponding frequency ω_u and time period $T_u = \frac{\omega_u}{2\pi}$ are called the ultimate frequency and ultimate time period, respectively [Astrom and Hagglund 1995]. We have employed the frequency response method, explained in the next Subsection, to develop new feedback based congestion control algorithms for active queue management.

7.4.1 Frequency Response Method

In order to determine the ultimate gain and ultimate frequency (ultimate point) on Nyquist curve, Ziegler and Nichols have provided frequency response method in [Ziegler and Nichols 1942]. This method is based on the fact that feedback systems can be made unstable under proportional control by choosing sufficiently high gain. In order to explain the frequency response method let us consider a proportional feedback control system as shown in Figure 7.10. The gain of above

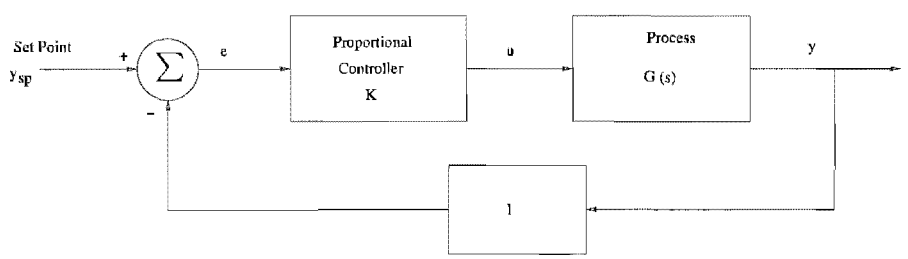

Figure 7.10 A proportional feedback control loop for a process $G(s)$.

system, K, is adjusted until the process is at the stability boundary. For setpoint $y_{sp} = 0$, the error signal, $e = y_{sp} - y$, reduces to $e = -y$. Control signal u and process output y are related by $u = -K \cdot y$. The overall loop gain must be unity to maintain oscillations, thus we have $K_u \cdot G(j\omega_u) = -1$ or $|K_u \cdot G(j\omega_u)| = 1$. This relation forms a basis of our development and design of router based congestion control algorithms in the proceeding Sections of this Chapter. In the next Subsection we briefly review the existing guidelines for determining the parameters of feedback controllers obtained through the frequency response method as explained in this Subsection.

7.4.1.1 Parameters of P, PI and PID Controllers

The rules for tuning the parameters of feedback controllers (P, PI and PID), first developed in [Ziegler and Nichols 1942] by using the frequency response method, are still widely used in the process control industry.

The tunable parameters of P, PI and PID controllers are the gain K_c, integral time T_i and derivative time T_d, [Franklin *et al.* 1994]; where K_c acts as the gain factor or multiplier, $\frac{1}{T_i}$ is a measure of speed of response or reset factor of the integrator and T_d determines the period of ahead estimate of control error (or prediction horizon) [Astrom and Hagglund 1995]. Furthermore, increasing K_c and $\frac{1}{T_i}$ tends to reduce system errors but also decreases system stability whereas increasing T_d tends to improve stability [Franklin *et al.* 1994]. The typical values of these parameters (K_c, T_i and T_d) as recommended by Ziegler-Nichols frequency response method are given in Table 7.1, along with an estimated closed loop period T_p. In the next

Controller	Parameters			
	K_c	T_i	T_d	T_p
P	$0.50K_u$	-	-	T_u
PI	$0.45K_u$	$0.85T_u$	-	$1.40T_u$
PID	$0.60K_u$	$0.50T_u$	$0.125T_u$	$0.85T_u$

Table 7.1 Ziegler-Nichols Continuous Cycling Method Tuning Rules.

Section we develop a new RED principle based control theoretic congestion control algorithm, N-RED, by employing the frequency response method just described.

7.5 N-RED CONGESTION CONTROL ALGORITHM

In this Subsection we modify the basic RED algorithm as given in the Section 5.14 of Chapter 5 by using the frequency response method, described in Subsection 7.4.1 of this Chapter, to develop the new N-RED algorithm. First we develop a general design of the N-RED algorithm. Later we consider two particular cases of general design and evaluate their performance using MATLAB and ns-based simulations.

7.5.1 General Design of N-RED Algorithms

Using the AQM plant transfer function equation (5.6) given in the Section 5.3.2 of Chapter 5 and the RED algorithm model given by equation (6.6) in the Section 6.2 of Chapter 6 we have the following overall open-loop transfer function:

$$G(s) = \left(\frac{\frac{RC^2}{2N^2}}{s + \frac{2N}{R^2C}} \cdot \frac{\frac{N}{R}}{s + \frac{1}{R}} \cdot \frac{K_{red}}{s + K_{red}} \right) \cdot e^{-sR}. \tag{7.10}$$

Thus, we have replaced control law $C(s)$ in Figure 7.1 by RED transfer function $C_{red}(s)$ as given in equation (6.6). Next, substituting s by $j\omega$ in equation (7.10) we get the following relation:

$$G(j\omega) = \left(\frac{\frac{RC^2}{2N^2}}{j\omega + \frac{2N}{R^2C}} \cdot \frac{\frac{N}{R}}{j\omega + \frac{1}{R}} \cdot \frac{K_{red}}{j\omega + K_{red}} \right) \cdot e^{-j\omega R}. \tag{7.11}$$

Taking the amplitude of equation (7.11) we get:

$$|G(j\omega)| = \left(\frac{\frac{RC^2}{2N^2}}{\sqrt{\omega^2 + \left(\frac{2N}{R^2C}\right)^2}} \cdot \frac{\frac{N}{R}}{\sqrt{\omega^2 + \left(\frac{1}{R}\right)^2}} \cdot \frac{K_{red}}{\sqrt{\omega^2 + K_{red}^2}} \right) \cdot e^{-j\omega R}. \tag{7.12}$$

The frequency response method, as given in the Section 7.4.1 of this Chapter, requires that:

$$|K_u \cdot G(j\omega_u)| = 1, \tag{7.13}$$

where ω_u is ultimate frequency and K_u is ultimate gain of a general closed loop feedback control system. Thus, for RED based AQM we obtain the following expression for ultimate gain, $K_{u_{rd}}$, at ultimate frequency $\omega_{u_{rd}}$, i.e. $\omega = \omega_{u_{rd}}$, by using equations (7.12) and (7.13):

$$K_{u_{rd}} = \frac{2N}{C^2 K_{rd}} \cdot \sqrt{\omega_{u_{rd}}^2 + \left(\frac{2N}{R^2C}\right)^2} \cdot \sqrt{\omega_{u_{rd}}^2 + \frac{1}{R^2}} \cdot \sqrt{\omega_{u_{rd}}^2 + K_{red}^2}. \tag{7.14}$$

The Nyquist stability criterion given in [Nyquist 1932] and [Franklin *et al.* 1994] gives the phase angle relation for the stability of open loop transfer function as $\angle G(j\omega) = -\pi$. Thus, computing the phase angle on both sides of equation (7.11) we get the following expression:

$$\omega \cdot R + \tan^{-1}\left(\omega \cdot \frac{R^2C}{2N}\right) + \tan^{-1}(\omega \cdot R) + \tan^{-1}\left(\frac{\omega}{K_{red}}\right) = \pi. \tag{7.15}$$

The expressions for K_{rd} and w_q, already given as equations (6.8) and (6.16) respectively, are:

$$K_{red} = -C \cdot \ln(1 - w_q), \tag{7.16}$$

$$w_q = \left(1 - e^{\frac{-1}{n \cdot RC}}\right). \tag{7.17}$$

Therefore, from equations (7.16) and (7.17) we can express K_{red} as follows:

$$K_{red} = \frac{1}{nR}. \tag{7.18}$$

At ultimate frequency, $\omega_{u_{rd}}$, substitute equation (7.18) into equation (7.15) to get the following expression:

$$\omega_{u_{rd}} \cdot R + \tan^{-1}\left(\omega_{u_{rd}} \cdot \frac{R^2C}{2N}\right) + \tan^{-1}\left(\omega_{u_{rd}} \cdot R\right) + \tan^{-1}\left(\omega_{u_{rd}} \cdot nR\right) = \pi. \quad (7.19)$$

As proved in Subsection 6.4.2 of Chapter 6 the stable operation of the RED algorithm can be achieved by setting $n = 10$. Thus, we can write equation (7.19) as:

$$\omega_{c_{rd}} + \tan^{-1}\left(\frac{\omega_{c_{rd}}W}{2}\right) + \tan^{-1}\left(\omega_{c_{rd}}\right) + \tan^{-1}\left(10\omega_{c_{rd}}\right) = \pi, \quad (7.20)$$

where $\omega_{c_{rd}} = \omega_{u_{rd}} \cdot R$ and window size is $W = \frac{RC}{N}$. The equation (7.20) is a transcendental equation which is difficult to solve exactly for $\omega_{c_{rd}}$. Thus, we need to use numerical analysis techniques such as least square curve fitting to get an approximate expression for $\omega_{c_{rd}}$.

The values of $\omega_{c_{rd}}$ obtained by substituting different values of congestion window, W, into equation (7.20) are given in Table 7.2. Since the standard TCP implementations today limit the maximum window size to 64 Kbytes, [Stevens 1994], our selected range of W, (2 to 128 packets), covers all of the practical scenarios. These values are plotted in Figures 7.11 and 7.12

W	$1/W$	$\omega_{c_{rd}} = \omega_{u_{rd}} \cdot R$	$\hat{\omega}_{c_{rd}}$	%Error=$\frac{\omega_{c_{rd}} - \hat{\omega}_{c_{rd}}}{\omega_{c_{rd}}} \times 100$
2	0.5	0.620	0.654	−5.48
4	0.25	0.506	0.460	+9.09
8	0.125	0.406	0.363	+10.59
16	0.0625	0.332	0.315	+5.12
32	0.03125	0.282	0.290	−2.83
64	0.015625	0.252	0.278	−10.31
128	0.0078125	0.235	0.272	−15.74

Table 7.2 Data for least square curve fitting for $\omega_{c_{rd}}$ of transfer function model of RED based AQM.

as $\omega_{c_{rd}}$ versus congestion window, W, and reciprocal of congestion window, $1/W$, respectively. The variations of $\omega_{c_{rd}}$ with W has a drooping characteristics which is not captured by lower order curve fitting equations. Whereas $\omega_{c_{rd}}$ varies almost linearly with $1/W$ as shown in Figure 7.12. Thus, we perform least square linear curve fitting for the values plotted in Figure 7.12. In general the least square linear curve fit for Figure 7.12 can be expressed as the following equation:

$$\omega_{c_{rd}} = \frac{A_{zr}}{W} + B_{zr}, \quad (7.21)$$

where A_{zr} and B_{zr} are constants. In order to compute the values of constants A_{zr} and B_{zr} in equation (7.21) we use the software package MAPLE [Maplesoft 1998]. Thereby we get the

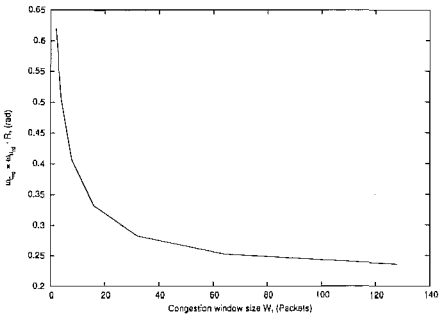

Figure 7.11 Variations in $\omega_{c_{rd}}$ with changes in congestion window, W, for AQM based on RED algorithm.

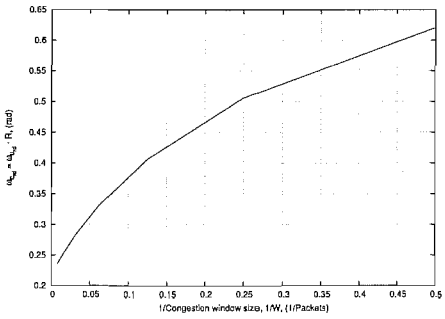

Figure 7.12 Variations in $\omega_{c_{rd}}$ with changes in reciprocal of congestion window, $1/W$, for AQM based on RED algorithm.

following equation for the graph given in Figure 7.12:

$$\omega_{c_{rd}} = \frac{0.7745}{W} + 0.2667. \tag{7.22}$$

The above relation can be denormalised to:

$$\omega_{u_{rd}} = \frac{0.7745}{WR} + \frac{0.2667}{R}. \tag{7.23}$$

Equation (7.23) gives an expression for ultimate (corner) frequency $\omega_{u_{rd}}$ in terms of window size W and round trip time R. In order to get an expression for ultimate gain K_u in terms of congestion window W and round trip time R, we substitute equation (7.23) into equation (7.14) to get the following expression:

$$K_{u_{rd}} = \frac{2N}{C^2 K_{red}} \cdot \sqrt{\left(\frac{0.7745}{WR} + \frac{0.2667}{R}\right)^2 + \left(\frac{2N}{R^2C}\right)^2} \cdot \sqrt{\left(\frac{0.7745}{WR} + \frac{0.2667}{R}\right)^2 + \frac{1}{R^2}} \cdot \sqrt{\left(\frac{0.7745}{WR} + \frac{0.2667}{R}\right)^2 + K_{red}^2}. \tag{7.24}$$

After substituting $W = \frac{RC}{N}$ and equation (7.18) into equation (7.24) and further simplifying we get:

$$K_{u_{rd}} = \frac{2}{W} \cdot \frac{n}{RC} \sqrt{\left(\frac{0.7745}{W} + 0.2667\right)^2 + \left(\frac{2}{W}\right)^2} \cdot \sqrt{\left(\frac{0.7745}{W} + 0.2667\right)^2 + 1} \cdot \sqrt{\left(\frac{0.7745}{W} + 0.2667\right)^2 + \frac{1}{n^2}}. \tag{7.25}$$

Substituting the value of constant $n = 10$, and bandwidth delay product $RC = (0.246)(3750) =$

922.5 packets into equation (7.25), and further simplifying we get:

$$K_{u_{rd}} = \frac{21.68 \times 10^{-3}}{W} \cdot \sqrt{\left(\frac{4.59}{W^2} + \frac{0.413}{W} + 0.0711\right)} \cdot \sqrt{\left(\frac{0.59}{W^2} + \frac{0.413}{W} + 1.0711\right)} \cdot \sqrt{\left(\frac{0.59}{W^2} + \frac{0.413}{W} + 0.0811\right)}. \tag{7.26}$$

As discussed in Subsection 7.3.2.1, plant transfer function $P(s)$ gives us ultimate gain $K_u = 1.862 \times 10^{-4}$, which after substitution in equation (7.26) and solving gives $W = 14.35$ packets. Similarly we can compute K_u for different values of W as given in Table 7.3. Now in order to

W	2	4	8	16	32	64	128
$K_{u_{rd}}$	0.010189	0.001905	0.000479	0.0001591	0.00006454	0.00002921	0.000013927

Table 7.3 Variations in ultimate gain, $K_{u_{rd}}$, with different values of congestion window, W (packets), for constant $n = 10$ and bandwidth delay product $RC = 922.5$ (packets).

get a proportional control system with a gain margin of γ_{rd}, we set $K_{c_{rd}} = \frac{1}{\gamma_{rd}} \cdot K_{u_{rd}}$, where $\gamma_{rd} > 1$. Then from equation (7.26) we get the following expression:

$$K_{c_{rd}} = \frac{1}{\gamma_{rd}} \cdot \left(\frac{21.68 \times 10^{-3}}{W}\right) \cdot \sqrt{\left(\frac{4.59}{W^2} + \frac{0.413}{W} + 0.0711\right)} \cdot \sqrt{\left(\frac{0.59}{W^2} + \frac{0.413}{W} + 1.0711\right)} \cdot \sqrt{\left(\frac{0.59}{W^2} + \frac{0.413}{W} + 0.0811\right)}. \tag{7.27}$$

Hence for a given set of values of max_{th}, min_{th} of the RED algorithm, already described in the Section 5.14 of Chapter 5, and required gain margin γ_{rd}, we can design the feedback congestion control algorithm N-RED by computing the values of $K_{c_{rd}}$, max_p and w_q through the following three step procedure:

- For a given window size W, we obtain control algorithm gain $K_{c_{rd}}$ from equation (7.27).
- Equating L_{red}, given in equation (6.6) of Subsection 6.2, to the controller gain $K_{c_{rd}}$ giving the following expression for maximum mark/drop probability max_p:

$$max_p = K_{c_{rd}} \cdot (max_{th} - min_{th}). \tag{7.28}$$

- Determine queue weight w_q by using $K_{red} = \frac{1}{nR}$ and $K_{red} = -C \cdot \ln(1 - w_q)$ as given in equation (6.8) of Subsection 6.2.

The transfer function of RED algorithm as given in equation (6.6) will be modified by above design procedure to give the following transfer function of N-RED algorithm:

$$C_{N-RED}(s) = \left(\frac{K_{c_{rd}}}{\frac{s}{K_{red}} + 1} \right), \tag{7.29}$$

where K_{cr} is given by equation (7.27). For brevity in further analysis and simulations we concentrate only on two values of gain margin: $\gamma_{rd} = 2$ and $\gamma_{rd} = 3$, giving us following two designs of feedback congestion control algorithm based on RED principle, i.e. the N-RED algorithm:

- *N-RED Algorithm Design 1 :*
  In this algorithm (N-RED-1) we select $\gamma_{rd} = 2$ giving $K_{c_{rd}} = 0.5 \cdot K_{u_{rd}}$ which is the standard guideline for proportional control algorithm as given in Table 7.1 and [Ziegler and Nichols 1942]. After the second step of the above design procedure, equation (7.28), we get the following expression for maximum mark/drop probability, max_p:

$$max_p = (max_{th} - min_{th}) \cdot \frac{10.84 \times 10^{-3}}{W} \cdot \sqrt{\left(\frac{4.59}{W^2} + \frac{0.413}{W} + 0.0711 \right)} \cdot \sqrt{\left(\frac{0.59}{W^2} + \frac{0.413}{W} + 1.0711 \right)} \cdot \sqrt{\left(\frac{0.59}{W^2} + \frac{0.413}{W} + 0.0811 \right)}. \tag{7.30}$$

- *N-RED algorithm design 2 :*
  For this design of N-RED algorithm (N-RED-2) we choose $\gamma_{rd} = 3$ giving $K_{c_{rd}} = 0.33 \cdot K_{u_{rd}}$. Similar to the previous case, equation (7.28) gives the following expression for maximum mark/drop probability, max_p:

$$max_p = (max_{th} - min_{th}) \cdot \frac{7.22 \times 10^{-3}}{W} \cdot \sqrt{\left(\frac{4.59}{W^2} + \frac{0.413}{W} + 0.0711 \right)} \cdot \sqrt{\left(\frac{0.59}{W^2} + \frac{0.413}{W} + 1.0711 \right)} \cdot \sqrt{\left(\frac{0.59}{W^2} + \frac{0.413}{W} + 0.0811 \right)}. \tag{7.31}$$

Next, in Figure 7.13 we plot variations in the ultimate gain $K_{u_{rd}}$ and controller gain $K_{c_{rd}}$, for both N-RED-1 and N-RED-2 algorithm, with changes in congestion window size W. Furthermore, after determining expression of $K_{u_{rd}}$ by substituting equation (7.21) in equation (7.14) and applying equation (7.28) of design procedure we get the following generalized form

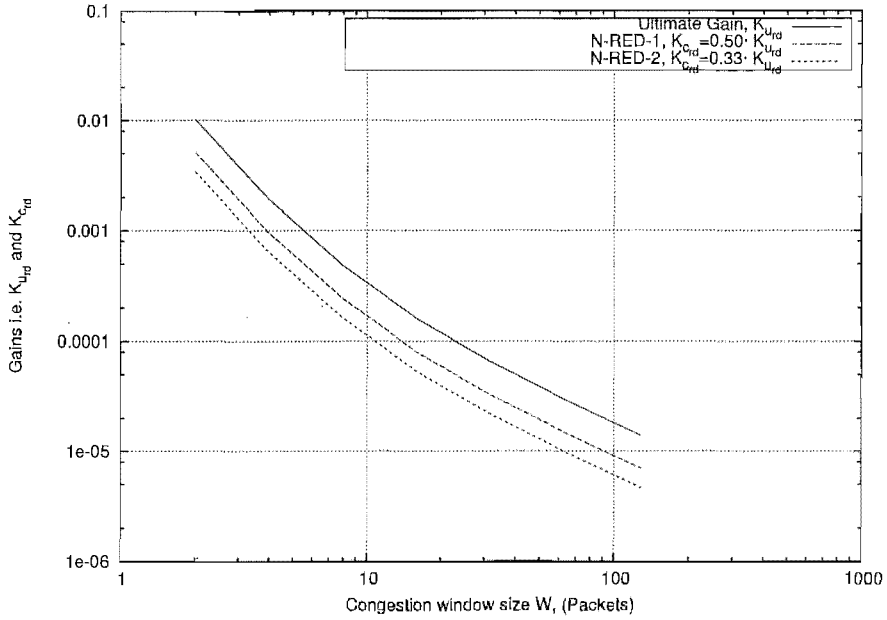

Figure 7.13 Variations of ultimate gain, $K_{u_{rd}}$, and controller gain, $K_{c_{rd}}$, with congestion window, W.

of max_p:

$$max_p = \frac{1}{\gamma_{rd}} \cdot (max_{th} - min_{th}) \cdot \left(\frac{2}{W} \cdot \frac{n}{RC}\right) \cdot \sqrt{\left\{\left(\frac{A_{zr}}{W} + B_{zr}\right)^2 + \left(\frac{2}{W}\right)^2\right\} \cdot \left\{\left(\frac{A_{zr}}{W} + B_{zr}\right)^2 + 1\right\} \cdot \left\{\left(\frac{A_{zr}}{W} + B_{zr}\right)^2 + \frac{1}{n^2}\right\}}, \tag{7.32}$$

where A_{zr} and B_{zr} are determined by the curve fitting equation for $\omega_{c_{rd}}$, whereas in equation (7.23) their values are 0.7745 and 0.2667 respectively. If we use a higher order curve fitting for $\omega_{c_{rd}}$ then the number of constants will increase appropriately. In the next Subsection we evaluate the performance of both designs of N-RED algorithms i.e. N-RED-1 and N-RED-2 as developed in this Subsection.

7.5.2 Simulations of N-RED Algorithms

In this Subsection first we compute the step responses and stability margins of both designs of N-RED algorithms (N-RED-1 and N-RED-2) as given by equations (7.30) and (7.31) respectively by using MATLAB. For both algorithms we have ultimate gain $K_{u_{rd}} = 1.862 \times 10^{-4}$ as determined from Figure 7.4 in the Section 7.3.2.1.

Later on the queue size variations of N-RED-1 and N-RED-2 algorithms are determined through the ns-based simulations performed on the network topology as given in Figure 7.3 and explained in the Section 7.3.2. For both N-RED-1 and N-RED-2 algorithms we have round trip

time $R = 0.246$ s, link capacity $C = 3750$ packets/s, $max_{th} = 800$ packets, $min_{th} = 100$ packets and $n = 10$. Substituting the values of R, C and n in equation (7.18) and equation (7.17) give $K_{red} = 0.4065$ and $w_q = 1.0840 \times 10^{-4}$ which will be used below in determining the transfer functions of N-RED-1 and N-RED-2 algorithms and ns-based simulations.

- N-RED-1 algorithm
  As given before in this case we have gain margin $\gamma_{rd} = 2$ giving $K_{c_{rd}} = 0.5 \cdot K_{u_{rd}} = 9.31 \times 10^{-5}$. Thus this algorithm employs the proportional controller guideline as given in Table 7.1 and [Ziegler and Nichols 1942]. Further the equation (7.28) gives maximum mark/drop probability $max_p = 0.06517$. Using equation (7.29) the transfer function of N-RED-1 can be written as follows:

$$C_{N-RED-1}(s) = 9.31 \times 10^{-5} \cdot \frac{0.4065}{s + 0.4065} \equiv \frac{3.784 \times 10^{-5}}{s + 0.4065}. \tag{7.33}$$

  The closed loop step response, open loop Bode plots showing stability margins, instantaneous queue size, EWMA queue size and mark/drop probability plot for N-RED-1 algorithm are shown in Figures 7.14, 7.15, 7.16, 7.17 and 7.18 respectively.

- N-RED-2 algorithm
  In this design we choose gain margin $\gamma_{rd} = 3$ giving $K_{c_{rd}} = 0.33 \cdot K_{u_{rd}} = 6.206 \times 10^{-5}$. Substituting values in equation (7.28) we get $max_p = 0.04344$ and the corresponding transfer function as obtained through equation (7.29) is given by:

$$C_{N-RED-2}(s) = 6.206 \times 10^{-5} \cdot \frac{0.4065}{s + 0.4065} \equiv \frac{2.52 \times 10^{-5}}{s + 0.4065}. \tag{7.34}$$

  The closed loop step response, open loop Bode plots showing stability margins, instantaneous queue, EWMA queue and mark/drop probability for N-RED-2 algorithm are shown in Figures 7.19, 7.20, 7.21, 7.22 and 7.23 respectively.

The comparison of step responses of N-RED-1 and N-RED-2 algorithms shown in Figures 7.14 and 7.19, respectively, reveals that latter has less overshoot and smaller settling time although its rise time is slightly higher. Thus on balance, N-RED-2 algorithm has a better transient response for changing traffic as compared to N-RED-1 algorithm. Also N-RED-2 has higher stability margins compared to N-RED-1 as seen from their Bode plots given in Figures 7.20 and 7.15, respectively.

However, in the ns-based simulations, both N-RED-1 and N-RED-2 algorithms have very close values for arithmetic mean and variance of instantaneous and EWMA queue size as given in Table 7.4 below. Thus overall performance of the N-RED-2 algorithm is better than the N-RED-1 algorithm. Equations (7.27) and (7.28) suggests that for higher values of gain margin, γ_{rd}, the values of both K_c and max_p would be smaller thus causing less number of packet drops for $min_{th} \leq \bar{q} \leq max_{th}$ as is evident from linear marking/dropping probability profile

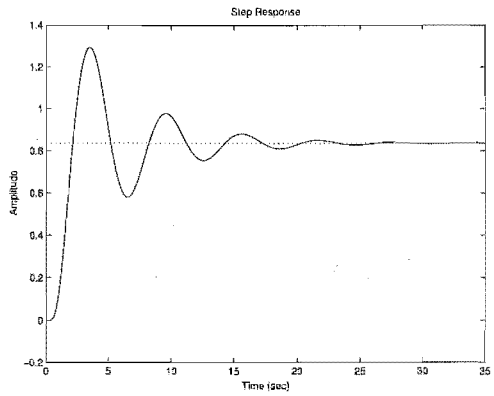

Figure 7.14 Step response of N-RED-1 algorithm.

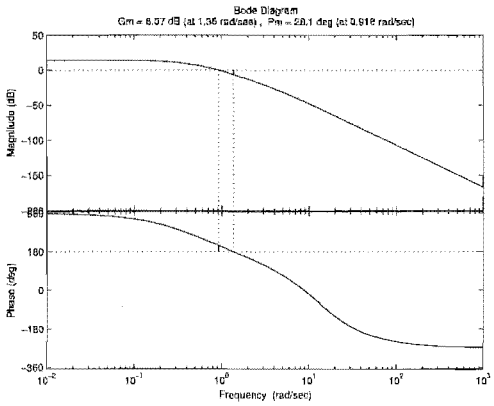

Figure 7.15 GM and PM of N-RED-1 algorithm.

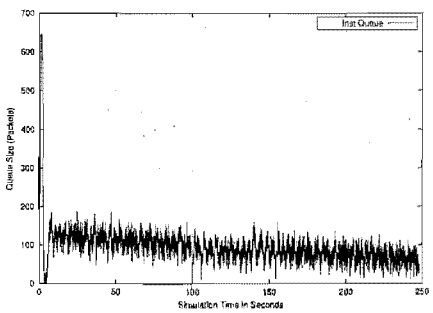

Figure 7.16 Instantaneous queue of N-RED-1 algorithm.

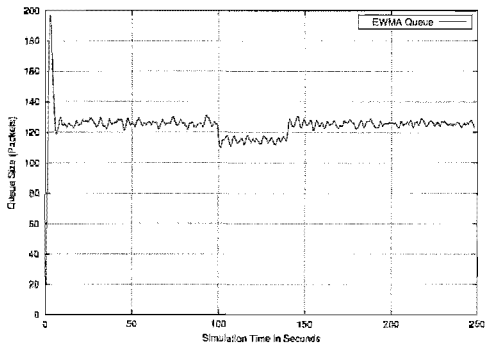

Figure 7.17 EWMA queue of N-RED-1 algorithm.

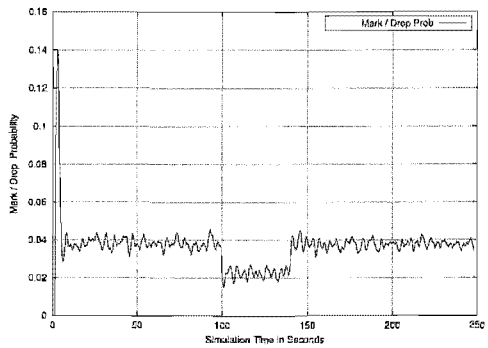

Figure 7.18 Mark/Drop probability for N-RED-1 algorithm.

Control algorithm	Instantaneous queue		EWMA queue, $w_q = 0.002$	
	Arithmetic Mean	Variance	Arithmetic Mean	Variance
N-RED-1/2	127	1974	124	102

Table 7.4 Arithmetic mean and variance of instantaneous and EWMA queue size (packets) of N-RED-1 and N-RED-2 algorithms.

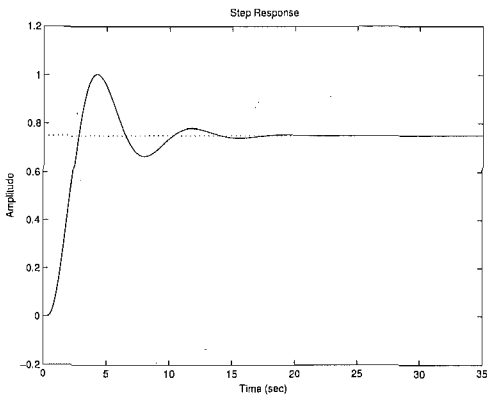

Figure 7.19 Step response of N-RED-2 algorithm.

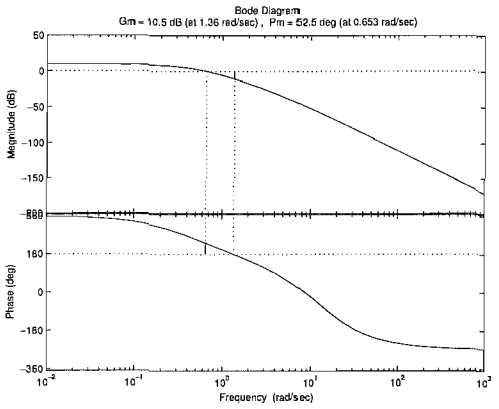

Figure 7.20 GM and PM of N-RED-2 algorithm.

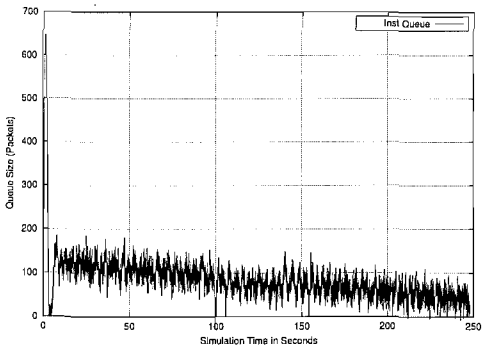

Figure 7.21 Instantaneous queue of N-RED-2 algorithm.

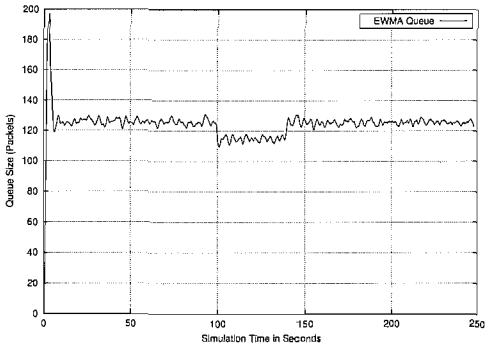

Figure 7.22 EWMA queue of N-RED-2 algorithm.

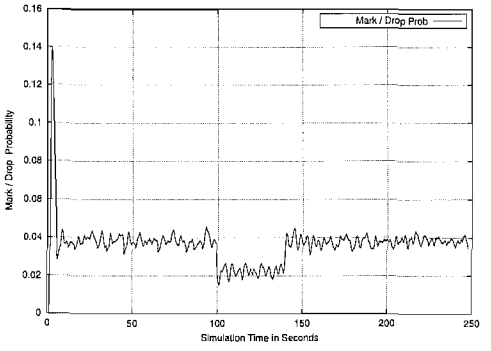

Figure 7.23 Mark/Drop probability of N-RED-2 algorithm.

of RED algorithm given in Figure 5.4 in Chapter 5. In the next Subsection we present a mutual comparison between both control theoretic approaches of RED algorithms i.e. a performance comparison between HO-RED algorithm described in Subsection 7.3.1 and N-RED algorithm described in Subsection 7.5 of this Chapter.

7.6 COMPARISON BETWEEN HO-RED AND N-RED ALGORITHMS

We can compare HO-RED and N-RED algorithms in terms of step response, variations of instantaneous queue size, EWMA queue size and mark/drop probability. The step response comparison is presented in Table 7.5, which show that HO-RED is slowest in response compared to other two designs of N-RED. It takes 63.3 s for settling and has rise time of 34.6 s which is quite slow. HO-RED design does not have overshoot and thus avoids queue oscillations at start. However, the slow response time makes it unsuitable for use with varying traffic loads.

On the other hand N-RED-1 has 3.24 times and N-RED-2 has 4.83 times less settling time then HO-RED, which show their superior characteristic in reaching steady state queue level. Also they have very sharp rise time which is necessary in case of sharply changing bursty traffic. The N-RED designs do have overshoot at the start but that transient dies out in a short time. In

Control algorithm	Parameters		
	Overshoot	*Settling Time*	*Rise Time*
HO-RED	-	63.3 s	34.6 s
N-RED-1	54.7%, 3.51s	19.5 s	1.17 s
N-RED-2	33.5%, 4.21s	13.1 s	1.58 s

Table 7.5 Comparison of step response of HO-RED, N-RED-1 and N-RED-2 algorithms.

Table 7.6 we present a comparison of the stability margins of HO-RED, N-RED-1 and N-RED-2 algorithms. It shows that the gain margin of N-RED-1 and N-RED-2 are 6.07 db (at 1.36 rad/s) and 10.5 db (at 1.36 rad/s) respectively, which are quite small, but more practical, as compared to 32.82 db (at 1.004 rad/s) of HO-RED algorithm. Whereas in the normal control practice, the typical values of gain margin vary from 2 to 5 and phase margin vary from 30 to 60 degrees as given in [Astrom and Hagglund 1995]. Thus, HO-RED has the limitation of slow response

Closed loop system	GM (db)	PM (deg)
P(s)+HO-RED	$+32.82\ db,\ 1.004\ rad/s$	$88.81^o\ 0.0502\ rad/s$
P(s)+N-RED-1	$+6.07\ db,\ 1.36\ rad/s$	$28.1^o\ 0.918\ rad/s$
P(s)+N-RED-2	$+10.5\ db,\ 1.36\ rad/s$	$52.5^o\ 0.653\ rad/s$

Table 7.6 Comparison of stability margins of closed loop feedback control AQM system formed by HO-RED, N-RED-1 and N-RED-2 algorithms.

time and N-RED has small gain margin, hence we need to develop and design other feedback

congestion control algorithms which have both good response time and high stability margins i.e. GM and PM at the same time.

Another basic limitation of algorithms derived from the basic principle of RED algorithm is the unpredictable level of the arithmetic mean of instantaneous and EWMA queue sizes. It is because that RED algorithm in itself does not have any single particular parameter which can set its EWMA queue level to a desired setpoint level. Also all of the four main parameters of the RED algorithms, $\{max_{th}, min_{th}, max_p, w_q\}$, are intertwined together as given in equations (5.9) and (5.11), thus they exert a strong influence on the performance of each other.

An advance estimate of the arithmetic mean of instantaneous queue size or EWMA queue size can be used to compute the queuing delays τ_d in the routers which can be used to determine the round trip time, R, using equation (3.8) and equation (3.9) as given before in Chapter 3. Hence, by having an estimate τ_d we can introduce the concept of quality of service in computer networks and the present best effort Internet [Floyd *et al.* 2001].

In Chapter 6 one solution to this problem has been achieved by developing an Auto-Tuning RED algorithm whose EWMA queue size converges to arithmetic mean of min_{th} and max_{th} parameters, also called reference queue size q_{ref}, as given by equation (6.12). This choice of q_{ref} is opted for maintaining the operating point of router in the middle of dynamic range of EWMA queue size i.e. $[max_{th}, min_{th}]$. However, in order to change the level of q_{ref} we have to change the values of min_{th} and max_{th} which will require the change in buffer size of RED router's queue. Also the change in min_{th} and max_{th} values will effect the throughput of RED algorithm as packet marking/dropping depends on the these thresholds as given in equation (5.9), and shown in Figure 5.4 of Chapter 5.

In order to avoid the above mentioned shortcoming in Auto-Tuning algorithm RED algorithm the operating point, q_{ref}, can be easily decoupled from the arithmetic mean of min_{th} and max_{th} (for instance, by relating it only to either of two thresholds). However, the Auto-Tuning RED algorithm also requires the proper choice of parameters ζ and ϵ otherwise its performance will not be optimized.

An alternative method for dynamically achieving the desired setpoint level of router queue size without affecting the other operational parameters is Proportional-Integral (PI) control which is widely used in industry for similar purposes. With the PI control algorithm we can safely anticipate to be able to change the setpoint or operating point of router's queue size in the closed loop formed by TCP/IP AQM, as already given in Figure 5.1 of Chapter 5, by simply adjusting a single independent parameter. Hence using the PI control loop based AQM we will be able to change reference queue size q_{ref} without hurting any other required characteristics (such as throughput) which is in contrast to RED based AQM control loop. Therefore, in the next Section we will develop new congestion control algorithms, based on the PI control principle [Franklin *et al.* 1994] and [Astrom and Wittenmark 1990], for AQM routers supporting TCP/IP flows.

7.7 PI-PRINCIPLE BASED CONGESTION CONTROL ALGORITHMS

PI controllers are well studied in control theory and are widely used in process control and many other industries for eliminating constant steady state errors. According to some surveys more then 90% of controllers being used in process control industry are operated as PI, [Astrom and Wittenmark 1990], which highlights the success of the PI-principle.

In this Section first we will present an existing PI-principle based congestion control algorithm for AQM routers, [Hollot *et al.* 2001b] and [Hollot *et al.* 2002], named herein as HO-PI. Next we develop a new and generalized PI-principle based congestion control algorithm for AQM routers, named as N-PI, by using the frequency response method as given before in the Section 7.4.1 of this Chapter. The PI controller design as given by the guidelines in Table 7.1 is also implemented in ns simulation as a special case of N-PI algorithm. In the last part of this Section we will compare the performance of HO-PI and two different designs of N-PI algorithm.

7.7.1 HO-PI Congestion Control Algorithm

This algorithm is derived on the basis of following standard transfer function of a PI controller as given in [Franklin *et al.* 1994]:

$$C_{PI}(s) = K_p\frac{(s+K_i/K_p)}{s} = K_i\frac{\left(\frac{s}{\frac{K_i}{K_p}}+1\right)}{s} \equiv K_{PI}\frac{(s/\tau+1)}{s}, \tag{7.35}$$

where K_p and K_i are proportional and integral gains respectively. In the next two subsections we present design and simulation results respectively for the HO-PI algorithm.

7.7.1.1 Design of HO-PI Algorithm

The design procedure of Hollot's PI (HO-PI) can be summarized in the following four steps:

[1] The transfer function of the PI-based congestion control algorithm is assumed to be of the form given in equation (7.35).

[2] Assume that the relationship between TCP pole and queue pole, as given by transfer functions in equations (5.4) and (5.5) respectively, is $\frac{2N}{R^2C} \ll \frac{1}{R}$. Then for critical frequency given by $\omega_g = \frac{2N}{R^2C}$ the PI gain in equation (7.35) is calculated as $K_{PI} = \omega_g \left|\frac{\left(\frac{j\omega_g}{p_{queue}}+1\right)}{\frac{(R^+C)^3}{(2N^-)^2}}\right|$. It gives a phase margin PM $\approx 90^o - \frac{180}{\pi}\cdot\omega_g^2$.

[3] Place the zero, τ, of $C_{PI}(s)$ in equation 7.35 at TCP pole, i.e. $\tau = \frac{-2N}{R^2C}$, and transform $C_{PI}(s)$ to $C_{PI}(z)$ using the Tustin transform, [Astrom and Wittenmark 1990], at the sampling time period T_s. In [Hollot *et al.* 2001b] it is suggested to use $T_s = 2\pi/\omega_g$, whereas we give an alternative guideline to select the sampling time/frequency in the next

part of this Subsection. The general form of z-domain transformed function, $C_{PI}(z)$, is then given by:

$$C_{PI}(z) = \frac{az - b}{z - 1}. \tag{7.36}$$

[4] From AQM control loop in Figure 7.1 we have $C_{PI}(z) = \frac{\delta p}{\delta q}$ where $\delta p = p - p_{ref}$ and $\delta q = q - q_{ref}$. After assuming that $p_{ref} = 0$ the inverse z-transform, [Hostetter 1988], of $C_{PI}(z)$ will yield the following expression as generalized PI based packet marking/dropping probability algorithm:

$$p(kT_s) = a \cdot [q(kT_s) - q_{ref}] - b \cdot [q\{(k-1)T_s\} - q_{ref}] + p\{(k-1)T_s\}. \tag{7.37}$$

Hence, the generalized form of closed loop of AQM as presented in Figure 7.1 can be simplified to the following PI-principle based closed loop as in [Hollot *et al.* 2001b] and [Hollot *et al.* 2002]:

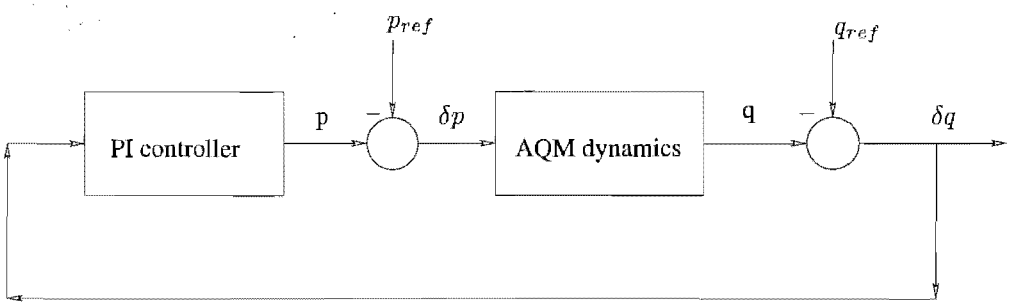

Figure 7.24 A simplified closed loop model of AQM formed by PI-principle based congestion control algorithm.

The PI-principle based packet marking/dropping probability algorithm as given in equation (7.37) and the AQM control loop as shown in Figure 7.24 will be further employed for development N-PI congestion control algorithm in this Chapter and its adaptive version in Chapter 8.

The rate of convergence of instantaneous queue size to a desired reference level, q_{ref}, of the HO-PI algorithm as given in equation (7.37) depends critically upon the proper selection of sampling time T_s and parameters $\{a, b\}$. Undersampling of queue size due to inappropriate value of T_s and/or incorrect setting of parameters $\{a, b\}$ will cause queue size to diverge from desired reference level q_{ref}, thus defeating the objectives of AQM. Hence, due to the critical importance of the proper setting of parameters $\{T_s, a, b\}$ of the HO-PI algorithm next we derive their analytical expressions. First we determine an expression for sampling time T_s and then we derive the expressions for the constants a and b. These expressions show the dependance of $\{T_s, a, b\}$ on the network parameters which are number of connections N, link capacity C and round trip time R. The fundamental lower bound to the sampling rate, ω_s, given by the sampling theorem, is at least twice the required closed loop bandwidth, ω_b, of the system, [Franklin

et al. 1992], i.e.

$$\frac{\omega_s}{\omega_b} > 2. \tag{7.38}$$

However, in order to get the reasonably smooth time response it is suggested in [Franklin *et al.* 1992] that $\omega_s \geq 20 \cdot \omega_b$. Thus, for a closed loop bandwidth of ω_g the sampling time period, T_s, can be determined by the following relation:

$$T_s \leq 0.05 \cdot \left(\frac{2\pi}{\omega_g}\right). \tag{7.39}$$

From the second step of the HO-PI design we can substitute $\omega_g = \frac{2N}{R^2C}$ in equation (7.39) to get:

$$T_s \leq 0.05 \cdot \left(\pi \cdot \frac{R^2C}{N}\right). \tag{7.40}$$

Further substituting equation (6.9) from Chapter 6, (i.e. $W = \frac{RC}{N}$), we can write equation (7.40) as:

$$T_s \leq 0.05 \cdot \pi \cdot W \cdot R. \tag{7.41}$$

Thus, the appropriate sampling frequency, f_s, is given by:

$$f_s \geq 20 \cdot \left(\frac{N}{\pi \cdot R^2C}\right). \tag{7.42}$$

The above equation can be used to determine the sampling frequency for the HO-PI algorithm and also for other PI-principle based congestion control algorithms developed in the next Subsection.

For the network given in Figure 7.3 of the Subsection 7.3.2 of this Chapter we will have $f_s \geq 1.683$ Hz. Thus, to be more conservative, the router's queue size can be oversampled by a factor of 100 to get $f_s = 168.31$ Hz instead of minimum value of $f_s = 1.683$ Hz. In our simulation experiments we will employ $f_s = 168$ Hz. In the case of the RED algorithm, Section 5.14 of Chapter 5, the sampling frequency is the same as link speed, which gives $f_s = 3750$ Hz for our simulation topology. Thus one of the advantages of HO-PI (and other PI based algorithms) over RED based congestion control algorithms is decreased queue sampling frequency or increased queue sampling time.

Further, in order to compute the analytical expressions for constants a and b in equation (7.37), we substitute TCP pole $\tau = \frac{2N}{R^2C}$ in equation (7.35) to get the following expression of transfer function, $C_{HO-PI}(s)$, of HO-PI algorithm:

$$C_{HO-PI}(s) = K_{PI}\left\{\frac{\left(\frac{s}{\frac{2N}{R^2C}}\right)+1}{s}\right\}. \tag{7.43}$$

Next, in order to transform $C_{PI}(s)$ to $C_{PI}(z)$ we use the following expression of Tustin transform as given in [Astrom and Wittenmark 1990]:

$$s = \frac{2}{T_s} \cdot \left(\frac{z-1}{z+1} \right), \tag{7.44}$$

where T_s is sampling time period. Therefore, substituting equation (7.44) in equation (7.43) for sampling time period $T_{s_{hp}}$, we get the following expression of $C_{HO-PI}(z)$:

$$C_{HO-PI}(z) \equiv \frac{a_{hp} \cdot z - b_{hp}}{z-1} = K_{PI} \left\{ \frac{\frac{2}{T_{s_{hp}}} \cdot \left(\frac{z-1}{z+1} \right) \cdot \left(\frac{R^2 C}{2N} \right) + 1}{\frac{2}{T_{s_{hp}}} \cdot \left(\frac{z-1}{z+1} \right)} \right\}. \tag{7.45}$$

Further simplification of equation (7.45) will yield the following expression:

$$C_{HO-PI}(z) \equiv \frac{a_{hp} \cdot z - b_{hp}}{z-1} = \frac{K_{PI}}{2N} \cdot \left\{ \frac{z(R^2 C + N T_{s_{hp}}) - (R^2 C - N T_{s_{hp}})}{(z-1)} \right\}. \tag{7.46}$$

Thus, from equation (7.46) we can get the following expression for the parameters a_{hp} and b_{hp} of the HO-PI algorithm as given by equation (7.37):

$$a_{hp} = \frac{K_{PI}}{2N} \cdot (R^2 C + N T_{s_{hp}}), \tag{7.47}$$

$$b_{hp} = \frac{K_{PI}}{2N} \cdot (R^2 C - N T_{s_{hp}}). \tag{7.48}$$

In the second step of HO-PI design we have the following expression for HO-PI gain K_{PI}:

$$K_{PI} = \omega_g \left| \frac{\left(\frac{j\omega_g}{p_{queue}} + 1 \right)}{\frac{(RC)^3}{(2N)^2}} \right|. \tag{7.49}$$

Again substituting $\omega_g = \frac{2N}{R^2 C}$ in equation (7.49) and further simplifying we get the following expression:

$$K_{PI} = \frac{8N^3}{R^5 C^4} \cdot \sqrt{1 + \left(\frac{2N}{R^2 C} \cdot \frac{1}{p_{queue}} \right)^2}. \tag{7.50}$$

From equation (5.5) in Chapter 5 the location of queue pole is given by $p_{queue} = \frac{-1}{R}$, which can be substituted in equation (7.50) to get:

$$K_{PI} = \frac{8N^3}{R^5 C^4} \cdot \sqrt{1 + \left(\frac{2N}{RC} \right)^2}. \tag{7.51}$$

Finally, put equation (7.51) in (7.47) and (7.48) to get the following expressions for a_{hp} and b_{hp}:

$$a_{hp} = \frac{4N^2}{R^5C^4} \cdot \sqrt{1 + \left(\frac{2N}{RC}\right)^2} \cdot (R^2C + NT_{s_{hp}}), \tag{7.52}$$

$$b_{hp} = \frac{4N^2}{R^5C^4} \cdot \sqrt{1 + \left(\frac{2N}{RC}\right)^2} \cdot (R^2C - NT_{s_{hp}}). \tag{7.53}$$

Thus, by using equations 7.41, 7.52 and 7.53 we can determine $\{T_{s_{hp}}, a_{hp}, b_{hp}\}$ parameters of HO-PI algorithm for a given set network conditions $\{N, R, C, W\}$.

7.7.1.2 Simulations of HO-PI Algorithm

Using the procedure outlined in previous Subsection the following HO-PI congestion control algorithm is designed in [Hollot *et al.* 2001b] for the network given in Figure 7.3 of the Section 7.3.2:

$$C_{HO-PI}(s) = 9.6426 \times 10^{-6} \cdot \left(\frac{\frac{s}{0.53} + 1}{s}\right). \tag{7.54}$$

We can use equations (7.52) and (7.53) with $T_{s_{hp}} = 1/168$ s, as determined before by using equation (7.40), to compute the values for parameters a_{hp} and b_{hp} to be used in equation (7.37). Alternatively, we can also directly discretize equation (7.54) at $f_{s_{hp}} = 168$ Hz by using MATLAB to get the following expression of $C_{PI}(z)$:

$$C_{HO-PI}(z) = \frac{1.822 \times 10^{-5} \cdot z - 1.816 \times 10^{-5}}{z - 1}. \tag{7.55}$$

Compare equation (7.55) with the general form of HO-PI transfer function i.e. $C_{PI}(z) = \frac{a_{hp} \cdot z - b_{hp}}{z-1}$ to get $a_{hp} = 1.822 \times 10^{-5}$ and $b_{hp} = 1.816 \times 10^{-5}$. Further, using equation (7.37) for the transfer function given in equation (7.55), we get the following expression of the HO-PI congestion control algorithm:

$$p(kT_{s_{hp}}) = 1.822 \times 10^{-5} \cdot \left[q(kT_{s_{hp}}) - q_{ref}\right] - 1.816 \times 10^{-5} \cdot \left[q\{(k-1)T_{s_{hp}}\} - q_{ref}\right] + p\{(k-1)T_{s_{hp}}\}. \tag{7.56}$$

The step response and Bode plots (for determining stability margins) of the transfer function given in equation (7.55), obtained by using MATLAB, are shown in Figure 7.25 and Figure 7.26, respectively. The HO-PI congestion control algorithm as given in equation (7.56) was simulated using ns for the network topology given in Figure 7.3. For a reference level of queue size $q_{ref} = 200$ packets the instantaneous queue size and the packet mark/drop probability are shown in Figures 7.27 and 7.28, respectively.

From the step response shown in Figure 7.25 we determine that the settling and rise times of HO-PI congestion control algorithm with $q_{ref} = 200$ packets are 5.61 s and 2.99 s which are small compared to the values for HO-RED, N-RED-1 and N-RED-2 congestion control algorithms as given in Table 7.5. Thus PI principle based congestion control algorithms have a faster response as compared to RED principle based congestion control algorithms.

The GM of HO-PI is 18.88 db at 3.49 rad/s and PM is 75.23 degrees at 0.25 rad/s as shown in Figure 7.26. Comparison of the stability margins of HO-PI with those of HO-RED, N-RED-1 and N-RED-2 congestion control algorithms as given in Table 7.6 show that HO-RED has higher values and N-RED-1 and N-RED-2 have lower values. But higher stability margins of HO-RED are at the expense of large settling and rise times as seen in Table 7.5.

The queue convergence characteristics of HO-PI are as expected. The arithmetic mean of instantaneous queue size is 203 packets which is close to $q_{ref} = 200$ packets. The variance of instantaneous queue size is 1972. Whereas in the case of N-RED-1 and N-RED-2 congestion control algorithms the instantaneous queue size converges to the arbitrary value of 127 packets with a variance of 1974. For the HO-RED algorithm the queue size does not converge properly during the simulation interval due to the extremely long settling time of 63.3 s as can be seen in Figure 7.8. It depicts that one can successfully control the converged level of the instantaneous queue size by using PI-principle based congestion control algorithms, unlike with RED-principle based congestion control algorithms.

The design procedure for the HO-PI congestion control algorithm as given in Subsection 7.7.1.1 depends upon placing the TCP pole $p_{tcp} = \frac{-2N}{R^2C}$ at the zero location τ of $C_{PI}(s)$ as given in equation 7.35. The number of TCP connections N through an AQM router are always changing, thus the location of zero of $C_{PI}(s)$ is highly variable. The same effect can be seen in the analytical expressions of parameters a and b as given by equations (7.52) and (7.53) respectively. Thus, the HO-PI design is not scale invariant as it requires different values of parameters a_{hp} and b_{hp} for different values of N and R. In the next Subsection we develop an alternative PI-principle based congestion control algorithm, N-PI, which is also not scale invariant in its basic form. However, it can be made scale invariant by its adaptive version developed in the next Chapter.

7.7.2 N-PI Congestion Control Algorithm

In this Subsection we develop an alternative design of PI-principle based congestion control algorithm using the frequency response method as given before in Subsection 7.4.1. Two cases of the N-PI algorithm are considered (N-PI-1 and N-PI-2) and their expressions are derived by generalising the guidelines for tuning PI controllers as given in Table 7.1. Next we evaluate the performance of both N-PI-1 and N-PI-2 algorithms by using MATLAB to determine the step response and stability margins, and ns-based simulations to study the queue size variation characteristics.

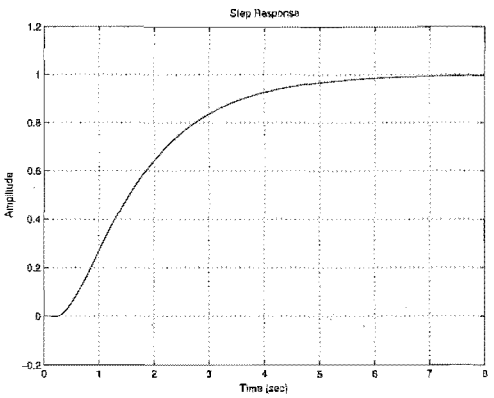

Figure 7.25 Step response of HO-PI algorithm.

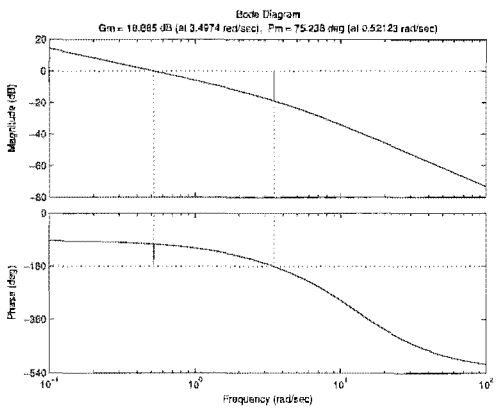

Figure 7.26 GM and PM of HO-PI algorithm.

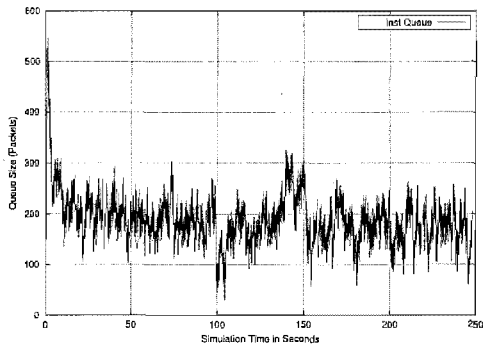

Figure 7.27 Instantaneous queue of HO-PI algorithm.

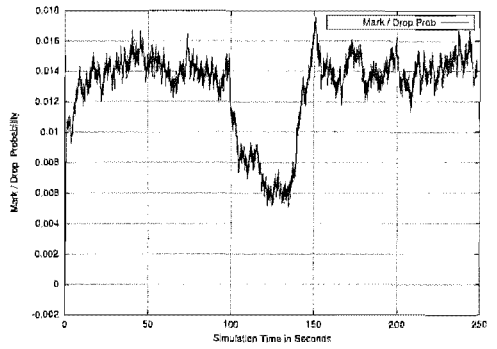

Figure 7.28 Mark/Drop probability of HO-PI algorithm.

7.7.2.1 General Design of N-PI Algorithms

For development of the general design of N-PI congestion control algorithm we start with the uncompensated plant transfer function, $P(s)$, as given in equation (5.6) of Chapter 5. Whereas in the case of the N-RED congestion control algorithm, as presented before in Subsection 7.5.1, we began with equation (7.10) which includes an additional term due to the low pass filtering action of EWMA process used in basic RED algorithm as given by equation (5.11) in the Section 5.14 of Chapter 5. Thus, compared to N-RED, the N-PI algorithm will have simpler expressions for both ultimate gain $K_{u_{pi}}$ and ultimate frequency $\omega_{u_{pi}}$.

In Subsection 5.3.2 of Chapter 5 we have given a time domain model of the closed loop formed by AQM in Figure 5.1 and its transfer function representation by equation (5.6) i.e.

$$P(s) \equiv P_{td}(s) \cdot P_{tcp}(s) \cdot P_{queue}(s) = \frac{\left(\frac{C^2}{2N}\right) e^{-sR}}{(s + \frac{2N}{R^2C})(s + \frac{1}{R})}. \tag{7.57}$$

We can write equation (7.57) in the frequency domain as:

$$P(j\omega) = \frac{\left(\frac{C^2}{2N}\right) e^{-j\omega R}}{(j\omega + \frac{2N}{R^2C})(j\omega + \frac{1}{R})}. \tag{7.58}$$

Equating the product of magnitudes of equation (7.58) at the ultimate frequency, $\omega_{u_{pi}}$, and corresponding ultimate gain, $K_{u_{pi}}$, to 1 gives the following expression:

$$|K_{u_{pi}} \cdot P(j\omega_{u_{pi}})| \equiv K_{u_{pi}} \cdot \frac{\frac{C^2}{2N}}{\sqrt{\omega_{u_{pi}}^2 + \left(\frac{2N}{R^2C}\right)^2} \cdot \sqrt{\omega_{u_{pi}}^2 + \frac{1}{R^2}}} = 1. \tag{7.59}$$

From equation (7.59) we get the following expression for $K_{u_{pi}}$:

$$K_{u_{pi}} = \left(\frac{2N}{C^2}\right) \cdot \sqrt{\omega_{u_{pi}}^2 + \left(\frac{2N}{R^2C}\right)^2} \cdot \sqrt{\omega_{u_{pi}}^2 + \frac{1}{R^2}}. \tag{7.60}$$

We need to find an expression for the ultimate frequency $\omega_{u_{pi}}$ for substitution in equation (7.60). As in the case of the N-RED algorithm in Subsection 7.5.1, the Nyquist stability criterion given in [Nyquist 1932] and [Franklin *et al.* 1994] gives the phase angle relation for critical stability of the open loop transfer function $G(s)$ as $\angle G(j\omega) = -\pi$. Thus, for the transfer function given by equation (7.58), we have the following transcendental equation:

$$\angle P(j\omega) \equiv -\omega R - \tan^{-1}\left(\frac{R^2C}{2N} \cdot \omega\right) - \tan^{-1}(\omega R) = -\pi. \tag{7.61}$$

Substituting $\omega = \omega_{u_{pi}}$ and $W = RC/N$ in equation (7.61) we get:

$$\omega_{u_{pi}} \cdot R + \tan^{-1}\left(\frac{WR}{2} \cdot \omega_{u_{pi}}\right) + \tan^{-1}(\omega_{u_{pi}} \cdot R) = \pi. \tag{7.62}$$

Further defining $\omega_{c_{pi}} = \omega_{u_{pi}} \cdot R$ and we get:

$$\omega_{c_{pi}} + \tan^{-1}\left(\frac{W}{2} \cdot \omega_{c_{pi}}\right) + \tan^{-1}(\omega_{c_{pi}}) = \pi. \tag{7.63}$$

Solving equation (7.63) numerically by using the software package Mathematica for the different practical values of window size W, we get the following table of values:

W	$1/W$	$\omega_{c_{pi}} = \omega_{u_{pi}} \cdot R$	$\hat{\omega}_{c_{pi}}$	%Error=$\frac{\omega_{c_{pi}} - \hat{\omega}_{c_{pi}}}{\omega_{c_{pi}}} \times 100$
2	0.5	1.306	1.326	−1.531
4	0.25	1.136	1.104	+2.816
8	0.125	1.017	0.993	+2.359
16	0.0625	0.945	0.938	+0.740
32	0.03125	0.904	0.910	−0.663
64	0.015625	0.882	0.896	−1.587
128	0.0078125	0.871	0.889	−2.066

Table 7.7 Data for least square curve fitting for ω_{cp} for PI based AQM.

The data given in Table 7.7 has been plotted in Figures 7.29 and 7.30. The graph shown in Figure 7.30 is almost linear and thus will be used for curve fitting. The general form of linear

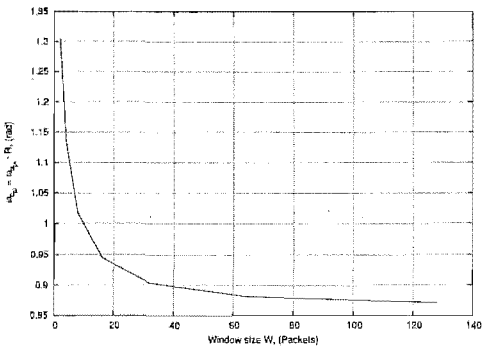

Figure 7.29 Variations in $\omega_{cp} = \omega_{up} \cdot R$ with changes in congestion window, W, for PI based AQM.

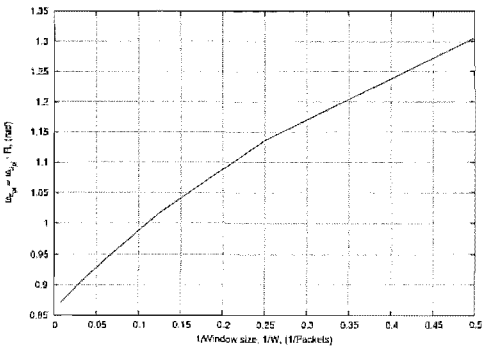

Figure 7.30 Variations in $\omega_{cp} = \omega_{up} \cdot R$ with changes in reciprocal of congestion window, $1/W$, for PI based AQM.

curve fit equation between ω_{cp} and W is given by:

$$\omega_{c_{pi}} = \left(\frac{A_{pi}}{W} + B_{pi}\right), \tag{7.64}$$

where A_{pi} and B_{pi} in the above equation are constants whose values are determined by the curve fitting method. Since we have already defined $\omega_{c_{pi}} = \omega_{u_{pi}} \cdot R$, thus from equation (7.64) we can obtain the following expression of $\omega_{u_{pi}}$:

$$\omega_{u_{pi}} = \left(\frac{A_{pi}}{WR} + \frac{B_{pi}}{R}\right). \tag{7.65}$$

Substituting equation (7.65) in equation (7.60) we get the following expression for the ultimate gain $K_{u_{pi}}$:

$$K_{u_{pi}} = \frac{2}{NW^2}\sqrt{\left(\frac{A_{pi}}{W} + B_{pi}\right)^2 + \left(\frac{2}{W}\right)^2}\sqrt{1 + \left(\frac{A_{pi}}{W} + B_{pi}\right)^2}. \tag{7.66}$$

After least squares linear curve fitting, between $\omega_{c_{pi}}$ and W, by using the software package Maple, [Maplesoft 1998], we get $A_{pi} = 0.888$ and $B_{pi} = 0.882$. Thus, we have the following equation for $\omega_{c_{pi}}$:

$$\omega_{c_{pi}} = \frac{0.888}{W} + 0.882. \tag{7.67}$$

In order to check the accuracy of this curve fitting we compute the estimated values of $\omega_{c_{pi}}$ for different values of congestion window, W, by using equation (7.67). These values are given in Table 7.7 as $\hat{\omega}_{c_{pi}}$. The percentage error between $\omega_{c_{pi}}$ and its estimated value $\hat{\omega}_{c_{pi}}$ is quite small (around 1.5 %) for the normal operating range for the congestion window. Hence equation (7.67) will be further used in the development of N-PI congestion control algorithm. From equation

(7.67) we get the following denormalised expression of $\omega_{u_{pi}}$:

$$\omega_{u_{pi}} = \frac{0.888}{RW} + \frac{0.882}{R}. \tag{7.68}$$

Now substituting equation (7.68) in equation (7.60), we get:

$$K_{u_{pi}} = \frac{2N}{C^2} \cdot \sqrt{\left(\frac{0.888}{RW} + \frac{0.882}{R}\right)^2 + \left(\frac{2N}{R^2C}\right)^2} \cdot \sqrt{\left(\frac{0.888}{RW} + \frac{0.882}{R}\right)^2 + \frac{1}{R^2}}. \tag{7.69}$$

Simplifying equation (7.69), we get:

$$K_{u_{pi}} = \frac{2}{NW^2} \cdot \sqrt{\frac{4.788}{W^2} + \frac{1.566}{W} + 0.777} \cdot \sqrt{\frac{0.788}{W^2} + \frac{1.566}{W} + 1.777}. \tag{7.70}$$

Thus, for a network modelled by the plant transfer function given in equation (7.57) and for a given value of congestion window size W and round trip time R we can determine the ultimate frequency, $\omega_{u_{pi}}$, by equation (7.68) and the ultimate gain, $K_{u_{pi}}$, by equation (7.70). Next we determine the expressions for the constants a_{hp} and b_{hp} in the packet marking/dropping algorithm given in equation (7.37). Let us consider the transfer function of a general PI controller, N-PI, as given in equation (B.1), i.e. :

$$C_{N-PI}(s) = K_{c_{pi}} \cdot \left(1 + \frac{1}{s \cdot T_{i_{pi}}}\right). \tag{7.71}$$

Discretizing the transfer function in above equation by using the Tustin transform at sampling time period of $T_{s_{pi}}$, as given by equation (7.44), we get following expression for $C_{N-PI}(z)$:

$$C_{N-PI}(z) \equiv \frac{a_{pi} \cdot z - b_{pi}}{z-1} = K_{c_{pi}} \cdot \left\{ \frac{z \cdot \left(1 + 0.5 \cdot \frac{T_{s_{pi}}}{T_{i_{pi}}}\right) - \left(1 - 0.5 \cdot \frac{T_{s_{pi}}}{T_{i_{pi}}}\right)}{z-1} \right\}. \tag{7.72}$$

Thus, from equation (7.72) we get the following expressions for a_{pi} and b_{pi}:

$$a_{pi} = K_{c_{pi}} \cdot \left(1 + 0.5 \cdot \frac{T_{s_{pi}}}{T_{i_{pi}}}\right), \tag{7.73}$$

$$b_{pi} = K_{c_{pi}} \cdot \left(1 - 0.5 \cdot \frac{T_{s_{pi}}}{T_{i_{pi}}}\right). \tag{7.74}$$

In order to evaluate a_{pi} and b_{pi} from equations (7.73) and (7.74), respectively, we need to determine the expressions for control algorithm gain $K_{c_{pi}}$ and integral time $T_{i_{pi}}$. We define the relationship between $K_{c_{pi}}$ and $K_{u_{pi}}$ as $\{K_{c_{pi}} = \sigma_{pi} \cdot K_{u_{pi}},\ 0 < \sigma_{pi} \leq 0.45\}$. Then using the expression for ultimate gain, $K_{u_{pi}}$, as given in equation (7.66) we get the control algorithm gain,

$K_{c_{pi}}$ as:

$$K_{c_{pi}} = \frac{2\sigma_{pi}}{NW^2}\sqrt{\left(\frac{A_{pi}}{W} + B_{pi}\right)^2 + \left(\frac{2}{W}\right)^2}\sqrt{1 + \left(\frac{A_{pi}}{W} + B_{pi}\right)^2}. \tag{7.75}$$

Similarly we define the relationship between integral time, $T_{i_{pi}}$, and ultimate time, $T_{u_{pi}}$ as $\{T_{i_{pi}} = \delta_{pi} \cdot T_{u_{pi}},\ \delta_{pi} \geq 0.85\}$, where $T_{u_{pi}}$ is given by the identity $T_{u_{pi}} = \frac{2\pi}{\omega_{u_{pi}}}$. Then by using equation (7.64) for the value of ultimate frequency ω_{up} we can get the following expression for $T_{i_{pi}}$:

$$T_{i_{pi}} = \frac{2\pi \cdot \delta_{pi}}{\frac{1}{R}\left(\frac{A_{pi}}{W} + B_{pi}\right)}. \tag{7.76}$$

The generalized expressions of $K_{c_{pi}}$ and $T_{i_{pi}}$ as given in equations (7.75) and (7.76), respectively, can be substituted in equations (7.73) and (7.74) to get values of a and b, which can be finally fitted in equation (7.37) to get the packet marking/dropping probability algorithm. Thus we have developed a generalized form of PI-principle based congestion control algorithm, N-PI, using the frequency response method as given in the Section 7.4.1.

From equations (7.75) and (7.76) it can be observed that for different values of $\{\sigma_{pi}, \delta_{pi}\}$ we will have different designs of N-PI algorithm. The corresponding control algorithm transfer function $C_{N-PI}(s)$ and values of a_{pi} and b_{pi} parameters can be obtained by substituting $K_{c_{pi}} = \sigma_{pi} \cdot K_{u_{pi}}$ and $T_{i_{pi}} = \delta_{pi} \cdot T_{u_{pi}}$ in equations (7.71), (7.73) and (7.74) respectively to get the following expressions :

$$C_{N-PI}(s) = \sigma_{pi} \cdot K_{u_{pi}} \cdot \left(1 + \frac{1}{s \cdot \delta_{pi} \cdot T_{u_{pi}}}\right), \tag{7.77}$$

$$a_{pi} = \sigma_{pi} \cdot K_{u_{pi}} \cdot \left(1 + 0.5 \cdot \frac{T_{s_{pi}}}{\delta_{pi} \cdot T_{u_{pi}}}\right), \tag{7.78}$$

$$b_{pi} = \sigma_{pi} \cdot K_{u_{pi}} \cdot \left(1 - 0.5 \cdot \frac{T_{s_{pi}}}{\delta_{pi} \cdot T_{u_{pi}}}\right). \tag{7.79}$$

Thus, by changing the values of σ_{pi} and δ_{pi} we can tailor N-PI algorithm according to the required characteristics of the AQM router. The equations (7.78) and (7.79) depend upon $K_{u_{pi}}$ and $T_{u_{pi}}$ which can be computed by using equations (7.70) and (7.76) respectively.

The expressions for $\omega_{u_{pi}}$ and $K_{u_{pi}}$ obtained through least square curve fitting ($A_{pi} = 0.888, B_{pi} = 0.882$) as given in equations (7.68) and (7.70), respectively, can be used in equations (7.78) and (7.79) to get the numerical values of a and b. Thus, packet marking/dropping algorithm N-PI as given in equation (7.37) is obtained.

Further in this Chapter we will concentrate only on two versions of N-PI algorithms which are N-PI-1 algorithm having $\{\sigma_{pi}, \delta_{pi}\} = \{0.45, 0.85\}$ and N-PI-2 algorithm having $\{\sigma_{pi}, \delta_{pi}\} = \{0.3, 0.85\}$. The former N-PI-1 algorithm follows the classical Ziegler Nichols design of PI controllers as given in [Ziegler and Nichols 1942] and reproduced in Table 7.1. The latter N-PI-2 algorithm is chosen to give a 50 % increase in the gain margin of N-PI-1 algorithm.

The expressions for control algorithm gain $K_{c_{pi}}$ and integral time $T_{i_{pi}}$ for N-PI-1 and N-PI-2 algorithms are obtained as follows:

N-PI-1 congestion control algorithm:

For N-PI-1 congestion control algorithm substitute $\{\sigma_{pi},\ \delta_{pi}\} = \{0.45,\ 0.85\}$ in equations (7.75) and (7.76) to get the following expressions of control algorithm gain $K_{c_{pi}}$ and integral time $T_{i_{pi}}$:

$$K_{c_{pi}} = \frac{0.9}{NW^2}\sqrt{\left(\frac{A_{pi}}{W} + B_{pi}\right)^2 + \left(\frac{2}{W}\right)^2}\sqrt{1 + \left(\frac{A_{pi}}{W} + B_{pi}\right)^2}, \tag{7.80}$$

$$T_{i_{pi}} = \frac{1.7\pi}{\frac{1}{R}\left(\frac{A_{pi}}{W} + B_{pi}\right)}. \tag{7.81}$$

Further substituting the least square curve fitting parameters $\{A_{pi},\ B_{pi}\} = \{0.888,\ 0.882\}$ in equation (7.80) and equation (7.81) we get:

$$K_{c_{pi}} = \frac{0.9}{NW^2}\sqrt{\frac{4.788}{W^2} + \frac{1.566}{W} + 0.777}\cdot\sqrt{\frac{0.788}{W^2} + \frac{1.566}{W} + 1.777}. \tag{7.82}$$

$$T_{i_{pi}} = \frac{1.7\pi}{\frac{1}{R}\left(\frac{0.888}{W} + 0.882\right)}. \tag{7.83}$$

N-PI-2 congestion control algorithm:

In this case we use $\{\sigma_{pi},\ \delta_{pi}\} = \{0.3,\ 0.85\}$ giving following expression for K_{cp}:

$$K_{c_{pi}} = \frac{0.6}{NW^2}\sqrt{\frac{4.788}{W^2} + \frac{1.566}{W} + 0.777}\cdot\sqrt{\frac{0.788}{W^2} + \frac{1.566}{W} + 1.777}, \tag{7.84}$$

Whereas the equation for T_i for N-PI-2 algorithm will be the same as equation (7.83). In order to compute the value of $K_{c_{pi}}$ or $T_{i_{pi}}$ for N-PI-1 or N-PI-2 algorithms we need to substitute values of σ_{pi}, δ_{pi}, N, R and W in equations (7.82), (7.83) and (7.84).

An alternative approach to compute the value of $K_{c_{pi}} = \sigma_{pi} \cdot K_{u_{pi}}$ and $T_{i_{pi}} = \delta_{pi} \cdot T_{u_{pi}}$ is to use the Bode plots of the overall plant transfer function as given in equation (5.6) of Chapter 5. These plots will give us the values of $K_{u_{pi}}$ and $T_{u_{pi}}$, as done in the Section 7.3.2.1, form which we can compute $K_{c_{pi}}$ and $T_{i_{pi}}$ for a given set values of σ_{pi} and δ_{pi}.

Thus, the values of constants a_{pi} and b_{pi} for N-PI-1 and N-PI-2 algorithms can be computed by substituting either equations (7.82) and (7.83) or equations (7.84) and (7.83) into equations (7.73) and (7.74), respectively. Finally the N-PI-1 or N-PI-2 algorithm based packet marking/dropping probability can be obtained by substituting the corresponding values of a_{pi} and b_{pi} into equation 7.37.

7.7.3 Stability of General N-PI algorithms

The stability of feedback-control closed loop of PI-principle based AQM, formed by plant and N-PI controller as given by equations (7.57) and (7.77) respectively, shown in Figure 7.24, depends upon the proper selection of values of σ_{pi} and δ_{pi}. The stability of the said closed loop is determined by the location of roots of the denominator, $D(s)$, of its transfer function which is given by:

$$D(s) = 1 + C_{N-PI}(s) \cdot P(s). \tag{7.85}$$

A direct substitution of equations (7.77) and (7.57) into equation (7.85) leads to the following transcendental equation:

$$\delta_{pi} \cdot T_{u_{pi}} \cdot s \cdot \left(s + \frac{2N}{R^2C}\right) \cdot \left(s + \frac{1}{R}\right) + \frac{C^2}{2N} \cdot \sigma_{pi} \cdot K_{u_{pi}} \cdot \left(\delta_{pi} \cdot T_{u_{pi}} \cdot s + 1\right) \cdot e^{-sR} = 0. \tag{7.86}$$

The first order Pade approximation, [Franklin *et al.* 1994], for $e^{-s \cdot R}$ term in the above equation leads to an algebraic quartic equation whereas the second order approximation leads to a fifth order algebraic equation, both of which can be solved using Mathematica or MAPLE software. However, we found that the root locus for different values of σ_{pi} and δ_{pi} contains an unacceptable error due to Pade approximation.

An alternative method to determine closed loop stability, involving no errors, is to simulate the step response characteristics of the closed loop transfer function, $T(s) = \frac{C_{N-PI}(s) \cdot P(s)}{1 + C_{N-PI}(s) \cdot P(s)}$, for different values of σ_{pi} and δ_{pi}. For the network topology under consideration, given in Figure 7.3, we computed the step response characteristics for different values of σ_{pi} and δ_{pi} ranging between 0 and 1. The stable combinations of σ_{pi} and δ_{pi} are shown in Figure 7.31, which corroborates our assumptions, $\sigma_{pi} \leq 0.45$ and $\delta_{pi} \geq 0.85$, made in the previous Subsection.

For $0 \leq \sigma_{pi} \leq 1$ the variations in rise time, settling time and % overshoot for the AQM closed loop with changes in δ_{pi}, in the stable region of operation, are shown in Figures 7.32, 7.33 and 7.34, respectively. Thus, required stable combinations of σ_{pi} and δ_{pi} can be chosen for the desired characteristics of the N-PI based AQM control loop.

7.7.3.1 Simulations of N-PI Algorithms

For performance evaluation of N-PI-1 and N-PI-2 congestion control algorithms we use the network topology shown in Figure 7.3. It gives the ultimate gain $K_{u_{pi}} = 1.862 \times 10^{-4}$ and ultimate time $T_{u_{pi}} = 1.629$ s, as determined before in Subsection 7.3.2.1. First we determine the transfer functions of N-PI-1 and N-PI-2 algorithms by using equation (7.77) and then determine their step responses and stability margins using MATLAB.

As described in the previous Subsection, the values of constants a and b for the N-PI-1 and N-PI-2 algorithms are computed to simulate the equation (7.37) for $q_{ref} = 200$ packets. Then

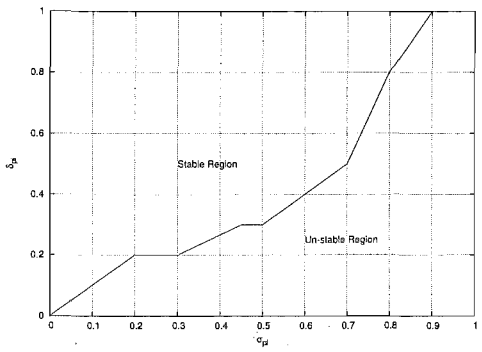

Figure 7.31 Stable combinations of σ_{pi} and δ_{pi}.

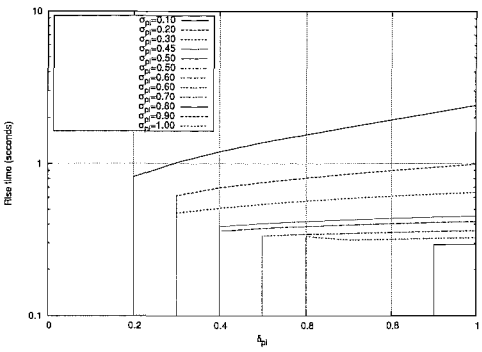

Figure 7.32 Variations in rise time

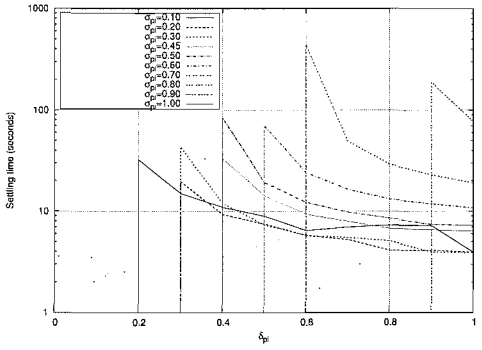

Figure 7.33 Variations in settling time

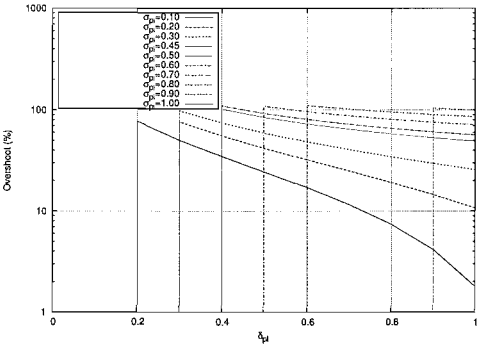

Figure 7.34 Variations in % overshoot.

we perform ns-based simulations using N-PI-1 and N-PI-2 algorithms for the network topology shown in Figure 7.3 with the traffic pattern as described in the Section 7.3.2.

The maximum sampling time period for the router's queue size is given by equation (7.40) which also gives the minimum sampling frequency f_s in equation (7.42). Thus for our simulation topology we will need to have minimum sampling frequency of 1.683 Hz. We oversample the N-PI router's queue length by a factor of 100 to get the sampling frequency of 168 Hz in the ns-based simulations. In case of the RED principle based congestion control algorithms, given in the Section 5.14, we will have a sampling frequency equal to the bottleneck link capacity (packets/s) which is 3750 Hz for the network topology under consideration. Thus, for N-PI-1 and N-PI-2 algorithms we need to sample the router queue size less frequently as compared to N-RED-1 and N-RED-2 algorithms.

At the end of ns simulations we plot instantaneous queue size and packet mark/drop probability at the bottleneck link (node C to node D in Figure 7.3) router. In order to compare N-PI-1 and N-PI-2 algorithms quantitatively, we also compute the arithmetic mean, variance and standard deviation of the instantaneous queue size and present them in a Table. The simulation results of N-PI-1 and N-PI-2 algorithms are presented as follows:

N-PI-1 congestion control algorithm:

After substituting the values of $K_{u_{pi}}$ and $T_{u_{pi}}$ as obtained before and $\sigma_{pi} = 0.45, \delta_{pi} = 0.85$ in equation (7.77) we obtain the following transfer function of the N-PI-1 congestion control

algorithm :

$$C_{N-PI-1}(s) = 6.0513 \times 10^{-5} \cdot \left(\frac{\frac{s}{0.722} + 1}{s} \right). \tag{7.87}$$

The step response and Bode plots for computing stability margins of transfer function in equation (7.87) are plotted in Figure 7.35 and Figure 7.36, respectively. The settling time of N-PI-1 algorithm is 6.67 s and rise time is 0.44 s as given in Table 7.8. Whereas gain margin is 5.02 db at 3.3.5 rad/s and phase margin is 26.4 degrees at 2.18 rad/s as given in Table 7.9. Substituting values in equations (7.78) and (7.79) we get $a_{pi} = 8.40 \times 10^{-5}$ and $b_{pi} = 8.36 \times 10^{-5}$ thus giving the following N-PI-1 congestion control algorithm :

$$p(kT_{s_{pi}}) \leftarrow 8.40{\times}10^{-5}{\cdot}\{q(kT_{s_{pi}}) - 200\}{+}8.36{\times}10^{-5}{\cdot}[q\{(k-1)T_{s_{pi}}\} - 200]{+}p\{(k{-}1)T_{s_{pi}}\}. \tag{7.88}$$

After performing simulations with congestion control algorithm as given in equation (7.88) the instantaneous queue size and packet marking/dropping probability are shown in Figures 7.37 and 7.38, respectively. The quantitative results for queue variations are given in Table 7.10. The arithmetic mean of instantaneous queue size is close to the desired reference value of 200 packets with smaller amount of variance as compared to that of N-RED algorithms as given in Table 7.4. Thus, performance of N-PI-1 algorithm is better than both versions of the N-RED algorithm.

N-PI-2 congestion control algorithm:

For this algorithm we substitute values of $K_{u_{pi}}$ and $T_{u_{pi}}$, same as in the previous N-PI-1 case, but with $\sigma_{pi} = 0.30, \delta_{pi} = 0.85$ into equation (7.77) to get the following transfer function:

$$C_{N-PI-2}(s) = 4.034 \times 10^{-5} \cdot \left(\frac{\frac{s}{0.722} + 1}{s} \right). \tag{7.89}$$

The step response and stability margins of transfer function (7.89) are plotted in Figures 7.39 and 7.40 respectively. The settling time of the N-PI-2 algorithm is 4.03 s while rise time is 0.623 s as given in Table 7.8. The gain margin achieved is 8.54 db at 3.35 rad/s while phase margin is 40.7 degrees at 1.57 rad/s as given in Table 7.9. Substituting values in equations (7.78) and (7.79) we get $a_{pi} = 5.60 \times 10^{-5}$ and $b_{pi} = 5.575 \times 10^{-5}$ thus giving the following N-PI-2 congestion control algorithm :

$$p(kT_{s_{pi}}) \leftarrow 5.60{\times}10^{-5}{\cdot}\{q(kT_{s_{pi}}) - 200\}{+}5.575{\times}10^{-5}{\cdot}[q\{(k-1)T_{s_{pi}}\} - 200]{+}p\{(k{-}1)T_{s_{pi}}\}. \tag{7.90}$$

After simulation of the N-PI-2 congestion control algorithm as given in equation (7.90) the instantaneous queue size and packet marking/dropping probability are given in Figures 7.41 and 7.42, respectively. The quantitative results for instantaneous queue size variations of this algorithm are given in Table 7.10, which shows that arithmetic mean of instantaneous queue

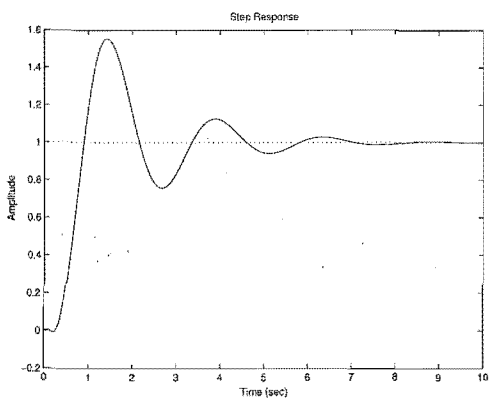

Figure 7.35 Step response of N-PI-1 algorithm.

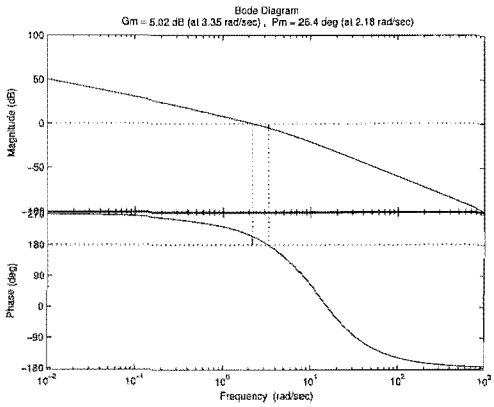

Figure 7.36 GM and PM of N-PI-1 algorithm.

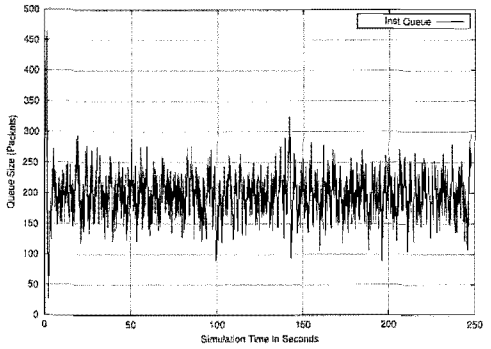

Figure 7.37 Instantaneous queue of N-PI-1 algorithm.

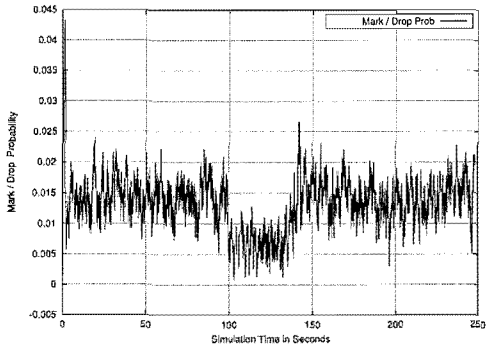

Figure 7.38 Mark/Drop probability of N-PI-1 algorithm.

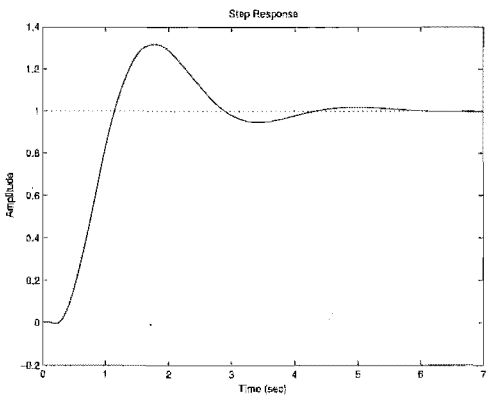

Figure 7.39 Step response of N-PI-2 algorithm.

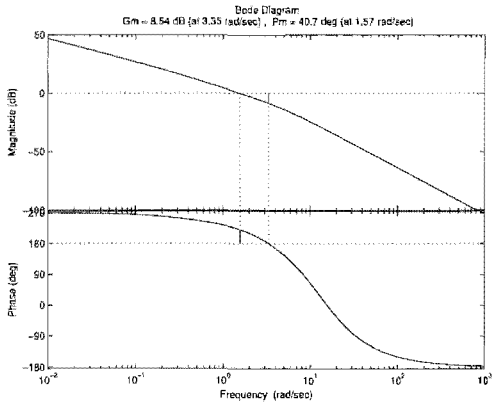

Figure 7.40 GM and PM of N-PI-2 algorithm.

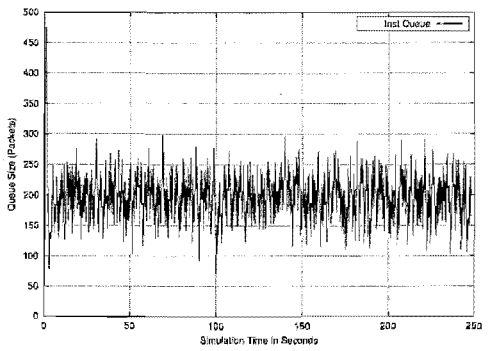

Figure 7.41 Instantaneous queue of N-PI-2 algorithm.

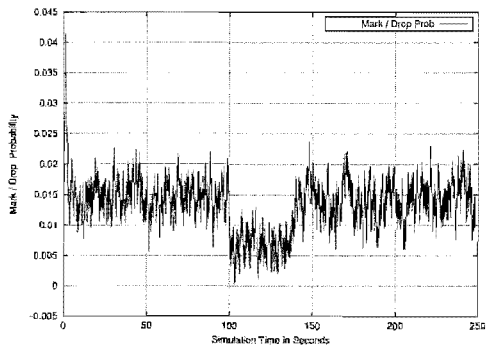

Figure 7.42 Mark/Drop probability of N-PI-2 algorithm.

size is close to the reference value of 200 packets. Comparison of this algorithm with N-RED algorithm is similar to that in the case of previous N-PI-1 algorithm.

Comparing the step response and stability margins characteristics of both N-PI-1 and N-PI-2 algorithms with that of N-RED algorithms as given in Tables 7.5 and 7.6, respectively, we find that generally the former algorithms have smaller settling and rise times while the latter algorithms have higher stability margins. Thus N-PI-1 algorithm will respond faster than N-RED algorithms in the case of sudden variations in traffic at the input port of the router. Whereas the N-RED algorithms are more stable as they have higher gain margin and phase margin as given in Table 7.6.

The variance in queue size of N-PI algorithm is less than that of N-RED algorithms as given in Table 7.4. Thus except for smaller stability margins the response to bursty traffic and convergence to reference queue size (or variance) characteristics of N-PI algorithm are better than both N-RED algorithms. Further, we present a comparison between HO-PI and N-PI algorithms in the next Subsection.

Control algorithm	Parameters			
	Peak Amplitude	Overshoot	Settling Time	Rise Time
HO-PI		-	5.61 s	2.99 s
N-PI-1	1.55	55.5 %, 1.43 s	6.67 s	0.441 s
N-PI-2	1.32	32.0 %, 1.76 s	4.03 s	0.623 s

Table 7.8 Comparison between step responses of HO-PI, N-PI-1 and N-PI-2 algorithms.

System	GM (db)	PM(deg)
P(s)+HO-PI	18.88 db, 3.49 rad/s	75.23^o 0.521 rad/s
P(s)+N-PI-1	5.02 db, 3.35 rad/s	26.4^o 2.18 rad/s
P(s)+N-PI-2	8.54 db, 3.35 rad/s	40.7^o 1.57 rad/s

Table 7.9 Comparison of stability margins of closed loop feedback control AQM system formed by HO-PI, N-PI-1 and N-PI-2 algorithms.

Control algorithm	Instantaneous Queue Size		
	Arithmetic Mean	Variance	Standard Deviation
HO-PI	204	1972	44.40
N-PI-1	199	908	30.13
N-PI-2	200	1117	33.42

Table 7.10 Comparison between Instantaneous queue size (packets) of HO-PI, N-PI-1 and N-PI-2 algorithms.

7.7.4 Comparison Between HO-PI and N-PI Algorithms

In this Subsection first we compare the performance of HO-PI and N-PI algorithms in light of their step response, stability margins, and queue variations characteristics as given in Tables 7.8, 7.9 and 7.10, respectively. Next we compare the N-PI-1 and N-PI-2 algorithms mutually.

The rise time of HO-PI algorithm is 6.78 and 4.8 times more than for N-PI-1 and N-PI-2 algorithms, respectively. Whereas the settling time of HO-PI algorithm is 1.18 times less than N-PI-1 and 1.39 times more than N-PI-2 algorithm. The gain margin of HO-PI is 13.86 db higher than N-PI-1 and 10.34 db higher than N-PI-2 algorithm. Whereas the phase margin of HO-PI is 48.83 degrees higher than N-PI-1 and 34.53 degrees higher than N-PI-2 algorithm. Thus HO-PI has higher stability margins than both versions of N-PI algorithms. On the other hand N-PI algorithms have smaller rise time but almost same settling time compared to the HO-PI algorithm. Thus, the N-PI algorithms will respond much faster than the HO-PI algorithm but will take almost the same time to reach steady state queue size after the expiry of settling time. Hence the N-PI algorithms can tackle bursty traffic better than the HO-PI algorithm.

Comparison of the instantaneous queue sizes of HO-PI and N-PI algorithms reveals that the arithmetic mean of former algorithm deviates more than the latter algorithm from the reference queue size q_{ref}. Also the variance in instantaneous queue size of HO-PI algorithm is almost two times larger than for both N-PI algorithms. Thus, HO-PI algorithm is sluggish in its response to

bursty traffic and also shows more deviations (less convergent) of the queue size as compared to both N-PI algorithms.

Further a mutual comparison of N-PI-1 and N-PI-2 algorithms highlights that except for a small difference in rise time (182 ms less for N-PI-1) the N-PI-2 algorithm has better step response characteristics (overshoot and settling time) as compared to N-PI-1 algorithm, as seen in Table 7.8. The convergence of the instantaneous queue size with N-PI-2 algorithm to q_{ref} level is slightly better than with N-PI-1 algorithm, as given in Table 7.10, whereas the N-PI-1 algorithm has a lower variance. The stability margins of N-PI-2 algorithm are higher than N-PI-1 algorithm, as given in Table 7.9. Thus, overall the N-PI-2 ($\sigma_{pi} = 0.30$, $\delta_{pi} = 0.85$) algorithm shows a better performance than N-PI-1 ($\sigma_{pi} = 0.45$, $\delta_{pi} = 0.85$) algorithm.

The HO-RED and N-RED algorithms have higher stability margins but poorer step response (high settling time and high rise time) characteristics as compared to the HO-PI and N-PI algorithms. Thus we need to find another alternative algorithm which can combine low settling time, low rise time high stability margins at the same time. One choice of such an algorithm is a PID principle based congestion control algorithm which will be developed in the next Section.

7.8 PID-PRINCIPLE BASED CONGESTION CONTROL ALGORITHM

Our next step in the development of a better feedback congestion control algorithm for AQM is to employ the Proportional-Integral-Derivative (PID) control algorithm principle, [Astrom and Hagglund 1995] and [Franklin *et al.* 1994], in packet marking/dropping decisions at the congested router. The proportional control algorithm (N-RED algorithm) acts like an amplifier whereas the proportional integral control algorithm (HO-PI and N-PI algorithms) reduces or eliminates the constant steady state errors [Franklin *et al.* 1994]. In the PID control algorithm the main function of proportional action is to reduce the constant steady state errors, integral action ensures that output agrees with setpoint value and derivative action is used to improve the closed-loop stability [Astrom and Hagglund 1995].

7.8.1 General Design of N-PID Algorithm

The general transfer function of a PID controller, having integral time $T_{i_{pid}}$ and derivative time $T_{d_{pid}}$, can be written as [Astrom and Wittenmark 1990] and [Franklin *et al.* 1994]:

$$C_{N-PID}(s) = K_{c_{pid}} \cdot \left(1 + \frac{1}{T_{i_{pid}} \cdot s} + T_{d_{pid}} \cdot s\right). \tag{7.91}$$

We use the following backward difference at sampling time period of $T_{s_{pid}}$, [Astrom and Wittenmark 1990], to transform equation (7.91) into the z-domain.

$$s = \frac{1}{T_{s_{pid}}} \cdot \left(\frac{z-1}{z}\right). \tag{7.92}$$

Substituting equation (7.92) into equation (7.91) we get the following expression:

$$\frac{\delta p(z)}{\delta q(z)} \equiv C_{PID}(z) = K_{c_{pid}} \cdot \left\{1 + \frac{T_{s_{pid}}}{T_{i_{pid}}} \cdot \frac{1}{(1-z^{-1})} + \frac{T_{d_{pid}}}{T_{s_{pid}}} \cdot (1-z^{-1})\right\}, \tag{7.93}$$

where $\delta p(z) = p(z) - p_{ref}$ and $\delta q(z) = q(z) - q_{ref}$ with $p_{ref} = 0$, as in Subsection 7.7.1.1, thus giving:

$$\delta p(z) = K_{c_{pid}} \cdot \left\{1 + \frac{T_{s_{pid}}}{T_{i_{pid}}} \cdot \frac{1}{(1-z^{-1})} + \frac{T_{d_{pid}}}{T_{s_{pid}}} \cdot (1-z^{-1})\right\} \cdot \delta q(z). \tag{7.94}$$

Next, multiplying both sides of equation (7.94) by $(1-z^{-1})$ and rearranging terms we get:

$$(1-z^{-1}) \cdot \delta p(z) = K_{c_{pid}} \cdot \left\{\left(1 + \frac{T_{s_{pid}}}{T_{i_{pid}}} + \frac{T_{d_{pid}}}{T_{s_{pid}}}\right) - \left(1 + \frac{2T_{d_{pid}}}{T_{s_{pid}}}\right) \cdot z^{-1} + \frac{T_{d_{pid}}}{T_{s_{pid}}} \cdot z^{-2}\right\} \cdot \delta q(z), \tag{7.95}$$

Further substituting $\delta q(z) = q(z) - q_{ref}$ in equation (7.95) and taking its inverse z-transform, as in [Hostetter 1988], we get the following expression:

$$\begin{aligned} p(kT_{spid}) &= p\{(k-1)T_{s_{pid}}\} + K_{c_{pid}} \cdot \left[\left(1 + \frac{T_{s_{pid}}}{T_{i_{pid}}} + \frac{T_{d_{pid}}}{T_{s_{pid}}}\right) \cdot q\left\{(k)T_{spid}\right\}\right] \\ &+ K_{c_{pid}} \cdot \left[-\left(1 + \frac{2T_{d_{pid}}}{T_{s_{pid}}}\right) \cdot q\left\{(k-1)T_{spid}\right\} + \frac{T_{d_{pid}}}{T_{s_{pid}}} \cdot q\left\{(k-2)T_{spid}\right\}\right] - \frac{T_{s_{pid}}}{T_{i_{pid}}} \cdot K_{c_{pid}} \cdot q_{ref}, \end{aligned} \tag{7.96}$$

In above equation (7.96) we substitute the guidelines for PID controller design as given in Table 7.1, $\{K_{c_{pid}} = 0.6 \cdot K_{u_{pid}}, T_{i_{pid}} = 0.5 \cdot T_{u_{pid}}, T_{d_{pid}} = 0.125 \cdot T_{u_{pid}}\}$, to get the following expression of N-PID congestion control algorithm for computing the probability for packets marking/dropping in the router:

$$\begin{aligned} p(kT_{spid}) &= p\{(k-1)T_{s_{pid}}\} + 0.6K_{u_{pid}} \cdot \left[\left(1 + \frac{2T_{s_{pid}}}{T_{u_{pid}}} + \frac{0.125T_{u_{pid}}}{T_{s_{pid}}}\right) \cdot q\left\{(k)T_{spid}\right\}\right] \\ &+ 0.6K_{u_{pid}} \cdot \left[-\left(1 + \frac{0.25T_{u_{pid}}}{T_{s_{pid}}}\right) \cdot q\left\{(k-1)T_{spid}\right\} + \left(\frac{0.125T_{u_{pid}}}{T_{s_{pid}}}\right) \cdot q\left\{(k-2)T_{spid}\right\}\right] \\ &- \frac{1.2T_{s_{pid}}}{T_{u_{pid}}} \cdot K_{u_{pid}} \cdot q_{ref}. \end{aligned} \tag{7.97}$$

In the next Subsection we perform simulations to evaluate the performance of N-PID algorithm as given in equation (7.97) above.

7.8.2 Simulations of N-PID Algorithm

For performance evaluation of N-PID congestion control algorithm we use the network topology shown in Figure 7.3 of Section 7.3.2. It gives the ultimate gain $K_{u_{pid}} = 1.862 \times 10^{-4}$ and ultimate time $T_{u_{pid}} = 1.629$ s, as determined in Subsection 7.3.2.1. Thus we will have $K_{c_{pid}} = 0.6 \cdot K_{u_{pid}} = 1.1172 \times 10^{-4}$, $T_{i_{pid}} = 0.5 \cdot T_{u_{pid}} = 0.8145$ s and $T_{d_{pid}} = 0.125 \cdot T_{u_{pid}} = 0.2036$ s. Substitute these values in equation (7.91) to get the following transfer function $C_{PID}(s)$:

$$C_{N-PID}(s) = 1.1172 \times 10^{-4} \cdot \left[1 + \frac{1}{0.8145 \cdot s} + 0.2036 \cdot s\right]. \tag{7.98}$$

The step response and Bode plot (for stability margins) of the transfer function given in equation (7.98) are shown in Figures 7.43 and 7.44 respectively. From the step response of the N-PID algorithm we determine the peak amplitude, overshoot, settling time and rise time as 1.44, 44% for 1 s, 2.89 s and 0.29 s respectively. It shows that N-PID has less peak and lesser overshoot as compared to that of the N-RED algorithms as given in Table 7.5 and that of the N-PI algorithms as given in Table 7.8. The rise time and settling time of N-PID is less than both versions of N-PI algorithms. Thus N-PID will respond faster to sudden changes in traffic load.

The gain margin and phase margin of N-PID congestion control algorithm are 7.44 db at 6.04 rad/s and 35.2 deg at 2.65 rad/s respectively. Comparing the gain and phase margins of N-PID with that of N-RED as given in Table 7.6 reveals that the former algorithm has higher margins than N-RED-1 but lower margins than N-RED-2. Thus, the stability margins of the N-PID algorithm are comparable to those of the N-RED algorithms. Further, the N-PID algorithm has higher values of stability margins as compared to the N-PI algorithms as given in Table 7.9. Thus, compared with the both N-PI-1 and N-PI-2 algorithms the N-PID algorithm has better stability margins as well as a better response time.

Next we perform ns-based simulations for N-PID algorithm after substituting the values of $K_{u_{pid}}$, $T_{u_{pid}}$ and $T_{s_{pid}}$ in equation (7.97) and setting $q_{ref} = 200$ packets as in the case of previous algorithms. Thus, we simulate the following N-PID congestion control algorithm :

$$p(kT_{s_{pid}}) \leftarrow p\{(k-1)T_{s_{pid}}\} + 3.934 \times 10^{-3} \cdot \left[q(kT_{s_{pid}})\right] - 7.755 \times 10^{-3} \cdot \left[q(k-1)T_{s_{pid}}\right] + 3.821 \times 10^{-3} \cdot \left[q(k-2)T_{s_{pid}}\right] - 1.632 \times 10^{-4}. \tag{7.99}$$

After simulation of the N-PID congestion control algorithm as given in equation (7.99) the resulting instantaneous queue size and packet marking/dropping probability are shown in Figures 7.45 and 7.46, respectively. The arithmetic mean of instantaneous queue size is 199 packets with a variance of 800. Comparing the queue variations characteristics of N-PID algorithm with N-PI-1 and N-PI-2 algorithms, given in Table 7.10, we find that former algorithm has lesser variance. Whereas the arithmetic means of both N-PID and N-PI algorithms are almost same and are very close to the desired value of $q_{ref} = 200$ packets.

Until this stage we have been assuming that all of the incoming traffic at the input port of router is TCP type or TCP compatible as described in Section 5.1 of Chapter 5. Next we consider the issue of non-responsive flows in AQM routers. These non-responsive flows grab the bandwidth of TCP and its compliant flows and thus cause gross unfairness among the competing flows in a router. One possible solution to deal with non-responsive flows is the CHOKe algorithm as already presented in Section 5.8 of Chapter 5. In the next Section we will modify the N-RED, N-PI and N-PID algorithms designed in this Chapter by coupling them with CHOKe algorithm at the router queue. We also present simulation results to evaluate the efficacy of N-RED, N-PI and N-PID algorithms coupled with CHOKe.

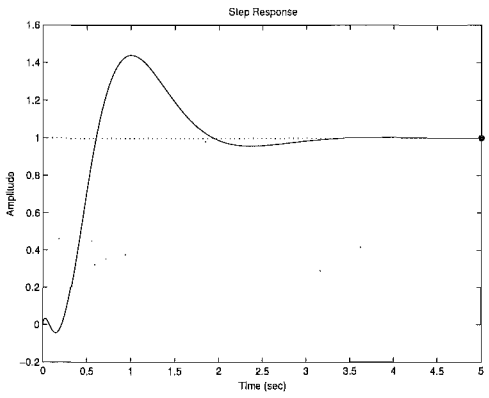

Figure 7.43 Step response of N-PID algorithm.

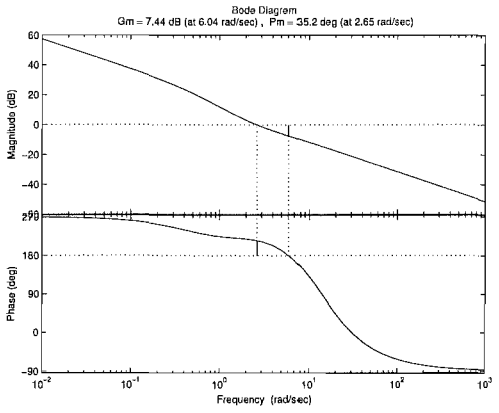

Figure 7.44 GM and PM of N-PID algorithm.

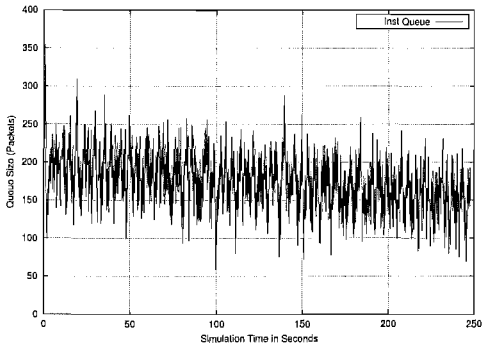

Figure 7.45 Instantaneous queue of N-PID algorithm.

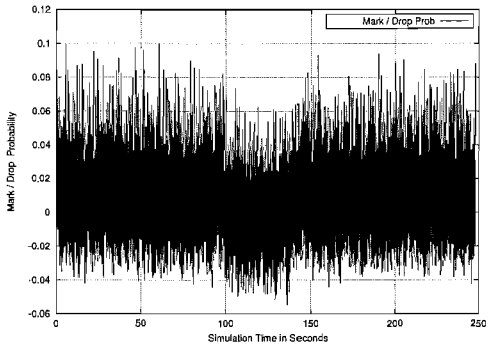

Figure 7.46 Mark/Drop probability of N-PID algorithm.

7.9 PERFORMANCE OF AQM WITH NON-RESPONSIVE FLOWS

In Section 5.1 of Chapter 5 we have outlined various types of traffic flows through a congested router. Later we developed various types of router based congestion control algorithms (specifically for TCP flows only) while excluding the non-responsive flows through a congested router. Recently the performance of AQM in the presence of non-responsive flows has been analyzed in [Hollot *et al.* 2003] wherein three different models of the non-responsive flows have been developed (Short-lived TCP, Markov ON-OFF and $M/G/\infty$ models). In [CAIDA 2003] it has

been estimated that 70-80% of the current Internet flows consists of non-responsive flows and they account for 10-20% of total byte volume. Therefore it is reasonable to make provision for dealing with non-responsive traffic flows in the congestion control algorithms of a router.

In [Floyd and Jacobson 1993] it is mentioned that the RED algorithm can be used to identify misbehaving flows at times of congestion and that information can be used to restrict their bandwidth by higher policy layers. Later this idea is extended in [Floyd *et al.* 1998], [Floyd and Fall 1997] and [Floyd and Fall 1999] wherein the packet drop history of RED algorithm is employed to identify and estimate the arrival rates of high bandwidth flows at the time of congestion. The core idea of using the drop history of RED algorithm can be explained as follows:

If a connection i has a fixed fraction p_i of the bandwidth through gateway and $S_{i,n}$ is number of n most recently marked packets from connection i then expected value of $S_{i,n}$ is np_i. In [Floyd *et al.* 1998], it is shown that for $1 \leq c \leq 1/p_i$ following inequalities holds for n,

$$Prob\left(S_{i,n} \geq cp_i n\right) \leq e^{-2n(c-1)^2 p_i^2}. \tag{7.100}$$

Using Chernoff-type bounds, [Hagerup and Rub 1990], we can write equation (7.100) as follows:

$$Prob\left(S_{i,n} \geq cp_i n\right) \leq \left[\left(\frac{1}{c}\right)^{cp_i} \cdot \left(\frac{1-p_i}{1-cp_i}\right)^{1-cp_i}\right]^n. \tag{7.101}$$

From equation (7.101) we can have:

$$n \geq \frac{\ln[p]}{\ln\left[\left(\frac{1}{c}\right)^{cp_i} \cdot \left(\frac{1-p_i}{1-cp_i}\right)^{1-cp_i}\right]}. \tag{7.102}$$

The number of packet drops, n, as given by equation (7.102) can be used to estimate the bandwidth of the highest bandwidth flow (non-responsive flow). This approach is useful only for RED type algorithms and is not suitable for other types of congestion control algorithms employed at routers such as feedback congestion control algorithms developed in this Chapter.

Another approach to combat non-responsive flows is RED with Preferential Dropping (RED-PD) given in [Mahajan and Floyd 2001]. It requires the choice of a fixed target round-trip time which may be difficult to estimate in practice. Therefore it is suggested to modify RED-PD algorithm with the dynamically varying round-trip time. Also a survey of algorithms to control the misbehaving or non-responsive flows through a congested router can be seen in [Mahajan and Floyd 2001]. All of these approaches require further investigation before they can be employed in real congested routers.

One novel and promising approach to deal with non-responsive flows is given in [Pan *et al.* 2000] which has been summarized in Section 5.8 of Chapter 5. We combine this CHOKe-principle with the RED, PI and PID based feedback congestion control algorithms developed in this Chapter. In this technique the CHOKe algorithm will filter out a portion of the aggressive

and non-responsive UDP flows before the aggregate traffic passes through the congestion control algorithms of routers which are designed mainly for TCP/IP (responsive) flows. The new cascaded form of CHOKe and feedback congestion control algorithms can be depicted by the block diagram shown in Figure 7.47. In the next Section first we present the simulation results showing the bandwidth sharing of non-responsive UDP flows (while competing with TCP flows) and secondly we show the efficacy of the proposed cascaded scheme (i.e. CHOKe with feedback congestion control algorithms) in limiting the unfair bandwidth grabbing by UDP flows.

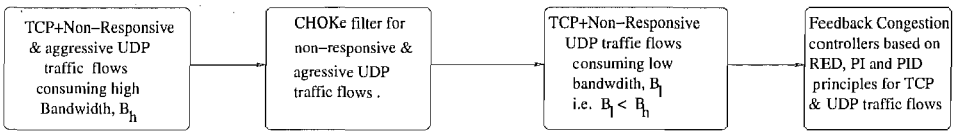

Figure 7.47 Cascading of CHOKe algorithm with feedback congestion control algorithms to filter out non-responsive and aggressive traffic UDP traffic flows.

7.9.1 Simulations

For simulation purposes we use the network topology given in Figure 7.48 which consists of 32 TCP (responsive flows) and 1 UDP (non-responsive flow) competing for the bandwidth of a bottleneck link having link capacity of 1 Mbps with propagation delay of 1 ms. All other links have a higher capacity of 10 Mbps but same propagation delay of 1 ms. We ran FTP traffic sources on all TCP connections and a CBR traffic source with data sending rate of 2 Mbps on UDP. The UDP will form the non-responsive flow which will grab the bandwidth from the TCP sources. The packet size is 1000 bytes, size of queue buffer is 300 packets and all simulation are run for a period of 250 s. In order to illustrate the effects of non-responsive UDP traffic flows

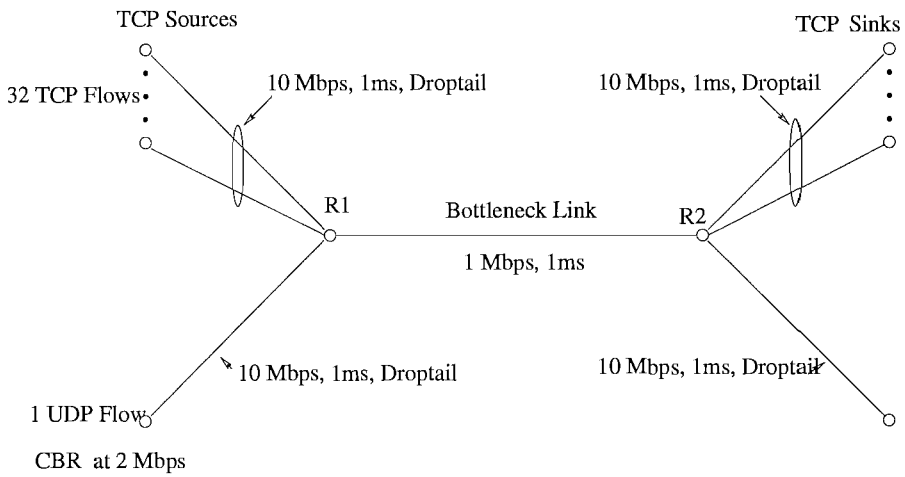

Figure 7.48 Network topology to show the effects of non-responsive traffic flows on bandwidth sharing.

(while competing with TCP traffic flows) on bottleneck link bandwidth sharing, we employ following algorithms at the bottleneck link router.

- Droptail algorithm as described in the Section 5.4.
- RED based feedback congestion control algorithm as developed in the Section 7.5.
- PI based feedback congestion control algorithm as developed in the Section 7.7.
- PID based feedback congestion control algorithm as developed in the Section 7.8.

The throughput or bandwidth consumed by UDP flows is shown in Figure 7.49, which shows their aggressiveness and unfair behavior for different router algorithms. After a short time (around 50 s) the UDP flows grab almost all of the bottleneck link bandwidth (1 Mbps or 125 packets/s), thus depriving the TCP and its compliant responsive flows from their fair share of bandwidth. This simulation suggests the need of a mechanism at the router which could reduce the aggressiveness of non-responsive UDP flows as discussed above. In the second simula-

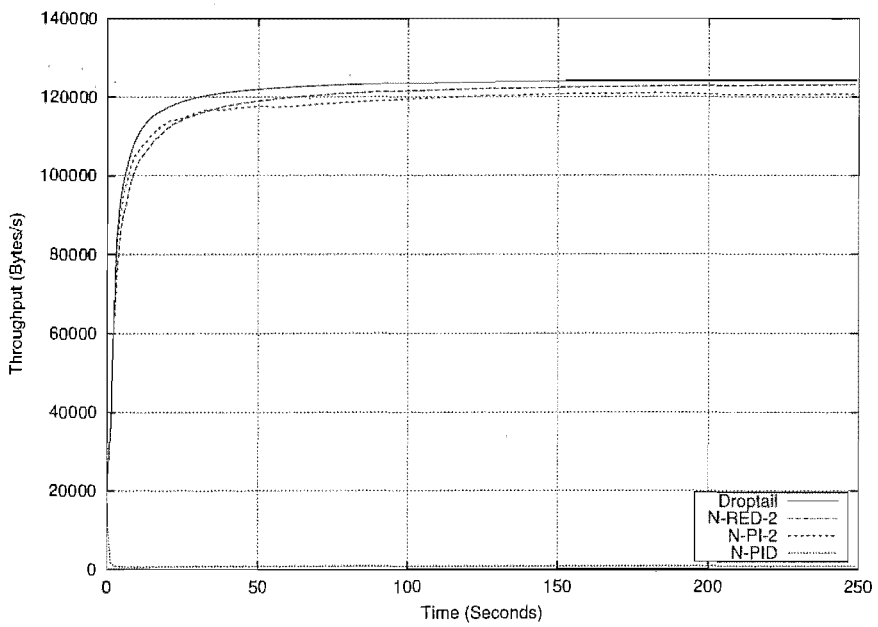

Figure 7.49 Bottleneck link bandwidth consumed by non-responsive UDP flows with different feedback congestion control algorithms.

tion we employ the cascaded algorithms (CHOKe+ N-RED-2, N-PI-2 and N-PID algorithms) at the bottleneck router and plot the UDP throughput (bandwidth consumed) in Figure 7.50. These results show that cascaded algorithms can be successfully employed to limit the bandwidth grabbing behavior of UDP flows.

7.10 CONCLUSIONS

This Chapter has developed three different types of congestion control algorithm using feedback control theoretic techniques. These algorithms are based on RED, PI and PID principles,

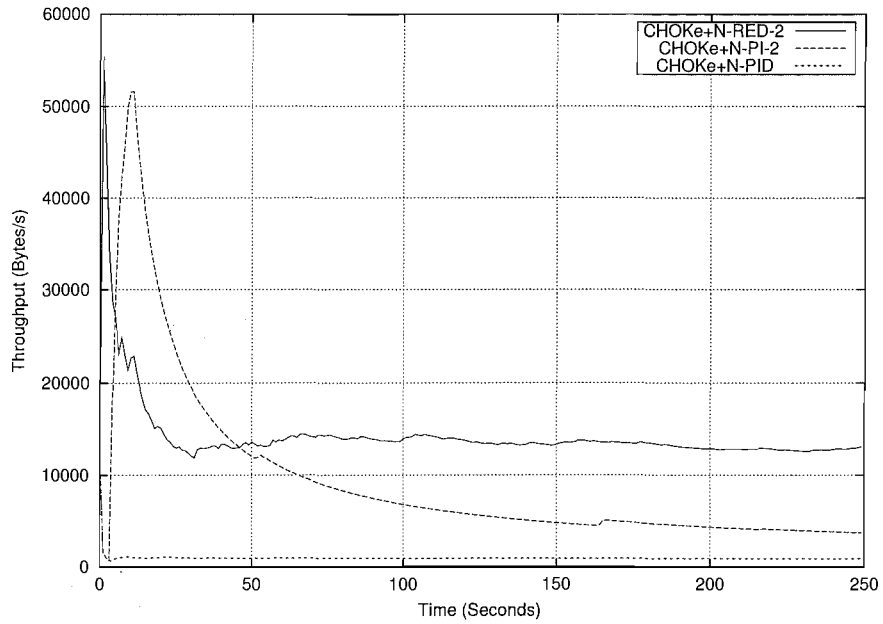

Figure 7.50 Bottleneck link bandwidth consumed by non-responsive UDP flows with different feedback congestion control algorithms combined with CHOKe algorithm.

respectively. All of the new algorithms are simulated using ns and their performance is compared with each other. It is found that in general the new congestion control algorithms have better performance than the existing algorithms.

The new control theoretic design of RED algorithm outperforms the existing designs as it gives more convergent queue level and is more responsive to changing traffic loads. We selected the value of max_p according to the proportional control law while maintaining the low pass filter characteristics of RED algorithm. This approach is in contrast to the existing heuristics being used for the RED algorithm where its value is selected arbitrarily equal to 0.1. We found that in general the selection of max_p based on the proportional control principle leads to a faster response and more convergent queue levels. Of the two RED variants introduced in this Chapter we have specifically found that the N-RED-1 (GM : $\gamma_{rd} = 2$, *i.e.* $K_{c_{rd}} = 0.50 \cdot K_{u_{rd}}$) design is slower to settle down than N-RED-2 (GM : $\gamma_{rd} = 3$, *i.e.* $K_{c_{rd}} = 0.33 \cdot K_{u_{rd}}$) design. Thus, we conclude that the gain margin γ_{rd} of N-RED algorithms affects the response time of an AQM router to bursty traffic. More work is required to optimize the selection of gain margin γ_{rd} of N-RED algorithms to get the required step response characteristics. Note that the general control theoretic design of RED algorithm presented in this Chapter can be easily tailored to the given value of γ_{rd} and network parameters (R, C, N and W).

In order to obtain a desired level of queue size we investigated PI principle based congestion control algorithms for AQM routers. We evaluated the HO-PI algorithm by simulations and found that it has large settling and rise times thus making it unsuitable for obtaining the faster response to sudden changes in traffic. But, despite its sluggishness the HO-PI algorithm gives good queue convergence characteristics to target level queue size, i.e. q_{ref}. Further we found

that the newly introduced N-PI algorithms can also clamp the instantaneous queue size to desired level q_{ref} very effectively and with much less variance. Thus, in general, we can safely determine the queuing delay introduced by N-PI algorithms. We studied two particular cases of N-PI algorithms which are for $K_{c_{rd}} = 0.30 \cdot K_{u_{rd}}$ and $K_{c_{rd}} = 0.45 \cdot K_{u_{rd}}$. The simulation results showed that the former design is better than the latter one in terms of step response characteristics. However, relating the step response characteristics (settling time, rise time etc) and the stability margins with the control gain is an open question.

Comparing the N-PI and N-RED algorithms we found that the former has smaller settling and rise times while latter has higher stability margins. In an effort to combine these two characteristics in one algorithm we further investigated PID control. We employed the standard techniques of Ziegler-Nichols tuned PID controllers to design a N-PID congestion control algorithm and performed ns-based simulations on the same network topology as before. To implement the PID principle based congestion control algorithm we have used the backward difference method to discretize s-domain transfer functions. We found that the performance of N-PID is best among other two approaches (N-RED and N-PI). We have employed only one type of PID controller structure while other forms of it need further exploration, such as given in [Astrom and Wittenmark 1990]. The relationships between step response and stability margins with scaling factors of $K_{c_{pid}}$, $T_{i_{pid}}$ and $T_{d_{pid}}$ needs further work. Also the use of a lead compensator for RED algorithms needs further investigations.

We have also demonstrated the effects of aggressive and non-responsive UDP flows on the bandwidth sharing with responsive TCP flows. We found that UDP flows can harm TCP and other responsive flows by grabbing their fair share of bandwidth. Later we devised a new cascaded scheme, based on the CHOKe algorithm, which can markedly reduce the bandwidth consumed by UDP flows by dropping more of their packets using simple comparison of their flow id's.

All feedback congestion control algorithms as presented in this Chapter require parameter settings by manual tuning which has to be done by network operators. Hence, another important issue relating to the implementation of N-RED, N-PI and N-PID congestion control algorithms is automatic or self-tuning for changing patterns of network traffic. This issue will be addressed in the next Chapter by using basic formulations given in [Mathis *et al.* 1997] to transform the algorithms presented in this Chapter into their adaptive forms.

Chapter 8

RED AND PI BASED ADAPTIVE FEEDBACK CONGESTION CONTROL ALGORITHMS

Broadly speaking, traffic in modern networks consists of varying mixtures of real time data such as speech or video, long ftp flows, short http flows or rate controlled sources. It may also contain non-responsive flows which may be highly bursty. Thus, in order to avoid congestion and to get optimized performance, the congestion control algorithms of routers need to be adaptive to variations in traffic loads. In uncongested routers the load adaptation can be achieved either manually by network operators or by employing adaptive algorithms. However, the design of a fully adaptive and self-tuned router for generalized traffic classes is not a trivial problem. Thus, in this Chapter we will design adaptive algorithms only for routers having a varying number of purely TCP/IP, or compatible, traffic flows. We propose modifications to the feedback congestion control algorithms for AQM routers supporting TCP/IP flows, which were designed in the previous Chapter, to make them adaptive to changes in the TCP/IP traffic load. To this end, techniques are derived for adapting the normally fixed parameters to changes in the number of TCP/IP connections.

As the TCP/IP traffic load in an AQM router changes, the level of congestion will also change and so the frequency of packet marking/dropping as well as the marking/dropping probability itself should also change accordingly. If the packets marking/dropping probability does not adapt itself to a change in the number of TCP connections or, in general, to changes in the total number of connections, then congestion may occur and packets may be lost because we will have too low a value of marking/dropping probability at higher numbers of connections, even while using ECN. We resolve this by relating changes in traffic load offered at the input of router to the packet marking/dropping probability and its EWMA value.

In this Chapter we develop RED and PI principles based adaptive feedback congestion control algorithms. The congestion window W is related to the number of connections, N, by equation (6.9) and to the packet marking/dropping probability, p, by equation (6.10). Hence, we can relate N and p by the equation $p = \frac{8}{3} \cdot \frac{N^2}{(RC)^2}$, which is used implicitly in this Chapter.

For RED principle based congestion control algorithms, the ultimate gain K_u of the linearized feedback control system is determined for different values of congestion window, W, and a function giving K_u in terms of W is fitted using MAPLE software[Maplesoft 1998]. From this analytical expression for K_u the control algorithm gain K_c is obtained by using the

procedure developed in the last Chapter. Finally an expression for packet marking/dropping probability is obtained. Similarly for the PI principle based adaptive congestion algorithms we develop expressions for ultimate gain and ultimate time using the approximate transfer function of TCP/IP model. We generalize the concept of using packet marking/dropping probability as an indicator of number of connections by introducing the EWMA of probability. We employ different weights to compute the EWMA probability and compare results with those for the non-adaptive algorithms developed in the last Chapter.

We begin by developing a RED based generalized adaptive feedback congestion control algorithm, in the Section 8.1. We consider two versions of RED based adaptive feedback congestion control algorithm and present their simulation results. Next we develop PI based feedback congestion control algorithms in the Section 8.2 which requires the adaptive algorithms for ultimate gain K_u and ultimate time T_u. Thus, the expressions for adaptive ultimate gain, K_u, controller gain K_c and ultimate time T_u are determined analytically. Based on EWMA of mark/drop probability p, the generalized expressions for adaptive K_u and T_u are also presented. Further, two designs of PI based feedback congestion control algorithms are developed and simulated. Finally conclusions are presented in the Section 8.3.

8.1 RED BASED ADAPTIVE FEEDBACK CONGESTION CONTROL ALGORITHM

In this Section we revisit the basic RED algorithm and develop a RED based adaptive feedback congestion control algorithm by using the fluid model of AQM presented in Chapters 5 and Chapter 6. In Section 7.5 of the previous Chapter, the ultimate gain, K_u, of an AQM feedback control loop using RED is determined in equation (7.14), and repeated below:

$$K_{u_{rd}} = \frac{2N}{C^2K} \cdot \sqrt{\omega_{u_{rd}}^2 + \left(\frac{2N}{R^2C}\right)^2} \cdot \sqrt{\omega_{u_{rd}}^2 + \frac{1}{R^2}} \cdot \sqrt{\omega_{u_{rd}}^2 + K_{red}^2}. \tag{8.1}$$

Under the assumption of fair allocation of total bandwidth C among a number N of TCP connections, the congestion window W of single TCP connection, having round trip time R, is determined in equation (6.9) as:

$$W = \left(\frac{C}{N} \cdot R\right). \tag{8.2}$$

For random packet loss at a probability p, we also have following relation for W, as equation (3.25) [Mathis *et al.* 1997]:

$$W = \sqrt{\frac{8}{3p}}. \tag{8.3}$$

Also an expression for the bandwidth K_{red} of the RED algorithm has been given in equation (7.18) as:

$$K_{red} = \frac{1}{n \cdot R}. \tag{8.4}$$

After substituting (8.2), (8.4) into equation (8.1) and defining $\omega_{c_{rd}} = \omega u_{rd} \cdot R$, we get the following expression:

$$K_{u_{rd}} = \frac{2n}{W} \cdot \frac{1}{RC} \cdot \sqrt{\omega_{c_{rd}}^2 + \left(\frac{2}{W}\right)^2} \cdot \sqrt{\omega_{c_{rd}}^2 + \frac{1}{n^2}} \cdot \sqrt{\omega_{c_{rd}}^2 + 1}. \tag{8.5}$$

Substituting $K_{n_{rd}} = K_{u_{rd}} \cdot \left(\frac{RC}{n}\right)$ in equation (8.5), we get

$$K_{n_{rd}} = \frac{2}{W} \cdot \sqrt{\omega_{c_{rd}}^2 + \left(\frac{2}{W}\right)^2} \cdot \sqrt{\omega_{c_{rd}}^2 + \frac{1}{n^2}} \cdot \sqrt{\omega_{c_{rd}}^2 + 1}. \tag{8.6}$$

Using equation (8.6) and the values of ω_c given in Table 7.2, we compute the values of $K_{n_{rd}}$ for different values of congestion window W and present them in the third column of Table 8.1. From the data given in Table 8.1 we plot $K_{n_{rd}}$ versus W and $1/W$ in Figures 8.1 and 8.2, respectively, whereas $K_{n_{rd}} \cdot W^2$ versus W is plotted in Figure 8.3. Using MAPLE software we

W	$\omega_{c_{rd}}$	$K_{n_{rd}}$	$K_{n_{rd}} \cdot W^2$	$\hat{K}_{n_{rd}}$	%Error=$\frac{K_{n_{rd}} - \hat{K}_{n_{rd}}}{K_{n_{rd}}} \times 100$
2	0.620	8.699×10^{-1}	3.479	7.913×10^{-1}	+9.03
4	0.506	2.056×10^{-1}	3.289	2.087×10^{-1}	−1.50
8	0.406	5.391×10^{-2}	3.450	5.766×10^{-2}	−6.95
16	0.332	1.620×10^{-2}	4.414	1.720×10^{-2}	−6.17
32	0.282	5.632×10^{-3}	5.767	5.736×10^{-3}	−1.84
64	0.252	2.228×10^{-3}	9.125	2.195×10^{-3}	+1.48
128	0.235	9.714×10^{-4}	15.915	9.724×10^{-4}	−0.10

Table 8.1 Comparison between $K_{n_{rd}}$ and $\hat{K}_{n_{rd}}$ for different values of congestion window W.

perform least square quadratic curve fitting for $K_{n_{rd}} \cdot W^2$ versus W plot, shown in Figure 8.3, due to its linear characteristics. The general form of the equation for $K_{n_{rd}}$ is given by:

$$K_{n_{rd}} = \left(A_{ar} + \frac{B_{ar}}{W} + \frac{C_{ar}}{W^2}\right). \tag{8.7}$$

The values of constants A_{ar}, B_{ar} and C_{ar} in above equation (8.7) can be determined by using software package MAPLE [Maplesoft 1998]. It is found that $A_{ar} = 0.0001150$, $B_{ar} = 0.08638$ and $C_{ar} = 2.9922$. Thus, we will have following equation for K_{pn} after having least square quadratic curve fitting to data given in Table 8.1:

$$K_{n_{rd}} = 0.0001150 + \frac{0.08638}{W} + \frac{2.9922}{W^2}. \tag{8.8}$$

In order to check the accuracy of the curve fitting equation (8.8), we compute the estimated values of $K_{n_{rd}}$ by using (8.8), denoted by $\hat{K}_{n_{rd}}$ in the fifth column of Table 8.1, and compare

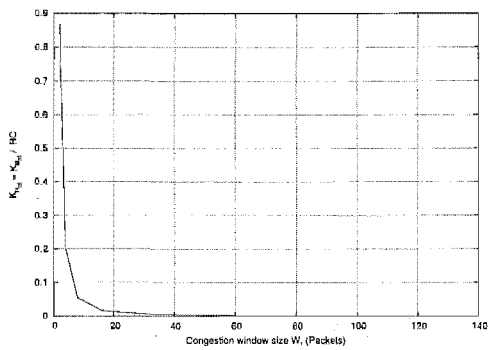

Figure 8.1 Variations in $K_{n_{rd}} = K_{u_{rd}} \cdot \left(\frac{RC}{n}\right)$ with congestion window ,W, for RED principle based adaptive feedback congestion control algorithm.

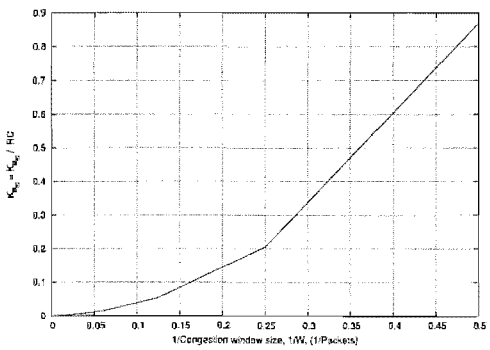

Figure 8.2 Variations in $K_{n_{rd}} = K_{u_{rd}} \cdot \left(\frac{RC}{n}\right)$ with reciprocal of congestion window ,$1/W$, for RED principle based adaptive feedback congestion control algorithm.

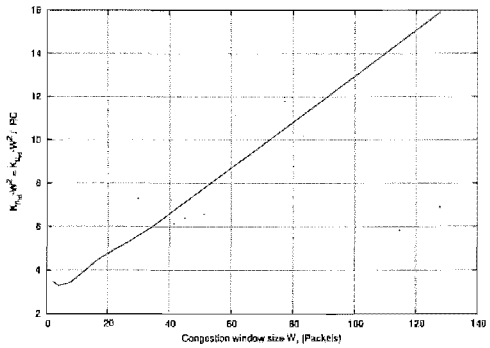

Figure 8.3 Variations in $K_{n_{rd}} \cdot W^2 = K_{u_{rd}} \cdot \left(\frac{RC}{n}\right) \cdot W^2$ with congestion window, W, for RED principle based adaptive feedback congestion control algorithm.

them with the actual values computed by (8.6), for different congestion window sizes. Clearly, the values of $K_{n_{rd}}$ and $\hat{K}_{n_{rd}}$ are close to each other for the normal operational range of values of W i.e. ($W = 32$ to $W = 64$ packets). Hence cubic or higher order expressions are not required for numerical solution of equation (8.6). Next, in order to get an expression for ultimate gain $K_{u_{rd}}$, we can rewrite equation (8.7) as:

$$K_{u_{rd}} = \frac{n}{RC} \cdot \left(A_{ar} + \frac{B_{ar}}{W} + \frac{C_{ar}}{W^2}\right). \tag{8.9}$$

As already given in Chapter 7, a general feedback congestion control algorithm can be obtained by having controller gain $K_{c_{rd}} = \frac{1}{\gamma_{rd}} \cdot K_{u_{rd}}$, where $\gamma_{rd} \geq 1$. Hence, from equation (8.7), we have the following expression for controller gain K_c:

$$K_{c_{rd}} = \frac{n}{\gamma_{rd} \cdot RC} \cdot \left(A_{ar} + \frac{B_{ar}}{W} + \frac{C_{ar}}{W^2}\right). \tag{8.10}$$

For the curve-fit expression given by equation (8.8) we can write the above equation (8.10) as:

$$K_{c_{rd}} = \frac{n}{\gamma_{rd} \cdot RC} \cdot \left(1.15 \times 10^{-4} + \frac{0.08638}{W} + \frac{2.9922}{W^2}\right). \tag{8.11}$$

Equating the gain of RED algorithm, L_{red}, as given by equation (6.7), to the controller gain K_c gives us the following expression:

$$K_{c_{rd}} = \frac{max_p}{max_{th} - min_{th}}, \tag{8.12}$$

from which we can get the following expression for max_p:

$$max_p = K_{c_{rd}} \cdot (max_{th} - min_{th}) . \tag{8.13}$$

Substituting the expression of K_c given in equation (8.10) into equation (8.13) gives:

$$max_p = (max_{th} - min_{th}) \cdot \left(\frac{n}{\gamma_{rd} \cdot RC}\right) \cdot \left(A_{ar} + \frac{B_{ar}}{W} + \frac{C_{ar}}{W^2}\right) . \tag{8.14}$$

The packet marking/dropping probability of RED in the region $\bar{q} \in [min_{th}, max_{th}]$ is given by equation (5.9) as:

$$p = \left(\frac{\bar{q} - min_{th}}{max_{th} - min_{th}}\right) \cdot max_p. \tag{8.15}$$

After substituting equation (8.14) into equation (8.15) we get the following adaptive, RED principle based, congestion control algorithm for AQM:

$$p \leftarrow (\bar{q} - min_{th}) \cdot \left(\frac{n}{\gamma_{rd} \cdot RC}\right) \cdot \left(A_{ar} + \frac{B_{ar}}{W} + \frac{C_{ar}}{W^2}\right) . \tag{8.16}$$

We can transform the above RED based adaptive control algorithm into a form which is function of the current packet marking/dropping probability p by substituting equation (8.3) into equation (8.16), i.e.

$$p \leftarrow (\bar{q} - min_{th}) \cdot \left(\frac{n}{\gamma_{rd} \cdot RC}\right) \cdot \left(A_{ar} + B_{ar} \cdot \sqrt{\frac{3}{8p}} + C_{ar} \cdot \frac{3}{8p}\right) . \tag{8.17}$$

It can be observed that the RED based adaptive algorithm given by equation (8.16) is in a general form through which different designs can be obtained by selecting different values of γ_{rd}. The values of γ_{rd} can be selected according to the desired stability margins of router. As in N-RED algorithms of Chapter 7, we will focus only on two versions of RED based adaptive congestion control algorithm which are obtained by substituting $\gamma_{rd} = 2$ and $\gamma_{rd} = 3$ in equation (8.16) and are named as AN-RED-1 and AN-RED-2, respectively. Next we present simulation results for these algorithms.

8.1.1 Simulations for RED Based Adaptive Feedback Congestion Control Algorithms

For the simulation study we use the network topology shown in Figure 7.3 which has $R = 0.246$ s, and $C = 3750$ packets/s. Further we choose $n = 10$, shown to be a good choice in Subsection 6.4.2 of Chapter 6. In order to compare simulation results with those obtained in Chapter 7, we select $min_{th} = 100$ and packets $max_{th} = 800$ packets. The queue weight, w_q, is obtained from equation (6.22) as 1.0840×10^{-4}. We run ns based simulations for 250 s and plot the resulting instantaneous queue size, EWMA queue size and packet marking/dropping probability. The expressions for the AN-RED-1 and AN-RED-2 algorithms are derived as follows.

AN-RED-1 Algorithm

For this algorithm we substitute $A_{ar} = 0.0001150$, $B_{ar} = 0.08638$ and $C_{ar} = 2.9922$, $R = 0.246$ s, $C = 3750$ packets/s and $\gamma_{rd} = 2$ in the general expression of RED based adaptive feedback congestion control algorithm given in equation (8.17) to get the following design of AN-RED-1 algorithm:

$$p \leftarrow (\bar{q} - min_{th}) \cdot \left(6.233 \times 10^{-7} + 2.868 \times 10^{-4} \cdot \sqrt{p} + 6.080 \times 10^{-3} \cdot p.\right) \qquad (8.18)$$

This algorithm ($\gamma_{rd} = 2$) follows the proportional controller design guideline as given in Table 7.1 and [Ziegler and Nichols 1942].

AN-RED-2 Algorithm

In this algorithm we substitute $\gamma_{rd} = 3$, which is two thirds of value for AN-RED-1 algorithm, along with other values as in AN-RED-1 algorithm in equation (8.17) to get the following expression for AN-RED-2 algorithm:

$$p \leftarrow (\bar{q} - min_{th}) \cdot \left(4.154 \times 10^{-7} + 1.911 \times 10^{-4} \cdot \sqrt{p} + 4.055 \times 10^{-3} \cdot p\right). \qquad (8.19)$$

This algorithm has a GM of 3 and is thus theoretically more stable than AN-RED-1 algorithm which has a GM of 2. For AN-RED-1 algorithm, given in equation (8.18), the instantaneous queue size, EWMA queue size and packet marking/dropping probability are given in Figures 8.4, 8.6 and 8.8, respectively. Similarly for AN-RED-2 algorithm, given in equation (8.19), the resulting instantaneous queue size, EWMA queue size and packet marking/dropping probability are plotted in Figures 8.5, 8.7 and 8.9, respectively.

8.1.2 Discussion of Simulations Results

In the previous Subsection we have presented two designs of RED based adaptive feedback congestion control algorithms, AN-RED-1 and AN-RED-2, for AQM routers supporting TCP/IP flows. A comparison of the simulation results is given in Table 8.2. The mean instantaneous queue size for AN-RED-1 algorithm (225 packets) is lower than that for AN-RED-2 algorithm

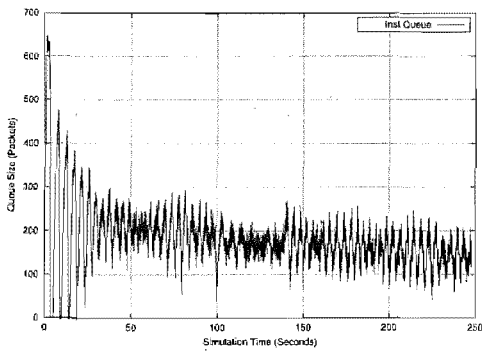

Figure 8.4 Instantaneous queue size for RED based adaptive feedback congestion controller design 1.

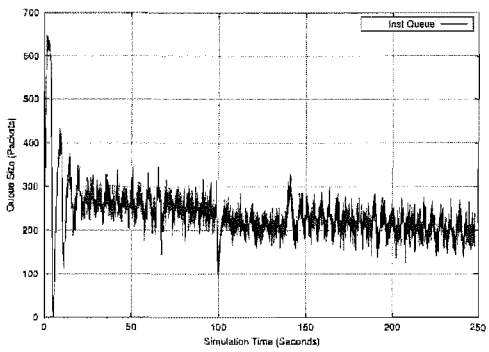

Figure 8.5 Instantaneous queue size for RED based adaptive feedback congestion controller design 2.

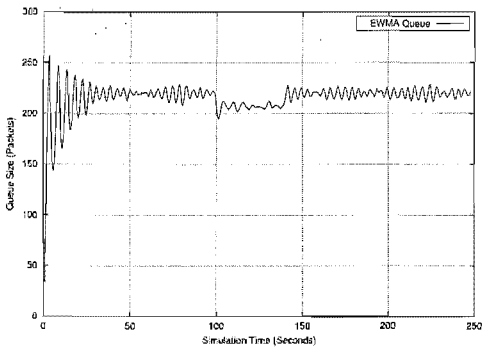

Figure 8.6 EWMA queue size for RED based adaptive feedback congestion control algorithm design 1.

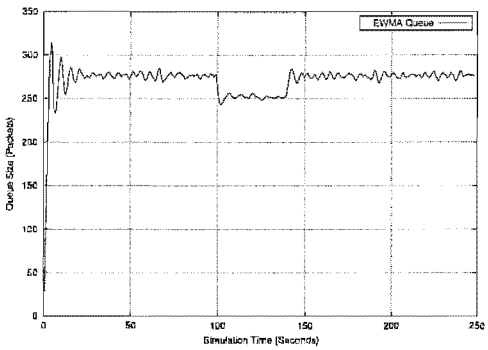

Figure 8.7 EWMA queue size for RED based adaptive feedback congestion control algorithm design 2.

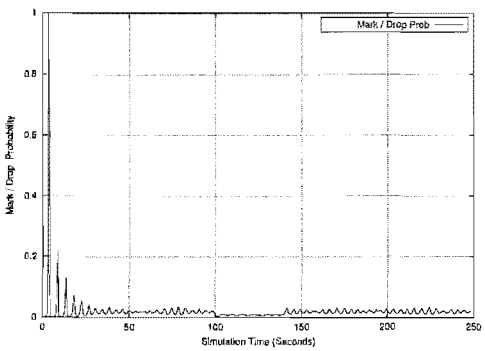

Figure 8.8 Mark/Drop probability for RED based adaptive feedback congestion control algorithm design 1.

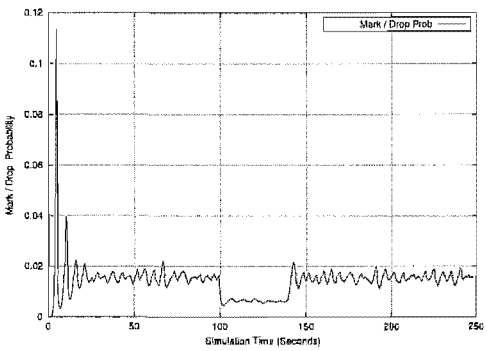

Figure 8.9 Mark/Drop probability for RED based adaptive feedback congestion control algorithm design 2.

(277 packets). Whereas the variance in instantaneous queue size of AN-RED-1 algorithm (3764) is higher than for AN-RED-2 algorithm (2638) as given in Table 8.2. It shows that AN-RED-2 algorithm has better convergence characteristics for instantaneous queue size (less variance) than AN-RED-1 algorithm. However, the mean EWMA queue size for AN-RED-1 algorithm is lower than for AN-RED-2 algorithm. Thus, compared to AN-RED-1, AN-RED-2 algorithm has higher steady state queue size leading to higher queuing delay but smaller variations in the instantaneous queue size. The smaller variance of the instantaneous queue size of AN-RED-2 algorithm is consistent with the results for N-RED-2 algorithm, developed in Chapter 7, where the $\gamma = 3$ design has better step response characteristics than the $\gamma = 2$ design.

Algorithm	$K_{c_{rd}}$	Instantaneous Queue		EWMA Queue, $w_q = 1.0840 \times 10^{-4}$	
		Arith Mean	Variance	Arith Mean	Variance
AN-RED-1	$0.50 \cdot K_{u_{rd}}$	225	3764	217	339
AN-RED-2	$0.33 \cdot K_{u_{rd}}$	277	2638	272	518

Table 8.2 A comparison between different designs of RED based adaptive feedback congestion control.

The speed of the transient response at start up and during traffic variations is also better for AN-RED-2 as compared to AN-RED-1 as evident from Figures 8.4 and 8.5. The packet marking/dropping probabilities of AN-RED-1 and AN-RED-2 are plotted in Figures 8.8 and 8.9, respectively. Notice that the former design has a higher packet marking/dropping probability during both high load and low load conditions. It can be explained by the fact that AN-RED-1 algorithm has lower gain margin as compared to AN-RED-2 algorithm. The packet marking/dropping probability of AN-RED-1 algorithm, as given in Figure 8.8, hits the maximum value of 1 at the starting transient thus causing loss of packets which consequently results in oscillations in the instantaneous queue size as shown in Figures 8.4. On the other hand the starting transient in packet marking/dropping probability of AN-RED-2 algorithm is small (around 0.115) thus causing lesser packets loss and lesser oscillations in the instantaneous queue size.

In order to observe the steady state characteristics of packet marking/dropping probability of AN-RED-1 and AN-RED-2 algorithms we neglect the first 50 s of simulation time and plot graphs in Figures 8.10 and 8.11 below:

From Figure 8.10 it can be observed that up to the simulation time of 100 s the packet marking/dropping probability of AN-RED-1 is centered around 0.02, during 100 s to 140 s (low load period) it is centered around 0.01 and after 140 s it is again centered around 0.02, approximately. Whereas Figure 8.11 suggests that for AN-RED-2 these values are 0.016, 0.006 and 0.016 for respective periods of time. Thus, AN-RED-2 design will mark/drop less packets as compared to AN-RED-1 both at starting and during normal operation and therefore leading to better utilization of resources.

In the expressions (8.18) and (8.19) the only variables are min_{th} and p, where the simulations showed that converged steady state queue size is not affected much by the changes in min_{th} due to the basic mechanism of the RED algorithm as given in equation (5.9) although it

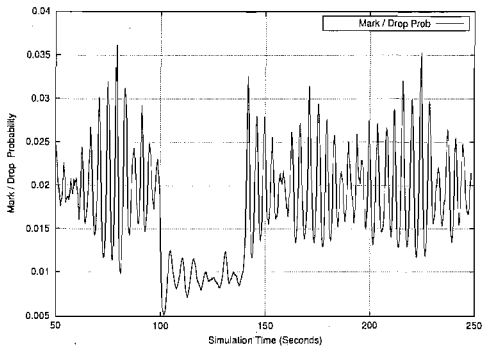

Figure 8.10 Steady state mark/drop probability for RED based adaptive feedback congestion control algorithm design 1.

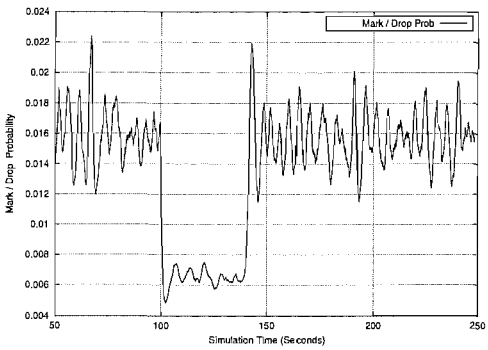

Figure 8.11 Steady state mark/drop probability for RED based adaptive feedback congestion control algorithm design 2.

is affected by varying the level of max_{th}. Furthermore, increasing or decreasing min_{th} will also alter the dynamic operating range, $(max_{th} - min_{th})$, of the underlying RED algorithm which will affect the overall operation of router based on AN-RED-1 or AN-RED-2 algorithm. Thus, it is difficult to vary the level of converged queue size by changing min_{th}.

But in order to determine the queuing delays we need to have an estimate of converged queue size as already described before in Chapter 7. We found that it is difficult to determine in advance the level of converged queue size both through N-RED algorithms described in Subsection 7.5.1 of Chapter 7 and their adaptive counterparts presented in this Chapter. Further comparing the queue variations characteristics of N-RED-1 and N-RED-2 as given in Table 7.4 with that of AN-RED-1 and AN-RED-2 as given in Table 8.2, we find that adaptive versions have higher level of queue as well higher variance. Thus, with reference to mean value of instantaneous and EWMA queue size and their variance we conclude that the adaptive algorithms underperform as compared to the previous non adaptive algorithms. One reason for it is that there is no single parameter which can effectively control queue level and intercoupling of all parameters $(min_{th}, max_{th}, max_p, w_q)$ in basic principle of RED type algorithms. Thus, for the RED based adaptive feedback congestion control algorithms controlling the level of converged queue size and reducing its variance is still an open research problem.

A comparison of queue convergence characteristics (to a given target level) of adaptive RED algorithms developed in this Chapter and Auto-Tuning RED algorithm developed in Chapter 6 reveals that former converges to desired target level, q_{ref}, of EWMA queue size in a nice fashion. Thus, it partially overcomes the inherent weakness of having no single parameter for controlling the converged queue (instantaneous or EWMA) size of RED type algorithms. The equation (6.12) suggests that the q_{ref} depends upon min_{th} and max_{th}, altering of which will effect the overall performance of RED and thus utilization of bottleneck bandwidth. Therefore the Auto-Tuning RED algorithm requires the proper selection of fixed parameters $(min_{th}, max_{th}, w_q)$ and optimization of tunable parameters ϵ and ζ. An incorrect setting of these parameters will lead to deterioration of the desired performance characteristics.

Hence, in order to overcome the high value of steady state queue length of RED based

adaptive feedback control algorithms (AN-RED-1 and AN-RED-2) and to have better control on converged queue size by least number of tunable parameters we investigate PI based adaptive congestion control algorithms in the next Section.

8.2 PI BASED ADAPTIVE FEEDBACK CONGESTION CONTROL ALGORITHM

In this Section we develop adaptive versions of the PI based feedback congestion control algorithms proposed in Chapter 7 for TCP/IP flows passing through AQM routers. A general PI based congestion control algorithm for packet marking/dropping probability in AQM routers has already been given by equation (7.37) i.e.

$$p(kT_s) = a \cdot [q(kT_s) - q_{ref}] - b \cdot [q\{(k-1)T_s\} - q_{ref}] + p\{(k-1)T_s\}, \tag{8.20}$$

where in equation (8.20) the parameters a and b have fixed designed values. However, the congestion control algorithm designed for certain specific number of TCP/IP connections will not necessarily work well for changing traffic. Therefore, our aim is to develop PI based adaptive feedback congestion control algorithms in which a and b should be able to adjust themselves according to changes in the load. Also there should be the least number of tuning parameters so that auto-tuning is easy to perform. To this end, we will modify the fixed parameter N-PI-1 and N-PI-2 algorithms developed in the Section 7.7.2 of Chapter 7 to make them adaptive. Since the comparison given in the Section 7.7.4 of Chapter 7 showed that step response and queue convergence characteristics of HO-PI algorithm are not good as compared to N-PI-1 and N-PI-2 algorithms thus we do not investigate HO-PI algorithm further in this Chapter. Replacing the parameters a and b (which are used for fixed number of TCP/IP connections) by new adaptive parameters α_{pi} and β_{pi} in congestion control algorithm equation (8.20), we get following expression at sampling time $T_{s_{pi}}$:

$$p(kT_{s_{pi}}) = \alpha \cdot \left[q(kT_{s_{pi}}) - q_{ref}\right] - \beta \cdot \left[q\{(k-1)T_{s_{pi}}\} - q_{ref}\right] + p\{(k-1)T_{s_{pi}}\}, \tag{8.21}$$

where q_{ref} in the above equation is a target value for queue size, i.e. the congestion control algorithm tries to keep instantaneous queue size close to q_{ref} as much as possible. In this Section we derive general equations for α_{pi} and β_{pi} and in the next subsections we derive adaptive expressions for ultimate gain $K_{u_{pi}}$, controller gain $K_{c_{pi}}$, and ultimate time $T_{u_{pi}}$. Let us consider the transfer function of a general PI controller as given in equation (7.71):

$$C_{N-PI}(s) = K_{c_{pi}} \cdot \left(1 + \frac{1}{s \cdot T_{i_{pi}}}\right). \tag{8.22}$$

In order to convert the continuous-time transfer function given in equation (8.22) to discrete time we use the Tustin transform as given in equation (7.44) of Chapter 7. The resulting z-domain transfer function is obtained as:

$$C_{N-PI}(z) = K_{c_{pi}} \cdot \left\{ \frac{z \cdot \left(\frac{T_{s_{pi}}+2T_{i_{pi}}}{2T_{i_{pi}}}\right) - \left(\frac{T_{s_{pi}}-2T_{i_{pi}}}{2T_{i_{pi}}}\right)}{z-1} \right\}. \tag{8.23}$$

The transfer function in equation (8.23) can be written in adaptive form as:

$$C_{AN-PI}(z) = \frac{\alpha_{pi} \cdot z - \beta_{pi}}{z-1}, \tag{8.24}$$

where α_{pi} and β_{pi} in equation (8.24) are given by:

$$\alpha_{pi} = K_{c_{pi}} \cdot \left(1 + 0.5 \cdot \frac{T_{s_{pi}}}{T_{i_{pi}}}\right), \tag{8.25}$$

$$\beta_{pi} = K_{c_{pi}} \cdot \left(1 - 0.5 \cdot \frac{T_{s_{pi}}}{T_{i_{pi}}}\right). \tag{8.26}$$

The expressions for α_{pi} and β_{pi}, as given by equations (8.25) and (8.26) respectively, indicate their dependence on controller gain $K_{c_{pi}}$ and controller time $T_{i_{pi}}$. Thus, the load adaptation of congestion control algorithm given by equation (8.21) depends upon adaptation of $K_{c_{pi}}$ and $T_{i_{pi}}$. In the following Subsection we develop expressions for the adaptive design of $K_{c_{pi}}$ and $T_{i_{pi}}$.

8.2.1 Derivation of Adaptive Gain for PI based Congestion Control Algorithm

In this Subsection we derive a load adaptive expression for the ultimate gain $K_{u_{pi}}$, through which we determine the expression for the adaptive gain $K_{c_{pi}}$ for PI based congestion control algorithm. These adaptive expressions for $K_{u_{pi}}$ and $K_{c_{pi}}$ will be used in the design of a PI based adaptive feedback congestion control algorithm in latter Sections. In Subsection 7.7.2.1, we obtained the following relation for $K_{u_{pi}}$,

$$K_{u_{pi}} = \frac{2N}{C^2} \cdot \sqrt{\omega_{u_{pi}}^2 + \left(\frac{2N}{R^2C}\right)^2} \cdot \sqrt{\omega_{u_{pi}}^2 + \frac{1}{R^2}}. \tag{8.27}$$

We can rewrite equation (8.27) as:

$$K_{u_{pi}} = \frac{2}{RC} \cdot \frac{N}{RC} \cdot \sqrt{(\omega_{u_{pi}} \cdot R)^2 + \left(\frac{2N}{RC}\right)^2} \cdot \sqrt{(\omega_{u_{pi}} \cdot R)^2 + 1}. \tag{8.28}$$

After defining $\omega_{c_{pi}} = \omega_{u_{pi}} \cdot R$ and substituting equation (6.9), i.e. $W = \frac{RC}{N}$, in equation (8.28) we get:

$$K_{u_{pi}} = \frac{2}{RC} \cdot \frac{1}{W} \cdot \sqrt{\omega_{c_{pi}}^2 + \left(\frac{2}{W}\right)^2} \cdot \sqrt{\omega_{c_{pi}}^2 + 1}. \tag{8.29}$$

After defining $K_{n_{pi}} = K_{u_{pi}} \cdot RC$ in above equation (8.29) we get the following expression:

$$K_{n_{pi}} = \frac{2}{W} \cdot \sqrt{\omega_{c_{pi}}^2 + \left(\frac{2}{W}\right)^2} \cdot \sqrt{\omega_{c_{pi}}^2 + 1}. \tag{8.30}$$

Using the values of $\omega_{c_{pi}}$ as given in Table 7.7, we iterate equation (8.30) for different values of congestion window, W, to compute the values of $K_{n_{pi}}$ which are given in Table 8.3. The vari-

W	$1/W$	$\omega_{c_{pi}}$	$K_{n_{pi}}$	$K_{n_{pi}} \cdot W^2$	$\hat{K}_{n_{pi}}$	%Error
2	0.5	1.306	2.705	10.820	2.636	+2.550
4	0.25	1.136	9.392×10^{-1}	15.027	9.406×10^{-1}	−0.149
8	0.125	1.017	3.734×10^{-1}	23.897	3.759×10^{-1}	−0.669
16	0.0625	0.945	1.639×10^{-1}	41.958	1.644×10^{-1}	−0.305
32	0.03125	0.904	7.634×10^{-2}	78.172	7.634×10^{-2}	0.0
64	0.015625	0.882	3.677×10^{-2}	150.609	3.674×10^{-2}	+0.081
128	0.0078125	0.871	1.805×10^{-2}	295.731	1.805×10^{-2}	0.0

Table 8.3 Data for least square curve fitting of $K_{n_{pi}} = K_{u_{pi}} \cdot RC$ for PI based adaptive feedback congestion controller.

ations in ultimate-gain bandwidth-delay product, $K_{n_{pi}} = K_{u_{pi}} \cdot RC$, with congestion window size, W, and its reciprocal $1/W$, as given in Table 8.3, are plotted in Figures 8.12 and 8.13, respectively. Also we have plotted $K_{n_{pi}} \cdot W^2$ versus congestion window size, W, in the Figure 8.14. Observing the Figures 8.13 and 8.14, we use $K_{n_{pi}} \cdot W^2$ versus W values in Table 8.3 for least square quadratic curve fitting by using MAPLE software package, [Maplesoft 1998]. The general form of the curve fitting equation can be written as:

$$K_{n_{pi}} \cdot W^2 = A_{ap} \cdot W^2 + B_{ap} \cdot W + C_{ap}, \tag{8.31}$$

which after substituting $K_{u_{pi}} = \frac{K_{n_{pi}}}{RC}$ gives the following expression for ultimate gain, $K_{u_{pi}}$:

$$K_{u_{pi}} = \frac{1}{RC} \cdot \left(A_{ap} + \frac{B_{ap}}{W} + \frac{C_{ap}}{W^2}\right), \tag{8.32}$$

where A_{ap}, B_{ap}, C_{ap} are constants, whose values are determined while curve fitting. For data given in Table 8.3 we determine these values as $A_{ap} = 9.525 \times 10^{-5}$, $B_{ap} = 2.251$ and $C_{ap} = 6.044$. A general class of PI based adaptive feedback control algorithms can be designed by defining $K_{c_{pi}} = \sigma_{pi} \cdot K_{u_{pi}}$, where $0 \leq \sigma_{pi} \leq 0.45$. Thus, we will have following general

expression of $K_{c_{pi}}$:

$$K_{c_{pi}} = \frac{\sigma_{pi}}{RC} \cdot \left(A_{ap} + \frac{B_{ap}}{W} + \frac{C_{ap}}{W^2} \right) \tag{8.33}$$

After substituting values of A_{ap}, B_{ap} and C_{ap} in equation (8.31) we get the following relation

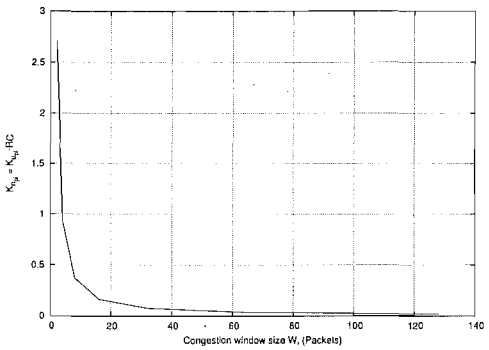

Figure 8.12 Variations in $K_{n_{pi}} = K_{u_{pi}} \cdot RC$ with congestion window, W, for PI based Adaptive feedback congestion controller.

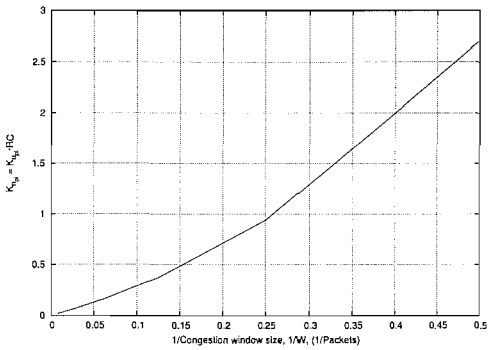

Figure 8.13 Variations in $K_{n_{pi}} = K_{u_{pi}} \cdot RC$ with reciprocal of congestion window, $1/W$, for PI based Adaptive feedback congestion controller.

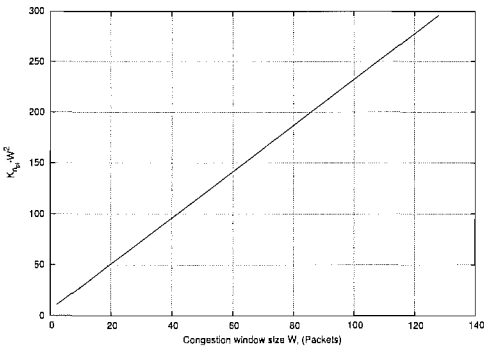

Figure 8.14 Variations in $K_{n_{pi}} \cdot W^2$ with congestion window, W, for PI based Adaptive feedback congestion controller.

between $K_{n_{pi}}$ and W:

$$K_{n_{pi}} = 9.525 \times 10^{-5} + \frac{2.251}{W} + \frac{6.044}{W^2}, \tag{8.34}$$

which gives the following expression for $K_{u_{pi}}$:

$$K_{u_{pi}} = \frac{1}{RC} \cdot \left(9.525 \times 10^{-5} + \frac{2.251}{W} + \frac{6.044}{W^2} \right). \tag{8.35}$$

In order to check the accuracy of equation (8.34), we evaluate $\hat{K}_{n_{pi}}$ as an approximation to $K_{n_{pi}}$ for different values of W and give its values in Table 8.3. The values of $K_{n_{pi}}$ and $\hat{K}_{n_{pi}}$ are quite close to each other with small amount of error. For ultimate gain expression given in equation

(8.35) we can have the following expression of $K_{c_{pi}}$:

$$K_c = \frac{\sigma_{pi}}{RC} \cdot \left(9.525 \times 10^{-5} + \frac{2.251}{W} + \frac{6.044}{W^2}\right). \tag{8.36}$$

The expressions of ultimate gain $K_{u_{pi}}$ and controller gain $K_{c_{pi}}$ as given by equation (8.35) and (8.36) respectively are function of congestion window, W, which are difficult to be implemented at router and thus need to be transformed in an alternative form. Hence, substituting equation (8.3) into equations (8.35) and (8.36) we get the following expressions of $K_{u_{pi}}$ and $K_{c_{pi}}$:

$$K_{u_{pi}} = \frac{1}{RC} \cdot \left(9.525 \times 10^{-5} + 1.379 \cdot \sqrt{p} + 2.267 \cdot p\right), \tag{8.37}$$

$$K_{c_{pi}} = \frac{\sigma_{pi}}{RC} \cdot \left(9.525 \times 10^{-5} + 1.379 \cdot \sqrt{p} + 2.267 \cdot p\right), \tag{8.38}$$

where in equations (8.37) and (8.38) the ultimate gain $K_{u_{pi}}$ and controller gain $K_{c_{pi}}$ have been expressed in terms of packet mark/drop probability, p, and bandwidth-delay product, RC (packets). These expressions will be further used in development of PI based congestion control algorithm. It can be observed that different values of σ_{pi} in equation (8.38) will give different designs of gain, $K_{c_{pi}}$, of congestion control algorithm. In this chapter we will concentrate only on two designs which are obtained for $\sigma_{pi} = 0.45$ and $\sigma_{pi} = 0.30$ where former is similar to PI controller design guidelines given in Table 7.1 and [Ziegler and Nichols 1942].

8.2.2 Derivation of Adaptive $T_{i_{pi}}$ for PI based Congestion Control Algorithm

The expression for ultimate frequency, $\omega_{u_{pi}}$, has already been determined in equation (7.68) of Chapter 7 as:

$$\omega_{u_{pi}} = \left(\frac{0.8884}{R \cdot W} + \frac{0.8827}{R}\right). \tag{8.39}$$

The corresponding ultimate time $T_{u_{pi}}$ is given by:

$$T_{u_{pi}} = \frac{2\pi}{\omega_{u_{pi}}}. \tag{8.40}$$

Substituting the value of $\omega_{u_{pi}}$ from (8.39) into (8.40) we get:

$$T_{u_{pi}} = \frac{2\pi R}{\left(\frac{0.8884}{W} + 0.8827\right)}. \tag{8.41}$$

Further substituting the expression of W from equation (8.3) into (8.41) we get:

$$T_{u_{pi}} = \frac{2\pi R}{0.544 \cdot \sqrt{p} + 0.882}. \tag{8.42}$$

The ultimate time $T_{u_{pi}}$ as given by equation (8.42) has been plotted as a function of packet marking/dropping probability, p, and round trip time, R, in Figure 8.15. Thus, for given values of R and p (thus congestion window) we can determine $T_{u_{pi}}$ from Figure 8.15. It shows that for smaller values of R the ultimate time of PI congestion control algorithm, $T_{u_{pi}}$, does not change much with changes in p, whereas for the smaller values of p it changes considerably with changes in R.

As already been shown in Figure 3.1 of Chapter 3, the packet marking/dropping probability p is small for the usual range of values of congestion window, W, of TCP, therefore we expect that $T_{u_{pi}}$ will vary considerably with changes in round trip time R in a network. For our simulated network topology, shown in Figure 7.3, the round trip time R is 0.246 s which after substitution into equation (8.42) gives the following expression of $T_{u_{pi}}$:

$$T_{u_{pi}} = \frac{1.545}{0.544 \cdot \sqrt{p} + 0.882}. \tag{8.43}$$

As given in Chapter 7, in general we can have integral or controller time $T_{i_{pi}} = \delta_{pi} \cdot T_{u_{pi}}$ where $\delta_{pi} \geq 0.85$. For PI based feedback congestion control algorithms $T_{i_{pi}}$ is selected as $0.85 \cdot T_{u_{pi}}$ as given in Table 7.1, i.e. $T_{i_{pi}} = 0.85 \cdot T_{u_{pi}}$, hence from equation (8.42) we get:

$$T_{i_{pi}} = \frac{1.7\pi R}{0.544 \cdot \sqrt{p} + 0.882}, \tag{8.44}$$

which is plotted in Figure 8.16. Our simulated network topology (R=0.246 s) gives the following expression of $T_{i_{pi}}$:

$$T_{i_{pi}} = \frac{1.313}{0.544 \cdot \sqrt{p} + 0.882}, \tag{8.45}$$

which leads to similar conclusion about variation in $T_{i_{pi}}$ as that of equation (8.42), plotted in Figure 8.15. Thus $T_{i_{pi}}$ varies considerably for the smaller values of p which is usually operating range of congestion window, W, of TCP/IP connections. Therefore, using Figures 3.1 (in Chapter 3), 8.15 and 8.16 we can graphically determine the value of $T_{i_{pi}}$ for given values of congestion window W and round trip time R.

8.2.3 Generalized Adaptive Gain and Adaptive $T_{i_{pi}}$

The adaptive formulae for $K_{u_{pi}}$, $K_{c_{pi}}$, $T_{u_{pi}}$ and $T_{i_{pi}}$ as presented in equations (8.37), (8.38), (8.42) and (8.44), respectively, depends upon packets mark/drop probability, p, which changes with variations in TCP/IP traffic load. These formulae can be further generalized using EWMA, $\bar{p}$, of mark/drop probability, p, which can be recursively determined as:

$$\bar{p} \leftarrow (1 - w_p) \cdot \bar{p} + w_p \cdot p, \tag{8.46}$$

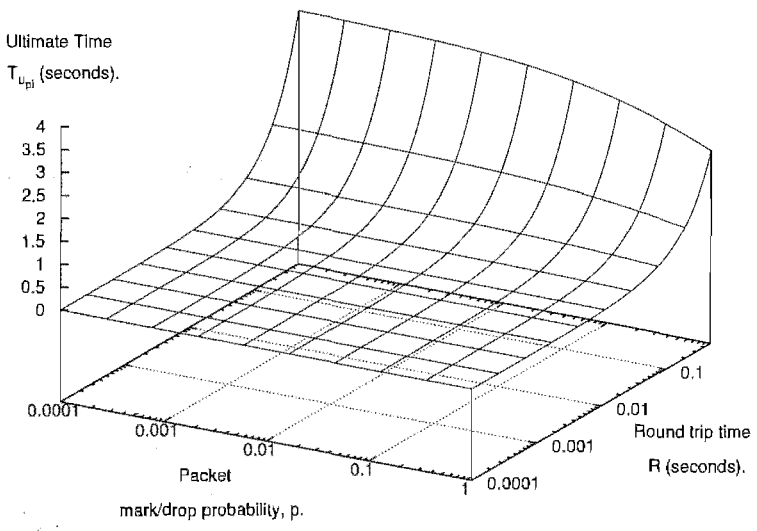

Figure 8.15 Variations in Ultimate Time $T_{u_{pi}}$ with packet mark/drop probability p and round trip time, R.

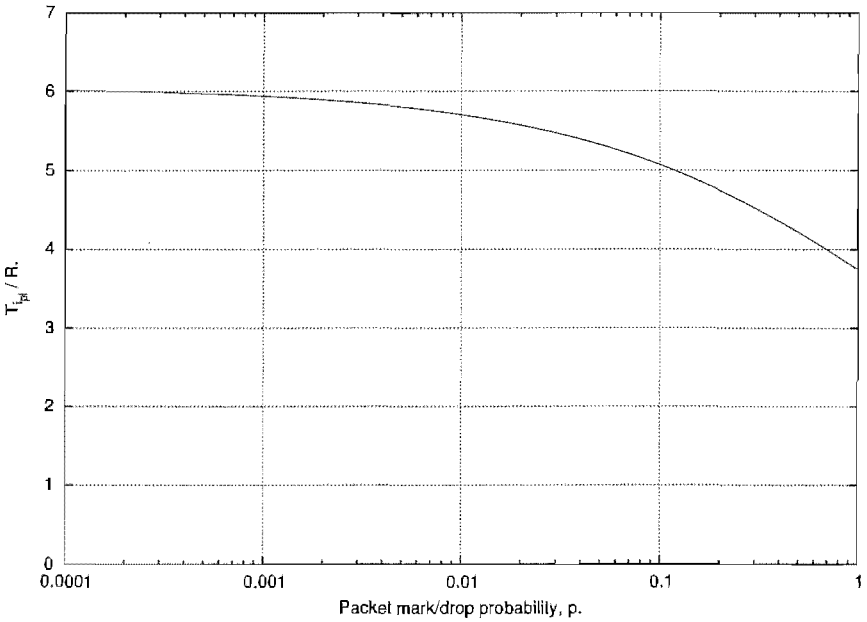

Figure 8.16 Variations of $\frac{T_{i_{pi}}}{R}$ with packet mark/drop probability p.

where in equation (8.46) w_p act as weight of the averaging filter. Next we generalize the concepts developed in previous two subsections by employing the filtered value of p, i.e. $\bar{p}$, to adapt K_c and T_u to traffic changes. Hence, after replacing p by $\bar{p}$ we will have following modified equations for adaptive $K_{u_{pi}}$, $K_{c_{pi}}$, $T_{u_{pi}}$ and $T_{i_{pi}}$:

$$K_{u_{pi}} = \frac{1}{RC} \cdot \left(9.525 \times 10^{-5} + 1.379 \cdot \sqrt{\bar{p}} + 2.267 \cdot \bar{p}\right), \tag{8.47}$$

$$K_{c_{pi}} = \frac{\sigma_{pi}}{RC} \cdot \left(9.525 \times 10^{-5} + 1.379 \cdot \sqrt{\bar{p}} + 2.267 \cdot \bar{p}\right), \tag{8.48}$$

$$T_{u_{pi}} = \frac{2\pi R}{0.544 \cdot \sqrt{\bar{p}} + 0.882}, \tag{8.49}$$

$$T_{i_{pi}} = \frac{1.7\pi R}{0.544 \cdot \sqrt{\bar{p}} + 0.882}. \tag{8.50}$$

In our simulations experiments, we vary w_p over a wide range of values i.e. from 1 to 2×10^{-5} to find optimum settings. The analytical expression for the optimum setting of the probability weight w_p is an open question. In the next Subsection we use equations (8.47), (8.48) and (8.49) to get a complete design of PI based adaptive feedback congestion control (AN-PI) algorithms.

8.2.4 Design of AN-PI Algorithms

In this Subsection we derive expressions for the α and β parameters for AN-PI algorithm as already presented by equation (8.21). We consider two designs of AN-PI algorithm which are the counterparts of the N-PI-1 and N-PI-2 algorithms developed in Chapter 7. These are obtained for $\{\sigma_{pi}, \delta_{pi}\} = \{0.45, 0.85\}$ and $\{\sigma_{pi}, \delta_{pi}\} = \{0.30, 0.85\}$ and are named as AN-PI-1 and AN-PI-2 algorithms, respectively, as given in Table 8.4. Substituting $K_{c_{pi}} = \sigma_{pi} \cdot K_{u_{pi}}$ and

Congestion Control Algorithms	Design Parameters			
	σ_{pi}	δ_{pi}	$K_{c_{pi}} = \sigma_{pi} \cdot K_{u_{pi}}$	$T_{i_{pi}} = \delta_{pi} \cdot T_{u_{pi}}$
AN-PI-1, [Astrom and Hagglund 1995]	0.45	0.85	$0.45 \cdot K_{u_{pi}}$	$0.85 \cdot T_u$
AN-PI-2	0.30	0.85	$0.30 \cdot K_{u_{pi}}$	$0.85 \cdot T_u$

Table 8.4 PI based Adaptive Feedback Congestion Control Algorithms design guide lines.

$T_{i_{pi}} = \delta_{pi} \cdot T_{u_{pi}}$ into equations (8.25) and (8.26) we get:

$$\alpha_{pi} = \sigma_{pi} \cdot \left(0.5 \cdot K_{u_{pi}} \cdot \frac{T_{s_{pi}}}{\delta_{pi} \cdot T_{u_{pi}}} + K_{u_{pi}}\right), \tag{8.51}$$

$$\beta_{pi} = \sigma_{pi} \cdot \left(0.5 \cdot K_{u_{pi}} \cdot \frac{T_{s_{pi}}}{\delta_{pi} \cdot T_{u_{pi}}} - K_{u_{pi}}\right). \tag{8.52}$$

As given in Table 8.4 to ensure good stability margins, we use $\delta_{pi} = 0.85$ for both AN-PI-1 and AN-PI-2 algorithms which is the same as recommended in [Ziegler and Nichols 1942]. In order to simulate the AN-PI-1 and AN-PI-2 algorithms the queue sampling time period, $T_{s_{pi}}$, as derived in equation (7.40), is given by:

$$T_{s_{pi}} \leq 0.05 \cdot \left(\pi \cdot \frac{R^2 \cdot C}{N}\right). \tag{8.53}$$

Thus, the minimum queue sampling frequency, $f_{s_{pi}}$, as derived in equation 7.42, is given by:

$$f_{s_{pi}} \geq 20 \cdot \left(\frac{N}{\pi \cdot R^2 \cdot C}\right). \tag{8.54}$$

In our simulation topology, shown in Figure 7.3, the maximum queue sampling time and minimum queue sampling frequency as given in equations (8.53) and (8.54) is 0.05941 s and 1.683 packets/s or Hz respectively. As already mentioned in the Section 7.7.1 of Chapter 7, we do oversampling of router's queue size by a factor of 100 giving $f_{s_{pi}}$=168.3 Hz. Thus, in our simulation experiments we select queue sampling frequency of 168 Hz. Whereas, in case of RED routers the minimum queue sampling frequency is the same as the link speed, which is the maximum value for AN-PI algorithm designs.

Next we determine expressions of α_{pi} and β_{pi} for AN-PI-1 and A-PI-2 algorithms.

- AN-PI-1 Congestion Control Algorithm

  Substituting the values $\sigma_{pi} = 0.45$, $\delta_{pi} = 0.85$ and $T_{s_{pi}} = 1/168$ s in equations (8.51) and (8.52) we get the following expressions for α_{pi} and β_{pi}:

$$\alpha_{pi} = 1.5756 \times 10^{-3} \cdot \left(\frac{K_{u_{pi}}}{T_u}\right) + K_{c_{pi}}, \tag{8.55}$$

$$\beta_{pi} = 1.5756 \times 10^{-3} \cdot \left(\frac{K_{u_{pi}}}{T_u}\right) - K_{c_{pi}}, \tag{8.56}$$

  where $K_{u_{pi}}$ and $T_{u_{pi}}$ are given by equations (8.47) and (8.49) respectively, and the expression for $K_{c_{pi}}$ can be obtained by substituting $\sigma_{pi} = 0.45$ into equation (8.48) to yield:

$$K_{c_{pi}} = \frac{1}{RC} \cdot \left(4.286 \times 10^{-5} + 0.620 \cdot \sqrt{\bar{p}} + 1.020 \cdot \bar{p}\right). \tag{8.57}$$

- AN-PI-2 Congestion Control Algorithm

  Similarly substituting the values $\sigma_{pi} = 0.30$, $\delta_{pi} = 0.85$ and $T_{s_{pi}} = 1/168$ s in equations (8.51) and (8.52) we get:

$$\alpha_{pi} = 1.0504 \times 10^{-3} \cdot \left(\frac{K_{u_{pi}}}{T_{u_{pi}}}\right) + K_{c_{pi}}, \tag{8.58}$$

$$\beta_{pi} = 1.0504 \times 10^{-3} \cdot \left(\frac{K_{u_{pi}}}{T_{u_{pi}}}\right) - K_{c_{pi}}. \tag{8.59}$$

In the above expressions $K_{u_{pi}}$ and $T_{u_{pi}}$ are given as before by equations (8.47) and (8.49) respectively, whereas expression for K_c can be obtained by substituting $\sigma_{pi} = 0.30$ into equation (8.48) which is given by:

$$K_{c_{pi}} = \frac{1}{RC} \cdot \left(2.857 \times 10^{-5} + 0.413 \cdot \sqrt{\bar{p}} + 0.680 \cdot \bar{p}\right). \tag{8.60}$$

The expressions of controller gain $K_{c_{pi}}$ as given in equations (8.57) and (8.60) are in generalised form and they can be simplified for given values of round trip time, R, and bottleneck link capacity, C. Furthermore, in equations (8.57) and (8.60) we can replace the packet marking probability, p, by its EWMA value, $\bar{p}$, to get a more general form of $K_{c_{pi}}$. In the next Subsection we simulate AN-PI-1 and AN-PI-2 congestion control algorithms as designed in this Subsection.

8.2.5 Simulations of AN-PI Algorithms

We use the network topology shown in Figure 7.3 for which $R = 0.246s$ and $C = 3750$ packets/s, giving bandwidth delay product of $RC = 922.5$ packets. In all simulation experiments, each TCP connection has target level for queue size as 200 packets and buffer size of 800 packets. There are no packet losses due to buffer overflow and thus the router marks/drops packets with a probability that is based on the AN-PI congestion control algorithms. The expressions for ultimate gain, $K_{u_{pi}}$, and ultimate time, $T_{u_{pi}}$, needed for both AN-PI-1 and AN-PI-2 algorithms can be obtained by substituting $RC = 922.5$ in equations (8.47) and (8.49) to get:

$$K_{u_{pi}} = 1.032 \times 10^{-7} + 1.494 \times 10^{-3} \cdot \sqrt{\bar{p}} + 2.457 \times 10^{-3} \cdot \bar{p}, \tag{8.61}$$

$$T_{u_{pi}} = \frac{1.545}{0.544 \cdot \sqrt{\bar{p}} + 0.882}. \tag{8.62}$$

Using the guidelines given in Table 8.4 and the expressions of $K_{u_{pi}}$ and $T_{u_{pi}}$ as given above by equations (8.61) and (8.62) we derive expressions of $K_{c_{pi}}$ and $T_{i_{pi}}$ to be used to compute α_{pi} and β_{pi} as given in equations (8.25) and (8.26) and latter modified in (8.51) and (8.52), respectively. These load adaptive values of parameters α_{pi} and β_{pi} are used to compute packet marking/dropping probability as given by equation (8.21).

To evaluate the performance of the adaptive congestion control algorithms for AQM we plot instantaneous queue size, q, and packet marking/dropping probability, p and EWMA of mark/drop probability, $\bar{p}$, (for different values of w_p) at the end of each simulation. The EWMA of packet marking/dropping probability, $\bar{p}$, for different values of w_p, will be used in as an indicator of change in TCP traffic load.

In the simulation setup we have a set of 40 ftp traffic sources which send data from 0 s until the end of simulation and an another set of 20 ftp sources which start sending data at

0 s and stop at 100 s and again re-start at 140 s and keep on sending data until the end of simulation, thus giving two transients in queue fluctuations i.e. a down transient at 100s and an up transient at 140 s. Also we have used 180 http traffic sources as background traffic. In the simulation experiments we compare the settling time of the initial starting transient, the downward transient and the up transient. In the following we present simulation results for PI based adaptive feedback congestion control algorithms AN-PI-1 and AN-PI-2.

AN-PI-1 congestion control algorithm:

In this case we simulate the congestion control algorithm as given by equation (8.21) with α_{pi} and β_{pi} as given by equation (8.55) and (8.56) respectively. For the AN-PI-1 algorithm we substitute $RC = 922.5$ in equation (8.57) to get the following expression for $K_{c_{pi}}$:

$$K_{c_{pi}} = 4.646 \times 10^{-8} + 6.727 \times 10^{-4} \cdot \sqrt{\bar{p}} + 1.105 \times 10^{-3} \cdot \bar{p}. \tag{8.63}$$

To compute the values of α_{pi} and β_{pi} as given in equations (8.55) and (8.56) we employ $K_{u_{pi}}$, $T_{u_{pi}}$ and $K_{c_{pi}}$ as given in equations (8.61), (8.62) and (8.63), respectively. Further, we use different values of weight w_p to compute EWMA of packet marking/dropping probability, $\bar{p}$. In the first set of experiments we use $w_p = 0.1, 0.01$ and 0.001 whereas in the second set of experiments we use $w_p = 1, 0.02$, 0.002 and 0.0002. For the latter case of w_p we show the queue variations and packet marking/dropping probability graphs as follows:

- AN-PI-1 with $w_p = 1$
  For this design, the instantaneous queue size is shown in Figure 8.17 whereas packet mark/drop probability is shown in Figure 8.18.
- AN-PI-1 with $w_p = 2 \times 10^{-2}$
  In this case, the instantaneous queue size is shown in Figure 8.19 whereas packet mark/drop probability and its EWMA are shown in Figures 8.20 and 8.21, respectively.
- AN-PI-1 with $w_p = 2 \times 10^{-4}$
  In this design, the instantaneous queue size is shown in Figure 8.22 whereas packet mark/drop probability and its EWMA are shown in Figures 8.23 and 8.24, respectively.

The arithmetic mean, variance and standard deviation of instantaneous queue size of AN-PI-1 algorithm are given in Tables 8.5 and 8.6.

Comparison of queue variation characteristics of AN-PI-1 algorithm as given in Tables 8.5 and 8.6 with that of N-PI-1 algorithm as given in Table 7.10 reveals that former has higher arithmetic mean and higher variance. Also we observe that as value of probability weighting factor w_p of AN-PI-1 algorithm is increased from 1 to 0.00002 the arithmetic mean becomes closer to the target queue level of 200 packets, whereas the variance increases. In case of $w_p = 1$ the arithmetic mean of AN-PI-1 is about 6 % more whereas variance is 10 % less than that of N-PI-1 algorithm. while in case of $w_p = 0.0002$ the arithmetic mean of AN-PI-1 is about 3 % more and variance is about 1.5 times more than that of N-PI-1 algorithm. Thus, the

AN-PI-1 algorithm slightly under performs as compared to N-PI-1 algorithm for arithmetic mean and variance characteristics. But AN-PI-1 algorithm does not requires manual tuning as in case of N-PI-1 algorithm. The response time to changes in traffic load of AN-PI-1

Algorithm	Probability weight, w_p	Instantaneous queue size		
		Arithmetic Mean	Variance	Standard Deviation
AN-PI-1	1	212	946	30.75
	0.02	220	1053	32.44
	0.002	209	1117	33.42
	0.0002	205	1644	40.54

Table 8.5 Queue convergence characteristics of AN-PI-1 algorithm.

Algorithm	Probability weight, w_p	Instantaneous Queue size		
		Arithmetic Mean	Variance	Standard Deviation
AN-PI-1	0.1	217	962	31.01
	0.01	217	1075	32.78
	0.001	207	1228	35.04

Table 8.6 Queue convergence characteristics of AN-PI-1 algorithm.

Probability weight, w_p	Transient at 0 s		Transient at 100 s		Transient at 140 s	
	Max q (Pkts)	Δt_{ss} (s)	Min q (Pkts)	Δt_{ss} (s)	Max q (Pkts)	Δt_{ss} (s)
1	488	0.66	63	1.44	329	0.51
0.02	497	0.74	94	0.68	343	0.80
0.002	540	1.59	86	1.15	332	0.53
0.0002	599	4.78	85	1.20	333	1.61

Table 8.7 Response time characteristics of AN-PI-1 algorithm with changes in traffic load.

Probability weight, w_p	Transient at 0 s		Transient at 100 s		Transient at 140 s	
	Max q (Pkts)	Δt_{ss} (s)	Min q (Pkts)	Δt_{ss} (s)	Max q (Pkts)	Δt_{ss} (s)
0.1	488	1.80	59	1.28	316	1.25
0.01	519	0.96	73	1.00	339	0.96
0.001	557	6.65	51	1.44	301	0.47

Table 8.8 Response time characteristics of AN-PI-1 algorithm with changes in traffic load.

algorithm are presented in Tables 8.7 and 8.8. We observe three transients which occur at 0 s, 100 s and 140 s. We determine the time, Δt_{ss}, required by instantaneous queue size to reach the target queue level of 200 packets for first time after change in TCP/IP traffic. It can be seen from Tables 8.7 and 8.8 that as w_p is increased the Δt_{ss} increases for all three transients because lower value of w_p makes the system sluggish.

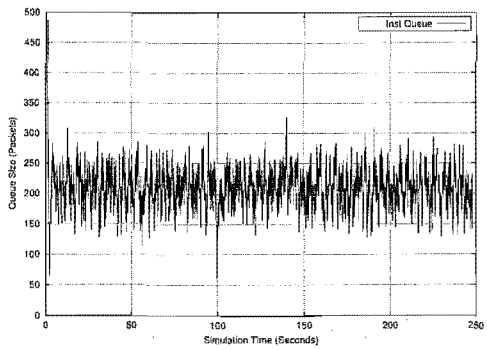

Figure 8.17 Instantaneous queue size of AN-PI-1 algorithm with $w_p = 1$.

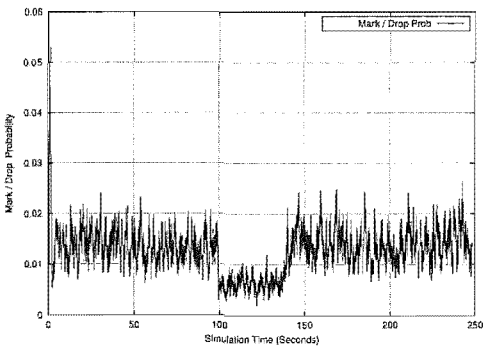

Figure 8.18 Packet mark/drop probability of AN-PI-1 algorithm with $w_p = 1$.

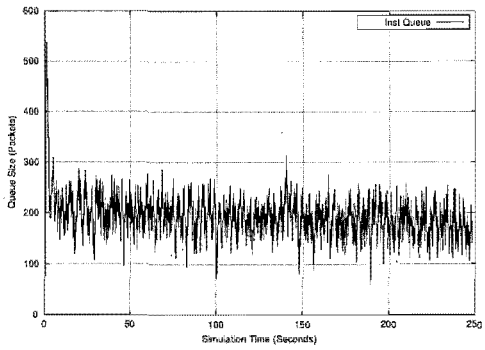

Figure 8.19 Instantaneous queue size of AN-PI-1 algorithm with $w_p = 0.002$.

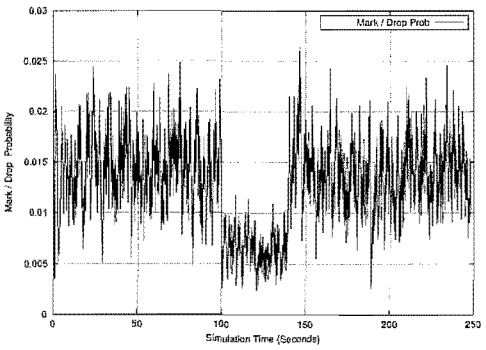

Figure 8.20 Packet mark/drop probability of AN-PI-1 algorithm with $w_p = 0.002$.

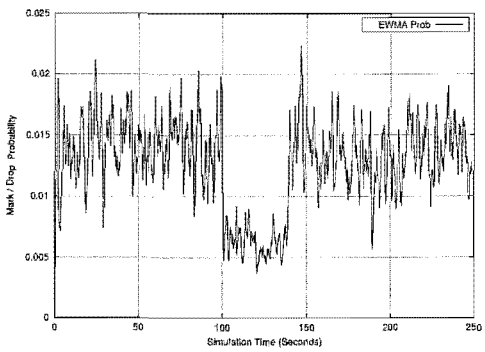

Figure 8.21 EWMA of packet mark/drop probability of AN-PI-1 algorithm with $w_p = 0.002$.

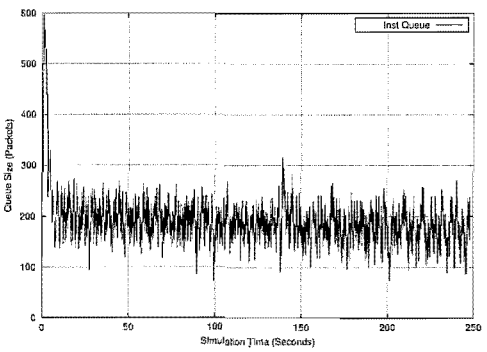

Figure 8.22 Instantaneous queue size of AN-PI-1 algorithm with $w_p = 0.0002$.

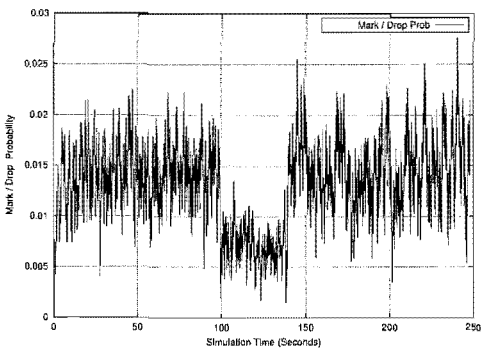

Figure 8.23 Packet mark/drop probability of AN-PI-1 algorithm with $w_p = 0.0002$.

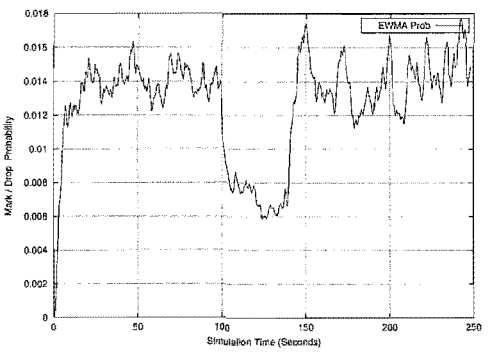

Figure 8.24 EWMA of packet mark/drop probability of AN-PI-1 algorithm with $w_p = 0.0002$.

AN-PI-2 congestion control algorithm:

For AN-PI-2 congestion control algorithm we use equation (8.21) with the values of α_{pi} and β_{pi} parameters as given by equations (8.58) and (8.59) respectively. The expression for control gain, $K_{c_{pi}}$, is obtained by substituting $RC = 922.5$ packets in equation (8.60) to give:

$$K_{c_{pi}} = 3.097 \times 10^{-8} + 4.484 \times 10^{-4} \cdot \sqrt{\bar{p}} + 7.372 \times 10^{-4} \cdot \bar{p}. \tag{8.64}$$

In order to compute the values of α_{pi} and β_{pi} parameters as given by equations (8.58) and (8.59) we employ equations (8.47), (8.49) and (8.64) to compute $K_{u_{pi}}$, $T_{u_{pi}}$ and $K_{c_{pi}}$, respectively. As in the case of AN-PI-1 we use different values of probability weight factor, w_p, to compute the EWMA of packet mark/drop probability: $w_p = 0.1, 0.01$ and 0.001 in the first set of experiments and $w_p = 1, 0.002$ and 0.0002 in the second set of experiments. For the latter case of w_p we show the queue variations and packet marking/dropping probability graphs as follows:

- AN-PI-2 with $w_p = 1$
  For $w_p = 1$, there is no filtering of mark/drop probability p i.e. we have $\bar{p} = p$ as given by equation (8.46). The instantaneous queue size is shown in Figure 8.25 whereas packet mark/drop probability is shown in Figure 8.26.

- AN-PI-2 with $w_p = 2 \times 10^{-3}$
  The instantaneous queue size is shown in Figure 8.27 whereas packet mark/drop probability and its EWMA are shown in Figures 8.28 and 8.29, respectively.

- AN-PI-2 with $w_p = 2 \times 10^{-4}$
  For this design, the instantaneous queue size is shown in Figure 8.30 whereas packet mark/drop probability and its EWMA are shown in Figures 8.31 and 8.32, respectively.

The arithmetic mean, variance and standard deviation of instantaneous queue size of AN-PI-2 algorithm are given in Tables 8.9 and 8.10. Comparison of queue variations characteristics of of AN-PI-2 algorithm as given in Tables 8.9 and 8.10 with that of N-PI-2 algorithm as given in Table 7.10 shows that former has higher higher arithmetic and variance as in previous case of AN-PI-1 algorithm.

In Table 8.9 we observe that as weight factor, w_p, of EWMA probability is increased from lower towards higher values the arithmetic mean of instantaneous queue size of AN-PI-2 algorithm come closer to target queue level of 200 packets. Also the variance of instantaneous queue size decreases with an increase in w_p. This observation for mean queue size is in contrast to AN-PI-1 algorithm characteristics given in Table 8.5 which showed the opposite trends.

Comparing arithmetic mean and variance characteristics of N-PI-2 algorithm given in Table 7.10 with that for AN-PI-2 algorithm as given in Table 8.9 show that AN-PI-2 algorithm underperforms as compared to N-PI-2 algorithm. The response time of AN-PI-2 algorithm to the changes in TCP/IP traffic load are given in Table 8.11. It shows that response time, Δt_{ss},

improves as we increase w_p from lower values towards 1. This observation of a decrease in response time with an increase in w_p is the same as in the case of previous AN-PI-1 algorithm.

Mutual comparison of AN-PI-1 and AN-PI-2 algorithms reveals that the former has slightly better queue variations and response time characteristics. Whereas in case of N-PI-1 and N-PI-2 algorithms, developed in Chapter 7, the difference in queue performance was less marked. Also we observe that packets mark/drop probability of both AN-PI-1 and AN-PI-2 algorithms reaches nearly same maximum value of around 0.014 which is quite low and it agrees with our design requirement of low packet loss. Thus both congestion control algorithms effectively controls the packet loss but AN-PI-2 has slower response time and slightly higher queue size as compared to AN-PI-1 algorithm.

Controller	Probability weight, w_p	Instantaneous Queue		
		Arith Mean	Variance	Standard Deviation
AN-PI-2	1	212	3821	61.81
	0.02	213	3780	61.48
	0.002	211	3930	62.68
	0.0002	215	4879	69.84

Table 8.9 Queue convergence characteristics of AN-PI-2 algorithm.

Controller	Probability weight, w_p	Instantaneous Queue		
		Arith Mean	Variance	Standard Deviation
AN-PI-2	0.1	212	3575	59.79
	0.01	213	3651	60.42
	0.001	212	4422	66.49

Table 8.10 Queue convergence characteristics of AN-PI-2 algorithm.

Probability weight, w_p	Transient at 0 s		Transient at 100 s		Transient at 140 s	
	Max q (Pkts)	Δt_{ss} (s)	Min q (Pkts)	Δt_{ss} (s)	Max q (Pkts)	Δt_{ss} (s)
1	616	8.72	47	5.77	380	7.37
0.02	616	8.74	61	1.91	387	9.42
0.002	617	10	68	6.68	379	1.58

Table 8.11 Response time characteristics of AN-PI-2 algorithm with changes in traffic load.

8.3 CONCLUSIONS

In this Chapter we have proposed two types of adaptive feedback congestion control algorithms for AQM which are based on RED and PI principles respectively. These algorithms adapt themselves to changes in traffic load and require minimal tuning. We have presented general algo-

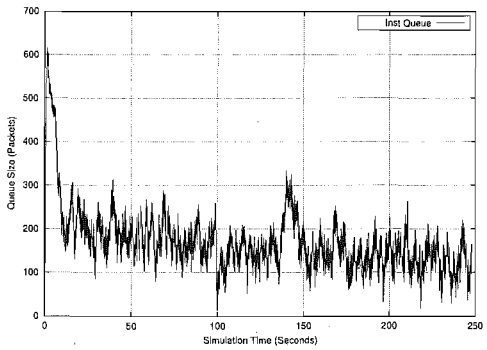

Figure 8.25 Instantaneous queue size of AN-PI-2 algorithm with $w_p = 1$.

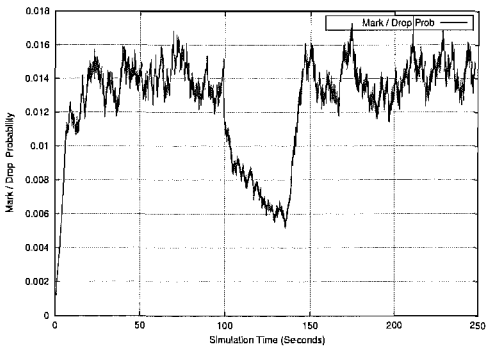

Figure 8.26 Packet mark/drop probability of AN-PI-2 algorithm with $w_p = 1$.

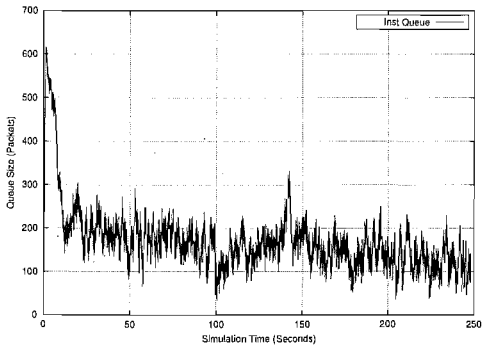

Figure 8.27 Instantaneous queue size of AN-PI-2 algorithm with $w_p = 0.002$.

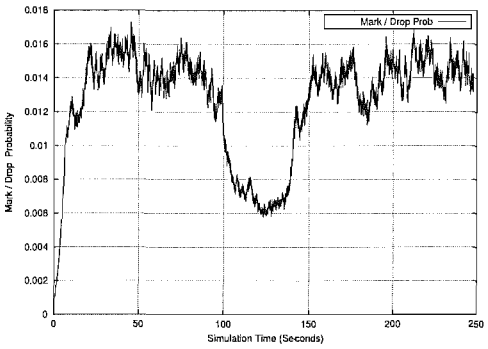

Figure 8.28 Packet mark/drop probability of AN-PI-2 algorithm with $w_p = 0.002$.

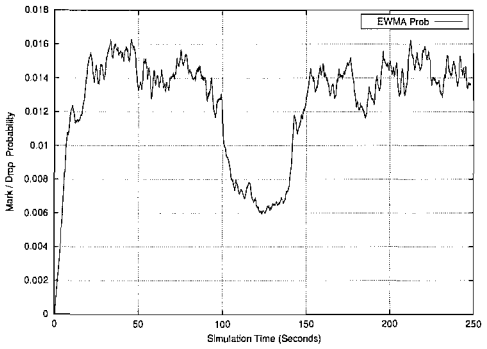

Figure 8.29 EWMA of packet mark/drop probability of AN-PI-2 algorithm with $w_p = 0.002$.

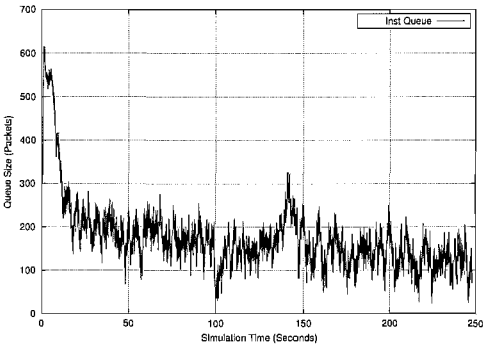

Figure 8.30 Instantaneous queue size of AN-PI-2 algorithm with $w_p = 0.0002$.

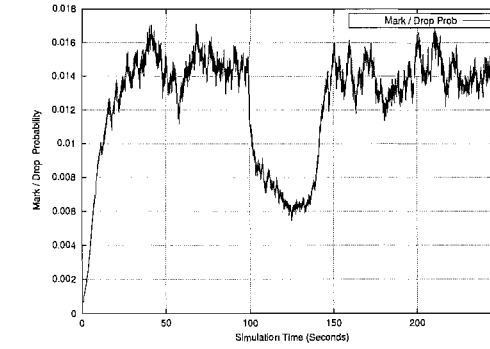

Figure 8.31 Packet mark/drop probability of AN-PI-2 algorithm with $w_p = 0.0002$.

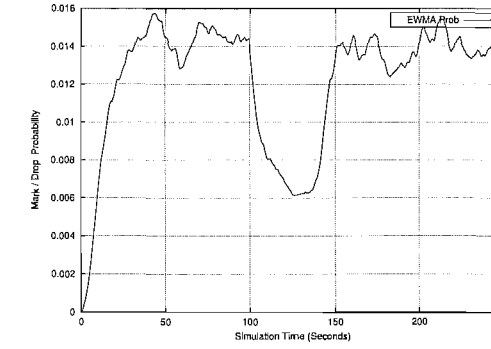

Figure 8.32 EWMA of packet mark/drop probability of AN-PI-2 algorithm with $w_p = 0.0002$.

Probability weight, w_p	Transient at 0 s		Transient at 100 s		Transient at 140 s	
	Max q (Pkts)	Δt_{ss} (s)	Min q (Pkts)	Δt_{ss} (s)	Max q (Pkts)	Δt_{ss} (s)
0.1	616	8.74	68	5.86	327	5.36
0.01	616	10.71	38	5.93	391	4.85
0.001	617	10.87	76	5.65	390	5.33

Table 8.12 Response time characteristics of AN-PI-2 algorithm with changes in traffic load.

rithms which can be tailored according to the desired accuracy and network scenarios. For each type we have simulated two designs and the results are compared with each other and with the corresponding fixed parameter algorithms developed in the previous Chapter, which however assume that the number of TCP connections is known.

The basis of the adaptive design of adaptive congestion control algorithms for AQM in Chapter 8 is square root of p formula, which in itself has approximations and consequent limitations. The square root of p formula has been derived for steady state behavior of TCP Reno running always under congestion avoidance phase and having random packet loss with constant probability. Also in deriving this formula the effects of slow start, fast retransmit, fast recovery and time outs have been omitted. Though controllers designed by using this formula will work excellently under most operating conditions and with most of TCP traffic conditions but they will not be 100% accurate due to approximations made in the square root formula. Another approach might be to use the bandwidth formulae given by [Padhye *et al.* 2000], but it seems that resulting formulae would be too complicated and more algebraic approximations would be needed. Also the Padhye formulae have approximations in derivation such as neglecting fast recovery phase, if a packet is lost in then all remaining packets transmitted in that round trip time are also lost. Thus for practical purposes square root p formula given in [Mathis *et al.* 1997] can be utilized.

The performance of adaptive feedback congestion control algorithms presented in this Chapter is slightly lower than that of their non-adaptive counterparts presented in the previous Chapter. We focused on the queue convergence to target level, marking/dropping probability level and speed of response to sudden change in traffic load. We implemented these controllers using EWMA of packet marking/dropping probability with different weights. It was confirmed that packet marking/dropping probability and its EWMA in general could be effectively used to estimate the change in traffic load on router and thus could be employed to auto-tune its congestion control algorithms. We found that as the weight of the probability filter increases from lower values towards 1 the speed of response improves. This is in agreement with theory of exponential weighted moving average processes as given in [Young 1984]. All of the congestion control algorithms have good convergence to target level queue size and have low probability of packet mark/drop. Filtering of packet mark/drop probability, p, was attempted because it was conjectured that EWMA would make adaptation more stable by reducing the random fluctuations in p. However, the simulation results showed that EWMA filtering of p actually degrades

performance of AQM based on AN-PI algorithms, perhaps because of the time delay it introduces.

In summary, in this Chapter we have been able to show that it is possible to develop adaptive feedback congestion control algorithms for TCP/IP based AQM by using fluid models in end-to-end architecture without requiring extra feedback information from the routers. The use of such adaptive feedback control algorithms will make it easier to implement AQM in the packet switched data networks and the Internet. Finally our proposed algorithms will preserve the existing architecture of TCP/IP based computer networks and are expected to improve the overall performance.

Chapter 9

CONCLUSIONS AND FUTURE WORK

In this Chapter we present the main contributions of this thesis and some suggestions for future work.

9.1 CONCLUSIONS

We considered the crucial problem of congestion control in present and future IP networks, including the Internet. We concentrated on two main strategies for controlling congestion in computer networks: congestion control through algorithms embedded in the end hosts' transport protocol and congestion control through router algorithms.

After introducing and motivating the problem in Chapter 1, a review of congestion control mechanisms in the main versions of the predominant transport protocol, TCP, was presented in Chapter 2. In particular, we analyzed the bias of TCP Reno against connections with longer round trip times. This well known phenomenon was also demonstrated in the next Chapter through simulations. We also briefly overviewed TCP AIMD, self-similarity in teletraffic and Internet 2. Different mathematical models of TCP, HighSpeed TCP for high bandwidth delay product networks, short TCP connections modelling and ECN were also discussed. Finally, different metrics for the performance evaluation of TCP are defined. The material presented in Chapter 2 served as the background for research work presented in the proceeding Chapters of this thesis.

In Chapter 3, we selected TCP Vegas, out of the several versions of TCP, for further research. The selection was due to better fairness properties of TCP Vegas as compared to the commonly used TCP Reno. After presenting an evolutionary history of proactive congestion control in end host protocols, we analyzed the heterogenous scenario in which both TCP Vegas and TCP Reno are competing with each other for the bandwidth of a bottleneck link. In our analysis and simulation experiments we considered a symmetric case of TCP Vegas, where AIAD mechanism is goverened by $\alpha_v = \beta_v$, for better fairness characteristics. First we analyzed the case in which both TCP Reno and TCP Vegas have the same round trip times and then we formulated a generalized and non-trivial problem of finding the Jain's Fairness Index, JFI, for a set of TCP Reno and TCP Vegas sources having different round trip times.

Next we presented a simulation based comparison between two TCP Reno sources (homogenous scenario) having the same and different round trip times. Similar simulation based comparison between TCP Vegas sources was also carried out. It was confirmed that TCP Reno sources show bias against connections with longer round trip times whereas such bias is not present among TCP Vegas sources. Then, we considered a heterogenous scenario in which both TCP Reno and TCP Vegas, having the same round trip times, compete with each other. We investigated the effects of changing α_v and β_v parameters of TCP Vegas on its congestion window dynamics, fairness (as measured by the JFI) and throughput.

We found that the throughput of TCP Vegas can be improved by increasing its α_v and β_v parameters in heterogenous environments. The overall JFI among the TCP Reno and TCP Vegas sources can also be improved until a certain value of α_v and β_v beyond which it starts decreasing because of TCP Vegas getting more bandwidth share than TCP Reno. Therefore, after ascertaining the improvements in throughput (thus fairness or JFI) of TCP Vegas by increasing its α_v and β_v parameters, in heterogenous environments, we employed RED based ECN to vary these parameters in response to changes in the traffic load at the router. Thereby we showed that TCP Vegas can be made compatible with TCP Reno by employing RED routers with ECN.

In Chapter 4 we considered the problem of the optimization of TCP congestion control mechanisms and presented two new algorithms, namely optimal minimum variance window control and generalized optimal minimum variance window control. First we presented an AR model of dynamic bandwidth share of a single TCP connection and then we obtained expressions for queue buffer dynamics. Afterward we derived the optimal minimum variance window control algorithm which was later simulated for the same and different values of q_{ref}, with five TCP sources having different round trip times. We also employed ON/OFF traffic with Pareto distribution to simulate self-similar traffic. It was found that for the same value of q_{ref} the bandwidth share among different TCP sources having different round trip times is nearly the same. It was also concluded that for different values of q_{ref} the bandwidth distribution is proportion to q_{ref}. For the case of self-similar traffic we found that the average queue variance decreases with an increase in the Hurst parameter.

Next we derived the generalized optimal minimum variance window control algorithm. We investigated its relation to $(p,\ 1)$ proportionally fair algorithm and determined conditions of its stable operation. Next we explored its steady state and transient fairness and introduced the concept of short term fairness index. Further we simulated the generalized algorithm and explored its fairness characteristics by changing its θ and q_{ref} parameters. It was found that for higher values of q_{ref} the value of θ should be small for smaller variance in congestion window sizes. However, for smaller variations in queue size the higher values of θ are preferred. It was concluded that the generalized algorithm can be operated with the desired characteristics by proper tuning of its parameters.

In the latter part of this thesis we focused our attention upon router based congestion control algorithms. To that end, we did a broad survey of different router based congestion control

algorithms in Chapter 5 with particular focus and emphasis on the RED algorithm which is currently being proposed for implementing AQM. We also did a comparative study of different router algorithms. We developed a new Hybrid RED algorithm which improves the loss rate and link utilization as compared to the basic RED algorithm. The Hybrid RED algorithm employs both instantaneous and EWMA queue sizes for making decisions on whether to mark/drop the incoming packets. A key feature of Hybrid RED is that it decreases the EWMA queue size by a factor ξ if the instantaneous queue size is below min_{th} for a specified number θ, of consecutive packet arrivals.

The background knowledge about the RED principle gained in Chapter 5 was used to design a new algorithm, named Auto-Tuning RED, in Chapter 6. It aimed at keeping the EWMA queue size to a desired reference level, q_{ref}, despite changes in the incoming traffic. It was designed by using an existing linearized model of AQM, and its queue convergence properties were found to compare favourably with those of another adaptive RED algorithm. We also gave the guidelines for settings of the fixed parameters of RED type algorithms.

In general, a badly tuned RED type algorithm will degrade the AQM performance. In Chapter 7, we employed feedback control theory to design some novel tuning techniques for AQM. The main method developed is a generalization of the frequency response method used for the empirical tuning of industrial process controllers. We developed a new RED principle based algorithm, N-RED, a Proportional Integral congestion control algorithm, N-PI, and a Proportional Integral Derivative congestion control algorithm, N-PID. All of the algorithms were simulated by using the ns simulator in the same network topology for uniformity of comparison. They were evaluated in terms of instantaneous queue variations, EWMA queue variations, stability margins and step responses.

We found that the frequency response method based new designs of RED and PI principle based algorithms for AQM, i.e. N-RED and N-PI, perform better than existing designs. The PI based algorithms followed the setpoint or target level of queue size, q_{ref}, quite accurately and thus can be used to estimate the queuing delays which would be useful for providing quality of service and estimating round trip times at end hosts. The RED based N-RED algorithms have higher stability margins but slower step response characteristics as compared to the PI based N-PI algorithms. Furthermore, we found that the PID based control algorithms for AQM have both high stability margins and fast step response characteristics. Thus, we concluded that PI based, and particularly PID based, congestion control algorithms are superior over RED type algorithms, in terms of queue convergence characteristics.

Towards the end of Chapter 7 we explored the effects of non-responsive flows traversing through AQM routers. We performed simulations to determine the bandwidth sharing among TCP flows and aggressive UDP flows which do not decrease their sending rates during congestion. We found that UDP flows grab the major share of the bandwidth. We modified our feedback congestion control algorithms by using the CHOKe principle at the front end so that they could cope with non-responsive flows as well. Our simulations results proved that CHOKe

algorithm can be effectively coupled with feedback congestion control algorithms for AQM to control their aggressive bandwidth grabbing. However, our new cascaded structures (CHOKe + N-RED, CHOKe + N-PI, CHOKe + N-PID) were not analyzed mathematically; this is left for future work.

We noticed that the parameters a and b of PI based congestion control algorithm and the parameters of PID algorithms need re-tuning when the number of TCP/IP connections is changed. Thus, underlining the need of adaptive or self-tuned feedback congestion control algorithms. Hence, in Chapter 8, we developed load adaptive feedback congestion control algorithms for AQM by modifying the algorithms developed in Chapter 7. We used control the theory to derive expressions for the ultimate gain margin and ultimate time for the AQM system. We then transformed the expression of controller gain K_c as a function of the congestion window to a function of the packet mark/drop probability, a quantity that is available to the router. Thereby we developed adaptive RED, and adaptive PI congestion control algorithms for AQM. We implemented these adaptive control algorithms in ns simulator and ran simulations on the same topology used previously in Chapter 7 to confirm their stability and performance.

In summary, in this thesis we have designed new algorithms for congestion control in present and future high speed networks. We designed fair and better congestion control algorithms to be used in protocols employed at end hosts as well we presented new congestion control algorithms to be used at congested routers for implementing AQM. The main aim of our research work was to develop new congestion control techniques for packet switched data networks and the Internet while remaining within the present end-to-end architecture of TCP/IP, augmented by congestion feedback from AQM routers to end hosts by packet dropping or ECN based marking. Thus, all of the algorithms developed in this thesis can be easily implemented in real networks. Our research has strengthened our belief that both end hosts and routers should take part in avoiding congestion control in present and future networks. Concentrating only on one side i.e. either end host protocols or on routers will not give optimized performance. We aimed to keep a balance between these two techniques of managing congestion control so as to get the best of both worlds.

9.2 FUTURE WORK

The field of congestion control is still wide open and a number of problems and issues still remain to be investigated. In this section we will present some of those problems which we have encountered during our research. In general we feel that almost every Chapter in this thesis can potentially open up a whole new field of research in the future.

In Chapter 3 we used the ECN mechanism to make TCP Vegas compatible with TCP Reno; this technique needs to be refined further as ECN itself is not without problems, some of which have already been mentioned in Chapter 2. Also, rigorous mathematical analysis of ECN based protocols is required before they can be widely used in real networks.

In Chapter 4 we have presented optimal minimum variance and generalized optimal minimum variance window control algorithms for fair end-to-end congestion control. A further direction of research would be to analyze the interaction of generalized minimum variance based protocols with the existing Reno based end host protocols such as TCP SACK and TCP New Reno.

In Chapters 5 and 6 we presented two new algorithms namely Hybrid RED and Auto-Tuning RED, which need to be integrated into a single coherent algorithm. In Hybrid RED algorithm we used instantaneous queue size to indicate true occupancy of the queue; it can be generalized by using a second EWMA with a higher value of its averaging weight. Thus, we can use two EWMAs with RED, one for marking/dropping packets and the other for indicating actual queue occupancy. An unsolved problem in RED algorithm is to design a truly adaptive algorithm which can adapt all of its four major parameters, i.e. $\{ min_{th}, max_{th}, max_p, w_q \}$, to changes in traffic load. Closely related to RED is the RIO algorithm as described in Chapter 5; another research area would be to apply the Hybrid RED, Auto-tuning and other algorithms to RIO.

Another direction for future work would be the problem of non-responsive flows in the Internet. As outlined in Chapter 7, that RED algorithm can be used for detecting and punishing the non-responsive traffic flows which still need more work. The design of feedback congestion control algorithms that can also tackle with rogue non-responsive traffic flows could be a new area of research. Also, as already been pointed out, another exciting area of future work is the mathematical analysis of performance of CHOKe based cascaded feedback congestion control algorithms in the environments of non-responsive traffic flows.

A possible extension of the work done in Chapter 8 of this thesis would be the development of an adaptive version of the PID based control algorithm for AQM. Also, we have looked only at one basic form of PID control but its different other structures, as given in [Astrom and Wittenmark 1990], needs further exploration.

One weakness of the work presented in Chapters 7 and 8 is the assumption of homogenous round trip time for different TCP flows, during the mathematical development of different control algorithms for a router implementing AQM. In the real Internet the round trip times of different TCP flows will be, usually, varying widely. Thus, in order to ensure the good performance of AQM, it is suggested that a formal analysis of control algorithms (under the condition of heterogenous round trip times) should be attempted before implementing them in real networks. Furthermore, it is also pointed out that global stability in a general network with feedback delays is still an open problem: see e.g. [Hollot and Chait 2001], [Wang and Paganini 2002], [Deb and Srikant 2003] and references mentioned therein.

The toplogy of the real Internet is also difficult to characterize, see [Floyd and Paxon 2001] and its further references. In order to capture the essence of the dynamics of the immensely changing topology of the current Internet, one has to resort to simplified network models. Thus, the performance of the different proposed algorithms has been investigated by simulating the standard network topologies having a single bottleneck link. An important direction of future

work would be to determine the performance of the proposed algorithms in large scale simulations, with more complicated network topologies, such as having multiple bottlenecks, and to modify them as needed.

Finally, another important area for future work can be practical implementation of the algorithms developed in this thesis, both for end host protocols and for routers, first in test bed environments and then in real networks. For experimentations on the end hosts congestion control algorithms for AQM, the use of open source operating systems such as FreeBSD or Linux is suggested.

APPENDICES

Appendix A

VARIANCE OF AUTO-REGRESSIVE PROCESS

In equation (4.1) we have the following relation:

$$B(k) = a_{ar} \cdot B(k-1) + (1 - a_{ar}) \cdot \bar{B} + b_{ar} \cdot w(k). \tag{A.1}$$

For a random variable X, the varinace $Var(X)$ is defined as:

$$Var(X) = E\left[\{X - E(X)\}^2\right]. \tag{A.2}$$

Thus we have:

$$Var\{B(k)\} = E\left[\{B(k) - \bar{B}\}^2\right]. \tag{A.3}$$

Expanding the RHS of equation (A.3) we get:

$$Var\{B(k)\} = E\left[a_{ar}^2 \cdot \{B(k-1) - \bar{B}\}^2 + 2 \cdot a_{ar} \cdot b_{ar} \cdot \{B(k-1) - \bar{B}\} \cdot w(k) + b_{ar}^2 \cdot \{w(k)\}^2\right]. \tag{A.4}$$

Further simplification of the above equation will yield

$$Var\{B(k)\} = a_{ar}^2 \cdot E\{B(k-1) - \bar{B}\} + 2 \cdot a_{ar} \cdot b_{ar} \cdot E\left[\{B(k-1) - \bar{B}\} \cdot w(k)\right] + b_{ar}^2 \cdot E\left[\{w(k)\}^2\right]. \tag{A.5}$$

The equation (A.5) reduces to the following:

$$Var\{B(k)\} = a_{ar}^2 \cdot Var\{B(k-1)\} + b_{ar}^2. \tag{A.6}$$

As $k \to \infty$, $Var\{B(k-1)\} \to Var\{B(k)\}$, giving:

$$Var\{B(k)\} = a_{ar}^2 \cdot Var\{B(k)\} + b_{ar}^2. \tag{A.7}$$

Hence, we get the following, which is same as equation (4.2):

$$Var\{B(k)\} = \frac{b_{ar}^2}{1 - a_{ar}^2}. \tag{A.8}$$

Appendix B

INTERPRETATION OF ZIEGLER-NOCHOLS CONTINUOUS CYCLING METHOD

The Ziegler-Nochols frequency domain technique can be interpreted as a method used to position the operating point on Nyquist curve of open loop transfer function of a feedback control system, [Astrom and Hagglund 1995].

Using N-PI or N-PID algorithms, developed in Chapter 7, the operating point of AQM model can be moved at any position on complex plane. The tranfer function of PI controller is given in [Franklin *et al.* 1994] as:

$$C_{PI}(s) = K_c \cdot \left(1 + \frac{1}{sT_i}\right). \tag{B.1}$$

At the ultimate frequency $\omega_u$we can write equation (B.1) as

$$C_{PI}(j\omega_u) = K_c \cdot \left(1 + \frac{1}{j\omega_u T_i}\right). \tag{B.2}$$

Furthermore by inserting $\omega_u = \frac{2\pi}{T_u}$ in equation (B.2) we get

$$C_{PI}(j\omega_u) = \left(K_c + \frac{K_c}{j \cdot \left(\frac{2\pi}{T_u}\right) \cdot T_i}\right). \tag{B.3}$$

Let us consider a general class of PI controllers with $K_c = \sigma_{pi} \cdot K_u$ and $T_i = \delta_{pi} \cdot T_u$ with $0 < \sigma_{pi} \leq 1$ and $0 < \delta_{pi} \leq 1$. Substituting values in equation (B.3) we get:

$$C_{PI}(j\omega_u) = K_u \cdot \left(\sigma_{pi} - j \cdot \frac{\sigma_{pi}}{\left(\frac{2\pi}{T_u}\right) \cdot T_i}\right). \tag{B.4}$$

Hence, the ultimate point will be shifted to a new location $\left(\sigma_{pi} - j \cdot \frac{\sigma_{pi}}{\left(\frac{2\pi}{T_u}\right) \cdot T_i}\right)$ with the phase angle of $\phi_{pi} = -Tan^{-1}(\frac{1}{2\pi\delta_{pi}})$.

Similarly the transfer function of PID controller at ω_u can be expressed as:

$$C_{PID}(j\omega_u) = K_c \cdot \left(1 + j\omega_u T_d + \frac{1}{j\omega_u T_i}\right), \tag{B.5}$$

$$C_{PID}(j\omega_u) = \left[K_c + \frac{K_c}{j \cdot \left(\frac{2\pi}{T_u}\right) \cdot T_i} + j \cdot K_c \cdot \left(\frac{2\pi}{T_u}\right) \cdot T_d\right]. \tag{B.6}$$

Substituting $K_c = \sigma_{pid} \cdot K_u$, $T_i = \delta_{pid} \cdot T_u$ and $T_d = \epsilon_{pid} \cdot T_u$ with $0 < \sigma_{pid} \leq 1$, $0 < \delta_{pid} \leq 1$ and $0 < \epsilon_{pid} \leq 1$, in equation (B.6) we get:

$$C_{PID}(j\omega_u) = K_u \cdot \left(\sigma_{pid} - j \cdot \frac{\sigma_{pid}}{2\pi \cdot \delta_{pid}} + j \cdot \sigma_{pid} \cdot 2\pi \cdot \epsilon_{pid}\right). \tag{B.7}$$

The new location of the ultimate point with PID controller will be $\left(\sigma_{pid} - j \cdot \frac{\sigma_{pid}}{2\pi \cdot \delta_{pid}} + j \cdot \sigma_{pid} \cdot 2\pi \cdot \epsilon_{pid}\right)$ with the phase angle of $\phi_{pid} = Tan^{-1}\left(\frac{4\pi^2 \epsilon_{pid}\delta_{pid}-1}{2\pi\delta_{pid}}\right)$.

In [Astrom and Hagglund 1995] the values of parameters from Table 7.1 are substituted into equation (B.1) and equation (B.6) to get the new location and the phase angle of the ultimate point, which are summarized in Table B.

Controller	Parameters (K_c, T_i, T_d)	New Location of ultimate point.	Phase Angle (degrees)
N-PI-1	$(0.45K_u, 0.85T_u, 0)$	$(0.45 - j0.0842)$	10.60 Lag
N-PI-2	$(0.30K_u, 0.85T_u, 0)$	$(0.30 - j0.0561)$	10.60 Lag
N-PID	$(0.60K_u, 0.50T_u, 0.125T_u)$	$(-0.60 - j0.28)$	25.01 Lead

Table B.1 Shifting of the ultimate point of PI and PID controllers designed by the frequency response method.

Appendix C

STABILITY MARGINS

To evaluate the stability of control systems two quantities called Gain Margin (GM) and Phase Margin (PM), along with associated cross over frequency are widely used in practice. These are defined as follows:

- The GM is defined as a factor by which open loop gain of a stable system must be changed to make it unstable. Its value can be obtained from the Bode plot of open loop transfer function $G_p(j\omega)$ by measuring vertical distance between $|G_p(j\omega)|$ curve and $|G_p(j\omega)| = 1$ line at $\angle G_p(j\omega) = 180$ degrees.
- The PM is defined as the amount by which the phase of $G_p(j\omega)$ exceeds -180 degrees when $|G_p(j\omega)| = 1$.
- The term cross over frequency refers to a frequency at which gain is unity or 0 db.

The GM and PM can also be obtained by using Nyquist plots as given in [Nyquist 1932] and [Franklin *et al.* 1994].

REFERENCES

AHN, J.S., DANZIG, P., LIU, Z. AND YAN, L. (1995), 'An Evaluation of TCP Vegas: Emulation and Experiment', *ACM Computer Communication Review*, Vol. 25, No. 4, October, pp. 185–195.

AIT-HELLAL, O. AND ALTMAN, E. (1997), 'Analysis of TCP Vegas and TCP Reno', *Proceedings of IEEE International Conference on Communications, ICC 97*, Vol. 1, June, pp. 495–499.

ALLMAN, M. AND FALK, A. (1999), 'On the Effective Evaluation of TCP', *ACM Computer Communication Review*, Vol. 5, No. 29, October, pp. 59–70.

ALLMAN, M. AND PAXON, V. (1999), 'On Estimating End-to-End Network Path Properties', *Proceedings of ACM SIGCOMM'99*, pp. 263–274.

ALLMAN, M., V.PAXON AND STEVENS, W. (1999), 'TCP Congestion Control', *Network Working Group, RFC 2581*, April.

ALTMAN, E., AVRACHENKOV, K. AND BARAKAT, C. (2000), 'A Stochastic Model of TCP/IP with Stationary Random Losses', *Proceedings of the ACM SIGCOMM'00*, Vol. 30, No. 4, August, pp. 231–242.

ASTROM, K.J. AND HAGGLUND, T. (1995), *PID Controllers: Theory, Design and Tuning*, Instrument Society of America, 67 Alexander Drive, P.O. Box 12277, Research Triangle Park, NC 27709, USA, 2nd ed.

ASTROM, K.J. AND WITTENMARK, B. (1990), *Computer Controlled Systems*, Prentice-Hall International, Inc, Englewood Cliffs, N.J. 07632, USA., 2nd ed.

ASTROM, K.J. AND WITTENMARK, B. (1995), *Adaptive Control*, Addison-Wesley Publishing Company, Inc, USA, 2nd ed.

ATHURALIYA, S. AND LOW, S.H. (2000), 'Optimization Flow Control-II: Implementation', Unpublished draft available at http://netlab.caltech.edu/, May.

ATHURALIYA, S., LI, V.H., LOW, S.H. AND YIN, Q. (2001), 'REM: Active Queue Management', *IEEE Network*, Vol. 15, No. 3, May-June, pp. 48–53. http://netlab.caltech.edu.

AWEYA, J., OUELLETTE, M., MONTUNO, D.Y. AND CHAPMAN, A. (2001), 'Enhancing TCP performance with a load-adaptive RED mechanism', *International Journal of Network Management*, Vol. 11, pp. 31–50. ISSN: 1099-1190.

BENMOHAMED, L. AND MEERKOV, S.M. (1993), 'Feedback control of congestion in packet switching networks: The case of a single congested node', *IEEE/ACM Transactions on Networking*, Vol. 1, No. 6, December, pp. 693–707.

BENNETT, J.C.R., PARTRIDGE, C. AND SHECTMAN, N. (1999), 'Packet Reordering is Not Pathological Network Behavior', *IEEE/ACM Transactions on Networking*, Vol. 7, No. 6, December, pp. 789–798.

BERNERS-LEE, T., FIELDING, R. AND FRYSTYK, H. (1996), 'Hypertext Transfer Protocol – HTTP/1.0', *Network Working Group*, May. Category: Informational.

BERTSEKAS, D. AND GALLAGER, R. (1996), *Data Networks*, Prentice-Hall International, Inc, Englewood Cliffs, N.J., USA, 2nd ed.

BIRO, J. AND LUONG, D.D. (2001), 'On the proportional fairness of TCP Vegas', *Proceedings of the IEEE GLOBECOM'01*, Vol. 3, pp. 1718–1722.

BLANTON, E. AND ALLMAN, M. (2002), 'On Making TCP More Robust to Packet Reordering', *ACM SIGCOMM, Computer Communication Review*, Vol. 32, No. 1, January, pp. 20–30.

BODE, H.W. (1940), 'Relations Between Attenuation and Phase in Feedback Amplifier Design', *Bell System Tech J.*, Vol. 19, July, pp. 421–454.

BOLOT, J.C., TURLETTI, T. AND WAKEMAN, I. (1994), 'Scalable Feedback Control for Multicast Video Distribution in the Internet', *Proceedings of the ACM SIGCOMM'94*, pp. 58–67.

BONALD, T. (1998), *Comparison of TCP Reno and TCP Vegas via Fluid Approximation*, Technical Report RR-3563, INRIA, France, November. ftp://ftp-sop.inria.fr/pub/rapports/RR-3563.ps.gz.

BONALD, T. AND MASSOULIE, L. (2001), 'Impact of fairness on internet performance', In *SIGMETRICS/Performance*, pp. 82–91.

BONALD, T., MAY, M. AND BOLOT, J.C. (2000), 'Analytic evaluation of RED performance ', *INFOCOM 2000. Proceedings of Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies.*, Vol. 3, pp. 1415–1424.

BOUTREMANS, C. AND BOUDEC, J.Y.L. (2000), 'A Note on the fairness of TCP Vegas', *Proceedings of International Zurich Seminar on Broadband Communications*, February, pp. 163–170.

BRADEN, B., CLARK, D. *et al.* (1998), 'Recommendations on Queue Management and Congestion Avoidance in the Internet', *Network Working Group, Request for Comments: 2309*, April. Category: Informational.

BRAKMO, L.S. AND PETERSON, L.L. (1995), 'TCP Vegas: End to End Congestion Avoidance on a Global Internet', *IEEE Journal on Selected Areas in Communications*, Vol. 13, No. 8, October, pp. 1465–1480.

BRAKMO, L.S., O'MALLEY, S.W. AND PETERSON, L.L. (1994a), 'TCP Vegas: New Techniques for Congestion Detection and Avoidance', *ACM Computer Communication Review*, Vol. 24, No. 4, October, pp. 24–35.

BRAKMO, L.S., MALLEY, S.W.O. AND PETERSON, L. (1994b), 'TCP Vegas: New Techniques for Congestion Detection and Avoidance', *Proceedings of the ACM SIGCOMM'94*, October, pp. 24–35.

BRUYERON, R., HEMON, B. AND ZHANG, L. (1998), 'Experimentations with TCP Selective Acknowledgment', *ACM Computer Communication Review*, Vol. 28, No. 2, April, pp. 54–77.

CAIDA (2003), 'Cooperative Association for Internet Data Analysis', Information available at http://www.caida.org.

CARDWELL, N., SAVAGE, S. AND ANDERSON, T. (1998), *Modeling the Perfromance of Short TCP Connections*, Technical Report, Dept. of Computer Science, University of Washington, November.

CARDWELL, N., SAVAGE, S. AND ANDERSON, T. (2000), 'Modeling TCP Latency', *Proceedings of IEEE INFOCOM 2000*, Vol. 3, March, pp. 1742–1751.

CERF, V. AND KAHN, R. (1974), 'A Protocol for Packet Network Interconnection', *IEEE Transactions on Communications*, Vol. COM-22, May, pp. 637–648.

CHAIT, Y., HOLLOT, C., MISRA, V., HAN, H. AND HALEVI, Y. (2002), 'Dynamic Analysis of Congested TCP Networks', *Proceedings of American Control Conference*, Vol. 3, May, pp. 2430–2435. Anchorage AK, USA.

CHIU, D.M. AND JAIN, R. (1989), 'Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks', *Computer Networks and ISDN Systems*, Vol. 17, June, pp. 1–14. http://www.cis.ohio-state.edu/~jain/papers/cong_av.htm.

CISCO (1998), 'Distributed Weighted Random Early Detection', Documentation available at http://www.cisco.com/univercd/cc/td/doc/product/software/ios111/cc111/wred.htm.

CRAWLEY, E., NAIR, R. *et al.* (1998), 'A Framework for QoS-based Routing in the Internet', *Network Working Group, RFC 2386*, August. Category: Informational.

CROWCROFT, J. AND OECHSLIN, P. (1998), 'Differentiated End-to-End Internet Services using a Weighted Proportional Fair Sharing TCP', *Computer Communications Review*, Vol. 28, No. 3, July, pp. 53–67.

DARPA/VINT, LBNL, XEROX, UCB AND USC/ISI (2001), 'Network simulator ns-2.1b8', Information available at http://www.isi.edu/nsnam/ns/. Code available at http://www.isi.edu/nsnam/dist/.

DEB AND SRIKANT (2003), 'Global Stability of Congestion Controllers for the Internet', *IEEE Transactions on Automatic Control*, Vol. 48, No. 6, June, pp. 1055–1060.

DEERING, S. AND HINDEN, R. (1995), 'Internet Protocol, Version 6 (IPv6)', *Network Working Group, RFC 1883*, December. Category: Standards Track.

FALK, A., FABER, T., BANNISTER, J., CHIEN, A., GROSSMAN, R. AND LEIGH, J. (2003), 'Transport Protocols for High Performance', *Communications of the ACM*, Vol. 46, No. 11, November, pp. 43–49.

FALL, K. AND FLOYD, S. (1996), 'Simulation based Comparisons of Tahoe, Reno and SACK TCP', *ACM Computer Communication Review*, Vol. 26, No. 3, July, pp. 5–21. ftp://ftp.ee.lbl.gov/papers/sacks.ps.Z.

FENG, W., KANDLUR, D., SAHA, D. AND SHIN, K. (1997), *Techniques for Eliminating Packet Loss in Congested TCP/IP Networks*, Technical Report CSE-TR-349-97, University of Michigan, USA, November.

FENG, W.C., KANDLUR, D.D., SAHA, D. AND SHIN, K.G. (1999a), 'A Self Configuring RED Gateway', *Proceedings of the IEEE INFOCOM 1999*, March, pp. 1320–1328. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies.

FENG, W.C., KANDLUR, D., SAHA, D. AND SHIN, K. (1999b), *BLUE: A New Class of Active Queue Management Algorithms*, Technical Report CSE-TR-387-99, University of Michigan, USA, April.

FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H. AND BERNERS-LEE, T. (1997), 'Hypertext Transfer Protocol – HTTP/1.1', *Network Working Group*, January. Category: Standards Track.

FIROIU, V. AND BORDEN, M. (2000), 'A Study of Active Queue Management for Congestion Control', *Proceedings of the IEEE INFOCOM 2000*, Vol. 3, March, pp. 1435–1444. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies.

FLOYD, S. (1991a), 'Connections with Multiple Congested Gateways in Packet-Switched Networks, Part 1: One-way Traffic', *ACM Computer Communication Review*, Vol. 21, No. 5, October, pp. 30–47.

FLOYD, S. (1991b), 'Connections with Multiple Congested Gateways in Packet-Switched Networks, Part 2: Two-way Traffic', December. Lawrence Berkeley Laboratory, 1 Cyclotron Road Berkeley, CA 94720, USA.

FLOYD, S. (1994), 'TCP and Explicit Congestion Notification', *ACM Computer Communication Review*, Vol. 24, No. 5, October, pp. 8–23.

FLOYD, S. (1995), 'TCP and Successive Fast Retransmits', Unpublished technical note available at ftp://ftp.ee.lbl.gov/papers/fastretrans.ps, May.

FLOYD, S. (1996), *Issues of TCP with SACK*, Technical Report, LBL Network Group, March. ftp://ftp.ee.lbl.gov/papers/issues_sa.ps.Z.

FLOYD, S. (1997), 'RED: Discussions of setting parameters', Email available at http://www.aciri.org/floyd/REDparameters.txt, November.

FLOYD, S. (2000a), 'Recommendation on using the gentle_ variant of RED', Technical note available at http://www.aciri.org/floyd/red/gentle.html, March.

FLOYD, S. (2000b), 'Congestion Control Principles', *Network Working Group, RFC 2914*, September. Category: Best Current Practice.

FLOYD, S. (2003), 'HighSpeed TCP for Large Congestion WIndows', *Internet Draft, Internet Engineering Task Force*, June. draft-floyd-tcp-highspeed-03.ps.

FLOYD, S. AND FALL, K. (1997), *Router Mechanisms to Support End-to-End Congestion Control*, Technical Report, Lawrence Berkeley National Laboratory, Berkeley, CA, USA, February. Network Research Group.

FLOYD, S. AND FALL, K. (1999), 'Promoting the Use of End-to-End Congestion Control in the Internet', *IEEE/ACM Transactions on Networking*, Vol. 7, No. 4, August, pp. 458–472.

FLOYD, S. AND HENDERSON, T. (1999), 'The New Reno Modification to TCP's Fast Recovery Algorithm', *Network Working Group, RFC 2582*, April. Category: Experimental.

FLOYD, S. AND JACOBSON, V. (1992), 'On Traffic Phase Effects in Packet-Switched Gateways', *Internetworking: Research and Experience*, Vol. 3, No. 3, September, pp. 115–156.

FLOYD, S. AND JACOBSON, V. (1993), 'Random Early Detection Gateways for Congestion Avoidance', *IEEE/ACM Transactions on Networking*, Vol. 1, No. 4, August, pp. 397–413.

FLOYD, S. AND JACOBSON, V. (1994), 'The Synchronization of Periodic Routing Messages', *IEEE/ACM Transactions on Networking*, Vol. 2, No. 2, April, pp. 122–136.

FLOYD, S. AND PAXON, V. (2001), 'Difficulties in Simulating the Internet', *IEEE/ACM Transactions on Networking*, Vol. 9, No. 4, August, pp. 392–403.

FLOYD, S., FALL, K. AND TIEU, K. (1998), 'Estimating Arrival Rates from the RED Packet Drop History', Online draft is available at http://www.icir.org/floyd/papers/red-dropping.ps, April. Code is available at http://www.icir.org/floyd/papers/collapse/ns2/red-dropping.html.

FLOYD, S., MAHDAVI, J., MATHIS, M. AND PODOLSKY, M. (2000a), 'An Extension to the Selective Acknowledgement (SACK) Option for TCP', *Network Working Group , Request for Comments: 2883*, July. Category: Standards Track.

FLOYD, S., HANDLEY, M., PADHYE, J. AND WIDMER, J. (2000b), 'Equation-Based Congestion Control for Unicast Applications', *Proceedings of the ACM SIGCOMM'00*, Vol. 30, No. 4, October, pp. 43–56. Stockholm, Sweden.

FLOYD, S., GUMMADI, R. AND SHENKER, S. (2001), 'Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management', Unpublished draft available at http://www.icir.org/floyd/papers/adaptiveRed.pdf, August.

FRANKLIN, G.F., POWELL, J.D. AND WORKMAN, M.J. (1992), *Digital Control of Dynamic Systems*, Addison-Wesley publishing company, USA, 2nd ed.

FRANKLIN, G.F., POWELL, J.D. AND EMAMI-NAEINI, A. (1994), *Feedback Control of Dynamic systems*, Addison-Wesley Publishing Company, USA, 3rd ed.

GIBBEN, R. AND KELLY, F. (1999), 'Resource pricing and evolution of congestion control', *Automatica*, Vol. 35, pp. 1969–1985. http://www.statslab.cam.ac.uk/~frank/evol.html.

GORINSKY, S. AND VIN, H. (2002), *Extended Analysis of Binary Adjustment Algorithms*, Technical Report TR2002-39, Dept of Computer Science, Univ of Texas at Austin, USA, August. http://www.cs.utexas.edu/users/gorinsky/pubs.html.

HAGERUP, T. AND RUB, C. (1990), 'A guided tour of Chernoff bounds', *Information Processing Letters*, Vol. 33, No. 6, pp. 305–308.

HASEGAWA, G., MURATA, M. AND MIYAHARA, H. (1999), 'Fairness and Stability of Congestion Control Mechanisms of TCP', *Proceedings of the IEEE INFOCOM 1999*, Vol. 3, March, pp. 1329–1336.

HASEGAWA, G., KURATA, K. AND MURATA, M. (2000), 'Analysis and Improvement of Fairness between TCP Reno and Vegas for Deployment of TCP Vegas to the Internet', *Proceedings of International Conference on Network Protocols*, pp. 177–186.

HASHEM, E. (1989), *Analysis of Random Drop for Gateway Congestion Control*, Technical Report LCS TR-465, Laboratory for Computer Science, MIT, Cambridge, MA, USA.

HASS, Z. AND WINTERS, J.H. (1991), 'Congestion Control by Adaptive Admission', *Proceedings of the IEEE INFOCOM 1991*, Vol. 2, April, pp. 560–569.

HASSAN, M. AND SIRISENA, H. (2001), 'Optimal control of queues in computer networks', *Proceedings of the IEEE International Conference on Commuincations, ICC 2001*, Vol. 2, June, pp. 637–641. Helsinki, Finland.

HAYKIN, S. (1998), *Adaptive Filter Theory*, Prentice Hall, Inc, Englewood Cliffs, NJ, USA, 3rd ed.

HENDERSON, T.R., SAHOURIA, E., MCCANNE, S. AND KATZ, R.H. (1998), 'On Improving the Fairness of TCP Congestion Avoidance', *Proceedings of the IEEE GLOBECOM'98*, Vol. 1, pp. 593–544.

HOE, J.C. (1995), *Start up Dynamics of TCP's Congestion Control and Avoidance Schemes*, Master's thesis, MIT, USA. http://ana-www.lcs.mit.edu/anaweb/ps-papers/hoe-thesis.ps.

HOE, J.C. (1996), 'Improving the Start-up Behavior of a Congestion Control Scheme for TCP', *Proceedings of the ACM SIGCOMM'96*, Vol. 26, No. 4, October, pp. 270–280. Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications.

HOLLOT, C.V. AND CHAIT, Y. (2001), 'Nonlinear Stability Analysis for a Class of TCP/AQM Networks', *Proceedings of the 40th IEEE Conference on Decision and Control*, Vol. 3, December, pp. 2309–2314.

HOLLOT, C.V., MISRA, V., TOWSLEY, D. AND GONG, W. (2000), *A Control Theoretic Analysis of RED*, CMPSCI Technical Report TR 00-41, University of Massachusetts, Amherst MA 01003, USA, July.

HOLLOT, C.V., MISRA, V., TOWSLEY, D. AND GONG, W.B. (2001a), 'A Control Theoretic Analysis of RED', *Proceedings of the IEEE INFOCOM 2001*, Vol. 3, April, pp. 1510–1519. Anchorage, Alaska, USA.

HOLLOT, C.V., MISRA, V., TOWSLEY, D. AND GONG, W.B. (2001b), 'On Designing Improved Controllers for AQM Routers Supporting TCP Flows', *Proceedings of the IEEE INFOCOM 2001*, Vol. 3, April, pp. 1726–1734. Anchorage, Alaska, USA.

HOLLOT, C.V., MISRA, V., TOWSLEY, D. AND GONG, W. (2002), 'Analysis and Design of Controllers for AQM Routers Supporting TCP Flows', *IEEE Transactions on Automatic Control*, Vol. 47, No. 6, June, pp. 945–959.

HOLLOT, C.V., LIU, Y., MISRA, V. AND TOWSLEY, D. (2003), 'Unresponsive Flows and AQM Performance', *Proceedings of the IEEE INFOCOM 2003*, Vol. 1, pp. 85–95.

HOSTETTER, G.H. (1988), *Digital Control System Design*, Holt, Rinehart and Winston, Inc., 111 Fifth Avenue, New York 10003, USA.

HUTCHINSON, N.C. AND PETERSON, L.L. (1991), 'The x-Kernel: An Architecture for Implementing Network Protocols', *IEEE Transactions on Software Engineering*, Vol. 17, No. 1, January, pp. 64–76. x-Kernel home page, http://www.cs.arizona.edu/xkernel/nsrg.html.

INTERNET2 (2003), 'Internet 2', Information available at http://www.internet2.edu/.

IPV6 (2003), 'Internet Protocol Version 6', Information available at http://www.ipv6.org/.

JACOBSON, V. (1988), 'Congestion Avoidance and Control', *ACM Computer Communication Review*, Vol. 18, No. 4, August, pp. 314–329.

JACOBSON, V. (1990), 'Modified TCP Congestion Avoidance Algorithm', Message to end2end-interest mailing list, available at ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt, April.

JACOBSON, V. AND BRADEN, R. (1988), 'TCP Extensions for Long-Delay Paths', *Network Working Group, Request for Comments: 1072*, October.

JACOBSON, V. AND KARELS, M.J. (1992), 'Congestion Avoidance and Control', Unpublished draft available at ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z.

JACOBSON, V., NICHOLS, K. AND PODURI, K. (1999), 'RED in a Different Light', Unpublished draft available at http://www.cnaf.infn.it/~ferrari/papers/ispn/red_light_9_30.pdf, September.

JAIN, R. (1989), 'A Delay-Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks', *ACM Computer Communication Review*, Vol. 19, No. 5, October, pp. 56–71.

JAIN, R. (1991), *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modelling*, John Wiley and Sons, Inc, New York, April.

JAIN, R. AND RAMAKRISHNAN, K. (1988), 'Congestion Avoidance in Computer Networks with a Connectionless Network Layer: Concepts, Goals and Methodology', *Proceedings of the Computer Networking Symposium*, April, pp. 134–143. Washington, DC, USA.

JAIN, R., CHIU, D.M. AND HAWE, W. (1984), *A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems*, Technical Report DEC-TR-301, Digital Euipment Corporation, 77 Reed Road, Hudson, MA 01749, USA, September. Available at http://www.cis.ohio-state.edu/~jain/papers/fairness.htm.

JAIN, R., RAMAKRISHNAN, K. AND CHIU, D. (1987), *Congestion Avoidance in Computer Networks with a Connectionless Network Layer*, Technical Report DEC-TR-506, Digital Equipment Corporation, 550 King St, Littleton, MA 01460, USA, August. Available at http://www.cis.ohio-state.edu/~jain/papers/cr5.htm.

JEONG, H.D.J. (2002), *Modelling of Self-Similar Teletraffice for Simulation*, PhD thesis, University of Canterbury, Christchurch, New Zealand, July. Available at http://www.cosc.canterbury.ac.nz/research/reports/PhdTheses/2003/phd_0302.pdf.

JIN, C., WEI, D.X. AND LOW, S.H. (2004), 'FAST TCP: Motivation, Architecture, Algorithms, Performance', March. Draft available at http://netlab.caltech.edu/FAST/.

KARN, P. AND PARTRIDGE, C. (1987), 'Improving Round-Trip Time Estimates in Reliable Transport Protocols', *Proceeding of the ACM SIGCOMM'87*, August, pp. 2–7.

KATABI, D., HANDLEY, M. AND ROHRS, C. (2002), 'Congestion Control for High Bandwidth-Delay Product Networks', *Proceeding of the ACM SIGCOMM'02*, Vol. 32, No. 4, October, pp. 89–102. Pittsburgh, PA, USA.

KELLY, F.P. (1997), 'Charging and Rate Control for Elastic Traffic', *European Transactions on Telecommunications*, Vol. 8, January-February, pp. 33–37.

KELLY, T. (2002), 'Scalable TCP: Improving Performance in Highspeed Wide Area Netwroks'. Unpublished Draft and code available at http://www-lce.eng.cam.ac.uk/~ctk21/scalable/#code.

KELLY, F.P., MAULLOO, A.K. AND TAN, D.K.H. (1998), 'Rate control in communication networks: shadow prices, proportional fairness and stability', *Journal of the Operational Research Society*, Vol. 49, pp. 237–252.

KONG, H., GE, N., RUAN, F., FENG, C. AND FAN, P. (2003), 'A Nonlinear Model on the AQM Algorithm GREEN', *IEICE Transaction on Communications*, Vol. E86-B, No. 2, February, pp. 622–629.

KUNNIYUR, S. AND SRIKANT, R. (2001), 'Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management', *Proceedings of the ACM SIGCOMM'01*, Vol. 31, No. 4, October, pp. 123–134. San diego California, USA.

LA, R.J. (2001), 'Fluid Model for Window-Based Congestion Control Mechanism', *Proceedings of the 2001 Winter Simulation Conference*, pp. 1282–1290.

LAI, Y.C. (2001), 'Improving the Performance of TCP Vegas in a Heterogeneous Environment', *Proceedings of the Eighth International Conference on Parallel and Distributed Systems, ICPADS 2001*, pp. 581–587.

LAI, Y.C. AND YAO, C.L. (2000), 'The Performance Comparison between TCP Reno and TCP Vegas', *Proceedings of seventh IEEE International Conference on Parallel and Distributed Systems*, pp. 61–66.

LAKSHMAN, T.V. AND MADHOW, U. (1997), 'The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss', *IEEE/ACM Transactions on Networking*, Vol. 5, No. 3, June, pp. 336–350.

LAPSLEY, D. AND LOW, S. (1999a), 'Random Early Marking : An Optimisation Approach to Internet Congestion Control', *Proceedings of IEEE International Conference on Networks, ICON '99*, September-October, pp. 67–74.

LAPSLEY, D. AND LOW, S. (1999b), 'Random Early Marking for Internet Congestion Control', *Proceedings of IEEE GLOBECOM '99*, Vol. 3, December, pp. 1747–1752.

LEINER, B.M., COLE, R., POSTEL, J. AND MILLS, D. (1985), 'The DARPA Internet Protocol Suite', *IEEE Communications Magazine*, Vol. 23, March, pp. 29–34.

LELAND, W.E., TAQQU, M.S., WILLINGER, W. AND WILSON, D.V. (1994), 'On the Self-Similar Nature of Ethernet Traffic', *IEEE/ACM Transactions on Networking*, Vol. 2, No. 1, February, pp. 1–15.

LIN, D. AND MORRIS, R. (1997), 'Dynamics of Random Early Detection', *ACM Computer Communication Review*, Vol. 27, No. 4, October, pp. 127–136. Cannes, France.

LOW, S.H. (2000), 'A Duality Model of TCP and Queue Management Algorithms', *Proceedings of ITC Specialist Seminar on IP Traffic Measurement, Modeling and Management*, September. Monterey CA, USA.

LOW, S.H. AND LAPSLEY, D.E. (1999), 'Optimization Flow Control-I: Basic Algorithm and Convergence', *IEEE/ACM Transactions on Networking*, Vol. 7, No. 6, December, pp. 861–874.

LOW, S.H., PETERSON, L.L. AND WANG, L. (2001), 'Understanding TCP Vegas: A Duality Model', *Proceedings of the ACM SIGMETRICS*, June, pp. 226–235.

MAHAJAN, R. AND FLOYD, S. (2001), *Controlling High Bandwidth Flows at the Congested Router*, Technical Report TR-01-001, AT&T Center for Internet Research at ICSI (ACIRI), April. Available at http://www.cs.washington.edu/homes/ratul/red-pd/.

MANKIN, A. (1990), 'Random Drop Congestion Control', *Proceedings of the ACM SIGCOMM'90*, pp. 1–7.

MAPLESOFT (1998), 'MAPLE V Release 5', Information available at http://www.maplesoft.com/, January.

MASCOLO, S. (1999), 'Congestion control in high-speed communication networks using the Smith principle', *Automatica*, Vol. 35, pp. 1921–1935. http://www.elsevier.com/locate/automatica.

MATHIS, M. (2001), 'The Wizard Gap', Slides available at http://www.psc.edu/~mathis/papers/JTechs200105/mgp00003.html, May.

MATHIS, M. AND MAHDAVI, J. (1996), 'Forward Acknowledgment: Refining TCP Congestion Control', *Proceedings of the ACM SIGCOMM'96*, Vol. 26, No. 4, October, pp. 281–191. Stanford, CA, USA.

MATHIS, M. AND MAHDAVI, J. (1997), 'TCP Rate-Halving with Bounding Parameters', Unpublished technical note available at http://www.psc.edu/networking/papers/FACKnotes/current/, December.

MATHIS, M., MAHDAVI, J., FLOYD, S. AND ROMANOV, A. (1996), 'TCP Selective Acknowledgement Options', *Network Working Group, Request For Comments: 2018*, October. Category: Standards Track.

MATHIS, M., SEMKE, J., MAHDAVI, J. AND OTT, T. (1997), 'The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm', *ACM Computer Communication Review*, Vol. 27, No. 3, July, pp. 67–82.

MATHIS, M., SEMKE, J., MAHDAVI, J. AND LAHEY, K. (1999), 'The Rate Halving Algorithm for TCP Congestion Control', Unpublished draft available at http://www.psc.edu/networking/ftp/papers/draft-ratehalving.txt, June.

MAY, M., BOLOT, J., DIOT, C. AND LYLES, B. (1999), 'Reasons not to deploy RED', *Proceedings of 7th International Workshop on Quality of Service*, June, pp. 260–262.

MAY, M., DIOT, C., LYLES, B. AND BOLOT, J. (2000), *Influence of Active Queue Management Parameters on Aggregate Traffic Performance*, Technical Report RR-3995, INRIA, France, August. ftp://ftp.inria.fr/INRIA/publication/publi-pdf/RR/RR-3995.pdf.

MCCANNE, S., JACOBSON, V. AND VETTERLI, M. (1996), 'Receiver-driven Layered Multicast', *Proceedings of the ACM SIGCOMM'96*, pp. 117–130. Palo Alto, California, USA.

MISRA, A. AND OTT, T. (2003), 'Performance sentivity and fairness of ECN-aware modified TCP', *Performance Evaluation*, Vol. 53, pp. 225–253.

MISRA, V., GONG, W.B. AND TOWSLEY, D. (1999), 'Stochastic Differential Equation Modeling and Analysis of TCP Windowsize Behavior', *Proceedings of Performance*, October. http://www-net.cs.umass.edu/papers/papers.html.

MISRA, V., GONG, W.B. AND TOWSLEY, D. (2000), 'Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED', *Proceedings of the ACM SIGCOMM'00*, pp. 151–160.

MISRA, A., OTT, T. AND BARAS, J. (2001), 'Effect of Exponential Averaging on the Variability of a RED Queue ', *Proceedings of IEEE International Conference on Communications, ICC 2001.*, Vol. 6, June, pp. 1817–1823.

MO, J. AND WALRAND, J. (2000), 'Fair End-to-End Window-Based Congestion Control', *IEEE/ACM Transactions on Networking*, Vol. 8, No. 5, October, pp. 556–567.

MO, J., LA, R., ANANTHARAM, V. AND WALRAND, J. (1999), 'Analysis and Comparison of TCP Reno and Vegas', *Proceedings of the IEEE INFOCOM 1999*, Vol. 3, pp. 1556–1563. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies.

NAGLE, J. (1984), 'Congestion Control in IP/TCP Internetworks', *RFC 896*, January.

NAGLE, J.B. (1987), 'On Packet Switches with Infinite Storage', *IEEE Transactions on Communications*, Vol. 35, No. 4, April, pp. 435–438.

NUA (2002), 'How Many Online ?', Nua Internet Surveys available at http://www.nua.ie/surveys/how_many_online/, May.

NYQUIST, H. (1932), 'Regenration Theory', *Bell System Tech J.*, Vol. 11, January, pp. 126–147.

OSTRING, S.A.M. (2001), *Reactive Traffic Control Mechanisms for Communication Networks with Self-Similar Bandwidth Demands*, PhD thesis, University of Canterbury, Christchurch, New Zealand, January.

OSTRING, S. AND SIRISENA, H. (2001), 'The influence of long range dependence on traffic prediction', *Proceedings of the IEEE International Conference on Commuincations, ICC 2001*, Vol. 4, June, pp. 1000–1005. Helsinki, Finland.

OTT, T. (1999), 'On the Ornstein-Uhlenbeck Process with Delayed Feedback', Unpublished draft available at http://web.njit.edu/~ott/Papers/, December.

OTT, T.J., KEMPERMAN, J.H.B. AND MATHIS, M. (1996), 'The Stationary Behavior of Ideal TCP Congestion Avoidance', Unpublished draft available at http://web.njit.edu/~ott/Papers/Mathis/TCPwindow.pdf, August.

OTT, T.J., LAKSHMAN, T. AND WONG, L.H. (1999), 'SRED: Stabilized RED', *Proceedings of the IEEE INFOCOM 1999*, March, pp. 1346–1355.

OZBAY, H., KALYANARAMAN, S. AND IFTAR, A. (1998), 'On Rate-Based Congestion Control in High Speed Networks: Design of an H^∞ Based Flow Controller for Single Bottleneck', *Proceedings of the American Control Conference*, Vol. 4, pp. 2376–2380.

PADHYE, J.D. (2000), *Towards a comprehensive congestion control framework for continous media flows in best effort networks*, PhD thesis, University of Massachusetts Amherst, USA, March.

PADHYE, J. AND FLOYD, S. (2002), 'TCP Behavior Inference Tool, TBIT', Information and Code available at http://www.icir.org/tbit/, April.

PADHYE, J., FIROIU, V., TOWSLEY, D.F. AND KUROSE, J.F. (2000), 'Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation', *IEEE/ACM Transactions on Networking*, Vol. 8, No. 2, April, pp. 133–145.

PAN, R., PRABHAKAR, B. AND PSOUNIS, K. (2000), 'CHOKe, A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation', *Proceedings of the IEEE INFOCOM 2000*, Vol. 2, March, pp. 942–951.

PAXON, V. AND ALLMAN, M. (2000), 'Computing TCP's Retransmission Timer', *Network Working Group, RFC 2988*, November. Standards Track.

PAXSON, V. (1997a), 'End-to-end Internet Packet Dynamics', *Proceedings of the ACM SIGCOMM'97*, Vol. 27, No. 4, September, pp. 139–154. Cannes, France.

PAXSON, V. (1997b), 'Automated Packet Trace Analysis of TCP Implementations', *Proceedings of the ACM SIGCOMM'97*, Vol. 27, No. 4, October, pp. 167–179. Cannes, France.

PAXSON, V. (1999), 'End-to-end Internet Packet Dynamics', *IEEE/ACM Transactions on Networking*, Vol. 7, No. 3, June, pp. 277–292.

PAXSON, V., ALLMAN, M. *et al.* (1999), 'Known TCP Implementation Problems', *Network Working Group, RFC 2525*, March. Category: Informational.

POSTEL, J. (1981), 'Transmission Control Protocol', *RFC 793*, September.

RAMAKRISHNAN, K. (1999), 'A Proposal to add Explicit Congestion Notification (ECN) to IP', *Network Working Group, RFC 2481*, January. Category: Experimental.

RAMAKRISHNAN, K. AND JAIN, R. (1990), 'A Binary Feedback Scheme for Congestion Avoidance in Computer Networks', *ACM Transactions on Computer Systems*, Vol. 8, No. 2, May, pp. 158–181.

RAMAKRISHNAN, K., FLOYD, S. AND BLACK, D. (2001), 'The Addition of Explicit Congestion Notification (ECN) to IP', *Network Working Group, RFC 3168*, September. Category: Standards Track.

REN, F. AND LIN, C. (2003), 'Speed up the Responsiveness of Active Queue Management System', *IEICE Transactions on Communications*, Vol. E86-B, No. 2, February, pp. 630–636.

RICHARD, J., WALRAND, J. AND ANANTHARAM, V. (1998), 'Issues in TCP Vegas', Unpublished draft, July. Department of Electrical Engineering and Computer Sciences University of California at Berkeley, USA.

RIZZO, L. (1996), 'Issues in the implementation of selective acknowledgements for TCP', Unpublished draft available at http://www.iet.unipi.it/~luigi/selack.ps, January.

RIZZO, L. (1997), 'RED and non-responsive flows', end2end-interest mailing list, ftp://ftp.isi.edu/end2end/end2end-interest-1997.mail;type=a, June.

SHENKER, S., ZHANG, L. AND CLARK, D.D. (1990), 'Some Observations on the Dynamics of a Congestion Control Algorithm', *ACM Computer Communication Review*, October, pp. 30–39.

SIRISENA, H., HAIDER, A. AND PAWLIKOWSKI, K. (2002), 'Auto-Tuning RED for Accurate Queue Control', *Proceedings of the IEEE GLOBECOM'02*, Vol. 2, November, pp. 2010–2015. Taipei, Taiwan. http://www.globecom2002.com.tw/.

SPRAGINS, J.D., HAMMOND, J.L. AND PAWLIKOWSKI, K. (1991), *Telecommunications Protocols and Design*, Addison-Wesley Publishing Company, Addison-Wesley Publishing Company, Inc, USA.

STEVENS, W.R. (1994), *TCP/IP Illustrated: The Protocols*, Vol. 1, Addison-Wesley. ISBN 0-201-63346-9, Information available at http://www.kohala.com/start/tcpipiv1.html.

STEVENS, W.R. (1995), *TCP/IP Illustrated: The Implementation*, Vol. 2, Addison-Wesley. ISBN 0-201-63354-X, Information available at http://www.kohala.com/start/tcpipiv2.html.

STEVENS, W.R. (1996), *TCP/IP Illustrated: TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols*, Vol. 3, Addison-Wesley. ISBN 0-201-63495-3, Information available at http://www.kohala.com/start/tcpipiv3.html.

STEVENS, W. (1997), 'TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms', *Network Working Group RFC 2001*, January. Category: Standards Track.

STOICA, I., SHENKER, S. AND ZHANG, H. (1998), 'Core Stateless Fair Queueing: Achieving Approximatley Fair Bandwidth Allocations in High Speed Networks', *Proceedings of the ACM SIGCOMM'98*, Vol. 28, No. 4, September, pp. 118–130.

SUN, J., KO, K.T., CHEN, G., CHAN, S. AND ZUKERMAN, M. (2003a), 'PD-RED: To Improve the Performance of RED', *IEEE Communications Letters*, Vol. 7, No. 8, August, pp. 406–408.

SUN, J., CHEN, G., KO, K.T., CHAN, S. AND ZUKERMAN, M. (2003b), 'PD-Controller: A New Active Queue Management Scheme', *Proceedings of the IEEE GLOBECOM'03*, Vol. 6, December, pp. 3103–3107.

T. ZIEGLER, S.F. AND BRANDAUER, C. (2000), *Stability Criteria for RED with TCP Traffic*, Technical Report, May. Available at http://www-rp.lip6.fr/~sf/WebSF/PapersWeb/red.net2000.pdf.

TANENBAUM, A.S. (1996), *Computer Networks*, Prentice-Hall International, Inc, Upper Saddle River, New Jersey 07458, USA, 3rd ed.

VERES, A., KENESI, Z., MOLNAR, S. AND VATTAY, G. (2000), 'On the Propogation of Long-Range Dependence in the Internet', *Proceedings of the ACM SIGCOMM'00*, pp. 243–254.

WANG, Z. AND CROWCROFT, J. (1991), 'A New Congestion Control Scheme: Slow Start and Search (Tri-S)', *ACM Computer Communication Review*, Vol. 21, No. 1, January, pp. 32–43.

WANG, Z. AND CROWCROFT, J. (1992), 'Eliminating Periodic Packet Losses in 4.3-Tahoe BSD TCP Congestion Control Algorithm', *ACM Computer Communication Review*, Vol. 22, No. 2, April, pp. 9–16.

WANG, Z. AND PAGANINI, F. (2002), 'Global Stability with Time-Delay in Network Congestion Control', *Proceedings of the 41st IEEE Conference on Decision and Control*, Vol. 4, December, pp. 3632–3637.

WEB100 (2002), 'Web100 Project', Information available at http://www.web100.org.

WEIGLE, E. AND CHUN FENG, W. (2001), 'A Case for TCP Vegas in High-Performance Computational Grids', *Proceedings of the IEEE International Symposium on High Performance Distributed Computing*, August, pp. 152–158.

WILLINGER, W., TAQQU, M.S. AND ERRAMILLI, A. (1996), *A bibliographical guide to self-similar traffic and performance modeling for modern high-speed networks, In Stochastic networks: Theory and applications*, Clarendon Press, Oxford.

WYDROWSKI, B. AND ZUKERMAN, M. (2002), 'GREEN: An Active Queue Management Algorithm for a Self Managed Internet', *IEEE International Conference on Communications, ICC'02*, Vol. 4, April-May, pp. 2368–2372.

YEUNG, F. (1997), 'Internet 2: Scaling up the Backbone for R&D', *IEEE Internet Computing*, Vol. 1, No. 2, March/April, pp. 36–37.

YOUNG, P. (1984), *Recursive Estimation and Time-Series Analysis*, Sringer-Verlag, Springer-Verlag Berlin, Heidelberg, Germany. ISBN 3-540-13677-0.

ZHANG, L. AND CLARK, D. (1990), 'Oscillating behavior of Network Traffic: A case study simulation', *Internetworking: Research and Experience*, Vol. 1, No. 2, pp. 101–112.

ZHANG, M., KARP, B., FLOYD, S. AND PETERSON, L. (2002), *Improving TCP's Performance under Reordering with DSACK*, Technical Report TR-02-006, International Computer Science Institute, July. http://www.icsi.berkeley.edu/techreports/2002.abstracts/tr-02-006.html.

ZHENG, B. AND ATIQUZZAMAN, M. (2001), *Low Pass Filter/Over Drop Avoidance (LPF/ODA): An algorithm to improve the performance of RED gateways* , Technical Report CS-TR-01-001, University of Oklahoma, USA, February. http://www.icir.org/floyd/red.html.

ZHENG, L., ZHANG, L. AND XU, D. (2001), 'Characteristics of Network Delay and Delay Jitter and its Effect on Voice over IP (VoIP)', *Proceedings of IEEE International Conference on Communications, ICC 2001*, Vol. 1, June, pp. 122–126.

ZIEGLER, J.G. AND NICHOLS, N.B. (1942), 'Optimum Settings for Automatic Controllers', *Transactions of American Society of Mechanical Engineers*, Vol. 64, November, pp. 759–768.