A Multi-objective Particle Swarm Optimized Fuzzy Logic Congestion Detection and Dual Explicit Notification Mechanism for IP Networks

Clement Nthambazale Nyirenda

Submitted in fulfillment of the academic requirements for the degree of Master of Science in Engineering in the School of Electrical, Electronic and Computer Engineering at the University of KwaZulu-Natal, Durban, South Africa

September 2006

Abstract

The Internet has experienced a tremendous growth over the past two decades and with that growth have come severe congestion problems. Research efforts to alleviate the congestion problem can broadly be classified into three groups: (1) Router based congestion detection; (2) Generation and transmission of congestion notification signal to the traffic sources; (3) End-to-end algorithms which control the flow of traffic between the end hosts. This dissertation has largely addressed the first two groups which are basically router initiated. Router based congestion detection mechanisms, commonly known as Active Queue Management (AQM), can be classified into two groups: conventional mathematical analytical techniques and fuzzy logic based techniques. Research has shown that fuzzy logic techniques are more effective and robust compared to the conventional techniques because they do not rely on the availability of a precise mathematical model of Internet. They use linguistic knowledge and are, therefore, better placed to handle the complexities associated with the non-linearity and dynamics of the Internet. In spite of all these developments, there still exists ample room for improvement because, practically, there has been a slow deployment of AQM mechanisms.

In the first part of this dissertation, we study the major AQM schemes in both the conventional and the fuzzy logic domain in order to uncover the problems that have hampered their deployment in practical implementations. Based on the findings from this study, we model the Internet congestion problem as a multi-objective problem. We propose a Fuzzy Logic Congestion Detection (FLCD) which synergistically combines the good characteristics of the fuzzy approaches with those of the conventional approaches. We design the membership functions (MFs) of the FLCD algorithm automatically by using Multi-objective Particle Swarm Optimization (MOPSO), a population based stochastic optimization algorithm. This enables the FLCD algorithm to achieve optimal performance on all the major objectives of Internet congestion control. The FLCD algorithm is compared with the basic Fuzzy Logic AQM and the Random Explicit Marking (REM) algorithms on a best effort network. Simulation results show

that the FLCD algorithm provides high link utilization whilst maintaining lower jitter and packet loss. It also exhibits higher fairness and stability compared to its basic variant and REM. We extend this concept to Proportional Differentiated Services network environment where the FLCD algorithm outperforms the traditional Weighted RED algorithm. We also propose self-learning and organization structures which enable the FLCD algorithm to achieve a more stable queue, lower packet losses and UDP traffic delay in dynamic traffic environments on both wired and wireless networks.

In the second part of this dissertation, we present the congestion notification mechanisms which have been proposed for wired and satellite networks. We propose an FLCD based dual explicit congestion notification algorithm which combines the merits of the Explicit Congestion Notification (ECN) and the Backward Explicit Congestion Notification (BECN) mechanisms. In this proposal, the ECN mechanism is invoked based on the packet marking probability while the BECN mechanism is invoked based on the BECN parameter which helps to ensure that BECN is invoked only when congestion is severe. Motivated by the fact that TCP reacts to the congestion notification signal only once during a round trip time (RTT), we propose an RTT based BECN decay function. This reduces the invocation of the BECN mechanism and resultantly the generation of reverse traffic during an RTT. Compared to the traditional explicit notification mechanisms, simulation results show that the new approach exhibits lower packet loss rates and higher queue stability on wired networks. It also exhibits lower packet loss rates, higher goodput and link utilization on satellite networks. We also observe that the BECN decay function reduces reverse traffic significantly on both wired and satellite networks while ensuring that performance remains virtually the same as in the algorithm without BECN traffic reduction.

To my wife Felistas and daughter Uchindami.
Glory be to GOD

Preface

The research work presented in this dissertation was performed by Mr. Clement Nthambazale Nyirenda, under the supervision of Prof. Dawoud Dawoud at the University of KwaZulu-Natal's School of Electrical, Electronic and Computer Engineering, in the Centre of Radio Access Technologies, which is sponsored by Alcatel and Telkom South Africa Limited as part of the Centre of Excellence programme.

Parts of this dissertation have been presented by the student at *SATNAC 2005* in the Drankesberg, South Africa, *IEEE World Congress on Computational Intelligence (WCCI 2006)* in Vancouver, Canada and at *SATNAC 2006* in the Cape Town, South Africa. Another part of this dissertation has been accepted for the *Mobile Computing and Wireless Communications International Conference (MCWC 2006)*, which will take place in Amman, Jordan while one more part has been submitted to the *Scientia Iranica, International Journal of science and Technology.*

The whole dissertation, unless otherwise indicated, is the student's original work and has not been submitted in part or in whole, to any other University for the purpose of examination.

Acknowledgements

I would like to express my sincere thanks to my supervisor, Prof. Dawoud Dawoud, for his guidance and support throughout my entire study period. His insightfulness and continued support have been very instrumental to the success of this work.

Many thanks to my wife Felistas for her understanding and encouragement as I spent long periods of time away from her and our daughter Uchindami who was born while I was doing this work. Special thanks are owed to my grandmother Nyangwira for instilling in me the spirit of hard work and discipline as I was growing up. I am very glad because God has kept her all this time so that she can see the fruits of her work upon my life. Many thanks are owed to all my relatives and all the brethren in Malawi for their prayers and support for me and my family during the entire period of my study.

Many thanks are owed to the African Network and Technological Institutions (ANSTI) for their financial support towards my graduate studies. Thanks are also owed to Telkom South Africa Limited and Alcatel for providing the equipment necessary for my studies and the financial support that enabled me to present parts of this work at three conferences.

I would like to thank Mrs. R. Trutter, Mrs. B. Bennet, and Mr. Bruce Harrison for their help in one way or another towards my studies. Finally, I would like to thank all my friends and fellow postgraduate students for their assistance and the wonderful time we have been together.

Table of Contents

ABSTRACT II

PREFACE V

ACKNOWLEDGEMENTS VI

TABLE OF CONTENTS VII

LIST OF FIGURES X

LIST OF TABLES XII

LIST OF ABBREVIATIONS XIII

CHAPTER 1: INTRODUCTION 1

1.1 GENERAL 1
1.2 AN OVERVIEW OF INTERNET CONGESTION CONTROL 3
1.2.1 End-to-end Mechanisms 3
1.2.1.1 TCP Tahoe 4
1.2.1.2 TCP Reno 6
1.2.1.3 TCP New Reno 6
1.2.2 Router-Based Detection Mechanisms 6
1.2.2.1 Random Early Detection 7
1.2.2.2 Post RED Mechanisms 9
1.2.3 Explicit Notification Mechanisms 11
1.2.3.1 Internet Control Message Protocol (ICMP) Source Quenches (ISQ) 12
1.2.3.2 Explicit Congestion Notification (ECN) 13
1.2.3.3 Backward Explicit Congestion Notification (BECN) 13
1.2.3.4 Explicit Underutilization Notification Mechanisms 14
1.3 MOTIVATION AND FOCUS OF THIS THESIS 15
1.4 EMPIRICAL VALIDATION 16
1.5 THESIS ORGANIZATION 16
1.6 ORIGINAL CONTRIBUTIONS OF THIS THESIS 17
1.7 PUBLISHED WORK 18

CHAPTER 2: LITERATURE SURVEY ON POST-RED AQM ALGORITHMS 19

2.1 INTRODUCTION 19
2.2 MAJOR TRADITIONAL AQM ALGORITHMS 19
2.2.1 BLUE 20
2.2.2 CHOKe 21
2.2.3 Adaptive RED 22
2.2.4 REM 23
2.2.5 GREEN 24
2.2.6 WRED 25
2.3 FUZZY LOGIC BASED AQM SCHEMES 27

2.3.1 Fuzzy Logic Control Theory 27
2.3.1.1 Fuzzy Sets and Operators *28*
2.3.1.2 Fuzzy Logic Control: Principles of Operation *30*
2.3.2 Evaluation of Fuzzy Logic AQM Schemes 32
2.3.2.1 The Algorithm of Ren et. al *32*
2.3.2.2 The Algorithm of Chrysostomou et. al *34*
2.3.2.3 Fuzzy BLUE Controller *35*
2.3.2.4 Adaptive Fuzzy RED *37*
2.3.2.5 Fast Adaptive Fuzzy Controller *37*
2.3.2.6 Fuzzy RED for DiffServ *38*
2.4 CHAPTER SUMMARY 39

CHAPTER 3: FUZZY LOGIC CONGESTION DETECTION USING MOPSO 40

3.1 INTRODUCTION 40
3.2 FUZZY LOGIC CONGESTION DETECTION ALGORITHM 41
3.3 MOPSO IN FLCD PARAMETER OPTIMIZATION 45
3.3.1 Basics of Multi-objective Optimization 46
3.3.2 MOPSO Theory 47
3.3.3 Problem Formulation 49
3.3.3.1 Development of Objective Functions *49*
3.3.3.2 Constraints *50*
3.3.3.3 Problem Statement *50*
3.3.4 MOPSO Implementation and Optimization Results 51
3.3.4.1 The Optimization Process *51*
3.3.4.2 Implementation of the FLCD Script *53*
3.3.4.3 Parameters for the Optimization process *54*
3.3.4.4 Optimization Results *55*
3.3.4.5 Best Compromise Solution *57*
3.4 SIMULATION RESULTS AND PERFORMANCE ANALYSIS IN BEST EFFORT IP NETWORKS 58
3.4.1 Experiment 1: Congestion Control with Packet Dropping 59
3.4.2 Experiment 2: Vary the congestion level at the bottleneck link 60
3.4.3 Experiment 3: Vary the rate of UDP Traffic 62
3.5 FLCD IN PROPORTIONAL DIFFERENTIATED SERVICES (PROPDIFFSERV) IP NETWORKS 63
3.5.1 Implementation 63
3.5.2 Simulation Results and Performance Analysis 65
3.5.2.1 Experiment 4: TCP Traffic Simulation *66*
3.5.2.2 Experiment 5: Real Time Traffic Simulation *68*
3.6 CHAPTER SUMMARY 69

CHAPTER 4: ONLINE SELF LEARNING AND ORGANIZATION 71

4.1 INTRODUCTION 71
4.2 SELF-LEARNING AND ORGANIZATION STRUCTURES 72
4.2.1 RTT Based Queue Sampling Mechanism 72
4.2.2 The Self-Learning and Adaptation Mechanism 73
4.2.2.1 Evaluation of the Queue Error Factor *75*
4.2.2.2 Evaluation of the Packet Loss Factor *77*
4.3 SIMULATION RESULTS IN DYNAMIC IP ENVIRONMENTS 79
4.3.1 Experiment 1: Queue Evolution in Dynamic Traffic Environments 80
4.3.2 Experiment 2: Dynamic Traffic Environments with varying propagation delays 84
4.4 SELF-ORGANIZED FLCD ALGORITHM IN WIRELESS LOCAL AREA NETWORKS 87
4.4.1 The Need for Congestion Control in WLANs 87

4.4.2 Simulation Model and Results 89
4.5 CHAPTER SUMMARY 93
CHAPTER 5:THE DUAL EXPLICIT CONGESTION NOTIFICATION MECHANISM 95
5.1 INTRODUCTION 95
5.2 RELATED WORK 96
5.2.1 Wired Networks 96
5.2.1.1 The ECN Mark-front strategy 96
5.2.1.2 The Combined BECN/ECN Approach 97
5.2.2 Satellite Networks 97
5.2.2.1 The ECN Mark-front strategy 98
5.2.2.2 The Multilevel ECN Strategy 99
5.3 THE PROPOSED DUAL EXPLICIT CONGESTION NOTIFICATION ALGORITHM 99
5.3.1 Dual Explict Congestion Notification with Reverse Traffic Reduction 100
5.3.1.1 RTT Estimation 101
5.3.1.2 BECN Decay Function 101
5.3.2 Behaviour of an FLCD based Dual Explicit Congestion Notification Router 102
5.3.3 Behaviour of an ECN+BECN-Capable TCP end host 103
5.4 PERFORMANCE EVALUATION 103
5.4.1 Wired Network 104
5.4.2 Satellite Network 110
5.5 CHAPTER SUMMARY 113
CHAPTER 6: CONCLUSIONS AND FUTURE WORK 115
6.1 SUMMARY 115
6.2 FUTURE WORK 117
APPENDICES 119
REFERENCES 122

List of Figures

CHAPTER 1

FIGURE 1.1: RED AQM SCHEME 8

CHAPTER 2

FIGURE 2.1: BLUE AQM 20
FIGURE 2.2: ADAPTIVE RED INITIAL DROPPING PROBABILITY 22
FIGURE 2.3: WRED MECHANISM 27
FIGURE 2.4: FUZZY MEMBERSHIP FUNCTION 29
FIGURE 2.5: FUZZY LOGIC CONTROLLER 31
FIGURE 2.6: MEMBERSHIP FUNCTIONS FOR REN'S ALGORITHM 33
FIGURE 2.7: AFRED ARCHITECTURE 37

CHAPTER 3

FIGURE 3.1: FUZZY LOGIC CONGESTION DETECTION ARCHITECTURE 41
FIGURE 3.2: MEMBERSHIP FUNCTION FOR THE BACKLOG FACTOR 43
FIGURE 3.3: MEMBERSHIP FUNCTION FOR THE PACKET ARRIVAL FACTOR 44
FIGURE 3.4: MEMBERSHIP FUNCTION FOR THE CHANGE IN PACKET MARKING PROBABILITY 44
FIGURE 3.5: NETWORK TOPOLOGY 54
FIGURE 3.6: QUEUE LENGTH VERSUS LOSS RATE VERSUS LINK UTILIZATION 56
FIGURE 3.7: LINK UTILIZATION VERSUS QUEUE LENGTH VERSUS QUEUE VARIANCE 56
FIGURE 3.8: LINK UTILIZATION VERSUS QUEUE VARIANCE VERSUS LOSS RATE 56
FIGURE 3.9: QUEUE LENGTH VERSUS QUEUE VARIANCE VERSUS LOSS RATE 56
FIGURE 3.10: OBJECTIVE MEMBERSHIP INFERENCE FUNCTION 57
FIGURE 3.11: PACKET LOSS RATE 60
FIGURE 3.12: LINK UTILIZATION 60
FIGURE 3.13: DELAY FOR UDP TRAFFIC 61
FIGURE 3.14: JITTER FOR UDP TRAFFIC 61
FIGURE 3.15: FAIRNESS 62
FIGURE 3.16: FUZZY LOGIC CONGESTION DETECTION IN THE PROPDIFFSERV NETWORK 63
FIGURE 3.17: FLCD ALGORITHM IN PROPDIFFSERV IP SCENARIO 64
FIGURE 3.18: PROPDIFFSERV ENQUE AND DEQUE ROUTINES 65
FIGURE 3.19: PROPDIFFSERV NETWORK TOPOLOGY 66
FIGURE 3.20: BACKLOG FOR THE FLCD ALGORITHM 67
FIGURE 3.21: BACKLOG FOR THE WRED ALGORITHM 67
FIGURE 3.22: AVERAGE DELAY FOR REAL-TIME TRAFFIC 68
FIGURE 3.23: AVERAGE JITTER FOR REAL-TIME TRAFFIC 68

CHAPTER 4

FIGURE 4.1: SELF-LEARNING AND ADAPTATION ARCHITECTURE 78

FIGURE 4.2: SELF-LEARNING AND ADAPTATION ALGORITHM 79
FIGURE 4.3: NETWORK TOPOLOGY 80
FIGURE 4.4: QUEUE EVOLUTION FOR THE FOUR SCHEMES 82
FIGURE 4.5: LOSS RATE WITH VARYING ROUND TRIP LINK DELAY 84
FIGURE 4.6: LINK UTILIZATION WITH VARYING ROUND TRIP LINK DELAY 84
FIGURE 4.7: FAIRNESS WITH VARYING ROUND TRIP LINK DELAY 85
FIGURE 4.8: UDP TRAFFIC DELAY WITH VARYING ROUND TRIP LINK DELAY 85
FIGURE 4.9: UDP TRAFFIC JITTER WITH VARYING ROUND TRIP LINK DELAY 85
FIGURE 4.10: GATEWAY CENTRIC WLAN ARCHITECTURE 88
FIGURE 4.11: WIRELESS AP-TO-AP MESH NETWORK 89
FIGURE 4.12: WLAN SIMULATION TOPOLOGY 90
FIGURE 4.13: UDP TRAFFIC DELAY WITH VARYING BUFFER SIZE 91
FIGURE 4.14: UDP TRAFFIC JITTER WITH VARYING BUFFER SIZE 91
FIGURE 4.15: PACKET LOSS RATE WITH VARYING BUFFER SIZE 91
FIGURE 4.16: THROUGHPUT WITH VARYING BUFFER SIZE 92

CHAPTER 5

FIGURE 5.1: FLCD ARCHITECTURE WITH BECN ACTIVATION MECHANISM 100
FIGURE 5.2: BECN PACKET MARKING ALGORITHM 102
FIGURE 5.3: ECN PACKET MARKING ALGORITHM 103
FIGURE 5.4: WIRED NETWORK SIMULATION TOPOLOGY 104
FIGURE 5.5: ISQ REVERSE TRAFFIC 105
FIGURE 5.6: QUEUE LENGTH 105
FIGURE 5.7: PACKET LOSS RATE 108
FIGURE 5.8: BOTTLENECK LINK UTILIZATION 108
FIGURE 5.9: SATELLITE NETWORK SIMULATION TOPOLOGY 110
FIGURE 5.10: ISQ REVERSE TRAFFIC 111
FIGURE 5.11: PACKET LOSS RATE 111
FIGURE 5.12: BOTTLENECK LINK UTILIZATION 112
FIGURE 5.13: GOODPUT 112

List of Tables

CHAPTER 2

TABLE 2.1: FUZZY CONTROL RULES FOR REN'S ALGORITHM 34

TABLE 2.2: LINGUISTIC RULES FOR FUZZY BLUE CONTROLLER 36

CHAPTER 3

TABLE 3.1: RULE BASE FOR THE FLC UNIT 42

TABLE 3.2: COMPARISON OF AQM SCHEMES WITH PACKET DROPPING 59

TABLE 3.3: PARAMETERS FOR TCP TRAFFIC SIMULATION 66

TABLE 3.4: PERFORMANCE RESULTS FOR TCP TRAFFIC SIMULATION 66

TABLE 3.5: PARAMETERS FOR REAL-TIME TRAFFIC SIMULATION 68

CHAPTER 4

TABLE 4.1: QUEUE LENGTH EVOLUTION STATISTICAL RESULTS 81

TABLE 4.2: REAL TIME TRAFFIC SIMULATION PARAMETERS 90

List of Abbreviations

ACK	Acknowledgement
AFRED	Adaptive Fuzzy Random Early Detection
AIMD	Increase Multiplicative Decrease
ANN	Artificial Neural Networks
ARPANET	Advanced Research Projects Agency Network
AQM	Active Queue Management
AVQ	Adaptive Virtual Queue
BECN	Backward Explicit Congestion Notification
BS	Buffer Size
CI	Computational Intelligence
DiffServ	Differentiated Services
DoS	Denial of Service
DSCP	Differentiated Services Code Point
DWDM	Dense Wavelength Division Multiplexing
EC	Evolutionary Computation
ECN	Explicit Congestion Notification
EMOO	Evolutionary MultiObjective Optimization
EUNITE	European Network for Intelligent Technologies
FAFC	Fast Adaptive Fuzzy Controller
FIFO	First In First Out
FLCD	Fuzzy Logic Congestion Detection
FTP	File Transfer Protocol
GA	Genetic Algorithm
GSO	Geostationary Orbit

ICMP	Internet Control Message Protocol (defined in RFC 792)
ID	Identity
IETF	Internet Engineering Task Force
IntServ	Integrated Services
IP	Internet Protocol (defined in RFC 791)
ISQ	ICMP Source Quench
LAN	Local Area Network
MOEA	MultiObjective Evolutionary Algorithm
MOPSO	Multi-objective Particle Swarm Optimization
OSI	Open System Interconnection
PHB	Per-hop Behaviour
PI	Proportional Integral
PID	Proportional Integral Derivative
PropDiffServ	Proportional Differentiated Services
PSO	Particle Swarm Optimization
QoS	Quality of Service
RED	Random Early Detection
REM	Random Exponential Marking
RFC	Request for Comments
RSVP	Resource ReSerVation Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VoIP	Voice over IP
WLAN	Wireless Local Area Network
WFQ	Weighted Fair Queuing
WWW	World Wide Web
RTT	Round Trip Time
RTO	Retransmission TimeOut

Chapter 1

Introduction

1.1 General

The Internet came into existence as an outgrowth of the Advanced Research Projects Agency Network (ARPANET), a United States Department of Defense project. From a mere interconnection of disparate computer systems, it has evolved into one of mankind's greatest achievements. Apart from being the universal source of information, the Internet is the most democratic of all the mass media. Consequently, the world has witnessed an exponential growth of the Internet over the past two decades [ZAK05]. It has actually grown from connecting 213 hosts in 1981 to over 353 Million in 2005. The number of World-Wide-Web (WWW) sites has grown from 1 in 1990 to over 70 Million in 2005 while the number of users has grown from 16 Million in December 1995 to over 1 Billion in December 2005. The Internet has also experienced a rapid introduction of applications. Some of the major applications include the WWW, File Transfer Protocol (FTP), Email, Video and Audio streaming, Voice-over-IP (VoIP) and ecommerce. The transmission media have also evolved immensely such that in addition to the traditional wired links, wireless systems and dense wavelength division multiplexing (DWDM), which offer multi-gigabit capacities, are becoming more and more ubiquitous.

The success of the Internet can largely be attributed to the strength of its underlying protocol suite, the Transmission Control Protocol/Internet Protocol (TCP/IP). TCP is a higher layer protocol which divides a message or file into TCP *segments* which are then packaged into packets, containing addresses of the source and the destination. TCP provides a reliable, in-order delivery from the source to destination. On the other hand, IP is a lower layer protocol which

handles the address part of each packet so that it gets to the right destination. When these packets arrive at the destination, they are reconverted into TCP segments by removing the source and destination addresses. The TCP receiver reassembles these segments into the original message. To ensure reliable delivery, the TCP receiver sends an acknowledgement (ACK) for every segment received. Every segment from the sender is sent with a sequence number identifying which bytes the segment contains. ACKs are cumulative. They identify the sequence number of the next in-order byte the receiver expects to receive from the sender. A TCP sender uses these ACKs to compute the *send window*, which roughly keeps track of how much data has been sent but not yet acknowledged. To provide in-order delivery, the TCP receiver must buffer any segments that are received out-of-order until gaps in the sequence number space have been filled. In each ACK, a TCP receiver includes the amount of space it has left in its buffer. This amount of space is called the receiver's advertised window *rwnd* . Upon receiving this window update, a TCP sender will not allow more than that amount of data to be unacknowledged in the network (i.e., if there is no room in the receiver's window, no new data will be sent). This is how TCP performs *flow control*. The goal of flow control is to make sure that the sender does not overrun the receiver's buffer. For flow control, the send window cannot be larger than the receiver's window.

Although TCP/IP has sustained the Internet for a long time, it was not designed to work optimally. The enormous growth of the Internet in terms of demand for access from its users and the increasing demand for new applications has really exposed the weaknesses in the TCP/IP protocol suite. One of the major weaknesses of TCP/IP relates to its failure to address the problems of congestion. These problems arise because the Internet is essentially a network of interconnected queues in which packets are switched from their respective sources to their destinations. Routers and switches contain queues which are used for buffering packets when the instantaneous arrival rate of packets is greater than the outbound traffic rate. These queues are generally first in/first out (FIFO) and have finite capacity. A new packet at a router/switch must wait for the packets in front of it to be transmitted first before it can be transmitted. If the queue is full, the packet is dropped. Queuing delays slow down the delivery of data from the sender to the receiver. This decreases the performance of applications from the perception of the user. It also affects the quality of service requirements of interactive applications such as telephony, video conferencing and interactive games since these applications require to be delivered quickly within certain delay constraints. The side effect of this problem is the issue of delay variation which causes jitter in the delivery of these bandwidth-sensitive applications. Apart from wasting

resources, lost packets have a major effect on the performance of TCP. TCP ensures a reliable delivery of segments, so if a TCP segment is dropped, subsequently received segments cannot be delivered to the application layer until the dropped segment has been successfully received. When a segment has been dropped, TCP detects the drop and retransmits the lost segment. This results in increased delays for the user. These problems call for an urgent need to re-examine the current Internet congestion control mechanisms and improve them in light of the unprecedented growth of the Internet.

1.2 An Overview of Internet Congestion Control

The need for Internet congestion control originally became apparent during several periods of 1986 and 1987, when the Internet experienced the "congestion collapse" condition predicted by Nagle [NAG84].The network was so overloaded with retransmissions of lost data such that no new data could get through. During this period, a large number of widely dispersed Internet sites experienced simultaneous slowdown or cessation of networking services for prolonged periods. The world was at the brink of an Internet meltdown [JK88],[FLO00a]. This condition triggered a wave of relentless research efforts which are still going on. It is now generally accepted in the Internet community that the problem of network congestion control remains a critical issue and a high priority one, especially given the growing size, increasing demand for new services with varying quality of service characteristics, and higher speed (bandwidth) demanded from an increasingly integrated services network. Research in this aspect has evolved along three interrelated fronts namely: *end-to-end mechanisms*, *router (gateway) based detection mechanisms* and *explicit notification mechanisms*.

1.2.1 End-to-end Mechanisms

End-to-end mechanisms attempt to address the congestion problem by making changes to TCP.They try to detect congestion by monitoring end-to-end measurements. The original TCP [CK74, CDS74] detected segment loss by setting a Retransmission Timeout (RTO) timer when a segment was sent. If the timer expired before the ACK for that segment was received, the segment was assumed to be lost and all segments starting at that sequence number were retransmitted (this scheme is known as "Go-Back-N"). Only flow control was implemented in TCP. Nothing in TCP dictated what should be done when congestion was encountered in the

network. In an immediate response to the congestion collapse scenario, Jacobson and Karels [JK88] implemented the first attempt at TCP congestion control which consists of several changes to the original TCP. These changes include the addition of a slow start phase, a congestion avoidance phase, and a fast retransmit phase. The modified TCP was called *TCP Tahoe*.

1.2.1.1 TCP Tahoe

TCP Tahoe requires each side of the connection to keep track of two additional variables: the congestion window *cwnd* and the threshold *ssthresh*. Before the introduction of this mechanism, the amount of data that the sender could inject into the network was limited by *rwnd* only. The introduction of *cwnd* imposed an additional constraint on how much traffic a host can send into a connection. Specifically, the amount of unacknowledged data that a host can have within a TCP connection may not exceed the minimum of *cwnd* and *rwnd*.

Before TCP Tahoe was introduced, TCP senders could send out segments as fast as possible at startup. The TCP senders, though, have no indication of how much data the network can handle at once, so often, these bursts led to packets being dropped at routers. TCP Tahoe introduced the *slow start* algorithm which is called into play either when a TCP connection starts up or after a packet loss. The congestion window *cwnd* is set to one segment at startup. TCP sends the first segment into the network and waits for an acknowledgement. If this segment is acknowledged before its timer runs out, the sender increases the congestion window by one and sends out two segments. If these segments are acknowledged before their timeouts, the sender increases the congestion window by one segment for each of the acknowledged segments, giving a congestion window of four segments, and sends out four segments. This procedure continues as long as (1) $cwnd < ssthresh$ (2) the acknowledgements arrive before their corresponding timeouts. During this phase of the congestion control procedure, the transmission rate starts slowly but accelerates rapidly afterwards. This enables TCP to slowly probe the network to determine the available capacity, in order to avoid congesting the network with an inappropriately large burst of data.

When $cwnd > ssthresh$, slow start ends and *congestion avoidance* begins. Congestion avoidance probes for additional bandwidth by linearly increasing the transmission rate. Once in congestion avoidance phase, *cwnd* is increased by $1/cwnd$ of a segment for each ACK received. The idea is

that in one round trip time (RTT), a TCP sender with a window of size *cwnd* will receive at most *cwnd* ACKs, so this results in a congestion window increase of at most one segment every RTT. This linear increase contrasts with slow start, which is an exponential increase with *cwnd* doubling every RTT. The congestion avoidance phase continues as long as the acknowledgements arrive before their corresponding timeouts. But the window size, and hence the rate at which the TCP sender can send, can not increase forever. Eventually, the TCP rate will become higher such that one of the links along the path becomes saturated. At this point, packet loss (and a resulting timeout at the sender) will occur. When a timeout occurs, the value of *ssthresh* is set to half the value of the current *cwnd* , and *cwnd* itself is reset to one segment. The sender then again grows the congestion window exponentially fast using the slow start procedure until $cwnd > ssthresh$, after which congestion avoidance takes over again. The TCP window adjustment mechanism is generally known as the Additive Increase Multiplicative Decrease (AIMD) algorithm. This is because TCP essentially increases its window size by one every RTT (and thus increases its transmission rate by an additive factor) when its network path is not congested, and decreases its window size by a factor of two every RTT when the path is congested.

The fast retransmit mechanism is a faster way of detecting segment loss which works by inferring segment loss through the receipt of three duplicate acknowledgements. Whenever a receiver receives an out-of-order segment (e.g., a gap in sequence numbers), it sends an acknowledgement for the last in-order segment it received, which would be a duplicate of the previous acknowledgement sent. The sender uses the receipt of three duplicates of the same ACK to infer that there was segment loss rather just segment re-ordering. The advantage of this mechanism is that it reduces the amount of time needed to detect a segment loss. Without fast retransmit, the expiration of the RTO timer would be required to detect loss. For flows with large congestion windows, multiple acknowledgements will typically arrive in one RTT.In this way, fast retransmit allows TCP to avoid large timeouts during which no data can be sent. When the third duplicate ACK is received, TCP performs a retransmission of what appears to be the missing segment, without waiting for the RTO timer to expire. The congestion window is set to 1 segment. Duplicate ACKs may continue to arrive, but no change is made to *cwnd* and hence no data segments are sent. When the ACK, that acknowledges that the lost segment has been successfully received, returns, *cwnd* is incremented to 2, transmission resumes with the next two segments in the pipeline. From this point, slow start continues as normal.

1.2.1.2 TCP Reno

In 1990, a feature known as fast recovery was added to TCP Tahoe by Van Jacobson. The new TCP is known as TCP Reno and is the de facto standard version of TCP on the Internet [APS99].When three duplicate ACKs are received, fast recovery is entered instead of slow start. In this mode, *ssthresh* is set to $1/2cwnd$ and *cwnd* is set to $ssthresh+3$ (one for each of the three duplicate ACKs, which imply that segments have left the network. For each additional duplicate ACK received, *cwnd* is incremented by one segment, as in slow start. New segments can be sent as long as *cwnd* allows. When the ACK arrives for a retransmitted packet, *cwnd* is set back to *ssthresh*.TCP then leaves fast recovery and returns to congestion avoidance. This mechanism enables the sender to probe for available bandwidth conservatively with less chance of overflowing network queues than with using slow start.

1.2.1.3 TCP New Reno

Recently, TCP New Reno has been proposed [FH99]. TCP New Reno addresses some of the problems encountered with TCP Reno. The fundamental problem with TCP Reno is that the first partial ACK brings the sender out of the fast recovery phase. This will result in the requirement of timeouts when there are multiple losses in a window, and thus stalling the TCP connection. TCP New Reno solves this problem by using a partial ACK as an indication of another lost packet and as such the sender retransmits the first unacknowledged packet. Unlike Reno, partial ACKs don't take New Reno out of Fast Recovery. This way, it retransmits one packet per RTT until all the lost packets are retransmitted and avoids requiring multiple fast retransmits from a single window of data.

1.2.2 Router-Based Detection Mechanisms

Router based mechanisms operate by detecting incipient congestion at the router queues and notifying the senders so that they reduce their transmission rates. Jacobson and Karels pointed out that the end-to-end mechanisms, while necessary and powerful, were not sufficient to provide good service in all circumstances [JK88]. They observed that while transport endpoints can ensure that network capacity is not exceeded, they cannot ensure fair sharing of that capacity. They further noted that enough information to control sharing and fair allocation is found only in

gateways (routers) at the convergence of flows [JK88]. Floyd and Jacobson [FJ93] further pointed out that the gateway can reliably distinguish between propagation delay and persistent queuing delay. Only the gateway has a unified view of queuing behaviour over time; the perspective of individual connections is limited by the packet arrival patterns for those connections. The default router algorithm for detecting congestion is known as the *drop-tail* mechanism. Routers usually employ the drop-tail technique along with FIFO-based queue scheduling. The first packet that arrives at the router is the first to be transmitted. The packet that arrives last is the likely candidate to be dropped in the event that the queue reaches its maximum length. Braden *et al.* [BRA98] noted that the drop-tail technique had served the Internet quite well for years but it has two major setbacks: lockout and full queues. The lock out phenomenon may occur when drop-tail allows a few connections or flows to monopolize queue space. This usually happens as a result of synchronization or other timing effects. The full queue phenomenon happens because of the nature of the drop-tail mechanism itself. The drop-tail mechanism allows queues to maintain a full queue status for long periods of time, since drop-tail signals congestion (via a packet drop) only when the queue has become full. Besides these two major setbacks, this mechanism is also biased against bursty traffic. The burstier the traffic from a particular connection, the more likely it is that the gateway queue will overflow when packets from that connection arrive at the gateway [FJ92].

1.2.2.1 Random Early Detection

Random Early Detection (RED) [FJ93], proposed by Floyd and Jacobson, marks one of the greatest milestones in router based congestion detection. The main goal behind RED is to provide congestion avoidance by controlling the average queue size. Congestion notification is performed by dropping packets. When packets are dropped, the sender TCP detects that there is congestion on the network either through duplicate ACKs or timeouts. Once congestion has been detected TCP invokes the congestion avoidance algorithms presented in Section 1.2.1.

RED uses a weighted average queue size and two thresholds min_{th} and max_{th} .When the weighted average queue size is below min_{th}, RED acts like a drop-tail router and forwards all packets. This is deemed as a congestion free zone. When the weighted average queue size is between min_{th} and max_{th}, RED drops the incoming packets probabilistically. All incoming packets are dropped when the weighted average queue size is greater than max_{th} .The RED AQM

scheme is illustrated in Figure 1.1. BS denotes the maximum buffer size which is usually set to $2 * max_{th}$.

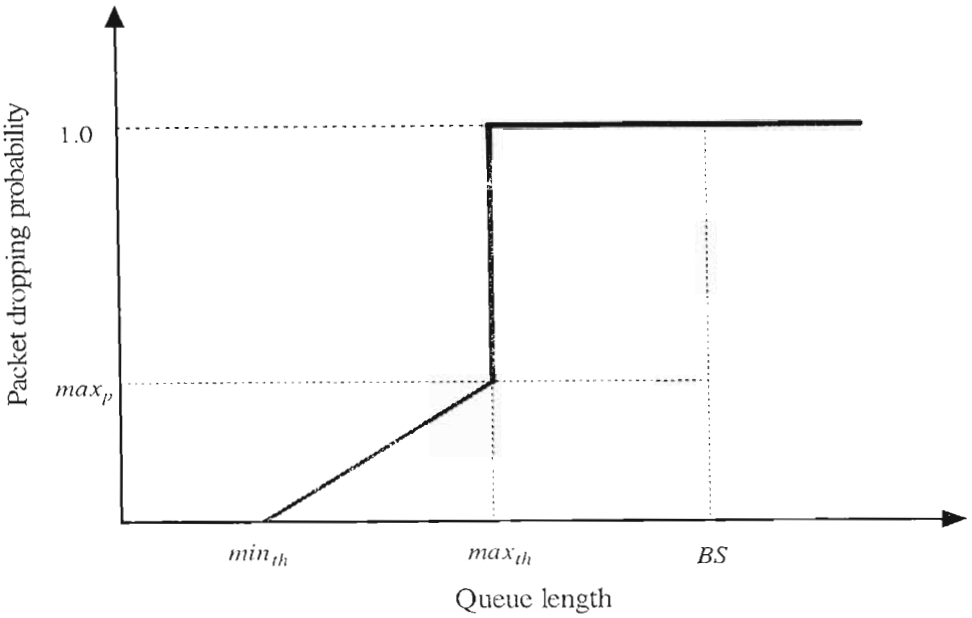


Figure 1.1: RED AQM Scheme

The equations that RED uses to compute the weighted average queue size and the drop probability are as follows:

- Exponential weighted moving average:

  $Avg_{new} = (1 - w_Q) Avg_{old} + w_Q * Q$

  where Q is the current queue length and w_Q is the weight parameter

- Drop probability: $p_a = p_b / (1 - count * p_b)$, where

  $p_b = (max_p (Avg - min_{th})) / (max_{th} - min_{th})$ and $count$ is the number of undropped packets since the last dropped packet. Parameter max_p denotes the maximum packet dropping probability before RED starts dropping all incoming packets.

RED ensures that packets are dropped in proportion to the input rates of the connections. Connections with higher input rates receive more drops of packets than connections with lower input rates. By so doing, RED tries to maintain equal rate allocation and removes biases against bursty connections. By using probabilistic packet dropping RED also eliminates global synchronization exhibited by the drop-tail approach. The Internet Engineering Task Force (IETF)

recommends the deployment of RED in RFC 2309 [BRA98]. Although RED has been widely implemented in today's routers, it has largely remained not switched on because there are still lots of doubts concerning its ability to address the congestion problem. The major weakness of RED relates to the setting of its four operational parameters: thresholds min_{th} and max_{th}, the weight parameter w_Q and the maximum packet marking probability max_p. The efficiency of RED is completely dependent on the proper configuration of these parameters [FKS99]. In [LOW02], Low *et. al.* perform a control-theoretic analysis of TCP/RED and points out that RED becomes unstable as RTT delay increases, or when network capacity increases. It has further been questioned whether RED really gives any benefit over drop-tail [MBD99].

1.2.2.2 Post RED Mechanisms

A plethora of router based congestion detection mechanisms, now generally known as Active Queue Management (AQM) has been proposed in order to address the weaknesses of RED [OLW99] [FEN99] [PPP00] [ALLY01] [FYX02]. Based on their architecture and principles of operation, these mechanisms can be classified into two broad categories: traditional (analytical) or fuzzy logic based.

Traditional (Analytical) Techniques

Traditional techniques attempt to address the weaknesses of RED by using formal mathematical models of the system and traditional control theoretic tools. The key traditional AQM algorithms include:

- BLUE [FEN99]: This algorithm uses packet loss and link-idle events rather than the queue length to control congestion. BLUE increases the packet drop probability in response to a buffer overflow (i.e., a packet drop) and decreases the packet drop probability when the link becomes idle.
- CHOKe [PPP00]: CHOKe is short for "CHOose and Keep for responsive flows, CHOose and kill for unresponsive flows. This is a stateless algorithm that attempts to ensure a fair bandwidth allocation to all the flows that share a FIFO based outgoing link of a congested router. It accomplishes this by dropping more packets from high-bandwidth unresponsive flows. The essence of this algorithm is that, when a packet arrives, a random packet is picked from the queue. If the randomly chosen packet is from the same source as the newly arrived packet, both packets are dropped.

- Adaptive Virtual Queue (AVQ) [KS01] uses a modified token bucket model as a virtual queue (VQ) to regulate buffer utilization rather than the queue length. AVQ adjusts the size and link capacity of the VQ proportional to the measured input rate and drops packets when the VQ overflows.
- Adaptive RED (ARED) [FGS01] attempts to maintain suitable operating parameters in RED by dynamically adjusting max_p in the RED algorithm based on observed queue length dynamics. ARED increases max_p when weighted average queue length exceeds the target queue length and decreases max_p when weighted average queue length goes below the target queue length.
- Random Exponential Marking (REM) [ALLY01] is an optimization based scheme for communicating congestion from links to sources by exponential marking. REM uses pricing algorithm to determine the congestion measure. The congestion measure is a function of the rate mismatch and the queue mismatch.
- Proportional Integral (PI) [HMTG01] is a control theoretic approach which regulates the queue length to a reference value by using instantaneous samples of the queue length taken at a constant sampling frequency as its input.
- GREEN [WZ02] is a feedback control function which adjusts the rate of congestion notification in response to the flow based congestion measure which denotes the estimated data arrival rate above the target link capacity.

Fuzzy Logic Based Techniques

Fuzzy logic is one of the tools that constitute what is commonly as Computational Intelligence (CI). The other components of CI are artificial neural networks (ANNs) and evolutionary computation (EC). Fuzzy logic was invented by Zadeh [ZA65] for handling uncertain and imprecise knowledge in real world applications. It has proved to be a powerful tool for decision-making, and manipulating imprecise and noisy data. Fuzzy control was introduced by Mamdani [MAM74] for controlling complex processes. Unlike classical logic which requires a deep understanding of a system, exact equations, and precise numeric values, fuzzy logic incorporates an alternative way of thinking, which allows modeling complex systems using a higher level of abstraction originating from our knowledge and experience. Fuzzy Logic has been found to be very suitable for embedded control applications. Several manufacturers in the automotive and aerospace industry are using fuzzy technology to improve quality and reduce development time. In consumer electronics, fuzzy logic improves time to market and helps to reduce costs. In

manufacturing, fuzzy logic is proven to be invaluable in increasing equipment efficiency and diagnosing malfunctions. Some of the benefits of Fuzzy Logic include performance, simplicity, lower cost, and productivity.

Li and Lee [LL89] and Prade [PRA80] stress that in practical queuing systems, the mean of the arrival rate and the mean service rate are frequently fuzzy ie. they cannot be expressed in exact terms. It is therefore very difficult to solve queuing problems by using traditional analytical techniques. Based on these developments, a number of fuzzy logic based congestion detection algorithms have been proposed with satisfactory results since 2003.The performance of these algorithms, in terms of packet loss rate, link delay and stability, is generally better than that of traditional approaches. The European Network for Intelligent Technologies (EUNITE) Roadmap [SLM04] further points out that the application of fuzzy control techniques to the problem of congestion control in IP-based networks is suitable due to the difficulties in obtaining a precise mathematical model using conventional analytical methods.

The general trend, in Fuzzy Logic based congestion detection techniques [FYX02] [CHR03a] [WAN03] [ANN04], is that they use queue length and/or traffic arrival rate as input variables. The system output is the probability which is used in the congestion notification process. The control law of the fuzzy algorithm is encapsulated in a set of simple linguistic rules and membership functions which are jointly known as the *rule base*. The efficiency of these algorithms is therefore largely dependent on the proper design of the rule base.

1.2.3 Explicit Notification Mechanisms

Efficient delivery of congestion signals is essential to the performance of the Internet. Packet dropping is the default method used by IP routers to inform the senders about their load levels. The senders detect these packet drops by using retransmission timeouts and the fast retransmit mechanism. Then they respond by limiting their packet injection rate in order to match the available network capacity. This method of delivering congestion signals has a number of disadvantages. The packet drops not only increase the amount of traffic in the network due to retransmissions but also add a large transfer delay. This mechanism also proves to be expensive because the dropped packets will have traversed a larger portion of the network by the time they get dropped. Therefore, a search for mechanisms which would offer direct feedback to the senders without dropping the packets became necessary. In the following subsections, we discuss

the explicit congestion notification mechanisms that are available in literature. We also discuss the explicit underutilization notification mechanisms. These mechanisms are tailored for high bandwidth capacity links (greater than 1 Gbps) where the link is underutilized i.e. congestion is not a concern.

1.2.3.1 Internet Control Message Protocol (ICMP) Source Quenches (ISQ)

The Internet Control Message Protocol (ICMP) was designed to be an integral part of IP [POS81]. ICMP enables the router or the destination host to communicate with the source host occasionally. The Source Quench (SQ) is an example of the messages sent by using ICMP. Using the drop-tail mechanism, a router discards incoming data packets if it does not have the buffer space needed to queue them for output to the next network on the route to the destination network. If a router discards a packet, it may send an ISQ message to the source host of the packet. A destination host may also send an ISQ message if packets arrive too fast to be processed. The ISQ message is a request to the host to cut back the rate at which it is sending traffic into the network. The router may send an ISQ message for every packet that it discards. On receipt of an ISQ message, the source host cuts back the rate at which it is sending traffic to the specified destination until it no longer receives source quench messages from the router. The source host can then gradually increases the rate at which it sends traffic to the destination until it again receives source quench messages.

RFC 1254 [MR91] points out that although ISQ messaging was well defined, the conditions for ISQ generation at the router and the appropriate reaction at the source host were not implemented in a standardized way. RFC 1812 [BAK95] further points out that the generation of ISQ messages creates an extra overhead on router resources (e.g. memory, processing time). The transmission of ISQ messages adds traffic in the reverse direction on what might be a congested path. Based on these observations, RFC 1812 disapproves the generation of ISQs from a router or a destination host but also specifies that a router that generates ISQs must be able to limit the rate at which they are sent. With these developments, interest in ISQ messaging as a mechanism for explicit congestion notification died down such that in the implementation of RED and all subsequent AQM algorithms, this mechanism was not supported.

1.2.3.2 Explicit Congestion Notification (ECN)

A major turning point in explicit congestion notification took place with the introduction of the ECN approach [FLO94], [RF01]. Two bits in the IP header have been reserved for this purpose. One of these bits is known as the *ECN Capable Transport (ECT)* bit. A packet with the ECT bit set i.e. equal to one, informs every ECN capable router along its path that its TCP sender is ECN capable. When the ECN capable router detects congestion, it marks the packet's second reserved bit which is known as the *Congestion Experienced (CE)* bit. It then forwards the packet to the next link. The packet traverses the network until it reaches its destination. Upon receiving a packet with the CE bit marked, the receiver echoes the ECN bit back to the source in the TCP header of the returning ACKs. The sender responds to ECN by halving the congestion window thereby reducing packet transmission rate. This happens once in an RTT. After responding to ECN, the sender sets the Congestion Window Reduced (CWR) bit in its TCP header. This serves to inform the receiver that action has been taken in response to the congestion signal. In order to act against the loss of ACKs, the receiver continues to set the ECN-Echo bit in subsequent ACKs even if further packets do not have the CE bit set until it receives a packet with CWR bit set in the TCP header.

Studies have shown that the use of the ECN mechanism for the notification of congestion to the end nodes prevents unnecessary packet drops and retransmissions [FLO94], [SA00]. A second benefit of ECN is that the sources can be informed of congestion quickly and unambiguously, without the source having to wait for either a retransmit timer or three duplicate *ACKs* to infer a dropped packet. For those cases where a dropped packet is not detected by the *Fast Retransmit* procedure, the use of ECN mechanisms can improve a bulk-data connection's response to congestion. ECN's downside relates to the long delay experienced in congestion notification. Under heavy load and large delay links such as satellite links, congestion will persist for a long time. These lead to higher queue variance, reduced throughput and longer transfer delays for short lived flows.

1.2.3.3 Backward Explicit Congestion Notification (BECN)

Recently, ISQ messaging has been revived with the introduction of Backward Explicit Congestion Notification (BECN) [HNS98].The BECN proposal clearly defines the guidelines for generating ISQs and responding to them in a TCP/IP network. It sorts out the reservations

expressed against the use of ISQs in [MR91].It also points out that the problem of extra router overhead and increased reverse network traffic generated by ISQs would no longer be a big concern as stated in [BAK95] because the ISQs would be generated only when the computed RED probability requires dropping or marking. Studies [AKU02] comparing ECN and BECN have shown that BECN exhibits lower transfer delay for interactive TCP applications, and improves goodput for bulk TCP applications.

BECN's major downside relates to the generation and transmission of ISQs. Therefore it is generally desirable to minimize the generation of ISQs in high speed routers. Another drawback of BECN is lack of reliability. In contrast to ECN, which ensures that the destination host sends ECN-Echo ACKs continuously until the sender notifies it that congestion notification has been received, BECN does not guarantee reliability because ISQ loss can not be detected by the sender.

1.2.3.4 Explicit Underutilization Notification Mechanisms

The main goal in these mechanisms is to allow TCP to switch off its traditional congestion avoidance algorithms and exponentially increase its sending rate so as to utilize the vast network capacity at its disposal. Router based mechanisms that have been proposed in this area include:

- *Anti-ECN*: In contrast to ECN, the Anti-ECN [KUN03] proposal is a simple scheme which uses a single bit in the packet header to allow a TCP connection to increase its sending rate aggressively over an underutilized high capacity link. It uses aggregate information to provide feedback and does not require the routers to maintain per flow state. The senders can increase their rates even in the middle of a transfer.
- *Quick-Start*: Quick-Start [SAF05] is a collaborative effort between sources and routers. A TCP source sends a packet that includes a Quick-Start Request in an IP option containing the requested rate, say X bytes/sec. Each router along the path either indicates agreement with the request or lowers the requested sending rate or implicitly signals that the Quick-Start option was not processed and hence the request was not approved. The data receiver reports the information received in the Quick-Start Response in a TCP option, and the data sender determines if all of the routers along the path have agreed to the request and sets the sending rate appropriately. The assumption is that routers will only approve Quick-Start requests when they are underutilized.

1.3 Motivation and Focus of this Thesis

Work on this thesis has been spurred by a number of factors. Firstly, it has been observed that even though literature is replete with AQM proposals, none of these mechanisms has really been implemented and activated in commercial routers. Practically speaking, routers are still using the drop-tail mechanism in order to detect congestion. Bitorika *et al.* [BRH04] carried out an evaluation of key AQM algorithms that had been proposed between 1999 and 2003. Their major finding was that each algorithm performs well only for specific metrics. None of the algorithms exhibited global optimal performance. It is worthy pointing out that AQM algorithms evaluated by Bitorika *et al.* are all based on formal analytical techniques. The introduction of fuzzy logic based AQM techniques, which perform better than their analytical counterparts, gives the Internet community a lot of hope. However, a crucial problem with fuzzy AQM algorithms relates to the design of linguistic rules and their membership functions. The fuzzy AQM schemes in literature rely on the expert knowledge of the designer who performs the rigorous process of tuning the fuzzy parameters until optimal performance is reached (from the designer's perspective). The human factor involved in this operation makes it difficult for these algorithms to achieve optimal performance for all the key AQM objectives. Therefore, this thesis focuses on finding a method of designing the fuzzy AQM scheme that achieves optimal performance in all the major performance metrics of Internet congestion control without relying on the designer's expert knowledge.

Secondly, we have noticed that, in trying to keep up with the variations, unmodelled system dynamics and other disturbances on the Internet, a few adaptive fuzzy AQM schemes have been proposed. Much as these algorithms perform better than the traditional AQM approaches, the manner in which they adapt themselves to the network is still based on the same analytical methods. Therefore, they intrinsically carry along with them the weaknesses of those methods. A look on the other hand shows that there is a large body of research works on online adaptation and self learning fuzzy logic systems [PM79], [PRF99], [SBM02], [PRG04]. This thesis therefore uses some principles learnt from these systems in order to develop online self-learning and organization structures for the fuzzy AQM scheme.

The third motivating factor for this work relates to the desire to implement an explicit congestion notification mechanism that combines the ECN and BECN approach so that the two approaches must complement each other. The first proposal to combine these mechanisms is based on RED

[AKU03]. Considering the weaknesses of RED and the superiority of fuzzy logic based AQM schemes, we felt that a dual explicit congestion notification mechanism based on fuzzy logic would perform better than the RED based proposal. This thesis also extends this concept to satellite networks which are characterized by long delays.

1.4 Empirical Validation

Throughout this research, we use the network simulator ns-2[NS05] for validation of all the proposed models, protocols and algorithms and for conducting performance comparison with various prior works in related areas. We developed our own ns-2 simulation source codes for the new algorithms and integrated them in the ns-2 package. In all simulations involving web traffic, we use the standard web traffic generator included with ns-2, with the following parameter settings: an average of 30 web pages per session, an inter-page parameter of 0.8, an average page size of 10 objects, an average object size of 400 packets and a Pareto II shape parameter of 1.002.

1.5 Thesis Organization

Chapter 2: Literature Survey of post-RED AQM algorithms. This Chapter begins by presenting the principles of operation, the efficiencies and deficiencies of the key analytical AQM schemes. The chapter continues by presenting the fuzzy logic control theory before reviewing the fuzzy AQM schemes in terms of their design principles, efficiencies and deficiencies.

Chapter 3: Fuzzy Logic Congestion Detection Algorithm Design using MOPSO. In this chapter, we propose a Fuzzy Logic Congestion Detection (FLCD) algorithm which combines the good attributes of both the traditional AQM approaches and the fuzzy logic based algorithms that have been reported in literature. We also introduce new concepts in order to address some of the demerits observed in Chapter 2. The membership functions (MFs) of the FLCD algorithm are designed automatically by using a Multi-objective Particle Swarm Optimization (MOPSO) algorithm in order to achieve optimal performance on all the major performance metrics of IP congestion control. The performance of this new algorithm is compared with that of the fuzzy logic based congestion control algorithm in [FYX03] and the Random Explicit Marking (REM), a highly rated analytical AQM algorithm. This Chapter extends the FLCD algorithm to Proportional Differentiated Services IP networks.

Chapter 4: Online Self-learning and Organization. This Chapter enhances the performance of the FLCD algorithm by proposing two online self organization structures that enable the FLCD algorithm to learn the system conditions and adjust itself accordingly thereby achieving optimal performance in dynamic traffic environments and a wide range of topologies. The first self organization structure adjusts the update interval in line with the prevailing link propagation delay. This would help to improve the FLCD algorithm's performance with respect to TCP traffic transmissions which depend on the value of the RTT. The second one implements a self-learning and adaptation mechanism based on concepts borrowed from the self organized fuzzy controllers in [PM79], [PRF99], [SBM02], [PRG04].

Chapter 5: A Dual Explicit Congestion Notification Mechanism. This Chapter proposes a fuzzy logic based dual explicit congestion notification mechanism which combines the merits of the Explicit Congestion Notification (ECN) and the Backward Explicit Congestion Notification (BECN) mechanisms. This Chapter also proposes an RTT based decay function which reduces the amount of reverse traffic without jeopardizing the performance of the BECN mechanism.

Chapter 6: Conclusion and Future Work. This Chapter presents conclusions drawn in this dissertation and gives direction for future work.

1.6 Original Contributions of this Thesis

The key contributions of this research are summarized as follows:

1. A proposal for a multi-objective particle swarm optimized Fuzzy Logic Congestion Detection (FLCD) mechanism in Chapter 3.
2. An extension of the Fuzzy Logic Congestion Detection algorithm to the Proportional Differentiated Services IP Networks in Chapter 3
3. A proposal for self-learning and organization structures for the FLCD algorithm in Chapter 4.
4. A proposal for a fuzzy logic based dual explicit congestion notification mechanism and a proposal for an RTT based BECN reverse traffic reduction mechanism in Chapter 5.

1.7 Published Work

[1] C.N. Nyirenda, D.S. Dawoud, "Performance Evaluation of the Swarm Optimized Fuzzy Logic Congestion Detection Mechanism in Proportional Differentiated Services IP Networks", *In proceedings of the Southern African Telecommunications Networks and Applications Conference (SATNAC 2006)*, Cape Town, South Africa, 3-6 September 2006.

[2] C.N. Nyirenda, D.S. Dawoud, "Multi-objective Particle Swarm Optimization for Fuzzy Logic Based Active Queue Management", *In Proceedings of the 15th IEEE International Conference in Fuzzy Systems as part of the Fourth IEEE World Congress on Computational Intelligence (IEEE WCCI 2006)*, Vancouver, BC, Canada, pages: 2231 - 2238, 16-21 July 2006.

[3] C.N. Nyirenda, "Fuzzy Logic Congestion Control for TCP/IP Networks: A Dual Explicit Notification", *Southern African Telecommunications Networks and Applications Conference (SATNAC) 2005 proceedings (CD ROM),* ISBN 0-620-34908-5, Champagne Sports, Drakensberg, South Africa, Sept 2005.

[4] C.N. Nyirenda, D.S. Dawoud, "A Fuzzy Logic Based Dual Explicit Congestion Control Mechanism for Satellite TCP/IP Networks", *Accepted for the Mobile Computing and Wireless Communication International Conference (MCWC 2006)*, Amman, Jordan, September 2006.

[5] C.N. Nyirenda, D.S. Dawoud, "Self-Organization in a Particle Swarm Optimized Fuzzy Logic Congestion Detection Mechanism for IP Networks", Submitted to *Scientia Iranica, International Journal of science and Technology.*

Chapter 2

Literature Survey on post-RED AQM Algorithms

2.1 Introduction

Chapter 1 has discussed the design principles, the strengths and the weaknesses of the Random Early Detection (RED) algorithm, which is an IETF standard Active Queue Management (AQM) algorithm. Chapter 1 also points out that a plethora of AQM algorithms has been proposed either to improve the RED algorithm or to introduce novel concepts for congestion detection. In this Chapter, we present an in-depth review of the major post-RED algorithms both in the traditional and the fuzzy logic domain. Although fuzzy logic algorithms exhibit better performance than traditional AQM algorithms, we think that they can be improved further by incorporating some good operational characteristics from traditional algorithms. This Chapter is organized as follows: In section 2.2, the major traditional AQM algorithms are reviewed. Their efficiencies and deficiencies are highlighted. Section 2.3 begins by presenting the Fuzzy Logic Control Theory and ends with a review of the major Fuzzy Logic AQM algorithms. The efficiencies and deficiencies of these algorithms are highlighted. Finally, section 2.4, presents a summary of this Chapter.

2.2 Major Traditional AQM Algorithms

This section reviews the following traditional AQM algorithms: BLUE [FEN99], CHOKe [PPP00], Adaptive RED [FGS01], REM [ALLY01] and GREEN [WZ02]. These algorithms have been adequately documented in peer-reviewed literature. Except for BLUE, all of these

algorithms have featured in the comparative study of AQM schemes by Bitorika *et al.* [BRH03]. We review them in the chronological order of their publication.

2.2.1 BLUE

BLUE algorithm [FEN99], proposed by Wu-Chang Feng *et al.*, uses packet loss and link under-utilization events, rather than queue size, to adjust the rate of congestion notification. The congestion notification rate p_m is increased at a set rate if the queue size exceeds a threshold L, and it is decreased if the link is idle. The notification rate is increased by d_1, every *freezetime* seconds when the queue size is over the L threshold. The notification rate decreases by d_2 every *freezetime* seconds when the link is idle. In this way, the congestion notification rate p_m converges to a value which controls the arrival rate so that queue is below the threshold L, and the link is not idle. Figure 2.1 shows the BLUE algorithm.

Upon Packet loss or $(B(t) > L)$ event:
if $((t - last\_update) > freezetime)$ then
$p_{m+1} = p_m + d_1$
$last\_update = t$

Upon link idle event:
if $((t - last\_update) > freezetime)$ then
$p_{m+1} = p_m - d_2$
$last\_update = t$

Where t = current time, $B(t)$ = current queue size

Figure 2.1: BLUE AQM

Simulation results in [FEN99] show that BLUE achieves a more stable marking probability than RED. It is also shown that BLUE has a more stable queue size than RED.

Although BLUE exhibits better performance compared to RED, further research [WZ02] has uncovered a number of shortcomings in it. The major problem relates to its design which necessitates that the queue wanders between 0 and the L threshold level in order for the marking

probability p_m to be adjusted to an equilibrium level required for a given load of TCP connections. This is because p_m is not adjusted unless the queue is either 0 or L. Unless the aggregate arrival rate into the link is matched perfectly to the link capacity, then the queue will decrease or increase, and cause adjustments to p_m when the link idle or queue threshold events are generated. It has been pointed out in [WY03] that such fluctuations in queue size result in delay jitter which is detrimental to the Quality of Service (QoS) of interactive applications, such as VoIP.

2.2.2 CHOKe

CHOKe [PPP00], which is short for "CHOose and Keep for responsive flows, CHOose and Kill for unresponsive flows" is a stateless active queue management scheme which aims to control source rates so that an equal sharing of bandwidth is achieved at the CHOKe link. The CHOKe algorithm is interesting because of its performance as well as its simple and elegant implementation.

CHOKe differentially penalizes non-responsive and TCP unfriendly flows by using queue buffer occupancy information of each flow. It calculates the average occupancy of the FIFO buffer using the exponential moving average just as RED [FJ93] does. It also marks two thresholds on the buffer, a minimum threshold *minth* and a maximum threshold *maxth*. If the average queue size is less than *minth*, every arriving packet is queued into the FIFO buffer .If the average queue size is greater than *maxth*, every arriving packet is dropped. If the average queue is between *minth* and *maxth*, each arriving packet is compared with a randomly chosen packet called a *drop candidate packet*, from the FIFO buffer. If they have the same flow ID, they are both dropped. Otherwise the randomly chosen packet is kept in the buffer (in the same position as before) and the arriving packet is dropped with a probability that depends on the average queue size. Results in [PPP00] show that this simple algorithm is able to control high-bandwidth unresponsive UDP flows, so that TCP connections can share the link more equitably. The CHOKe algorithm also helps to protect the networks from network anomalies such as Denial of Service (DoS) attacks and routing loops [HMMD02] which may flood the network.

Although the CHOKe algorithm is good at addressing the problem of fairness, it is easy to see that this algorithm is not good at addressing the other major AQM objectives such as maximization of link utilization and minimization of packet loss rates.

2.2.3 Adaptive RED

Adaptive RED is a modification to RED which addresses the difficulty of setting appropriate RED parameters [FGS01].Adaptive RED adapts the maximum drop probability max_p so that the average queue size is halfway between min_{th} and max_{th}. The value of max_p is kept in the range 1- 50% and is adapted gradually. Adaptive RED adds the increment and the decrement factors to RED. These factors control the increase and the decrease rates of max_p respectively. When the average queue is below the target value, the value of max_p is decreased. When the average queue is above target value, the value of max_p is increased. Adaptive RED includes another modification to RED, called "gentle RED" [FLO00b]. In gentle RED, when the average queue size is between max_{th} and the maximum buffer size BS, the drop probability is varied linearly from max_p to 1, instead of being set to 1 as soon as the average is greater than max_{th}.These modifications to max_p and the drop probability are shown in Figure 2.2. Results in [FGS01] show that ARED removes the sensitivity to parameters that affect RED's performance and can reliably achieve a specified target average queue length in a wide variety of traffic scenarios.

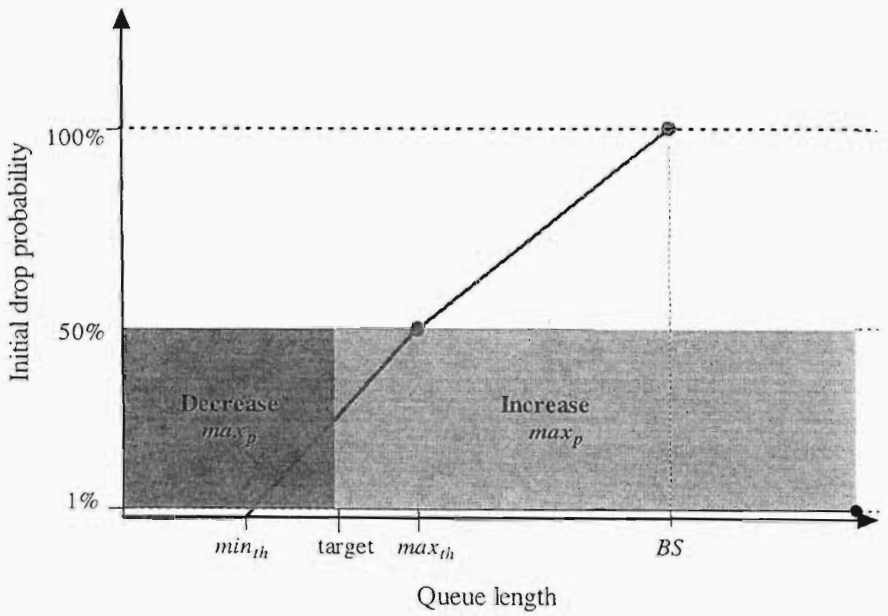


Figure 2.2: Adaptive RED Initial Dropping Probability

2.2.4 REM

Random Explicit Marking (REM) is a framework for communicating congestion information from links to sources by exponential marking. A REM link marks a packet at link l with a probability based on the link price p_l state, and a global encoding constant $\phi(1 < \phi)$:

$$m_l(t) = 1 - \phi^{-p_l} \tag{2.1}$$

Assuming links mark packets independently, the overall probability of a packet being marked has an exponent with the sum of the link prices:

$$1 - \prod_{l \in L} (1 - m_l(t)) = 1 - \phi^{-(p1+p2+p3+\ldots)} \tag{2.2}$$

Because sources know the value of ϕ, they can compute the total end-to-end path congestion price. Therefore, in a complete deployment, REM requires a REM link algorithm and a source algorithm capable of decoding REM information. In the present Internet implementation link REM AQM algorithm deployed with the TCP source algorithm. In this case, the price $p_l(t)$ state variable can be interpreted as the marking rate, just as the other AQMs discussed. Three different alternative pricing algorithms PC1-PC3 constitute REM.

$$\text{PC1: } p_l(t+1) = \gamma b_l(t) \tag{2.3}$$

$$\text{PC2: } p_l(t+1) = [p_l(t) - \gamma(x^l(t) - c_l)]^+ \tag{2.4}$$

$$\text{PC3: } p_l(t+1) = [p_l(t) - \gamma(\alpha_l b_l(t) + x^l(t) - c_l)]^+ \tag{2.5}$$

where $p_l(t)$ is the congestion notification rate, c_l is a target capacity just below the actual link capacity, $b_l(t)$ is the backlog, and γ and α are control gain constants which affect speed and stability of control.

PC1 control law is very similar to RED because the congestion notification rate is proportional to backlog. PC2 and PC3 measure the arrival rate to the link to compute the congestion notification rate instead of using the backlog. The congestion notification rate is controlled by an integral controller, whose error term is the discrepancy between the aggregate arrival rate to the link and the target link capacity. The difference between PC2 and PC3 is that PC3 adds a backlog penalty term to the control process, which makes the marking rate increase with greater rate if there is a backlog. This improves the transient response of the basic PC2 controller, and reduces the amount of backlog during transient periods when the load changes.

Although REM works very well in a steady state situation, experimental results in [BZ02] show that its behaviour in transient conditions and with realistically constrained buffer sizes is not necessarily optimal.

2.2.5 GREEN

The GREEN algorithm is a feedback control function which adjusts the rate of congestion notification in response to the flow based congestion measure x_{est} the estimated data arrival rate. GREEN is based on a threshold function. If the link's estimated data arrival rate x_{est} is above the target link capacity c_1, the rate of congestion notification P is incremented by ΔP at a rate of $1/\Delta T$ .Conversely, if x_{est} is below c_1, P is decremented by ΔP at a rate of $1/\Delta T$. The algorithm applies probabilistic marking of incoming packets at the rate P, either by dropping packets, or setting the ECN. Let the step function $U(x)$ is defined by:

$$U(x) = \begin{cases} +1 & x \geq 0 \\ -1 & x < 0 \end{cases} \tag{2.6}$$

therefore

$$P = P + \Delta P.U(x_{est} - c_t). \tag{2.7}$$

The target link capacity c_t is assigned a value just below the actual link capacity c, typically $0.97c$, so that the queue size converges to 0. Incoming data rate estimation is performed using exponential averaging:

$$x_{est} = (1 - \exp(-Del/K)) * (B/Del) + \exp(-Del/K) * x_{est} \tag{2.8}$$

where Del is the inter-packet delay, B the packet size and K the time constant. There is a relationship between REM PC3 and GREEN. If equation (2.1) is linearised, $m = P$, the exponential marking is eliminated. Furthermore if the buffer term $\alpha = 0$ and the linear constant γ is replaced with the step function (2.6), GREEN's congestion notification rate P becomes equivalent to REM's price p_l.

Results in [WY03] show that GREEN exhibits higher link utilization, lower packet rate and delay than RED and Drop-tail algorithms. Its overall performance is, however very similar to that of REM because their control laws are similar.

Although GREEN comes with all these advantages relative to its predecessors, a comparative evaluation of AQM algorithms in [BRH03] uncovers one major fundamental problem. Results in [BRH03] show that GREEN does not keep track of the queue length because its only congestion metric is traffic load. Thus it can reach steady-state when the queue is full while keeping the incoming traffic rate close to the target. In this situation its behaviour is just the same as that of the Drop-tail mechanism.

2.2.6 WRED

Weighted RED (WRED) is Cisco's proprietary algorithm which belongs to the family of multi-RED algorithms which have been proposed for congestion control in Differentiated Services (DiffServ) IP Networks [BLA98]. Other algorithms in this family include RIO-C and RIO-DC [ML00].Differentiated Services (DiffServ) is a paradigm that has been proposed for QoS provisioning on the Internet. Research in DiffServ IP Networks was triggered by the desire to satisfy QoS requirements for different applications as the Internet evolves. For instance, applications such as World Wide Web (WWW) and file transfers prefer low data loss rates while tolerating large delays. On the other hand, multimedia applications (Voice over IP, Video-on-Demand etc.) require low delays but can tolerate a certain amount of loss rates. The first paradigm that was proposed to meet these demands is known as the Integrated Services (IntServ) [WHI97], [GKP98], [SSZ98]. IntServ ensures end-to-end and per-flow QoS. All connections reserve the resources needed and routers maintain their reservation parameters on a per-flow basis. The main problem with this approach relates to the issue of scalability. As the number of users or connections increases, it becomes difficult to maintain reservation parameters on a per-flow basis. In order to reduce the problems of IntServ, DiffServ was introduced to provide QoS with aggregation of flow and per-class service.

DiffServ classifies packets into different service classes by setting a 6-bit pattern in the IP header, called the Differentiated Services Code Point (DSCP). Each of the 64 possible classes is associated with a particular forwarding mechanism at a node, called a Per-hop Behaviour (PHB). The PHB determines the relative QoS of each class. The classes are called Behaviour Aggregates (BA). The IETF DiffServ standards describe qualitatively the behaviour expected for each BA but do not regulate the specific forwarding mechanisms required at the routers and switches to achieve these behaviours. The original DiffServ proposal [BLA98] is composed of two broad classes: the expedited forwarding (EF) class and the assured forwarding (AF) class. The

Expedited Forwarding (EF) class is the highest priority class and is for applications requiring low-loss, low-latency and low-jitter. Typical applications for the EF class include voice over IP (VoIP), interactive games and online trading programs. The Assured Forwarding AF class has a lower priority than EF. AF is comprised of a number of subclasses with different grades of service priority. The AF class is intended for best-effort applications such as FTP, and WWW. Each subclass within AF is identified by the notation AFxy, where x specifies the service class and y the packet drop precedence. The AF class is subdivided into four service classes, implemented as Premium (AF1y), Gold (AF2y) Silver (AF3y) and Bronze (AF4y). The IETF specifies [WH99] that each of these subclasses should receive a guaranteed minimum share of link capacity. Typically, the bandwidth available B(AFxy) to a class x in a bottleneck link is such that B (AF1y) > B(AF2y) > B(AF3y) > B(AF4y). Within each subclass, one of three levels of the packet drop precedence (y) is specified. The packet drop probability dP(AFxy) of each level should be dP (AFx1) <= dP(Afx2) <= dP(Afx3).

In order to ensure that QoS requirements for different traffic classes are met, the DiffServ environment employs a distinct scheduling algorithm in the deque routine. This is in contrast to best effort networks where the simple FIFO scheduling mechanism is used. Examples of DiffServ scheduling disciplines include: Round Robin (RR), Priority Scheduling (PS) and Weighted Fair Queuing (WFQ). A typical DiffServ implementation uses a priority scheduler to give the EF class packets absolute priority over AF class packets. The bandwidth available to the AF class is typically shared between each AF subclass by using a WFQ scheduler. The WFQ scheduler ensures that each service class received a pre-configured portion of bandwidth during overload. The WRED AQM scheme is implemented on each of the four AF queues while the EF queue is controlled by either RED or a Drop-tail approach.

WRED computes a single average queue size which includes packets from all drop precedences. By convention, packets of the three drop precedence within an AF class are delimited by the colors red, green and yellow, such that the packet drop probability of each color is; Pr(red) > Pr(yellow) > Pr(green). WRED gives each color a separate packet dropping probability as shown in Figure 2.3.

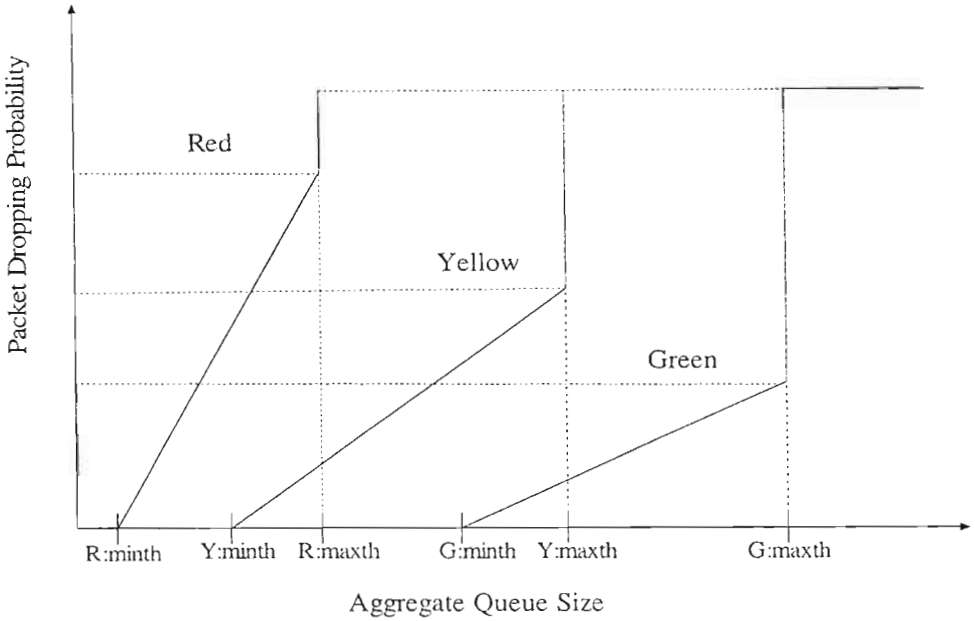


Figure 2.3: WRED Mechanism

Although WRED has been deployed in commercial routers, it generally remains disabled [BRH03] because of the problems associated with RED.

2.3 Fuzzy Logic Based AQM Schemes

We present the Fuzzy Logic Control Theory before analyzing Fuzzy Logic AQM schemes. This theory is the backbone for all the Fuzzy Logic AQM schemes such that a proper understanding of this theory, before delving into the specifics of different AQM schemes that are emanating from it, is very important.

2.3.1 Fuzzy Logic Control Theory

We first of all present an overview on fuzzy sets and operators. The principles of operation of the Fuzzy Logic Controller (FLC) are presented in the second part of this section.

2.3.1.1 Fuzzy Sets and Operators

Fuzzy logic is a concept that brings together the reasoning used by computers and the reasoning used by people. The concept of fuzzy logic was first presented by Zadeh [ZAD65], known as the father of fuzzy theory. In the conventional (crisp) sets, members are always fully categorized and there is no ambiguity or dichotomy about membership. Zadeh contends that human thinking does not embrace precise definitions, but classes of definitions known as *fuzzy sets* in which the transition from membership to non-membership is gradual rather than abrupt. The degree of membership is specified by a number between 0, non-membership, and 1, full membership.

The *fuzzy set A* in *X* is characterized by a membership function $\mu_A(x)$, which associates each element in X with a real number in the interval [0, 1]. $\mu_A(x)$ is known as the grade of membership. Hence the fuzzy set on the universe of discourse X is defined as:

$$A = \{(x, \mu_A(x)) \mid x \in X\} \tag{2.9}$$

The *fuzzy set* has three principal features as shown in Figure 2.4:

1. The range of values (domain) called the *Universe of Discourse* over which the fuzzy set is valid (the x-axis).
2. The degree of membership (μ) axis (the y-axis)
3. The fuzzy set function which maps the domain to the degree of membership.

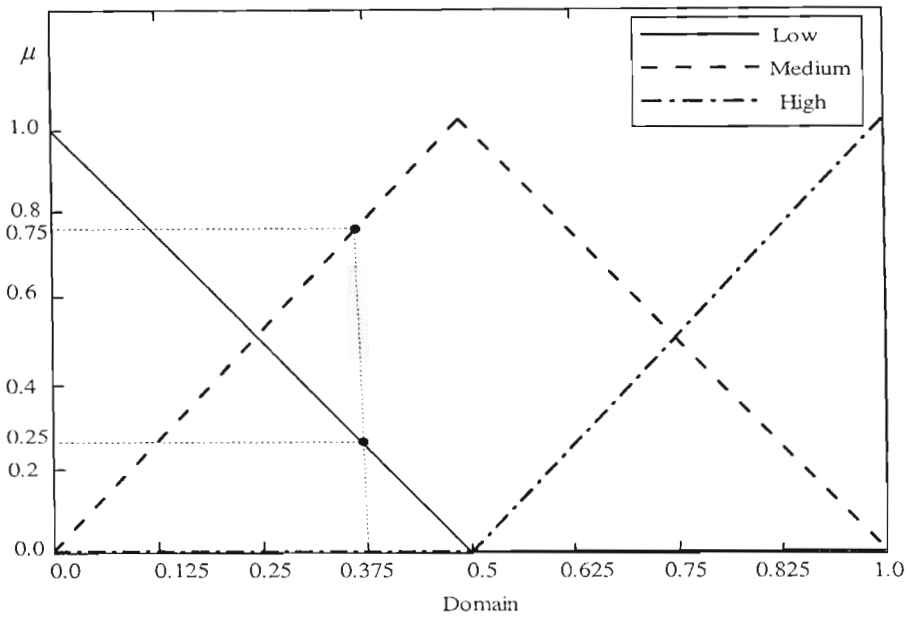


Figure 2.4: Fuzzy Membership Function

The membership functions in Fig 2.4 are *triangular*. They are overlapping each other by at least 50%. As a result, a discrete value on the *universe of discourse* (domain) can be a member of two or more fuzzy sets. For example, the discrete (crisp) value of 0.375 has a membership grade or μ of 0.25 in the fuzzy set *LOW* , a membership grade or μ of 0.75 in the fuzzy set *MEDIUM* and a membership grade of 0.0 in the fuzzy set *HIGH*. Besides triangular membership functions, other types of membership functions include *trapezoidal*, *gaussian*, *bell*, *sigmoid* and *asymmetric*.

Like *conventional sets*, there are specifically defined operations for combining and modifying fuzzy sets. Since fuzzy sets are not crisply partitioned in the same sense as Boolean sets, these operations are applied at the truth membership level. These set theoretic functions provide the fundamental tools of the logic. Following the conventional fuzzy logic operations initially defined by Zadeh, the basic operations are

$$\text{Intersection: } A \cap B = min(\mu_A(x), \mu_B(x)) = \mu_A(x) \wedge \mu_B(x) \tag{2.10}$$

$$\text{Union: } A \cup B = max(\mu_A(x), \mu_B(x)) = \mu_A(x) \vee \mu_B(x) \tag{2.11}$$

$$\text{Complement: } \sim A = 1 - \mu_A(x) \tag{2.12}$$

$$\text{Subset: } A \subseteq B = \mu_A(x) \leq \mu_B(x) \tag{2.13}$$

The *intersection* in (2.10) and the *union* in (2.11) are the most frequently used examples of the *T-norm (Triangular norm)* and the *T-conorm operators* respectively.

The T-norm operator, $T:[0,1]\times[0,1]\rightarrow[0,1]$ gives a general specification of the intersection of two fuzzy sets A and B. It aggregates two membership grades as follows

$$\mu_A(x)\cap\mu_B(x)=T(\mu_A(x),\mu_B(x))=\mu_A(x)\otimes\mu_B(x) \tag{2.14}$$

where $\otimes$ is the T-norm operator. This two-placed function meets the following basic requirements:

$$\text{Boundary: } T(0,0)=0, T(a,1)=T(1,a)=a \tag{2.15}$$

$$\text{Monotonicity: } T(a,b)\le T(c,d) \text{ if } a\le c \text{ and } b\le d \tag{2.16}$$

$$\text{Commutativity: } T(a,b)=T(b,a) \tag{2.17}$$

$$\text{Associativity: } T(a,T(b,c))=T(T(a,b),c) \tag{2.18}$$

The T-conorm operator, $S:[0,1]\times[0,1]\rightarrow[0,1]$ gives a general specification of the union of two fuzzy sets A and B. It aggregates two membership grades as follows

$$\mu_A(x)\cup\mu_B(x)=S(\mu_A(x),\mu_B(x))=\mu_A(x)\oplus\mu_B(x) \tag{2.19}$$

where $\oplus$ is the T-conorm operator. This two-placed function meets the following basic requirements:

$$\text{Boundary: } S(1,1)=1, S(0,a)=S(a,0)=a \tag{2.20}$$

$$\text{Monotonicity: } S(a,b)\le S(c,d) \text{ if } a\le c \text{ and } b\le d \tag{2.21}$$

$$\text{Commutativity: } S(a,b)=S(b,a) \tag{2.22}$$

$$\text{Associativity: } S(a,S(b,c))=S(S(a,b),c) \tag{2.23}$$

2.3.1.2 Fuzzy Logic Control: Principles of Operation

A fuzzy logic controller is an approximate reasoning-based controller, which does not require exact analytical models and is much closer in spirit to human thinking and natural language than a traditional logic system. Fuzzy rules are the backbone of a fuzzy logic system. A simple fuzzy rule can be written as

$$\textit{if } x \textit{ is HIGH then } y \textit{ is POSITIVE BIG} \tag{2.24}$$

where *HIGH* and *POSITIVE BIG* are linguistic values defined by fuzzy sets on the *Universes of Discourse X* and *Y* respectively. The if-part, *x is HIGH* ,is known as *the antecedent* and the then-part, *y is POSITIVE BIG* , is known as *the consequent*. A set of linguistic rules used to map fuzzy inputs to outputs is known as a *rule base*. Apart from the *rule base*, other parts of a Fuzzy Logic System include *the Fuzzifier*, *the Inference Engine* and *the Defuzzifier* as shown in Figure 2.5.

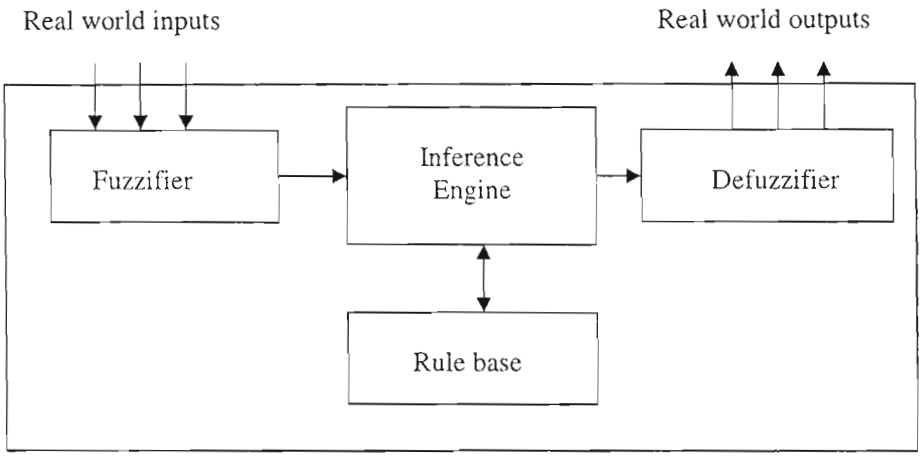


Figure 2.5: Fuzzy Logic Controller

The Fuzzifier translates the real world input variables into fuzzy representation by calculating suitable sets of degree of membership for each of the inputs. The Inference Engine evaluates output fuzzy sets from input sets using the predefined fuzzy rules contained in the rule base. The Defuzzifier transforms the output fuzzy sets into real world output variables. The Inference Engine calculates *the degree of activation* of every rule in the rule base. If the antecedent for rule j contains one variable, the rule's degree of activation is equal to the degree of membership of that single variable. If $\mu_j^1(x_1)$ denotes the degree of membership of input x_1 for rule j then $\mu(r_j)$ the degree of activation of rule j is expressed as follows

$$\mu(r_j) = \mu_j^1(x_1) \tag{2.25}$$

If the antecedent for rule j contains more than one variable in the form

$$\text{rule } j: \text{IF } A_j^1 \text{ AND } A_j^2 \text{ AND ... AND } A_j^n \text{ THEN } b_j \tag{2.26}$$

where A_j^k is a fuzzy set with membership function $\mu_j^k : \Re \rightarrow [0,1], j = 1,...,m, k = 1,...,n, b_j \in \Re$. In this case, the degree of activation for rule j, $\mu(r_j)$ is determined using the *t-norm operator* as follows

$$\mu(r_j) = \mu_j^1(x_1) \otimes \mu_j^2(x_2) \otimes ... \otimes \mu_j^n(x_k) \tag{2.27}$$

Therefore at the output of the Inference Engine there will always be a fuzzy set $\mu(r_j)$ for $j = 1,2,...,m$. This fuzzy set is composed of the fuzzy sets output by each of the rules using equation (2.27). In order to be used in the real world, the fuzzy output needs to be interfaced to the crisp domain by using a defuzzifier. There are several defuzzification methods but the widely applied one is the Centre of Gravity (COG) technique, which computes the weighted-average of the centre of gravity of each membership function. The COG of the system with m rules is as follows

$$y(x) = \frac{\sum_{j=1}^{m} b_j \mu(r_j)}{\sum_{j=1}^{m} \mu(r_j)} \tag{2.28}$$

where b_j is the centre of the membership function recommended by the consequent of rule j.

2.3.2 Evaluation of Fuzzy Logic AQM Schemes

This subsection presents an analytical evaluation of the key Fuzzy Logic AQM schemes that are available in the public domain.

2.3.2.1 The Algorithm of Ren *et. al.*

This algorithm is arguably the first fuzzy logic AQM algorithm to be published. It is inspired by the PI congestion control mechanism developed by Holot *et al.* [HMTG01]. Holot *et al.* approximated the non-linear and dynamic TCP/AQM model, proposed by Misra *et al.* [MGT00], as a linear constant system by small-signal linearization about an operating point, and then designed the traditional PI controller using the classical control theory. The PI proves useful and helpful in the analysis and explanation of the instability of RED under some network parameter configuration. In spite of this advantage, Ren *et al.* [FYX02] argue that the PI controller is prone

to instability and poor performance because the real network is full of dynamic parameters such as number of active connections.

In their design of a fuzzy based congestion controller, Ren *et. al.* [FYX02] define EQ as the expected value of queue length. They also define VQ as the estimated value of the maximum range that the queue length variation can reach during one sampling interval. They fix the sampling frequency at 160Hz.They also fix VQ at 0.5BS (where BS is the buffer size).They use two input linguistic variables: error of queue length e and the error's varying rate Δe.The latter effectively describes the local dynamic of the difference between the arrival rate and the service rate. The chosen output is linguistic variable which represents the increment of the packet marking or dropping probability. The model of fuzzy system, comprising the control rules and the term sets of the variables with their related fuzzy sets is obtained through a tuning process that starts from a set of the initial insight considerations. This process continues until the system reaches a level of performance considered to be adequate. The membership functions are as shown in Figure 2.6.

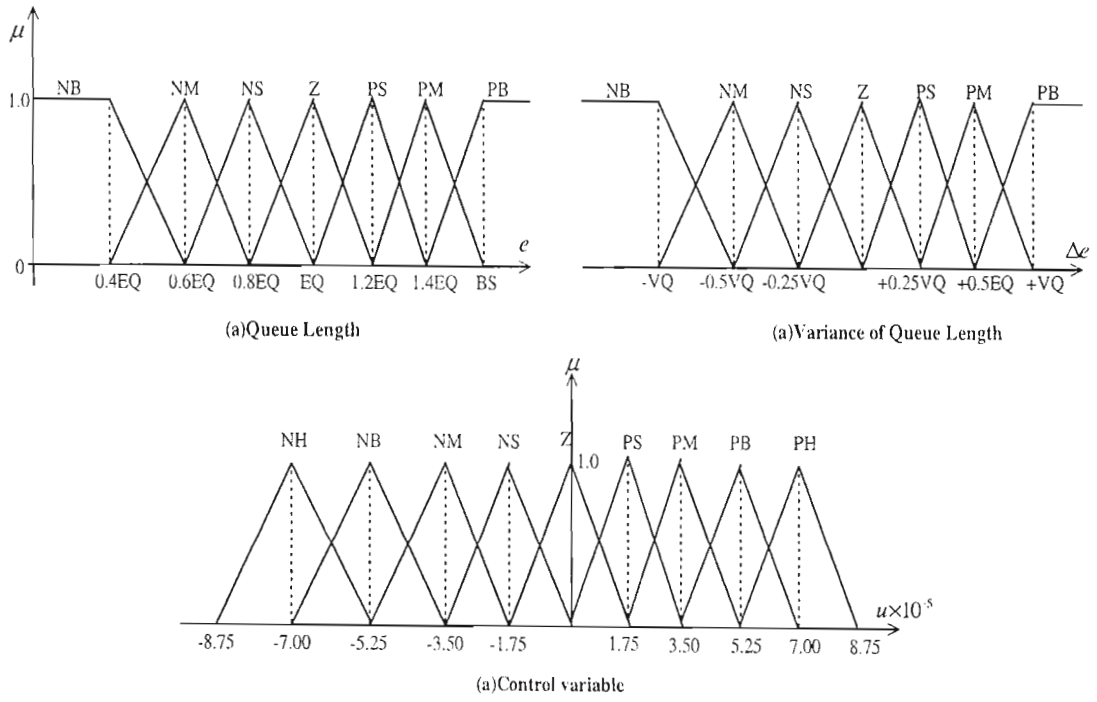


Figure 2.6: Membership Functions for Ren's Algorithm

Both of the input variables have seven fuzzy term sets, which are negative big (NB), negative medium (NM), negative small (NS), zero (Z), positive small (PS), positive medium (PM) and

positive big (PB). The output variable is greater by two term sets in number; the extra sets are negative huge (NH) and positive huge (PH).The control rule base is as follows:

Table 2.1: Fuzzy Control Rules for Ren's Algorithm

e	Δe						
	NB	**NM**	**NS**	**Z**	**PS**	**PM**	**PB**
NB	NH	NH	NH	NB	NB	NM	NM
NM	NH	NB	NB	NM	NM	NS	NS
NS	NB	NM	NS	Z	Z	Z	PS
Z	NM	NS	Z	Z	Z	PS	PM
PS	NS	Z	Z	Z	PS	PM	PB
PM	PS	PS	PM	PM	PB	PB	PB
PB	PM	PM	PB	PB	PB	PH	PH

The fuzzy sets in the matrix denote the output control variable under different input conditions. For example, if queue error e is *Negative Big (NB)* and error varying rate Δe is *Negative Small (NS)*, then the *control variable* is *Negative High (NH)*.

This algorithm was implemented on the NS-2 platform and had its performance compared with the PI controller under various scenarios. Simulation results show that it has superior steady and transient state performance, exhibits great adaptability to variances in link delay and capacity, and provides more robustness against noise and disturbance. However, its major weakness lies in the fact that its control rules and membership functions are obtained through a manual tuning process which is based on the designer's insight. The human factor involved in this operation makes it difficult for these algorithms to achieve optimum performance for all the key AQM objectives.

2.3.2.2 The Algorithm of Chrysostomou *et. al.*

Chrysostomou *et al.* [CHR03a] use a similar approach to Ren *et al.* [FYX02]. The fuzzy control system is designed to regulate the queues of IP routers by achieving a specified desired Target Queue Length (TQL) q_{des} in order to maintain both high utilization and low mean delay. All the quantities are considered at the discrete instant kT, with T the sampling period. Two input variables are used: (1) $e(kT) = q_{des} - q$, the error on controlled variable q at each sampling period

and (2) $e(kT-T)$, the error of the queue length at the previous sampling period. The output of the fuzzy control system is the mark probability $p(kT)$. The two input linguistic variables have the same membership function with fuzzy term sets which are negative very big (NVB), negative big (NB), negative small (NS), zero (Z), positive small (PS), positive small(PM) and positive very big (PVB).This approach is just the same as in [FYX02]. The output variable, the mark probability has a membership function with fuzzy term sets which are zero (Z), tiny (T), very small (VS), small (S), big (B), very big (VB) and huge (H). Just like in [FYX02], the design of the rule base is also achieved through a tuning process that starts from a set of the initial insight considerations and progressively modifying the parameters of the system until it reached a level of performance considered to be adequate. A certain level of intuition and experience is used to design the rule base.

This FLC is implemented in the NS-2 platform and has its performance compared against PI, Adaptive RED and REM under various scenarios. This controller is shown to exhibit to exhibit many desirable properties, like robustness and fast system response, and behaves better than other AQM schemes (PI, ARED, REM) in terms of queue fluctuations and delay, packet losses, and link utilization. However, Chrysostomou *et al.* [CHR03a] were quick to point out the need for future work to include the design of a fuzzy model reference learning controller, which can tune the parameters of the fuzzy logic learning controller on line, using measurements from the system, to obtain a better performance. They also point out the need to investigate the implementation of this fuzzy logic congestion controller in a differentiated service environment in TCP/IP networks, using separate linguistic rules for each predefined class of service.

Just like the proposal in [FYX02], its major weakness lies in the fact that its control rules and membership functions are obtained through a manual tuning process which is based on the designer's insight. The human factor involved in this operation makes it difficult for these algorithms to achieve optimum performance for all the key AQM objectives.

2.3.2.3 Fuzzy BLUE Controller

Fuzzy Blue Controller (FBC), proposed by Yaghmaee and Toosi [YT03], is an extension to the traditional BLUE mechanism [FEN99]. FBC is a two-input-single-output fuzzy logic controller. The input linguistic variables are packet loss and the normalized queue length. The output

linguistic variable is the drop probability p_m. The term set of linguistic variables packet loss and normalized queue length are defined as follows:

- T(packet loss) ={ small, med (medium), big}
- T(normalized queue length) ={low, mid (middle), high }

The output term set of fuzzy logic controller is also defined as follows:

- T(Pm)={zero, low, moderate, high}

The design of rule base is based on experience and beliefs on how the system should work. Table 2.2 presents the fuzzy linguistic rules used in the simulation. The tuning (trial and error) approach is used along with the theory approach in order to design the rule base. In the theory approach, rules are designed in such a way that specific functionality of a parameter (such as throughput) is guaranteed.

The performance of the Fuzzy BLUE Controller was compared with that of the traditional BLUE mechanism. A number of different trials were performed to test the correctness of the algorithm. Based on simulation results, it was shown that the Fuzzy BLUE could achieve near 100% throughput [YT03]. It was also shown that the Fuzzy BLUE mechanism has better loss performance and queue length behavior than traditional BLUE mechanism.

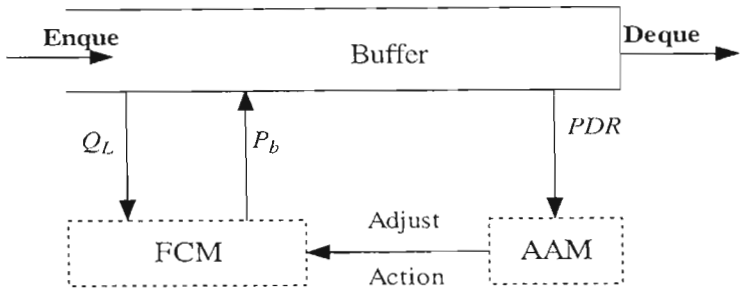
Table 2.2: Linguistic rules for Fuzzy Blue Controller

/* Linguistic rules of FBC */
if packet loss is *small* and normalized queue length is low then pm is zero;
if packet loss is *small* and normalized queue length is med then pm is zero;
if packet loss is *small* and normalized queue length is high then pm is zero;
if packet loss is *med* and normalized queue length is low then pm is zero;
if packet loss is *med* and normalized queue length is med then pm is zero;
if packet loss is *med* and normalized queue length is high then pm is moderate;
if packet loss is *big* and normalized queue length is low then pm is zero;
if packet loss is *big* and normalized queue length is med then pm is low;
if packet loss is *big* and normalized queue length is high then pm is high;

Just like the proposals in [FYX02], [CHR03a], its major weakness lies in the fact that its control rules and membership functions are obtained through a manual tuning process.

2.3.2.4 Adaptive Fuzzy RED

To our knowledge, Adaptive Fuzzy RED (AFRED) [WAN03] is the first algorithm that employs an online adaptation mechanism. This algorithm uses the instantaneous queue length as the only input variable to determine the packet marking or dropping probability. It employs an Adaptive Adjust Module (AAM) which is triggered periodically to calculate the real packet drop rate and output adjust conditions to the Fuzzy Controller Module (FCM). Figure 2.7 shows AFRED's architecture.



Q_L: *Queue Length* *PDR: Packet Drop Ratio*

P_b: *Packet Marking/Drop Probability*

Figure 2.7: AFRED Architecture

The novel principle exhibited by AFRED is that the real packet drop ratio (pdr) can show the congestion degree coarsely at least since heavy (or light) congestion will trigger lots of (or few) packet drops. Although this proposal introduces the concept of online adaptation, it falls short in two areas. Firstly, it uses the instantaneous queue length as a sole input variable. As explained in [FKS99], [ALLY01], [WZ02], queue size is not a good indicator of the severity of congestion, and the level of congestion notifications issued may be too great and bursty, leading to excessive packet loss. Secondly, it uses only packet loss in the adjust process. Other important performance metrics such as link utilization, fairness, delay and jitter are not considered.

2.3.2.5 Fast Adaptive Fuzzy Controller

In this algorithm, the authors use Lyapunov's Direct Method for stability analysis [PY98] based on the mathematical model for the internet which was developed by Frank Kelly [KEL01] and generalized to support multiple TCP sessions by Crowcroft and Oeschslin [CO].They also incorporate the classical Proportional Integral Derivative (PID) controller for online adaptation.

This algorithm exhibits better queue stability and lower packet loss rates compared to RED and the Proportional Integral Derivative (PID) AQM algorithm [YC03]. However, the internet mathematical model [KEL01], [CO] used in this algorithm is based on the principle that the Internet is predominantly TCP. It neglects the effect of non-responsive flows (such as UDP) and network anomalies such as Denial of Service attacks and rooting loops [HMMD02]. A recent study of internet traffic [FKM03] shows that UDP accounts for (22±11) % packet composition of internet traffic while TCP accounts for (75±12) %. Therefore, ignoring the UDP component compromises the composition of traffic on the Internet. The other problem relates to the fact that this algorithm puts more emphasis on queue stability without incorporating other important performance metrics such as link utilization, packet loss rates and fairness.

2.3.2.6 Fuzzy RED for DiffServ

Fuzzy RED [CHR03b] is the DiffServ implementation of best-effort Fuzzy AQM in [CHR03a]. This algorithm removes the fixed maximum and minimum queue thresholds from the RED queue for each class, and replace them with dynamic network state dependant thresholds calculated using a fuzzy inference engine (FIE).Two sets of linguistic rules are used in order to generate two *mark* probabilities for high and low priority traffic. The *mark* probability behavior based on two network-queue state inputs: the instantaneous queue size and the queue rate of change.NS-2 simulation results show that Fuzzy-RED behaves well and delivers almost identical throughput results under various conditions without any retuning or parameterization. Fuzzy-RED also performs equally well using homogeneous or heterogeneous traffic sources (in this case TCP/FTP traffic and TCP/Web-like traffic) without any change in the way it is defined or needing any special tuning. Chrysostomou *et al.* [CHR03b] point out that future work can include further refinement of the rule base, self-tuning, or different fuzzy based control strategies for the design of the fuzzy rule base and its tuning.

Fuzzy RED has two major shortcomings. The first one is that it inherits the tuning problems from its forerunner in [CHR03a]. The second one relates to the DiffServ queuing architecture for which it is tailored. A closer look at the Fuzzy RED algorithm shows that it implements only one queue and performs service differentiation by using two different marking schemes. The issue of scheduling is not explicitly mentioned such that it can be assumed that default FIFO scheduling is used. Recent developments in DiffServ research show that architectural trends are shifting towards Proportional Differentiated Services (PropDiffServ) [DSR02] model where each traffic

class is assigned its own queue. In other words, PropDiffServ model removes the concept of drop precedences within a DiffServ class. This reduces the implementation difficulties associated with the original DiffServ framework [BLA98].The PropDiffServ ensures the quality spacing between classes of traffic to be proportional to certain pre-specified class differentiation parameters. An AQM scheme is implemented on each queue (class). The Weighted Fair Queue (WFQ) [LTC00] scheduling algorithm determines the allocation of bandwidth in each class as well as packet transmission order based on pre-specified class differentiation parameters.

2.4 Chapter Summary

In this Chapter, we have presented operational characteristics, the efficiencies and deficiencies of the major traditional AQM schemes. We have also presented the Fuzzy logic Control theory which is the backbone of all fuzzy logic based AQM schemes. We have evaluated the operational characteristics, efficiencies and deficiencies of the major fuzzy logic based AQM schemes in literature. We have uncovered a number of deficiencies in these schemes. Firstly, we have found out that their control rules and membership functions are obtained through a manual tuning process which is based on the designer's insight. The human factor involved in this operation makes it difficult for these algorithms to achieve optimal performance for all the key AQM objectives. Secondly, these algorithms are generally designed with an assumption that the Internet is predominantly composed of TCP traffic, whose sources respond to congestion notification signals from routers by reducing their sending rates. Practically, the situation is not like that because apart from the non-responsive UDP traffic which accounts for (22±11)% of Internet traffic [FKM03], the Internet is nowadays facing a growing list of non-responsive flows and anomalies such as Denial of Service (DoS) attacks and routing loops [HMMD02]. These flows do not reduce their sending rates in times of congestion as responsive TCP flows reduce their rates. Therefore, fairness in these schemes would diminish exponentially as the number of non-responsive flows increases. Chapter 3 presents a new fuzzy logic AQM scheme which addresses these deficiencies by employing multiobjective evolutionary optimization. CHOKe, a traditional AQM scheme is incorporated in the new approach in order to address the fairness issue.

Chapter 3

Fuzzy Logic Congestion Detection using MOPSO

3.1 Introduction

In this chapter, we propose a Fuzzy Logic Congestion Detection (FLCD) algorithm which combines the good characteristics of both the traditional AQM approaches and the fuzzy logic based AQM algorithms. We also introduce new concepts in order to address some of the problems observed in Chapter 2. The membership functions (MFs) of the FLCD algorithm are designed by using a Multi-objective Particle Swarm Optimization (MOPSO) algorithm in order to achieve optimal performance on all the major performance metrics of IP congestion control. The FLCD algorithm is implemented on both the best effort and the Proportional Differentiated Services (PropDiffServ) IP networks .In the best effort implementation, the performance of the FLCD algorithm is compared with that of the fuzzy logic based congestion control algorithm in [FYX03] and the Random Explicit Marking (REM) [ALLY01] algorithm. In the PropDiffServ environment, the performance of the FLCD algorithm is compared with that of Cisco's WRED [CISCO02], which has been deployed in commercial routers. This Chapter is organized as follows: In section 3.2, the FLCD algorithm is presented. Section 3.3 presents the MOPSO theory and the formulation of the IP congestion problem. Section 3.4 implements the MOPSO scheme, generates optimization results and draws the best compromise solution which is used in the configuration of the practical FLCD algorithm. In Section 3.5, we present simulation results and comparative performance analysis of the FLCD algorithm in best effort IP networks. In Section 3.6, the FLCD algorithm is implemented in PropDiffServ IP networks. Section 3.6 also presents

simulation results and a comparative performance analysis of the FLCD algorithm in PropDiffServ IP networks. Finally section 3.7 gives the summary of this chapter.

3.2 Fuzzy Logic Congestion Detection Algorithm

The FLCD algorithm is composed of the Fuzzy Logic Controller (FLC), the Probability Adjuster (PA) and the CHOKe Activator (CA). Figure 3.1 shows the proposed FLCD architecture.

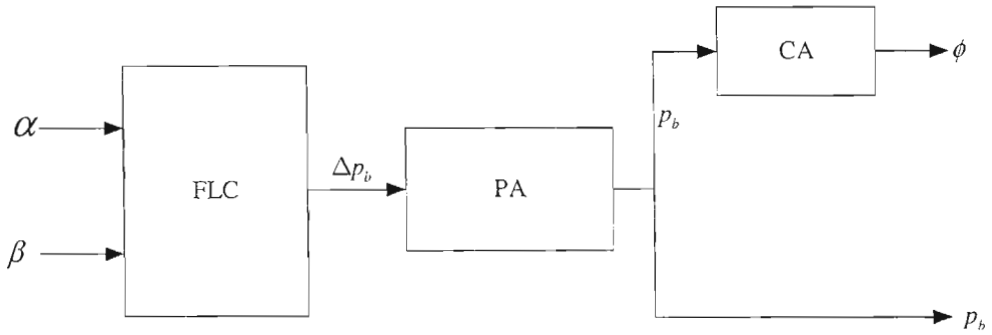


Figure 3.1: Fuzzy Logic Congestion Detection Architecture

A single FIFO buffer in which all packets are treated equally is assumed. The queue status is sampled at a period τ of 0.002 seconds just as in [ALLY01] in order to obtain the queue-occupation size (backlog) $q(t)$ and the traffic arrival rate $r(t)$. The backlog $q(t)$ is translated into the backlog factor α which is the ratio of backlog with respect to the Buffer Size BS :

$$\alpha = q(t)/BS \tag{3.1}$$

In contrast to the proposals in [FYX02], [CHR03a], [CHR03b], which use the variation of queue length in order to determine the packet arrival rate, the FLCD algorithm determines the packet arrival rate by counting the actual number of packets that arrive at the buffer (both those that are queued and those that are dropped) during sampling period τ. When the buffer is prevalently full, the variation of queue length is very small such that it fails to capture the packet arrival rate because most packets are dropped before they get queued. We let n denote the number of packets that arrive at the buffer during period τ. We also let ω_1 denote the measuring weight and r_m

denote the maximum packet arrival rate. The weighted average packet arrival rate $\overline{r(t)}$ and the packet arrival factor β are determined as follows

$$\overline{r(t)} = \omega_1 * \overline{r(t-\tau)} + (1-\omega_1) * n \tag{3.2}$$

$$\beta = \begin{cases} \overline{r(t)} / r_m & \overline{r(t)} < r_m \\ 1.0 & \overline{r(t)} \geq r_m \end{cases} \tag{3.3}$$

The FLC Unit determines the change in packet marking/dropping probability Δp_b by using the fuzzified values of parameters α and β. The set of linguistic rules that govern the inference process in the FLC is shown in Table 3.1.

Table 3.1: Rule Base for the FLC Unit

If α is LOW and β is LOW ***then*** Δp_b is Negative Big.
if α is LOW and β is MEDIUM ***then*** Δp_b is Negative Small.
if α is LOW and β is HIGH ***then*** Δp_b is Zero.
if α is NORMAL and β is LOW ***then*** Δp_b is Negative Small.
if α is NORMAL and β is MEDIUM ***then*** Δp_b is Zero.
if α is NORMAL and β is HIGH ***then*** Δp_b is Positive Small.
if α is HIGH and β is LOW ***then*** Δp_b is Positive Small.
if α is HIGH and β is MEDIUM ***then*** Δp_b is Positive Big.
if α is HIGH and β is HIGH ***then*** Δp_b is Positive Big.

The PA computes the new packet market probability $p_b(t)$ as follows

$$p_b(t) = p_b(t-\tau) + \Delta p_b(t) \tag{3.4}$$

Packets are either marked (if ECN or BECN is enabled) or dropped with the probability $p_b(t)$ in order to inform the sending sources that there is congestion in the network. The sources respond by reducing their sending rates according to TCP's congestion avoidance mechanisms.

In order to address the issue of fairness in light of non-responsive flows and network anomalies such as Denial of Service (DoS) attacks and routing loops [HMMD02] which may dramatically flood the network as the responsive flows back off, we incorporate the CHOKe Activator (CA)

which uses $p_b(t)$ to generate a fuzzy parameter $\phi \in [0,1]$. Let P_{thresh} denote the CHOKe threshold then the fuzzy parameter ϕ is derived as follows

$$\phi = \begin{cases} 0 & p_{thresh} > p_b \\ \left(\frac{p_b - p_{thresh}}{1 - p_{thresh}} \right)^2 & p_{thresh} \le p_b \end{cases} \tag{3.5}$$

When $p_{thresh} > p_b$ (low congestion), ϕ is 0.0. During this period there is no CHOKe activity. When $p_{thresh} \le p_b$ (high congestion), the value of ϕ increases rapidly. As a result, more packets from non-responsive and TCP unfriendly flows are dropped at the bottleneck link. An arriving packet is picked probabilistically based on the value of ϕ. This packet is compared with a randomly chosen packet from the buffer. If they have the same flow ID, they are both dropped. Otherwise the randomly chosen packet is kept in the buffer (in the same position as before) and the arriving packet is queued if the buffer is not full; otherwise it is dropped.

Figures 3.2 and 3.3 show the membership functions (MF1 and MF2) that are used for fuzzifying the variables α and β respectively. Figure 3.4 shows the membership function (MF3) used in the defuzzification process in order to generate the change in packet marking/dropping probability Δp_b.

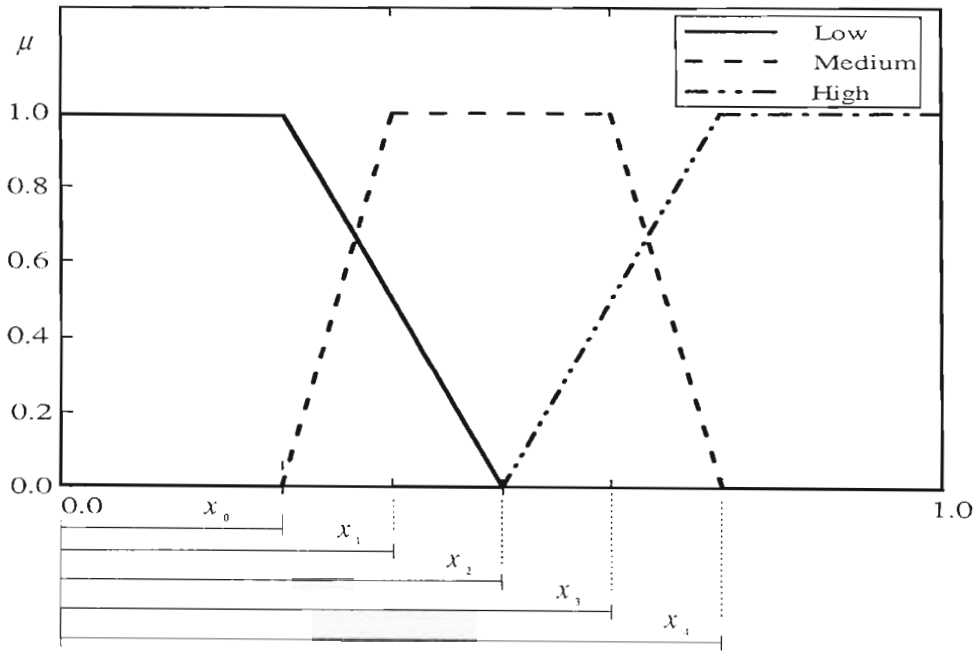


Figure 3.2: Membership Function (MF1) for the backlog factor

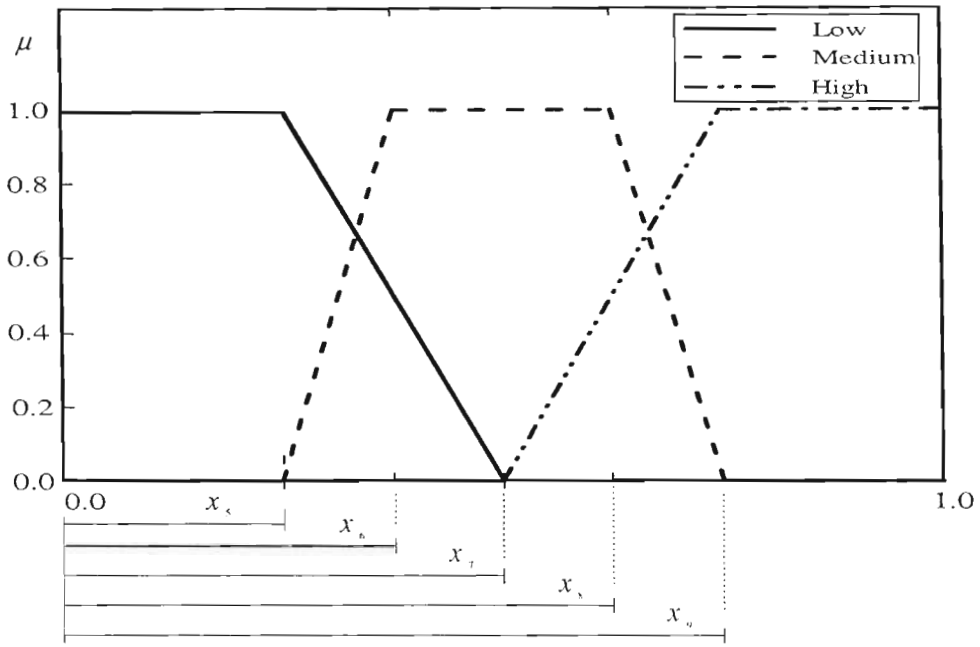


Figure 3.3: Membership Function (MF2) for the packet arrival factor

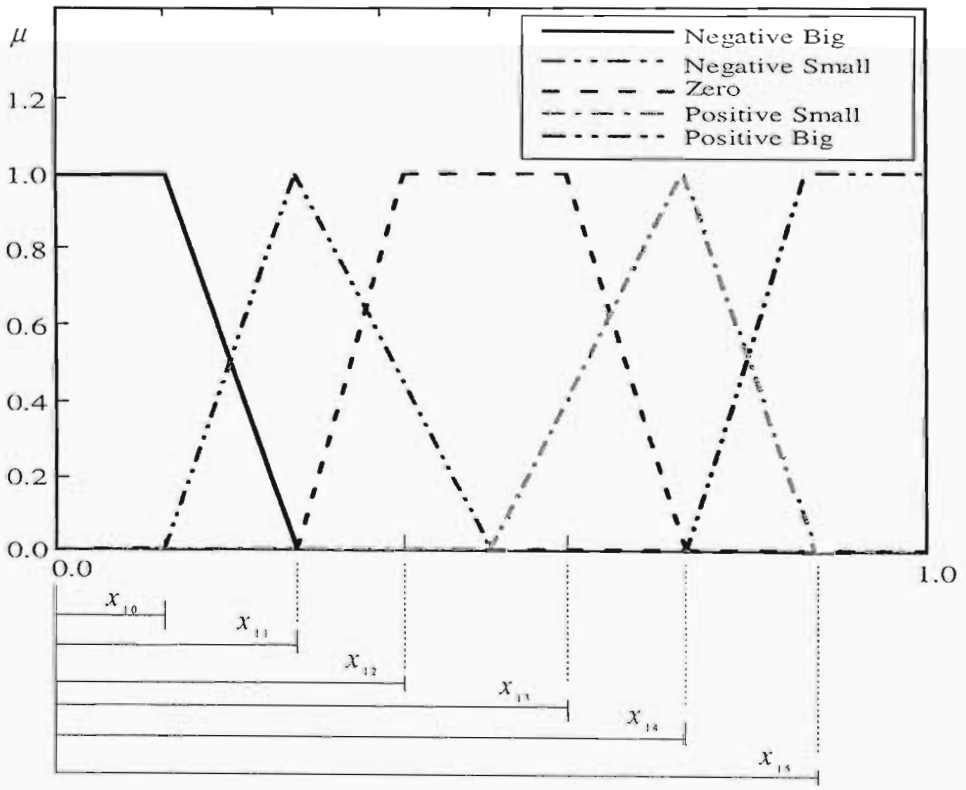


Figure 3.4: Membership Function (MF3) for the change in packet marking probability

The 18-dimensional parameter vector P that determines the membership functions and packet marking probability variation is expressed as follows

$$P = [x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}, x_{16}, x_{17}] \tag{3.6}$$

The definition of these elements is presented as follows:

1. x_0, x_1, x_2, x_3, x_4 are parameters for the backlog factor (α) membership function (MF1) as shown in Figure 3.2.
2. x_5, x_6, x_7, x_8, x_9 are parameters for the packet arrival (β) rate membership function (MF2) as shown in Figure 3.3.
3. $x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}$ are parameters for the change in packet marking probability (Δp_b) membership function (MF3) as shown in Figure 3.4.
4. x_{16}, x_{17} denote the maximum negative and positive variations (ΔP_{neg} and ΔP_{pos}) of the change in packet marking probability. The output from the defuzzification process which falls in the range [0,1.0] is scaled to $[\Delta P_{neg}, \Delta P_{pos}]$.

Parameters for individual membership functions must always be sorted in ascending order. For instance, for MF1 the following

$$x_0 < x_1 < x_2 < x_3 < x_4 \tag{3.7}$$

must always be true. The same applies to MF2 and MF3. The elements in equation (3.6) are determined in Section 3.3.

3.3 MOPSO in FLCD Parameter Optimization

An evaluation framework [BRH03] for AQM schemes outlines link utilization, packet loss rate, delay, jitter and fairness as the main metrics for evaluating AQM schemes. These metrics are, in most cases, conflicting and non-commensurable. For instance, when link utilization is high, the packet loss rate also becomes high because the buffer is generally full. A comparative study [BRH04] of AQM schemes that were proposed between 1999 and 2003 further reveals that these schemes perform well for a particular metric and poorly for another. For example, the CHOKe algorithm [PPP00] provides much better fairness but fails to keep link utilization high. Although GREEN [BZ02] maintains a very low queuing delay its behaviour is closest to the Drop-tail, except that the queue is generally empty or small one while the Drop-tail queue is always full.

These observations motivated us to model IP congestion detection as a multi-objective (MO) problem. We use the Multi-objective Particle Swam Optimization in order to achieve optimal performance on all the major metrics of IP congestion control.

3.3.1 Basics of Multi-objective Optimization

Let $S \subset \mathrm{R}^N$ be an N-dimensional search space and $f_i(x): S \subset \mathrm{R}^N \rightarrow \mathrm{R}$, $i = 1,...,k$, be k objective functions defined over S. Therefore, a general MO problem can be defined in the following format [COE99] [ZIT99]:

$$\text{Optimize} \quad f(x) = \left[f_1(x), f_2(x),..., f_k(x)\right]^T : f : \mathrm{R}^N \rightarrow \mathrm{R}^k \tag{3.8}$$

$$\text{subject to} \quad g_j(x) \leq 0 \; for \; j = 1,.., p \tag{3.9}$$

$$\text{and} \quad h_j(x) = 0 \; for \; j = p+1,.., m \tag{3.10}$$

where $x = (x_1, x_2,..., x_n) \in S$, $g_j(x)$ and $h_j(x)$ are the equality and inequality constraints respectively.

As already mentioned, the objective solutions are generally competing and non-commensurable such that it is impossible to obtain the global optimum at the same point for all the objectives. The goal of MO is to provide a set of *Pareto optimal* solutions to the aforementioned problem. *Pareto dominance* and *optimality* are defined as follows [ZIT99]:

Definition 1 (Pareto Dominance): A given vector $x = (x_1, x_2,..., x_n)$ is said to dominate $x' = (x_1', x_2',..., x_n')$ if and only if $\forall i \in \{1,2,...,n\}, x_i \leq x_i'$ and $\exists i \in \{1,2,...,n\}, x_i < x_i'$. This property is used to define Pareto optimal points.

Definition 2 (Pareto Optimality): For a general MO problem, a given solution $f(x) \in F$ (where F is the feasible solution space) is *Pareto Optimal* if and only if there is no $f(x') \in F$ that dominates $f(x)$. The set of all Pareto optimal solutions of an MO problem is called a *Pareto optimal set* and it is denoted as P^*.

Definition 3 (Pareto Front): The set $PF^* = \{(f_1(x), f_2(x),..., f_k(x)) \mid x \in P^*\}$. A Pareto front PF^* is called *convex* if and only if there exists $x'' \in PF^*$, such that

$$\lambda \| x \| + (1-\lambda) \| x' \| \geq \| x'' \|, \quad \forall x, x' \in PF^*, \; \forall \lambda \in (0,1) \tag{3.11}$$

and it is called *concave* if and only if there exists $x" \in PF^*$, such that

$$\lambda \| x \| + (1-\lambda) \| x' \| \leq \| x" \|, \quad \forall x, x' \in PF^*, \ \forall \lambda \in (0,1) \tag{3.12}$$

A Pareto front can be convex, concave or partially convex and/or concave and/or discontinuous. The last three cases present the greatest difficulty for most MO techniques.

3.3.2 MOPSO Theory

MultiObjective Particle Swarm Optimization (MOPSO) is a special case of the Particle Swarm Optimization (PSO) algorithm that is specifically tailored for problems with multiple objectives. PSO is a relatively new population-based stochastic algorithm introduced by Kennedy and Eberhart [KE95]. Like Ant Colony optimization (ACO), it belongs to the category of *Swarm Intelligence* methods, which are inspired from the social dynamics and the emergent behavior that arise in socially organized colonies [BDT99] [KE95].Like other evolutionary computation techniques such as Genetic Algorithms (GA) [BF92], PSO is initialized with a population of random solutions known as a swarm which evolve by updating the generations until an optimal solution or a termination criteria is reached. However, unlike GA, PSO has no evolution operators such as crossover and mutation. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles. Compared with genetic algorithms (GAs), the information sharing mechanism in PSO is significantly different. In GAs, chromosomes share information with each other. So the whole population moves like one group towards an optimal area. In PSO, only the global best value, *gBest* (or *lBest)* is shared amongst the particles. The evolution only looks for the best solution. Compared with GA, all the particles tend to converge to the best solution much faster.

A particle moves at an adaptable velocity within the search space and retains a memory of the best position it ever encountered. A particle's movement vector is dynamically adjusted according to its own and others' experiences. Assume a D-dimensional search space $f : S \subset \mathbb{R}^D \rightarrow \mathbb{R}$ and a swarm $\mathbb{S} = \{X_1, X_2, ..., X_N\}$ of N particles. The i-th particle $X_i \in \mathbb{S}$ is in effect a D-dimensional vector $X_i = (x_{i0}, x_{i1}, ..., x_{i(D-1)})$. The velocity of the particle $V_i \in S$ and the best previous position encountered by the particle $P_i \in \mathbb{S}$ are also both D-dimensional vectors expressed as $V_i = (v_{i0}, v_{i1}, ..., v_{i(D-1)})$ and $P_i = (p_{i0}, p_{i1}, ..., p_{i(D-1)})$ respectively. Assume P_{g_i} to be

the global best position among all particles in the neighborhood of the i-th particle, and t to be the iteration counter, and then the swarm velocity equation is as follows:

$$V_i(t+1) = \omega V_i(t) + c_1 r_1 (P_i(t) - X_i(t)) + c_2 r_2 (P_{g_i}(t) - X_i(t)) \tag{3.13}$$

where $i = 1,...,N$, c_1 and c_2 are constants denoting cognitive and social parameters respectively. In [KE95]The values of c_1 and c_2 are chosen in the range $[0.5, 2.5]$. They are applied in order to include the influence of the particle's previous best position $P_i(t)$ and the best position $P_{g_i}(t)$ among all particles in the neighborhood of the i-th particle respectively. Parameters r_1 and r_2 are random numbers uniformly distributed within $[0, 1]$. Parameter ω known as the inertia weight helps to dampen the velocities of the particles and to assist in the convergence to the optimum point at the end of the optimization iteration.

The first part in (3.11), known as the inertia component, is the current velocity of the particle providing momentum for the particle to move at the same speed. The second part, known as the cognitive component, represents the thinking of an individual particle. It accelerates the particle towards its own best position. The last part known as the social part accelerates the particle towards the best position of all particles in order to converge to the global optimum value. Kennedy [KE97] further defined the arbitrary parameter $V_m = (v_{m1}, v_{m2}, ..., v_{mD}) \in \mathbb{S}$ to be the upper limit of the velocity. Whenever, a vector element exceeds the corresponding element of V_m, that element is reset to its upper limit. The position of each particle is updated at each iteration by using

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{3.14}$$

For the basic PSO (single objective case), one objective function is evaluated after every generation update until an optimal solution or a termination criterion is reached. Its high speed of convergence and its relative simplicity has made PSO a highly viable candidate for solving not only single objective functions but also multiobjective optimization problems. Recently, a plethora of MOPSO algorithms has been reported in literature [FIE04] [PC04] [COE04]. These algorithms exhibit better performance characteristics compared to the traditional multiobjective evolutionary algorithms (MOEAs) such as the Pareto Archived Evolution Strategy (PAES) [KC00] and the Nondominated Sorting Genetic Algorithm II (NSGAII) [DAP02].

3.3.3 Problem Formulation

The formulation of the IP congestion control problem requires two stages: the development of objective functions and the determination of the constraints for the optimization process.

3.3.3.1 Development of Objective Functions

Objective functions are developed based on the following metrics: link utilization, packet loss rate, delay and jitter. The fairness metric is not included in this process partly because it has been addressed in section 3.2 and partly because the evaluation of fairness is more computationally intensive compared to the other metrics. Therefore, the four objective functions are derived as follows:

Maximizing Link Utilization (U)

Let R be the total number of packets that have successfully traversed the bottleneck link during simulation time T ; let b be the packet size in bytes; let C denote the network capacity in bytes per second. The link utilization U is:

$$U = \frac{Rb}{CT} \tag{3.15}$$

Since MO problems are generally solved by minimizing the objective functions, the link utilization objective function F_1 is presented as a reciprocal of link utilization

$$F_1 = \frac{CT}{Rb} \tag{3.16}$$

Minimizing Packet Drop Rate (PDR)

Let R' be the total number of packets received at various destination nodes, and d the total number of packets dropped during simulation time T .The packet drop rate objective function F_2 is determined as follows

$$F_2 = \frac{d}{R'} \tag{3.17}$$

Minimizing Average Delay

Average delay is a function of the average queue size. Let q_{av} denote average queue size. Assuming that a total of N samples $q_1, q_2, ..., q_N$ of queue size are taken over the simulation time T, then the average delay objective function F_3 is found as follows

$$F_3 = q_{av} = \frac{\sum_{i=1}^{N} q_i}{N} \tag{3.18}$$

Minimizing Average Jitter

This strives to minimize the end-to-end delay variation experienced by the packets. Jitter causes unfairness in average transfer delay which affects intelligibility in real-time traffic. The average jitter objective function F_4 is found as follows

$$F_4 = \frac{\sum_{i=1}^{N} (q_i - q_{av})^2}{N-1} \tag{3.19}$$

3.3.3.2 Constraints

- Most congestion detection algorithms exhibit typically very high link utilization. Therefore, we limit the link utilization U as follows

$$U > 0.95 \tag{3.20}$$

- Most congestion detection algorithms exhibit very low packet drop rates utilization. Therefore, we limit the packet drop rate PDR as follows

$$PDR < 0.15 \tag{3.21}$$

3.3.3.3 Problem Statement

The congestion detection MO problem can therefore be stated as follows:

$$min\left[F_1, F_2, F_3, F_4\right] \tag{3.22}$$

subject to the constraints in (3.20) and (3.21).

3.3.4 MOPSO Implementation and Optimization Results

3.3.4.1 The Optimization Process

This process is motivated by the work in [PHS05] where a basic PSO is used in order to optimize a rule-based system. In our multiobjective approach, the Adaptive MOPSO (AMOPSO) algorithm, proposed by Pulido and Coello [PC04], is used in order to optimize the membership function parameters defined in (3.6) and referred to as a particle in this section and beyond. The AMOPSO algorithm divides the population of particles into several swarms (each with a fixed size). Each swarm over-flies a specific region of the Pareto optimal set (i.e. the decision variable space), and has its own niche of particles and a swarm of particle guides known leaders. The hierarchical single clustering algorithm is used to associate leaders to a swarm. A particle randomly chooses a leader from the corresponding swarm of leaders. The AMOPSO algorithm was validated by using three standard test functions which are currently adopted in the evolutionary multiobjective optimization community. Its performance was compared with the Nondominated Sorting Genetic Algorithm II (NSGAII) [DAP02], the Pareto Archived Evolution Strategy (PAES) [KC00] and the MOPSO algorithm proposed in [COE04]. Results in [PC04] show that the AMOPSO algorithm generates significantly improved Pareto fronts when compared to the other algorithms. Quantitative assessment based on Error ratio (ER), Generational Distance (GD) and Spacing as performance metrics also shows that the AMOPSO algorithm shows that the average performance of the AMOPSO algorithm is generally good.

In order to solve the IP congestion problem, the 18-dimensional decision variable space is defined as follows:

1. Parameters $x_0, x_1, ..., x_{15}$ in (0.0,1.0).
2. Parameter x_{16} in [−0.0005,0.0).
3. Parameter x_{17} in (0.0,0.0005].

The range for parameters x_{16} and x_{17} must be small in order to avoid drastic changes in the congestion notification. The computational flow of the optimization process is as follows:

function AMOPSO Algorithm

BEGIN

For each swarm

1. Initialize randomly the velocity and position of $n_{particles}$ particles and maintain the particles within the search space.
2. Run the FLCD script using the initialized particle positions.
3. Evaluate the four objective functions (See Section 3.3) based on the results from the FLCD script.
4. Initialize *gleader* set (i.e. the set of global leaders)

ENDFor

DO

For each swarm

DO

For each particle

5. Select a leader
6. Perform flight
7. Update values for velocity and position using equations (3.13) and (3.14).
8. Run the FLCD script using the updated particle positions.
9 Evaluate the four objective functions based on the results from Step 8.

If it is a leader **then** add to *gleader* set

EndFor

While maximum number of internal iterations *sgmax* is not reached

10. Store leaders in *gleader* set in n_{swarms}.

EndFor

11. Assign each leader group to a swarm

While maximum number of iterations *GMax* is not reached.

END

The AMOPSO algorithm requires the following parameters:

- *GMax*: it refers to the maximum number of generations that the algorithm will be executed.
- $n_{particles}$: it refers to the total number of particles that will be over-flying the search space.

- n_{swarms} : it refers to the number of particle groups.
- $sgmax$: it refers to the maximum number of internal generations that the particles of each swarm will run before sharing their leaders.

The complete execution process of the AMOPSO algorithm is divided in three stages: initialization, flight and generation of results. At first, every swarm is initialized. Each swarm creates and randomly initializes its own particles within the decision variable described in the earlier stage of this section. The particle elements are then sorted in ascending order for each membership function as per equation (3.7). For each swarm, the FLCD algorithm is evaluated using the initialized particle positions. Details of the FLCD script which invokes the FLCD algorithm are presented in next section. The four objective function values are evaluated for each particle based on the results from the FLCD script. A set of leaders among the particle swarm set is generated based on Pareto ranking subject to the constraints in (3.20) and (3.21). Next, the algorithm executes the flight of every swarm; then it performs the clustering algorithm to group the particles. This is performed until $GMax$ iterations are reached. The execution of the flight of each swarm can be seen as an entire PSO process (with the difference that it only optimizes a specific region of search space). First, each particle selects a leader from the swarm of leaders associated with it. The particle's velocity and position are updated in the direction of the selected leader. The particle elements are then sorted in ascending order for each membership function as per equation (3.7).Then the FLCD algorithm is evaluated using the new particle position. The four objective functions are evaluated. If the updated particle satisfies the constraints in (3.20) and (3.21) and is not dominated by any member of the leaders set, it becomes a new leader. The execution of the swarm starts again until a total of $sgmax$ iterations are reached. Once all the swarms have finished their flights, a clustering algorithm is invoked in order to group the closest particle guides into n_{swarms} swarms. These particle guides will try to outperform each swarm in the next iteration. The third and final stage reports all nondominated solutions found.

3.3.4.2 Implementation of the FLCD Script

The FLCD script is implemented on the Network Simulator (NS-2.28). Figure 3.5 shows the network topology on which it is implemented. The bottleneck bandwidth is 10Mbps with a propagation delay of 40ms. The buffer is set to 90 packets. All the other links have 100Mbps

capacity with 2ms propagation delay. Traffic flow is from Router1 to Router2. The FLCD algorithm is activated on the bottleneck link between Router1 and Router2. All simulations use NS-2.28's *NewReno* TCP variant with an initial congestion window *cwnd* of 3 segments (per [AFP02]), a Maximum Segment Size (MSS) of 1500 bytes and the receiver acknowledging each segment. Packet-based ECN marking is used. The optimization script runs for 100 seconds. 60 persistent FTP flows start randomly in the interval [0s-5s] while 10 UDP traffic flows are activated in the following intervals, [20s-30s] and [80s-90s].

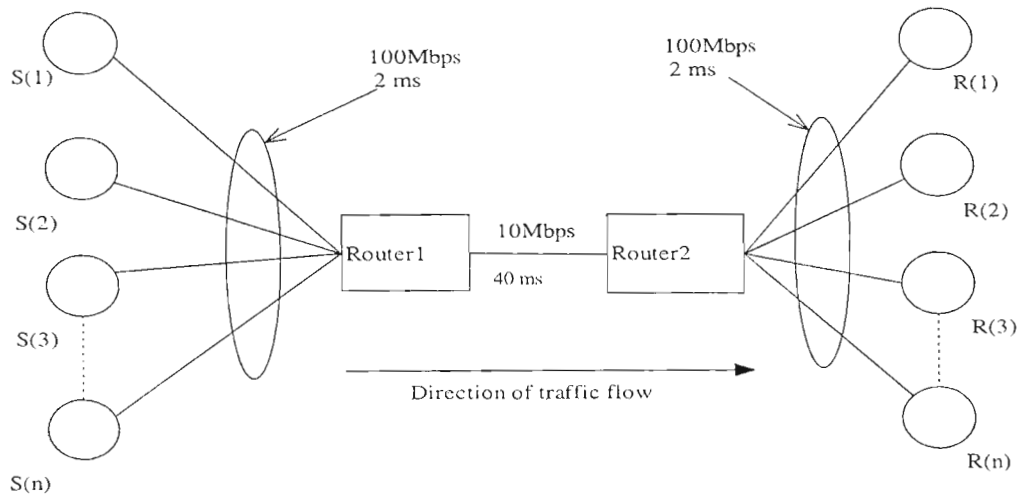


Figure 3.5: Network Topology

3.3.4.3 Parameters for the Optimization process

The C++ source code of the AMOPSO algorithm was downloaded from the EMOO repository located at: http://www.lania.mx/~ccoello/EMOO. After customizing it to the congestion MO problem, it was compiled using Fedora Core2's GNU C compiler running on a Dell Optiplex GX280 PC with the following system characteristics: A CPU frequency of 3.00GHz and a RAM of 1.00GB. Since the FLCD script is implemented on the Network Simulator, the exchange of parameters between the FLCD script and the AMOPSO algorithm was done by using text files. A single optimization run was used because of two reasons:

- The AMOPSO algorithm has already been extensively validated against some of the state-of-the-art algorithms. Therefore, we have nothing prove about its viability. We rather concentrate on the task of ensuring that the decision variable space is thoroughly searched by employing a larger number of generations $GMax$. The choice of a large

GMax is also partly influenced by the fact that there exists no knowledge of the true Pareto front for the problem at hand. Except for *GMax*, we use the same parameters as in [PUL05].Other application specific implementations of MOEAs have used a single optimization run as evidenced by the work of Zhao and Cao [ZC05] and Rivas-Davalos and Irving [RI05].

- The implementation of multiple independent optimization runs would be computationally expensive considering the fact that the AMOPSO algorithm periodically invokes the FLCD script, which runs under the NS simulator, in order to evaluate the four objective functions. We therefore stick to a single optimization run with a large *GMax*.

Except for *GMax* all the parameters for the optimization process were set as in [PUL05].The settings are as follows: $n_{particles} = 40$, $GMax = 80$, $sgmax = 5$ and $n_{swarms} = 8$.These values were empirically derived in [PUL05]. A *GMax* of 80 denotes 16000 fitness function evaluations. The AMOPSO algorithm produces very competitive results using only 2000 fitness function evaluations. Therefore by using 16000 fitness function evaluations, a closer and better picture of the true Pareto Front would be found. Since the optimization results are non-reproducible, the use of more fitness function evaluations enhances the likelihood of getting similar results on different runs. We following parameters for the FLCD algorithm: $w_1 = 0.9$, $r_m = 5.0$ and $P_{thresh} = 0.15$.

3.3.4.4 Optimization Results

The optimization process generated 120 non-dominated solutions. The true Pareto front for this problem is a spread of solutions in the 4D space depicting the four objective functions. Since the graphical presentation of such a spread of solutions is practically difficult, we present four views of the generated Pareto front in 3D using Figure 3.6 – Figure 3.9.

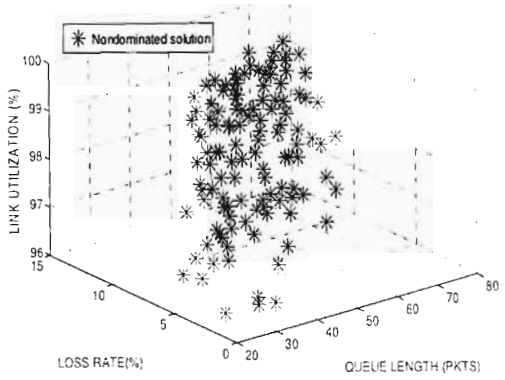


Figure 3.6: Queue Length versus Loss Rate versus Link Utilization

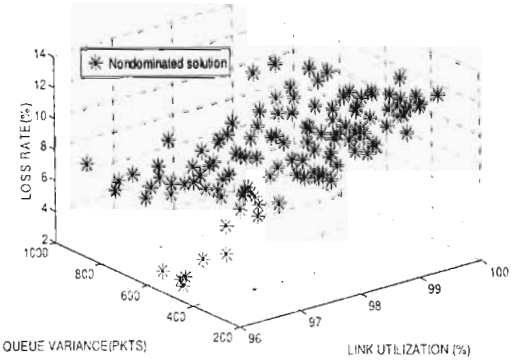


Figure 3.8: Link Utilization versus Queue Variance versus Loss Rate

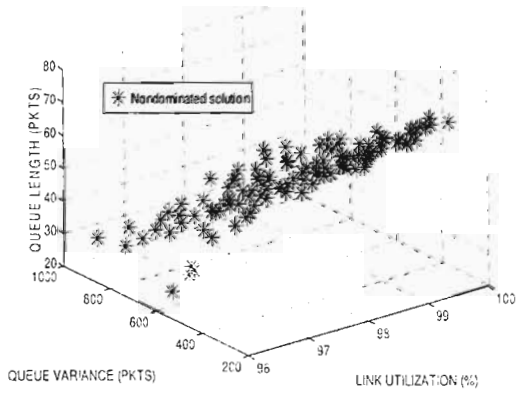


Figure 3.7: Link Utilization versus Queue Length versus Queue Variance

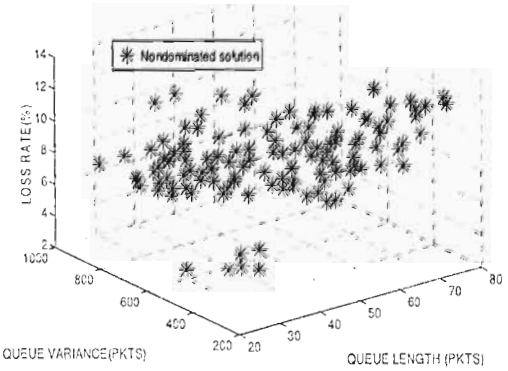


Figure 3.9: Queue Length versus Queue Variance versus Loss Rate

Figure 3.6 shows that link utilization increases as queue length increases. This leads to large queuing delays and packet loss rates. Figure 3.7 shows that queue variance is low when link utilization and queue length are high. This leads to low jitter for interactive applications. A major setback of this good attribute is, however, that it comes at the expense of delay which increases as queue length increases. The spread of particles in Figures 3.8 and 3.9 is quite similar. The only difference between the two plots is on the X-axis. The X-axis variable is link utilization for Figure 3.8 while it is queue length for Figure 3.9. The Y-axis and Z-axis variables are queue variance and loss rate respectively in both cases. The similarity in the spread of particles further confirms the fact that increasing link utilization comes at a cost of increased link delay.

3.3.4.5 Best Compromise Solution

In order to implement a practical FLCD algorithm, there is a need to obtain the best compromise solution from the set of Pareto optimal solutions. In order to do this, a Fuzzy inference algorithm [ZC05] is employed. A simple linear membership function is considered for each of the objective functions in the inference process. Let $F_i^{\min}$ and $F_i^{\max}$ denote minimum and maximum values for the i^{th} objective function for the entire set of Pareto Optimal solutions (120 in this case). Let F_i and $w_i \in (0,1]$ respectively denote the value and the weight of the i^{th} objective function for the solution at hand. The membership function u_i is defined as follows:

$$u_i = \frac{w_i * (F_i^{\max} - F_i)}{F_i^{\max} - F_i^{\min}} \tag{3.23}$$

Figure 3.10 illustrates the typical shape of the membership function. The membership function represents the degree of achievement of the original objective as a value between 0 and w_i with $u_i = w_i$ as completely satisfactory and $u_i = 0$ as completely unsatisfactory.

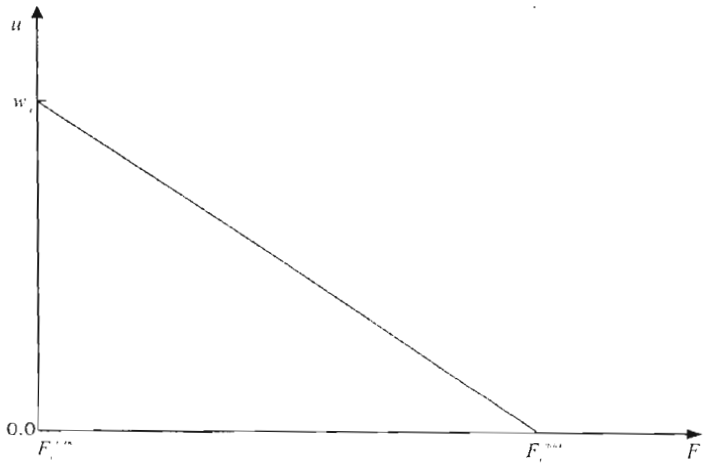


Figure 3.10: Objective Membership Inference function

We introduce a simple algorithm which configures the thresholds $F_i^{\min}$ and $F_i^{\max}$ automatically prior to the Fuzzy inference algorithm. This algorithm parses the set of Pareto optimal solutions

in order to obtain the minimum and maximum thresholds $F_i^{\min}$ and $F_i^{\max}$. For each non-dominated solution k, the normalized membership function u^k is calculated as

$$u^k = \frac{\sum_{i=1}^{N_{obj}} u_i^k}{\sum_{k=1}^{M}\sum_{i=1}^{N_{obj}} u_i^k} \tag{3.24}$$

where M is the number of non-dominated solutions, and N_{obj} is the number of objective functions. The solution that attains the highest membership u^k in the fuzzy set is chosen as the best compromise solution.

Let w_1 denote the weight for link utilization, w_2 denote the weight for the loss rate, w_3 for delay and w_4 for the jitter. In this implementation, the link utilization is deemed to be the most important, seconded by the loss rate objective because these objectives are necessary for all types of traffic. The jitter and delay objectives are least significant because they are generally tailored for real-time (UDP) traffic which accounts for (22±11) % packet composition of internet traffic. Therefore, we employ the following weighting mechanism:

$$w_1 = 2w_2 = 3w_3 = 3w_4 \tag{3.25}$$

When the Fuzzy Inference algorithm is applied to the Pareto set of optimal solutions the dimensions of the best compromise solution obtained are as follows:

$$\begin{aligned} P = [&0.01, 0.02, 0.03, 0.04, 0.29, 0.95, 0.96, 0.97, 0.98, 0.99, \\ &0.01, 0.02, 0.03, 0.34, 0.61, 0.64, -0.0005, 0.0005] \end{aligned} \tag{3.26}$$

These parameters are used in configuring the membership functions of the practical FLCD algorithm.

3.4 Simulation Results and Performance Analysis in Best effort IP Networks

After obtaining the optimal membership functions, three experiments are conducted in order to compare the optimized FLCD with the basic Fuzzy AQM [FYX02] and the Random Explicit Marking (REM) [ALLY01] algorithms based on the five metrics for network performance evaluation as proposed in [BRH03]. These metrics are given in Appendix A. We use the same

network topology (Figure 3.5) and simulation platform (NS-2.8). Next, we describe three experiments and the observed results.

3.4.1 Experiment 1: Congestion Control with Packet Dropping

In this experiment, we compare the performance of the MOPSO FLCD algorithm against REM, basic Fuzzy AQM and the Drop-tail mechanism. ECN is disabled in all the four cases such that congestion notification is done through packet dropping. We simulate 80 persistent FTP flows, competing for bottleneck link. 10 UDP flows are introduced in the following intervals [20s-30s] and [100s-110s]. The FTP flows start randomly within the first 5s and they run up to the end of the simulation. The simulation runs for 150s. The buffer size was set at 90. In addition to the three schemes, we also simulated a simple drop-tail buffer and used it as a baseline for performance comparison. Table 3.2 shows the packet loss rate, link utilization, average queue length, standard deviation of the queue length and fairness for the four schemes.

Table 3.2: Comparison of AQM schemes with Packet Dropping

AQM	**Loss Rate**	**Link Utilization**	**Average Queue Length (packets)**	**Queue Variance (packets)**	**Fairness (%)**
Drop-tail	22.26%	99.65	78.14	124.36	88.96
REM	22.66%	95.07	40.07	1101.09	97.19
Basic Fuzzy	20.64%	96.48	44.65	743.44	97.14
MOPSO FLCD	18.31%	98.52	52.38	665.61	98.0

From Table 3.2, we see that the congestion control algorithms do not offer much help in reducing the packet loss rate compared to the drop-tail mechanism. The MOPSO FLCD algorithm, however, manages to achieve the lowest packet rate and fairness. The drop-tail mechanism registers the highest link utilization but at cost of high average queue length. This will result in network latency. REM exhibits low average queue length but at a cost of link utilization and queue instability as portrayed by the queue variance. The MOPSO FLCD algorithm generally outperforms the other algorithms in that it achieves the lowest loss rate, high link utilization and moderate values for average queue length, queue variation and fairness.

Since ECN marking is believed to be a better choice for congestion notification in TCP connections, we will only study the congestion control schemes in the rest of the simulations.

3.4.2 Experiment 2: Vary the congestion level at the bottleneck link

In this experiment, packet loss rate and link utilization are evaluated. Delay and jitter are evaluated for the embedded UDP traffic. 30 web servers are connected to Router1 with a corresponding number of web clients connected to Router2. We also attach 15 web clients to Router1 and 15 web servers to Router2 to provide background traffic on the return path. We activate 5 web sessions on each client-server connection. The number of FTP Traffic flows from Router1 to Router2 is varied by using 10, 20, 30,40,50,60,70,80,90 and 100 flows in order to establish different levels of congestion. The FTP flows start randomly within the initial 5s of the simulation while the web-traffic connections start within the first 10s. 10 UDP flows from Router to Router2 are activated in the following intervals [20s-25s], [100s-110s] and [140s-150s].UDP traffic rate is set at 1Mbps.The simulations run for 150s at every instance. Figure 3.11- Figure 3.14 show the results.

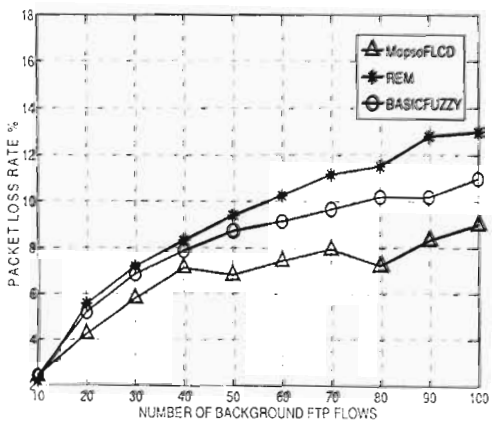


Figure 3.11: Packet Loss Rate

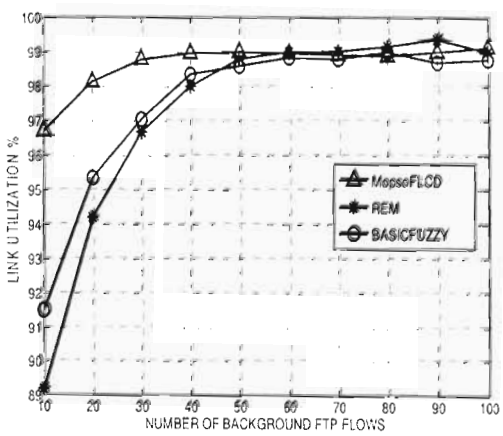


Figure 3.12: Link Utilization

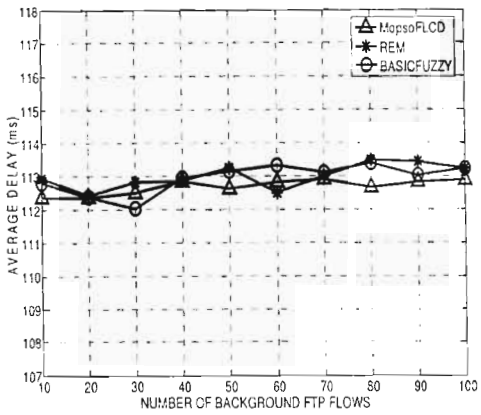


Figure 3.13: Delay for UDP traffic

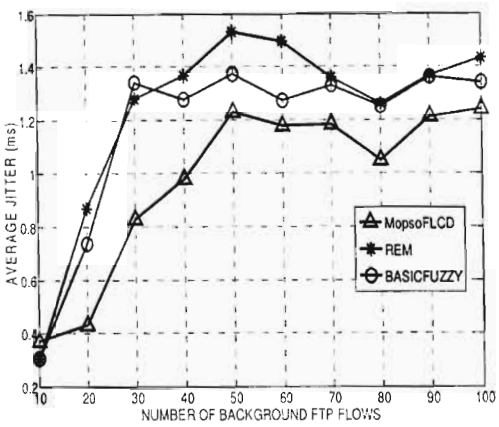


Figure 3.14: Jitter for UDP traffic

Fig.3.11 shows that the MOPSO FLCD algorithm has the lowest packet loss rate ranging from 2.392% to 9.083% while the basic Fuzzy algorithm comes second with a loss rate ranging from 2.409% to 11.07%. REM, with a loss rate ranging from 2.192% to 12.935%, competes fairly well with the fuzzy approaches when the number of background FTP flows is less than 40 but as the number of background FTP flows increases the REM control law fails to adjust to detect the increasing congestion levels.

Fig.3.12 shows that the MOPSO FLCD algorithm exhibits the highest link utilization (96.716% to 99.018%) when the number of FTP flows is low. The basic Fuzzy algorithm (91.518% to 98.61%) and the REM algorithm (89.203% to 98.818%) come second and third respectively. When the number of FTP flows exceeds 50, all the three mechanisms exhibit predominantly high levels of link utilization.

Fig.3.13 shows the queuing delay link exhibited by the three algorithms as experienced by UDP packets as they traverse the bottlenecck. The average delays are as follows: 112.59ms for MOPSO FLCD, 113ms for Basic Fuzzy and 113.048ms for REM.Although the MOPSO FLCD algorithm slightly outperforms the other two algorithms, the performance on this metric is basically the same.

Fig.3.14 shows that the MOPSO FLCD algorithm exhibits the least jitter with an overall average of 0.985ms. The basic Fuzzy algorithm comes second with an overall average value of 1.1975ms while the REM algorithm comes third with an overall average value of 1.2735ms.

3.4.3 Experiment 3: Vary the rate of UDP Traffic

In this experiment, we simulated 50 FTP flows, competing for bottleneck link. 20 UDP flows are introduced in the following intervals [20s-30s] and [100s-110s] while the FTP flows start randomly within the first 20s and they run up to the end of the simulation. The simulation runs for 150s. UDP flow rate is varied by using 0Mbps, 1Mbps, 2Mbps up to 15 Mbps in order to determine the Fairness metric as the data rate of unresponsive flows increases. Figure 3.15 shows the performance of the three algorithms as UDP traffic rate increases.

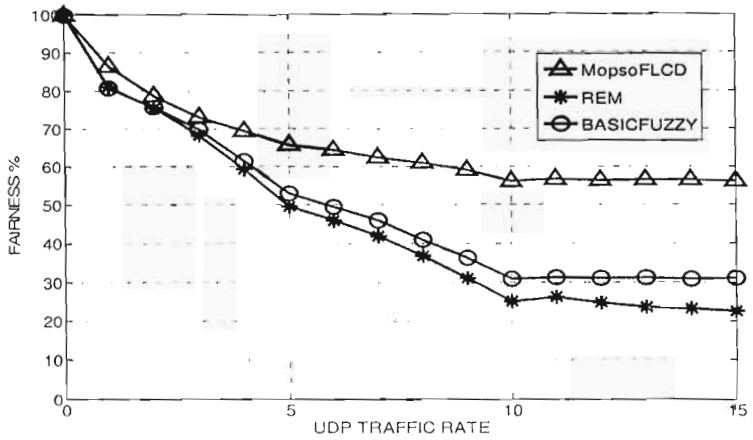


Figure 3.15: Fairness

Figure 3.15 shows that all three mechanisms exhibit very high levels of fairness when UDP traffic rate is low. The situation changes as UDP traffic rate increases. The MOPSO FLCD algorithm maintains a fairly high level of fairness as UDP traffic rate increases while the level of fairness in REM and Basic Fuzzy algorithms decreases exponentially. The MOPSO FLCD algorithm exhibits the highest fairness with an average of 66.155%. The Basic Fuzzy algorithm (49.97%) and REM (45.97%) come second and third respectively. Beside the effect of the optimization process, the MOPSO FLCD algorithm achieves a high fairness level courtesy of the embedded CHOKe algorithm. Apart from ensuring that TCP flows are guaranteed a fair share of the bottleneck link in light of unresponsive flows, the MOPSO FLCD algorithm is also a good tool for averting Denial of Service (DoS) attacks and routing loops.

3.5 FLCD in Proportional Differentiated Services (PropDiffServ) IP Networks

The superior performance of the MOPSO FLCD algorithm in best effort IP networks gave us the impetus to implement it in the PropDiffServ IP network environment.

3.5.1 Implementation

The implementation of the MOPSO FLCD congestion detection mechanism in the Prop-DiffServ is shown in Figure 3.16. The number of service classes is denoted by N .When packet arrives at the Prop-DiffServ link it is enqued into a particular queue based on its class. Each queue is managed by a separate FLCD algorithm. Therefore, each queue (class) is treated as a FIFO buffer just like in the best-effort FLCD algorithm.

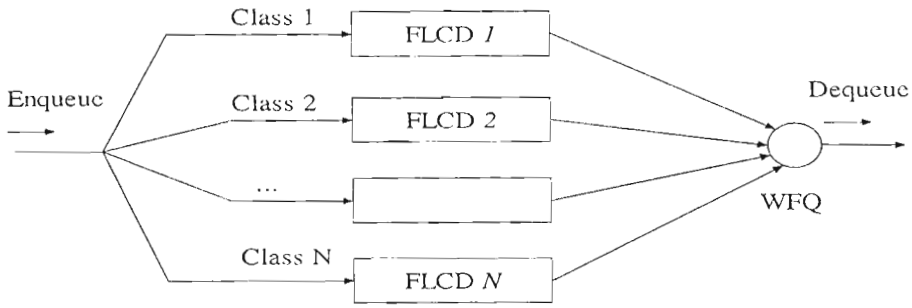


Figure 3.16: Fuzzy Logic Congestion Detection in the PropDiffServ Network

The Weighted Fair Queuing (WFQ) is used as the scheduling mechanism in the dequeuing routine. The WFQ algorithm is parameterized by a weight vector W where W_i is the proportion of capacity class i when there are packets available for transmission for all classes. This ensures that the performance of the low priority queues is guaranteed. The constraints for the weights are

$$W_i > 0 \tag{3.27}$$

and

$$\sum_{i=1}^{N} W_i = 1. \tag{3.28}$$

Recently, Joutsensalo *et al.* [JHP03] have proposed an adaptive weighted fair queue based algorithm for channel allocation. The weights are adapted by using revenue as a target function. We have used this version of WFQ along with the FLCD algorithm. The Nortel DiffServ

implementation in NS-2.28 does not employ the WFQ scheduling algorithm. Therefore, we had to download the C++ source codes for the WFQ algorithm from http://www.cc.jyu.fi/~sayenko/src/ and patch them to the DiffServ codes in NS-2.28. The pseudo codes for the PropDiffServ FLCD algorithm and the PropDiffServ enque and deque routines are shown in Figure 3.17 and Figure 3.18 respectively.

Every τ seconds:

for$(i = 1; i++; i \le N)$*{*

// Computing the packet marking probability for queue i

Evaluate $\alpha(i)$ *and* $\beta(i)$;

Evaluate $\Delta p_{bi}(t)$;

$p_{bi}(t) \leftarrow \Delta p_{bi}(t-\tau) + \Delta p_{bi}(t)$;

}

Parameters for queue *i*:

$\alpha(i)$ = backlog factor, $\beta(i)$ = packet arrival factor,

$\Delta p_{bi}(t)$ = change in packet marking probability at time t,

$p_{bi}(t)$ = packet marking probability at time t.

Figure 3.17: FLCD algorithm in PropDiffServ IP scenario

```
Enque:
  for every incoming packet{
      Extract the codepoint from the packet's header;
      Extract queue k (from the PropDiffserv Table) that matches packet's codepoint;
      if (queue k is full) Drop incoming packet;
      else {
          Generate random number R ∈ [0,1.0];
          if (R < p_bk ) Mark the Congestion Experienced (CE) bit;
          Enque the packet in queue k;
        }
     }

Deque:
    WFQ selects queue k to deque based on Weight vector W;
    Deque fore packet from queue k;

where k = 1,2,...,N; N = number of traffic classes (queues);
      p_bk = packet marking probability for queue k.
```

Figure 3.18: PropDiffServ Enque and Deque Routines

3.5.2 Simulation Results and Performance Analysis

In this Section, we compare the performance of the PropDiffServ version of the MOPSO FLCD algorithm with the WRED algorithm. Figure 3.19 shows the PropDiffServ simulation topology. We use the same WFQ scheduler [JHP03] in both cases. We simulate both schemes using the Network Simulator (NS-2.28). A four-class PropDiffServ mechanism is implemented on the router. Class k is composed of traffic from source $S(k)$, where $k = 1,2,3,4$. The traffic sources are connected to the router through 100 Mbps, 2ms delay links. Traffic flow is from sources *S(1)*, *S(2)*, *S(3)* and *S(4)* through the router to the Destination. The router is connected to the Destination through a 10 Mbps, 40ms delay link. The buffer size is set to 50 for each queue. ECN marking is employed for all TCP flows. Two experiments are conducted in our simulations.

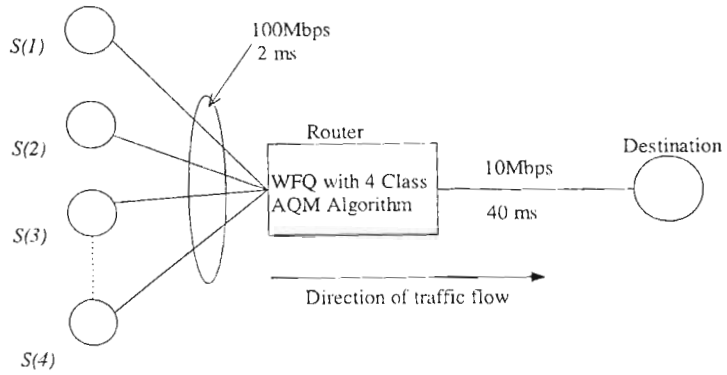


Figure 3.19: PropDiffServ Network Topology

3.5.2.1 Experiment 4: TCP Traffic Simulation

The aim of this experiment is to compare the packet loss rate, buffer occupancy and link utilization metrics of the FLCD algorithm with that of the WRED algorithm when traffic in all classes is generated by TCP sources. The simulation time for each scenario is 200sec.The simulation metrics for this experiment are presented in Table 3.3.

Table 3.3: Parameters for TCP Traffic Simulation

Class	W_i	Sources	Start	Stop
1	8/15	25 TCP	0	120
2	4/15	50 TCP	20	140
3	2/15	50 TCP	40	160
4	1/15	25 TCP	60	200

Table 3.4 shows the Packet loss rate and the link utilization metrics for the two algorithms.

Table 3.4: Performance Results for TCP Traffic simulation

Metric	FLCD	WRED
Packet loss rate	0.95%	2.73%
Link utilization	95.655%	94.956%

Table 3.4 shows that the FLCD algorithm reduces the packet loss rate by 65% while link utilization remains virtually the same in both cases. Traditionally, packet loss rate increases as link utilization increases. Any attempt to reduce packet loss rate results in drastic drops in link utilization. The situation is different in the FLCD algorithm because it is optimized to offer optimal performance on all the major metrics of IP congestion control.

Figure 3.20 and Figure 3.21 show the aggregate backlog for the FLCD algorithm and the WRED algorithm respectively.

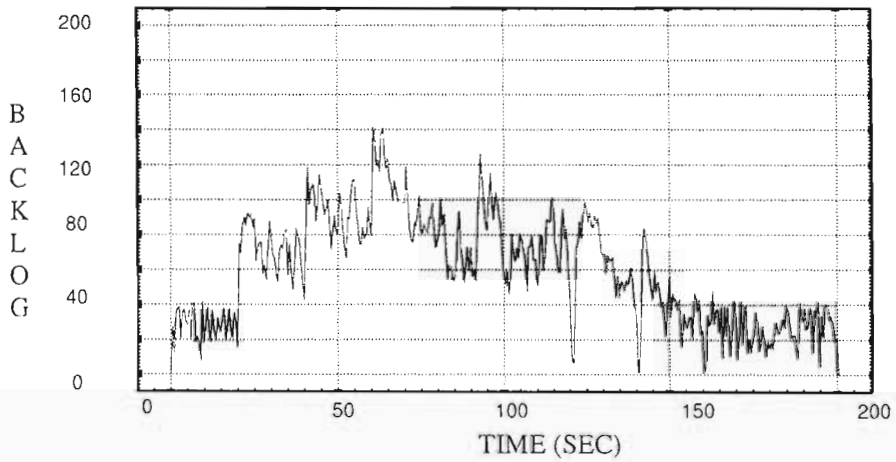


Figure 3.20: Backlog for the FLCD algorithm

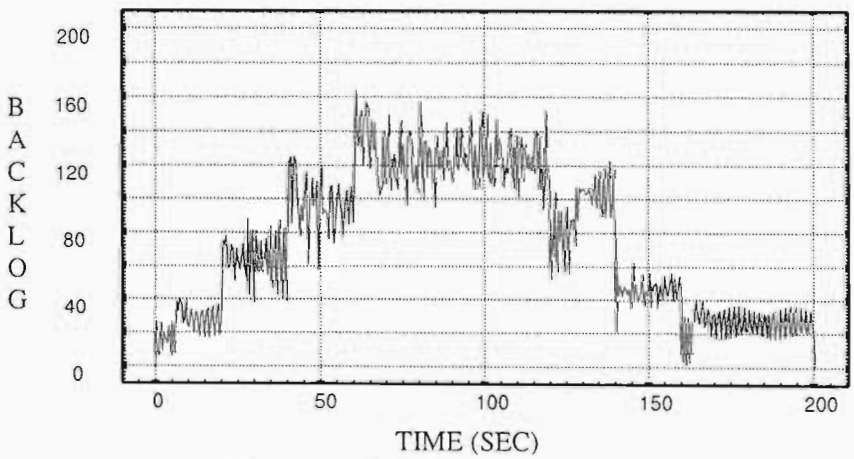


Figure 3.21: Backlog for the WRED algorithm

The average values of backlog are 59.5 for FLCD and 79.04 for WRED. This implies that the buffer size requirement in the FLCD algorithm is lower than in the WRED algorithm. The other

implication is that the queuing delay incurred by packets in the FLCD algorithm would be lower than in the WRED approach. This characteristic is very important for real-time traffic.

3.5.2.2 Experiment 5: Real Time Traffic Simulation

The aim of this experiment is to compare the performance of the FLCD algorithm with that of the WRED algorithm when one class carries higher priority video traffic while the rest of the classes carry TCP traffic of the same priority.NS-2.28's Constant Bit Rate (CBR) traffic generator is used in order to generate video traffic which is sent at a rate of 128Kbytes/sec. This traffic configuration has also been used in [MNT04]. All the traffic flows start at 0sec and stop at 200sec. The other simulation parameters are presented in Table 3.5. Figure 3.22 and Figure 3.23 show the results.

Table 3.5: Parameters for Real-time Traffic Simulation

Class	W_i	Sources
1	0.4	5,10,15,...,35 CBR UDP
2	0.2	40 FTP
3	0.2	40 FTP
4	0.2	40 FTP

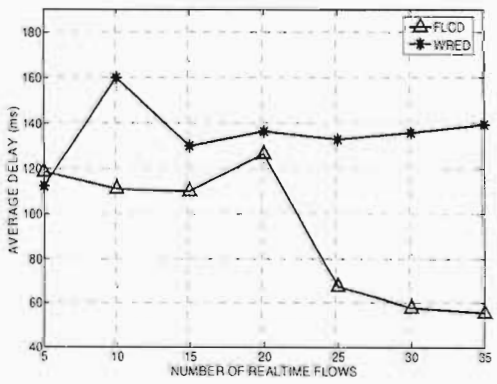


Figure 3.22: Average Delay for Real-time traffic

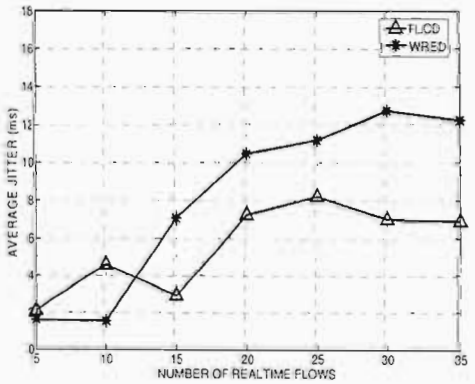


Figure 3.23: Average Jitter for Real-time traffic

Figure 3.22 shows that the FLCD algorithm exhibits a lower average delay. It's average delay is virtually constant when the number of real-time traffic flows is 20 or less. It however exhibits an

exponential decrease in delay as the number of flows increases beyond 20. It finally saturates around 55ms which is just 15ms higher than the link propagation delay (40ms). The overall average delay in the FLCD algorithm is 92.59ms. The WRED algorithm is unstable when the number of real-time traffic sources is low. Its average delay is 112.28ms for 5 real-time flows. It then shoots to an overall high of 159ms for 10 real-time flows before stabilizing as the number of real-time flows increases. WRED registers an overall average delay of 135.25ms.The exponential decrease in average delay in the FLCD algorithm is attributed to the fact that this algorithm becomes more aggressive in dropping real-time packets as their arrival rate increases. This helps to minimize queuing delay for the packets that have been enqueued successfully. This characteristic is very important for real-time traffic because this type of traffic operates within stringent time delays. If the arrival of a packet at the destination node is outside its time delay threshold, that packet is just dropped. Therefore, dropping packets aggressively as their arrival rate increases helps to get rid of useless late packets from the network.

Figure 3.23 shows that the WRED algorithm exhibits a lower jitter when the number of real-time flows is low i.e. 5 and 10. This advantage is however offset by the huge delay as shown in Figure 3.22. As the amount of real-time traffic increases, we observe that the FLCD algorithm outperforms the WRED algorithm. The overall average jitter is 5.57ms for FLCD and 8.112ms for WRED. This means that the intelligibility of real-time traffic is higher in the FLCD algorithm.

3.6 Chapter Summary

In this Chapter, we have developed a Fuzzy Congestion Detection algorithm by fusing the strengths of some of the traditional AQM schemes into the Fuzzy logic AQM framework. For instance, we have extended the basic Fuzzy algorithm by incorporating the CHOKe algorithm in order to address the issue of fairness which was not explicitly addressed in all the preceding fuzzy logic AQM approaches. We then modeled the congestion control problem as a multi-objective (MO) problem and used MOPSO in the automatic design of membership functions for the Fuzzy Logic Congestion Detection algorithm. The proposed algorithm addresses the major objectives of AQM by optimizing the membership functions of the input and output variables based on four objective functions. These objective functions are derived based on the following requirements: maximizing link utilization, minimizing loss rate, minimizing link delay and jitter. The effectiveness of the proposed approach is proved on both best effort and PropDiffServ IP

networks. In the best effort implementation, the performance of the proposed approach is compared with the basic Fuzzy algorithm and the REM algorithm. Performance results show that the proposed approach exhibits highest link utilization and fairness. It also exhibits the lowest packet loss rates and UDP traffic jitter. Its performance in terms of UDP traffic delay is similar to REM and the basic Fuzzy algorithm. In the PropDiffServ implementation, the performance of the proposed approach is compared with that of the WRED algorithm. Simulation results show that the FLCD approach achieves higher link utilization, lower packet loss rate, jitter and delay. The superior performance of the MOPSO FLCD is attributed to the effectiveness of the MOPSO dynamics in producing the Pareto set of optimal solutions from which the best compromise solution, which is used in the configuration of the FLCD algorithm, is drawn. The other striking advantage of this algorithm, compared to the basic Fuzzy algorithm, is that it uses fewer fuzzy sets leading to a smaller rule base and minimized memory requirements. Chapter 4 proposes a mechanism for the online fine-tuning of the algorithm.

Chapter 4

Online Self Learning and Organization

4.1 Introduction

In this Chapter, we propose self-learning and organization structures for the FLCD algorithm which has been proposed in Chapter 3. Although the FLCD algorithm exhibits good performance, it is prone to poor performance in certain network conditions because its optimization process is implemented offline based on a single optimization script. This script can obviously not manage to capture all the traffic dynamics, pattern variations and network topologies. Self-learning and organization structures are therefore necessary in order to enable the FLCD algorithm to fine tune itself in light of traffic variations, unmodelled system dynamics and other external disturbances without disrupting the structure of the optimized membership functions. In order to achieve this, we introduce two concepts: an RTT based sampling mechanism and a self-learning and adaptation mechanism. The RTT based sampling mechanism would enable the FLCD algorithm to adjust its update interval in line with the prevailing link propagation delay. This would help to improve the FLCD algorithm's performance with respect to TCP traffic transmissions which depend on the value of RTT. The self-learning and adaptation mechanism learns the link conditions and adjusts the fuzzy rule base periodically. This mechanism departs from the classical approaches used in [WAN03], [ANN04] by using some concepts learnt from the state-of-the-art algorithms [SBM02], [PRG04] that have been proposed for self-learning and adaptive fuzzy logic controllers. The underlying principles of the self-learning and organization structures are presented in Section 4.2. Simulation results in dynamic traffic environments are presented in Section 4.3. Section 4.4 extends the Self-organized FLCD algorithm to Wireless Local Area

Networks (WLANs) where the problem of congestion exists at the interface between the wired and wireless networks due to the natural differences between these two types of networks. This Chapter is summarized in Section 4.5.

4.2 Self-learning and Organization Structures

4.2.1 RTT Based Queue Sampling Mechanism

The motivation to implement an RTT based queue sampling mechanism stems from the fact that the rate at which TCP injects packets into the network is largely dependent on RTT because TCP is an acknowledgement based end-to-end algorithm. The FLCD algorithm is optimized at sampling rate τ of 0.002s at a bottleneck link propagation delay D of 0.04s.If this sampling rate is used on links with shorter propagation delays, the incoming queue would be undersampled. This situation would lead to higher loss rates due to buffer overflows because the traffic arrival rate is high. On the other hand, if this sampling rate is used on links with longer propagation delays, the incoming queues would be oversampled such that the packet arrival rate would always be very low.The effect of this scenario is that the change in packet marking probability will always be low because the contribution of the packet arrival factor to the fuzzy output value is always low. The system will not be able to increase the packet marking probability in times of congestion such that it will easily degenerate into a drop-tail mechanism with large losses and underutilization. Therefore, the sampling rate is modified based on the link propagation delay by using a linear relationship as follows

$$\tau' = \frac{\tau * D}{0.04} \tag{4.1}$$

where τ' and D denote the sampling rate and the propagation delay for the new link.

The modification of the sampling rate necessitates the adjustment of the maximum packet arrival rate r_m from Chapter 3.The MOPSO optimization process uses a static value of 5.0 for this parameter. In order to cater for dynamic situations while at the same time preserving all the performance characteristics yielded by the optimization process, we use 5.0 as a startup value for r_m.If the weighted average packet arrival rate $\overline{r(t)}$ is greater than r_m, r_m is adjusted by a weighting procedure otherwise it remains unchanged. This process is illustrated in the following

$$r_m(t) = \begin{cases} 5.0 & t = 0 \\ \omega_1 * r_m(t-\tau') + (1-\omega_1) * \overline{r(t)} & \overline{r(t)} > r_m(t-\tau') \\ r_m(t-\tau') & \overline{r(t)} < r_m(t-\tau') \end{cases} \quad (4.2)$$

where ω_1 is the measuring weight just like in Chapter 3.

4.2.2 The Self-Learning and Adaptation Mechanism

This mechanism adjusts the FLCD algorithm in line with the prevailing system conditions. The implementation of online adaptation and self-learning fuzzy systems is an active research area [PM79], [SBM02], [PRF99],[PRG04].The general trend in these systems is that the rule consequents and the membership functions defined in the premises of the fuzzy rules are tuned using various algorithms based on the prevailing plant conditions. For instance the approach in [PRG04] uses two control blocks: the Adaptation Block (A-Block) which is responsible for adapting the consequents of the main controller's rules to minimize the error arising from the plant output, and the Global Learning Block (GL-Block) which compiles real input-output data obtained from the plant. The A-Block is responsible for coarse tuning of the fuzzy rules in the initial stages. As the process advances, the A-Block gives way to the GL-Block which fine-tunes both the membership functions (premises) and the consequents.

In our approach, we use some concepts learnt from [SBM02], [PRG04]. We, however, only fine-tune the rule consequents because of two reasons:

- The membership functions and parameters of the FLCD algorithm have already been optimized offline in Chapter 3. With these membership functions and parameters, optimal performance on all the major AQM objectives is guaranteed. Further tuning of membership functions would disrupt their optimal parameter settings thereby defeating the whole purpose of the proposal in Chapter 3. This is not the case with the proposal in [PRG04] in which there is no model of the plant such that the controller's rules and parameters defining it are optimized from a "void" fuzzy controller.
- It has been reported in [ANN04] that the modification of membership functions uses a lot of memory resources. This process would also take up more of the router's processing time. Therefore, the performance of Internet routers could be adversely affected.

The FLCD self-learning and adaptation mechanism is built on the principle of monotonicity which is evident in [PM79], [SBM02], [PRF99],[PRG04].It evaluates the current state of the plant and proposes the correction of the rules responsible for the existence of such a state, either as a reward or a penalty. In [PRF99], [PRG04], this modification is proportional to the degree with which the rule was activated in achieving the control output $u(t-d)$ now being evaluated at instant t .The system has to wait d iterations in order to evaluate $u(t-d)$ .This calls for the definition of a queue, with the depth given by the delay of the plant, where the degrees of activation of the rules are stored. While such an arrangement works well in [PRF99], [PRG04], it is not suitable for the FLCD algorithm. There are two reasons for this assertion. Firstly, the implementation of a dynamic queue would not only consume the precious memory resources of the router but it would also increase the processing overhead on Internet routers. This must be minimized at all costs because the router's primary function is to route packets. Secondly, the evaluation of plant delay in real time is a complex process because of the dynamics of Internet traffic. In light of these observations, we use the weighted average degrees of activation in the adaptation mechanism. If $F_{AB}(t)$ denotes the adaptation parameter and $\overline{\mu_j(t)}$ denotes the weighted average degree of activation for rule j at instant t , then the proposed change in the output scalar ($b_j(t)$) for rule j would be expressed as follows

$$\Delta b_j(t) = \overline{\mu_j(t)}.F_{AB}(t) \tag{4.3}$$

The weighted average degree of activation for rule j is realized as follows

$$\overline{\mu_j(t)} = \omega_1 * \overline{\mu_j(t-\tau')} + (1-\omega_1) * \mu_j(t) \tag{4.4}$$

In real-time, the system must be stable under different traffic patterns and network topologies. Therefore, as proposed in [ANN04], the variation of queue length must play a role in the adaptation mechanism. The system must also be capable of adjusting itself based on the observed packet losses. It has been pointed out in [WAN03] that packet losses can show the degree of congestion coarsely at least. Therefore, we implement $F_{AB}(t)$ as a sum of the queue error factor $Q_f(t)$ and the packet loss factor $P_f(t)$ i.e. $F_{AB}(t) = Q_f(t) + P_f(t)$ .We let these factors contribute equally to $F_{AB}(t)$ such that

$$Q_f(t) = P_f(t) = \frac{F_{AB}(t)}{2} \tag{4.5}$$

The two parameters $Q_f(t)$ and $P_f(t)$ defining $F_{AB}(t)$ are discussed in the next subsections.

4.2.2.1 Evaluation of the Queue Error Factor

Let $Q(t)$ denote queue length at instant t .The queue variation at instant t with respect to Q_{ref} is expressed as follows

$$\Delta Q(t) = Q(t) - Q_{ref} \tag{4.6}$$

When $\Delta Q(t) > 0$, we know that the level of congestion is increasing hence the need to increase the packet marking/dropping probability by adjusting the rule consequents in the positive direction. As a result, TCP sources will reduce their sending rates while more UDP packets will be dropped. When $\Delta Q(t) < 0$, we know that the level of congestion level is abating hence the need to reduce the packet marking/dropping probability by adjusting the rule consequents in the positive direction. As a result, TCP sources will increase their sending rates while less UDP packets will be dropped thereby increasing the overall utilization of the link. Based on these concepts, the queue error factor $Q_f(t)$ can be expressed as follows

$$Q_f(t) = \begin{cases} C_1 . \dfrac{\Delta Q(t)}{BS} & \Delta Q(t) < 0 \\ C_2 . \dfrac{\Delta Q(t)}{BS} & \Delta Q(t) > 0 \end{cases} \tag{4.7}$$

where BS is the Buffer Size while C_1 and C_2 are constants for negative and positive adjustment respectively.

Constants C_1 and C_2 are directly proportional to the maximum negative and positive variations (ΔP_{neg} and ΔP_{pos}) of the change in packet marking probability respectively. These relationships are presented mathematically as follows

$$C_1 = S_1 . | \Delta P_{neg} | \tag{4.8}$$

$$C_2 = S_2 . \Delta P_{pos} \tag{4.9}$$

where S_1 and S_2 denotes the negative and positive error scaling factors respectively.

If S_1 and S_2 are too small, the resulting $Q_f(t)$ is also very small such that the contribution of the adaptation mechanism is negligible. If S_1 and S_2 are too big, the resulting $Q_f(t)$ would be too big thereby driving the system into instability. By definition, both $\overline{\mu_j(t)}$ and $b_j(t)$ vary within the

range [0.0,1.0] .When $\overline{\mu_j(t)} = 1.0$, $\Delta b_j(t)$ in equation (4.3) becomes equal to $F_{AB}(t)$. For stable operation of the system, we limit the range of variation for $\Delta b_j(t)$ to 8% of the maximum value of $b_j(t)$ .Therefore, the range of variation for $Q_f(t)$ can be limited to 4% because $Q_f(t) = F_{AB}(t)/2$ as given by equation (4.5). This implies that $Q_f(t)$ would fall within the range $[-0.02, 0.02]$ .Typical values for Q_{ref} would be between $0.25BS$ and $0.75BS$ .To cater for extreme cases, we fix Q_{ref} within the interval $(0.0, BS)$. If $Q_{ref} \rightarrow BS$, $\Delta Q(t)$ would fall in the interval $(-BS, 0.0)$ while $Q_f(t)$ is in the range $[-0.02, 0.0]$ .Therefore, for an extreme negative variation, we let $\Delta Q_f(t) = -BS$ and $Q_f(t) = -0.02$ in the negative component of equation (4.7).This yields C_1 as follows

$$C_1 = 0.02 \tag{4.10}$$

The optimization process in Chapter 3 defines -0.0005 as a value for ΔP_{neg}. Substituting $C_1 = 0.02$ and $|\Delta P_{neg}| = 0.0005$ into equation (4.8) yields the negative variation scaling factor as follows

$$S_1 = 40.0 \tag{4.11}$$

If $Q_{ref} \rightarrow 0$, $\Delta Q(t)$ would fall in the interval $(0.0, BS)$ while $Q_f(t)$ is in the range $[0.0, 0.025]$ .Therefore for an extreme positive variation, we let $\Delta Q_f(t) = BS$ and $Q_f(t) = 0.02$ in the positive component of equation (4.7).This yields C_2 as follows

$$C_2 = 0.02 \tag{4.12}$$

The optimization process in Chapter 3 defines 0.0005 as a value for ΔP_{pos}. Substituting $C_1 = 0.025$ and $\Delta P_{pos} = 0.0005$ into equation (4.9) yields

$$S_2 = 40.0 \tag{4.13}$$

The values for S_1 and S_2 are the same in this case because $\Delta P_{pos} = |\Delta P_{neg}|$ but cases would easily arise from the optimization process whereby $\Delta P_{pos} \neq |\Delta P_{neg}|$ .In such cases, the values of S_1 and S_2 would be different.

4.2.2.2 Evaluation of the Packet Loss Factor

In contrast to the queue error factor which is proactive, this factor is a reactive one. It is based on the notion that an increasing number of lost packets entails that congestion is increasing hence the need to increase the packet marking probability. The evaluation of $P_f(t)$ is based on the weighted packet loss rate $\overline{pdr(t)}$ which is evaluated after every τ' seconds in keeping with the RTT based sampling mechanism proposed earlier. This can be expressed as follows

$$\begin{aligned} pdr(t) &= \frac{ndp(t)}{n(t)} \\ \overline{pdr(t)} &= \omega_2 * \overline{pdr(t-\tau')} + (1-\omega_2) * pdr(t) \end{aligned} \tag{4.14}$$

where $ndp(t)$ and $n(t)$ denote the number of dropped packets and the number of arrival packets in the interval $[(t-\tau'),t]$ respectively. $pdr(t)$ denotes the actual packet drop ratio in the interval $[(t-\tau'),t]$ while ω_2 is the measuring weight.

Let pdr_{max} and pdr_{min} denote the maximum packet drop rate and the minimum packet drop rate. When $\overline{pdr(t)} = pdr_{min}$, the rule consequents must remain static because the congestion level is deemed to be within the proper limits. When $\overline{pdr(t)} \rightarrow pdr_{max}$, we know that congestion is becoming more severe hence the need to adjust the rule consequents in the positive direction .This will increase the packet marking/dropping probability and as a result the amount of traffic injected into the network will decrease. Based on these concepts, the packet loss factor $P_f(t)$ can be expressed as follows

$$P_f(t) = C_3 . \frac{\overline{pdr(t)} - pdr_{min}}{pdr_{max} - pdr_{min}} \tag{4.15}$$

Constant C_3 is directly proportional to the maximum positive variations (ΔP_{pos}) of the change in packet marking probability. This relationship is presented mathematically as follows

$$C_3 = S_3 . \Delta P_{pos} \tag{4.16}$$

where S_3 denotes the positive loss scaling factor.

In equation (4.13), $P_f(t)$ falls within the range $[0, C_3]$ because $\overline{pdr(t)}$ is restricted to $\left[pdr_{min}, pdr_{max}\right]$. The range of variation for $\Delta b_j(t)$, and consequently for $F_{AB}(t)$, is limited to 8% of the maximum value of $b_j(t)$ Therefore, the range of variation for $P_f(t)$ can be limited to 4% because $P_f(t) = F_{AB}(t)/2$ (equation (4.5)). This implies that $P_f(t)$ would fall within the range $[0.0, 0.04]$. Therefore

$$C_3 = 0.04 \tag{4.17}$$

Substituting $C_3 = 0.04$ and $\Delta P_{pos} = 0.0005$ into equation (4.16) yields

$$S_3 = 80 \tag{4.16}$$

The self-learning and adaptation architecture and algorithm are shown in Figure 4.1 and Figure 4.2 respectively.

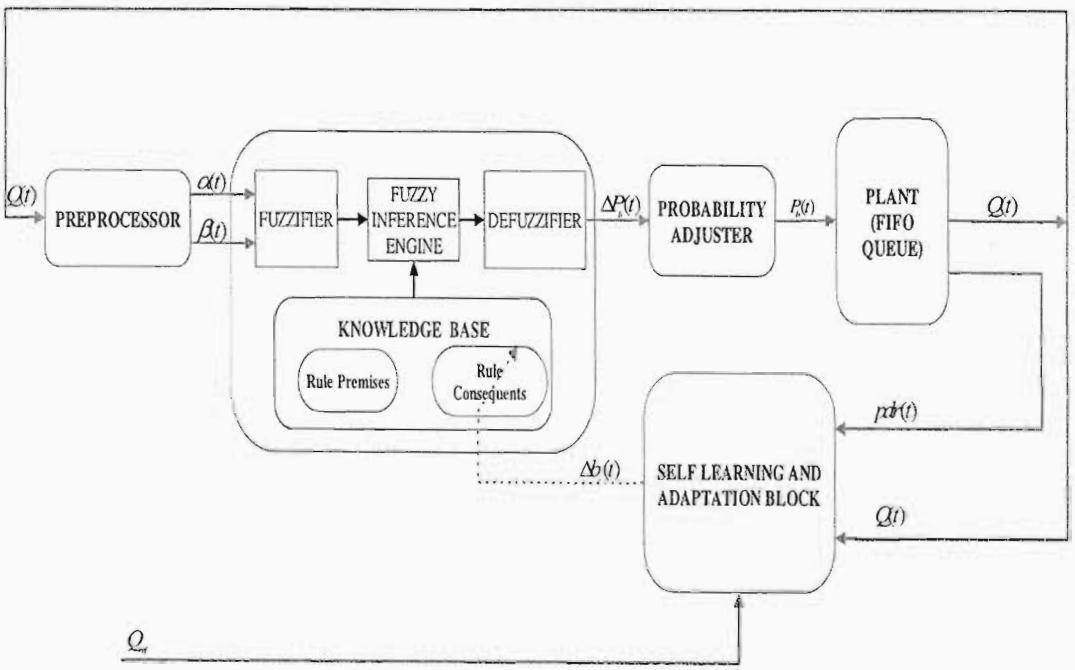


Figure 4.1: Self-learning and Adaptation Architecture

Initialization:

$\tau' \leftarrow \tau * D/0.04$

Every τ' seconds:

Evaluate $Q_f(t)$:

if $(\Delta Q(t) < 0)$ $Q_f(t) \leftarrow C_1.\Delta Q(t)/BS$

else $Q_f(t) \leftarrow C_2.\Delta Q(t)/BS$

Evaluate $P_f(t)$:

$P_f(t) \leftarrow C_3.\left(\overline{pdr(t)} - pdr_{min}\right)/pdr_{max} - pdr_{min}$

Evaluate $F_{AB}(t)$:

$F_{AB}(t) \leftarrow Q_f(t) + P_f(t)$

Evaluate $\Delta b_j(t)$ and update $b_j(t)$ for rules $1, 2, ..., m$:

for $(j = 0; j < m; j++)\{$

$\overline{\mu_j(t)} \leftarrow \omega_1 * \overline{\mu_j(t-\tau')} + (1-\omega_1) * \mu_j(t)$

$\Delta b_j(t) \leftarrow \overline{\mu_j(t)}.F_{AB}(t)$

Generate $b_j(t)$ using the Fuzzy Inference Engine(FIE):

$b_j(t) \leftarrow \text{FIE}(\alpha(t), \beta(t))$

$b_j(t) \leftarrow b_j(t) + \Delta b_j(t)$

$\}$

Evaluate $p_b(t)$:

$prod \leftarrow 0.0; sum \leftarrow 0.0$

for $(j = 0; j < m; j++)\{$

$prod \leftarrow prod + b_j(t) * \mu_j(t)$

$sum \leftarrow sum + \mu_j(t)$

$\}$

$\Delta p_b(t) \leftarrow prod/sum$

$p_b(t) \leftarrow \Delta p_b(t-\tau') + \Delta p_b(t)$

Figure 4.2: Self-learning and Adaptation Algorithm

4.3 Simulation Results in Dynamic IP environments

Two experiments were conducted on the NS-2.8 simulation platform in order to compare the Self-Organized FLCD (Self-Org FLCD) algorithm with the basic Fuzzy AQM [FYX02], the unorganized FLCD and the Adaptive RED (ARED) [FGS01] algorithms. ARED is an adaptive version of the basic RED [FJ93] algorithm against which proposals in [ANN04], [WAN03] are

benchmarked. ARED is more stable than the basic RED algorithm. It has been pointed out in [FGS01] that ARED is capable of restoring the average queue back to the target range within 10 seconds when traffic rate increases by ten times. The basic RED algorithm does not manage to recover the average queue with such a sharp increase in traffic.

We use the following metrics: packet loss rate, link utilization, jitter, delay and link fairness. These metrics are presented in the Appendix A. The reference queue length is set to 40% of the full buffer size in all the three algorithms. All simulations use NS-2.28's *NewReno* TCP variant with an initial congestion window *cwnd* of 3 segments (per [AFP02]), a Maximum Segment Size (MSS) of 1500 bytes and the receiver acknowledging each segment. The full buffer size is set to 90 packets. ECN marking is used. 30 web servers are connected to Router1 with a corresponding number of web clients connected to Router2. We also attach 15 web clients to Router1 and 15 web servers to Router2 to provide background traffic on the return path. Simulations for these experiments are implemented on the network topology in Figure4.3.

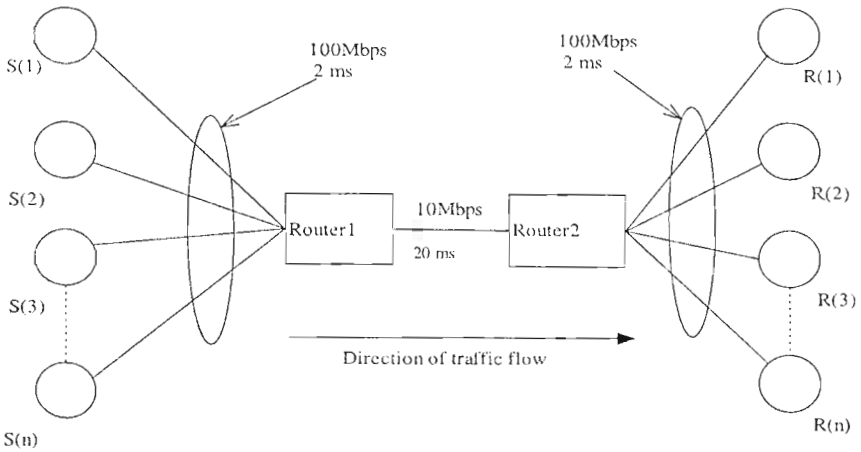


Figure 4.3: Network Topology

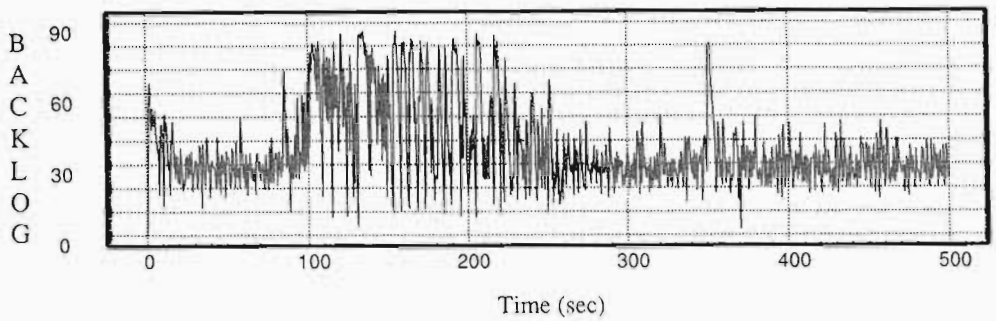
4.3.1 Experiment 1: Queue Evolution in Dynamic Traffic Environments

In this experiment, we compare the sensitivity of the four schemes when flows are introduced and dropped dynamically during a simulation period of 500s. We simulate 50 FTP flows from Router1 to Router2.These flows start randomly within the first 5s and remain active throughout the simulation period. At time=120s, the number of FTP flows from Router1 to Router2 is 50. We increase this number by 50 at each 1 second interval until the simulation time reaches 144

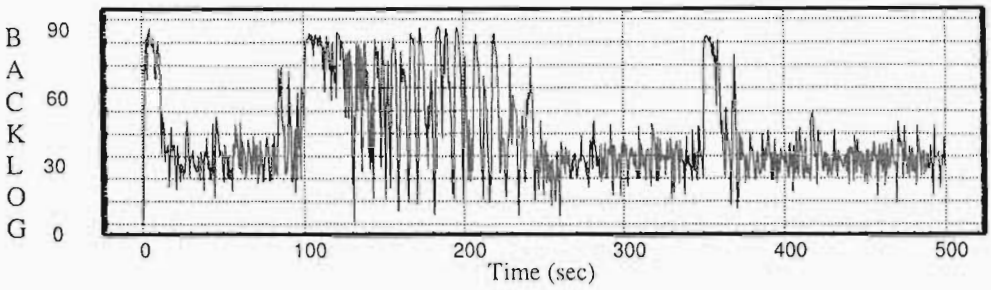
seconds. Between 144seconds and 220 seconds, the number of FTP flows remains constant. When time reaches 220 seconds the number of FTP flows is reduced by 50 at each 1 second interval until time=244 seconds after which the number of FTP flows remains constant. 10 UDP flows from Router1 to Router2 are activated in the following intervals [120s-130s] and [350s-370s].UDP traffic rate is set at 0.5Mbps. We activate 10 web sessions on each client-server connection. Table 4.1 shows the queue evolution statistics for the four schemes. Fig. 4.4 shows the queue length evolution dynamics for the four schemes.

Table 4.1: Queue Length Evolution Statistical results

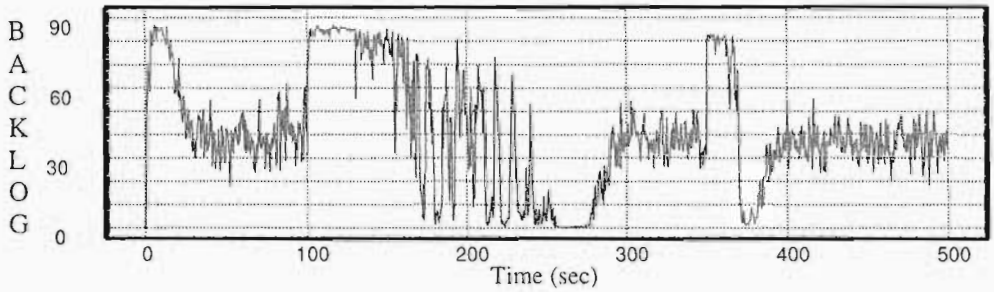
Metric	ARED	Fuzzy(basic)	FLCD	SelfOrg-FLCD
Average Queue Length (Packets)	61.6	40.915	38.814	**36.74934**
Queue variance (Packets)	363.348	596.7366	425.872	**319.50838**



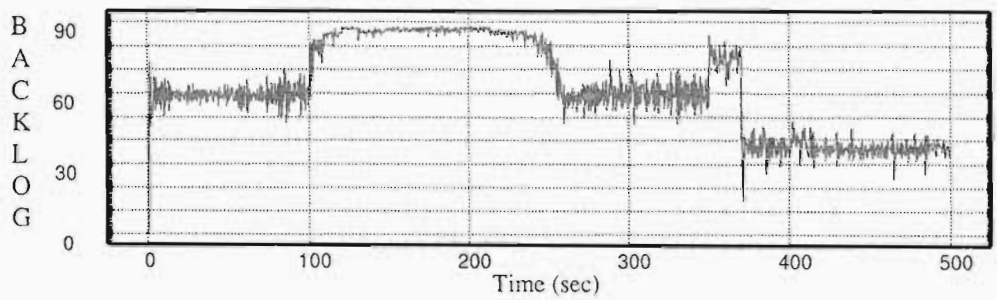
(a) SelfOrg FLCD



(b) FLCD



(c) Basic Fuzzy



(d) Adaptive RED

Figure 4.4: Queue Evolution for the four schemes

Table 4.1 and Figure 4.4 show that the Self organized FLCD algorithm is more stable than the other approaches. It really attempts to limit the length of the queue to 36 packets (40% of full buffer size). The unorganized FLCD algorithm ranks second while the basic Fuzzy algorithm and the ARED algorithm rank third and fourth respectively. Right from the onset, even without the introduction of dynamic traffic, the ARED queue stabilizes at a much higher value. The TCP traffic inflow during slow-start completely overwhelms it such that it fails to recover the queue length to the desired target. The three other approaches also register high queue length immediately after startup but they manage to recover and maintain the queue within the precincts of the target. Of the three well performing approaches, the self organized FLCD approach registers the shortest queue length during the startup phase (approximately 60 packets) while the unorganized FLCD algorithm (approximately 82 packets) and the basic Fuzzy algorithm (approximately 85 packets) rank second and third respectively. In terms of recovery time during the startup phase, the self organized FLCD algorithm still ranks first with a recovery time of approximately 5seconds while the unorganized FLCD algorithm (approximately 10 seconds) and the basic Fuzzy algorithm (approximately 15 seconds) rank second and third respectively. When UDP traffic is introduced in intervals [120s-130s] and [350s-370s], the queue's high period is smallest in the self organized FLCD algorithm compared to the other approaches. With the introduction of dynamic TCP traffic, all the algorithms except ARED which just shifts the queue even higher close to the buffer limit, become unstable as they try to limit the queue length to the set target. Once again the self organized FLCD algorithm performs better than the unorganized FLCD algorithm and the basic Fuzzy algorithm in that it attempts to bring the queue down to 36 packets right from the time dynamic TCP traffic starts entering the link. The self organized FLCD algorithm also exhibits good recovery performance when the dynamic TCP traffic stops flowing. The basic fuzzy algorithm suffers severe underutilization when dynamic TCP traffic stops flowing. After approximately 390 seconds up to 500 seconds, all the four schemes limit the queue length to approximately 36 packets. It is worthy pointing out that ARED is more stable during this period but that advantage is offset by its very high average queue length. The effect of long queues is twofold. Besides enhancing the need for larger buffers long queues have an effect of increasing packet delays.

4.3.2 Experiment 2: Dynamic Traffic Environments with varying propagation delays

In this experiment, we compare the performance of the three schemes when the round trip delay of the bottleneck link is varied. We use the network topology shown in Figure 4.3.We vary the round trip link delay by using 20ms, 40ms, 60s up to 180 ms.The simulations run for 200 seconds. We simulate 50 FTP flows, competing for bottleneck link from Router1 to Router2. These flows start within the first 5 seconds. We activate 4 web sessions on each client-server connection. 10 UDP flows are introduced in the following intervals [50s-60s] and [150s-160s] while the FTP flows start randomly within the first 5s and they run up to the end of the simulation. At time 60s, 200 new FTP flows start, with 40 starting every 7.5 seconds. When time reaches 140 seconds, the new FTP flows are removed from the traffic mix in steps of 40 flows every 7.5 seconds. Figure 4.5- Figure 4.9 show the results.

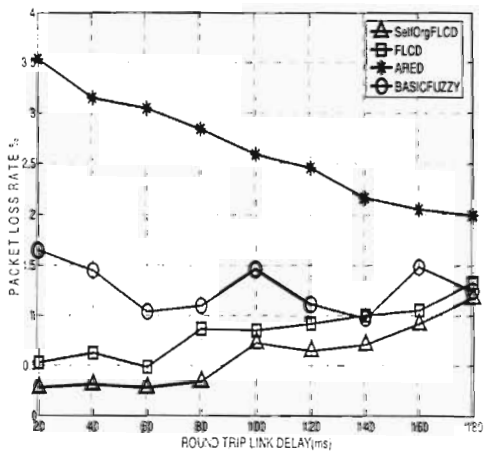


Figure 4.5: Loss rate with varying Round trip link delay

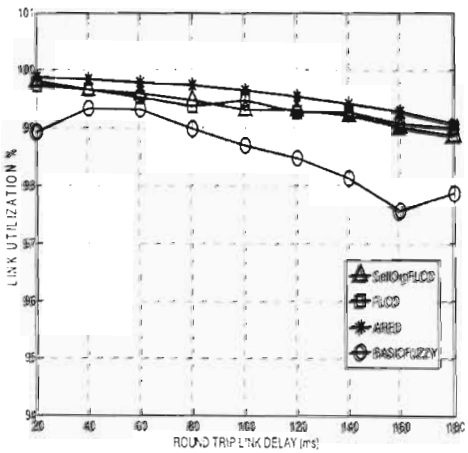


Figure 4.6: Link Utilization with varying Round trip Link delay

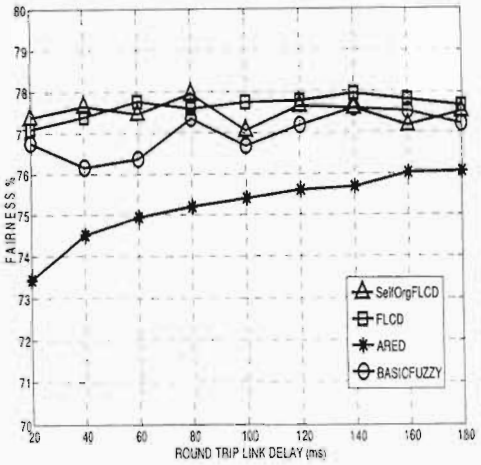


Figure 4.7: Fairness with varying Round trip link delay

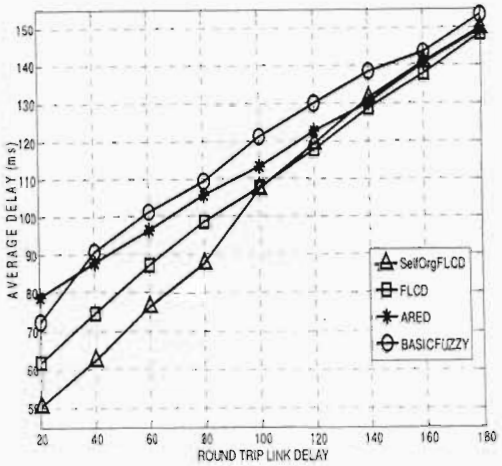


Figure 4.8: UDP traffic delay with varying round trip link delay

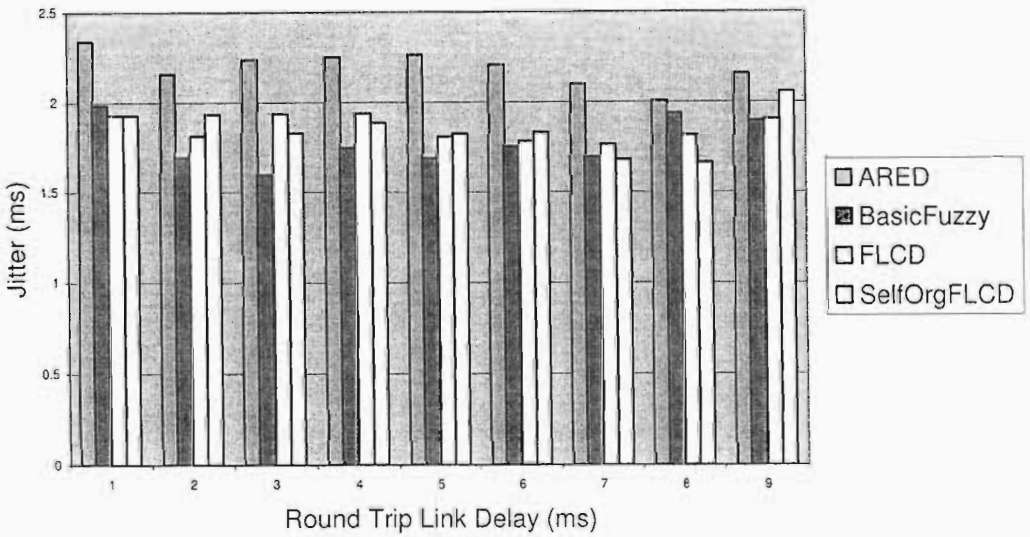


Figure 4.9: UDP Traffic Jitter with varying Round trip Link Delay

Figure 4.5 shows that the Self organized FLCD algorithm has the lowest packet loss rate ranging from 0.28% to 1.181%.The organized FLCD algorithm comes second with a loss rate ranging from 0.532% to 1.324%. The basic Fuzzy algorithm comes third with a loss rate ranging from 0.972% to 1.647%. The ARED algorithm comes fourth with a loss rate ranging from 1.995% to 3.533%.The self organized FLCD algorithm's low packet loss is due to role of the packet loss factor which is embedded in the self learning and adaptation mechanism. This factor helps to

increase the packet marking probability when packet losses have been detected. This helps to reduce further packet losses. ARED's poor packet loss performance is due to the fact that the ARED control law fails to keep the queue within the desired precincts. When dynamic traffic sets in, buffer overflows are inevitable as seen in Fig.4.4.

Fig.4.6 shows that ARED and both FLCD approaches achieve high link utilization throughout the simulation run. The average link utilization values are as follows: 99.34% for self organized FLCD, 99.36% for unorganized FLCD, 99.57% for ARED, 98.58% for basic Fuzzy. The basic Fuzzy algorithm exhibits lower link utilization because it suffers from severe underutilization soon after the dynamic TCP traffic stops flowing. It fails to recover the queue to the target after such a sharp decrease in traffic (See Figure 4.4).

Figure 4.7 shows that both FLCD approaches achieve the highest average link fairness (77.6% for FLCD and 77.5% for self organized FLCD). The basic Fuzzy algorithm follows them closely with an average of 76.98% while ARED comes last with an average of 75.2%.From this, we observe that the self organized FLCD algorithm does not jeopardize the fairness element of the FLCD algorithm.

Figure 4.8 shows that for round trip link delays shorter than 100ms, self organized FLCD algorithm achieves the lowest UDP packet delay (76.8ms). The FLCD algorithm (86ms) comes second. The ARED (98ms) algorithm and the basic Fuzzy algorithm delays (98.93ms on average) exhibit similar UDP traffic delays. However, when the round trip delay exceeds 100seconds, the FLCD algorithms and ARED exhibit similar UDP traffic delay performance while the basic Fuzzy algorithm exhibits slightly longer delays. The self organized FLCD algorithm exhibits better UDP delay performance for shorter round trip time because of the RTT based update mechanism which forms part of the self learning and organization structure. This mechanism enables the FLCD algorithm to frequently update the packet marking probability for links with shorter RTTs. The effect of this is that buffer overflows are minimized. It becomes easier for the FLCD algorithm to keep the queue close to its target thereby improving the end-to-ed delay.

Figure 4.9 shows that the basic Fuzzy algorithm exhibits the lowest jitter (with an average of 1.78ms). The FLCD schemes rank second with averages of 1.854ms for FLCD and 1.847seconds for self organized FLCD algorithm. The ARED algorithm comes last with an average jitter of 2.19ms.

4.4 Self-Organized FLCD algorithm in Wireless Local Area Networks

4.4.1 The Need for Congestion Control in WLANs

IEEE 802.11 wireless LANs (WLANs) have gained strong popularity in a number of vertical markets, health-care, retail, manufacturing, warehousing and academic institutions. In contrast to traditional wired networks, WLANs are characterized by mobility support, installation simplicity and flexibility, reduced cost-of-ownership and scalability. The widespread deployment of WLANs is due to the productivity, convenience and cost advantages offered by these characteristics. Despite the growing popularity, the available bandwidth in IEEE802.11 networks is much smaller than in wired local area networks since IEEE 802.11 networks are non-switched half duplex. These networks suffer from interference from radio sources like microwave ovens, cordless phones and wireless computer devices such as Bluetooth. They also suffer from the hidden node problem which results in collisions and reduced channel performance [SCH00]. The maximum peak transmission rate possible in 802.11a/g stations is 54Mbps. However studies [KS03] have shown that, due to a large overhead per frame transmission, the maximum efficiency is only 50-60%. In addition, maximum channel throughput can only be achieved in close proximity to the Access point (AP). As distance from the Access point increases, throughput diminishes more or less rapidly due to effects of packet loss, reflection, diffraction and scattering. The actual channel throughput also heavily depends on the frame payload size. When only frames as are typical for VoIP are sent, the maximum throughput on the wireless channel on the wireless channel can drop below 1Mbps even at a data rate of 11 Mbps [KS03].

At present, an IEEE working group (IEEE802.11n) is working on a new standard which builds on the previous 802.11 standards by adding MIMO (Multiple-Input Multi-Output) concepts in order to offer wireless transmissions at rates greater than 100Mbps. It is projected that 802.11n will also offer a better operating distance than current 802.11 networks. While the ultimate advantage of this standard is obviously speed, it is reported in [COX03] that just like its preceding standards, maximum throughput can only be achieved in close proximity to the Access Point (AP). As distance increases from the Access Point (AP), throughput still diminishes more or less rapidly. It further reported in [COX03] that in almost every case today, an Access Point is a shared medium: whatever throughput it can deliver is divided up among the users that connect to that one access point. This implies that for 10 users sharing one Access Point, the throughput per user translates to 10 Mbps. This new standard was designed in order to match the 100Mbps switched Ethernet

capacity in the Distribution System (DS). However, new standards offering higher data rates are arising in switched Ethernet .While 1-Gigabit Ethernet is now widely available and 10-Gigabit products are becoming more available, the IEEE and the 10-Gigabit Ethernet Alliance [CISCO] are working on 40, 100, or even 160Gbps standards. Therefore, the disparity in channel capacity between the wired and the wireless interfaces of the Access Point will continue to present a significant bottleneck in the downstream direction. The incoming link will continue to be oversubscribed resulting in frequent buffer overflow. As a result, the implementation of a congestion control scheme in either the Gateway or the Access Point will always remain a viable solution for avoiding congestion and ensuring that delays for packets with real-time constraints are minimized in IEEE 802.11 WLANs. In [YI04], a proxy-RED congestion control scheme is proposed for WLANs. The main idea of this scheme is to reduce overhead at the access point by implementing the AQM functionality at the gateway. The instantaneous queue length at the access point is sampled periodically to calculate the estimated average queue length, which is used for determining the drop/marking probability at the gateway. Results [YI04] show that this congestion control scheme significantly improves packet loss rate and goodput for a small buffer, and delay for a large buffer. This approach is based on the WLAN architecture in Figure 4.10. Each Access Point (AP) connects its associated Basic Service Set (BSS) to the gateway. The gateway provides Internet connectivity to the APs.

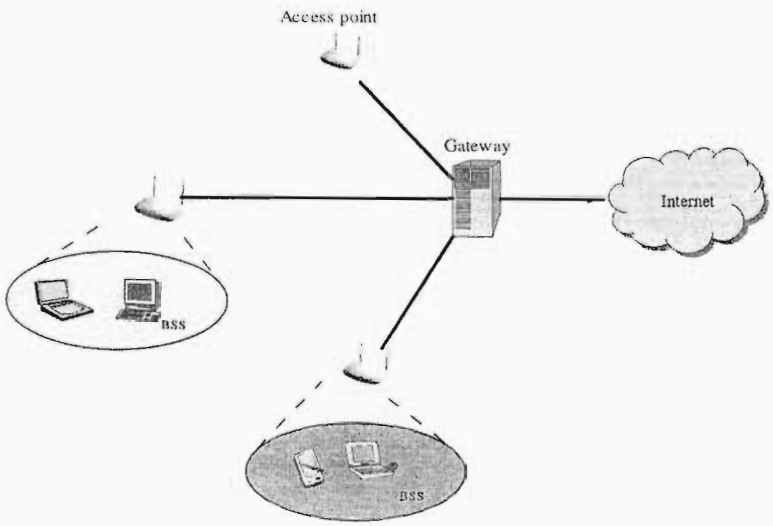


Figure 4.10: Gateway centric WLAN Architecture

However, WLAN architectural trends [ZR05] show that the WLAN architecture in Figure 4.10 is not highly scalable because the cost of deploying a large scale WLAN network dramatically

increases as the network expands. An emerging architecture which presents a possible solution to the scalability problem is the (static) multi-hop.11 (*mesh*) network (See Figure 4.11).

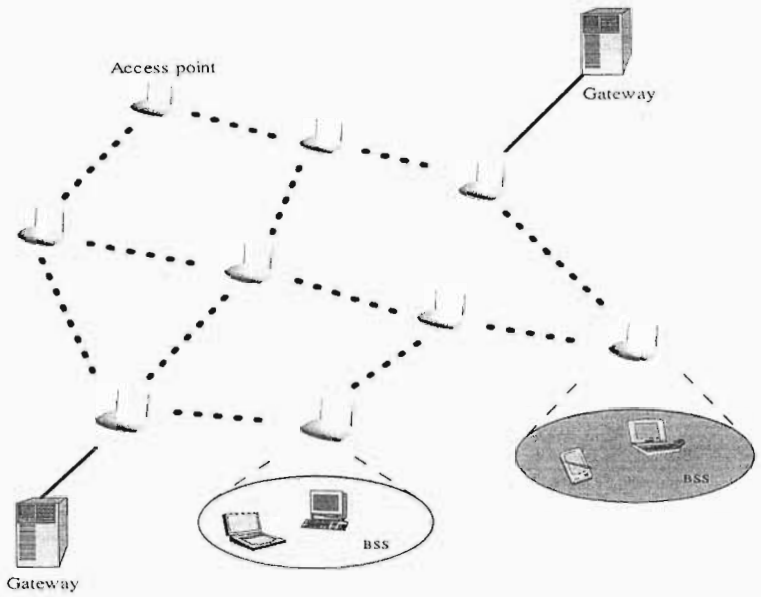


Figure 4.11: Wireless AP-to-AP Mesh Network

Interest in this approach is indicated not only by the newly formed Mesh Task Group within IEEE 802.11 but also mesh solutions offered by several companies [MESHD, MESHN]. The deployment of a proxy-based (gateway-centric) congestion control scheme in the future *wireless AP-AP mesh network* is not a scalable approach because there will be numerous APs without direct connection to the gateway. Therefore, congestion control schemes will still have to be implemented in the access point because the access point has direct control to a particular Base Service Set (BSS).

4.4.2 Simulation Model and Results

To investigate the performance of the self-organized FLCD algorithm in WLAN, we run some simulations on the network topology shown in Figure 4.12. Servers S1, S2, S3 and S4 are connected to the Gateway which is connected to the Access Point. The bandwidth and propagation delay between the servers and the Gateway and between Gateway and the Access Point are set to 100Mbps and 2ms respectively. The Access Point feeds two fixed nodes (FN1 and FN2) and two mobile nodes (MN1 and MN2)) in 36Mbps WLAN architecture. We compare the

performance of the self-organized FLCD algorithm with that of the basic Fuzzy logic AQM, Adaptive RED and the Drop-tail mechanism which is used in WLAN networks at present. These schemes are configured in the Access Point at the WLAN interface.

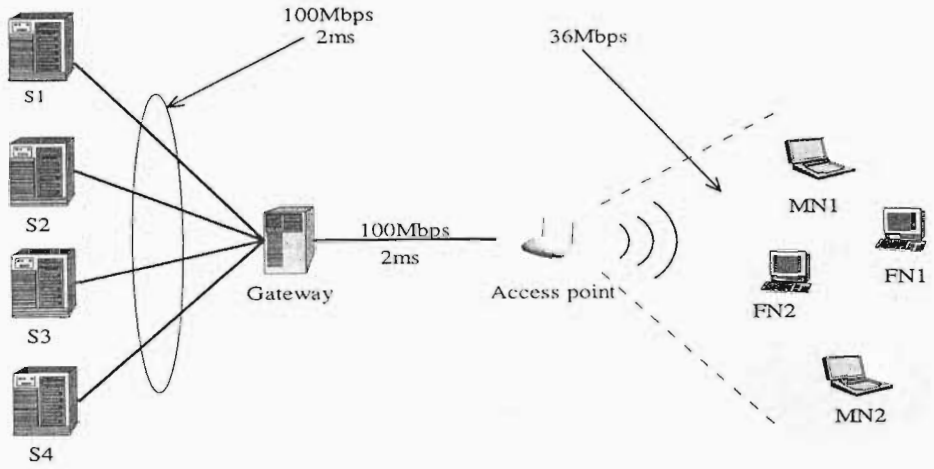


Figure 4.12: WLAN Simulation Topology

We vary the buffer size by using 50,100, 150 up to 400 packets. The simulations run for 250 seconds. 48 FTP flows and 4 web traffic flows, each having 4 sessions, are configured to flow from the servers to all the WLAN nodes throughout the simulation period. To provide background traffic on the return path., we configure 4 web traffic flows, each having 4 sessions, to flow from the WLAN nodes to the servers. Audio, video and basic UDP traffic was generated based on Table 4.2.

Table 4.2: Real Time Traffic Simulation parameters

	Audio	Video	Background
Transport Protocol	UDP	UDP	UDP
Packet Size	160 bytes	1280 bytes	1500 bytes
Packet Interval	20 ms	10 ms	12.5 ms
Flow Rate	8 Kbytes/s	128 Kbytes/s	120 Kbytes/s

Video traffic flows from Servers S1 and S2 to the mobile nodes in the interval [40s-70s]. Audio traffic flows from Servers S1 and S2 to the mobile nodes in the interval [130s-160s]. Basic UDP traffic flows from all the 4 Servers to all the WLAN nodes in the intervals [25s-30s] and [215s-220s]. Figure 4.13- Figure 4.16 show the results.

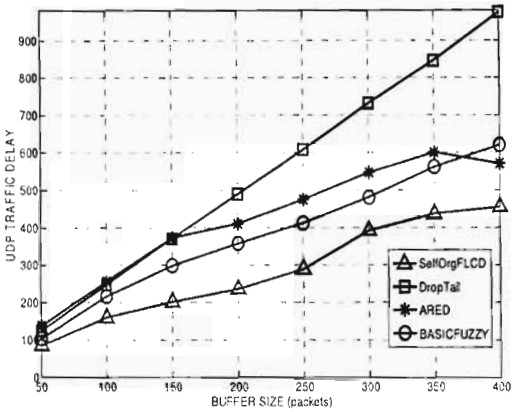


Figure 4.13: UDP Traffic Delay with varying Buffer Size

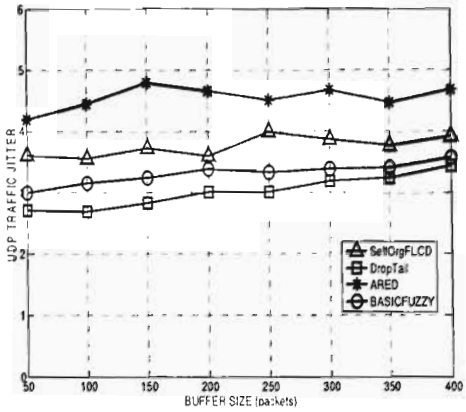


Figure 4.14: UDP Traffic Jitter with varying Buffer Size

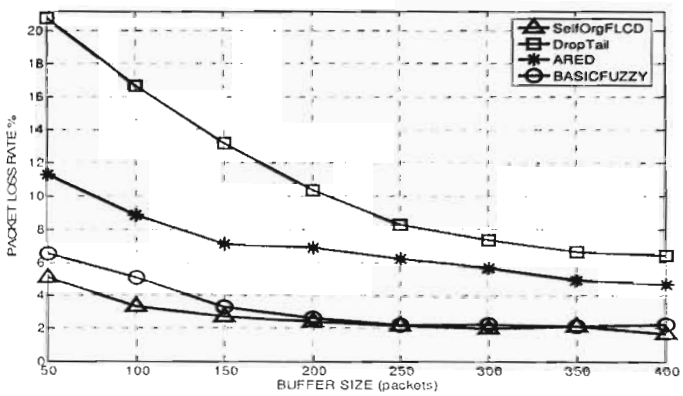


Figure 4.15: Packet Loss rate with varying Buffer Size

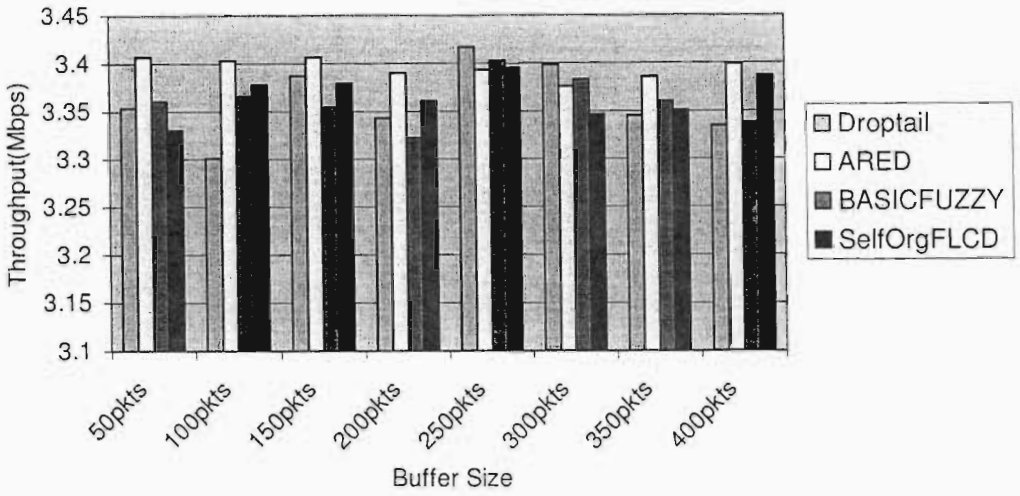


Figure 4.16: Throughput with varying Buffer Size

Figure 4.13 shows that the Self organized FLCD algorithm exhibits the lowest UDP traffic delay ranging from 85.7ms (for a buffer of 50 packets) to 455.7ms (for a buffer of 400 packets).The basic Fuzzy AQM algorithm comes second with the UDP traffic delay ranging from 104.8ms (for a buffer of 50 packets) to 621ms (for a buffer of 400 packets). ARED comes third with UDP traffic delay ranging from 137.76ms to 571.34ms (for a buffer of 400 packets).The Drop-tail mechanism comes fourth with the UDP traffic delay ranging from 125.47ms (for a buffer of 50 packets) to 974.6ms (for a buffer of 400 packets).In terms of averages, the UDP traffic delay performance is as follows: 282.587ms for the Self organized FLCD, 381.727ms for basic Fuzzy AQM, 421.4ms for ARED and 548ms for Drop-tail. The Self organized FLCD algorithm achieves the lowest delay because it maintains the shortest queue amongst the four mechanisms. The Drop-tail mechanism exhibits the highest delay because its queue is perpetually long because it does not employ any special mechanism to limit it. More UDP packets would be discarded at the receivers in the Drop-tail mechanism due to late arrival because of the high link delay. We also observe that in all cases, UDP traffic delay increases with varying proportions as buffer size increases. This happens because queuing delay increases as buffer size increases.

Figure 4.14 shows that the Drop-tail algorithm exhibits the best UDP traffic jitter with an average value of 3.01ms.The basic Fuzzy AQM algorithm comes second with an average value of 3.31ms while the Self organized FLCD algorithm comes third with an average value of 3.75ms.ARED comes fourth with an average value of 4.547ms The optimization results in Chapter 3 reveal that

jitter and delay are non-commensurate i.e. they cannot be improved at the same time. This is proven again in this case because the Drop-tail mechanism exhibits the lowest UDP traffic jitter at the expense of UDP traffic delay while the Self organized FLCD algorithm exhibits the shortest UDP traffic delay at the expense of UDP traffic jitter. A closer look at the jitter performance shows that jitter values for the three mechanisms are very close such that the performance of the Self organized FLCD algorithm in terms of jitter would not be very far from the other two approaches.

Figure 4.15 shows that the Self organized FLCD algorithm exhibits the lowest packet loss rate for small buffer sizes. When the buffer size increases, its loss rate is almost similar to that of the basic Fuzzy algorithm. On average, the FLCD algorithm exhibits an average loss rate of 2.687%.The basic Fuzzy algorithm , which ranks second, has average loss rate of 3.31% .RED ranks third with a loss rate of 6.98% while the Drop-tail mechanism ranks fourth with an average of 11.226%. The Drop-tail mechanism performs poorly because it does not detect incipient congestion as a result it fails to minimize packet loss rates. The basic Fuzzy AQM algorithm exhibits higher packet losses for smaller buffers because it maintains larger queues which easily overflow.

Figure 4.16 shows that all the four schemes exhibit low average throughput rates. The average values are as follows: 3.395Mbps for ARED, 3.366Mbps for the Self organized FLCD algorithm, 3.361Mbps for Drop-tail and 3.36Mbps for the basic Fuzzy AQM algorithm. These values account for less than 10% of the available bandwidth i.e.36Mbps. From these results, we observe that the introduction of an AQM in WLAN environments does not improve throughput significantly.

4.5 Chapter Summary

This Chapter has proposed online self organization structures for the Fuzzy Logic Congestion Detection (FLCD) algorithm. These structures include an RTT based sampling mechanism and a self-learning and adaptation mechanism. The latter modifies the algorithm's update interval in line with the prevailing outgoing link propagation delay while the former fine-tunes the algorithm according the prevailing system conditions. The effectiveness of the proposed approach is proved by comparing the performance of the self organized FLCD algorithm with that of the unorganized

FLCD, the Adaptive RED and the basic Fuzzy algorithms under dynamic traffic patterns. Performance results show that the proposed approach achieves a much more stable queue compared to the other approaches. Apart from enhancing the stability of the FLCD algorithm the new approach also reduces UDP traffic delay for short round trip propagation delays. It also reduces the FLCD algorithm's loss rate. We have also observed that the addition of the self-organization structures to the FLCD algorithm does not jeopardize other performance metrics like utilization, jitter and fairness.

The final part of this Chapter extended the concept of Fuzzy logic congestion detection to WLAN networks. In this implementation, the FLCD algorithm helps to minimize UDP traffic delay, packet loss rates. In terms of throughput, the FLCD algorithm exhibits similar performance to the other schemes. Its UDP jitter is slightly higher than its basic variant and the Drop-tail mechanism but better than that of ARED.

Chapter 5

The Dual Explicit Congestion Notification Mechanism

5.1 Introduction

Chapter 1 of this thesis described the explicit congestion related notification mechanisms that have been proposed for IP networks. These mechanisms have been broadly classified into two groups: congestion notification and underutilization notification mechanisms. Chapter 1 also explains the merits and demerits of these mechanisms. Of all these mechanisms, the Explicit Congestion Notification (ECN), an IETF standard for congestion notification in IP networks, is the most widely used. Research efforts [LJ01], [SDJ04] are still taking place in order to improve ECN.

This thesis does not implement underutilization notification mechanisms because the method for estimating link utilization in literature [SH02] is more accurate in this aspect. The mechanism in [SH02] estimates utilization based on the microflow status of the output link. Apart from the periodic utilization updates, it also makes utilization updates whenever a packet leaves the queue. Any link utilization estimation algorithm based on the packet marking/dropping probability (from the FLCD in this case) cannot achieve that level of accuracy because its utilization information is determined after specific periods depending on the rate at which queue length and packet arrival rate are sampled. This thesis, therefore, only focuses on the congestion notification mechanisms i.e. ECN and BECN.

This Chapter begins by presenting an overview on related work in this area in Section 5.2. An algorithm, that combines the ECN and BECN mechanisms based on the output of the Fuzzy Logic Congestion Detection (FLCD) algorithm which was presented in Chapter Three, is then developed in Section 5.3. An RTT based BECN reduction mechanism is also proposed in the same section. Section 5.4 presents the performance evaluation of the proposed models in both wired and satellite networks. This Chapter is summarized in Section 5.5.

5.2 Related Work

5.2.1 Wired Networks

Although congestion control research efforts in conventional wired IP networks have been taking place for more than two decades, explicit congestion notification has only been introduced recently [RF01]. The resurgence of BECN is also provoking more research efforts from the Internet community. Notable research works for wired networks in literature are as follows: The ECN mark-front strategy [LJ01] and the Combined ECN/BECN approach [AKU03].

5.2.1.1 The ECN Mark-front strategy

Liu and Jain [LJ01] improve the ECN mechanism by introducing the mark-front strategy. Instead of marking a packet from the tail of the queue, this strategy marks the packet in front of the queue and thus delivers faster congestion notification signals to the source. With a simplified model, they analyze the buffer size requirement for both the mark-front and mark-tail strategies. They tested link efficiency, fairness and more complicated scenarios using simulations based on the RED. Results [LJ01] show that the mark-front strategy provides a better congestion control that helps TCP to achieve smaller buffer size requirement, higher link efficiency and better fairness among users. Based on these results, the recent AQM schemes [ALLY01, HMTG01] employ the ECN mark-front strategy.

5.2.1.2 The Combined BECN/ECN Approach

Akujobi *et al.* [AKU03] proposed the first algorithm to combine the ECN and BECN mechanisms using RED as the congestion detection algorithm. This mechanism invokes ECN for low to medium level congestion and BECN for medium to high level congestion. A new threshold, *becnthresh* is introduced between RED's minimum (*minth)* and maximum (*maxth)* threshold, in order to identify the point at which BECN is triggered. ECN is invoked when the average queue length is between *minth* and *becnthresh* while BECN is invoked when the average queue length is between *becnthresh* and *maxth*. Results show that this mechanism combines the merits of BECN and ECN algorithms effectively and leads to improved performance compared to individual ECN and BECN schemes. It also results in significant reduction in ISQ reverse traffic compared to the BECN mechanism. The study of the impact of different *becnthresh* settings and the development of its preferred values has been pointed out as an area of future research in [AKU03].

5.2.2 Satellite Networks

Satellite networks play an indispensable role in the deployment of global communication networks because they offer global coverage, broadcast capabilities, flexibility in bandwidth allocation, support for mobility and ease of deployment in areas of low subscriber density and with little infrastructure. Based on these characteristics, satellite networks are suited for the provision of broadband internet *access* to remote locations, as well high speed *backbone* networks. Currently, satellite architectures are classified as Geostationary Orbit (GSO) and Non-geostationary Orbit (NGSO). The NGSO architecture comprises the Medium earth orbit (MEO) and the Low earth orbit (LEO) [DKG01], [KDJ01]. In the GSO architecture, satellites are positioned on equatorial orbit. Their altitude is approximately 35,800km above the surface of the earth. The satellites appear to an observer on earth as being stationary. In the NGSO architecture, satellites operate in orbits much closer to the earth. These satellites change their position relative to ground position quickly.

The provisioning of quality-of-service (QoS) presents the greatest obstacle to the further development of satellite network systems. GSO systems are characterized by large delays due to their high altitude while NGSO systems are characterized by large delay variations because they are not stationary. These characteristics seriously affect TCP performance because TCP is an acknowledgement and time-out-based congestion control mechanism. Its performance is

inherently related to the delay-bandwidth product of the connection. TCP round-trip time estimations are also sensitive to delay variations, which may cause false time-outs and transmissions. As a result, congestion control issues in satellite networks are more complex than those of lower-latency terrestrial networks [DSL01a], [DSL01b], [GD99], [HEN99].

The other obstacle to good TCP performance over satellite networks is its non-negligible bit-error rates (BER) because of transmission link errors. TCP provides reliable byte-streams to applications by ensuring that the sender retransmits corrupted data. However, packet loss is also used by TCP to determine the level of congestion in the network because traditionally, in wired networks, the bulk of packet loss comes from congestion. TCP responds to congestion by reducing its congestion window (*cwnd*) and therefore its sending rate. The reduction of *cwnd* when packet loss has been caused by channel errors [KSE04] will lead to network underutilization. Therefore, several solutions have been proposed to distinguish network congestion effects from corruption effects so that the TCP sender decreases its *cwnd* only when there is congestion. Explicit loss notifications (ELN) [BK98], [SAM99] explicitly notify the TCP senders about packet losses due to channel errors while explicit congestion notifications (ECN) [FLO94], [SA00], [AKU02] explicitly notify the senders by marking, instead of dropping, packets when the link is oversubscribed. In satellite networks, the need to minimize packet losses is even greater because network bandwidth is expensive and the packets to be dropped will have already traversed through a series of precious links. The dropped packets will also exacerbate the consequences of the already high latency and latency variations. Notable research works on explicit congestion notification for satellite networks are as follows: The ECN mark-front strategy [DSL01a] and the Multilevel ECN strategy [DSL01b].

5.2.2.1 The ECN Mark-front strategy

Durresi *et al.* [DSL01a] extend the mark-front strategy [LJ01] to satellite networks. They achieved similar results to those in [LJ01].The effect of their results had much more impact on satellite networks than on wired networks. In satellite networks, the reduction of buffer size requirements for a condition without losses translates into less complex devices and less delay. By sending faster feedback signaling about congestion, the source adjusts its *cwnd* in time to avoid packet loss and link idling, consequently improving the link efficiency. This result is important because in satellite networks, link bandwidth is more expensive and less available than in terrestrial networks.

5.2.2.2 The Multilevel ECN Strategy

Durresi *et al.* [DSL01b] implement a Multi-level ECN (MECN) mechanism in a satellite network scenario. This mechanism uses the CE and the ECT bits to implement three different congestion levels. The CE-ECT bit combination '01'-indicates no congestion, '10' indicates incipient congestion and '11' indicates moderate congestion. Packet drop occurs if there is severe congestion in the router and buffers overflow. The RED scheme is modified to include another threshold called *midth* between *minth* and *maxth.* If the size of the average queue is between *minth* and *midth*, there is incipient congestion and the CE, ECT bits are marked as '10' with a maximum probability of P1max. If the average queue is between *midth* and *maxth*, there is moderate congestion and the CE, ECT bits are marked as '11' with the maximum probability P2max.If the average queue is above *maxth* all packets are dropped. The receivers reflect the bit marking in the IP header, through TCP ACKs by using a combination of 2 bits 8, 9 (CWR, ECE) in the reserved field of the TCP header. The response of the TCP senders is defined as follows. When there is no congestion, the *cwnd* is allowed to grow additively as usual. When there is packet drop, the *cwnd* is decreased multiplicatively by $\alpha_3 = 50\%$. When the marking is '10' (incipient congestion), *cwnd* is decreased multiplicatively by $\alpha_1 = 20\%$ .When the marking is '11' (moderate congestion) *cwnd* is decreased multiplicatively by $\alpha_2 = 40\%$ .This enables TCP senders to have a better tuned response to congestion. MECN converges faster, with fewer losses than simple ECN and improves other QoS parameters such as link utilization and delay.

5.3 The proposed Dual explicit Congestion Notification Algorithm

This algorithm is developed based on the motivation drawn from the work in [AKU03] and the superior performance of the FLCD algorithm. ECN and BECN are invoked based on the level of congestion as depicted by the packet marking probability p_b from the FLCD algorithm. Figure 5.1 shows the FLCD architecture with the BECN mechanism incorporated.

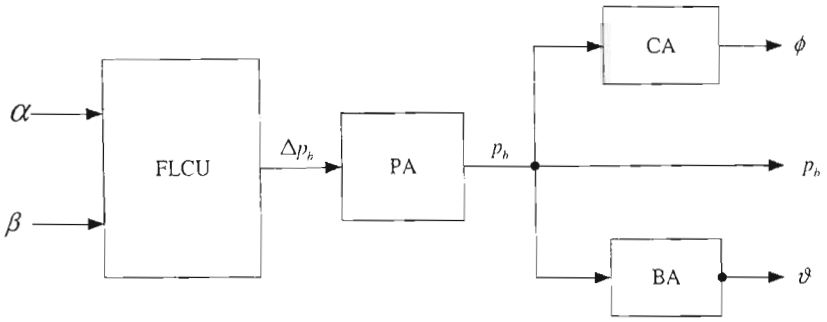


Figure 5.1: FLCD Architecture with BECN Activation Mechanism

The BECN factor $\vartheta \in [0,1]$, generated by the BECN Activator, is directly proportional to the packet marking probability p_b .If K denotes the BECN constant then the BECN factor is determined as follows

$$\vartheta = K * p_b \tag{5.1}$$

ECN and BECN are invoked probabilistically based on the values of p_b and ϑ respectively. ECN marking is invoked in the deque routine in order to reduce the delay incurred in delivery of congestion information to the sender. BECN marking, which triggers the flow of reverse ICMP source quench messages, is invoked in the enque routine in order to inform the sender about incipient congestion as soon as it is detected. In order to ensure reliable delivery of the congestion notification signal, a corresponding packet traversing the link in the forward path is ECN marked every time BECN is invoked. If the ISQ message fails to reach the sender then the ECN Echo in the next ACK message will trigger the sender to reduce its congestion window. When $K = 1$, the system is almost 100% BECN dependent. We say almost 100% because of the unreliability of ISQ messages since they are not guided by acknowledgements such that ECN will still play a role in the notification mechanism (especially in lossy environments). When $K = 0$, the system is 100% ECN dependent. No BECN ISQ message is generated under this condition because ϑ is equal to 0. In this proposal, we set K to 0.5.

5.3.1 Dual Explicit Congestion Notification with Reverse Traffic Reduction

Chapter 1 of this thesis pointed out that the generation of ICMP packets must be minimized at all costs partly because it oversubscribes the router in terms of processing overhead and partly because it increases the amount of traffic in the reverse path. We argue that it is possible to reduce the generation and transmission of reverse ICMP traffic without seriously affecting the

performance of the system. This is due to the fact that the reception of multiple congestion notification signals, in succession, at the sending TCP does not lead to multiple reductions in the sending rate as TCP reduces its sending rate only once each roundtrip time (RTT) [RF01], [AKU03].Therefore the implementation of the reverse traffic reduction mechanism based on the value of RTT will not only reduce the effects of ISQ generation on router performance but will also minimize the flow of control traffic in the reverse path. We develop this mechanism by first of all estimating the value of RTT.

5.3.1.1 RTT Estimation

We use the method used in [FKGS01] in order to estimate RTT. Let L denote the delay on the outgoing link, C denote the outgoing link capacity in packets/second and K_1 a constant. RTT is derived as follows

$$RTT = K_1 * (L + 1/C) \tag{5.2}$$

5.3.1.2 BECN Decay Function

A BECN decay function helps to ensure that the amount of ISQ generation is reduced exponentially during an RTT. A *BECN timer* is used for the timing of this function. This timer is reset immediately after the fuzzy congestion detection algorithm becomes operational. A time threshold $t_{thresh} = 0.1 * RTT$ is maintained in the router. The decay factor D is therefore determined as follows

$$D = \begin{cases} 1 & for \quad t < t_{thresh} \\ e^{-(t - t_{thresh})/(a * RTT)} & for \quad t_{thresh} \le t < RTT \end{cases} \tag{5.3}$$

where parameter a is the weighting factor.

When time $t < t_{thresh}$, the number of ISQ traffic flows is not restricted. This is done in order to ensure that all the sources, responsible for the present congestion status, get the BECN ISQ message. When time $t > t_{thresh}$, the number of ICMP messages deteriorates according to the Decay function in equation (5.3).When time $t > RTT$, the *BECN timer* is reset i.e. $t = 0$ sec. At

this point, the number of ISQ flows is not restricted until t reaches t_{thresh}. When $t > t_{thresh}$, the BECN decay function is activated again. The process is repeated and so on.

5.3.2 Behaviour of an FLCD based Dual Explicit Congestion Notification Router

An FLCD based dual explicit congestion notification router essentially behaves as a pure ECN router under low congestion. Under heavier congestion, both ECN and BECN mechanisms are used for congestion notification. The computational flow of the BECN ISQ generation and marking process in the enque routine is as follows

```
Generate a random number R ∈ [0,1.0];
if (buffer is full){
   Drop the incoming packet;
   if ( ECT bit is set in IP header){ // Checking if the sender is ECN or BECN capable
     // Activate BECN with ISQ reduction mechanism
     if (BECN Decay Function is enabled){
         if (R < D) Send ISQ due to a dropped packet back to the sender;
       }
     // Activate BECN without ISQ reduction mechanism
     else Send ISQ due to a dropped packet back to the sender;
    }
  }
else if (ECT bit is set){
   // Activate BECN with ISQ reduction mechanism
   if (BECN Decay Function is enabled){
       if (R < D * ϑ) Mark packet (CE bit) and send ISQ
         due to a marked packet back to the sender;
     }
   else{  // Activate BECN without ISQ reduction mechanism
       if (R < ϑ) Mark packet (CE bit) and send ISQ
          due to a marked packet back to the sender;
     }
  }
else if ((R < p_b) and ECT bit is not set) Drop the incoming packet;
```

Figure 5.2: BECN Packet Marking Algorithm

The computational flow of ECN marking process in the deque routine is as follows

```
Generate a random number R ∈ [0,1.0];
if ((R < p_b) and ECT bit is set) and if (packet is not already marked) {
        Mark packet (CE bit);
  }
```

Figure 5.3: ECN Packet Marking Algorithm

5.3.3 Behaviour of an ECN+BECN-Capable TCP end host

The ECN+BECN TCP end hosts perform initial end-to-end negotiations to establish ECN capability just like pure ECN hosts. The TCP sender responds to either ECN or BECN (depending on whichever signal arrives first). Whichever notification reaches the sender first should cause the window reduction. When the sender receives an ECN Echo-ACK or an ISQ message, it reduces its congestion window and the slow start threshold to one-half of the current window. If the ISQ is due to a marked packet, the sender waits a full RTT after window reduction before it starts increasing its window. If the ISQ is due to a dropped packet, the sender follows the TCP congestion control algorithm immediately after reducing the window. The sender does not respond to congestion signals more than once in an RTT [AKU03].

5.4 Performance Evaluation

In this section, we compare the performance of the proposed dual explicit congestion notification algorithm with some of the congestion notification schemes described in section 5.2. In order to show the effect of the ISQ reduction mechanism, the proposed notification algorithm has been implemented in two modes i.e. with and without ISQ reduction mechanisms. The simulations considered two topologies: wired and satellite networks. The FTP traffic model in the Network Simulator is used with an infinite amount of traffic to send. TCP type is New Reno with a data packet size of 1000 bytes and ACK packet size of 40 bytes. The TCP clock granularity is set to 100ms. Delayed ACKs are not used in these simulations. Packet-based marking in all cases. For RED, the maximum probability for ECN marking is set to 1.0. For the FLCD BECN reduction

approach, we set the weighting factor a to 0.75 .The buffer size is set to 90 packets in all cases. The following performance metrics are used:

- Bottleneck link utilization: This refers to the number of data packets respectively that successfully traverse the bottleneck link and are received by the receiver.
- Queue Length: This is the average of queue length samples that are recorded every 0.5seconds as the simulation runs.
- Percentage loss: This measures the ratio of packets dropped at the bottleneck link to the total number of packets injected into the bottleneck link for a particular flow or set of flows
- Percentage ISQ reverse traffic: This is computed as the number of BECN ISQs generated in the reverse direction of data flow as a ratio of the total number of packets received from that direction.
- TCP Goodput: This is computed as the total amount of TCP traffic that traverses the network as a ratio of the total capacity of the network.

5.4.1 Wired Network

For the wired network, we use the simulation topology in Figure 5.4.

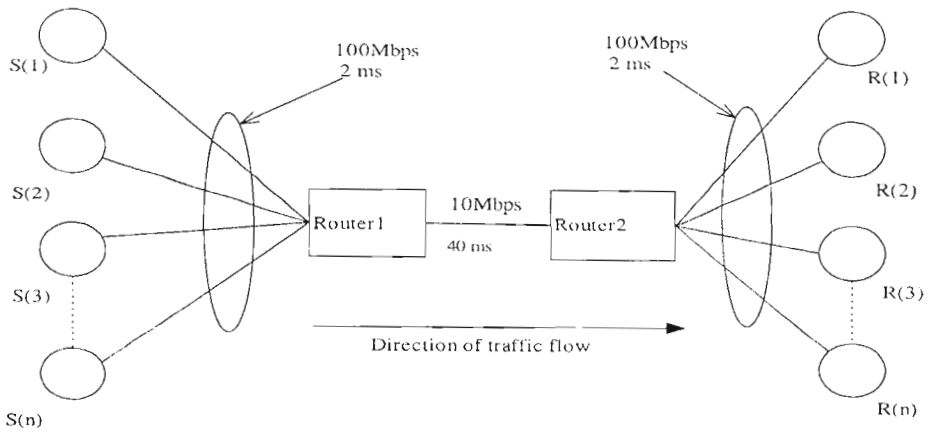


Figure 5.4: Wired Network Simulation Topology

We compare the performance of the two versions of the proposed algorithm (with and without ISQ reduction) with the combined ECN/BECN approach [AKU03].The bottleneck bandwidth is 10Mbps with a propagation delay of 40ms. All other links have 100Mbps with a propagation

delay of 2ms.Traffic flow is from sources , S(1)...S(n) through routers, Router1 and Router2 to receivers, R(1)...R(n). We follow the guidelines used in [AKU03] in setting RED queue parameters as follows: *minth = 15KB, maxth = 3*minth, buffer size = 2*maxth, becnthresh = 30KB, maxp =0.1, wq=0.002, mean packet size=1000bytes*. Traffic configuration is done as follows.30 web servers are connected to Router1 with a corresponding number of web clients connected to Router2. We also attach 15 web clients to Router1 and 15 web servers to Router2 to provide background traffic on the return path. We activate 8 web sessions on each client-server connection. The number of FTP Traffic flows from Router1 to Router2 is varied by using 10, 20, 30,40,50,60,70,80,90 and 100 flows in order to establish different levels of congestion. The FTP flows start randomly within the initial 5s of the simulation while the web-traffic connections start within the first 10s.For the FLCD BECN reduction approach, we set K_1 to 3 just as in [FKGS01]. Packet-based dropping for RED is used. This is done basically for comparison purposes because our scheme also employs packet-based dropping. These simulations run for 150s.Figure 5.5–Figure 5.8 show the results.

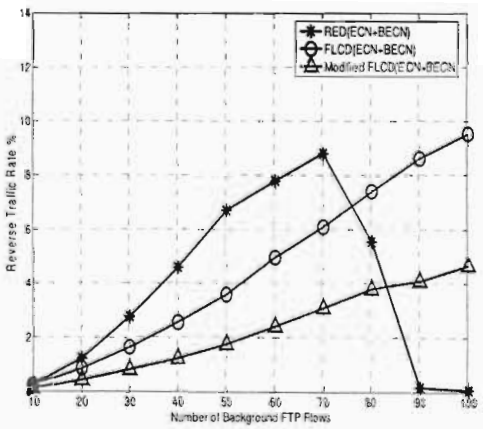


Figure 5.5: ISQ Reverse traffic

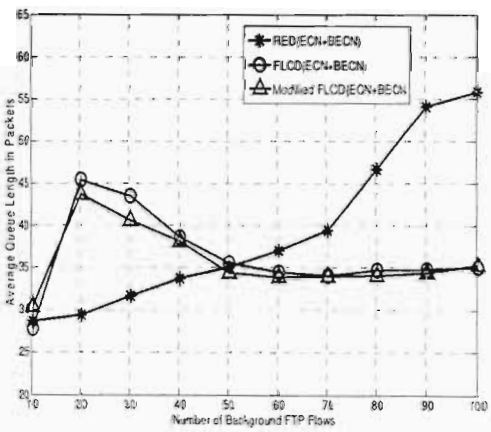


Figure 5.6: Queue Length

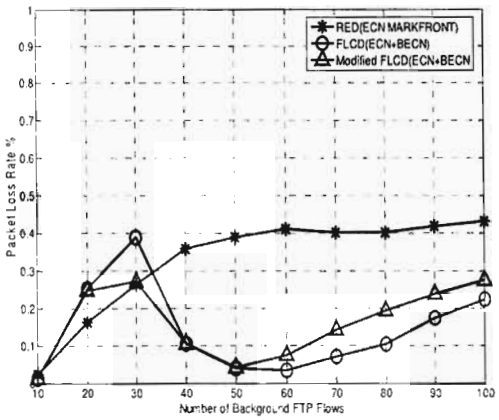


Figure 5.7: Packet Loss Rate

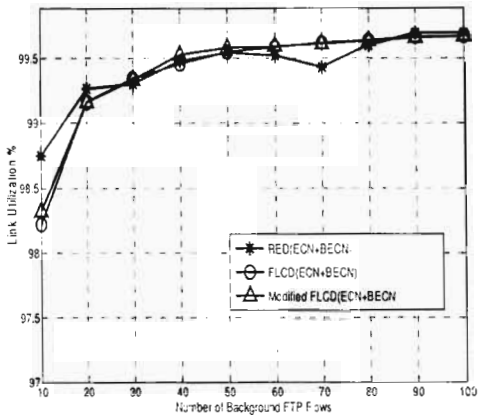


Figure 5.8: Bottleneck Link Utilization

Figure 5.5 shows that the RED (ECN+BECN) approach exhibits the highest ISQ reverse traffic when the number of FTP flows is 70 or less. This happens because the length of the queue is predominantly between *becnthresh* (30packets) and *maxth*(45packets) during this period (See Figure 5.6).As a result more packets are BECN marked leading to the generation of ISQ reverse traffic. After 70 FTP flows, the ISQ reverse traffic decreases drastically because the length of the queue is predominantly above *maxth* (45 packets). During this period ECN is the predominant congestion notification mechanism. The ISQ reverse traffic generated at this stage is mainly due to buffer overflows. It is worthy pointing out that the ISQ reverse traffic decreases drastically at a time when it was supposed to be increasing because this is the stage (100 FTP flows) when the network faces real congestion as opposed to the stage when the number of FTP flows is 70.This just illustrates the difficulty pointed out in [AKU03] pertaining to the placement of the BECN threshold.

From Figure 5.5, we also observe that the amount of reverse traffic in the basic FLCD algorithm is directly proportional to the number of FTP flows. This happens because the packet marking probability increases at a rate that is directly proportional to the number of FTP flows traversing the link because the amount of web traffic is constant. Therefore, the rate of reverse traffic, which is a function of the BECN factor, also increases as the number of FTP flows increase. In the case of the modified FLCD algorithm, the amount of reverse traffic is also directly proportional to the number of FTP flows but its gradient is approximately half of the basic FLCD algorithm.

On the overall, the average percentages of ISQ reverse traffic are as follows: 3.8% for RED (ECN+BECN), 4.6% for FLCD (ECN+BECN) and 2.24% for Modified FLCD (ECN+BECN).

We note that the modified FLCD algorithm achieves the lowest ISQ reverse traffic .Compared to basic FLCD algorithm, it reduces the ISQ reverse traffic by more than 50%.

Figure 5.6 shows the FLCD approaches exhibiting similar performances in terms of queue length. They register their longest queue length (approximately 45packets for the basic FLCD and 44 packets for the modified FLCD algorithm) when the number of FTP flows is 20. The long queue lengths when the number of FTP flows is low can be attributed to the high proportion of bursty web traffic flows in the traffic mix coupled with the short round trip time. The other reason is that at this stage, the level of congestion is low such that the main mechanism of congestion notification is ECN which is slow thereby leading to longer queues. Beyond 50 FTP flows, the FLCD queues stabilize to 34 packets up to the end of the simulation. The RED approach exhibits shorter queue lengths when the number of FTP flows is low because its control law is entirely queue length based such that it is insensitive to variations in the arrival rate of the bursty web traffic flows. The other reason for this behaviour relates to the fact that RED enjoys the benefits of BECN when the congestion level is low as shown in Figure 5.5. Since BECN is faster than ECN, RED achieves shorter queues when congestion levels are low. This advantage is however short-lived because the RED approach fails to control the queue as the number of FTP flows increases. It reaches 56 for 100 FTP flows. At this point the amount of BECN traffic is very low. Figure 5.7 shows the RED approach exhibiting the highest average packet loss rate (0.327%) while the FLCD approaches exhibit similar performance (0.1463% for FLCD and 0.1608% for modified FLCD). Besides the fact that the RED algorithm is inferior to the FLCD algorithms, this trend can also be attributed to the fact that the BECN marking mechanism in the RED approach is misplaced as pointed out earlier. Most of the losses take place as the number of FTP flows increase. Unfortunately, that is the time when the BECN marking mechanism is withdrawn.

From the result in Figure 5.7, we also observe that the FLCD algorithms achieve virtually the same packet loss rates even though the ISQ reverse traffic in modified FLCD algorithm is reduced by more than 50%.This confirms our assertion that most of the ISQ reverse packets are wasted because TCP responds to ISQs once every round trip time.

Figure 5.8 shows the three mechanisms exhibiting equally high levels of average link utilization. The RED approach however registers slightly higher link utilization when the number of FTP flows is 10.This happens because at this stage the RED queue is predominantly below *minth* such that most packets are not marked. This results into more packets being injected into the bottleneck

link. The FLCD approaches tend to be more aggressive in marking packets at this stage because they are rate based as well as queue based. Once the arrival rate increases, even without any meaningful queue change, the packet marking probability is adjusted upwards. As a result, they exhibit slightly lower link utilization at this stage.

We also note the RED approach exhibits a slight slump at about 70 FTP flows. This happens because the algorithm switches from the BECN notification back to the ECN notification mechanism as the queue length becomes predominantly above *maxth*. The curves for the FLCD approaches are smooth because of the fuzziness employed in the invocation of the two congestion notification mechanisms. Therefore, the FLCD approaches are more stable.

5.4.2 Satellite Network

For the satellite network, we use the simulation topology in Figure 5.9. We compare the performance of the two versions of the proposed algorithm (with and without ISQ reduction) with the combined ECN Mark front strategy [DSL01a].

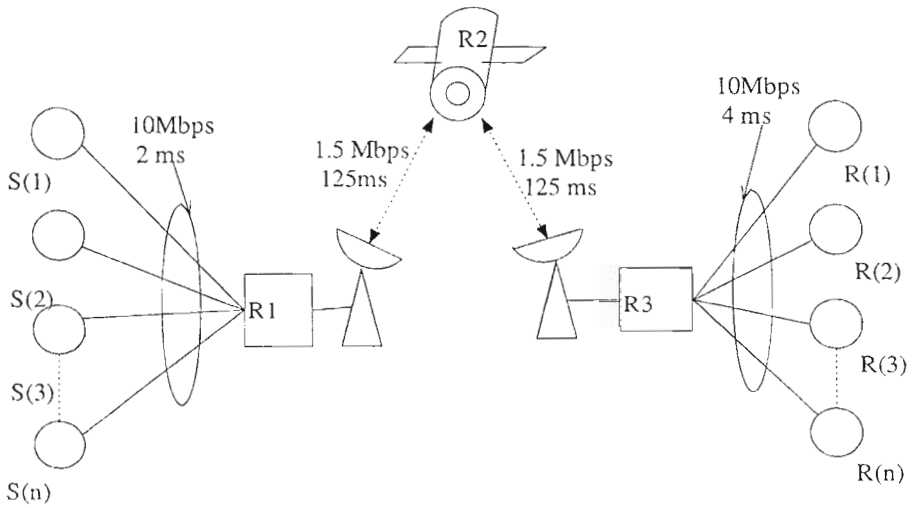


Figure 5.9: Satellite Network Simulation Topology

Traffic sources S(1)...S(n) are connected to Router R1 through 10 Mbps, 2ms delay links. Router R1 is connected to R2 through a 1.5 mbps, 125ms delay link. R2 is connected to R3 through a 1.5 mbps, 125ms delay link. The overall round trip link delay is 500ms which is the typical for GSO satellite networks [HL01]. A number of receivers R(1)...R(n) are connected to R3 through 10

Mbps 4ms delay links while a number of traffic sources are connected to R1 through 10 Mbps 2ms delay links. Link speeds are chosen so that congestion will only happen between R1 and R2 where our scheme is tested. Traffic configuration is done as follows.30 web servers are connected to Router1 with a corresponding number of web clients connected to Router3. We also attach 15 web clients to Router1 and 15 web servers to Router3 to provide background traffic on the return path. We activate 8 web sessions on each client-server connection. The number of FTP Traffic flows from Router1 to Router3 is varied by using 10, 20, 30,40,50,60,70,80,90 and 100 flows in order to establish different levels of congestion. The FTP flows start randomly within the initial 5s of the simulation while the web-traffic connections start within the first 10s. The target queue length is set to 35% of the buffer size in all the three cases. For the FLCD BECN reduction approach, we set K_1 to 6 because in satellite networks there are two links with the same delay between the source and the destination. The simulation period is 200s for all runs. Figure 5.10–Figure 5.13 show the results.

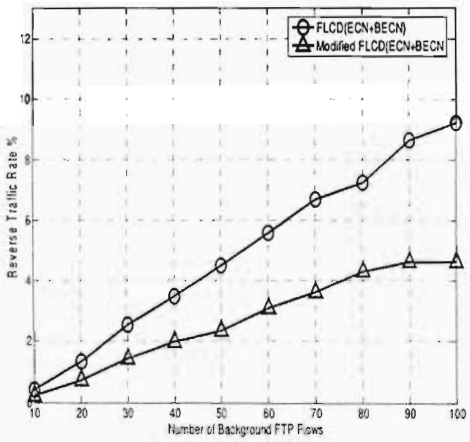


Figure 5.10: ISQ Reverse traffic

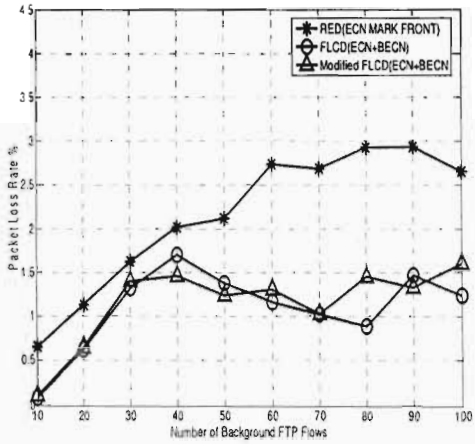


Figure 5.11: Packet loss rate

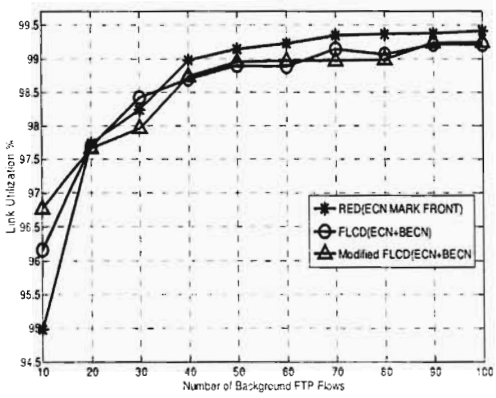


Figure 5.12: Bottleneck ink Utilization

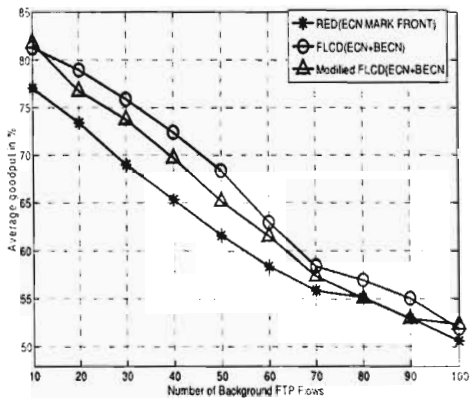


Figure 5.13: Goodput

Figure 5.10 shows that the amount of reverse traffic in the basic FLCD algorithm is directly proportional to the number of FTP flows. This happens because the packet marking probability increases at a rate that is directly proportional to the number of FTP flows traversing the link because the amount of web traffic is constant. Therefore, the rate of reverse traffic, which is a function of the BECN factor also increases as the number of FTP flows increases. In the case of the modified FLCD algorithm, the amount of reverse traffic is also directly proportional to the number of FTP flows but only up to 80 FTP flows after which the reverse traffic rate saturates to about 4.62%. For the period when the amount of reverse traffic is directly proportional to the number of FTP flows, the rate of increase is approximately 0.43% for every 10 FTP flows introduced as opposed to 0.9% for a similar number of FTP flows in the basic FLCD algorithm. The overall average reverse traffic rates are 4.96% and 2.7% for the basic and the modified algorithms respectively. This implies that the modified algorithm reduces the amount of reverse traffic by 45.46%.

Figure 5.11 shows that the FLCD algorithms exhibit lower packet loss rate than the RED/ECN mark front algorithm. The average packet loss rates are 1.09% and 1.16% for the basic and the modified algorithms respectively. The average loss rate for RED/ECN mark front algorithm is 2.15%.The FLCD approaches register lower loss rate because they benefit from early congestion notifications through the BECN mechanism. We also observe that the modified FLCD algorithm maintains a competitive packet loss rate although its amount of reverse ISQ traffic is reduced by 45.46%. This confirms the fact that most of the reverse ISQ traffic in the original FLCD

algorithm is wasted because the TCP sender responds to a congestion notification message only once in an RTT.

Figure 5.12 shows that although the level of link utilization is very high in all the three cases the RED/ECN mark front strategy registers the lowest level of link utilization (94.99%) when the number of FTP flows is 10 while the fuzzy approaches register higher levels of utilization at this stage i.e. 96.771% for the modified case and 96.165% for the basic case. When the number of FTP flows is low, the proportion of web traffic is high. The burstiness of web traffic coupled with the long delay associated with ECN causes a series of backlog variations on the link which lead to underutilization in the RED/ECN approach. The situation is different with the FLCD approaches because they enjoy a fair share of BECN messages in the reverse path. This results in fast delivery of congestion information and as a result the link does not experience severe delay based backlog variations thereby ensuring high link utilization. As the number of FTP flows increases, the RED/ECN algorithm registers a slightly higher level of link utilization than the fuzzy approaches because long lived FTP flows, which are more stable, begin to dominate.

In Figure 5.13, the FLCD algorithms register higher average goodput than the RED/ECN mark front algorithm. The average goodput values are 66.23% and 64.66% for the basic and the modified algorithms respectively. The RED/ECN algorithm registers an average goodput value of 61.95%. This shows that although the RED/ECN algorithm registers high bottleneck link utilization, the actual amount of useful data relayed to the receiver is actually low because of retransmissions which happen as a result of the higher loss rates.

5.5 Chapter Summary

In this chapter, we have proposed a fuzzy logic based dual explicit congestion notification mechanism based on the output of the FLCD algorithm. We have also proposed an RTT based decay function which helps to reduce the amount of ICMP reverse traffic by taking advantage of the fact that TCP responds to congestion signals only once during an RTT. We have compared the performance of the two FLCD (BECN+ECN) approaches with the RED (ECN+ BECN) on wired networks and with the RED (Mark Front Strategy) on satellite networks. Results show that the FLCD approaches exhibit better performance in terms of packet loss rate, queue stability on wired networks. On satellite networks, they exhibit better packet loss rates, goodput and link

utilization. We have also observed that the RTT based decay function helps to reduce the amount of reverse traffic by more than 50% for wired networks and by more than 40% for satellite networks while ensuring that performance remains virtually the same as in the original algorithm and much better than the RED based approaches.

Chapter 6

Conclusions and Future Work

6.1 Summary

This dissertation has tackled two aspects of IP congestion control: router-based congestion detection and explicit notification mechanisms. The first part of this research focused on the development of a Fuzzy Logic Congestion Detection (FLCD) mechanism which achieves optimal performance on all the major performance metrics of Internet congestion control. The second part focused on the development of an FLCD based dual explicit congestion notification mechanism which combines the merits of Explicit Congestion Notification (ECN) and the Backward Explicit Congestion Notification (BECN) mechanisms.

In Chapter 1, we have outlined the origins of congestion on the Internet. We have categorized Internet congestion control research into three interrelated fronts: *end-to-end mechanisms*, *router based detection mechanisms* and *explicit notification mechanisms*. We have described the key research developments on each front. We have also explained the reasons behind the emergence of fuzzy logic based AQM schemes. In this Chapter, we have also presented the motivation of the research done and the original contributions of this dissertation.

In Chapter 2, we studied the principles of operation, the efficiencies and deficiencies of the key traditional AQM schemes. We also presented the fuzzy logic control theory. We also studied the fuzzy AQM schemes in terms of their design principles, their efficiencies and deficiencies. From these studies, we observed that fuzzy control rules and membership functions are obtained through a manual tuning process which is based on the designer's insight. The human factor involved in this operation makes it difficult for these algorithms to achieve optimum performance for all the key AQM objectives. We also observed that these algorithms are generally designed with an assumption that the Internet is predominantly composed of TCP traffic, whose sources

respond to congestion notification signals from routers by reducing their sending rates. We also observed that practically, the situation is not like that because apart from the non-responsive UDP traffic which accounts for a reasonable traffic proportion, the Internet is nowadays facing a growing list of non-responsive flows which should also be taken into account.

In Chapter 3, we have developed the Fuzzy Logic Congestion Detection (FLCD) algorithm. The CHOKe algorithm is incorporated in the FLCD architecture in order to address the issue of fairness which was not addressed explicitly in all the preceding fuzzy logic AQM approaches. We then modeled the congestion control problem as a multi-objective (MO) problem and used Multi-Objective Particle Swarm Optimization (MOPSO) in designing the membership functions for the Fuzzy Logic Congestion Detection algorithm. The optimization process was based on four objective functions. These objective functions were derived based on the following requirements: maximizing link utilization, minimizing loss rate, minimizing link delay and jitter. In the best effort implementation, the performance of the proposed approach was compared with the basic Fuzzy algorithm and the REM algorithm. From performance results so far obtained, we observed that the FLCD algorithm exhibits highest link utilization and fairness. It also exhibits the lowest packet loss rates and UDP traffic jitter. Its performance in terms of UDP traffic delay is similar to REM and the basic Fuzzy algorithm. We extended the FLCD algorithm to PropDiffServ IP networks where its performance was compared with that of the WRED algorithm. From performance results so far obtained, we observed that the PropDiffServ FLCD approach achieves higher link utilization, lower packet loss rate, jitter and delay.

In Chapter 4, we proposed self organization structures in order to enable the FLCD algorithm to learn the system conditions and fine-tune itself accordingly thereby achieving optimal performance in dynamic traffic environments and a wide range of topologies. These structures include an RTT based sampling mechanism and a self-learning and adaptation mechanism. The former modifies the algorithm's update interval in line with the prevailing outgoing link propagation delay while the latter fine-tunes the algorithm according the prevailing system conditions. The performance of the self-organized FLCD algorithm is compared with that of the unorganized FLCD, the Adaptive RED and the basic Fuzzy algorithms under dynamic traffic patterns. From performance results so far obtained, we observed that the self organized FLCD algorithm achieves a much more stable queue compared to the other approaches. Apart from enhancing the stability of the FLCD algorithm the self-organization structures also reduce UDP

traffic delay for short round trip propagation delays. They also reduce the FLCD algorithm's loss rate. We also observed that the addition of the self-organization structures to the FLCD algorithm does not jeopardize other performance metrics like utilization, jitter and fairness. In the final part of this Chapter, we extended the FLCD concept to WLAN networks. From performance results so far obtained, we observed that the FLCD algorithm minimizes UDP traffic delay, packet loss rates. It also maintains a stable throughput for all buffer sizes. Its UDP jitter is slightly higher than its basic variant and the Drop-tail mechanism.

In Chapter 5, we proposed a dual explicit congestion notification mechanism based on the output of the FLCD algorithm. This mechanism combines the BECN and the ECN protocols in order to combine their merits. We also proposed an RTT based decay function which helps to reduce the amount of ICMP reverse traffic by taking advantage of the fact that TCP responds to congestion signals only once during an RTT. We compared the performance of the two FLCD approaches (with and without reverse traffic reduction) with the RED (ECN+ BECN) on wireline networks and with the RED (Mark Front Strategy) on satellite networks. From performance results so far obtained, we observed that the FLCD approaches exhibit better performance in terms of packet loss rate, queue stability on wireline networks. On satellite networks, they exhibit better packet loss rates, goodput and link utilization. We also observed that the RTT based decay function helps to reduce the amount of reverse traffic significantly on both wireline and satellite networks while ensuring that performance remains virtually the same as in the FLCD algorithm without reverse traffic reduction.

6.2 Future Work

The greatest challenge that we have encountered in this research relates to the expensiveness of the process of optimizing the FLCD algorithm in Chapter 3. The iterations and the continuous exchange of parameters between the AMOPSO and the FLCD algorithms have been observed to be very time-consuming processes for a single processor machine. In order to speed up the optimization process, we suggest exploring the use of High End Computing (HEC) techniques where multiple processors would be used in the optimization process. This would also enable us to increase the number of iterations in order to achieve a better Pareto front than the present one. The number of parameters in the particle vector, which captures the fuzzy set parameters, could also be increased in order to come up with even better results.

Another observation prompting future work relates to the uncertainties associated with the regular two-dimensional (type-1) fuzzy sets [MJ02] which have been used in the FLCD algorithm and all its predecessors. Recently the three dimensional type-2 fuzzy sets have been proposed in order to get rid of the uncertainties associated with type-1 fuzzy sets. It has however been pointed out that type-2 fuzzy sets are more complex than type-1 fuzzy sets [MJ02]. As part of future work, we suggest implementing the FLCD algorithm using type-2 fuzzy sets and comparing its performance (in terms of uncertainty levels and complexity) with that of the type-1 fuzzy set FLCD algorithm proposed in this dissertation. A search for mechanisms that would reduce the complexity of the type-2 fuzzy set FLCD algorithm while maintaining good performance (in terms of uncertainties) would also be an important research venture.

Lastly, we suggest the extension of the concepts proposed in this dissertation to mobile adhoc networks (MANET). Nodes in such networks are characterized by limited resources. Constraints are given on processing power, available memory and processing bandwidth [HKK04]. Therefore any attempt of extending the FLCD algorithm and the dual explicit congestion mechanism to MANETs must take care of these constraints.

Appendices

A MAJOR NETWORK PERFORMANCE METRICS

This Appendix presents the five major network performance metrics as proposed in [BRH03].

A.1 Notation

A.1.1 Notation for All Traffic Flows

The total simulation time is denoted by τ and network capacity by C. Let F denote the total (both UDP and TCP) flows indexed by $i \in [1, F]$ traversing the bottleneck link in time τ. For flow i, the following variables are defined:

- S_i, the total size of the data received
- $S_i^{'}$, the total size of the data sent

A.1.1 Notation for UDPTraffic Flows

Let N denote the number of UDP (real-time) flows indexed by $j \in [1, N]$ traversing the bottleneck link in time τ .For each UDP flow j, the following variables are defined:

- R_j, the total size of UDP data received
- $\overline{D_j}$, the average delay
- $\overline{J_j}$, the average jitter
- P_j, the total number of packets

In UDP flow j, for each packet indexed by $k \in [1, P_j]$, the following variables are defined:

- J_k, jitter between packet k and packet $k+1$
- D_k, delay for packet k
- s_k, the time packet k was sent from the sender
- r_k, the time packet k was received at the receiver

A.2 Metric definitions

The five general network performance metrics are:

A.2.1 Utilization Metric

$$\frac{\sum_{i=1}^{F} S_i}{C\tau} \tag{A.1}$$

A.2.2 Fairness Metric

$$\frac{\left(\sum_{i=1}^{F} S_i\right)^2}{F\sum_{i=1}^{F} S_i^2} \tag{A.2}$$

A.2.3 Drop Metric

$$1-\frac{\sum_{i=1}^{F} S_i}{\sum_{i=1}^{F} S_i'} \tag{A.3}$$

A.2.4 Delay Metric

$$\frac{\sum_{j=1}^{N} R_j \overline{D_j}}{\sum_{j=1}^{N} R_j} \tag{A.4}$$

A.2.5 Jitter Metric

$$\frac{\sum_{j=1}^{N} R_j \overline{J_j}}{\sum_{j=1}^{N} R_j} \tag{A.5}$$

A.3 Computation of Metrics

The first three metrics which are used for overall performance evaluation have been presented just as they are defined in [BRH03]. The Delay metric been modified in order to only cater for real-time flows whose performance is heavily dependent on delay and jitter.

A.3.1 Delay Metric Computation

For packet k in UDP flow j the delay is given by:

$$D_k = r_k - s_k \tag{A.6}$$

Average delay is then computed as

$$\overline{D_j} = \frac{\sum_{k=1}^{P_j} D_k}{P_j} \tag{A.7}$$

Therefore, the weighted average delay which takes into account the amount of UDP traffic that has been transferred successfully becomes

$$\frac{\sum_{j=1}^{N} R_j \overline{D_j}}{\sum_{j=1}^{N} R_j} \tag{A.8}$$

A.3.1 Jitter Metric Computation

The Jitter metric is derived based on the definition of jitter for real-time flows [SCF03]. The jitter of a packet stream is defined as the mean deviation of the difference in packet spacing at the receiver compared to the sender, for a pair of packets. Jitter between packet k and packet $k+1$ is expressed as

$$\begin{aligned} J_k &= |(r_{k+1} - r_k) - (s_{k+1} - s_k)| \\ &= |(r_{k+1} - s_{k+1}) - (r_k - s_k)| \end{aligned} \tag{A.9}$$

Average delay is then computed as

$$\overline{J_j} = \frac{\sum_{k=1}^{P_j - 1} J_k}{P_j - 1} \tag{A.10}$$

Therefore, the weighted average jitter which takes into account the amount of UDP traffic that has been transferred successfully becomes

$$\frac{\sum_{j=1}^{N} R_j \overline{J_j}}{\sum_{j=1}^{N} R_j} \tag{A.11}$$

References

[AFP02] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's Initial Window. RFC 3390, October 2002.

[AKU02] F. Akujobi *et al.* BECN for Congestion Control in TCP/IP Networks: Study and Comparative Evaluation. In *Proceedings of Globecom 2002.*

[AKU03] F. Akujobi *et al.* Congestion control in TCP/IP networks: A combined ECN and BECN approach. In *Proceedings of MILCOM 2003 - IEEE Military Communications Conference*, vol. 22, no. 1, Oct 2003 pp. 248-254

[ALLY01] S. Athuraliya, V.H. Li, S.H. Low, Q. Yin. REM: Active Queue Management. *IEEE Network Magazine*, 15(3), pp.48-53, May 2001.

[ANN04] Y. Hadjadj Aoul, A. Nafaa, D. Negru and A. Mehaoua. FAFC: Fast Adaptive Fuzzy AQM Controller for TCP/IP Networks. In *Proc. of IEEE GLOBECOM 2004*, Dalas, Texas, USA, November 29, 2004

[APS99] M. Allman, V. Paxson and W. Richard Stevens. TCP Congestion Control. RFC 2581, April 1999.

[BAK95] J Baker. Requirements for IP Version 4 Routers. Internet RFC 1812, June 1995.

[BDT99] E. Bonabeau, M. Dorigo and G. Théraulaz. *From Natural to Artificial Swarm Intelligence.* Oxford University Press, New York, 1999

[BF92] Bill P. Buckles and Fred E. Petry. *Genetic Algorithms.* IEEE Computer Society Press, 1992.

[BK98] H. Balakrishnan and R.H. Katz. Explicit Loss Notification and Wireless Web Performance. In *Proceedings of IEEE Globecom Internet Mini-conference*, Sydney, Australia, November 1998

[BLA98] S. Blake *et al.* Architecture for Differentiated Services. IETF RFC 2475, 1998.

[BRA98] B. Braden *et al.* Recommendations on Active Queue Management and Congestion Avoidance in the Internet. IETF RFC2309, April 1998.

[BRH03] A. Bitorika, M. Robin, and M. Huggard. A Survey of Active Queue Management Schemes. Trinity College Dublin, Department of Computer Science, Tech. Rep., September 2003.

[BRH04] A. Bitorika, M.Robin and M. Huggard. A comparative study of Active Queue Management schemes. In *Proceedings International Conference on Communications (ICC 2004),* June 2004.

[CDS74] Vint Cerf, Yogen Dalal, and Carl Sunshine. Specification of Internet Transmission Control Program. RFC 675, December 1974.

[CHR03a] C. Chrysostomou et al. Fuzzy Logic Congestion Control in TCP/IP Best-Effort Networks, *2003 Australian Telecommunications Networks and Applications Conference (ATNAC 2003)*, Melbourne, Australia, 8 - 10 December 2003 (CD ROM - ISBN: 0-646-42229-4).

[CHR03b] C. Chrysostomou, A. Pitsillides, L. Rossides, A. Sekercioglu. Fuzzy Logic Controlled RED: Congestion Control in TCP/IP Differentiated Services Networks. In *Special Issue on The Management of Uncertainty in Computing Applications in Soft Computing Journal - A Fusion of Foundations, Methodologies and Applications*, Vol 8, Number 2, pp. 79 - 92, December 2003.

[CISCO] http://www.cisco.com/warp/public/cc/techno/media/lan/gig/tech/10gig_sd.htm

[CISCO98] Technical Specification from Cisco. Distributed Weighted Random Early Detection. http://www.cisco.com/univercd/cc/td/doc/product/software/ios111/wred.pdf

[CISCO02] CISCO. Weighted Random Early Detection on the Cisco12000 Series Router. http://www.cisco.com/univercd/cc/td/doc/product/software/ios112/ios112p/gsr/wred_gs.htm: CISCO Ltd., 2002.

[CK74] Vint Cerf and Bob Kahn. A protocol for packet network intercommunication. *IEEE Transactions on Communications*, 22(5), May 1974.

[C0] J. Crowcroft and P. Oeschslin. Services using a weighted proportionally fair sharing TCP. *ACM Computer Communications review*, 28, 53-67

[COE99] C.A Coello Coello. A Comprehensive survey of evolutionary based multiobjective optimization techniques. *Knowledge and Information Systems*, vol.1, no.3, pp.269-308, Aug.1999

[COE04] C.A Coello Coello, G.T. Pulido and M.A Lechunga. Handling Multiple Objectives With Particle Swarm Optimization. In Proceedings of the 2002 Congress on evolutionary Computation. In *IEEE Transactions on Evolutionary Computation*, 8(3), pp.256-279, June 2004.

[COX03] Cox, John. Wireless LAN throughput on the rise, Network World Fusion, 26 September, 2003. http://www.nwfusion.com/news/2003/0929802111n.html

[DAP02] K. Deb, S. Agrawal, A. Pratap and T. Miyerivan. A Fast Elitist Multi-Objective Genetic Algorithm: NGSA-II. In *IEEE Transactions on Evolutionary Computation* 6(2002) 182-187.

[DAV02] B Davie et al. An Expedited Forwarding PHB, RFC 3246, March 2002.

[DKG01] A. Durresi, S. Kota, M. Goyal, R. Jain and V. Bharani. Achieving QoS for TCP Traffic in Satellite Networks with Differentiated Services. *Journal of Space Communications* , Volume 17, Number 1-3, pp. 125-136, 2001

[DSL01a] 2001A. Durresi, M. Sridharan, C. Liu and Raj Jain. Improving Congestion Control in Satellite Networks. In *Proceedings of SPIE Conference on Quality of Service over the Net Generation Data Networks*, Denver, August 20-24, 2001, Vol. 4524, pp. 293-303.

[DSL01b] A. Durresi, M. Sridharan, C. Liu, M. Goyal and R. Jain. Congestion Control using Multilevel Explicit Congestion Notification in Satellite Networks. In *Proceedings of 10th IEEE International Conference on Computer Communications and Networks (ICCCN2001)* , Scottsdale, AZ, October.

[DSR02] C. Dovrolis, D. Stiliadis, and P. Ramanathan. Proportional differentiated services: Delay differentiation and packet scheduling. *IEEE/ACM Transactions on Networking*, 10(1):12–26, 2002.

[FEN99] W. Feng et al Blue: A New Class of Active Queue Management Algorithms. *Technical Report CSE-TR-387-99*, University of Michigan, 1999

[FGS01] S. Floyd, R. Gummadi, S.Shenker. Adaptive RED: An Algorithm for increasing the robustness of RED's Active Queue Management. *Technical report ICSI*, August 2001

[FH99] S. Floyd, T. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 2582, April 1999.

[FIE04] J.E. Fieldsend. Multi-Objective Particle Swarm Optimization Methods. *Technical Report #419*, Department of Computer Science, University of Exeter, March 2004.

[FJ92] S. Floyd and V. Jacobson. On Traffic Phase Effects in Packet-Switched Gateways. Internetworking: Research and Experience, V.3 N.3, September 1992, p.115-156.

[FJ93] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. In *IEEE/ACM Transactions in Networking*, 1(4):397-413, August 1993

[FKS99] W. Feng, D. Kandlur, D. Saha and K. Shin. A Self-Configuring RED Gateway. *INFOCOM'99*, March 1999, pp.1320-1328

[FLO94] S. Floyd, TCP and Explicit Congestion Notification. In *ACM Computer Communications review*, V.24, N.5, p.10-23, October 1994

[FLO00a] S. Floyd. Congestion Control Principles. RFC 2914, ACIRI, September 2000

[FLO00b] S. Floyd. Recommendation on using Gentle variant RED, http://www.icir.org/floyd/red/gentle.html. March 2000.

[FYX02] Ren Fengyuan, Ren Yong and Shan Xiuming. Design of a fuzzy controller for active queue management. *Computer Communications 25 (2002)*, 874-883

[GD99] N. Ghani and S. Dixit. TCP/IP Enhancements for satellite networks. *IEEE Communications Magazine*, pp.64-72, vol.37, No 7, July 1999

[GKP98] R. Guerin, S. Kamat, V. Peri, and R. Rajan. Scalable QoS provision through buffer management. in *Proc. ACM SIGCOMM' 98*, 1998

[HEI99] J. Heinanen, et al., Assured Forwarding PHB Group, RFC 2597, Jun 1999

[HEN99] T.R. Henderson. *Networking over next generation satellite systems*. PhD dissertation, University of California at Berkeley, 1999

[HKK04] V. Hadzinski, A. Köpke, H. Karl, C. Frank, and W. Drytkiewicz, Improving the Energy efficiency of Directed Diffusion Using pervasive Clustering. In *Proceedings of First European Workshop in Wireless Sensor Networks (EWSN)*, Berlin, Germany (2004) 172-187.

[HL01] Y. Hu and V.O.K. Li. Satellite-Based Internet: A tutorial. *IEEE Commuinications*, March 2001.

[HMMD02] U. Hengartner, S. Moon, R. Mortier and C. Diot. Detection of routing loops and analysis of routing loops in packet traces. In *Proceedings ACM SIGCOMM Internet measurement Workshop*, Marseille, France, November 2002.

[HMTG01] C. V. Hollot, V. Misra, D Towsley, W.B. Gong. A Control theoretic analysis of RED. *In Proceedings of IEEE INFOCOM 2001*, vol.3, pp. 1510-1519,Anchorage, Alaska, 2001

[HNS98] S.J. Hadi, B. Nandy, N. Seddigh. A Proposal for Backward ECN for the Internet Protocol (IPv4/IPv6) , Internet Draft, July 1998

[JAC99] V. Jacobson, et al. An Expedited Forwarding PHB. RFC 2598, Jun 1999.

[JHP03] J. Joutsensalo, T. Hämäläinen, M. Pääkkönen and A. Sayenko. Adaptive weighted fair scheduling method for channel allocation. *IEEE International Conference on Communication 2003 (ICC '03)*, Volume 1, pp. 228-232, 2003

[JK88] Van Jacobson and Michael Karels. Congestion Avoidance and Control. In *Proceedings ACM SIGCOMM '88*, 1988, pp. 314-29.

[KC00] K.D. Knowles and D. Corne. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolution Computation*, 8(2):149-172, 2000.

[KDJ01] S. Kota, A. Durresi, R. Jain. Realizing Future Broadband Satellite Network Services. Chapter in *On Modelling and Simulation Environment for Terrestrial and Satellite Networks,* Edited by A. Nejat Ince, Published by Kluwer, ISBN: 0-7923-7547-5, October 2001.

[KE95] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, Vol. IV, pp. 1942-1948. IEEE service center, Piscataway, New Jersey, 1995.

[KE97] J.Kennedy. The particle swarm: Social adaptation of knowledge. In *IEEE Int. Congress on Evolutionary Computation* (1997), IEEE Press.

[KEL01] F. Kelly. *Mathematical Modeling of the Internet.* Springer-verlag, Berlin (2001)

[KS01] S. Kunniyur and R. Srikant. Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management. *Technical Report*, UIUC, Feb. 2001.

[KS03] Martin Kappes and Sachin Garg. An Experimental Study of Throughput for UDP and VoIP Traffic in IEEE 802.11b Networks. *In Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, March 2003

[KSE04] R. Krishnan, J. P.G. Sterbenz, W.M. Eddy, C. Partridge, and M. Allman. Explicit Transport Error Notification (ETEN) for Error-Prone Wireless and Satellite Networks. *Pre-print: Accepted for publication in Elsevier Computer Networks*, October 2004.

[KUN03] S. Kunniyur. AntiECN Marking: A Marking Scheme for High Bandwidth Delay Connections. In *Proceedings of ICC 2003*, May 2003.

[LJ01] Chunlei Liu and Raj Jain. Improving Explicit Congestion Notification with the Mark-Front Strategy. *Computer Networks*, Vol. 35, no 2-3, pp 185-201, February 2001.

[LI00] C. Li, S. Tsao, M.C. Chen, Y. Sun, and Y. Huang. Proportional Delay Differentiation Service Based on Weighted Fair Queuing, In *Proceedings of IEEE ICCCN 2000.*

[LL89] R.J. Li and E.S. Lee (1987). Analysis of fuzzy queues. *Comput. Math Applications*, 17(1989), 1143-1147

[LOW02] S.H. Low, F. Paganini, J. Wang, S. Adlakha, and J.C. Doyle. Dynamics of TCP/AQM and a scalable control. In *Proceedings of IEEE INFOCOM*, June 2002

[LR98] T. Li, Y. Rekhter. A Provider Architecture for Differentiated Services and Traffic Engineering (PASTE). IETF RFC 2430, October 1998.

[LTC00] C. Li, S. Tsao, M.C. Chen, Y. Sun and Y. Huang. Proportional delay differentiation service based on weighted fair queuing. In *Proc. IEEE ICCN 2000*, 2000

[MAM74] E.H. Mamdani, Application of fuzzy algorithm for control of simple dynamic plant. In *Proceedings of IEEE*, 121 (1974), p. 12.

[MBD99] M. May, J. Bolot, C. Diot, and B. Lyles. Reasons Not to Deploy RED. In *Proceedings of Seventh. International Workshop on Quality of Service*

(IWQoS'99), pages 260-262), London, June 1999.

[MESHD] MeshDynamics, http://www.meshdynamics.com

[MESHN] MeshNetworks, http://www.meshnetworks.com

[MGT00] V. Misra, W.B. Gong, D.Towsley. Fluid-based analysis of a network of AQM routers supporting TCP flows with application to RED, In *Proceedings of ACM/SIGCOMM 2000.*

[MJ02] J.M. Mendel and R.I.B. John. Type-2 Fuzzy Sets made simple. In *IEEE Transactions on Fuzzy Systems*, 10(2), pp.117-127, April 2002.

[ML00] R.Makkar, I. Lambaridis *et. al.* Empirical Study of Buffer Management Scheme for DiffServ Assured Forwarding PHB. *Proceedings of Ninth Conference on Computer Communications and Networks*, Las Vegas, Nevada, 2000.

[MNT04] M. Malli, Qiang Ni, Thierry Turletti, and Chadi Barakat. Adaptive Fair Channel Allocation for QoS Enhancement in IEEE 802.11 Wireless LANs. *IEEE International Conference on Communications (ICC 2004)*, Paris, June 2004.

[MR91] A. Mankin, K. Ramakrishnan. Gateway Congestion Control survey. RFC 1254, August 1991.

[NAG84] John Nagle. Congestion Control in IP/TCP Internetworks. RFC 896, FACC Palo Alto, 6 January 1984.

[ND06a] C.N. Nyirenda, D.S. Dawoud. Performance Evaluation of the Swarm Optimized Fuzzy Logic Congestion Detection Mechanism in Proportional Differentiated Services IP Networks. *Southern African Telecommunications Networks and Applications Conference (SATNAC) 2006 proceedings (CD ROM),*Cape Town, South Africa, 3-6 September 2006.

[ND06b] C.N. Nyirenda, D.S. Dawoud. Multi-objective Particle Swarm Optimization for Fuzzy Logic Based Active Queue Management. *In Proceedings of the 15th IEEE International Conference in Fuzzy Systems as part of the Fourth IEEE World Congress on Computational Intelligence (IEEE WCCI 2006)*, Vancouver, BC, Canada, pages: 2231 - 2238, 16-21 July 2006.

[NS05] NS-2 network simulator, *http://www.isi.edu/nsnam/ns/.*

[NYI05] C.N. Nyirenda. Fuzzy Logic Congestion Control for TCP/IP Networks: A Dual Explicit Notification. *Southern African Telecommunications Networks and Applications Conference (SATNAC) 2005 proceedings (CD ROM),* ISBN 0-620-34908-5, Champagne Sports, Drakensberg, South Africa, Sept 2005.

[OLW99] T.J. Ott, T.V. Lakshman, and L. Wong. SRED: Stabilized RED. In *Proceedings of IEEE INFOCOM 1999.*

[PC04] G.T. Pulido and C.A. Coello Coello.Using Clustering Techniques to Improve the Performance of a Multi-Objective Particle Swarm Optimizer. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Springer-Verlag, Lecture Notes in Computer Science Vol. 3102, pp. 700--712, Seattle, Washington, USA, June 2004.

[PHS05] S. Patchararungruang, S. Halgamuge, N. Shenoy. Optimized Rule Based Delay Proportion Adjustment for Proportional Differentiated Services. In *IEEE Journal on Selected Areas in Communications*, February 2005.

[PM79] P. Procyk and E. Mamdani: A linguistic self organizing process controller, *Automatica* 15 (1) (1979) 15-30.

[POS81] J. Postel. Internet Control Message Protocol. RFC 792, September 1981

[PPP00] R. Pan, B. Prabhakar, and K. Psounis. Choke - a stateless active queue management scheme for approximating fair bandwidth. In *Proceedings of INFOCOM'00*, March 2000, pp. 942-951

[PRA80] H.M Prade. An Outline of fuzzy or probabilistic models for queuing systems. In *Proceedings of Symposium for Policy Analytical Information Systems*, in Durham, NC, 1980, 147-153

[PRF99] H. Pomares, I. Rojas, F.J. Fernandez, M. Anguita, E. Ros and A. Prieto: A new approach for the design of fuzzy controllers in real time. In *Proc. 8th Int. Conf. Fuzzy Systems*, Seoul, Korea, August 1999, pp. 522-526.

[PRG04] H. Pomares, I. Rojas, J. Gonzalez, M. Damas, B. Pino and A. Prieto: Online Global Learning in Direct Fuzzy Controllers, *IEEE Transactions on Fuzzy Systems*, Vol. 12 (2), April 2004.

[PUL05] G.T. Pulido. *On the Use of Self-Adaptation and Elitism for Multiobjective Particle Swarm Optimization*, PhD Thesis, Computer Science Section, Department of Electrical Engineering, CINVESTAV-IPN, September 2005.

[PY98] K. M. Passino and S. Yurkovich. *Fuzzy Control*. Edited by Prentice Hall, ISBN 0-201-18074-X (1998).

[RF01] K. Ramakrishnan and S. Floyd. The Addition of Explicit Congestion Notification (ECN) to IP. RFC-3168, September 2001.

[RI05] F. Rivas-Davalos and M.R. Irving. An Approach based on the Strength Pareto Evolutionary Algorithm 2 for power Distribution System Planning, in C.A. Coello Coello and A. A. H. Aguirre and E. Zitzler(editors), *Evolutionary Multi-Criterion Optimization. Third International Conference, EMO 2005,* pp.707-720, Springer. Lecture Notes in Computer Science Vol. 3410, Guanajuato, Mexico, March 2005

[SA00] J. Hadi Salim, U. Ahmed. Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks. RFC 2884, July 2000.

[SAF05] P. Sarolahti, M. Allman and S. Floyd: Evaluating Quick-Start for TCP. *Manuscript*, February 2005.

[SAM99] N. Samaraweera. Non-Congestion Packet Loss Detection for TCP Error Recovery Using Wireless Links. In *IEE Proceedings in Communications*, Volume 146, Number 4, August 1999, pp.222-230.

[SBM02] I. Skrjanc, S. Blazic and D. Matko: Direct fuzzy model reference adaptive control, *International Journal on Intelligent Systems*, vol.17, pp. 943-963, 2002.

[SCF03] H. Schulzrinne, S.Casner, F. Frederick, and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. draft-ietf-avt-rtp-new-12.txt, March 2003.

[SCH00] J. Schiller. *Mobile Communications*.ISBN 0-201-39386-2, 2000.

[SDJ04] M. Sridharan, A. Durresi and R. Jain. Adaptive Multi-level Explicit Congestion Notification. In *Proceedings of 2004 Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'04)*, San Jose, CA, July 25-29, 2004

[SH02] S. Sundarrajan and J. Heidemann. Study of TCP Quick-Start with NS-2. Unpublished report, University of South California, 2002.

[SLM04] M. Spott, K. Leiviskä and T. Martin. *Roadmap Contribution IBA C Applications in Telecommunications, Multimedia and Services. European Network on Intelligent Technologies (EUNITE) for Smart Adaptive Systems (SAS)*, July 2004

[SSZ98] I. Stoica, S. Shenker and H. Zhang. Core stateless fair queuing: Achieving approximately fair bandwidth allocations in high speed networks. In *Proc. ACM SIGCOMM '98*, 1998.

[WAN03] W. Chonggang *et al.* AFRED An Adaptive Fuzzy-based Control Algorithm for Active Queue Management. In *Proceedings of the 28th Annual IEEE International Conference on Local Computer Networks 2003*

[WES02] Westberg et. al, Resource Management in Diffserv (RMD): A functionality and performance behaviour. In *Proceedings of the Seventh International Workshop on Protocols For High-Speed Networks (PfHSN'2002)*

[WH99] W.Weiss, J. Heinanen, F. Baker and J. Wroclawski. Assured Forwarding PHB Group. The Internet Society RFC 2597, Juine 1999.

[WHI97] P.P. White. RSVP and integrated services in the internet: A tutorial. *IEEE Communications Magazine*, pp. 100-106, May 1997.

[WY03] Bartek P. Wydrowski. *Technique in Internet Congestion Control*. PhD Thesis, University of Melbourne, February 2003.

[WZ02] B. Wydrowski and M. Zukerman. GREEN: An active queue management algorithm for a self managed traffic. In *Proceedings of ICC'02*, vol.4, May 2002, pp. 2368-2372.

[YC03] F. Yanfei and F.L. Chuang. Design a PID controller for Active Queue Management. *ISCC 2003* (July 2003).

[YI04] S. Yi *et al.* Proxy-RED: An AQM scheme for wireless LANS. In *Proc. ICCCN*, Chicago, Oct. 2004

[YM85] S. Yasunobu, S. Miyamoto. Automatic Train Operation by Predictive Fuzzy Control. *In Industrial Applications of Fuzzy Control*, M. Sugeno, Ed., pp. 1-18, Elsevier Science Publishers, 1985.

[YT03] M. H. Yaghmaee and H. A. Toosi. A Fuzzy Based Active Queue Management Algorithm. In *Proceedings of 2003 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'03)*, Montreal, Canada, July, 2003.

[ZA65] L.A Zadeh, Fuzzy sets. *Information and Control*, 8:338-353, 1965.

[ZAK05] R.H. Zakon, Hobbes' Internet Timeline, http://www.zakon.org/robert/internet/timeline/, Robert Hobbes, 2005.

[ZC05] B. Zhao and Y. Cao. Multiple objective particle swarm optimization technique for economic load dispatch. *Journal of Zhejiang University SCIENCE 2005*, 6A(5):pages 420-427

[ZIT99] Eckart, Zitzler. *Evolutionary algorithms for multiobjective optimizations: methods and applications*. PhD Thesis, Swiss Federal Institute of Technology, Zurich, 1999

[ZR05] Jing Zhu, and Sumit Roy. 802.11 Mesh Networks with Two-Radio Access Points. *ICC 2005*, May 2005, Seoul, Korea.