PERFORMANCE MODELLING AND ANALYSIS OF CONGESTION CONTROL MECHANISMS FOR COMMUNICATION NETWORKS WITH QUALITY OF SERVICE CONSTRAINTS

R. H. A. FARES

PhD

2010

Performance Modelling and Analysis of Congestion Control Mechanisms for Communication Networks with Quality of Service Constraints

An investigation into new methods of controlling congestion and mean delay in communication networks with both short range dependent and long range dependent traffic

Rasha Hamed Abdel Moaty Fares

Submitted for the degree of
Doctor of Philosophy

Department of Computing
School of Computing, Informatics and Media
University of Bradford

2010

Abstract

Rasha Hamed Abdel Moaty Fares

Performance modelling and analysis of congestion control mechanisms for communication networks with quality of service constraints

Keywords: Congestion Control – Mean Delay – Quality of Service – Short Range Dependent – Long Range Dependent – Communication Networks.

Active Queue Management (AQM) schemes are used for ensuring the Quality of Service (QoS) in telecommunication networks. However, they are sensitive to parameter settings and have weaknesses in detecting and controlling congestion under dynamically changing network situations. Another drawback for the AQM algorithms is that they have been applied only on the Markovian models which are considered as Short Range Dependent (SRD) traffic models. However, traffic measurements from communication networks have shown that network traffic can exhibit self-similar as well as Long Range Dependent (LRD) properties. Therefore, it is important to design new algorithms not only to control congestion but also to have the ability to predict the onset of congestion within a network.

An aim of this research is to devise some new congestion control methods for communication networks that make use of various traffic characteristics, such as LRD, which has not previously been employed in congestion control methods currently used in the Internet. A queueing model with a number of ON/OFF sources has been used and this incorporates a novel congestion prediction algorithm for AQM. The simulation results have shown that applying the algorithm can provide better performance than an equivalent system without the prediction. Modifying the algorithm by the inclusion of a sliding window mechanism has been shown to further improve the performance in terms of controlling the total number of packets within the system and improving the throughput.

Also considered is the important problem of maintaining QoS constraints, such as mean delay, which is crucially important in providing satisfactory transmission of real-time services over multi-service networks like the Internet and which were not originally designed for this purpose. An algorithm has been developed to provide a control strategy that operates on a buffer which incorporates a moveable threshold. The algorithm has been developed to control the mean delay by dynamically adjusting the threshold, which, in turn, controls the effective arrival rate by randomly dropping packets. This work has been carried out using a mixture of computer simulation and analytical modelling. The performance of the new methods that have been produced has been evaluated against existing methods with encouraging results.

Declaration

I hereby declare that this thesis has been genuinely carried out by myself and has not been used in any previous application for a degree. The invaluable participation of others in this thesis has been acknowledged where appropriate.

Rasha Hamed Fares

Dedication

To my parents, my husband and my children

Acknowledgments

In the Name of Allah, the Most Gracious, the Most Merciful

All Praise is Due to Allah for His Glorious Ability and Great Power. I would like to thank Allah for giving me the power, patience and knowledge to complete this doctoral thesis.

I would like to express my gratitude to all those who gave me the support and encouragement to complete this thesis**.** First of all, I would like to express my sincere appreciation and profound gratitude to my supervisor Professor Mike Woodward, who provided invaluable guidance throughout the course of this research. I would like to express my appreciation for his helpful suggestions, continual and unwavering encouragements, patience and endless support through the entire research and thesis writing process.

I would like to express my appreciation to all the staff at the department of Computing, especially: Professor Irfan Awan, Professor Demetres Kouvatsos, Professor Valentina Zharkova, Dr. Geyong Min and Mrs Bev Yates. I would also like to thank my colleagues: Dr. Iman Alansari, Dr. Khalid Alawfi, Dr. Fahad Alraddady, Amina Alswai, Ibtehal Nafea, Guzlan Miskeen, Yasmin Bashon and Roquia Abdelrahman. Many thanks also go to my friend Hadjer Djellat.

I would like to gratefully acknowledge the provision of the scholarship from the Ministry of Higher Education in Egypt and the Egyptian Cultural Centre and Educational Bureau in London.

I am deeply indebted to my family who offered me great support and encouragement throughout my studies. My heartfelt thanks go to my parents, my sisters and my brother for their unfailing love, prayers and selfless support.

Last but not least, my affectionate thanks go to my husband Dr. Amin Asfor and to my lovely children Mariam and Anaas for their sacrifices, support and patience. Words are inadequate to express my gratitude to them for their tremendous support and continuous encouragement.

Publications

1. Rasha Fares and Mike Woodward, “A new Algorithm for Controlling the Mean Queue Length in a Buffer with Time Varying Arrival Rate”, The 4th International Conference for Internet Technology and Secured Transactions (ICITST-2009), In proceedings of IEEE Computer Society, London, UK, 9th-12th November 2009.

2. Rasha Fares and Mike Woodward, “Maintaining Delay Constraints through a Buffer Using Dynamic Queue Threshold”, 1st Annual Conference for the Egyptian students in UK and Ireland. The Egyptian Cultural Centre and Educational Bureau, London, 15th October 2009.

3. Rasha H. Fares and Mike E. Woodward, “The use of Long Range Dependence for Network Congestion Prediction”, The First International Conference on Evolving Internet (INTERNET 2009), In proceedings of IEEE Computer Society, Cannes/La Bocca, French Riviera, France, 23rd - 29th August 2009.

4. R. Fares, M. Woodward and I. Awan, “Congestion Control Mechanisms for Multi-Class Traffic”, the 9th Informatics Workshop for Research Students, University of Bradford, Bradford, UK, June 2008.

5. R. Fares, M. Woodward and I. Awan, “Performance Analysis of Buffer Management Schemes under Multi-Class Traffic”, The 9th Annual Postgraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting, Liverpool John Moores University, Liverpool, UK, 23rd – 24th June 2008.

Contents

List of Tables

List of Figures

List of Abbreviations

ACK	Acknowledgement
AIMD	Additive Increase Multiplicative Decrease
AQM	Active Queue Management
ARED	Adaptive RED
avg	Average queue length
BRED	Balanced RED
CBT-RED	Class Based Threshold-RED
cdf	Cumulative distribution function
cwnd	Congestion window
DC-RED	Dual Class RED
DES	Discrete Event Simulation
$\mathbf{D_M}$	Measured mean delay
DSRED	Double Slope RED
$\mathbf{D_T}$	Target mean delay
ERD	Early Random Drop
EWMA	Exponential Weighted Moving Average
FIFO	First-In First-Out
FRED	Flow Random Early Drop
FTP	File Transfer Protocol
H	Hurst parameter
HDLC	High-level Data Link Control
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force

IP	Internet Protocol
LRD	Long Range Dependent
max_p	Maximum dropping probability
max_{th}	Maximum threshold
min_{th}	Minimum threshold
MMPP	Markov Modulated Poisson Process
MQL	Mean Queue Length
pdf	Probability density function
pmf	Probability mass function
PQM	Passive Queue Management
QoS	Quality of Service
RED	Random Early Detection
REM	Random Exponential Marking
RTT	Round Trip Time
sec	Seconds
SRD	Short Range Dependent
SRED	Stabilized RED
ssthrsh	Slow start threshold
TCP	Transmission Control Protocol
TWL	Time Window Length
UDP	User Datagram Protocol
VBR	Variable Bit Rate
W	Time Window
w_q	Queue weight
WWW	World Wide Web

CHAPTER 1

Introduction

1.1 Introduction

With the rapid proliferation of the Internet, there has been an enormous growth in both the demand for access from its users and in the demand for new services. The success of the Internet is largely dependent on the strength of its protocols. Over the last decade, the Transmission Control Protocol (TCP) has been the predominant transport protocol used by the Internet Protocol (IP) technology to support various Internet services. TCP has consistently met the challenge of new applications but due to the massive growth of the Internet, weaknesses in TCP have become increasingly apparent [1]. For instance delay, packet losses and decreasing network efficiency are examples of drawbacks that can arise when the traffic conditions change.

Internet traffic congestion occurs when the aggregate demand exceeds the capacity of the available resources and hence causes performance degradation. Recently, as demands for access have exceeded the ability for providers to upgrade network paths, the networks' efficiency has deteriorated and congestion has become a persistent problem [2]. Congestion results in packets being dropped or lost from the network during transmission. The main reason for dropping these packets is that when subsequent packets arrive at each network element, there is not enough buffer space to accommodate all the transmitted packets. Using large buffers can absorb more bursty traffic but will increase the end-to-end delay as well, which will decrease the overall network performance.

In order to guarantee Quality of Service (QoS) to diverse Internet services, it is important to employ effective buffer management schemes at Internet routers. Various buffer management mechanisms have been proposed to control Internet traffic congestion and satisfy specified QoS requirements. In an attempt to address the growing needs of applications, the Internet Engineering Task Force (IETF) has recommended the use of Active Queue Management (AQM) schemes for congestion control [3]. The IETF also recommended the use of the Random Early Detection (RED) algorithm [4] as the default mechanism for managing queue lengths. RED aimed at avoiding congestion by predicting when it will occur rather than reacting to it.

RED has been prone to some configuration problems due to its sensitivity to parameter settings [5]. Many significant modifications have been done on RED in order to improve its performance such as Adaptive RED (ARED) [6], BLUE [7], Random Exponential Marking (REM) [8-9] and Double Slope RED (DSRED) [10]. RED and its variants usually operate in conjunction with TCP but they rely on static thresholds which can be restrictive when they operate with sources with varying arrival rates. This suggests the need for a new adaptive algorithm for AQM that is simple to implement and can control congestion when the arrival rate varies with time.

In order to design a robust and a reliable congestion control algorithm, it is important to address the characteristics of some types of modern network traffic such as Long Range Dependent (LRD) and self-similar traffic [11-12]. Recent studies have shown that network traffic can exhibit self-similar as well as LRD properties [11, 13-14]. Traditional models are also known as Short Range Dependent (SRD) traffic models, and their use in networks characterized by self-similar processes can lead to overestimations about the performance of the analyzed networks [12]. The properties of self-similar traffic are very different from the properties of traditional models based on Poisson or the Markovian models. The scale-invariant characteristics of the self-similar traffic are in strong contrast to traditional network traffic models which show burstiness at short time scales but are smooth at large time scales [15]. The scale-invariant burstiness implies the existence of some periods of high activity at a wide range of time scales which badly affects congestion control. In order to address some of these issues, this thesis focuses on investigating two extremely important challenges to today's Internet: the use of LRD for network congestion prediction and bounding mean delay in a buffer with a time-varying arrival rate.

1.2 Motivation

The ability to predict traffic congestion within a network is one of the fundamental requirements of modern network design. Congestion prediction has become a fundamental objective of some network management algorithms to guarantee a better QoS to users [16]. This is considered a challenging and laborious problem that encompasses several components: the transport protocols, the network design, the control mechanisms and the traffic's nature itself [14]. Therefore, high performance predictors are required that are efficient and simple to implement. LRD implies the existence of a correlation structure, which may be exploitable for congestion prediction purposes. The correlation structure present in LRD traffic can be used to predict the future traffic levels [15, 17-18]. The feasibility of predicting the congestion under self-similar traffic conditions with sufficient reliability can be effectively utilized for congestion control purposes. The predictability structure present in LRD traffic can be used for improving network performance based on the feedback algorithm presented in this thesis.

The rapid growth of the Internet and the increased demand to use the Internet for time-sensitive applications has necessitated the need for effective congestion control algorithms. To support the requirements for real-time applications such as audio and video applications, communication networks must provide service guarantees to connections, including guarantees on throughput and network delays. For the most demanding applications, such as safety critical ones, the network should offer a service which provides a bounded delay guarantee. However, for the packet switched technology considered, this is not feasible since instantaneous delay is a random variable that changes from instant to instant. However, provision of a bounded mean delay, although sometimes not considered the best metric to use in isolation, can guarantee average bit rates and be invaluable for services that require a prompt delivery. This applies whether these services are real-time services or not. In summary therefore, this thesis focuses on the extremely important challenges to today’s Internet and the development of new mechanisms to control congestion and mean delay in communication networks.

1.3 Aims and Objectives

Providing QoS guarantees for real-time traffic has become an increasingly important and challenging topic in the design of high-speed networks. One aim of this research is to develop new algorithms to provide mean delay guarantee for real-time traffic to satisfy QoS requirements. This thesis also aims at investigating new mechanisms for controlling congestion and mean delay in communication networks with both SRD and LRD traffic. It also aims at developing a new algorithm that makes use of a network characterized with LRD and self-similar traffic in order to achieve congestion prediction.

In order to achieve the aims of this research, the objectives of the thesis are set as follows:

- To review the development of the congestion control mechanisms and to learn about the different queue management algorithms.
- To develop a multi-class queueing system based on RED with two classes of traffic and to examine the effect of each class on the other in a shared buffer.
- To identify the difference between SRD and LRD traffic characteristics.
- To understand the LRD and self-similar traffic characteristics and their use in congestion prediction.
- To develop a new algorithm to maintain average delay constraints through a buffer with time-varying arrival rate.
- To test and validate analytical models by comparing them with corresponding simulation models running in a steady state.

1.4 Original Contributions

The principal contributions of this thesis are:

- The development of a new congestion prediction algorithm based on LRD and self-similar characteristics found in modern Internet traffic. It has been shown that the mean time spent ON for each node can be used as an indicator of which node is causing congestion. Modifying the algorithm by the inclusion of a sliding window mechanism has demonstrated a further improvement in performance.

- The use of a dynamic moving threshold in a buffer with time-varying arrival rate to maintain the average delay at a constant value when the arrival rate varies with time. An equation has been developed that relates the threshold position to the target mean delay over each time window. The accuracy of the analytical model has been verified using a simulation model. The algorithm has been generalized for other arrival processes in order to examine the effects of LRD and self-similarity which are the characteristics of some types of modern network traffic.

1.5 Thesis Organization

The rest of the thesis is organized as follows:

Chapter 2 surveys related work on the congestion problem and the different mechanisms for controlling congestion. It also presents the concepts of TCP congestion control.

Chapter 3 reviews the queue management algorithms. It addresses the significant weaknesses in the Passive Queue Management (PQM) algorithms and explains the AQM algorithms that have been developed to address the drawbacks of the PQM schemes. It also explains in detail the RED algorithm and its variants.

Chapter 4 gives a detailed explanation of the simulation model which has been used as a basis for the rest of the simulation models throughout the thesis.

Chapter 5 investigates the effects of using a multi-class traffic in a shared buffer. It tests the marginal performance as well as the overall performance.

Chapter 6 is concerned with providing a new feedback algorithm for congestion prediction. The developed algorithm makes use of a network characterized with LRD and self-similar traffic in order to predict the onset of congestion within a network.

Chapter 7 describes a novel mechanism for AQM that provides average delay guarantees for real-time applications. It provides an analytical model that incorporates a control strategy which uses a dynamic moving threshold. The algorithm has been developed to control the mean delay by dynamically adjusting the threshold, which, in turn, controls the effective arrival rate by randomly dropping packets. This work has been carried out using a mixture of computer simulation and analytical modelling.

Chapter 8 summarizes the contributions of this thesis and suggests the recommendations for future work of interest.

CHAPTER 2

Internet Traffic Congestion and its Control

2.1 Introduction

Computer networks have experienced an explosive growth over the years. The increasing number of wired and wireless networks has caused severe congestion problems. Congestion is a problem that occurs on shared networks when multiple users compete for access to the same resources. Congestion typically occurs where multiple links feed into a single link. It also occurs at routers when nodes are subjected to more traffic than they are designed to handle. During congestion periods, congestion persists and losses can be significant. Congestion losses cannot be avoided by modest increases in buffer capacity, as excessive buffer size can lead to excessive delay and hence degradation of the performance.

Congestion control is about using the network as efficiently as possible so as to avoid congestion collapse. Many algorithms have been proposed to control congestion. The goal of the congestion control mechanisms is to use the network efficiently and attain the highest possible throughput while maintaining a low loss ratio and small delay

[2]. Despite the fact that a number of mechanisms have been proposed to control congestion, the search for new mechanisms continues [19]. The reasons for this are: first, it is difficult to get a satisfactory solution because there are many requirements for congestion control schemes. Second, there are several network design policies that affect the design of the congestion control mechanisms. Therefore, a mechanism that is designed for one network may not work on another network with different architecture.

2.2 Congestion Management

Technological advances and customer demands are rapidly ushering in high-speed networks. This has made congestion management an important issue in recent networks in order to provide good service under heavy load. Early congestion notification would clearly decrease the effects of congestion such as packet loss. A number of control mechanisms for congestion control have been suggested and found to increase the performance of the Internet [20]. These studies started in the late 80's [21] and some form the basis for current implementations. For example, the proposals by Jacobson [21] form the basis for the TCP congestion control in current implementations [22]. Congestion management has three aspects [23]: congestion prevention, congestion avoidance and congestion recovery and these are described in what follows.

2.2.1 Congestion Prevention

Congestion prevention [23] is the most essential aspect of congestion management. It involves designing a network that minimizes the probability that congestion will occur [24]. By applying prevention algorithms, the peak demands can be predicted with reasonable precision. Congestion prevention should include well-designed routing algorithms and queueing polices to ensure that a user's access rate does not exceed its subscribed traffic rate and to protect critical classes of traffic.

2.2.2 Congestion Avoidance

Congestion avoidance [25] involves detecting when congestion is imminent and aims at keeping the operation of a network at or near the maximum power. It allows a network to operate in the region of low delay and high throughput with minimal queueing, thereby preventing it from entering the congestion state [26]. Congestion avoidance is action that is taken by the network to prevent congestion, so it is preventive in nature. Actions should be taken before performance degradation occurs to reduce the chance of congestion. Congestion can be avoided by monitoring traffic patterns and considering changing the packet routing tables to route traffic around a heavily loaded network [27]. There are several alternatives for a source to detect when congestion is imminent [28]:

- Congestion occurs when the output buffers at a switch are full. Congestion avoidance can be initiated when some fraction of the buffers are full.
- By monitoring output line usage as congestion occurs when usage exceeds a threshold.
- By monitoring round trip delays as an increase in these delays causes an increase in queue sizes and congestion.
- By setting a timer that sets off an alarm when a packet is not acknowledged in time.

2.2.3 Congestion Recovery

Congestion recovery is action taken by the network after performance degradation is detected. When congestion occurs, actions are taken to help the network to recover. The goal of congestion recovery is to limit the effects of congestion and to restore the operation of the network to its normal state after congestion has occurred [29]. Without congestion recovery algorithms, networks may crash entirely when congestion is detected. Therefore, congestion recovery schemes would still be required even if a network adopts a strategy of congestion avoidance. The reason for this is to retain throughput in the case of abrupt changes in the network that may cause congestion.

2.3 Congestion Collapse

The Internet first experienced the congestion collapse problem in the 1980s [2]. John Nagel was one of the earliest researchers who mentioned the term ‘congestion collapse’ in 1984 [30]. In particular, when IP gateways connect networks of widely different bandwidth then the IP gateways are vulnerable to the congestion collapse phenomenon. The normal behaviour in heavily loaded pure datagram networks is as follows: as nodes become congested, the Round Trip Time (RTT) through the network increases and the count of datagrams also increases; this is normal as long as there is only one copy of each datagram in transit. This indicates that congestion is under control. If the RTT becomes shorter than the sending host’s measurements of RTT, this can indicate that the network is running into serious trouble. This indicates that the network is entering the congestion collapse phase [30]. When the RTT exceeds the maximum retransmission interval for any host, more and more copies of the same datagram will be introduced into the network. This causes all the available buffers to be full and packets to be dropped. Congestion collapse happens when the RTT is at its maximum and hosts are sending each packet several times [30]. When collapse occurs, a larger fraction of the packets in the network will be duplicated and goodput will be reduced.

2.4 Congestion Control

Congestion control in packet switching networks may involve different components of the congestion management strategy including congestion avoidance and congestion recovery. Such mechanisms have to be provided to avoid congestion collapse. Congestion control became a high priority in network design due to ever growing network bandwidth and the rapidly expanding Internet applications [29]. Controlling congestion is the combined responsibility of network gateways and end point hosts. Gateways are responsible for congestion detection, controlling queue size and arrival rate control. Sources are responsible for the data transmission rates adjustments to enable the gateways to achieve their goals. Congestion control is concerned with allocating the network resources such that the network can operate at an acceptable performance level under heavy load [19]. Congestion control schemes are recovery mechanisms which help the network to recover from the congestion state. These schemes protect the network from being flooded by its users and help improve the performance after congestion has occurred [31].

Due to increasing mismatch in link speeds caused by intermixing of old and new technology, congestion became a significant problem. Recent technological advances have resulted in a significant increase in the bandwidths of computer network links. This heterogeneity has resulted in mismatch of arrival rates and service rates in the intermediate nodes, causing increasing queueing and congestion. Without proper protocol redesign, the congestion problem will not be solved and, thus reduce performance. This has led to the following myths about congestion [19]:

- Congestion is caused by a shortage of buffer space. This problem cannot be solved with a large buffer space because with infinite buffers, the queues and the delays can get so long that by the time the packets come out of the buffer, most of them have already timed out and have been retransmitted. Large buffers are considered more harmful than smaller ones since packets have to be dropped after they have consumed precious network resources [32].

- Congestion is caused by slow links. This cannot simply be solved with high-speed links. Introducing high-speed links without proper congestion control can lead to reduced performance and increased instability. With high-speed links, the arrival rate will be much higher than the service rate, leading to long queues, buffer overflows and packet losses. This means that high-speed links have to be managed and the protocols have to be designed specifically to ensure that this increasing range of link speeds does not degrade the performance.
- Congestion is caused by slow processors. The congestion problem cannot be solved by high-speed processors. A high-speed processor may increase the mismatch of speeds within a network and hence increase the chances of congestion.
- Congestion can be caused by all of the above.

The conclusion is that congestion is a dynamic problem and cannot be solved with static solutions alone. Without proper protocol implementations, the congestion problem will be even worse. The protocols need to be dynamic in order to detect and react to congestion. A properly designed congestion control algorithm will ensure that users are able to increase their traffic load as long as this does not significantly affect the response time.

The congestion problem can be solved in connection-oriented networks by reserving the resources at all routers during connection setup. In connectionless-networks the congestion problem can be solved by choke packets (explicit messages) from the network to the sources or by timeout on a packet loss [26]. Congestion control mechanisms consist of two parts: a feedback mechanism which allows the network to inform its users of the current state of the network, and a control mechanism which allows the users to adjust their loads on the network. Therefore, congestion control in computer networks can be viewed as a control system for maintaining the overall traffic within certain normal levels [29]. Current literature classifies most congestion control approaches into two categories [2, 29]:

- Open loop congestion control algorithms
- Closed loop congestion control algorithms

The basis for the classification is based on the characteristics of how each algorithm extracts information for their control decision. Figure (2.1) represents a classification of congestion control algorithms.

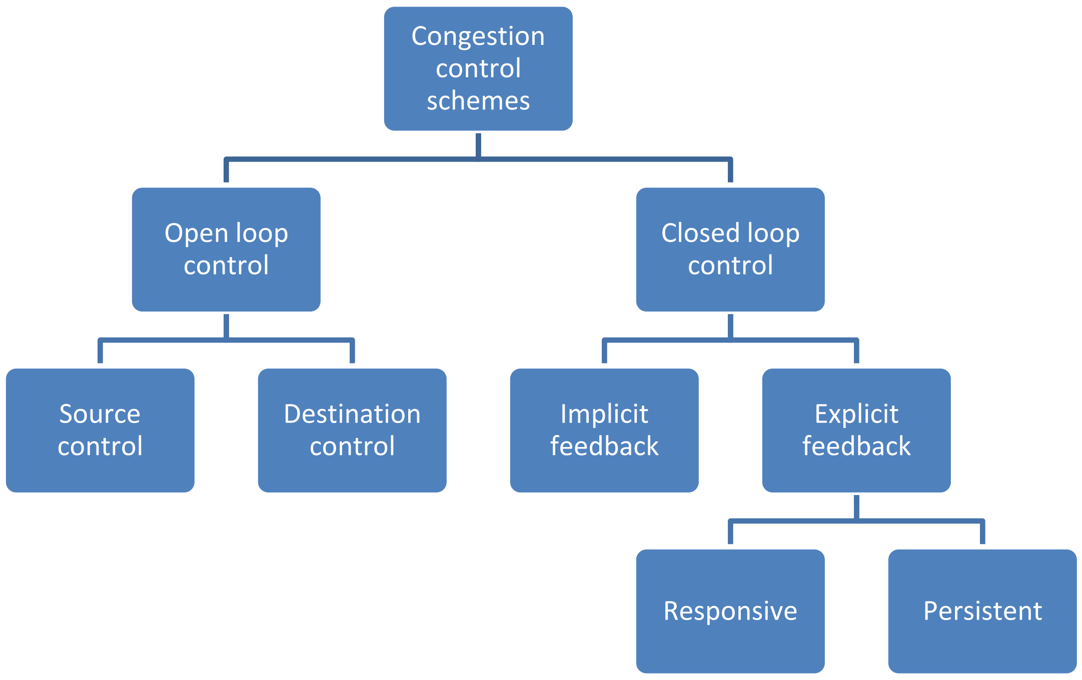


Figure 2.1: Classification of congestion control algorithms

2.4.1 Open Loop Congestion Control

Open loop control systems are systems which have no feedback. Open loop congestion control algorithms are algorithms in which the control decisions do not depend on any sort of feedback information from the congested point in the network [29]. Applying open loop control in computer networks means that these algorithms do not monitor the state of the network dynamically. A network that is based on open loop control would use resource reservation, that is, a new flow would only be admitted if the admission control entity allows it to enter [2]. Open loop schemes have a continuous activation feature and an admission handling mechanism but are not robust enough and cannot guard the network against all traffic patterns. Open loop congestion control algorithms can be classified as source control and destination control. Control algorithms at the source tend to control the flow rate at the sources. For example: the Leaky bucket algorithm [33] and the stop and go policy [34]. The destination control algorithms tend to control traffic either at the destination or the intermediate nodes. For example: the selective packet discarding schemes [35].

2.4.2 Closed Loop Congestion Control

Closed loop control systems are systems that use feedback. They make their control decisions based on feedback information to the sources. The feedback can be either global or local [29]. Global feedback means the feedback information goes from destination to source whereas local feedback means the feedback information comes only from intermediate nodes. These algorithms can dynamically monitor the network performance. The feedback involved in the closed loop algorithms can be either explicit or implicit. In explicit feedback algorithms, feedback information is sent in separate messages (explicitly). In implicit feedback there is no need to send the feedback explicitly. For example: the TCP slow start scheme [21]. The explicit feedback can be classified into two categories: persistent feedback if the feedback is available at all times; for example: the adaptive admission congestion control scheme [36] and the binary feedback scheme [37], and responsive feedback if the feedback is only available under certain conditions. The feedback information is generated in response to the traffic conditions in the network for example: the choke packet scheme [31] and the dynamic time windows algorithm [38].

2.5 TCP Congestion Control

The problem of congestion control has been the subject of extensive research over the past two decades [25]. A variety of congestion control schemes have been proposed over the years but have encountered difficulties because of the uncertainties involved in modelling the statistical behaviour of many types of traffic sources. Congestion control and traffic management in high-speed networks is further complicated by the diverse mix of traffic types and service requirements. Over the past years, TCP congestion control mechanisms have been used to effectively regulate the rates of individual connections sharing network links [2] and have been instrumental in controlling packet loss and in preventing congestion collapse across the Internet [5].

TCP is a core protocol for the Internet [2]. It provides reliable data transmission and provides a communication service at an intermediate level between an application program and IP. TCP is a connection oriented protocol which means that a connection should be established between the source and the destination before transmission. TCP is the most widely used protocol in the transport layer on the Internet [39]. One of TCP's primary functions is to match the transmission rate of the sender to that of the receiver to ensure good performance. TCP implements a sliding window scheme to perform the flow control. The receiver carries out the flow control by granting the sender a certain window length of data. The sender must not send more than the full window length without waiting for acknowledgements at any time. TCP achieves reliability by sending a segment and waiting for its acknowledgment (ACK). If the ACK does not arrive, the segment should be retransmitted. When an ACK arrives, TCP can transmit new segments not exceeding the number of bytes acknowledged. In practice, the window size is adjusted dynamically according to the available buffer space [40].

In TCP, congestion is detected by a loss of packet or time out. TCP responds to congestion by reducing the transmission rate. TCP congestion control mechanisms consist of four algorithms [21, 41-42]: slow start, congestion avoidance, fast retransmit and fast recovery.

2.5.1 TCP Slow Start

The slow start algorithm is used during the initial data transfer phase of a TCP connection. The main principle behind the slow start algorithm is to start with a small window size and to increase it slowly when acknowledgements arrive. In addition to the window already maintained by the sender, slow start adds another window called the congestion window (cwnd). Initially, the congestion window is set to one segment then after each time an ACK is received; the window is increased by one segment. The congestion window is considered as flow control imposed by the sender, while the advertised window is considered as flow control imposed by the receiver [42]. The congestion window increases exponentially by doubling the congestion window size, and thus the transmission rate, every RTT as illustrated in Figure (2.2). The sender can transmit up to the minimum of the advertised window and the congestion window.

When ACK is received the cwnd increases from one to two then two segments can be transmitted. When each of the two segments is acknowledged, the cwnd is increased to four and so on. The router will start discarding packets if the capacity of the Internet has been reached and the congestion window has become so large.

2.5.2 TCP Congestion Avoidance

In the slow start algorithm, the exponential growth of the congestion window can quickly lead to congestion unless it is checked at some point. TCP implements the congestion avoidance algorithm [21] to avoid congestion before it happens. In this algorithm, TCP sources keep track of a threshold value which is dynamically adjusted through a variable called the slow start threshold (ssthresh). The algorithm forces a linear increase of the congestion window after it reaches the ssthresh value as illustrated in Figure (2.2). When the window size exceeds the ssthresh value, TCP enters the congestion avoidance phase [43]. In this phase the congestion window increases by 1/cwnd each time an ACK is received. This means that the congestion window is effectively increased by one segment per RTT; hence the congestion window grows linearly rather than exponentially [43]. TCP detects a packet loss through the receipt of duplicate acknowledgement from the receiver or a time out occurring [42]. Each time a time out occurs, TCP assumes that a packet loss has occurred and immediately cuts its transmission rate in half by setting the ssthresh value to half the current congestion window. Then TCP must invoke slow start to get things going again by setting the congestion window to one segment [40-41]. After retransmitting the dropped packet, the TCP sender uses the slow start algorithm to increase the window from 1 to the new value of ssthresh, at this point congestion avoidance again takes over [41].

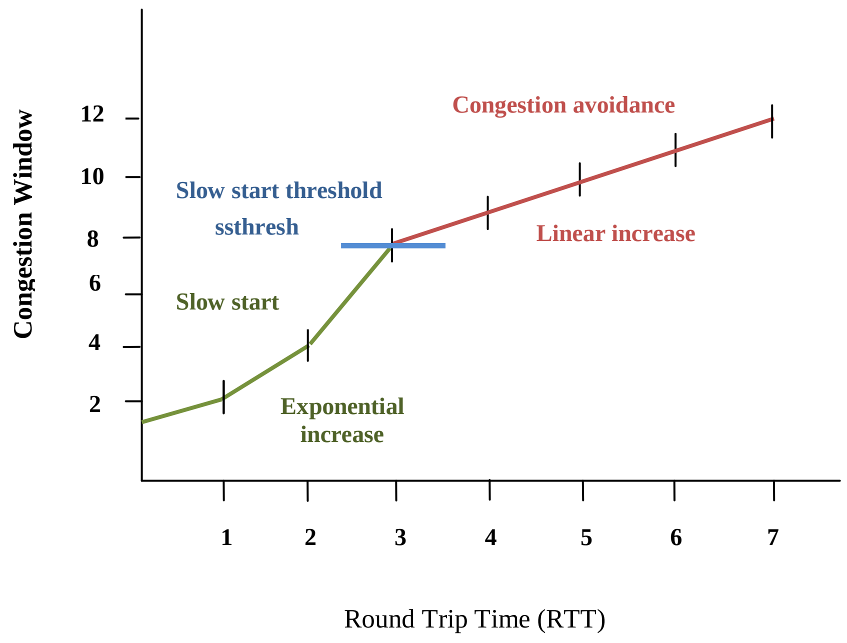


Figure 2.2: TCP congestion avoidance [40]

2.5.3 TCP Fast Retransmit

When a duplicate acknowledgment is received, the sender does not know if it is because a segment was lost or a segment was delayed and received out of order at the receiver. The purpose of this duplicate acknowledgment is to let the source know that a segment was received out of order and to tell it what sequence number is expected [42]. Since it is not known whether a duplicate ACK is caused by a lost segment or just a delayed one, TCP waits for a small number of duplicate acknowledgements to be received. If there are only one or two duplicate ACKs, it is considered as just a reordering of the delayed segments. Then a new ACK will be generated. If three or more duplicate ACKs are received in a row, this indicates that a segment has been lost. Then TCP retransmits the missing segment without waiting for a retransmission timer to expire. This process is called the fast retransmit algorithm [42].

2.5.4 TCP Fast Recovery

The fast retransmit algorithm should be used by the TCP sender to detect and repair losses, based on incoming duplicate ACKs. After the fast retransmit algorithm sends the missing segment, the fast recovery algorithm should be applied to govern the transmission of new data until a non duplicate ACK arrives. The fast recovery algorithm is an improvement that allows high throughput under moderate congestion especially with large windows [42]. The reason for not applying slow start after the receipt of duplicate ACKs is that TCP does not want to reduce the flow abruptly by going into slow start [42]. Usually the fast recovery algorithm is implemented after the fast retransmit algorithm as follows [41-42]:

- After the third duplicate ACK in a row is received, set ssthresh to half the current cwnd. Retransmit the missing segment. Set cwnd to ssthresh plus three times the segment size.
- Increment cwnd by the segment size each time another duplicate ACK is received. Transmit a segment if allowed by the new value of cwnd.
- Set cwnd to ssthresh as in the first step when new data is received and acknowledged. This ACK should acknowledge all the intermediate segments sent between the lost segment and the receipt of the first duplicate acknowledgement. TCP reduces the rate to half the rate it was when the packet was lost, in this step TCP is in congestion avoidance.

2.6 Other Protocols

Although TCP and its variants are by far the most common protocols currently used in the transport layer of the Internet, there are other transport protocols, such as User Datagram Protocol (UDP) [44], that do not involve retransmission. These are used mainly for real-time applications in which prompt delivery is much more important than accurate delivery. UDP can involve a lot of packet loss and these packets are not recovered so UDP can only be useful for services where lost packets can be tolerated. Because retransmission is eliminated at the transport layer, network resources are more efficiently used at the expense of increased datagram loss and so it would be expected that such protocols would give rise to lower levels of congestion than transport protocols such as TCP.

Similarly, protocols in the link layer, such as the High-level Data Link Control (HDLC) protocol [45-46], can also have a significant effect on congestion levels in that these protocols usually involve some form of retransmission, but this time on a link-by-link basis rather than an end-to-end basis. The primary purpose of such protocols is to make the links appear to higher layers to be free from transmission errors and so error detection and retransmission is usually involved. This can significantly increase end-to-end delay, particularly in the case of noisy links, and so may cause datagrams to time out at the link layer when TCP is used. This, in turn, involves end-to-end retransmission and so a subsequent increase in congestion.

2.7 Summary

Congestion is a complex phenomenon which occurs when the number of transmitted packets through a network approaches or exceeds the network capacity. When congestion occurs, the transmission delay for individual packets increases and packets are discarded. Many algorithms have been proposed to control congestion in order to avoid the congestion collapse problem. Congestion control algorithms are used to maintain the number of packets within the network below the level at which performance falls off dramatically.

In order to effectively regulate the rates of individual connections sharing network links, the TCP congestion control algorithms have been implemented. TCP congestion control mechanisms consist of four algorithms: slow start, congestion avoidance, fast retransmit and fast recovery. In TCP, congestion is detected by a loss of packet or time out. TCP responds to congestion by reducing the transmission rate. One problem with the TCP congestion control algorithms is that TCP sources reduce their transmission rates only after queue overflow. This is a problem since considerable time may pass between the packet drop and its detection. AQM algorithms have been proposed to prevent losses due to buffer overflow. The goal of AQM algorithms is to detect congestion early and convey congestion notification before queue overflow and packet loss occurs. The next chapter will give an overview about the queue management algorithms and will investigate the different AQM algorithms in details.

CHAPTER 3

Queue Management Algorithms

3.1 Introduction

The Internet has grown from a small data transfer oriented network with little congestion to a large multiservice network. Various types of real and non-real time traffic are transmitted over the Internet. With the growth of the Internet, it has become necessary to deploy queue management algorithms to improve QoS. Queue Management algorithms are the algorithms that manage the queue length by dropping or marking packets when necessary in order to notify sources of congestion [3]. The aim of this chapter is to survey some queue management algorithms in terms of their structure and classification.

3.2 Queue Management

Queue management algorithms play an important role in fair bandwidth allocation [47]. Queue management algorithms can be classified into two categories: Passive Queue Management (PQM) and Active Queue Management (AQM) [40]. The first category does not recognize congestion till the buffer is full and then starts dropping packets; an example of this is Drop Tail [48]. The second category drops packets probabilistically before the buffer gets full and hence makes early congestion notification, such as the RED algorithm [4].

3.3 Passive Queue Management

PQM is a traditional and simple method of controlling a buffer. PQM algorithms do not employ any preventive packet drop before the buffer gets full. All arriving packets are dropped with a probability of one if the buffer level has been reached. Drop Tail, Drop Front and Random Drop are some examples of algorithms that fall under the PQM category. These algorithms do not send early congestion notification to sources to decrease their traffic rate which means that they have only two states either 100% packet drop or no packet drop.

3.3.1 Drop Tail

Drop Tail (also known as Tail Drop) is the traditional technique for managing router queue lengths. It is the most commonly used algorithm by Internet routers because of its robustness and simple implementation [3, 49]. It tends to penalize bursty connections by discarding arriving packets when the gateway's buffer space is exhausted. The algorithm works by setting a maximum queue length, once the number of packets in the queue has reached its limit, it then drops all the subsequent arriving packets. The process of dropping packets continues until there is a packet transmitted from the queue and congestion is eliminated. Unfortunately, such a method often causes high packet delays and bursty packet drop. There are two main drawbacks for the Drop Tail algorithm; 'Lock-Out' and 'Full Queues' [3].

- Lock-Out: In some situations Drop Tail allows a single connection or a few connections to monopolize the queue space of the router. This prevents other connections from getting in the router queue. This results in a fairness problem due to the unfair sharing of network resources among the connections.
- Full Queues: Because Drop Tail drops packets only when the queue is full, it results in the router buffer being full for a long period of time. This results in long queueing delay.

Drop Tail with full queues over a long period of time causes global synchronization. With global synchronization all sources will lower their sending rate until congestion is eliminated. As a result, this period of low link utilization will cause a reduction in the overall throughput. The only way to achieve better performance with Drop Tail is to detect congestion early in such a way as to avoid the full buffer problems and to be able to absorb data bursts.

3.3.2 Drop Front

Drop Front is similar to Drop Tail in dropping packets when the buffer is full. In Drop Front, when a packet arrives at a full buffer, the packet at the front of the queue should be discarded [50]. This enables the newly arriving packets to be accommodated at the end of the queue while the packet that has been buffered at the front of the queue is dropped. This algorithm compared with Drop Tail, causes duplicate acknowledgements to be sent one whole buffer drain time earlier than in the Drop Tail case. Therefore, Drop Front can be considered as an early congestion notification algorithm which prevents the over reaction by the sources and hence decreases or prevents global synchronization. This could increase the throughput which can be considered as an advantage over the Drop Tail algorithm.

Drop Front has more advantages over Drop Tail, such as avoiding unnecessary packet loss, solving the lock-out problem, and fairness because it partially counteracts TCP’s bias against connections with larger round trip times [3]. The disadvantages of this algorithm are that the buffer is full most of the time which causes high delay, and in the case of unresponsive or fast flows there will be a very high loss rate.

3.3.3 Random Drop

Random Drop is an alternative queue discipline to Drop Tail. It was proposed to provide both congestion control and avoidance to network gateways. With Random Drop, a router drops a packet which is randomly selected from the queue when the queue is full and there is an arriving packet to the queue. Random Drop relies on the hypothesis that a packet randomly selected from the queue belongs to a particular connection with a probability matching that connection’s average transmission rate [51]. Dropping a randomly selected packet from the buffer results in users generating much traffic having a greater number of packets dropped compared with those generating less traffic. Dropping a packet randomly from the queue, results in a drop distribution proportional to the bandwidth distribution among all TCP connections [52].

There are some drawbacks for the Random Drop algorithm such as, wasting the processing time because it drops the packets which have been queued for a period of time. Also the increased complexity of the algorithm compared with Drop Tail and Drop Front, which is caused by moving the packets forward in place of the dropped packets. The major drawback is that the algorithm does not result in fairness, which was the primary goal that the Random Drop algorithm attempted to achieve [52-53]. Also Random Drop does not improve the congestion recovery behaviour of the gateways in that its behaviour is worse in a topology with single gateway bottlenecks than in those with multiple bottlenecks [51].

3.3.4 Drawbacks of Passive Queue Management

PQM algorithms do not send an early congestion warning to senders to decrease their sending rate with a view to relieving network congestion. They drop the packets after the buffer is full which increases the queueing delay and causes global synchronization. A common drawback between all the PQM algorithms is the unfairness of the connections sharing the buffer. PQM reacts only when the buffer is full which causes a high loss rate as the buffer is full most of the time. These problems however, have been observed when they are used to control congestion in buffers that carry traffic that is controlled by TCP congestion control algorithms [3].

Because of the inherent problems of PQM, the IETF recommended AQM for Internet routers to improve the performance and to avoid the problems related to the PQM algorithms [3]. Detecting congestion early is better as it can avoid performance degradation.

3.4 Active Queue Management

AQM provides preventive measures to manage a buffer to eliminate the problems associated with PQM [40]. AQM employs preventive packet drop before the buffer gets full to achieve high link utilization, low queuing delay and fair bandwidth allocation for the competing connections. AQM mechanisms attempt to avoid congestion and regulate the average queue length around a certain level by sending congestion notification such as marking or dropping packets to the sources to decrease their sending rate. They are designed to detect incipient congestion and start dropping or marking the arriving packets to avoid future congestion. Preventive packet drop provides implicit feedback to notify senders of the onset of congestion. The feedback is used by the senders to reduce their rate to relieve the level of congestion.

The gateway implements AQM to drop packets early and prevent the subsequent increase of dropping in routers to improve throughput. With TCP the sending rate increases when there is no congestion notification. With AQM dropping a packet early helps to save a large number of packets being dropped. The probability of preventive packet drop increases with increasing levels of congestion. Arriving packets are dropped randomly, which prevents all sources from backing off simultaneously and eliminates global synchronization. It also avoids the lock-out behaviour by sharing the bandwidth fairly among the competing flows.

Many AQM algorithms have been proposed in the literature; for example, ERD [52], RED [4] and ARED [6] are examples of well known AQM algorithms. The recent developments in this area have shown that the dynamic parameter configuration of existing algorithms can lead to better performance [52].

3.4.1 Early Random Drop (ERD)

ERD is an AQM that uses one threshold to detect congestion and drop the arriving packets if they exceed the threshold. The algorithm uses the instantaneous queue length which is represented by the actual number of packets in the queue. When the number in the queue reaches the buffer size, every new arriving packet is dropped. The mechanism benefits from earlier congestion notification and has shown a lower degree of global synchronization when compared with Drop Tail [52]. It is also capable of controlling aggressive users more than Drop Tail but the degree of controlling aggressive users was not satisfactory [52-53]. The fixed drop probability does not work well with dynamic traffic and it uses the actual queue size to detect the congestion, which is not suitable to detect congestion at a router for different traffics [53]. The authors in [52] stressed that in future implementations the drop probability and the drop level should be adjusted dynamically, depending on network traffic and that ERD gateways deserve further investigation.

3.4.2 Random Early Detection (RED)

In the current Internet, the TCP detects congestion only after a packet has been dropped at the gateway. Therefore, with increasingly high speed networks, it is increasingly important to have mechanisms that detect the onset of congestion while keeping throughput high but average queue sizes low. The RED gateway [4] is an AQM algorithm for routers which has been designed for networks where a single marked or dropped packet is sufficient to signal the presence of congestion to the transport layer protocol.

RED is a proactive approach to control congestion, which has been proposed to be used in the implementation of AQM to control and manage congestion in networks. RED is a congestion avoidance algorithm and, as its name implies, works on congestion at an early stage i.e. before it occurs. It tries to prevent congestion, rather than just reacting to it, by dropping packets before the gateway’s buffers are completely exhausted.

RED interacts with TCP, as TCP defines how the source rates are adjusted while AQM defines how the congestion measure is updated. Depending on the transport protocol, RED can mark a packet by dropping it at the gateway or by setting a bit in the packet header. In contrast to the Drop Tail algorithm which drops packets only when the buffer is full, RED drops or marks the arriving packets probabilistically. The probability function is a piecewise linear and increasing function of the congestion measure. This probability increases with the average queue length and the number of packets accepted since the last time a packet was dropped. RED's goal is to drop packets from each flow in proportion to the amount of bandwidth the flow uses on the output link. It does this by dropping each arriving packet with equal probability. Therefore, the misbehaving connections with the largest input rate will have the biggest drop percentage among total dropped packets [4].

The performance benefits of RED are:

- Reduces the number of packets dropped in routers.
- By maintaining the average queue size at a low level, it succeeds in reducing the delay.
- RED prevents global synchronization by having a random marking probability.
- Prevents lock-out behaviour by ensuring that for each packet arrival there is always a buffer available.
- Decreases the end-to-end delay for both responsive TCP and non-responsive real-time traffic UDP.
- Prevents a large number of consecutive packet losses even with bursty traffic.

Because RED has provided the above substantial performance benefits, the IETF has recommended the use of RED in Internet routers [3, 54].

The RED algorithm involves four parameters to regulate its performance. These parameters are: minimum threshold (min_{th}), maximum threshold (max_{th}), maximum dropping probability (max_p) and average queue length (avg). RED uses an Exponential Weighted Moving Average (EWMA) queue length as an indicator of congestion. Calculating the *avg* is done by using a low-pass filter with exponentially weighted moving average. Figure (3.1) shows the drop/mark probability versus the buffer size for the RED gateway algorithm. Packets are dropped when the average queue length falls between the two thresholds with linear probability. Packets are dropped with probability equals to one if the average queue length is greater than the maximum threshold. Figure (3.2) shows the flowchart for the RED mechanism. The RED algorithm can be summarized in Table (3.1) while Table (3.2) shows the detailed algorithm for a RED gateway.

Table 3.1: Summarized RED Algorithm

```
if $avg < min_{th}$
No packets are dropped
if $min_{th} \leq avg < max_{th}$
Mark/drop the arriving packet with probability $p_a$
else if $max_{th} < avg$
Mark/drop the incoming packet
```

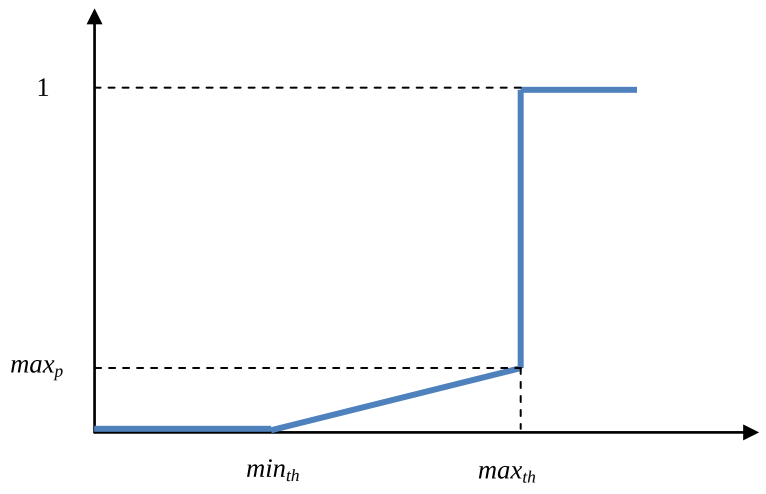


Figure 3.1: Drop/mark probability of RED

The main disadvantages of RED are:

- A lack of significant performance improvement for pure web traffic [55] and mixtures of UDP, File Transfer Protocol (FTP) and Hypertext Transfer Protocol (HTTP) traffic [56].
- The tradeoffs between stability and responsiveness of the system [57].
- The difficulties in tuning RED parameters [6, 55, 57].
- Bandwidth unfairness.

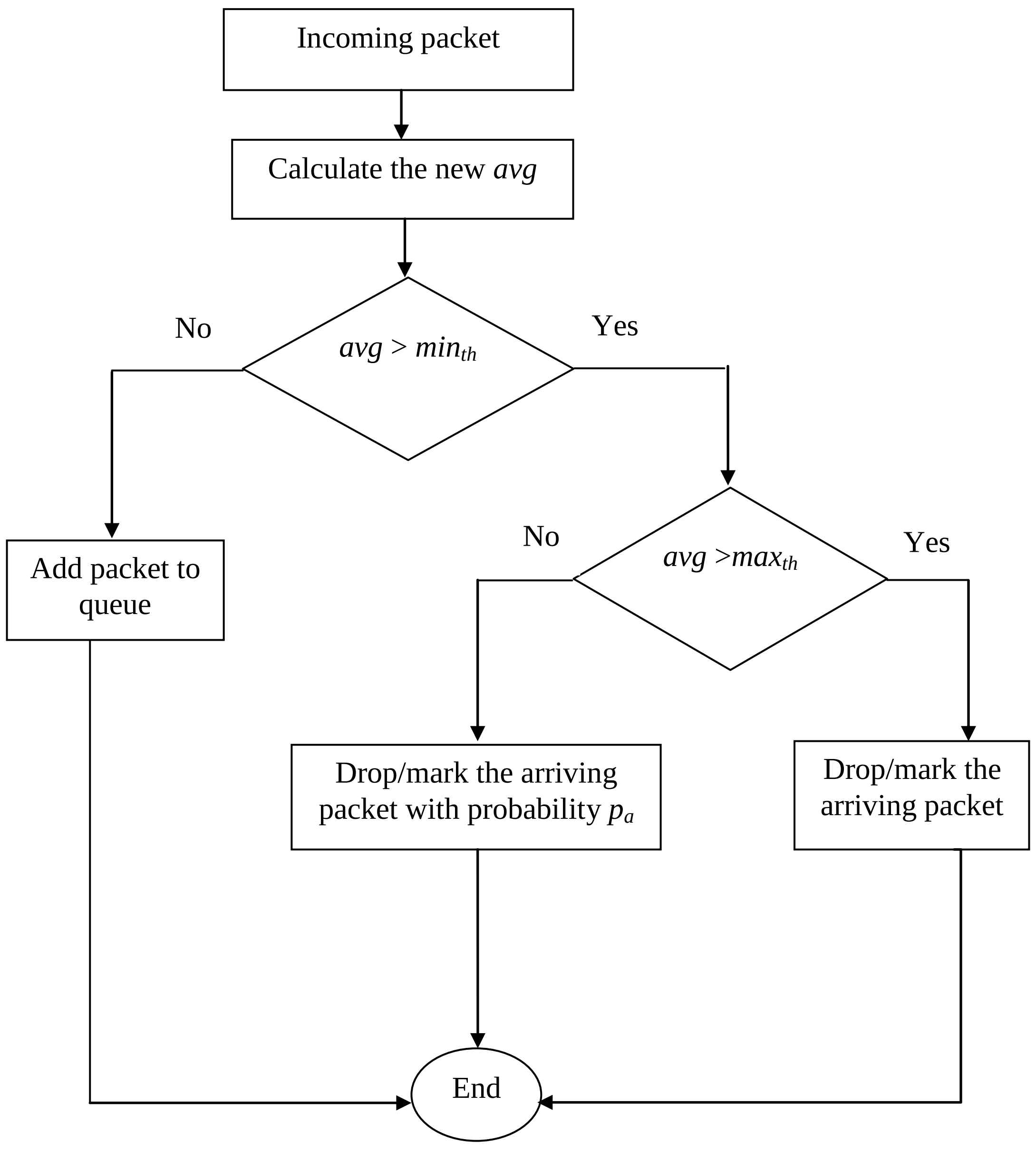


Figure 3.2: Flowchart of the RED algorithm

Table 3.2: Pseudocode for the RED Algorithm

```
Initialization:
        $avg = 0$
        $count = -1$
For each packet arrival
   Calculate the new $avg$:
            if   the queue is nonempty
                  $avg = (1 - w_q)\, avg + w_q\, q$
            else
                  $m = f\,(time - q\_time)$
                 $avg = (1 - w_q)^m\, avg$
    if   $min_{th} \leq avg < max_{th}$
          Increment $count$
          Calculate probability $p_a$:
                   $p_b = max_p\,(avg - min_{th}) / (max_{th} - min_{th})$
                   $p_a = p_b / (1 - count.p_b)$
         with probability $p_a$:
                  mark the arriving packet
                  $count = 0$
    else if $max_{th} < avg$
                  mark the arriving packet
                  $count = 0$
           else $count = -1$
when queue becomes empty
            $q\_time = time$
```

Saved Variables:

avg: average queue length

q_time: start of the queue idle time

count: packets since last marked packet

Fixed parameters:

w_q: queue weight

min_{th}: minimum threshold for queue

max_{th}: maximum threshold for queue

max_p: maximum value for p_b

Other:

p_a: current packet marking probability

q: current queue size

time: current time

$f\,(t)$: a linear function of the time t

3.4.3 Adaptive RED (ARED)

Although RED can improve TCP performance under certain parameter settings and network conditions, the RED algorithm is still susceptible to several problems such as high delay/jitter and parameter settings [5]. It has been found that one of RED's weaknesses is that the average queue length varies with the level of congestion and parameter settings. Achieving predictable average delays with RED requires constant tuning of RED's parameters to adjust to traffic conditions [6, 86]. Also, RED does not perform well when the average queue size becomes larger than max_{th}, resulting in significantly decreased throughput and increased dropping rates [6]. Again avoiding this regime would require constant tuning of the RED parameters. Several proposals for AQM schemes have been proposed to avoid these problems.

The original Adaptive RED proposal by Feng *et al* [5] retains RED's basic structure and adjusts the maximum dropping probability max_p to keep the average queue size between the two thresholds min_{th} and max_{th}. The pseudocode for the original Adaptive RED proposal by Feng *et al* is presented in Table (3.3).

Table 3.3: Pseudocode for the original Adaptive RED by Feng *et al*

```
Every ($avg$) update:
if ($min_{th} < avg < max_{th}$)
status =Between

if (($avg < min_{th}$) && (status ≠Below))
status =Below
$max_p = max_p / \alpha$

if (($avg > max_{th}$) && (status ≠Above))
status=Above
$max_p = max_p . \beta$
```

To overcome the limitations of the basic RED algorithm, the Adaptive RED (ARED) algorithm has been proposed by Floyd *et al* [6]. The new version of ARED which has been proposed by Floyd achieves the target average queue length, without sacrificing RED’s other benefits. Thus, ARED reduces both the packet loss rate and the variance in queueing delay. It appears to solve the problem of setting RED parameters.

The ARED algorithm is designed to set the w_q automatically based on the link speed and adapting max_p in response to measured queue lengths. The reason for adapting max_p is to keep the average queue size between min_{th} and max_{th} and to keep the average queue size within a target range. max_p has been adapted using an Additive Increase Multiplicative Decrease (AIMD) policy. To avoid the performance degradation of the ARED algorithm, the max_p should be restricted within the range [0.01, 0.5]. The ARED algorithm has been given in detail in Table (3.4).

Table 3.4: Pseudocode for the ARED algorithm

```
For every interval seconds:
if ($avg > target$ and $max_p \leq 0.5$)
    increase $max_p$:
    $max_p = max_p + \alpha$
else if ($avg < target$ and $max_p \geq 0.01$)
    decrease $max_p$:
    $max_p = max_p \ \beta$
```

Variables:

avg: average queue length

Fixed parameters:

interval: time; 0.5 seconds

target: target for *avg* [$min_{th} + 0.4\ (max_{th} - min_{th})$, $min_{th} + 0.6\ (max_{th} - min_{th})$]

α: increment; min (0.01, $max_p/4$)

β: decrease factor; 0.9

3.5 RED Variants with Aggregate Control

There are different variants of RED aimed at improving its performance and removing its sensitivity to parameter settings. These variants aimed at obtaining high throughput while having low delay. Classifying the RED variants can be done based on the key aspects of each algorithm. The first category called aggregate control, deals with modifying the calculation of the control variable and/or dropping function. The second category called per-flow control is concerned with configuring and setting RED's parameters.

RED uses aggregate control to determine the packet dropping probability. However, RED suffers from large queueing delay variance (jitter) because of the oscillation of queue level. It also suffers from low throughput when poorly setting parameters. In RED variants with aggregate control, all connections have the same dropping probability (i.e., the dropping probability is non-discriminative to connections) [40]. These variants address some of the limitations of the basic RED algorithm.

3.5.1 Stabilized RED (SRED)

The SRED algorithm [58] has been proposed to make the performance of the RED mechanism stable. SRED pre-emptively drop packets with a load dependent probability when the buffer is congested. SRED drops packets depending on the number of active flows and the instantaneous queue length instead of calculating the average queue length. SRED helps in stabilizing the buffer fill level, by estimating the number of active connections or flows. The final packet dropping probability in SRED is given by Equation (3.1).

$$P_{SRED} = \begin{cases} P(q) & for\ a\ large\ number\ of\ active\ flows \\ \frac{P(q)}{65536}(number\ of\ flows)^2 & for\ a\ small\ number\ of\ active\ flows \end{cases} \tag{3.1}$$

The dropping probability $P(q)$ is given by Equation (3.2) as follows:

$$P(q) = \begin{cases} max_{\mathrm{p}} & \frac{B}{3} \leq q < B \\ \frac{1}{4} max_{\mathrm{p}} & \frac{B}{6} \leq q < \frac{B}{3} \\ 0 & 0 \leq q < \frac{B}{6} \end{cases} \tag{3.2}$$

max_p is the maximum dropping probability, B is the buffer capacity and q is the instantaneous queue length.

SRED overcomes the scalability problems associated with some AQM algorithms [59] because it does not collect or analyze state information on individual flows. The simulation results of the SRED algorithm show that the normalized throughput is very low even with a few traffic flows.

3.5.2 Random Exponential Marking (REM)

Flow control algorithms are distributed algorithms to share network resources among competing sources. They often consist of two sub-algorithms: a link algorithm executed inside the network at routers, and a source algorithm executed at edge routers or host computers. The REM algorithm [8] consists of a link algorithm that probabilistically marks packets inside the network, and a source algorithm that adapts source rate to observed marking. The end-to-end marking probability is exponential, which allows a source to estimate its path congestion measure and adjusts its rate. The REM algorithm does not require per flow information. Table (3.5) shows the link algorithm of REM.

The REM algorithm has been proposed to achieve both high utilization and negligible loss and delay. Its key idea is to use a variable called ‘price’ as a congestion measure. The price variable is used to determine the marking probability. It is updated periodically based on rate mismatch and queue mismatch. Rate mismatch represents the difference between input rate and link capacity, while queue mismatch represents the difference between queue length and target. The price variable increases if the weighted sum of these mismatches is positive and decreases otherwise. The weighted sum is positive when the input rate exceeds the link capacity or when there is excess build-up to be cleared and negative otherwise.

The most important difference between RED and REM is that REM decouples congestion measure from performance measure such as queue length, delay or loss. Another difference between RED and REM is that RED has a linear marking probability while REM has an exponential marking probability, as illustrated in Figure (3.3). Despite this, REM can help stabilize the gateway queue and reduce packet loss but it has two main drawbacks. The first is configuring its parameters to ensure the desired performance. The second issue concerns hardware implementation; if REM is going to be implemented in hardware then only a few values of $\emptyset$ are easily implemented, where $\emptyset$ is a base value used in the marking probability computation [40].

Table 3.5: Pseudocode for the REM algorithm

```
Periodically
Update aggregate input rate:
        $in \leftarrow (1-\delta)\, in + (\delta)\, new\_in$
Update marking probability $m_l$:
        $p_l \leftarrow \max \quad \{p_l + \gamma(\alpha_l(buffer) + in - capacity), 0\}$
        $m_l \leftarrow 1 - \emptyset^{-p_l}$
End periodically
  while buffer $\neq 0$
        Mark packet with probability $m_l$
  End while
```

Saved variables:

in: aggregate input rate estimate

p_l: link price

m_l: current marking probability

Fixed parameters:

δ: weight in aggregate input rate estimation

γ: stepsize in price adjustment

α_l: weight of puffer in price adjustment

$\emptyset$: base in marking probability computation

Temporary variables:

new_in: current aggregate input rate

$buffer$: current buffer occupancy

$capacity$: current link capacity

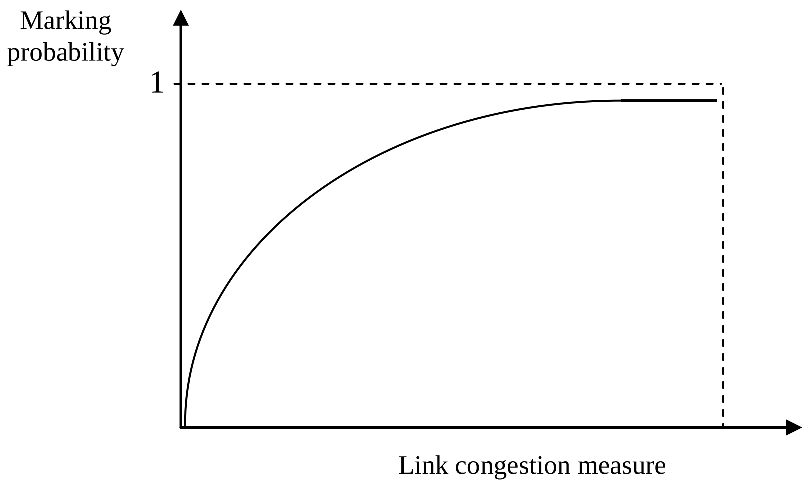


Figure 3.3: REM marking probability

3.5.3 Double Slope RED (DSRED)

Many AQM algorithms have been proposed to improve RED performance. These algorithms have attempted to modify RED parameters but have resulted in limited improvement in throughput. Zheng and Atiquzzaman proposed the DSRED algorithm [10] in order to improve the throughput and delay characteristics of RED. The idea of the DSRED algorithm is to divide the gateway buffer segment between the minimum threshold (K_l) and the maximum threshold (K_h) into two sub-segments as shown in Figure (3.4). Each segment uses a linear drop function with different slopes α and β respectively. The slopes are complementary and are adjusted by the mode selector γ. The algorithm for DSRED is shown in Table (3.6). The dropping function P_d of DSRED is given by Equation (3.3) and is presented in Figure (3.5).

$$P_d = \begin{cases} 0 & avg < K_l \\ \alpha\,(avg - K_l) & K_l \le avg < K_m \\ 1 - \gamma + \beta\,(avg - K_m) & K_m \le avg < K_h \\ 1 & K_h \le avg < N \end{cases} \tag{3.3}$$

α, β and avg are given by:

$$\alpha = \frac{2(1-\gamma)}{K_h - K_l} \tag{3.4}$$

$$\beta = \frac{2\gamma}{K_h - K_l} \tag{3.5}$$

$$avg = (1-w_q)\,avg + w_q\,q \tag{3.6}$$

where avg is the average queue length, q is the instantaneous queue length, w_q is the queue weight and N is the buffer capacity.

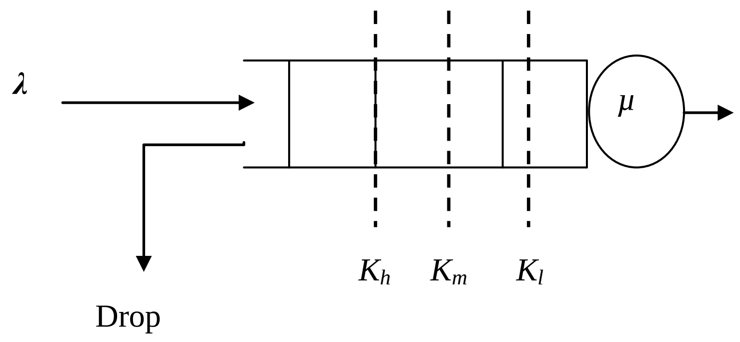


Figure 3.4: Model for DSRED buffer

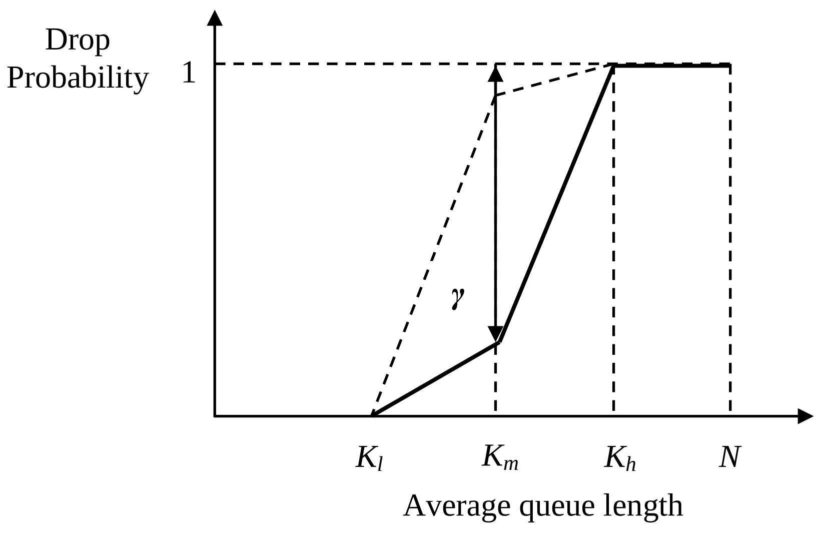


Figure 3.5: Dropping function for DSRED

Table 3.6: Pseudocode for the DSRED algorithm

```
For each packet arrival:
Calculate the average queue length ( $avg$ )
if ( $avg < K_l$ )
    No drop
else if ( $K_l \leq avg < K_m$ )
    Calculate dropping probability based on slope $\alpha$
    Drop packet

else if ( $K_m \leq avg < K_h$ )
    Calculate dropping probability based on slope $\beta$
    Drop packet
else
    Drop packet
```

Compared with RED, DSRED results in higher throughput and lower queueing delay because it adapts the level of congestion by changing the slope of the dropping function. DSRED is similar to RED in using the average queue length as the control variable; therefore, it inherits the advantages of RED.

3.5.4 BLUE

BLUE [7] is an AQM algorithm which uses packet loss and link utilization to measure network congestion and uses a marking or dropping probability p_m. If there is buffer overflow, BLUE increments p_m. If the queue becomes empty or if the link is idle, BLUE decreases p_m. This allows BLUE to learn the correct rate it needs to send back congestion notification. BLUE uses two other parameters which controls p_m over time. The first is *freez_time* which determines the minimum time interval between two successive updates of p_m. The other parameters used are d_1 and d_2. They determine the amount by which p_m is incremented when the buffer overflows or is decremented when the link is idle. The BLUE algorithm is shown in Table (3.7). Simulation and test results have shown that BLUE keeps the gateway queue stable and reduces the packet loss rate [7].

Table 3.7: Pseudocode for the BLUE algorithm

```
Upon packet loss or buffer overflow:
if ( ( now - $last\_update$) > $freeze\_time$ ) then
    $p_m = p_m + d_1$
    $last\_update$ = now
Upon link idle:
if ( ( now – $last\_update$) > $freeze\_time$ ) then
    $p_m = p_m - d_2$
    $last\_update$ = now
```

3.6 RED Variants with Per-Flow Control

The RED variants with per-flow control are concerned with configuring and setting RED's parameters. With per-flow algorithms, each connection has its own drop probability and the thresholds can be set according to the traffic type [40].

3.6.1 Flow Random Early Drop (FRED)

FRED [59] was proposed to solve the fairness problem among TCP connections. FRED provides selective dropping based on per-active-flow buffer accounting. In FRED, the loss rate is calculated by using the average queue length for each flow. FRED maintains its state only for flows for which it has packets buffered and not for all flows that traverse the Internet gateway. The FRED algorithm differs from the RED algorithm in doing the averaging at both packet arrivals and departures. FRED uses a linear dropping function which is similar to RED. So FRED can be considered as a modification to RED that improves fairness when different traffic types share a gateway. FRED provides better protection for bursty and low speed flows and is more effective in isolating unresponsive flows.

3.6.2 Class Based Threshold RED (CBT-RED)

Shared memory buffers provide efficient usage of memory and improve packet loss performance at the time of congestion but they have some technical challenges, such as speed, access and memory management [60]. CBT-RED has been proposed in [61] to solve the fairness problem when TCP traffic competes with UDP traffic. UDP sources do not respond to packets dropped by RED because UDP traffic does not employ any congestion avoidance scheme. As a result, UDP sources are getting more bandwidth than TCP sources. This results in unfairness between UDP and TCP traffic. CBT-RED solves the fairness problem between TCP and UDP traffic by setting the queue thresholds according to the traffic type and its priority. The algorithm sets a dropping threshold for the UDP traffic which is different from TCP's dropping threshold. This results in the TCP traffic being protected from the UDP traffic.

3.6.3 Balanced RED (BRED)

BRED [62] considered the problem of fair bandwidth sharing between adaptive flows (TCP) and non-adaptive flows (UDP) at Internet gateways. The BRED algorithm drops packets preventively in an attempt to penalize the non-adaptive traffic that takes more than its fair share of bandwidth. BRED regulates the bandwidth of a flow by doing per-flow accounting for the buffer active flows. The dropping decision is based on the buffer occupancy of the flow. BRED maintains a variable *qlen* which is a measure of the number of packets from flow (*i*) in the buffer. The buffer is divided into four segments, each having a different drop probability. The decision of dropping or accepting an arriving packet is based on the number of packets from that flow that already exist in the buffer. If the different flows have different packets sizes, the algorithm will be working in the byte mode and not in the packet mode.

Compared with other gateway algorithms, the performance of the algorithm achieves a more balanced allocation among different flows. BRED is very effective in ensuring fair bandwidth division among the adaptive and non-adaptive flows. The algorithm maintains minimum flow state information and is scalable. Although the algorithm can minimize the differences in the bandwidth obtained by each flow, it needs to maintain the flow states, which means that its implementation complexity is proportional to the router buffer size.

3.7 Summary

Queue management algorithms are the mechanisms that keep the network away from congestion collapse, a situation that makes the network completely non functional. The performance of TCP-based applications critically depends on the choice of queue management in the network. Queue management algorithms are divided into two categories: PQM and AQM. PQM algorithms work only after buffer overflow and do not employ any preventive packet drop before the buffer gets full; for example, Drop Tail, Drop Front and Random Drop. The second category is AQM; for example, RED and ARED. These mechanisms employ preventive packet drop before the router buffer overflows. These mechanisms avoid the problems associated with the PQM algorithms. They eliminate global synchronization and improve QoS of networks. RED is found to improve the performance of TCP/IP, and is therefore recommended by the IETF to be used in Internet routers. Studies have shown that RED has several drawbacks such as: low throughput, large delay/jitter, sensitivity to parameter settings and unfairness to connections. As a result, to improve the performance of RED, a number of variants to the original RED algorithm have been proposed such as SRED, REM, BRED, FRED and BLUE. The RED variants have improved the performance of RED. However, although these variants all have their own advantages, they also all have their own shortcomings.

CHAPTER 4

Simulation Validation

4.1 Introduction

Due to a network's complexity, simulation plays a vital role in characterizing the behaviour of any networking system [63]. Simulation is one of the most widely used techniques in Internet traffic research. With simulation it is easy to test and analyze the performance of the network. If the proposed network is not available for measurement, simulation can be considered as a convenient way to predict the performance and provide more details than an analytical model.

This chapter gives a detailed explanation of the simulation model which has been used as a basis for the rest of the simulation models throughout the thesis. The simulation used in this study is a purpose built Discrete Event Simulation (DES) implemented using Java programming [64]. Despite the large number of simulation packages available such as ns-2 [65] or Omnet++ [66], a purpose built DES has been used for its flexibility and simplicity in programming.

Most of the simulation models implemented in the thesis have been implemented as a modification to the RED algorithm. This is why the simulator used in this research has been tested and validated based on the simulation of the well known RED algorithm [4]. The validation has been carried out by comparison of results produced by the simulator in controlling the Mean Queue Length (MQL) with those reported in [4] using the same configuration and parameters for the RED mechanism as specified in [4].

4.2 Simulation Model Components

Networks used in practice are very complex and often cannot be accurately modelled for exact or approximate mathematical analysis. Simulation can be considered as an efficient way to analyse complex systems. The type of the simulator used is very vital to accurately construct the simulation model. In the simulation model, the full range of parameters and methods that are used in order to build the networking model are implied. In this study, the simulator is built using the DES [67-68] method. In DES, It is important to store the states of the system in a set of system state variables. An event list should be created to store the changes which happen to the state variables.

All DES models share a number of common components [69]. These components are initialisation routine, timing routine and event routine. The simulation begins by setting the simulation clock to zero and initialising all the state variables in the initialisation routine. The information about the next event type (either arrival or departure) can be obtained from the timing routine. The clock should be advanced at the end of the timing routine. The event routine updates the system state when a particular type of event occurs and generates future events to be added to the event list. The desired measures of performance should be produced in a report when the simulation ends.

4.3 Simulation Validation

Model validation has been carried out by running the model under the same input conditions for a well known model then comparing the results in order to test the accuracy of the proposed model. The simulation of the RED algorithm is used as the core model for other simulation programs in the thesis. Figure (4.2) represents the simulation of the RED algorithm with the recommended values as in [4]. The RED gateway parameters are represented in Table (4.1).

Table 4.1: RED configuration parameters

Parameter	Value
w_q	0.002
max_p	0.02
min_{th}	5
max_{th}	15
queue size	30
λ	6
μ	5

The main objective for the RED algorithm is to control the MQL between the two thresholds as represented in Figure (4.1). By comparing Figure (4.2) with Figure (4.1) it is noticeable that the results obtained from the simulation model show the same behaviour as the actual RED results and the MQL is fluctuating between the minimum threshold value and the maximum threshold value. This is an indication that the simulator is working successfully and controlling the MQL between the two thresholds.

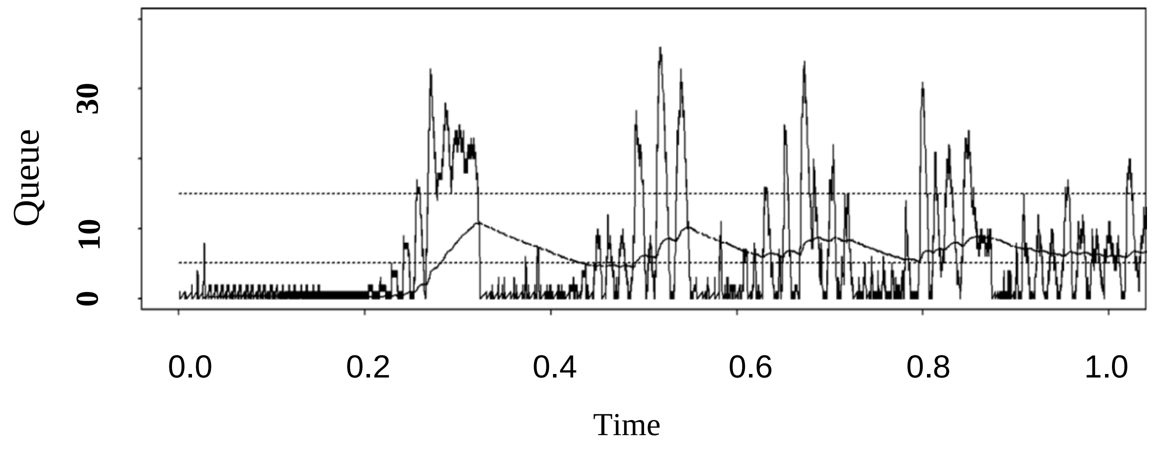


Figure 4.1: Average queue size profile of RED [4]

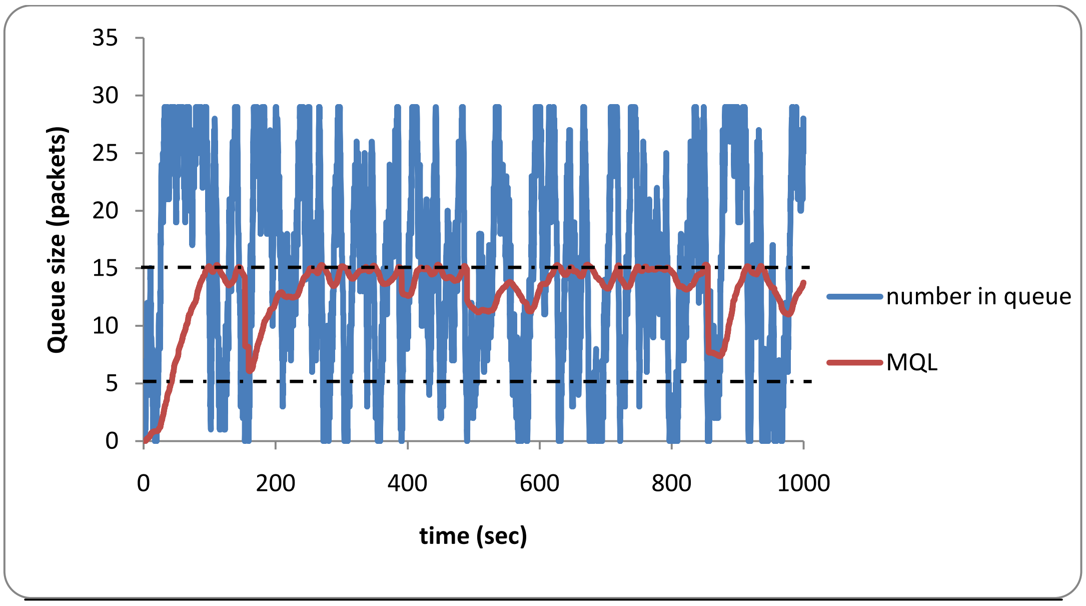


Figure 4.2: Simulation of the RED algorithm

In order to validate the consistency of the algorithm, the results obtained have been plotted with a 95% confidence interval. The simulation time has been divided into time windows, where the length of each window is 20 seconds (sec) and the value for the MQL has been measured ten times. By taking the mean value for the MQL, the 95% confidence interval E can be calculated using Equation (4.1) [70]:

$$E = \text{the mean value} \pm 1.96\sqrt{\frac{variance}{n}} \tag{4.1}$$

n represents the number of trials and equals 10 in this instance.

Figure (4.3) represents the MQL with 95% confidence intervals and this demonstrates an acceptable accuracy for the algorithm used.

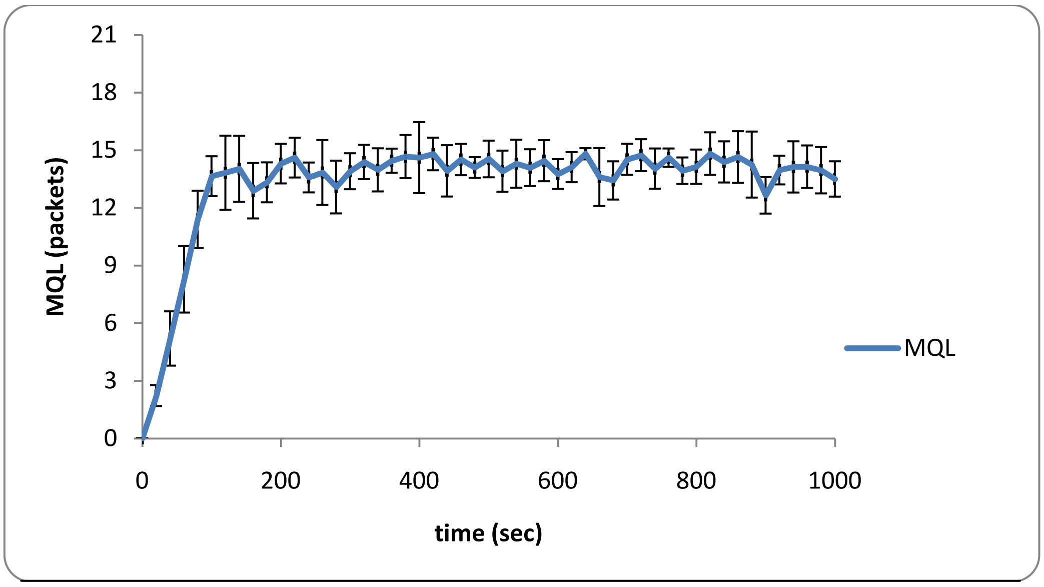


Figure 4.3: Simulation of the RED algorithm with 95% confidence intervals

4.4 Summary

Simulation plays a vital role in analyzing the performance of complex networks. Simulation programs can be used to closely replicate the networks to be modelled and in many cases can capture details that may be impossible to obtain from analytical models. This is because the latter can become intractable without introducing simplifying assumptions, especially for large networks. Since RED is the recommended queue management for routers, it has been used as the core model for evaluation throughout the thesis and the simulation of the RED algorithm has been implemented using the DES. The performance of the simulator has been validated by using the same configuration parameters as used in the original RED algorithm. The simulator accurately controlled the MQL between the two thresholds and gave a similar performance to the original RED mechanism.

Chapter 5

Dual Class RED

5.1 Introduction

Today's networks require the integration of a variety of data flows into buffers that may not be precisely suited to handle the requirements and characteristics of the traffic. Network switches send and receive real-time traffic as well as non real-time traffic. Each type of traffic has different scheduling requirements. Also, some traffic may carry higher priority than the other. The queues located at routers and switches must have the ability to handle these traffic types especially in a shared buffer.

This chapter focuses on examining the performance of the RED mechanism under two streams of traffic, under different traffic conditions. The developed algorithm is called Dual Class RED (DC-RED). It shows the effect of varying the parameters of one class on the other. It assigns two sets of thresholds per class. The effect of varying the thresholds positions on the marginal performance of each class as well as the overall performance has been investigated in terms of mean delay and packet dropping probability.

Although it is obvious that there must be some degree of dependency between multiple classes of traffic in a shared buffer and the use of separate buffers for each class in such a situation has long been accepted as a necessary requirement, to the best of our knowledge no previous studies have reported specific quantitative results for this scenario. It is therefore the aim of this chapter to conduct such a study so that the results can give some insight into the interdependencies between multiple classes of traffic in a shared buffer.

5.2 DC-RED Model

The DC-RED model is based on the RED model which is currently the most popular AQM mechanism for the Internet. The model considered is a First-In First-Out (FIFO) single server queuing system with two classes of traffic as represented by Figure (5.1). The arrival rate from each class follows a different Poisson process where class1 has average arrival rate λ_1 and class2 has average arrival rate λ_2. The service time is exponentially distributed with average service rate μ. The thresholds for class1 traffic are (L_{A1}, L_{A2}) and the thresholds for class2 traffic are (L_{B1}, L_{B2}). The parameters used in simulating the DC-RED model are summarized in Table (5.1). The values of the arrival rates (λ_1 and λ_2) and the service rate (μ) have been chosen to attain certain conditions:

- $\lambda_1 + \lambda_2 < \mu$
- $\lambda_1 + \lambda_2 = \mu$
- $\lambda_1 + \lambda_2 > \mu$

Table 5.1: DC-RED configuration parameters

Parameter	Value
w_q	0.002
max_p	0.1
$L_{A1}=L_{B1}$	5
$L_{A2}=L_{B2}$	10
queue size	50
λ_1	6, 9, 12
λ_2	3, 6, 9
μ	6, 12, 18

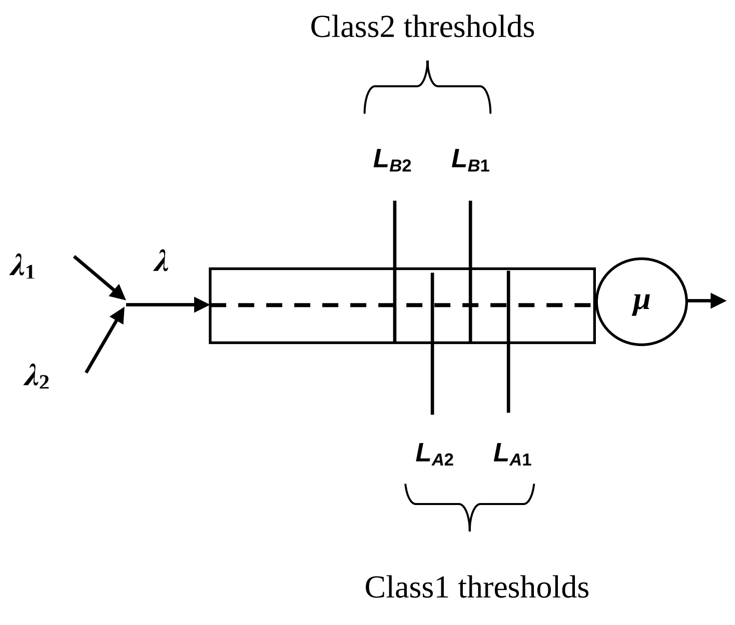


Figure 5.1: Single buffer with two thresholds per class

In order to perform a steady state analysis of the system, the performance of the model has been measured for four different cases. The different cases of the model have been implemented by fixing the thresholds of class1 and varying only the positions of the class2 thresholds as elucidated in Figure (5.2). The case where the two sets of thresholds for both classes are identical was first considered. By keeping class1 thresholds fixed where L_{A1}=5 and L_{A2}=10 and moving only class2 thresholds towards the end of the queue, the other cases of the buffer with the different threshold positions have been obtained. The separation between the two thresholds in each class is the same (L_{B2}-L_{B1}=L_{A2}-L_{A1}=5) and the distance between the thresholds of the two classes is the same (L_{B1}-L_{A1}=L_{B2}-L_{A2}).

The following relations have been used to calculate the dropping probability for each class:

$$P_{drop1} = \max{}_p \frac{(avg - L_{A1})}{(L_{A2} - L_{A1})} \tag{5.1}$$

$$P_{drop2} = \max{}_p \frac{(avg - L_{B1})}{(L_{B2} - L_{B1})} \tag{5.2}$$

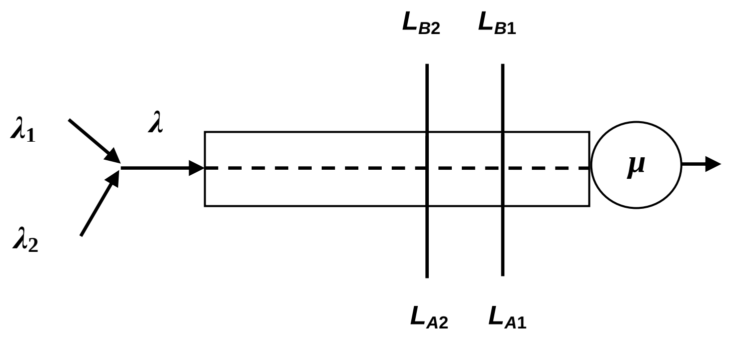
max_p is the maximum dropping probability for both classes and equals (0.1). The average queue length (*avg*) is calculated using the EWMA as in the RED algorithm [4]. By using Equations (5.1) and (5.2) it is noticeable that:

- Both classes will have the same dropping probability when they have the same minimum and maximum threshold values (L_{A1}=L_{B1}) and (L_{A2}=L_{B2}) as in Figure (5.3).
- The dropping probability for class1 increases linearly from the minimum threshold value (L_{A1}) to the maximum threshold value (L_{A2}) as in Figure (5.4).
- The dropping probability for class2 increases linearly from the minimum

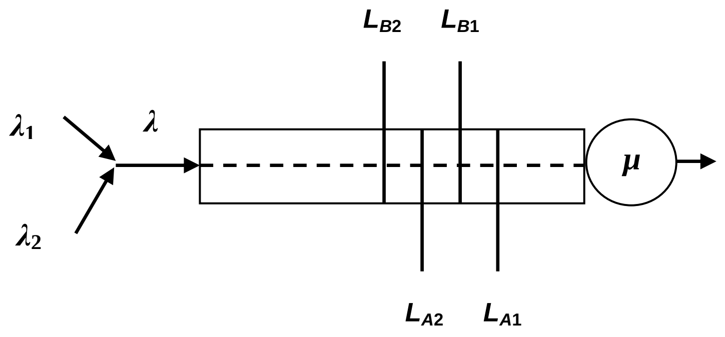
threshold value (L_{B1}) to the maximum threshold value (L_{B2}) also shown in Figure (5.4).

Class1 packets will be dropped according to dropping probability P_{drop1} when they fall between class1 thresholds (L_{A1}, L_{A2}). If the MQL exceeds the second threshold for class1 (L_{A2}), all the corresponding packets from class1 will be dropped. Class2 packets will be dropped according to dropping probability P_{drop2} when they fall between class2 thresholds (L_{B1}, L_{B2}). If the MQL exceeds the second threshold for class2 (L_{B2}), all the arriving packets from class2 will be dropped. The detailed algorithm for the DC-RED model has been explained in detail in Table (5.2).

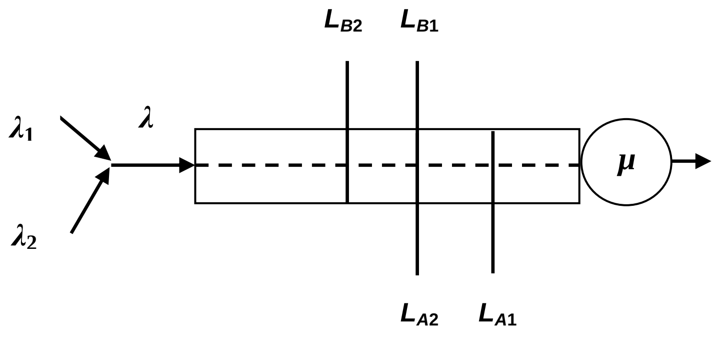
(Case 1)



(Case 2)



(Case 3)



(Case 4)

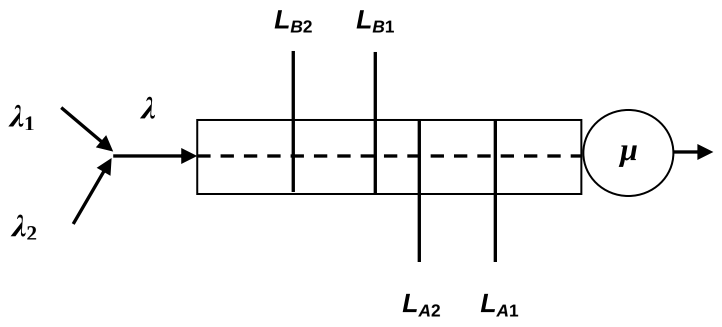


Figure 5.2: Different cases for the single buffer with two thresholds per class

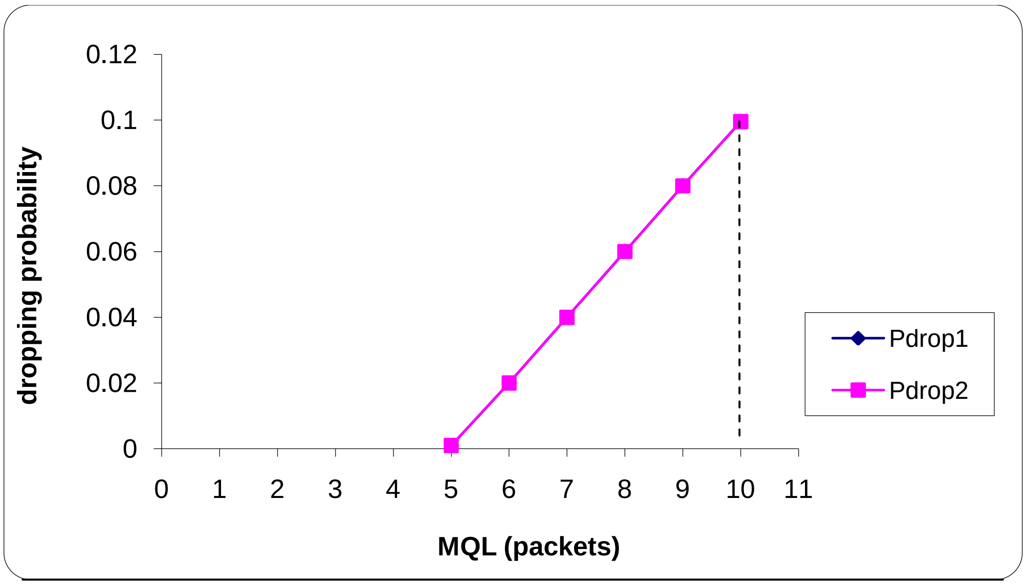


Figure 5.3: Dropping probability for both classes when they have the same minimum and maximum thresholds values ($L_{A1}=L_{B1}$) and ($L_{A2}=L_{B2}$)

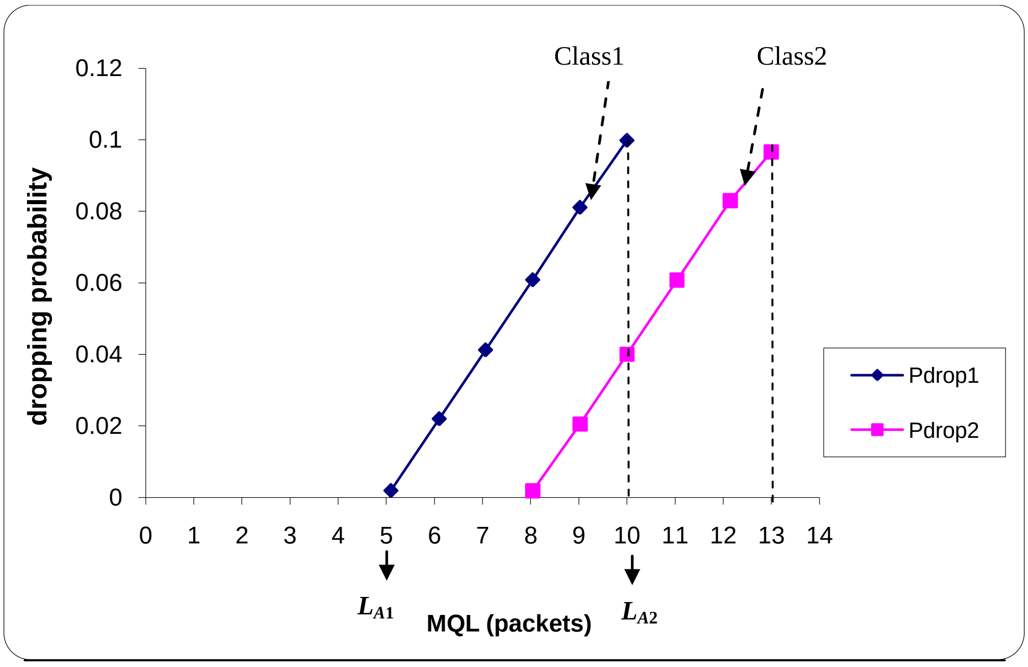


Figure 5.4: Dropping probability for both classes when $L_{A1}<L_{B1}<L_{A2}$

Table 5.2: Pseudocode for the DC-RED algorithm

```
Initialization
For each packet arrival
{
   Calculate the new average queue length ($avg$)
   $avg = (1-w_q)\ avg + w_q\ q$
   if ($avg < L_{k1}$)
       add the packet to the queue
   else if (($avg \geq L_{k1}$) && ($avg \leq L_{k2}$))
          calculate the dropping probability $P_{dropj}$
          $P_{dropj} = (avg-L_{k1})\ (max_p / (L_{k2}-L_{k1}))$
          drop the arriving packet with dropping probability $P_{dropj}$
   else if ($avg > L_{k2}$)
          drop the arriving packet
}
```

Saved Variables:

avg: average queue size

q: Instantaneous queue size

Fixed parameters:

w_q: queue weight

L_{k1}: minimum threshold for class k and k=A, B

L_{k2}: maximum threshold for class k and k=A, B

max_p: maximum value for P_{dropj} and j=1, 2

5.3 Performance Analysis

In this section the performance of the DC-RED model is investigated by varying the difference between the class1 and class2 thresholds. The performance metrics are presented by varying the two thresholds for class2 and examining the effect on class 1 in terms of average delay and packet dropping probability. Also, the effect of changing the parameters of one class on the other class and on the overall performance is examined. For each individual class the average queue length has been calculated as in the RED algorithm. The packet dropping probabilities have been calculated using Equations (5.1) and (5.2). The packet loss probability (when packets are lost due to buffer overflow) is not presented as one of the performance metrics of the proposed model since it was found too small to be measured as the buffer never fills up to its limit to cause any remarkable loss. Intensive simulation tests have been done in order to evaluate the performance under different conditions, like changing the values for the service rate μ or the arrival rate from each class, λ_1 or λ_2.

5.4 Marginal Mean Delay Analysis

Due to the increase in network services including real-time video and audio applications, mean delay has become one of the most important performance metrics for many Internet applications and is an important QoS metric. The mean delay has been used to measure the performance of proposed model. The mean delay can be calculated as the average time a packet spends in the system, which is the time spent waiting in the queue plus the service time.

5.4.1 The Effect of the Service Rate on the Marginal Mean Delay

In this section the effect of varying the service rate μ on the marginal mean delay is examined. The values of the arrival rate from both classes are kept fixed at the following values λ_1=9 and λ_2=3. The value for L_{A1}=5 and for L_{A2}=10.

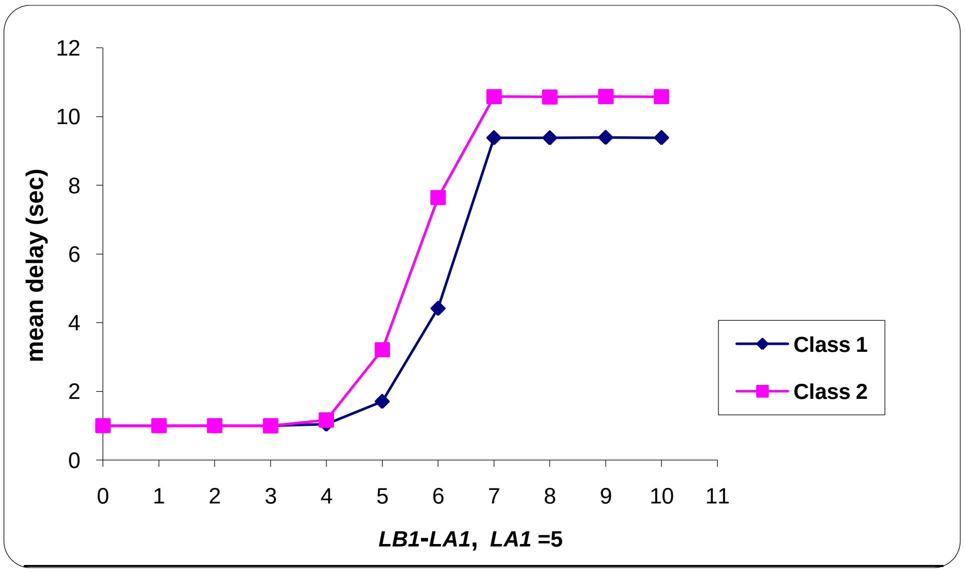


Figure 5.5: Marginal mean delay at μ=6, λ_1=9 and λ_2=3

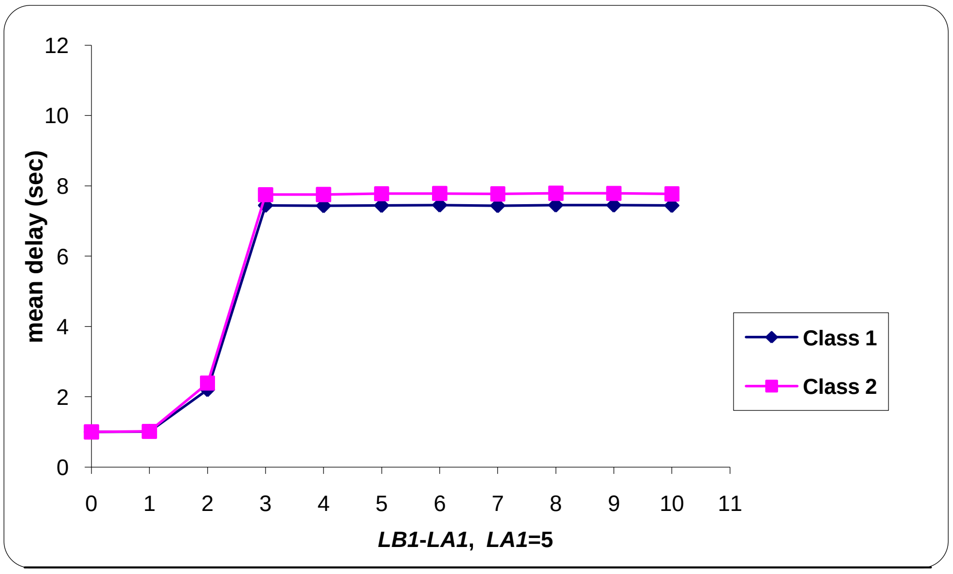


Figure 5.6: Marginal mean delay at μ=12, λ_1=9 and λ_2=3

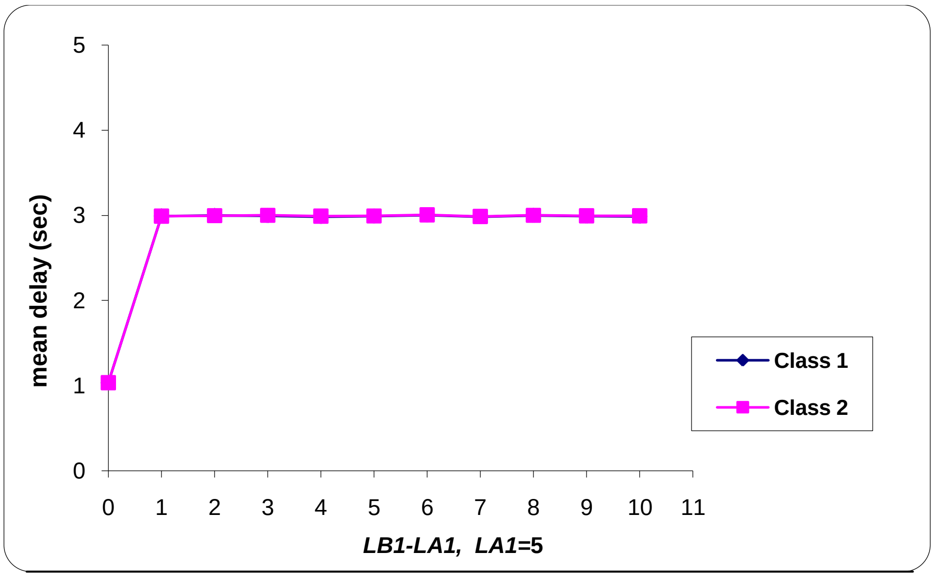


Figure 5.7: Marginal mean delay at μ=18, λ_1=9 and λ_2=3

Figure (5.5) indicates that by increasing the separation between class2 and class1 thresholds, this makes the delay for class2 higher than the delay for class1. The reason is by moving class2 thresholds the buffer then accepts more packets from class2 rather than class1. This means that packets from class2 spend more time in the queue and so suffer from more delay than the packets from class1. At a threshold difference of 5 everything from class1 is dropped after the second threshold L_{A2}=10, so class1 delay levels off. Now the effective arrival rate is only λ_2 since all of λ_1 packets are dropped. Since $\lambda_2=3 < \mu$ then class2 delay also does not move up any further, since queue does not build up beyond its steady state value, even if class2 thresholds are moved back further.

By increasing the value of μ the delay for both classes will decrease as shown in Figure (5.6) but class1 still has lower delay than class2. When μ=18, which is a very high value, it is noticed that the results are superimposed and identical and so appear merged as shown in Figure (5.7). The performance of both classes is identical and the delay has reached a very low value, this is because the service rate is higher than any of the arrival rates and also higher than the total arrival rate (λ), where

$$\lambda = \lambda_1 + \lambda_2 \tag{5.3}$$

5.4.2 The Effect of Class1 Arrival Rate on the Marginal Mean Delay

From Figures (5.8), (5.9) and (5.10) it is noticeable that by increasing the number of arrivals from class1 the mean delay for both classes increases, even though class2 arrival rate is fixed. This is because increasing any of the arrival rates will affect the total arrival rate, as indicated by Equation (5.3). The two classes will have the same delay value when λ_1=6 as shown in Figure (5.8), then increasing λ_1 will give higher delay values for class1 and class2 as shown in Figures (5.9) and (5.10). Increasing the separation value between the two classes will allow more packets from class2 which will cause dramatic increase in both classes' marginal mean delay and makes class2 delay higher than class1 delay at the same threshold separations.

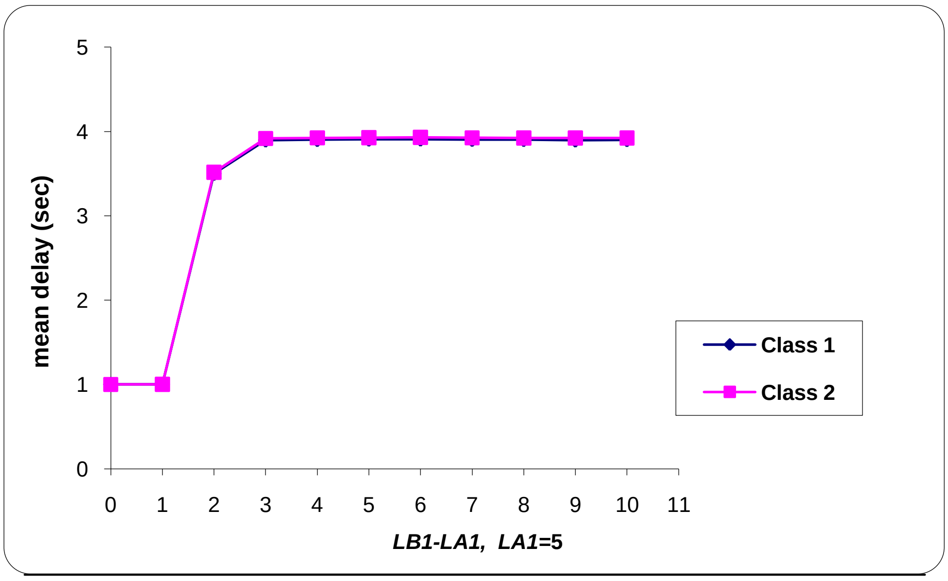


Figure 5.8: Marginal mean delay at λ_1=6, μ=12 and λ_2=3

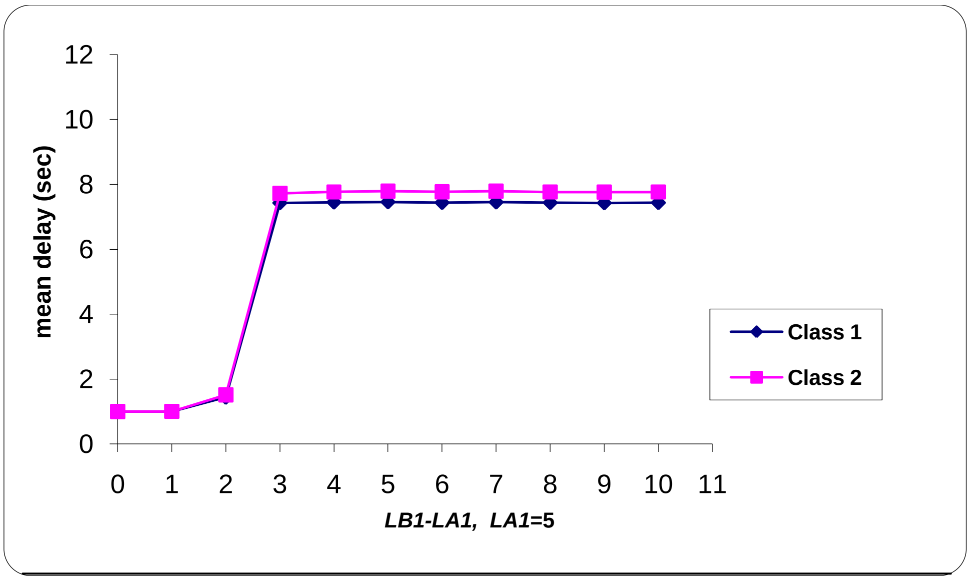


Figure 5.9: Marginal mean delay at λ_1=9, μ=12 and λ_2=3

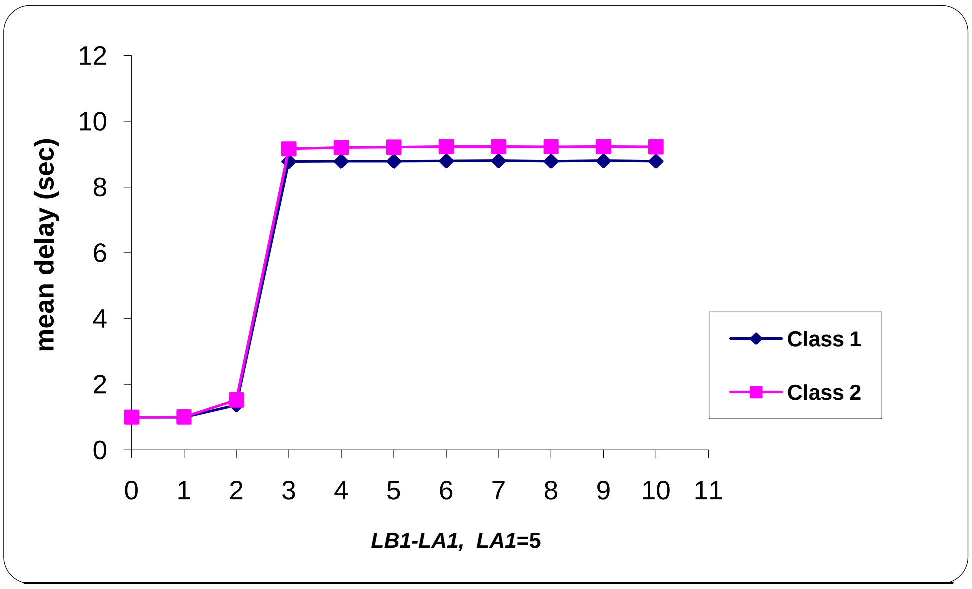


Figure 5.10: Marginal mean delay at λ_1=12, μ=12 and λ_2=3

5.4.3 The Effect of Class2 Arrival Rate on the Marginal Mean Delay

By fixing the values for μ at 12 and λ_1 at 6 as in Figures (5.11), (5.12) and (5.13). It is noticed that increasing λ_2 will cause an increase in the marginal mean delay for each class but this effect can be noticed at wider separation between (L_{B1}, L_{A1}). When λ_1 is fixed and only λ_2 is increasing the delay for both classes should be increasing but because L_{B1} is increasing allowing the acceptance of more packets from class2 this will take longer for the queue to be filled with packets from class2 and this causes the mean delay to increase by increasing the value of L_{B1}.

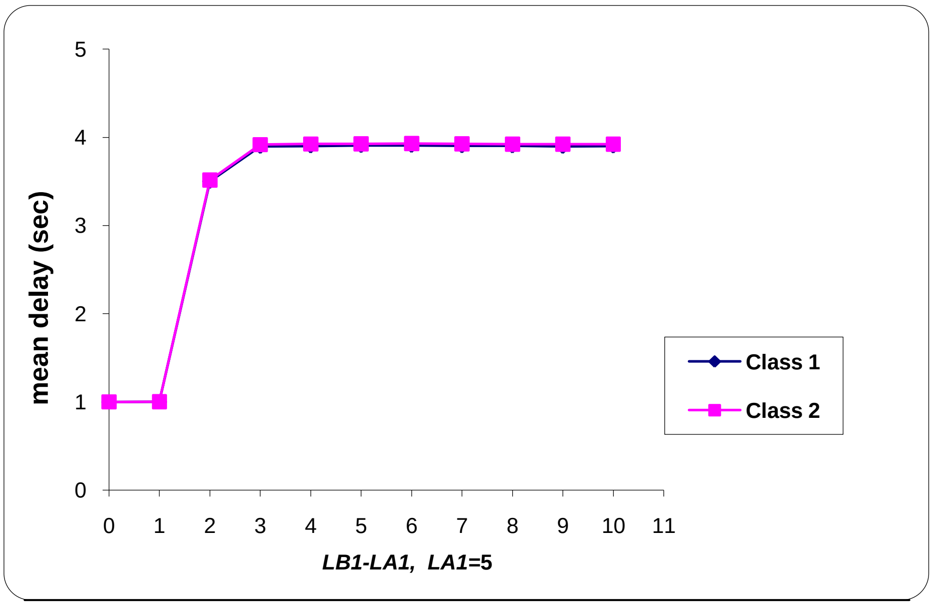


Figure 5.11: Marginal mean delay at λ_2=3, μ=12 and λ_1=6

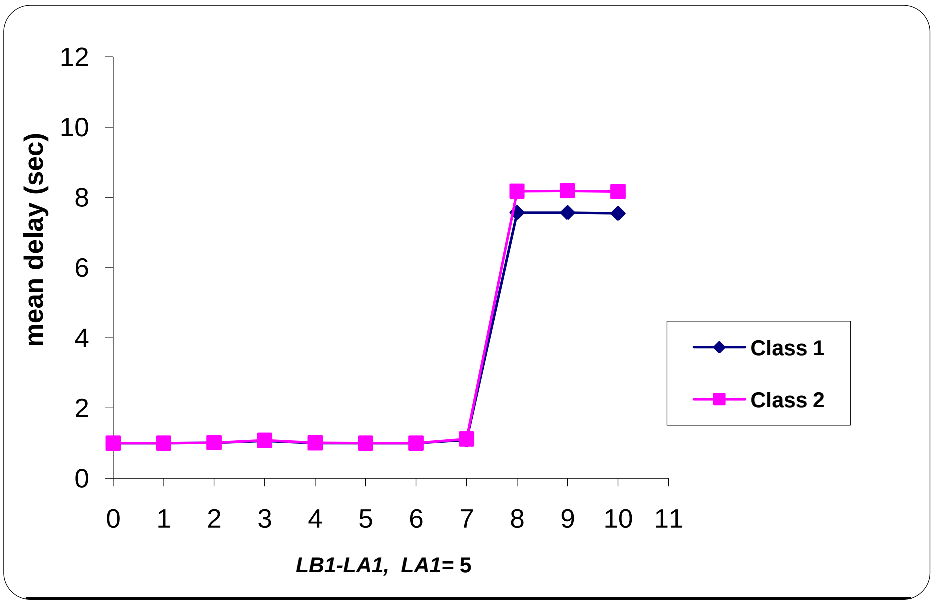


Figure 5.12: Marginal mean delay at λ_2=6, μ=12 and λ_1=6

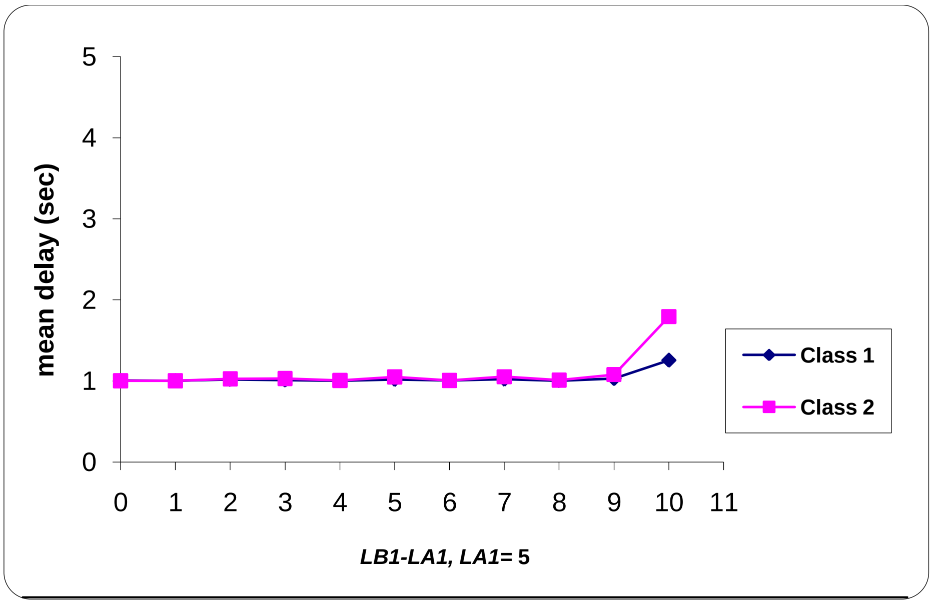


Figure 5.13: Marginal mean delay at λ_2=9, μ=12 and λ_1=6

5.5 Marginal Dropping Probability Analysis

By applying the algorithm stated in Table (5.2) class1 packets are dropped when the MQL falls between class1 thresholds L_{A1} and L_{A2}. If the MQL exceeds the maximum threshold L_{A2}, all the packets which belong to class1 should be dropped. The same process happens for class2 as class2 packets will be dropped if the MQL falls between class2 thresholds L_{B1} and L_{B2}. If the MQL exceeds L_{B2} all the arriving packets from class2 should be dropped.

5.5.1 The Effect of the Service Rate on the Dropping Probability

Figure (5.14) indicates that increasing L_{B1} will affect the number of packets dropped from both classes, as expected; by increasing class2 thresholds the buffer is accepting more packets from class2 which makes the marginal dropping probability for class2 decrease. On the other hand class1 packet dropping probability suffers from a slight increase and this is because class1 thresholds are fixed and the arrival rate from class1 is higher than the service rate μ. Increasing the value of μ will give lower dropping probability for both classes, and it is clear from Figures (5.15) and (5.16) this is because μ is greater than any of the arrival rates or the total arrival rate.

It is clear from Figures (5.14), (5.15) and (5.16) that moving L_{B1} will cause sharp decrease in the marginal dropping probability for both classes. When the difference between L_{B1} and L_{A1} is high the dropping probability for class2 goes to zero and for class1 is a very low value but higher than class2 for the same threshold settings. Figure (5.16) shows that when the service rate is much higher than the total arrival rate both classes behave the same in that the dropping probability is nearly zero over the whole range of threshold separations.

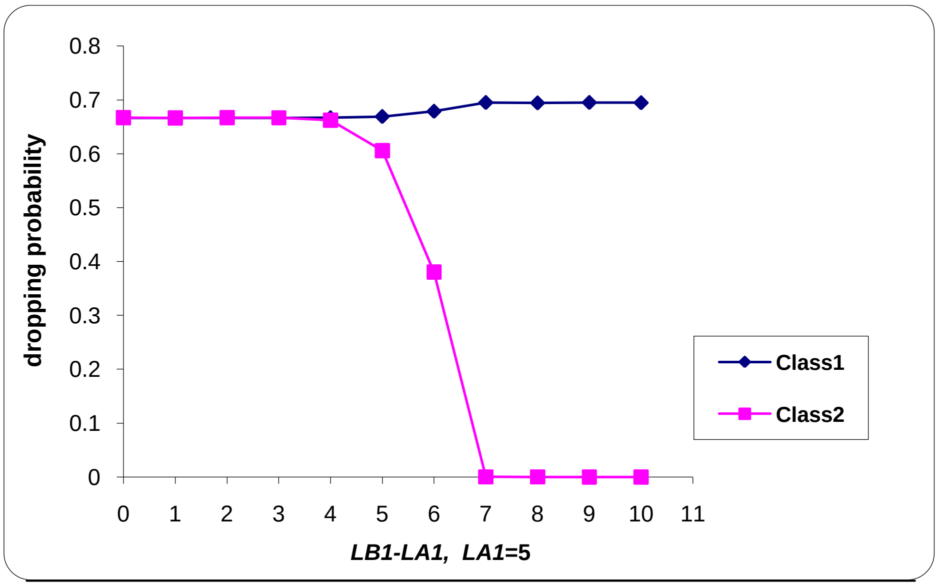


Figure 5.14: Marginal dropping probability at μ=6, λ_1=9 and λ_2=3

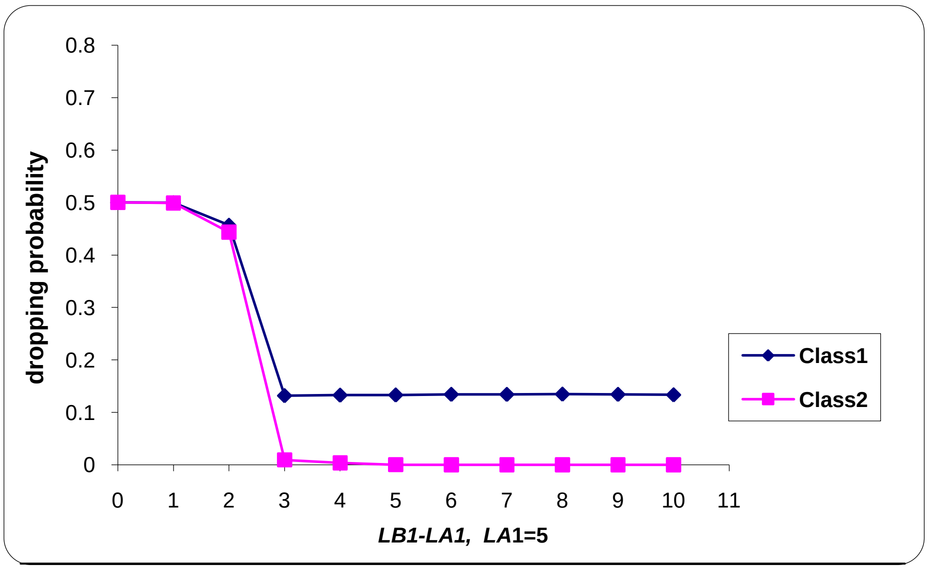


Figure 5.15: Marginal dropping probability at μ=12, λ_1=9 and λ_2=3

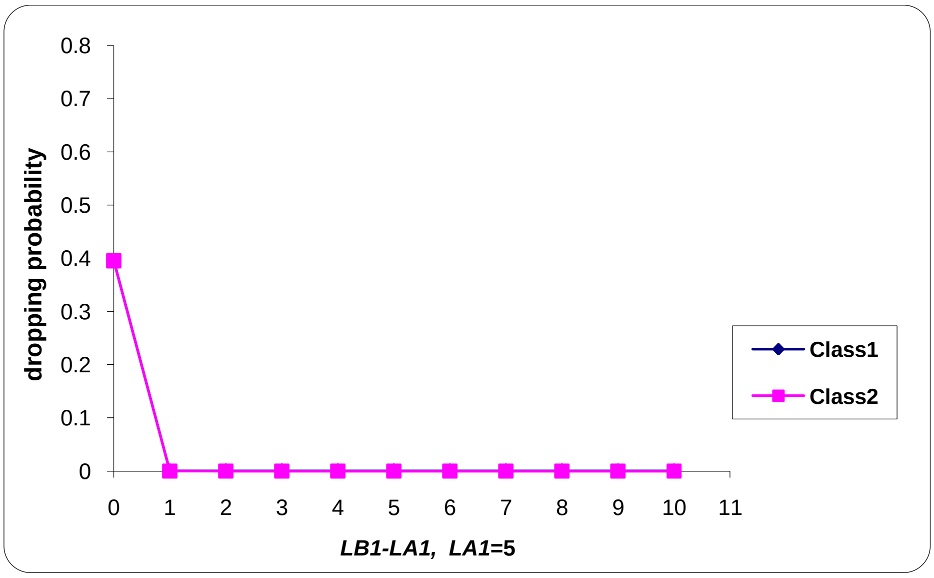


Figure 5.16: Marginal dropping probability at μ=18, λ_1=9 and λ_2=3

5.5.2 The Effect of Class1 Arrival Rate on the Dropping Probability

It is noticeable from Figures (5.17), (5.18) and (5.19) that increasing λ_1 will increase the values of the marginal dropping probability for both classes. These figures also indicate that by increasing the difference between the two classes' thresholds the packet dropping probability for each class will decrease dramatically until it reaches zero for class2 and a very low value for class1. Increasing the separation between L_{B1}, L_{A1} as well as increasing the value of λ_1 gives higher loss for class1 than for class2.

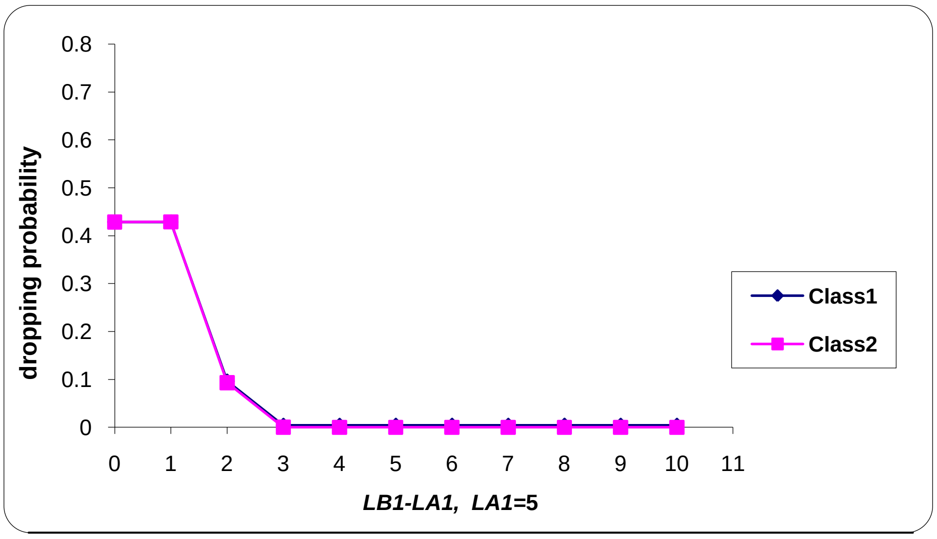


Figure 5.17: Marginal dropping probability at λ_1=6, μ=12 and λ_2=3

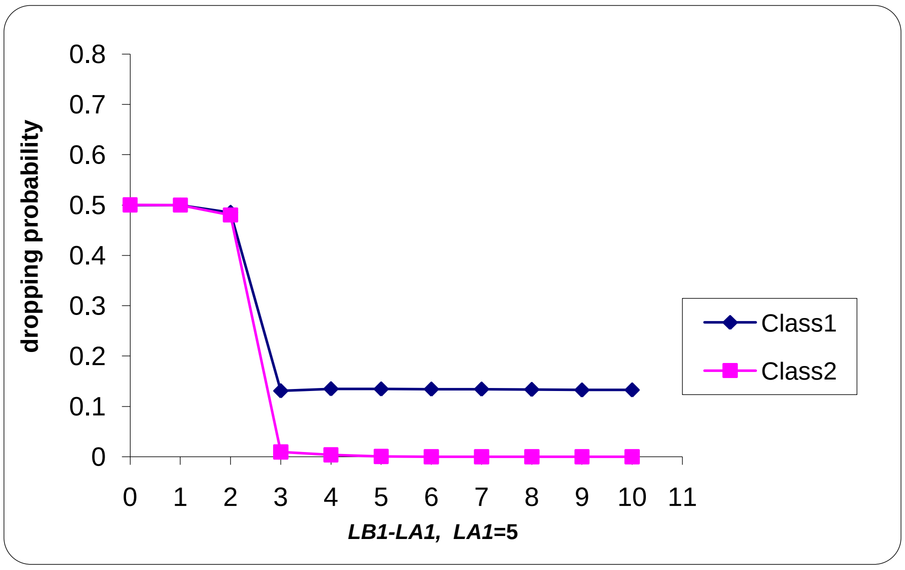


Figure 5.18: Marginal dropping probability at λ_1=9, μ=12 and λ_2=3

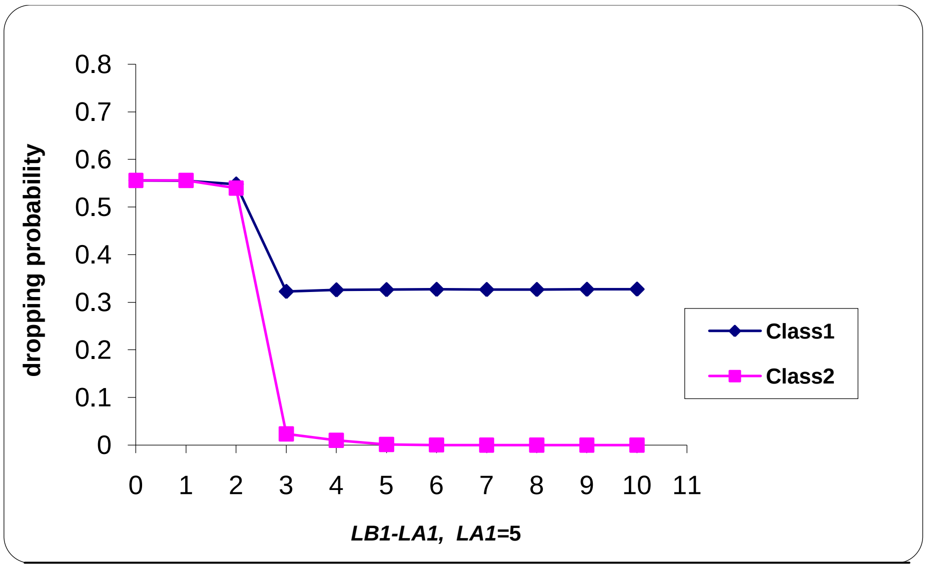


Figure 5.19: Marginal dropping probability at λ_1=12, μ=12 and λ_2=3

5.5.3 The Effect of Class2 Arrival Rate on the Dropping Probability

As it has been noticed before, increasing any of the arrival rates and at the same time moving class2 thresholds will affect the marginal dropping probability for both classes. So increasing λ_2 from 3 to 9 causes an increase in the dropping probability for both classes. Figures (5.20) and (5.21) show that lower values for the marginal dropping probability for both classes can be obtained by using a wide separation between the two classes' thresholds. Figure (5.22) shows that when class2 packets arrive at a very high rate, even higher than class1 arrival rate, both classes will suffer from a very high packet dropping probability. But by moving class2 thresholds, the buffer will be able to accept more packets from class2 but still cannot accept any more packets from class1, which makes its dropping probability still very high.

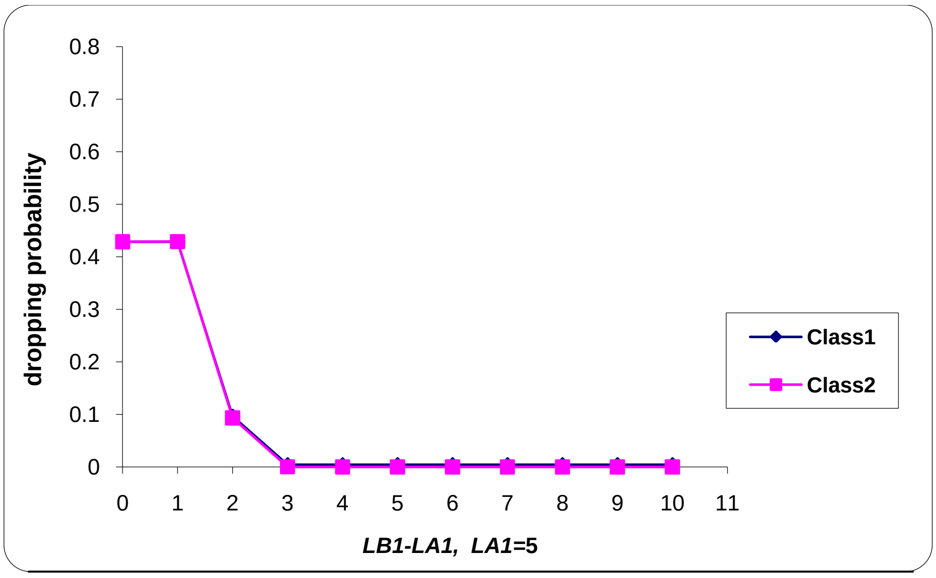


Figure 5.20: Marginal dropping probability at λ_2=3, μ=12 and λ_1=6

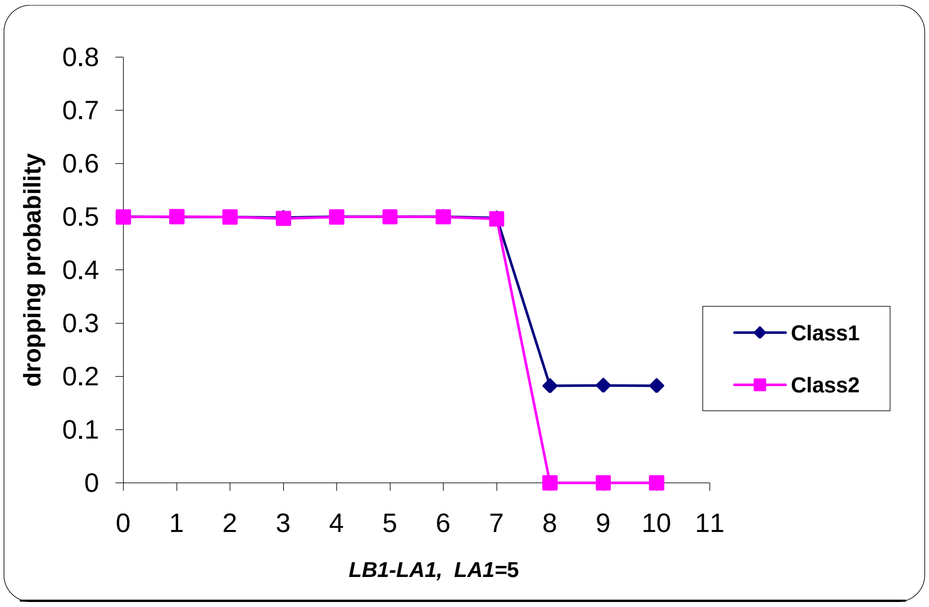


Figure 5.21: Marginal dropping probability at λ_2=6, μ=12 and λ_1=6

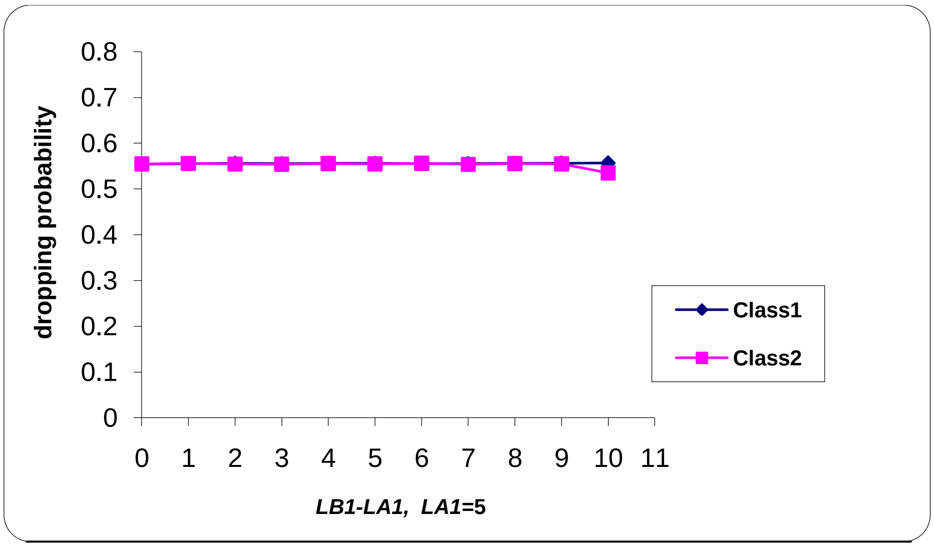


Figure 5.22: Marginal dropping probability at λ_2=9, μ=12 and λ_1=6

5.6 Overall Mean Delay Analysis

This section represents the effect of moving thresholds on the overall mean delay at different values of the arrival rate from class1. As expected by increasing the arrival rate the overall mean delay increases especially while maintaining class1 thresholds fixed. The effect of moving thresholds on the overall mean delay only takes effect at lower separation values till the delay reaches its highest value then the trend of the delay remains fixed as in Figure (5.23).

It is noticeable that beyond the second threshold L_{B1} all packets are dropped which results in the characteristic saturation of mean delay at specific levels, which is a feature of all mean delay graphs.

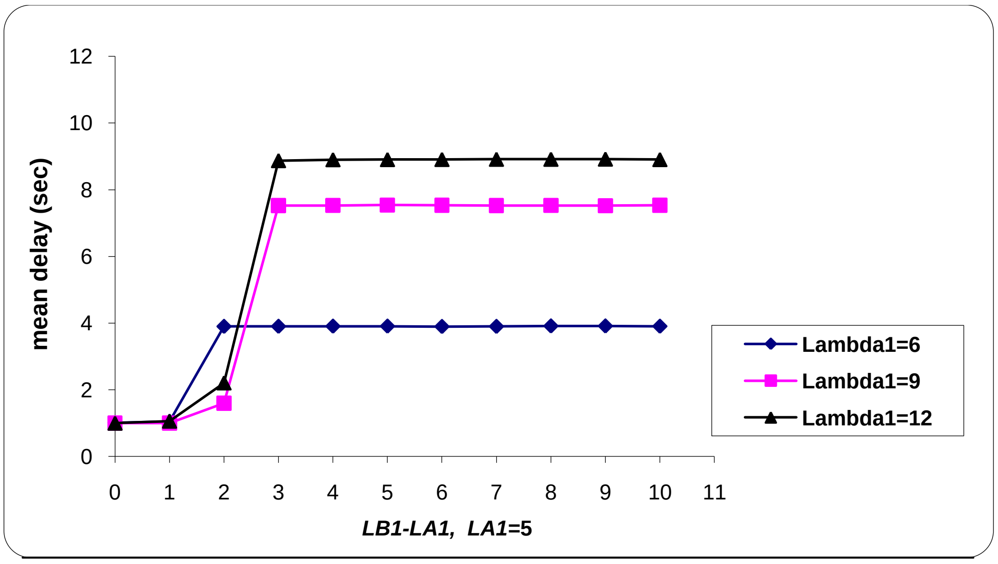


Figure 5.23: The overall mean delay at different values of λ_1 where μ=12 and λ_2=3

5.7 Overall Dropping Probability Analysis

Figure (5.24) shows that increasing class1 arrival rate causes a large increase in the overall dropping probability. By increasing L_{B1} the overall dropping probability will decrease although the lower dropping probability could be achieved at lower arrival rates.

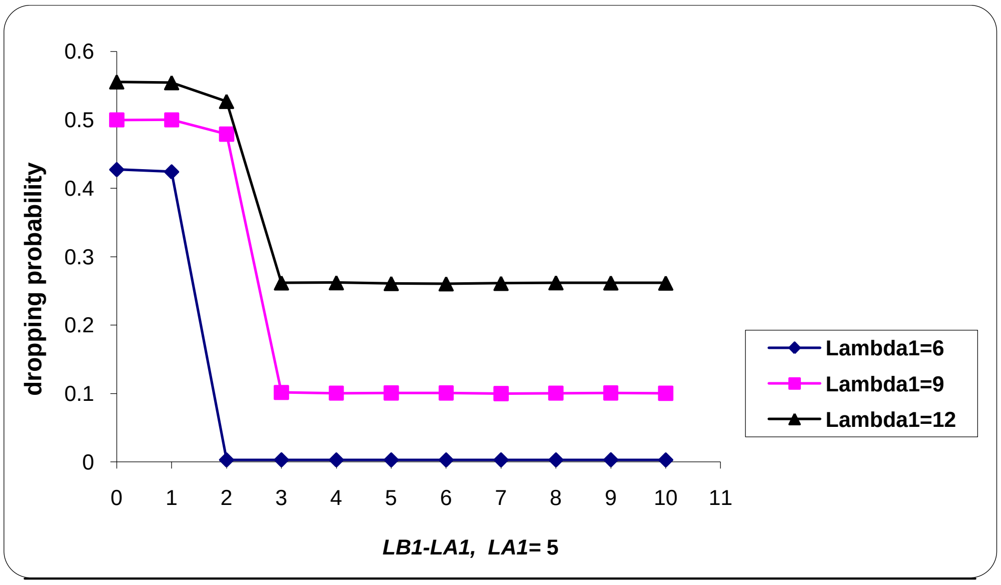


Figure 5.24: The overall dropping probability at different values of λ_1 where μ=12 and λ_2=3

5.8 Summary

A DC-RED model has been implemented to test the effect of applying the RED mechanism on two streams of traffic in a shared buffer. Intensive simulation analysis has been done to test the performance of the model with two thresholds per class. The performance has been assessed by looking at different combinations of conditions, such as varying the arrival rate from either class1 or class2 or by varying the value of the service rate. The performance analysis has demonstrated the significant impact of the threshold positions on the performance measures of both classes. The results clearly demonstrate how different load settings can provide different tradeoffs between delay and dropping probability to suit different service requirements. It has also been demonstrated that moving class2 thresholds not only affects class2 performance but also the performance of class1 which is something that was expected. More significantly, the results indicate that one class can completely dominate the other, which gives an unacceptable situation. Altering any of the arrival rates or the service rate has an apparent effect on the overall performance which would make it very difficult to reach a steady state condition for both classes with the shared buffer if an adaptive strategy was used, such as that in ARED. Also, the results suggest that to apply the DC-RED in a LRD or Variable Bit Rate (VBR) situation is likely to prove impossible because of the changes in the interdependencies caused by the changing traffic levels. The focus of the thesis from now on will thus involve experiments on a single class of traffic in a single buffer.

CHAPTER 6

Congestion Prediction in Networks with LRD Traffic

6.1 Introduction

Traffic measurements from communication networks have shown that network traffic can exhibit self-similar as well as LRD properties. In telecommunication networks, congestion events tend to persist, producing large delays and packet loss resulting in performance degradation. In order to guarantee QoS to diverse Internet services, congestion prediction has become a fundamental objective of some network management algorithms [16, 71]. Therefore high performance predictors are required that are efficient and simple to implement.

The ability to predict traffic congestion within a network is one of the fundamental requirements of modern network design. A number of recent studies have shown that network traffic exhibits self-similar and LRD properties [11-12, 14]. The use of traditional models, for example Poisson and the Markovian models, in networks characterized by self-similar processes can lead to overestimations about the performance of the analyzed networks [12]. In real traffic networks, packets tend to arrive in clusters, causing a phenomenon called the burst phenomenon. Due to LRD, the burst phenomenon can still be observed over large time scales and cannot be smoothed out by aggregating the traffic in a larger time scale [72]. The burst phenomenon within self-similarity, exists only in measured traffic and cannot be predicted with traditional traffic models [72-73]. Therefore, an efficient mechanism for prediction of the onset of congestion within networks that exhibit self-similarity and LRD is required.

Traffic modelling plays a significant role in the analysis of real network traffic. In order to design a robust and a reliable network, it is important to understand the traffic characteristics of the network and the best model to represent it. For instance, SRD, LRD, and self-similarity are examples of processes found in communication networks. This chapter explains each process in detail. It also investigates the impact of LRD on congestion prediction.

6.2 Short Range Dependence

Classical models are SRD processes, such as the Poisson process and the Markov chain models. SRD is the most widely used model for modelling traditional network traffic. Definition [11]: Consider a discrete time stochastic process $X(t)$, $t \in \mathbb{Z}$ where the autocorrelation function $r(k) = \gamma(k)/\sigma^2$, $\sigma^2 = E[(X(t) - \mu)^2]$ and $\mu = E[X(t)]$ for all $t \in \mathbb{Z}$, then the autocorrelation function $r(k)$ of the SRD process is summable and decays exponentially fast. This implies:

$$\sum_{k=-\infty}^{\infty} r(k) < \infty \tag{6.1}$$

6.2.1 Poisson Distribution

The Poisson distribution is a continuous-time, discrete-state probability distribution that expresses the probability of a number of events occurring in a fixed period of time. In computer network applications, the Poisson process is widely used to represent the distribution of the number of arrivals.

Definition [74]: Assume *N* (*t*) represents the number of events in the interval (0, *t*] . If the events occur successively in time, so that the intervals between successive events are independent and identically distributed according to an exponential distribution. Then the stochastic process $\{N(t), t \geq 0\}$ is a Poisson process with mean rate $\lambda>0$.

In a pure Poisson process with rate λ, the number of points occurring in a fixed interval of length *t* has the Poisson probability mass function (pmf) given by:

$$p(x) = \begin{cases} \dfrac{e^{-\alpha}\alpha^{x}}{x!}, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{6.2}$$

The cumulative distribution function (cdf) is given by:

$$F(x) = \sum_{i=0}^{x} \frac{e^{-\alpha}\alpha^{i}}{i!} \qquad ,\alpha > 0 \tag{6.3}$$

The Poisson distribution has an important property that the mean and the variance are both equal to α [68].

6.2.2 Exponential Distribution

The exponential distribution is used to describe the times between events in a Poisson process. It has been used to model inter-arrival times when arrivals are completely random and to model service times.

Definition [68]: A continuous random variable X is said to be exponentially distributed with parameter λ if its probability density function (pdf) is given by:

$$f(x) = \begin{cases} \lambda\, e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{6.4}$$

the cdf can be obtained by integrating Equation (6.4) such that:

$$F(x) = \begin{cases} 1 - e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{6.5}$$

The exponential random number generated to represent the service time or the inter-arrival time in a simulation is calculated by solving for x in Equation (6.5) using the inverse transform technique as follows:

$$R = 1 - e^{-\lambda x} \tag{6.6}$$

$$e^{-\lambda x} = 1 - R \tag{6.7}$$

$$-\lambda x = \ell n\,(1 - R) \tag{6.8}$$

$$x = -\frac{1}{\lambda}\ell n\,(1 - R) \tag{6.9}$$

R is a uniform random number distributed on [0, 1] and λ is the rate in service completions or arrivals per unit time.

The exponential distribution has a memoryless property which is one of its most important properties. It is also called the Markov property. The memoryless property means that for all $s \geq 0$ and $t \geq 0$, $P(X > s + t \mid X > s) = P(X > t)$ so that the time to the next event is independent of both the past and the future [68].

6.3 Long Range Dependence

Modern network traffic, such as Ethernet data and VBR video traffic are LRD processes [72, 75]. LRD is a process whose autocorrelation function is not summable and decays hyperbolically or decays with lag time as a power law [11]. This implies:

$$\sum_{k=-\infty}^{\infty} r(k) \to \infty \tag{6.10}$$

Definition [11]: Let X = (X_t: t >= 1) be a wide sense stationary process with mean $\mu=E[X_t]$ and variance $\partial^2=E[(X_t - \mu)^2]$. X is called an asymptotic LRD process if the autocorrelation function r (k) is given by:

$$r(k) \sim k^{-\beta} \quad , k \to \infty \quad , 0 < \beta < 1 \tag{6.11}$$

The Hurst parameter H is related to β by H=1-β/2 and 0.5<H<1. The Hurst parameter is an indicator of the LRD. Careful choice of H is very important as H values greater than 1 are prohibited due to the stationary condition on X and H=0.5 is the condition for SRD [15]. Processes with a low Hurst parameter (near 0.5) are less bursty while those with a high Hurst parameter (in the vicinity of 1) are highly bursty [76]. Thus the Hurst parameter is indicative of the resultant aggregated stream.

6.3.1 Heavy Tailed Distribution

The heavy tailed distribution, also called the long tailed distribution, is a probability distribution whose tail is heavier than the exponential distribution. Heavy tailed distributions behave quite differently from the exponential distribution, which have tails that decline exponentially fast. In contrast, heavy tailed distributions have tails that decline hyperbolically slowly and have infinite variance, reflecting the extremely high variability that these distributions capture.

Definition [77]: A random variable Z is said to be a heavy tailed distribution if

$$P(Z > x) \sim cx^{-\alpha} \qquad , x \to \infty \tag{6.12}$$

0 < α < 2 is called the tail index and c is a positive constant.

A number of recent studies have shown evidence that file sizes and connection durations have heavy tailed distributions and measurements of computer network traffic have shown that autocorrelations are often related to heavy tails, which is a phenomenon of self-similarity [77].

By modelling a number of ON/OFF sources with heavy tailed probability distributions (for example Pareto distributions) for both ON and OFF periods, a self-similar aggregated traffic can be generated [13]. The ON period corresponds to a single transmission session time and the OFF period corresponds to the silent period of a source.

6.3.2 Pareto Distribution

The Pareto distribution (also referred to as the hyperbolic distribution and the power-law distribution [12, 78]) is the simplest heavy tailed distribution to model self-similar and LRD processes [79]. It can be defined using the cdf [78] such that:

$$F(x) = \begin{cases} 1 - \left(\frac{k}{x}\right)^{\alpha} & , x \geq k \\ 0, & x < k \end{cases} \tag{6.13}$$

The pdf is given by [78]:

$$f(x) = \alpha\, k^{\alpha}\, x^{-\alpha-1} \qquad , x \geq k,\ \alpha, k > 0 \tag{6.14}$$

$0 < \alpha < 2$ is the shape parameter (the tail index) and k is the scale parameter. If $\alpha \leq 2$ then the Pareto distribution will have infinite variance and if $\alpha \leq 1$ it will have infinite mean [12]. k represents the smallest possible value of the Pareto random number [78].

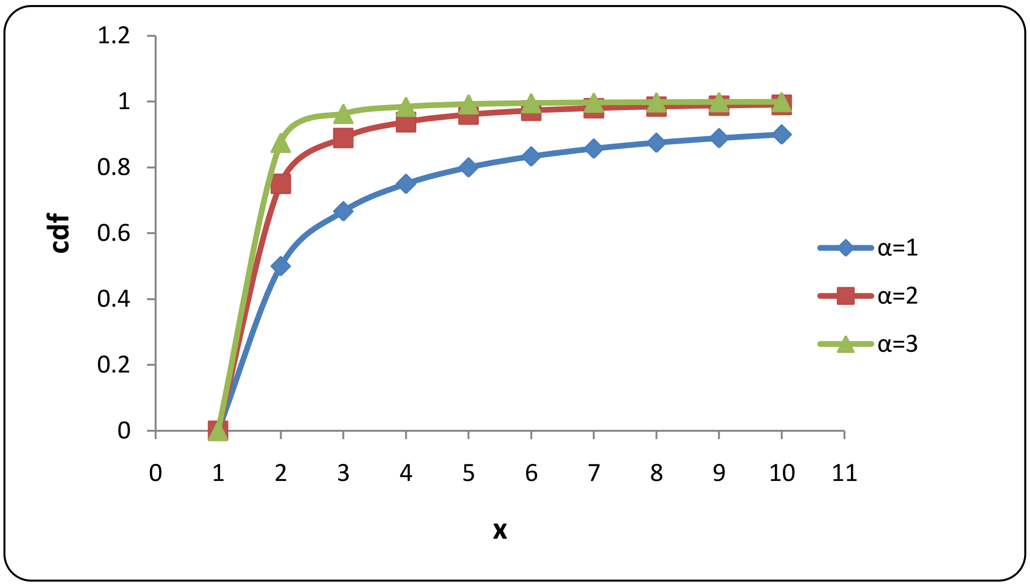


Figure 6.1: The cdf of a Pareto distribution at different values of α and k=1

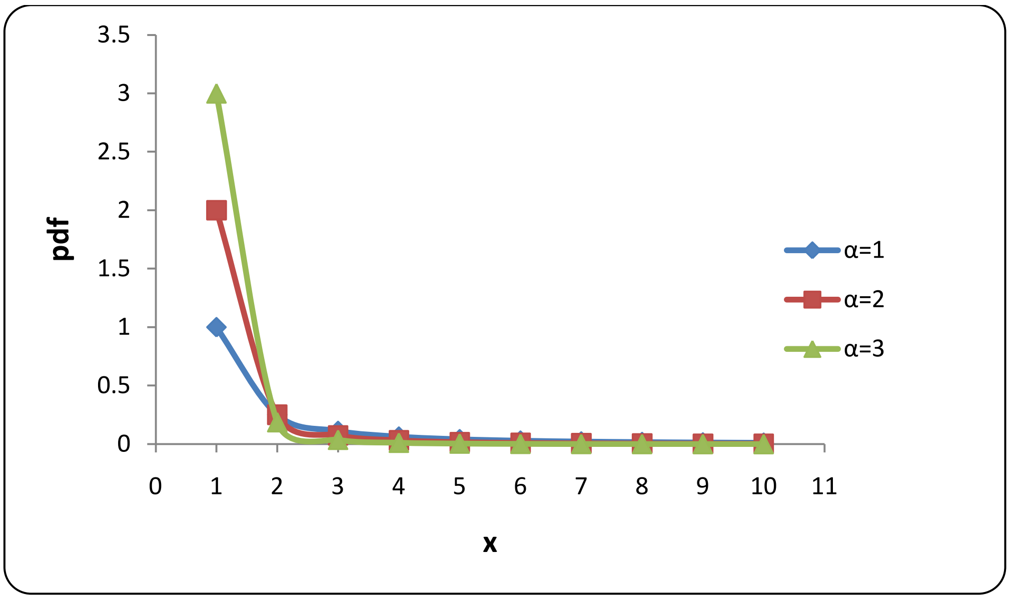


Figure 6.2: The pdf of a Pareto distribution at different values of α and k=1

Figures (6.1) and (6.2) show that for the Pareto distribution, as the value of α decreases, more of the probability mass is located in the tail of the distribution [12, 79]. The Pareto random number generated in a simulation can be computed by applying the inverse transform technique on the cdf Equation (6.13) as follows:

$$F(x) = 1 - \left(\frac{k}{x}\right)^{\alpha} \tag{6.15}$$

$$1 - R = \left(\frac{k}{x}\right)^{\alpha} \tag{6.16}$$

$$(1 - R)^{\frac{1}{\alpha}} = \left(\frac{k}{x}\right) \tag{6.17}$$

$$x = \frac{k}{(1 - R)^{\frac{1}{\alpha}}} \tag{6.18}$$

R is a uniform random number that can have a value in the range [0, 1] and *x* is the Pareto random number.

6.4 Self-Similarity

Self-similarity is a notion introduced by Mandelbrot [80]. The authors in [79] provided evidence and possible causes of self-similarity in World Wide Web (WWW) traffic. Self-similarity can be classified into two categories: deterministic and stochastic.

6.4.1 Deterministic Self-Similarity (Scale Invariance)

A mathematical object is self-similar, if its parts, when magnified, resemble the shape of the whole in a suitable sense [15]. Figure (6.3) depicts the symmetrical and scale-invariant properties found in the Koch snowflake.

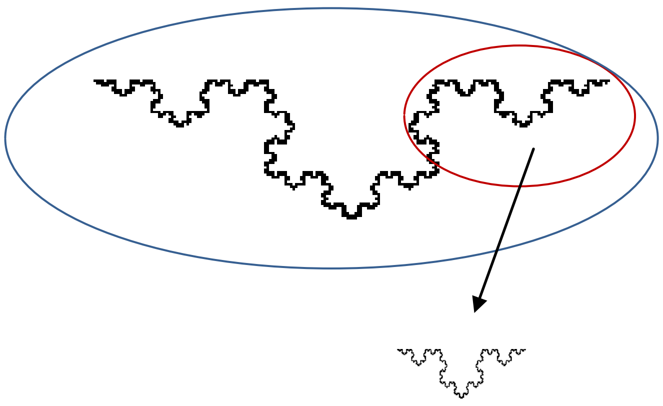


Figure 6.3: Fractals of Koch snowflake [81]

6.4.2 Stochastic Self-Similarity

Stochastic self-similarity is a phenomenon found in real traffic networks. The probabilistic properties of the self-similar processes remain unchanged when the process is viewed at varying time scales [15, 82].

The properties of self-similar traffic are very different from properties of traditional models based on Poisson or the Markovian models. As shown in Figure (6.4), Poisson processes lose their burstiness and flatten out when time scales are changed. However, they can exhibit burstiness over a short range of time scales.

Definition [79]: Let X = (X_t: t =1, 2, 3, ...) be a covariance stationary stochastic process with mean μ, variance of ∂^2 and autocorrelation function $r(k)$, $k \geq 0$. It is also assumed that the autocorrelation function, $r(k)$, has the form [11]:

$$r(k) \sim k^{-\beta} \quad ,k \to \infty \qquad , 0 < \beta < 1 \tag{6.19}$$

Then the new covariance stationary time series $X^{(m)}$ is obtained by averaging X over non-overlapping blocks of size m [15]:

$$X^{(m)}(k) = \frac{1}{m} \sum_{t=m(k-1)+1}^{mk} X(t) \quad ,k \geq 1 \qquad ,m = 1,2,3,\ldots \tag{6.20}$$

The aggregated series X is called a self-similar process, if $X^{(m)}$ is the same as X at least with respect to their autocorrelation function [11] with self-similarity parameter H, where $0<H<1$, $H \neq 0.5$.

In order to guarantee that the modelled traffic implies self-similarity and LRD as well, H should be restricted by $0.5<H<1$, as not all self-similar processes are LRD and vice versa [15].

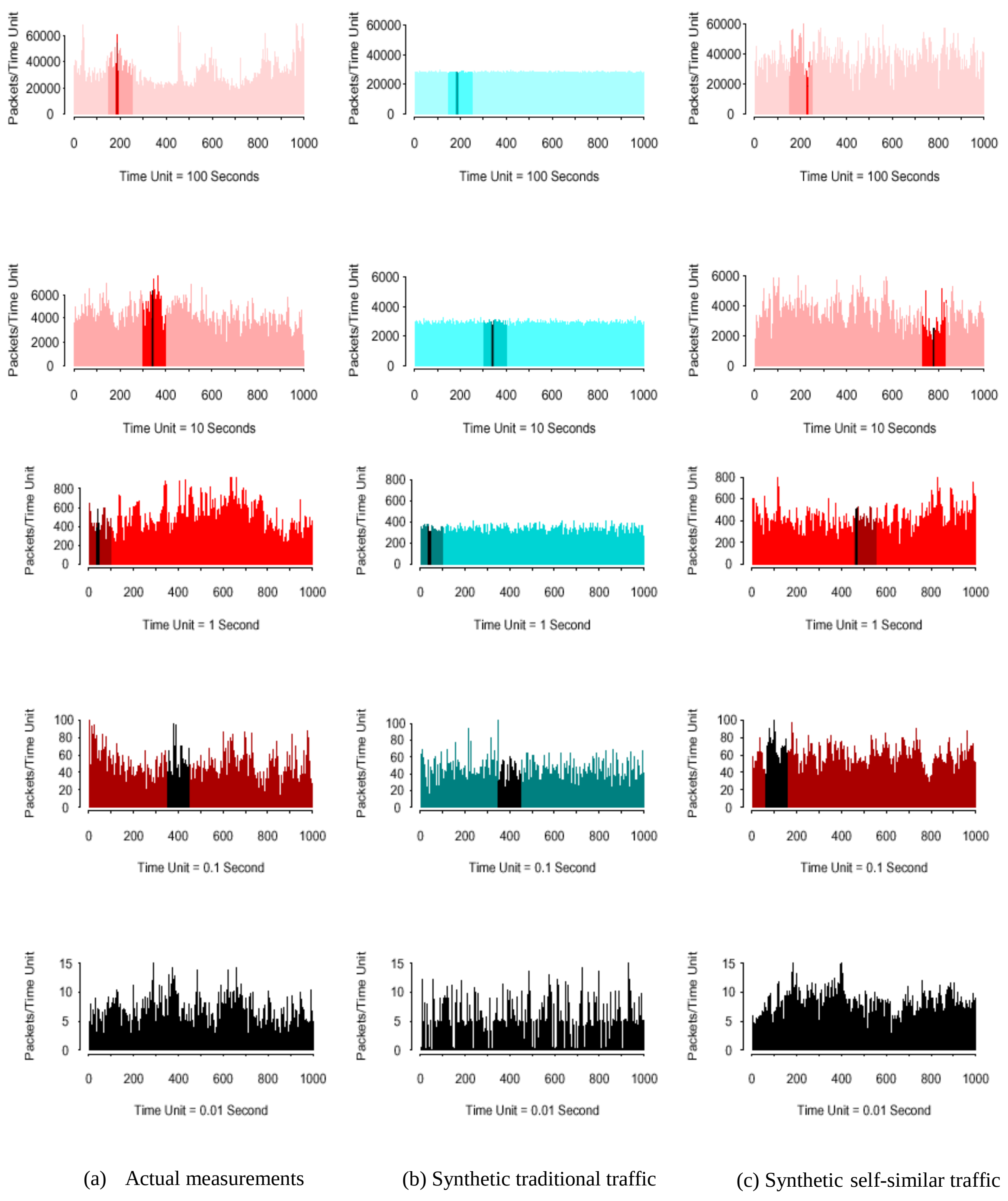


Figure 6.4: Comparison of actual, traditional and self-similar Ethernet traffic viewed on different time scales [82]

6.5 Impact of LRD on Congestion Prediction

Satisfying QoS requirements while achieving high utilization has become a very complex and difficult task due to the scale-invariant burstiness of LRD traffic. Scale-invariant burstiness implies the existence of some periods of high activity at a wide range of time scales which badly affects congestion control. However, the very fact that traffic is LRD implies the existence of a correlation structure which may be exploitable for congestion prediction purposes; that is, the correlation structure present in LRD traffic can be used to predict the future behaviour. Also, predicting the onset of congestion under self-similar traffic conditions with sufficient reliability can be effectively utilized for congestion control purposes [83]. The predictability structure present in LRD traffic can be used for improving network performance based on the feedback algorithm presented in this chapter.

6.5.1 Motivation

The authors in [17-18] theoretically demonstrated that the properties of LRD can be used to predict traffic behaviour in the not-too-distant future and argued that controlling the LRD traffic sources can be done by admitting new sources, removing existing sources or changing the levels of existing sources by turning OFF some of the sources when in the ON state or vice versa. The latter is the core idea for the prediction algorithm presented in this chapter. The novelty in this approach is to combine the use of LRD prediction with an existing congestion control algorithm, namely RED [4].

The algorithm used has a feedback control strategy that depends on the mean time spent ON for each node to control the number of packets through the buffer. A queueing model with a number of ON/OFF sources has been used to generate LRD, self-similar network traffic. An algorithm has been developed involving a novel congestion prediction for AQM. The algorithm is based on the hypothesis that a connection that has been active for a time duration which exceeds a certain limit is more likely to persist in this state for a long period when the traffic exhibits LRD [15, 17].

6.5.2 Simulation Model

It is assumed that the buffer has a finite capacity of K packets, including the server, with two thresholds (L_1) and (L_2) as shown in Figure (6.5). The source consists of a superposition of N (ON/OFF) nodes. The queueing discipline is FIFO. When the average queue length is less than the minimum threshold (L_1), there is no dropping and the source operates normally. If the average queue length exceeds the maximum threshold (L_2), then the source is signalled to stop sending packets by dropping all the excess packets. Packet transmission can commence after the next departure. If the average queue length in the system falls between the first threshold (L_1) and the second threshold (L_2), then the arriving packets should be dropped with dropping probability P_d [4].

$$P_d = \max{}_p \frac{(avg - L_1)}{(L_2 - L_1)} \tag{6.21}$$

where max_p is the maximum dropping probability and avg is the average queue length calculated as in [4].

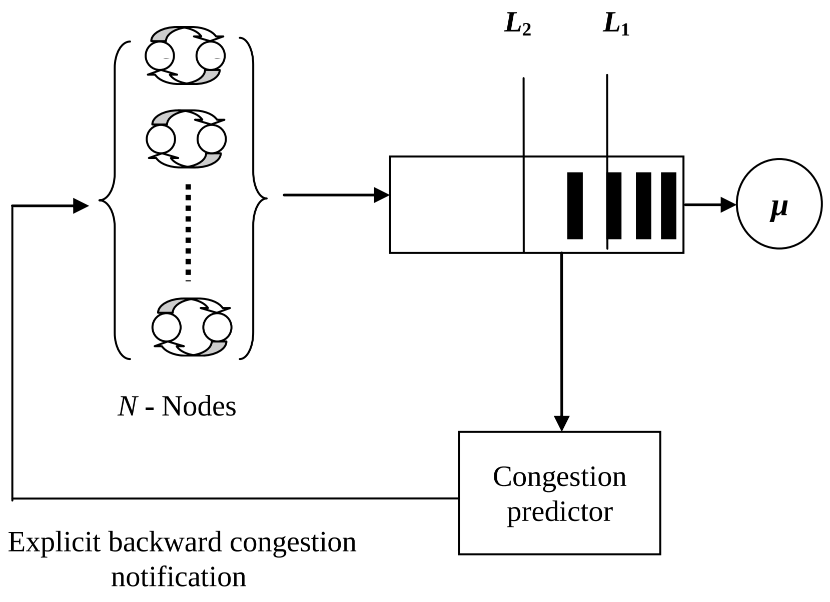


Figure 6.5: Schematic diagram of the congestion prediction model

The implementation of the congestion predictor would be appropriate at edge routers so that the multiplexer that feeds the queue has knowledge of the nodes. This is because LRD characteristics are more prevalent at the edge of networks and flatten out within the network. If all the LRD is not removed then the congestion predictor might need to be applied on a stage by stage basis. Tracking of information from the nodes adds considerable complexity to the model because the status of each node needs to be monitored. The ON times of an individual source are measured as the intervals during which transmission of a packet is taking place from that source and the OFF times correspond to the intervals between these periods when no transmission of packets is taking place. Feedback to a source can be done using explicit backward congestion notification. These explicit messages can be sent using control packets, which might be transmitted periodically or through a separate signalling channel if one is available, but this depending on the system.

6.5.3 Prediction Algorithm

In the proposed continuous-time queueing system, the initial state for all the nodes has been chosen randomly. It is assumed that the *N* connections are identical and share the same specifications. During an ON state, packets are sent according to a Poisson distribution with rate λ. When idle, the node is said to be in the OFF state. The transition rates from ON to OFF or from OFF to ON follow the Pareto distribution. The sojourn time distribution is chosen to be a heavy-tailed one in order to capture the long term dependencies in the arrival process.

For each input node two time events have been generated. The first event time is the residence time which has been generated using a Pareto random number (giving switchover time). The second event time is the next arrival time (giving time of next arrival at that node) which has been generated using an exponential random number generator. The node number, event type (arrival, departure or transition) and event time are placed in an event list. Scheduling the next event type can be done by searching the event list for the shortest event time. The prediction algorithm presented in Table (6.1) is based on calculating the expected number of packets transmitted from all the nodes as follows:

$$\text{Expected number of packets} = \sum_{i=1}^{N} n_i \, \lambda_i \, \tau_i \tag{6.22}$$

n_i: the status of node *i* (ON=1,OFF=0)

λ_i: arrival rate from node *i* (packets/sec.)

τ_i: mean time spent ON for node *i* (sec.)

N: maximum number of nodes

It should be pointed out that equations (6.21) and (6.22) are used in both the simulation and also would be used in practice in the implemented model. The previous equations in this chapter relate to the simulation only since these relate to such things as traffic models and random number generation which are, of course, simply representative of features found in real traffic, such as LRD for example. This implies that the congestion predictor would, in practice, need to determine the mean queue length in the buffer and also status of the individual sources and their ON and OFF periods by observing and monitoring the traffic from each of the individual sources. Also, the number of packets sent by the individual sources received in a given period would need to be counted to determine average arrival rates and the average queue length can be determined as in [4].

If source *i* has been ON for time τ_i and the number of packets calculated using Equation (6.22) exceeds a specific limit, this can be considered as an indication of LRD behaviour which can possibly lead to congestion. The algorithm then searches a node status vector for the node which may cause congestion; this means the one with the highest τ_i. This is based on the hypothesis that if a LRD source has been active for a long time, there is a very high probability that it will remain active for a long time in the future and so may cause congestion as a consequence [17]. By identifying the node with the highest τ_i, the algorithm forces this node to the OFF state in an attempt to avoid congestion. Forcing the node to the OFF state can be done using explicit backward congestion notification which can be sent through control packets as shown in Figure (6.5).

The mean time spent ON for each node is calculated as follows [84]:

$$\text{The mean time spent ON} = \int_{k}^{\infty} f(x)\,dx = \frac{k_{on}\,\alpha_{on}}{\alpha_{on}-1} \tag{6.23}$$

where k_{on} and α_{on} are the scale and shape parameters respectively for the node in the ON state.

Although equation (6.23) is used in the simulation, in practice the mean time spent ON would be determined by the congestion predictor which would use its measured data for the ON times of the individual sources to compute these averages over a specific period of time.

Table 6.1: Pseudocode for the prediction algorithm

```
Initialization
Create node status vector ($N$)
While (simulation time $\leq$ simulation time required)
{
  Timing (node vector)
  For each packet arrival
  Calculate the new $avg$
    if ($L_1 < avg < L_2$)
      Drop the arriving packet with probability $P_d$
    else if ($L_2 < avg$)
      Drop the arriving packet
    else
      Add the arriving packet to the queue

  The expected number of packets= $\sum_{i=1}^{N} n_i \lambda_i \tau_i$
   if (measured number of packets $\geq$ Limit)
  {
    Identify the node with the highest mean time spent ON
    Force the node to OFF state
   }
}
```

6.5.4 Performance Results

Due to the growing complexity of modern telecommunication networks, simulation has become the best paradigm for their performance evaluation**.** A discrete event simulation has been implemented using Java programming to assess the performance of the proposed model. The parameters used have been initialized as in Table (6.2). The two thresholds L_1 and L_2 have been set as recommended by [4]. α is related to the Hurst parameter H by H= (3-α)/2 [15] giving H_{on}=0.9 and H_{off} =0.75.

Table 6.2: The congestion predictor configuration parameters

Parameter	Value
w_q	0.002
max_p	0.1
L_1	5
L_2	15
queue size	30
λ	5
μ	7
N (number of nodes)	5
α_{on}	1.2
α_{off}	1.5
$k_{on}= k_{off}$	1

The performance of the proposed model has been examined in terms of the total number of packets transmitted from all the nodes, the MQL, the average delay, the dropping probability and the normalized throughput. Three different limit values have been used ($30x10^6$, $20x10^6$ and $10x10^6$), in order to examine the effect of changing the number of packets limit on the performance.

Figure (6.6) represents a comparison between the number of packets transmitted from all the nodes with and without prediction. By using a limit of $30x10^6$ packets for the target number of packets, it is noticeable that after applying the prediction algorithm the number of packets has been controlled by not exceeding the specified limit instead of increasing over the simulation time as in the case without prediction. Figures (6.7) and (6.8) represent the effect of using a lower limit value on the total number of packets. These graphs represent the effectiveness of the algorithm in controlling the number of packets to lower values.

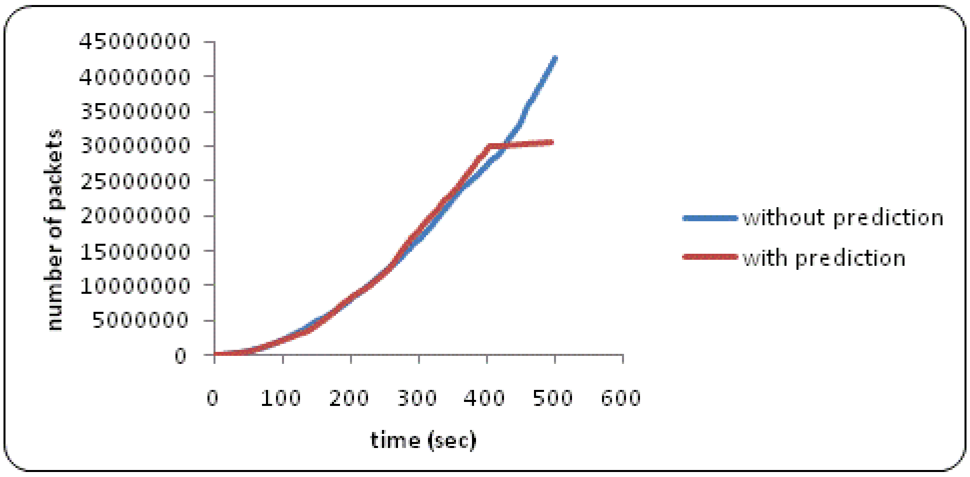


Figure 6.6: The number of packets using limit=30x10^6

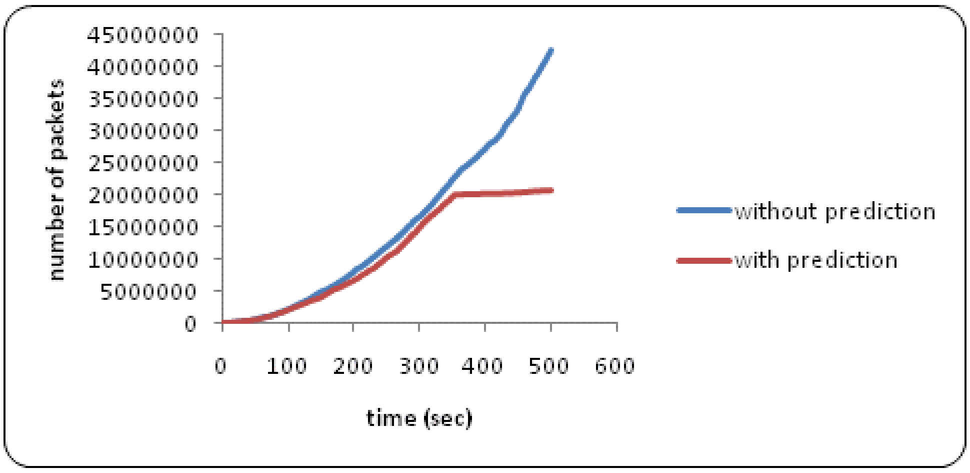


Figure 6.7: The number of packets using limit=20x10^6

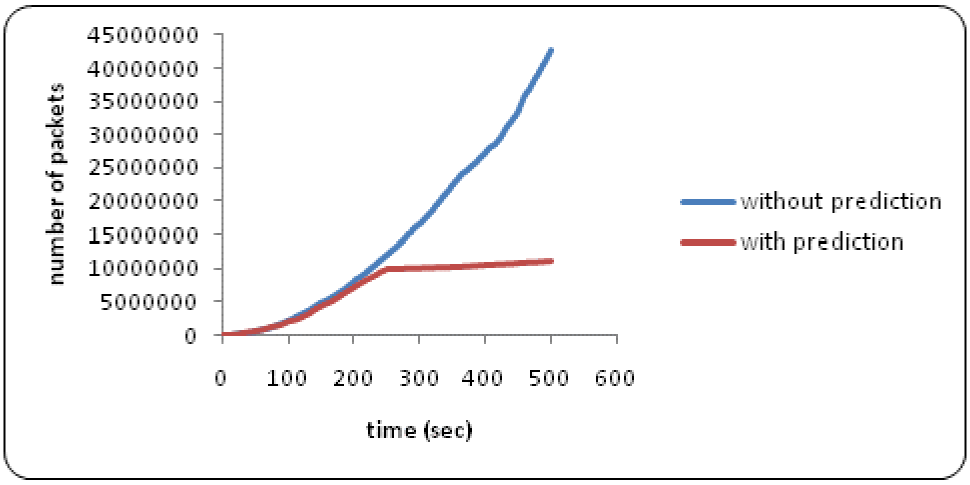


Figure 6.8: The number of packets using limit=10x10^6

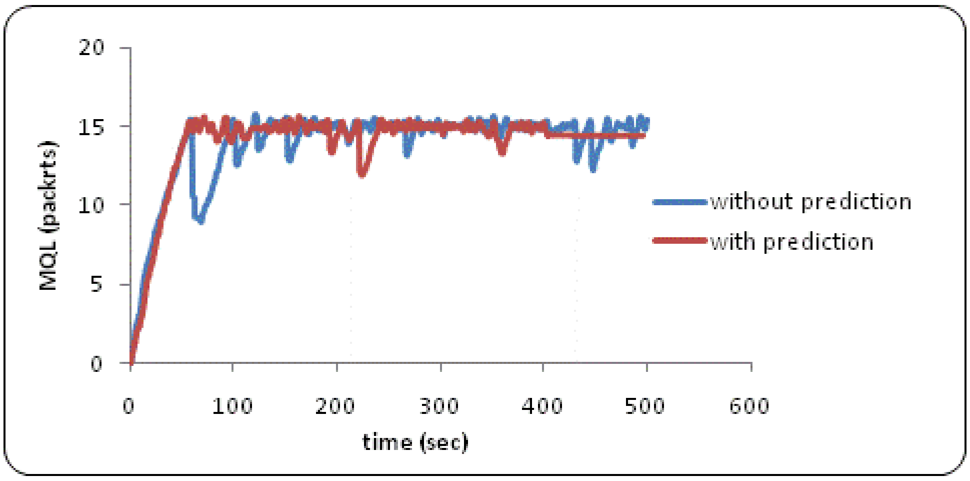


Figure 6.9: The MQL using limit=30x10^6

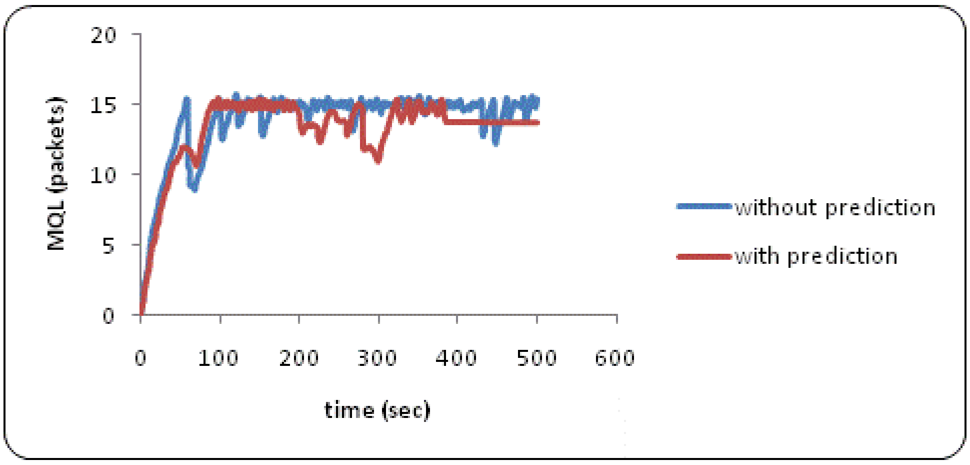


Figure 6.10: The MQL using limit=20x10^6

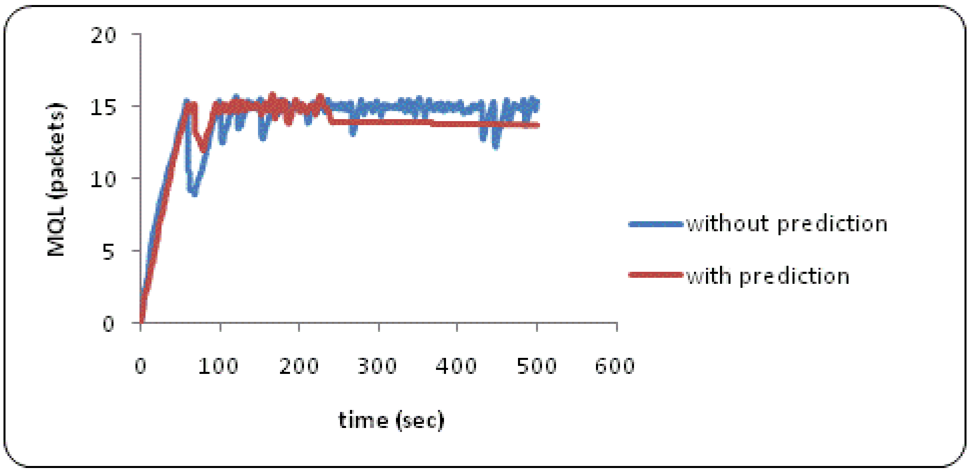


Figure 6.11: The MQL using limit=10x10^6

The effect of applying the algorithm on the MQL is presented in Figures (6.9), (6.10) and (6.11). The MQL becomes much more stable after applying the algorithm. Thus, it can be said that, based on the LRD properties, the algorithm managed to control the MQL over the simulation time. From Figure (6.11), it is noticeable that the lower the limit value, the more effective the algorithm in detecting congestion as the MQL has been controlled at time ≈200 while in Figure (6.9) the MQL has been controlled at time ≈400. The limitations appear to be that the value of the MQL still remained high and close to the maximum threshold position even after offending flows are dropped.

Figure (6.14) shows that the algorithm gives lower values for the average delay and at the point the algorithm activates, the delay becomes limited. Compared with the results obtained in Figures (6.12) and (6.13), the average delay values obtained in Figure (6.14) have lower values which imply earlier notification of congestion as a result of using a lower limit value. Applying the prediction algorithm also gave better performance in terms of the dropping probability as shown in Figures (6.15), (6.16) and (6.17).

Based on the results obtained, the lower the limit value, the better the algorithm works as it gives earlier congestion notification and congestion control but is likely to drop more flows to do so. Despite the fact that using a lower limit value gave better berformance than using the high limit values in terms of the number of packets, the MQL, the average delay and the dropping probability it showed inferior performance in terms of the normalized throughput as shown in Figures (6.18), (6.19) and (6.20). Figure (6.20) shows the huge degradation in the normalized throughput as a result of using a small limit value and dropping many flows. Based on the results obtained, the additional complexity of the model would need to be considered in any decision to implement the model.

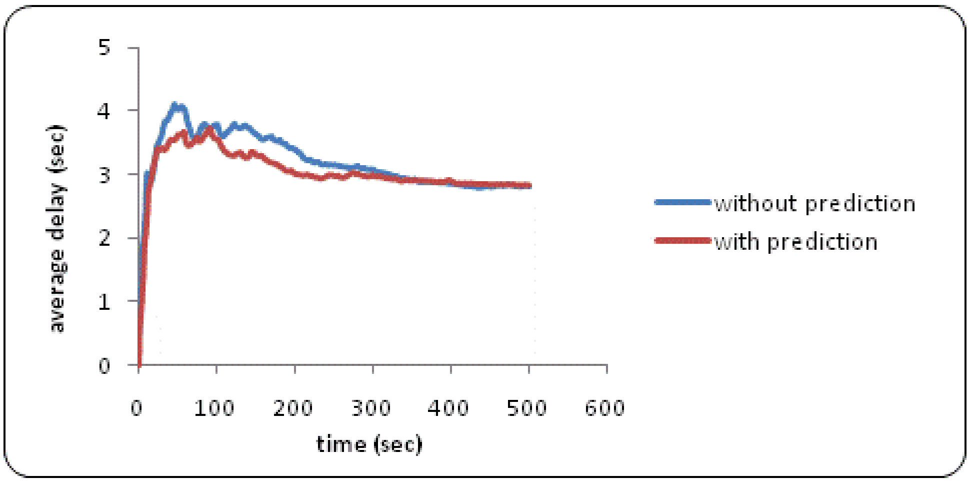


Figure 6.12: The average delay using limit=30x10^6

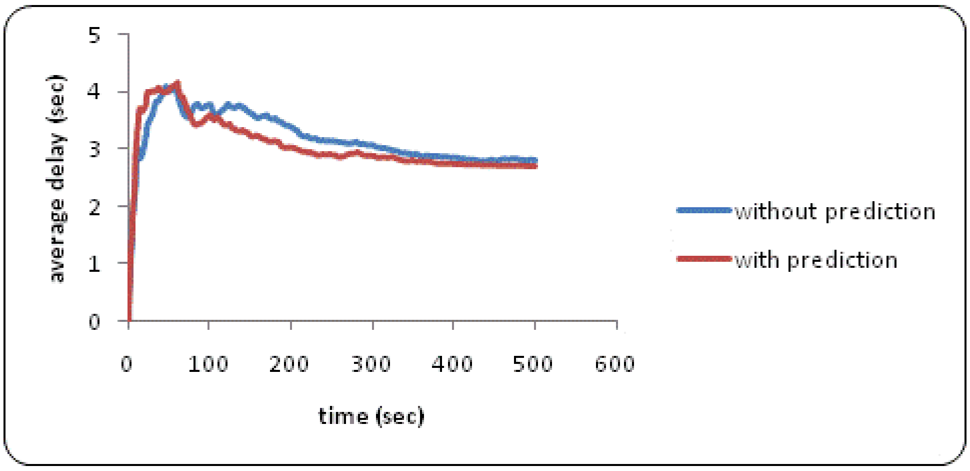


Figure 6.13: The average delay using limit=20x10^6

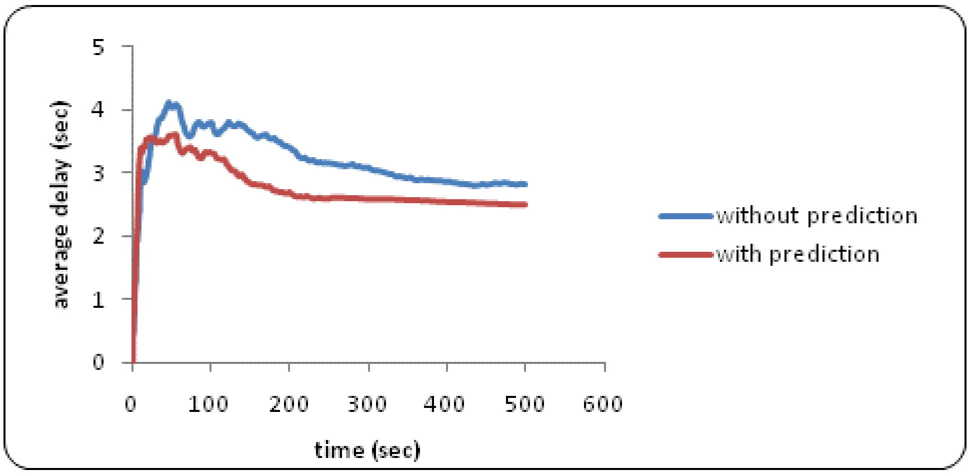


Figure 6.14: The average delay using limit=10x10^6

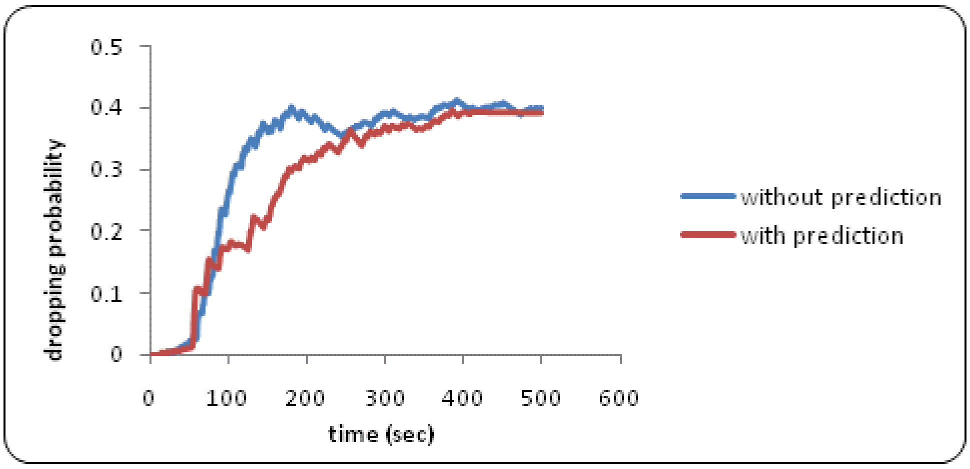


Figure 6.15: The dropping probability using limit=30x10^6

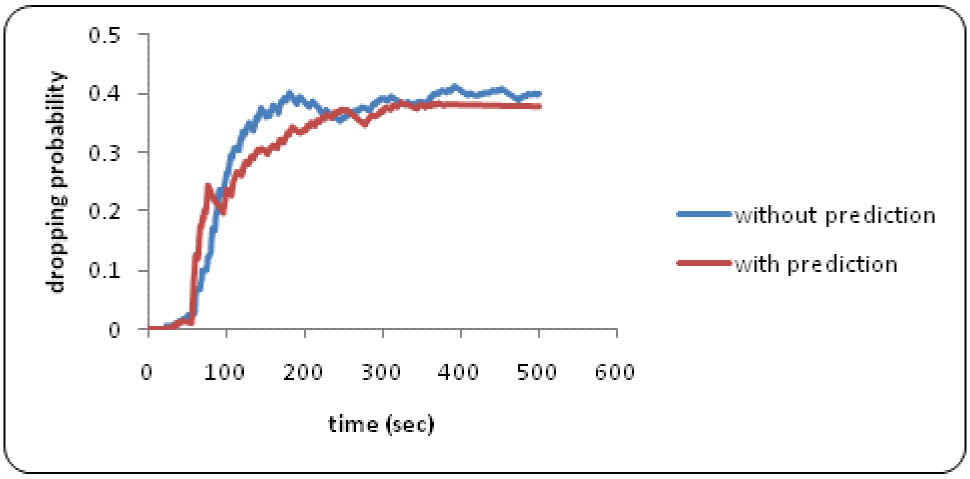


Figure 6.16: The dropping probabilty using limit=20x10^6

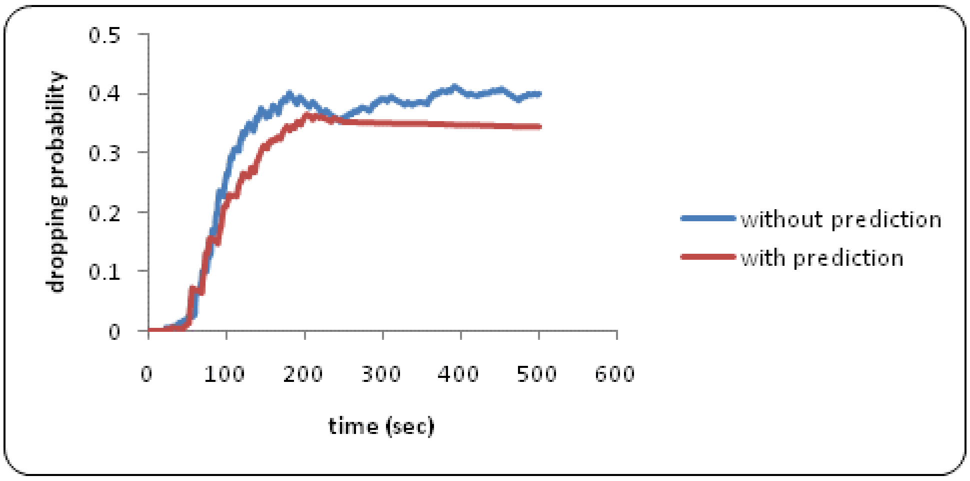


Figure 6.17: The dropping probability using limit=10x10^6

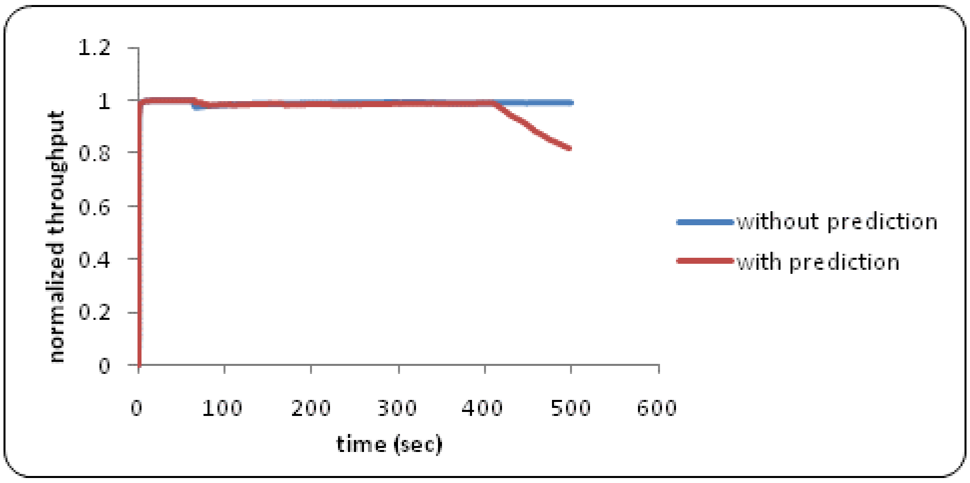


Figure 6.18: The normalized throughput using limit=$30x10^6$

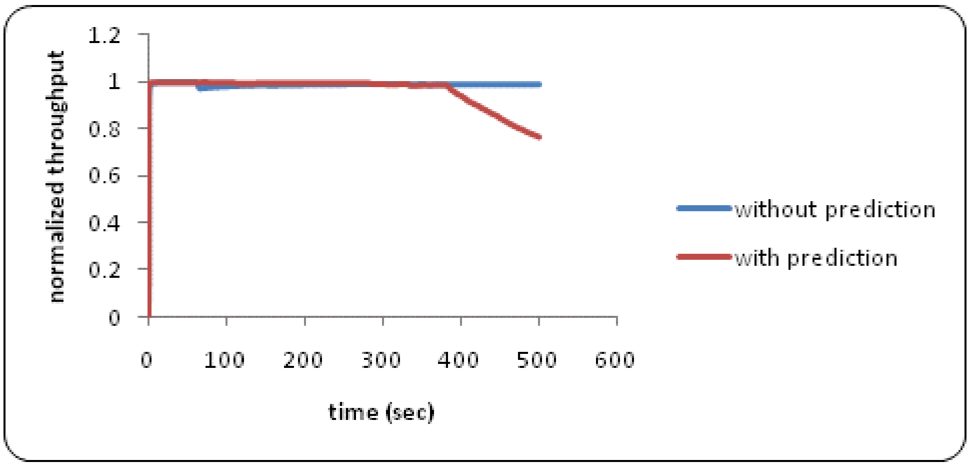


Figure 6.19: The normalized throughput using limit=$20x10^6$

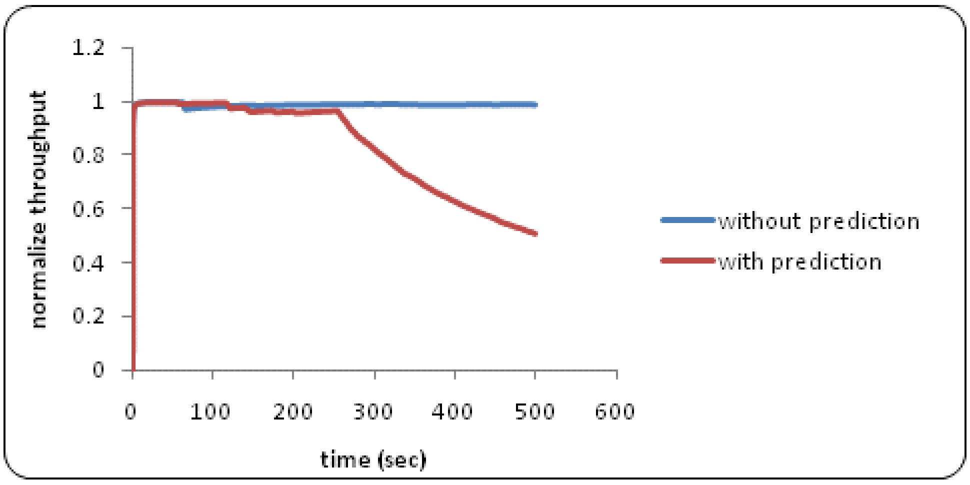


Figure 6.20: The normalized throughput using limit=$10x10^6$

6.6 Impact of System Parameters

From the previous sections it is clear that the performance of the model depends on the limit value. Choosing the right limit value is very important as it impacts on the performance of the proposed model. Therefore, there is a need to remove the system dependency on the limit value. This can be done using a sliding window algorithm.

By observing the behaviour of the proposed model, it appears that the prediction algorithm managed to control the number of packets over the simulation time. From Figure (6.21) it is noticeable that controlling the number of packets is not in a steady way but it encounters a slight increase over time. To avoid the number of packets increasing indefinitely, a sliding window algorithm could be used to limit the total number of packets in Equation (6.22).

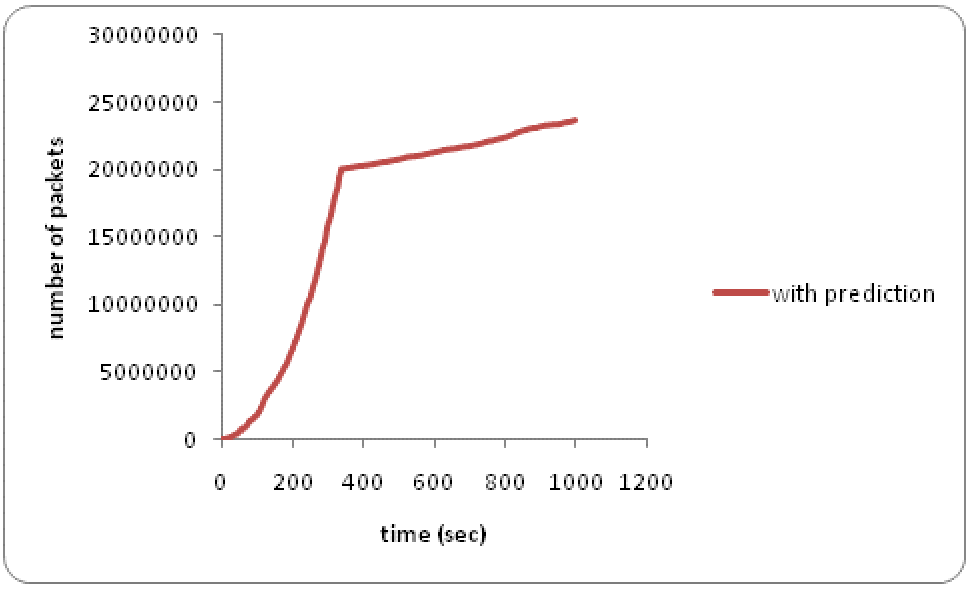


Figure 6.21: The number of packets with prediction using limit=$20x10^6$

6.6.1 Applying a Sliding Window Algorithm

As a policy in real settings it would clearly be better if flow rates of offending sources were reduced by a specified amount rather than flows dropped to mitigate the reduction in throughput. Perhaps using a sliding window algorithm could be used in collaboration with the algorithm to do this [85]. Using a sliding window algorithm has many benefits as it mimics the behaviour of a real life network, controls the number of packets in a steady way and removes the system dependency on the limit value. Figure (6.22) depicts the details of the sliding window algorithm.

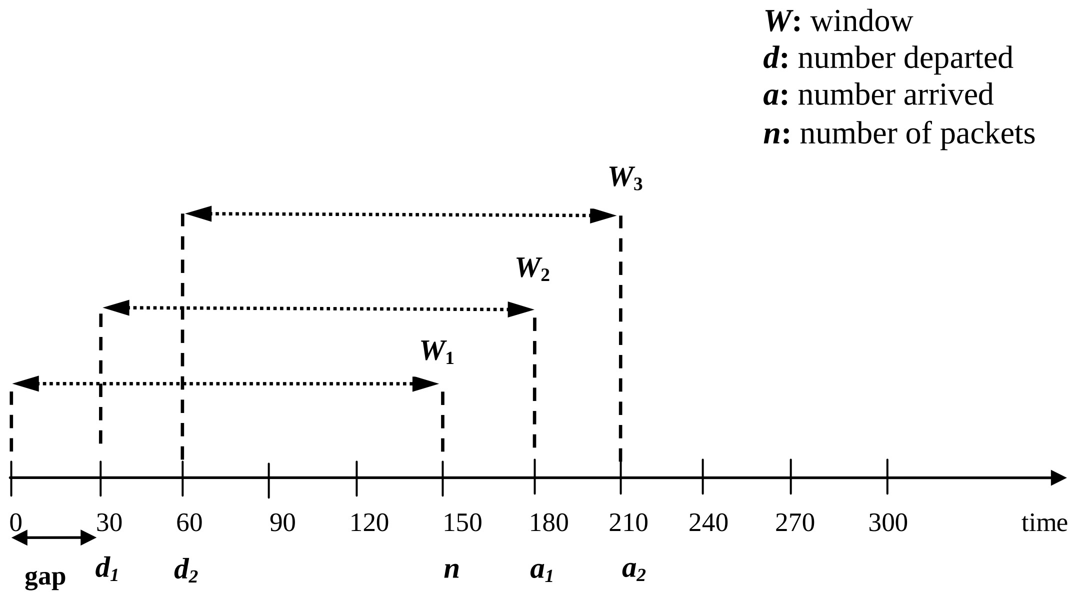


Figure 6.22: Schematic diagram of the sliding window algorithm

The basic idea for the sliding window algorithm is to cut the simulation time into large intervals of equal lengths called windows (*W*). Each window should be cut into small intervals of equal lengths called gaps. These gaps will be used to slide the window into the new position. At the end of each gap the number of packets is calculated using Equation (6.22). These values which are considered as the number departed values (*d*) are saved in an array to be used later in the algorithm. At the end of the first window the number of packets calculated using Equation (6.22) is called (n). At the end of the first window the prediction algorithm explained in Table (6.1) should be applied to force the offending sources to the OFF state based on the value of (n). The number of packets calculated at the end of the new gap (point 180 in Figure (6.22)) is called the number arrived (a). The new value for (n) which is called (n') is calculated by adding the value of the number arrived (a) to the value of (n) and subtracting the value of the number departed (d). (n') represents the new number of packets at the end of the new window (W_2) which will replace the old value of (n). So the number of packets used for the prediction algorithm in this case (n') is calculated as follows:

$$n' = n - d + a \qquad (6.24)$$

The algorithm will repeat the same steps but with sliding the window to new positions and repeating the algorithm at the end of each window, as explained in Table (6.3). For example the new value for the number departed (d_2) will be the value at 60 in Figure (6.22) and the new value for the number arrived (a_2) will be the value at 210. This is to keep the window length fixed (equals 150 in this example) and sliding from point to point over the simulation time.

The sliding window algorithm has been applied in the prediction algorithm to examine the effect of applying the new algorithm on the performance of the model. The prediction algorithm has been applied up to the simulation time=500. The reason for applying the prediction algorithm for the first part of the simulation is to control the number of packets to the limit value. Then the sliding window algorithm has been applied from simulation time >500 till the end of the simulation. When the simulation time is >500 the time is divided into windows of equal lengths and the prediction algorithm is applied only at the end of each window. The sliding window algorithm is used to maintain the stability of the number of packets and to prevent the number increasing indefinitely

Table 6.3: The sliding window algorithm

- Cut the simulation time into windows of equal length (*TWL*).
- Divide the window into small intervals called gaps.
- Calculate the number of expected packets at the end of each gap (*d*).
- Calculate the number of expected packets at the end of the first window (*n*).
- Apply the prediction algorithm based on the value of (*n*).
- Slide the window by adding the value of the number arrived (*a*) and subtracting the value of the number departed (*d*).
- Apply Equation (6.24) to calculate the number of packets at the end of the new window (*n*').
- Repeat the previous steps until the end of the simulation time.

To examine the effect of applying the sliding window algorithm on the performance, the configuration parameters have been set as in Table (6.2). The limit value has been set to 20x10^6, gap=50 and *TWL*=150. Figure (6.23) shows the stability in controlling the number of packets. Compared with Figure (6.21); the number of packets has been controlled to its limit value and is not increasing anymore. Figure (6.24) shows a comparison between the performance of the algorithm without applying the sliding window algorithm and after applying the sliding window algorithm. It is clear from Figure (6.24) that using the sliding window algorithm is more effective in controlling the number of packets in a stable way.

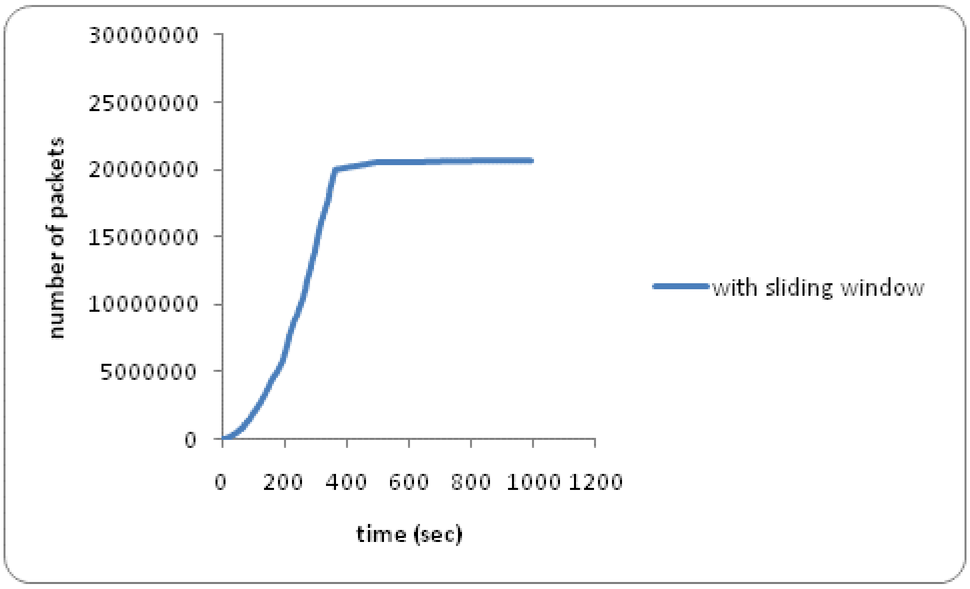


Figure 6.23: The number of packets after applying the sliding window algorithm using limit=$20x10^6$

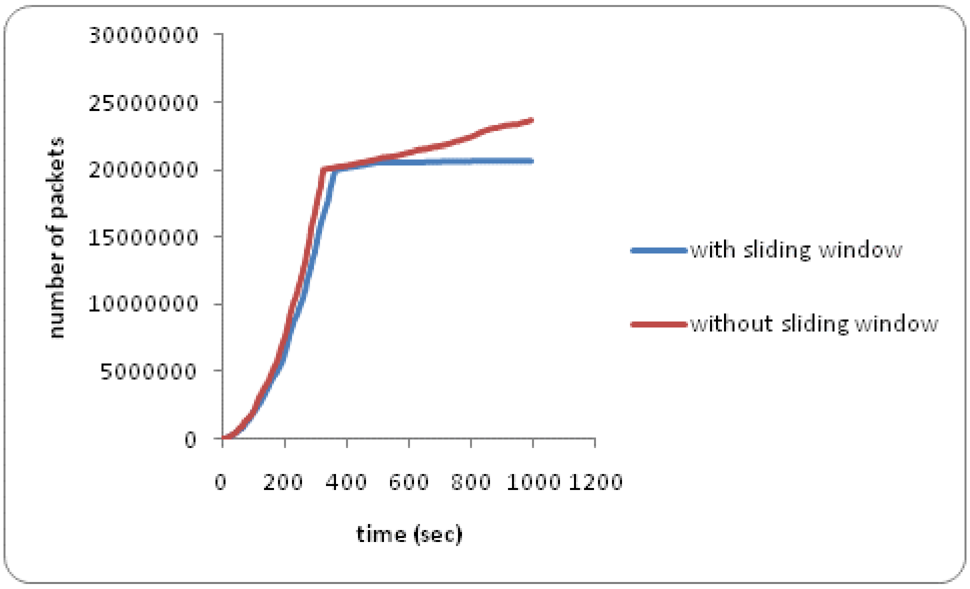


Figure 6.24: Comparing the number of packets with and without applying the sliding window algorithm using limit=$20x10^6$

Figure (6.25) shows the improvement in the normalized throughput after applying the sliding window algorithm. Using the prediction algorithm alone caused degradation in the normalized throughput as it appears from Figures (6.18), (6.19) and (6.20). While using the sliding window algorithm adds an advantage to the performance as it allows the normalized throughput to rise again. This is because the process of forcing the node to the OFF state is only done at the end of each window not after each arrival.

The limitation to the sliding window algorithm is that the MQL is still around the maximum threshold value as it appears from Figure (6.26). There is not much change in the performance of the average delay as shown in Figure (6.27). Also the dropping probability presented in Figure (6.28) is not differing too much from the results obtained in Figure (6.16).

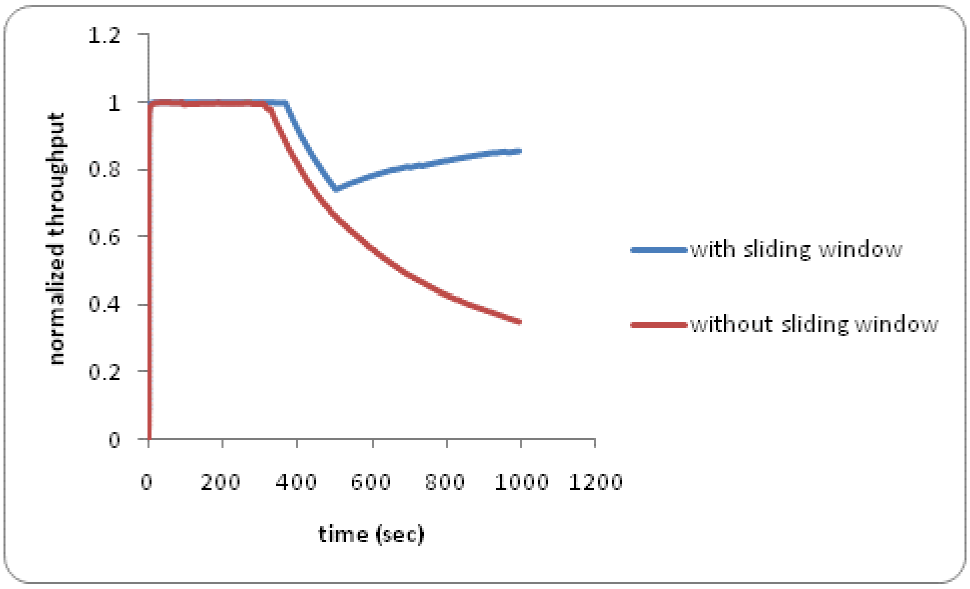


Figure 6.25: The normalized throughput after applying the sliding window algorithm using limit=20x10^6

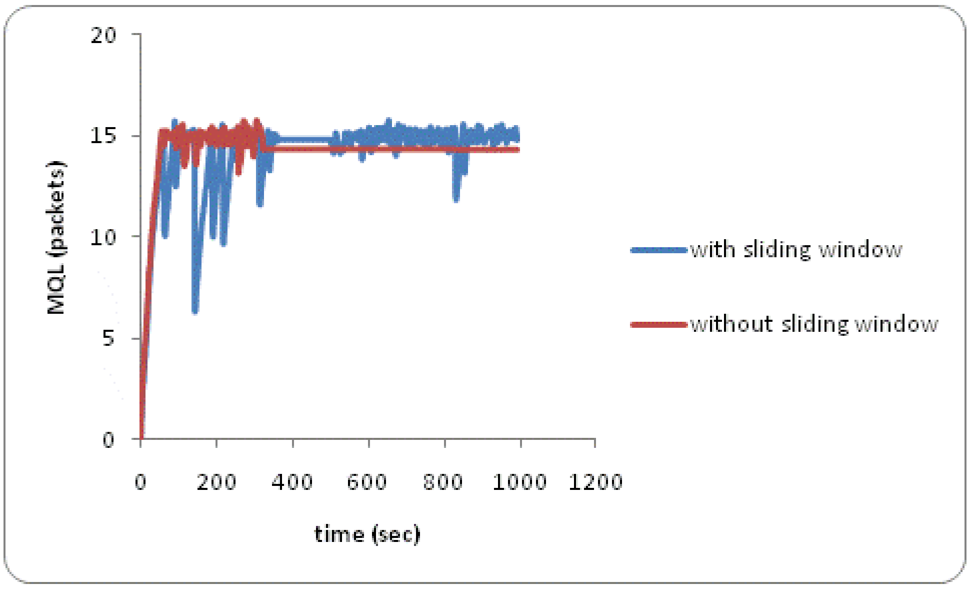


Figure 6.26: The MQL after applying the sliding window algorithm using limit=20x10^6

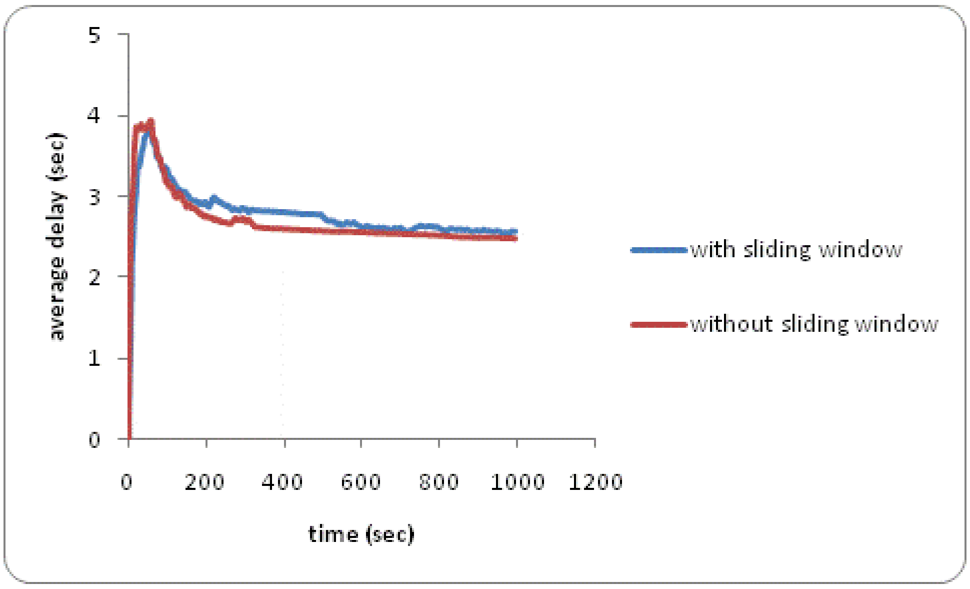


Figure 6.27: The average delay after applying the sliding window algorithm using limit=20x10^6

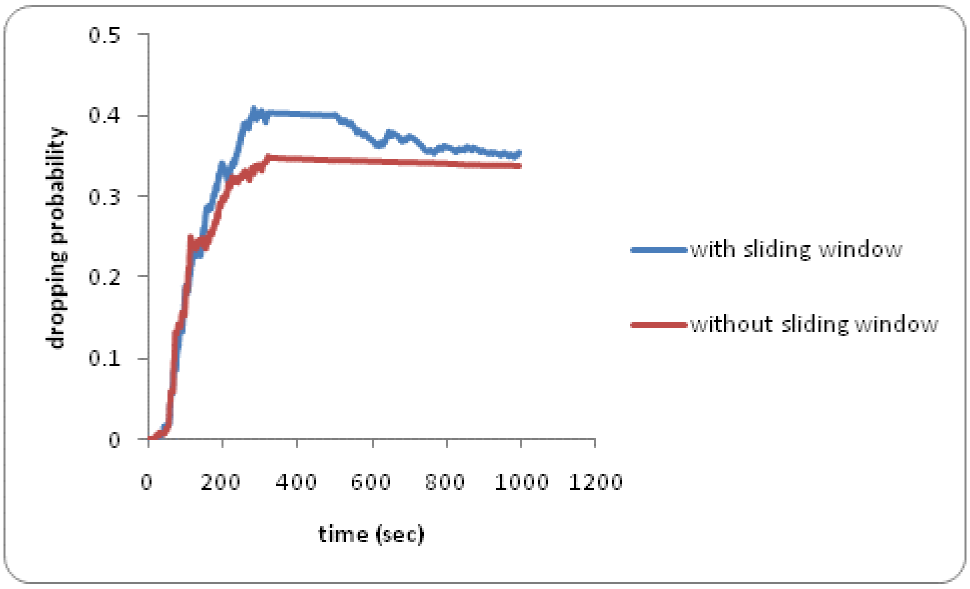


Figure 6.28: The dropping probability after applying the sliding window algorithm using limit=20x10^6

Setting the gap length has been done by trial and error and it showed that using small gaps is not good as it causes instability in the measurements of the number of packets as it is shown in Figures (6.30) and (6.31) while the results obtained in Figure (6.29) are more stable. The reason is that by choosing a too small gap the algorithm is more likely to force more sources to the OFF state while using large gaps gives more stable measurements as it avoids turning OFF sources too frequently.

In the following sections the effect of other parameters on the performance is examined. To do this, one of the parameters is varied (for example the window length) while the other variables are kept fixed (for example the gap length and the limit). Then one of the fixed variables will be varied and the other variables will be fixed and so on.

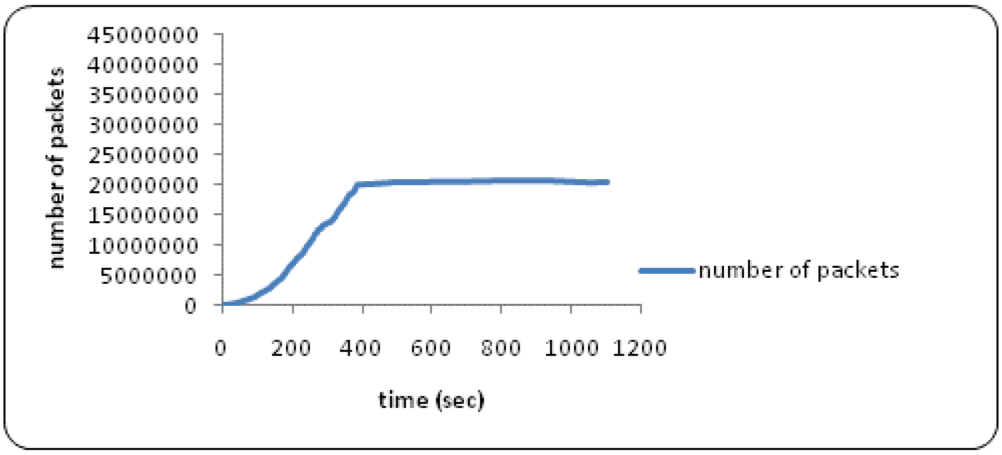


Figure 6.29: The number of packets using gap=30

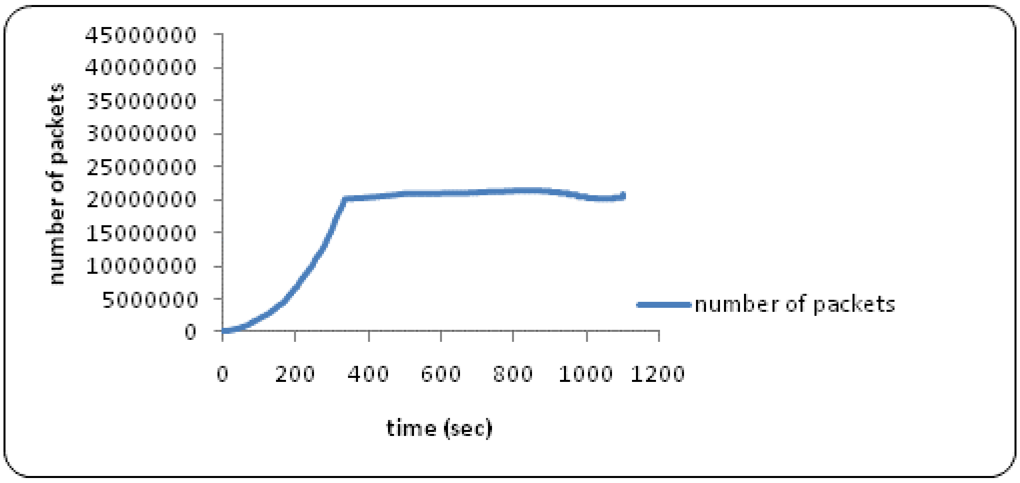


Figure 6.30: The number of packets using gap=25

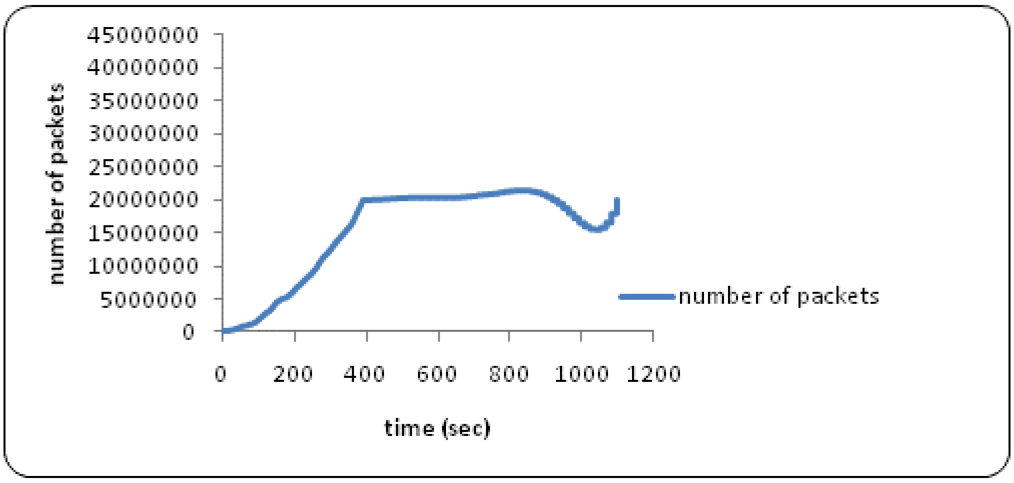


Figure 6.31: The number of packets using gap=15

6.6.2 The Effect of Using Different Limits

To examine the effect of changing the limit value on the performance, different limit values have been used ($5x10^6$, $10x10^6$, $15x10^6$, $20x10^6$ and $25x10^6$) while keeping the *TWL* fixed at 150 and the gap length=50. Figure (6.32) represents the relationship between the limit and the average number of packets. It represents a linear relationship between the limit value and the value obtained for the average number of packets.

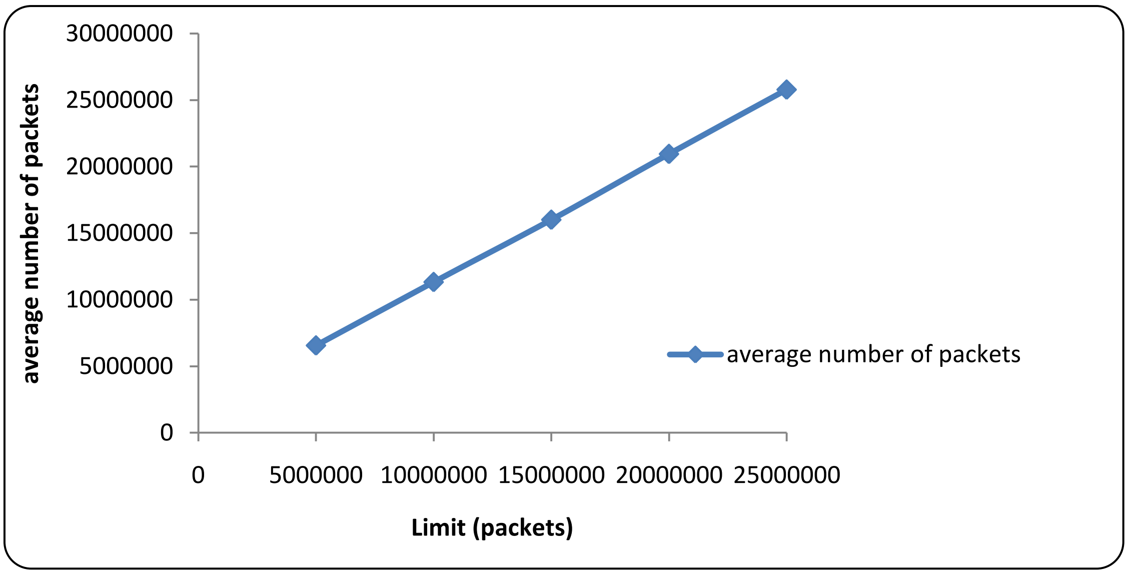


Figure 6.32: The effect of changing the limit on the average number of packets

Setting the limit to a high value increases the number of packets allowed in the system and hence increases the average delay as in Figure (6.33) and also increases the average throughput as shown in Figure (6.34).

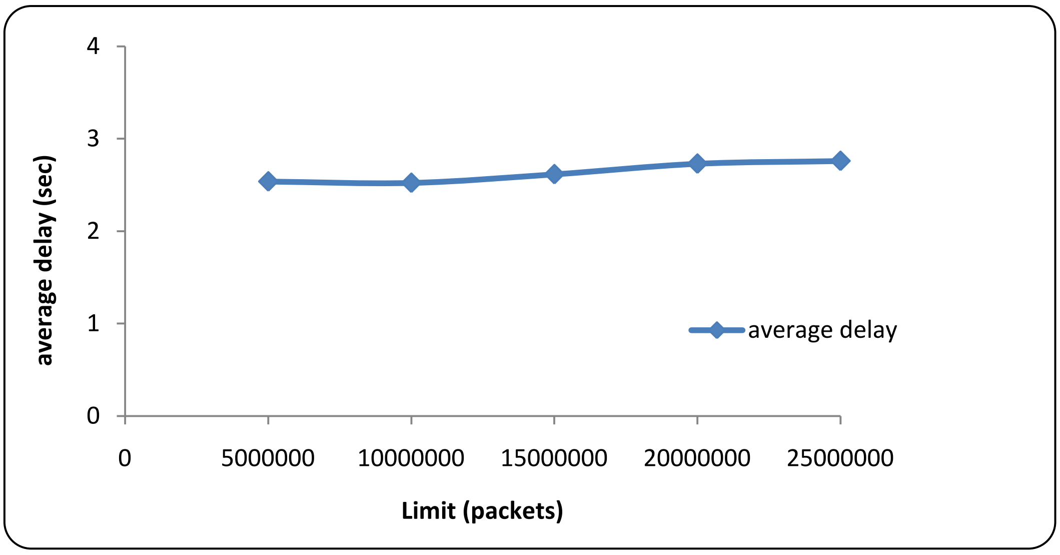


Figure 6.33: The effect of changing the limit on the average delay

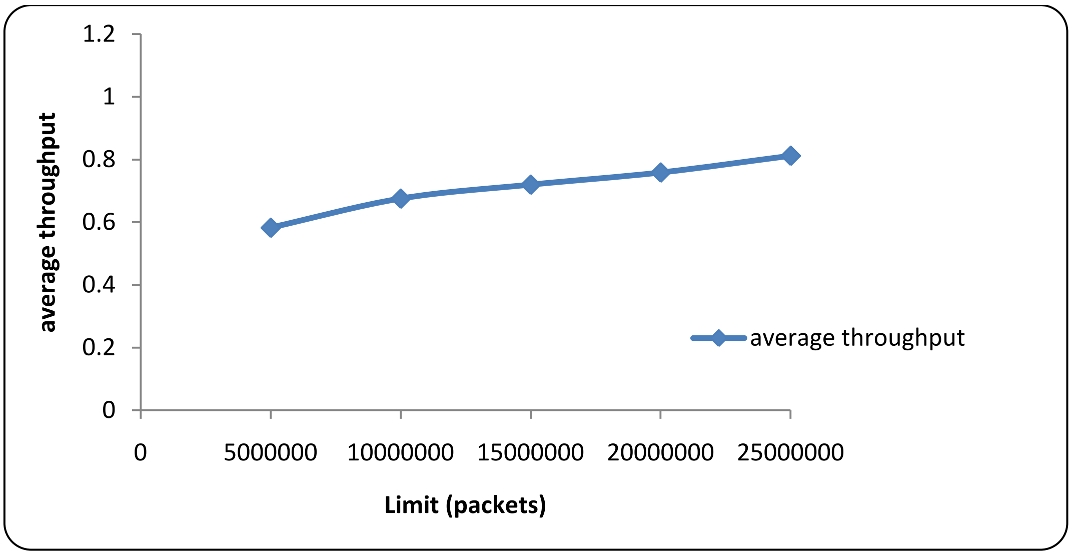


Figure 6.34: The effect of changing the limit on the average throughput

6.6.3 The Effect of Using Different Window Lengths

Figure (6.35) shows the effect of changing the *TWL* on the average number of packets. It is noticeable from Figure (6.35) that setting the window length to any value does not affect on the average number of packets which has been set to $20x10^6$. The window length has been set to (100, 150, 200, 250 and 300) while each window has been cut to gaps of equal length=50.

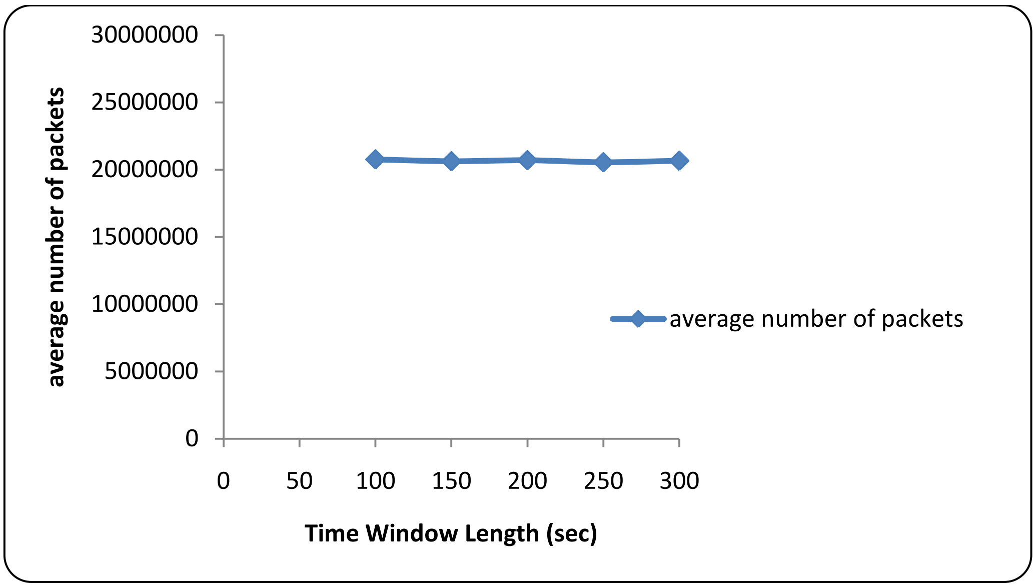


Figure 6.35: The effect of changing the window length on the average number of packets

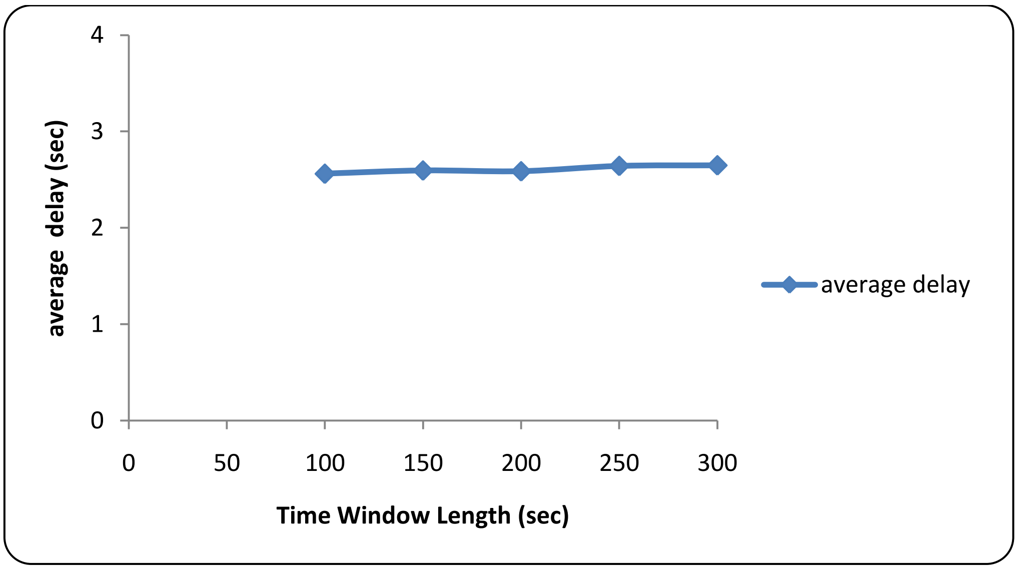


Figure 6.36: The effect of changing the window length on the average delay

Figures (6.36) and (6.37) show the effect of the window length on the average delay and the average throughput, respectively. Clearly, changing the window length does not affect the average delay or the average throughput significantly, and this can be considered as another advantage for the sliding window algorithm as the algorithm can be used with any reasonable window length without having the problem of parameter setting.

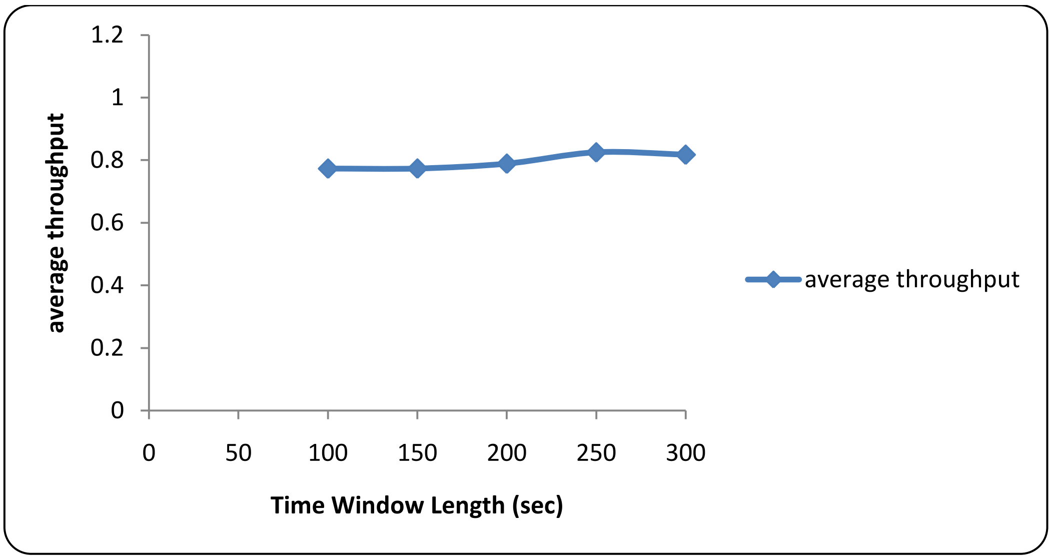


Figure 6.37: The effect of changing the window length on the average throughput

6.6.4 The Effect of Using Different Gaps

In this section the limit has been set to 20x10^6 and the *TWL* has been kept fixed at 150. The gap has been chosen to take the values (10, 15, 25, 30 and 50). These values have been chosen to cut the window into equal parts. As it has been mentioned in Section (6.6.1), setting the gap length to small values impacts on the stability of measuring the average number of packets and this is supported by the results in Figure (6.38). From Figures (6.39) and (6.40) it noticeable that the gap length does not significantly affect the average delay or the average throughput.

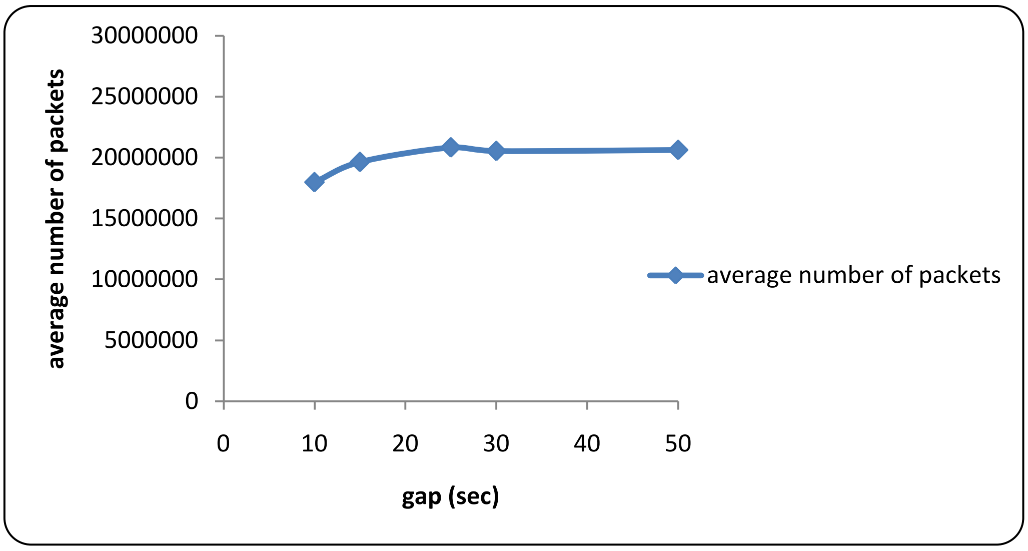


Figure 6.38: The effect of changing the gap length on the average number of packets

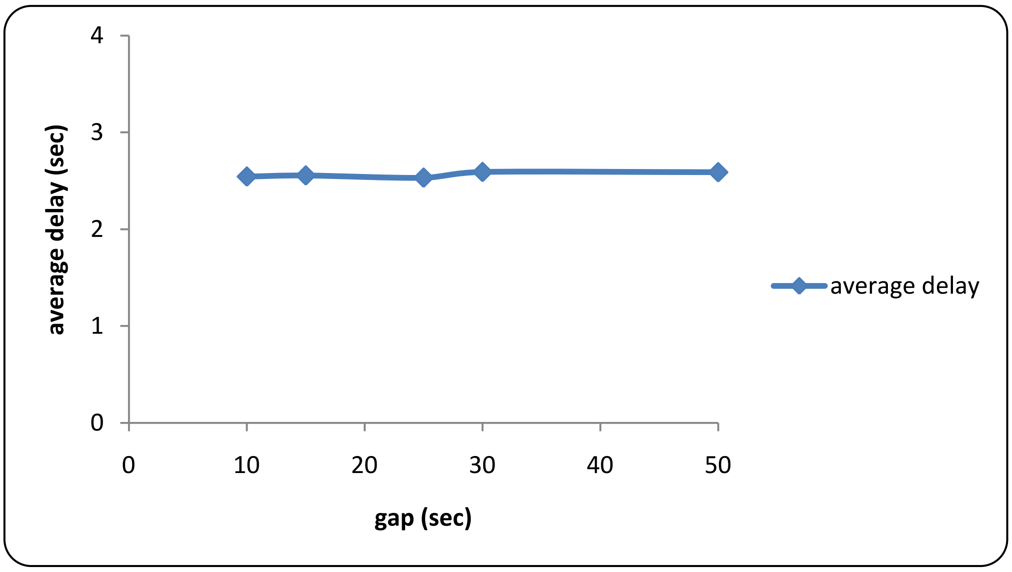


Figure 6.39: The effect of changing the gap length on the average delay

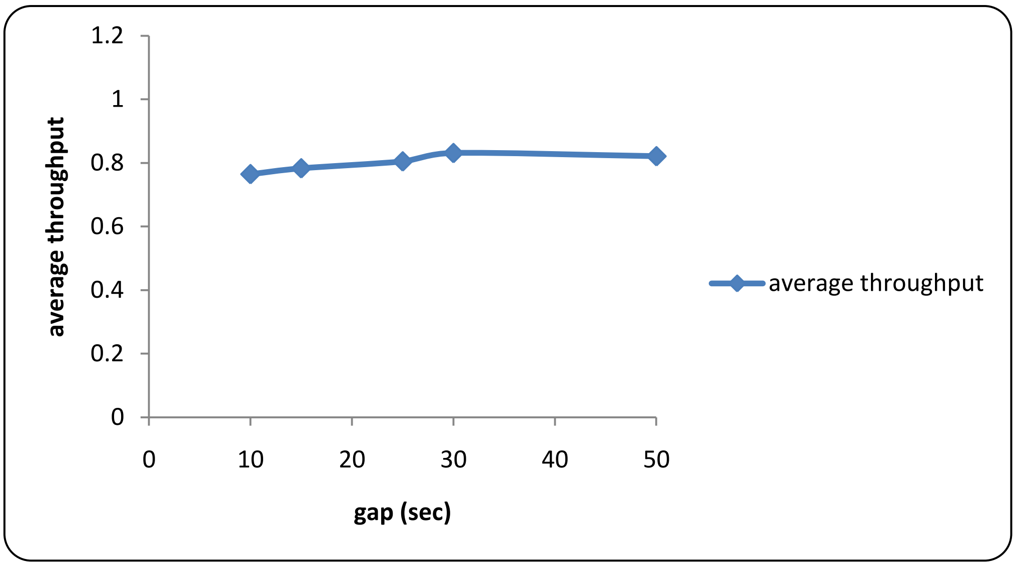


Figure 6.40: The effect of changing the gap length on the average throughput

6.6.5 The Effect of Using Different Gap/Window Ratios

The main reason for applying the sliding window algorithm was to remove the system dependency on the parameter settings. The performance of the model has been shown to be independent on the limit value or the window length for the system used, but suggested that small gaps should be avoided to give better stability of the system measurements. Therefore, it is important to demonstrate that the ratio of the gap length to the window length does not affect the performance. This is to show that the model is flexible in setting the gap length and the window length in a way that removes the dependency of each parameter on the other.

Table 6.4: The results obtained by using variable gap lengths and window length=150

TWL	gap	Ratio (gap/*TWL*)	average number of packets	average throughput	average delay
150	10	0.06667	1.80E+07	0.764564765	2.542475984
150	15	0.1	1.96E+07	0.782914916	2.553714714
150	25	0.16667	20840083.97	0.804687758	2.5305566
150	30	0.2	20543520.43	0.83093143	2.589305047
150	50	0.33333	20629645.02	0.820921183	2.586782804

Table 6.5: The results obtained by using variable gap lengths and window length=300

TWL	gap	Ratio (gap/*TWL*)	average number of packets	average throughput	average delay
300	20	0.06667	2.15E+07	0.77335463	2.51100771
300	30	0.1	21642378.39	0.761135461	2.548404156
300	50	0.16667	21317925.77	0.749999518	2.611807451
300	60	0.2	2.07E+07	0.786421746	2.610130056
300	100	0.33333	2.09E+07	0.780905776	2.659269695

Table (6.4) gives the results obtained by fixing the *TWL* at 150 and changing the gap length. It provides the relationship between the ratio (gap / *TWL*) and the average number of packets, the average delay and the average throughput. Table (6.5) keeps the same ratios as in table (6.4) by fixing the *TWL* at 300 and varying the gap length.

The results obtained in Figures (6.41), (6.42) and (6.43) show that having the same ratios gave similar performance, regardless of changing the window length or the gap length. The results obtained therefore, suggested that the model is not dependant on the parameter settings and that the average number of packets, the average delay and the average throughput do not substantially depend on the ratio of the gap length to the window length.

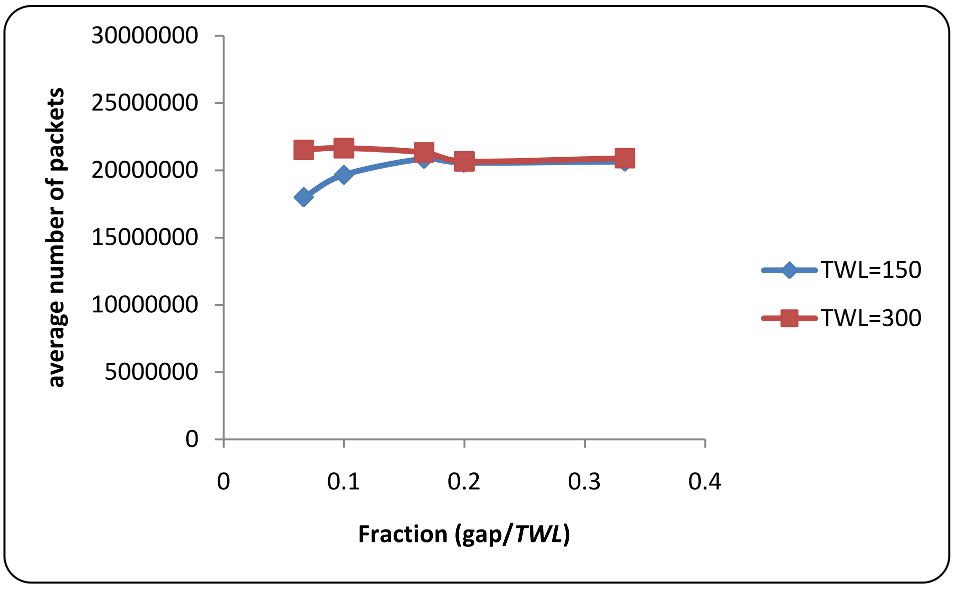


Figure 6.41: The relationship between the average number of packets and the ratio (gap/*TWL*)

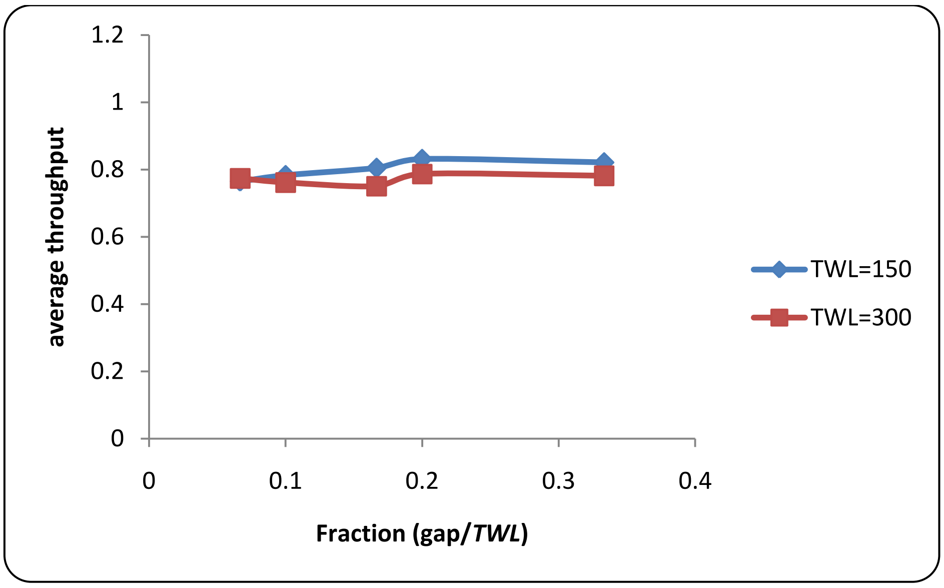


Figure 6.42: The relationship between the average throughput and the ratio (gap/*TWL*)

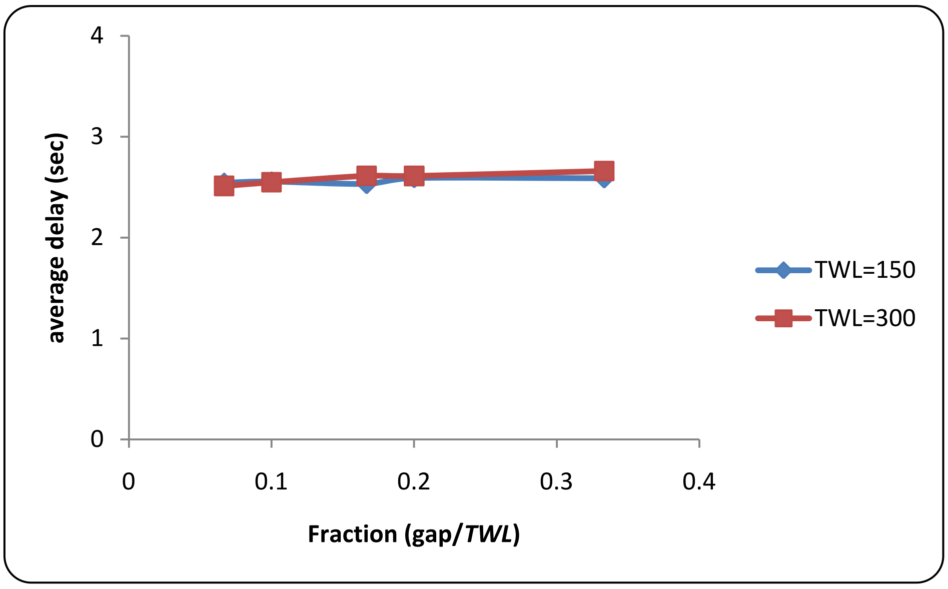


Figure 6.43: The relationship between the average delay and the ratio (gap/*TWL*)

6.7 Summary

Predicting the future traffic levels from past observations when the traffic exhibits LRD appears a useful way to control congestion. The proposed model introduces a new algorithm for congestion prediction. The congestion predictor would be appropriate for implementation at edge routers where LRD is more prevalent. The proposed approach uses a feedback control strategy which uses the mean time spent ON for each node as an indicator of which node is causing congestion. It forces the node with the highest mean time spent ON to the OFF state to avoid congestion in the not-too-distant future. The feasibility of applying the algorithm under self-similar and LRD conditions can thus be effectively utilized for congestion control. The algorithm provided better performance in terms of the average delay, the MQL, the dropping probability and in controlling the number of packets.

The drawbacks were that the MQL was high and turning the offending sources OFF caused degradation in the throughput. The throughput/good-put has been lowered because flows were selectively dropped. Also the performance of the model appeared to be dependent on the limit value. It has been shown that the lower the limit value the better the performance is. But on the other hand, using a lower limit value caused degradation in the throughput. Another drawback is the complexity in measuring the ON periods and in notifying the offending sources about congestion. The additional complexity of the model needs to be considered in any decision to implement the model.

To remove the system dependency on parameter settings, specifically setting the limit value, the algorithm has been modified to a sliding window algorithm. The results obtained after applying the sliding window algorithm showed better performance, especially for the average throughput as the modified algorithm avoided turning OFF sources too often. On the other hand, applying the sliding window algorithm did not cause remarkable improvements in the average delay or the dropping probability. It also did not improve the performance of the MQL. The model also provided better results in terms of the average number of packets. The sliding window algorithm controlled the number of packets in a stable way and the algorithm has been shown to be largely independent of parameter settings for the system investigated.

CHAPTER 7

Maintaining Average Delay Constraints in a Buffer with Time-Varying Arrival Rate

7.1 Introduction

In order to guarantee QoS to diverse Internet services, it is important to employ effective buffer management schemes at Internet routers. Various buffer management mechanisms have been proposed to control traffic congestion and satisfy specified QoS requirements. Most of these studies rely on static thresholds which can be restrictive when they operate with sources with varying arrival rates.

Constraining the average delay to a specified value is a key QoS requirement and one of the most important considerations for real-time services. Bounding delay not only applies in TCP networks [90], but also in other kinds of networks such as wireless networks [91]. An efficient mechanism to control the delay is therefore vitally important if delay is to be constrained to a specified value and jitter is to be minimized. This chapter represents a novel approach for maintaining average delay constraints in a buffer with time-varying arrival rate. The proposed feedback mechanism is used to control the mean delay by adjusting a moveable threshold in order to control the effective arrival rate by randomly dropping packets. The mechanism was also evaluated for a source with LRD traffic characteristics under different arrival rate conditions.

7.2 Impact of a Dynamic Moving Threshold

The aim of applying a dynamic moving threshold is to propose a new adaptive prediction algorithm for AQM that is simple to implement and can maintain the average delay at a constant value when the arrival rate varies with time. To achieve this, a control strategy has been used to bound the delay to a specified value using a dynamic moving threshold. The proposed algorithm depends on the instantaneous queue length to switch the arrival rate at appropriate times by dynamically adjusting the queue threshold. The instantaneous queue length has been used rather than using the average queue length because the possibility of buffer overflow will be reduced if congestion is detected using the instantaneous queue length [92].

7.2.1 The Feedback Control Strategy

It is assumed that the buffer has a finite capacity of K packets, including the server, with two thresholds (L_1) and (L_2) as shown in Figure (7.1). The queueing discipline is FIFO. When the number of packets in the buffer is less than the minimum threshold (L_1), there is no dropping and the source operates normally. If the number of packets exceeds the maximum threshold (L_2), then the source is signalled to stop sending packets by dropping all the subsequent packets. Packet transmission can commence after the next departure (service completion). In this way the majority of packet loss due to buffer overflow might be avoided. If the number of packets in the system at time t falls between the first threshold (L_1) and the second threshold (L_2), then the arriving packets are dropped with dropping probability $P_d(t)$.

$$P_d(t) = \max_p \frac{(q(t) - L_1)}{(L_2 - L_1)} \tag{7.1}$$

max_p is the maximum dropping probability and $q(t)$ is the instantaneous queue length at time t.

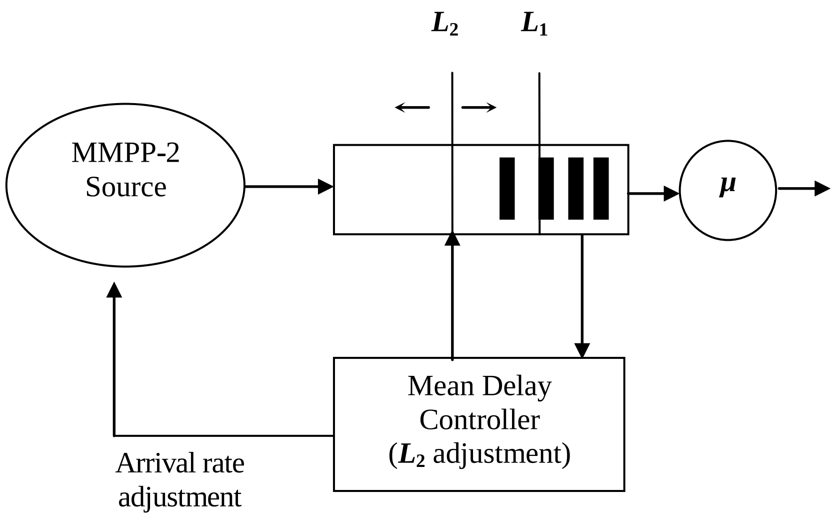


Figure 7.1: Schematic diagram of the mean delay controller model

The parameters used in the feedback control mechanism are:

D_T: target mean delay

D_M: measured mean delay

λ_1: measured mean arrival rate over each time window (effective arrival rate)

λ_2: measured mean arrival rate at L_2

μ: service rate

L_1: first threshold (fixed)

L_2: second threshold (moveable over each time window)

The basic idea for the controller is to measure the mean delay and the actual mean arrival rate over each time window (W). The actual mean arrival rate is measured by dividing the number of arrivals within each time window by the length of the time window. These measurements are used to calculate the new position of the threshold L_2 for the next time window (W+1) in order to maintain the mean delay at the required value. Changing the position of L_2 causes changes in the effective arrival rate by randomly dropping packets. This process is considered as an explicit feedback mechanism to control the mean delay.

7.2.2 The Arrival Process

A basic assumption is made that the arrival rate is varying with time and between step changes follows a Poisson arrival process. A two state Markov Modulated Poisson Process (MMPP) source has been used to provide a time-varying arrival rate in the system. The abrupt changes in the state of the MMPP could be considered to approximately mimic those characteristics of TCP when changing the length of its congestion windows. The use of a single source is to emphasize the large changes in the arrival rate which would not be apparent had multiple sources been used. This is because the use of multiple sources tends to smooth out any fluctuations in the overall traffic levels and this would tend to defeat the objective of the investigation.

The MMPP [93-95] is a doubly stochastic Poisson process where the mean Poisson arrival rate is defined by the state of a Markov chain, as in Figure (7.2). The arrival process has two distinct states, state1 and state2. When the arrival process is in state1, it generates arrivals that follow a Poisson distribution with rate (λ_{11}). Similarly, when the arrival process is in state2, it generates arrivals that follow a Poisson distribution with rate (λ_{22}). The transition rate from state1 to state2 is (γ_1), and from state2 to state1 is (γ_2). Because the arrivals are entering the queue from two different states, it is important to calculate the effective arrival rate within each time window by counting the number of arrivals within each window and dividing it over the window length.

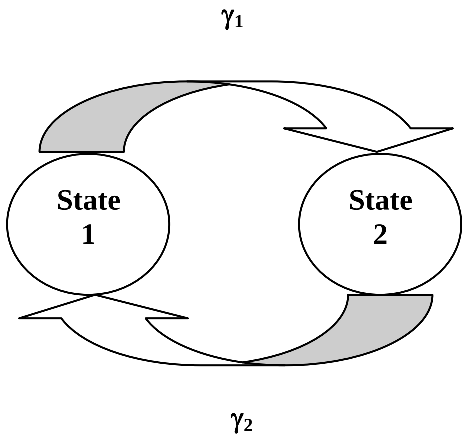


Figure 7.2: A two-state MMPP source

The MMPP is characterized by the transition rate matrix *Q* of the modulating Markov chain and the arrival rate matrix Λ as follows:

$$Q = \begin{bmatrix} -\gamma_{11} & \gamma_{12} \\ \gamma_{21} & -\gamma_{22} \end{bmatrix} \tag{7.2}$$

$$\Lambda = \begin{bmatrix} \lambda_{11} & 0 \\ 0 & \lambda_{22} \end{bmatrix} \tag{7.3}$$

In the proposed continuous-time queueing system, the time has been divided into slots of equal length. These slots (time windows) are assumed large enough to accommodate a relatively large number of events (arrivals and departures). It is also assumed that the RTT from the arrival process to the queue and back to the arrival process is less than one time window. This assumption has been considered to enable the arrival process to switch states from one time window to the next.

The time window length (*TWL*) is assumed to be much smaller than the mean time in each state of the arrival process. This is to allow the system to assume a steady state between changes in the arrival rate, on average, so for most time windows the arrival process will be in the same state.

7.2.3 Performance Metrics

The queueing model used can be considered as a modification of a MMPP/M/1/K queue. However, setting the *TWL* less than the mean time the arrival process remains in each state (for example $TWL = 0.1/\gamma_1$) [96], then over most time windows the model can be viewed as a modification of the M/M/1/K queue. Due to the addition of the two thresholds in the model, the queue needs to be considered as two parts in order to calculate the steady state probabilities. The state transition diagram for the proposed system with the two thresholds L_1 and L_2 is shown in Figure (7.3). The first threshold L_1 is fixed and should be initialized at the beginning of the simulation. The second threshold L_2 can be adjusted to any position in the queue. The balance equations of the continuous-time finite queue can be obtained through the state transition diagram (c.f. Figure (7.3)). The equilibrium probabilities can be expressed in terms of $p_{(0)}$ as follows:

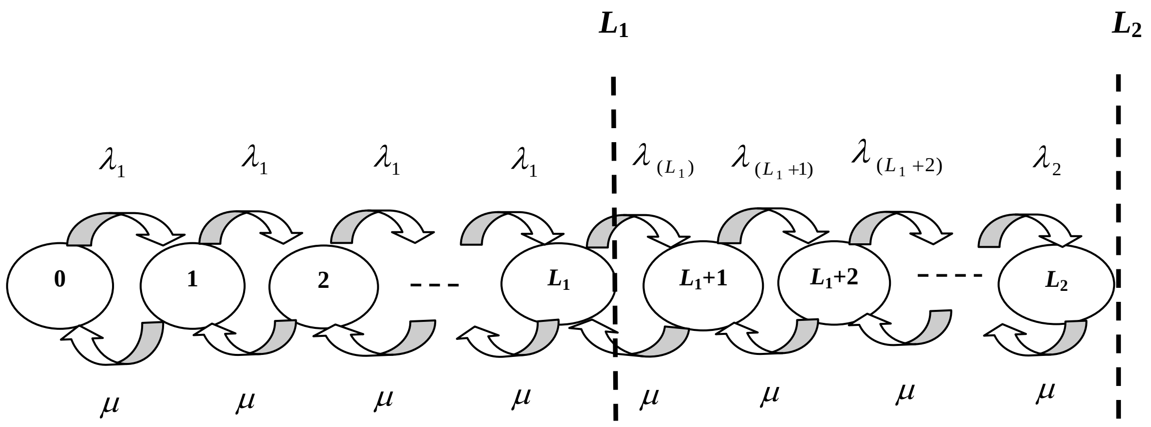


Figure 7.3: The state transition diagram

From state (0) to state (L_1):

$$\lambda_1 \; p_{(0)} = \mu \; p_{(1)} \tag{7.4}$$

$$p_{(1)} = \frac{\lambda_1}{\mu} \; p_{(0)} \tag{7.5}$$

$$\lambda_1 \; p_{(1)} = \mu \; p_{(2)} \tag{7.6}$$

$$p_{(2)} = \frac{\lambda_1}{\mu} \; p_{(1)} = \left(\frac{\lambda_1}{\mu}\right)\left(\frac{\lambda_1}{\mu}\right) p_{(0)} = \left(\frac{\lambda_1}{\mu}\right)^2 p_{(0)} \tag{7.7}$$

From (7.5) and (7.7), the general equation is:

$$p_{(x)} = \rho^x \; p_{(0)} \qquad , x = 0, 1, \cdots, L_1 \tag{7.8}$$

where $\rho = \dfrac{\lambda_1}{\mu}$

From state (L_1+1) to state (L_2):

From (7.8)

$$p_{(L_1)} = \rho^{L_1} \; p_{(0)} \tag{7.9}$$

$$\lambda_{(L_1)} \; p_{(L_1)} = \mu \; p_{(L_1+1)} \tag{7.10}$$

$$p_{(L_1+1)} = \frac{\lambda_{(L_1)}}{\mu} \; p_{(L_1)} \tag{7.11}$$

From (7.9) and (7.11)

$$p_{(L_1+1)} = \frac{\lambda_{(L_1)}}{\mu} \rho^{L_1} \; p_{(0)} \tag{7.12}$$

$$\lambda_{(L_1+1)} \; p_{(L_1+1)} = \mu \; p_{(L_1+2)} \tag{7.13}$$

$$p_{(L_1+2)} = \left(\frac{\lambda_{(L_1+1)}}{\mu} \right) p_{(L_1+1)} \tag{7.14}$$

From (7.12) and (7.14)

$$p_{(L_1+2)} = \left(\frac{\lambda_{(L_1+1)}}{\mu} \right) \left(\frac{\lambda_{(L_1)}}{\mu} \right) \rho^{L_1} \; p_{(0)} \tag{7.15}$$

Generalizing this:

$$p_{(L_1+i)} = \prod_{j=0}^{i-1} \left(\frac{\lambda_{L_1+j}}{\mu} \right) \rho^{L_1} \; p_{(0)} \qquad ,i = 1, 2, \cdots, L_2 - L_1 \tag{7.16}$$

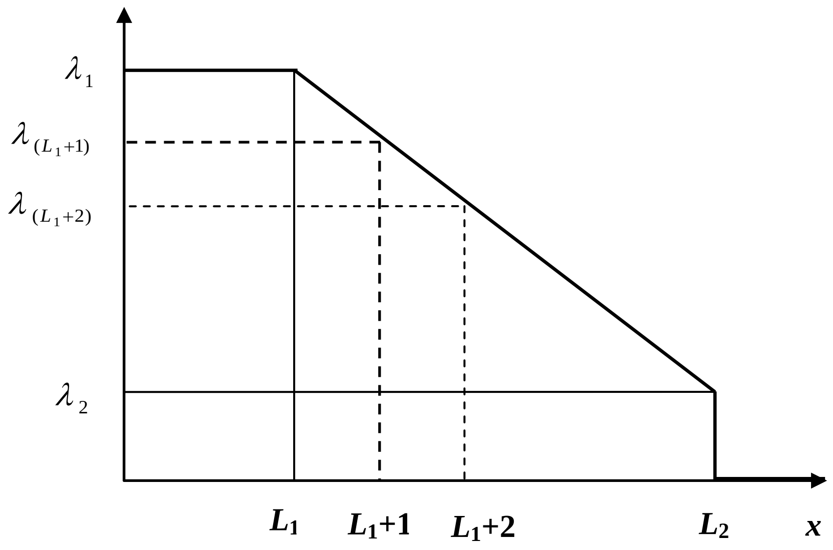


Figure 7.4: The change in the arrival rate due to packet dropping

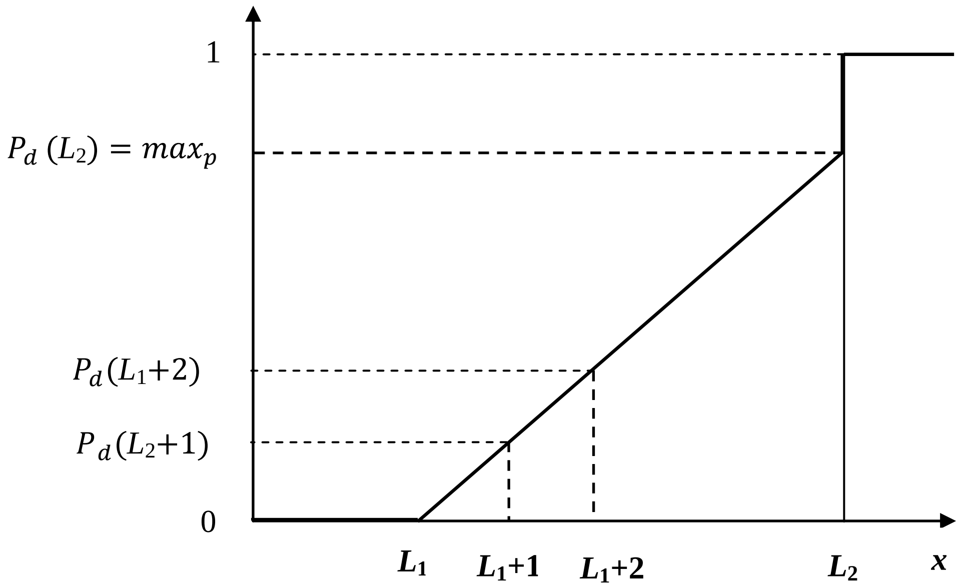


Figure 7.5: The change in the dropping probability due to packet dropping

With reference to Figures (7.4) and (7.5), Equation (7.16) can be written in terms of the slope of the arrival rate characteristics as follows:

$$\text{slope} = \Delta = \frac{\lambda_1 - \lambda_2}{L_2 - L_1} = \frac{\lambda_1 - \lambda_{(L_1+1)}}{L_1 + 1 - L_1} \tag{7.17}$$

$$\Delta = \lambda_1 - \lambda_{(L_1+1)} \tag{7.18}$$

$$\lambda_{(L_1+1)} = \lambda_1 - \Delta \tag{7.19}$$

$$\Delta = \frac{\lambda_1 - \lambda_{(L_1+2)}}{L_1 + 2 - L_1} \tag{7.20}$$

$$\Delta = \frac{\lambda_1 - \lambda_{(L_1+2)}}{2} \tag{7.21}$$

$$\lambda_{(L_1+2)} = \lambda_1 - 2\Delta \tag{7.22}$$

In general:

$$\lambda_{(L_1+k)} = \lambda_1 - k\Delta \quad , k = 0,1,\ldots,L_2 - L_1 \tag{7.23}$$

From (7.16) and (7.23), the equilibrium probability can be solved in terms of $p_{(0)}$:

$$p_{(L_1+y)} = \prod_{i=1}^{y} \left(\frac{(\lambda_1 - (i-1)\Delta)}{\mu} \right) \rho^{L_1} \; p_{(0)} \qquad , y = 1,2,\cdots,L_2 - L_1 \tag{7.24}$$

Then by using the normalization equation $\sum_{i=0}^{L_2} p_i = 1$, $p_{(0)}$ can be obtained as follows:

$$1 = p_{(0)} \left[\sum_{x=0}^{L_1} \rho^x + \rho^{L_1} \sum_{y=1}^{L_2 - L_1} \prod_{i=1}^{y} \left(\frac{(\lambda_1 - (i-1)\Delta)}{\mu} \right) \right] \tag{7.25}$$

By applying the summation formula for the geometric series

$$\sum_{x=0}^{L_1} \rho^x = \frac{1-\rho^{L_1+1}}{1-\rho} \tag{7.26}$$

Substituting (7.26) in (7.25):

$$p_{(0)} = \frac{1}{\left[\left(\frac{1-\rho^{L_1+1}}{1-\rho}\right) + \rho^{L_1} \sum_{y=1}^{L_2-L_1} \prod_{i=1}^{y} \left(\frac{(\lambda_1-(i-1)\Delta)}{\mu}\right)\right]} \tag{7.27}$$

The special case for $p_{(0)}$ when $\lambda_1=\mu$ is given by:

$$p_{(0)} = \frac{1}{\left[(1+L_1) + \sum_{y=1}^{L_2-L_1} \prod_{i=1}^{y} \left(\frac{(\lambda_1-(i-1)\Delta)}{\mu}\right)\right]} \tag{7.28}$$

The *MQL* is given by:

$$MQL = E[N] = \sum_{x=0}^{L_1} x\,p_{(x)} + \sum_{n=L_1+1}^{L_2} n\,p_{(n)} \tag{7.29}$$

$$MQL = p_{(0)} \left[\sum_{x=0}^{L_1} x\rho^x + \rho^{L_1} \sum_{n=L_1+1}^{L_2} n \prod_{i=1}^{n-L_1} \left(\frac{(\lambda_1-(i-1)\Delta)}{\mu}\right)\right] \tag{7.30}$$

The target now is to find the value of L_2 to achieve the target delay D_T for this finite buffer. By applying Little's law [97]:

$$D_T = \frac{MQL}{throughput} = \frac{MQL}{\mu(1-p_{(0)})} \tag{7.31}$$

From (7.30) and (7.31):

$$D_T = \frac{p_{(0)} \left[\sum_{x=0}^{L_1} x\rho^x + \rho^{L_1} \sum_{n=L_1+1}^{L_2} n \prod_{i=1}^{n-L_1} \left(\frac{(\lambda_1-(i-1)\Delta)}{\mu}\right)\right]}{\mu(1-p_{(0)})} \tag{7.32}$$

Now L_2 is the only unknown in the following equation:

$$D_T - \left(\frac{p_{(0)} \left[\sum_{x=0}^{L_1} x\rho^x + \rho^{L_1} \sum_{n=L_1+1}^{L_2} n \prod_{i=1}^{n-L_1} \left(\frac{(\lambda_1 - (i-1)\Delta)}{\mu} \right) \right]}{\mu\,(1 - p_{(0)})} \right) = 0 \tag{7.33}$$

Equation (7.33) forms the core of the proposed feedback control algorithm. The bisection method [98] has been used in order to find the roots of this equation to obtain the threshold position L_2 for the next time window (W+1). Then L_2 should be moved to the new position to keep the mean delay around its target.

7.2.4 Performance Validation

In order to test and validate the analytical model used as the basis for the proposed model, a test model has been implemented. In the test model instead of using a dynamic moving threshold, two fixed thresholds have been used to ensure that the model is working correctly in a steady state. The source used has an arrival rate that follows a Poisson distribution. The arrival rate has been varied only to mimic the changes in the arrival rate within each time window that happens as an effect of the MMPP source changing states. The values of the configuration parameters are summarized in Table (7.1).

Figure (7.6) represents the normalized throughput obtained from the analytical model compared with the simulation model and it is clear that they are matching. Figure (7.7) shows that the mean delay obtained from the simulation is also matching with the mean delay obtained using the analytical model. Figure (7.8) also indicates a good match between the dropping probability results obtained from simulation and analytical model.

Table 7.1: Validation configuration parameters

Parameter	Value
max_p	0.1
L_1	5
L_2	15
queue size	40
μ	5

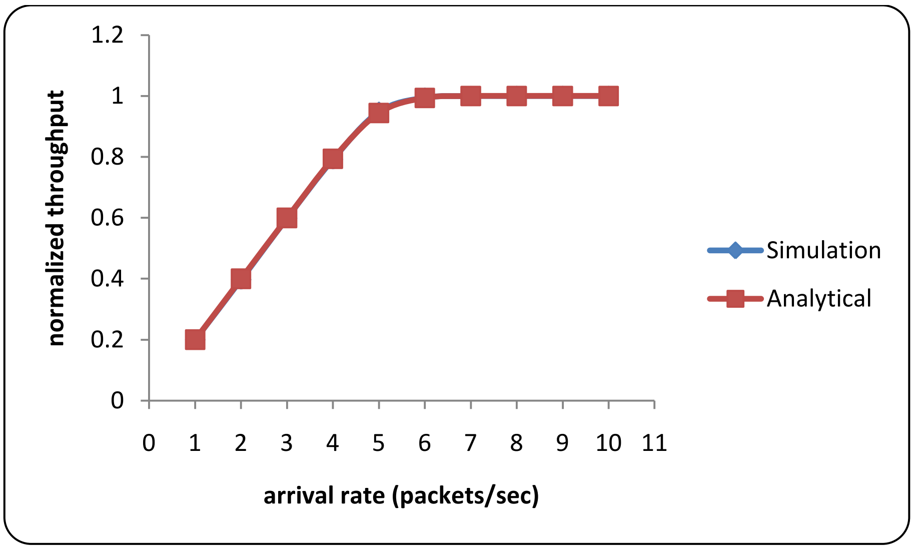


Figure 7.6: The normalized throughput vs. traffic load

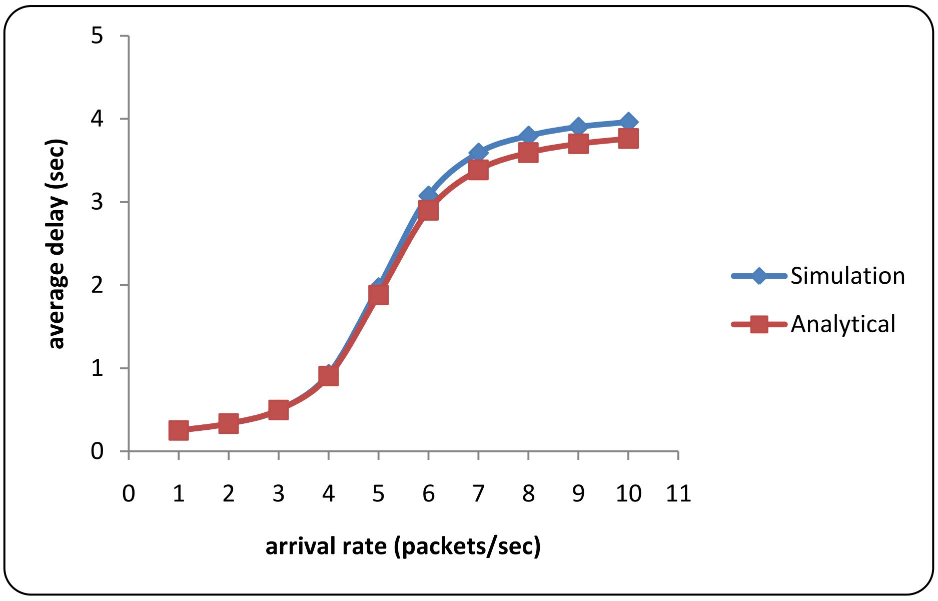


Figure 7.7: The mean delay vs. traffic load

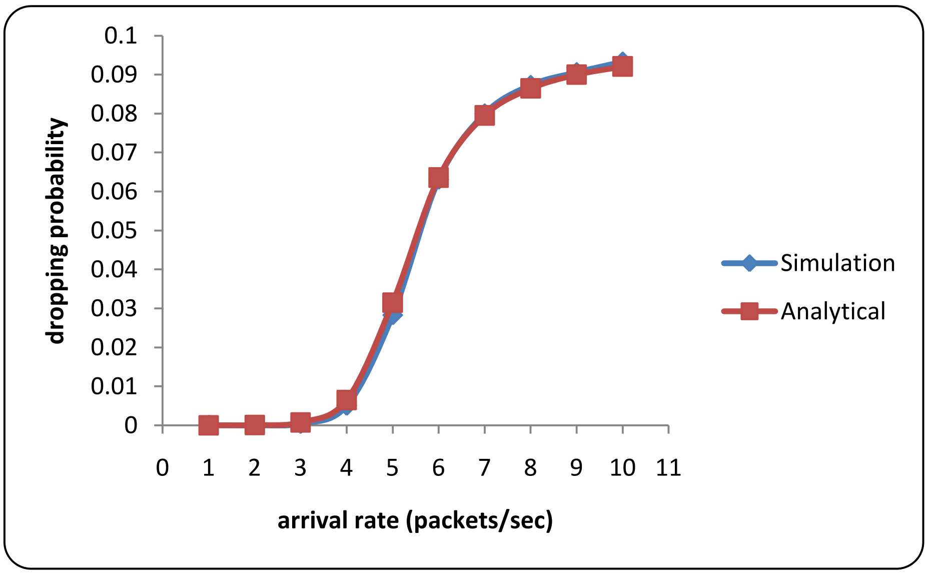


Figure 7.8: The dropping probability vs. traffic load

In Figure (7.8) when the arrival rate is less than the service rate there is no dropping because there is no congestion. But when the arrival rate is approximately equal to the service rate then in the simulation, statistical fluctuations in the arrival rate will switch the system to a congestion situation and back again throughout the simulation. When in a congestion situation the queue will rapidly build up and packets will be dropped. In the analytical model, this remains in a steady state throughout the modelling process thus resulting in packets being dropped only between L_1 and L_2. In the simulation the dropping probability increases when the arrival rate exceeds the service rate till it reaches its maximum value (0.1) when all the subsequent packets are dropped. In the modelling process the dropping probability is obtained as follows:

$$\text{Dropping probability (analytical)} = \sum_{x=L_1+1}^{L_2} p_{(x)}\, P_d(x) \tag{7.34}$$

$P_d(x)$ is the probability that packet x will be dropped, and can be calculated using Figure (7.9) as follows:

$$P_d(x) = \max{}_p \frac{(x - L_1)}{(L_2 - L_1)} \tag{7.35}$$

$p_{(x)}$ can be obtained using Equation (7.24) as follows:

$$p_{(x)} = \prod_{i=1}^{x-L_1} \left(\frac{(\lambda_1 - (i-1)\Delta)}{\mu} \right) \rho^{L_1} p_{(0)} \tag{7.36}$$

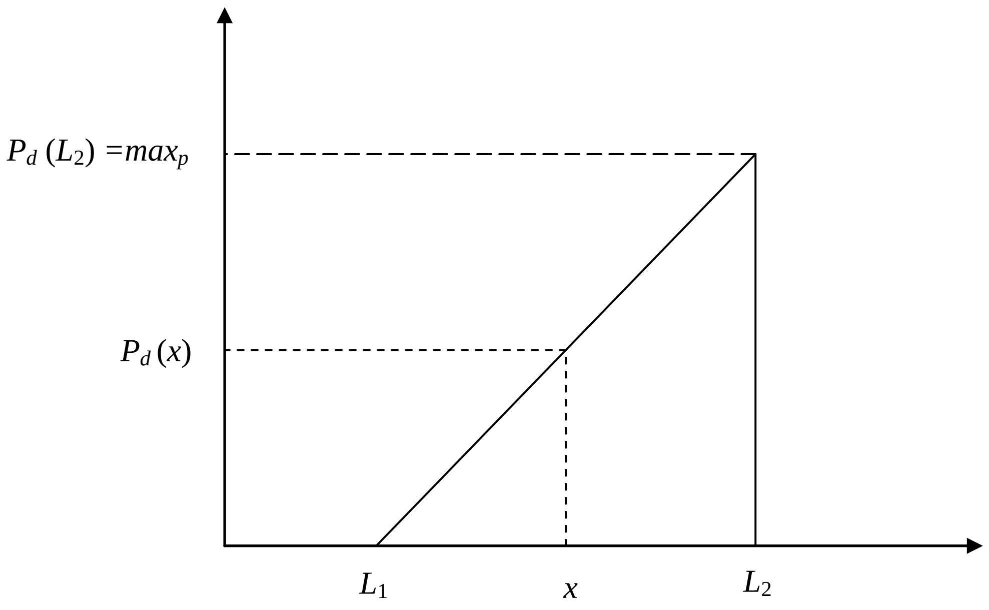


Figure 7.9: The calculation of the dropping probability $P_d\,(x)$

In the simulation model the dropping probability is calculated between L_1 and L_2 as follows:

$$\text{Dropping probability} = \frac{\text{total number of packets dropped in the system}}{\text{total number of packets arriving to the system}} \tag{7.37}$$

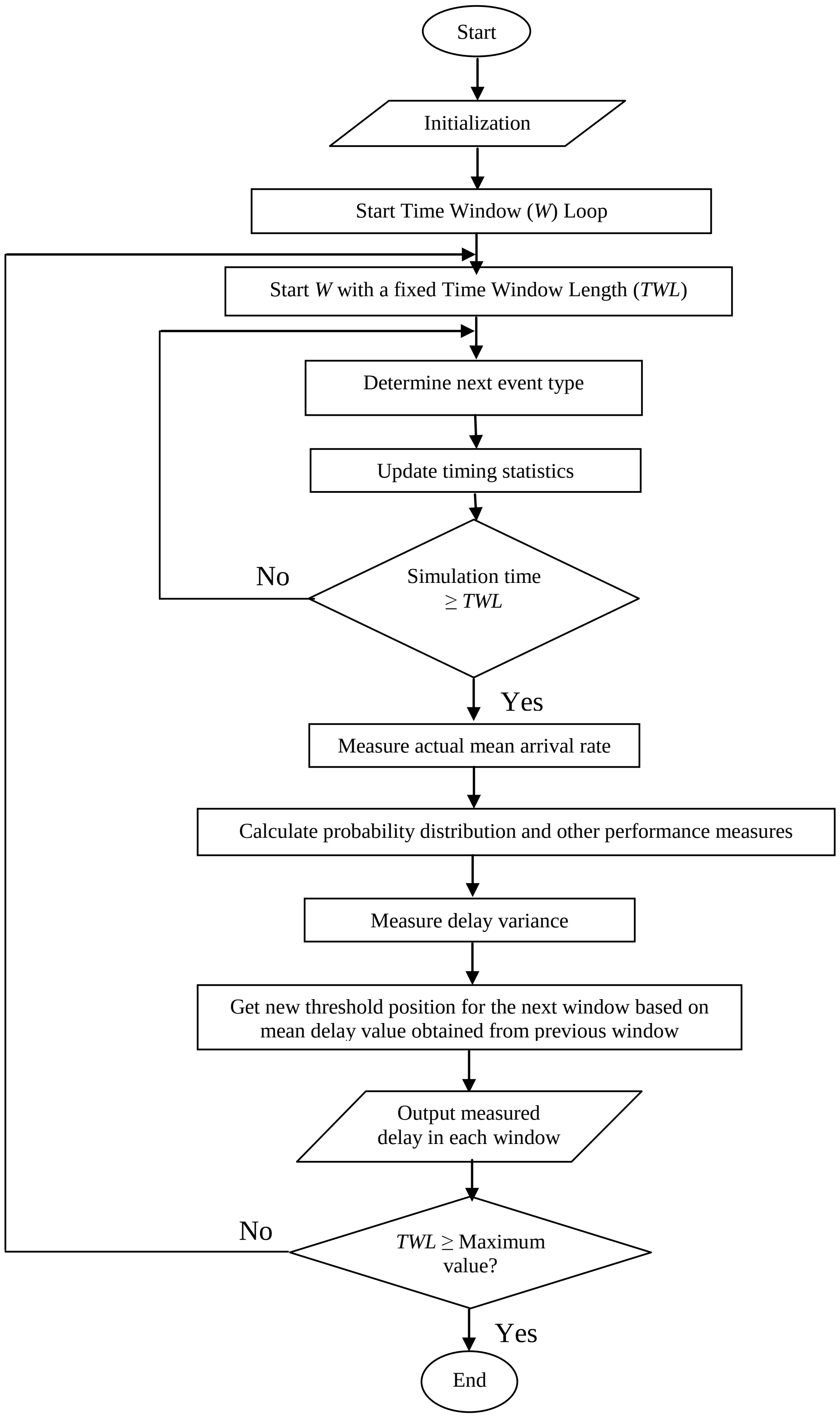

Start
Initialization
Start Time Window (*W*) Loop
Start *W* with a fixed Time Window Length (*TWL*)
Determine next event type
Update timing statistics
Simulation time ≥ *TWL*
No
Yes
Measure actual mean arrival rate
Calculate probability distribution and other performance measures
Measure delay variance
Get new threshold position for the next window based on mean delay value obtained from previous window
Output measured delay in each window
TWL ≥ Maximum value?
No
Yes
End

Figure 7.10: Flowchart of the mean delay controller model

7. 2.5 Simulation Results

A discrete event simulation has been implemented using Java programming to assess the performance of the proposed model. The simulation time has been divided into time windows of fixed length. The bisection method has been used to solve Equation (7.33) at the end of each time window to find the new position for L_2 that bounds the delay around its target value. A flowchart for the simulation is given in Figure (7.10). The values of the simulation parameters are summarized in Table (7.2). The two arrival rates have been chosen higher than the service rate to demonstrate the effectiveness of the algorithm in constraining an increasing delay. The values of L_1 and max_p have been chosen as recommended in [4].

Table 7.2: The mean delay controller configuration parameters

Parameter	Value
max_p	0.1
L_1	5
D_T	5
queue size	40
λ_{11}	10
λ_{22}	6
μ	5
γ_1	0.02
γ_2	0.01

Figure (7.11) represents the measured mean delay (D_M) compared with the target mean delay (D_T). The measured mean delay is calculated as the time spent in the system by all packets divided by the total number of packets served at the end of each time window [69]. Figure (7.11) shows the results obtained by using *TWL*=15, the measured delay achieved is 5.553985 and the variance of measurement is 0.340679. The high variance is a consequence of using a long *TWL*=15. By using a shorter *TWL*=10 it is noticeable that the measured mean delay is approaching the target mean delay which is (5). The mean delay achieved is 5.422202 and the variance of measurement is 0.283505. The results of this are shown in Figure (7.12).

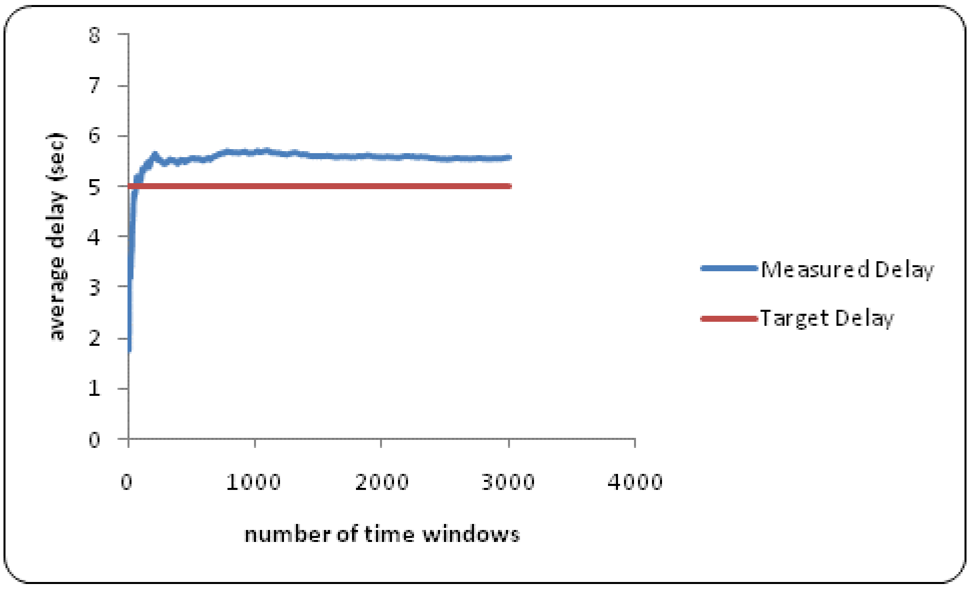


Figure 7.11: Measured mean delay compared with target delay at *TWL*=15

Finding the optimum window length is an important issue and impacts on the accuracy of the measurements. Using a very long window is good in having enough arrivals to accurately measure the mean but these arrivals will most likely be at different arrival rates since the MMPP source might change states within the wide window. Thus, changing the arrival rates will not give accurate measurements of the actual mean arrival rate. Using too small window gives better measurements for the actual mean arrival rate as the source probably is less likely to change its state within the window but it is still not very accurate as there will not be many arrivals to measure the mean. This implies that the accurate tracking of changes in the arrival rate is another factor that impacts on the performance of the simulation. The ideal situation is to find the optimum window length that tracks the variations in the arrival rate as closely as possible so that the arrival rate can be measured accurately.

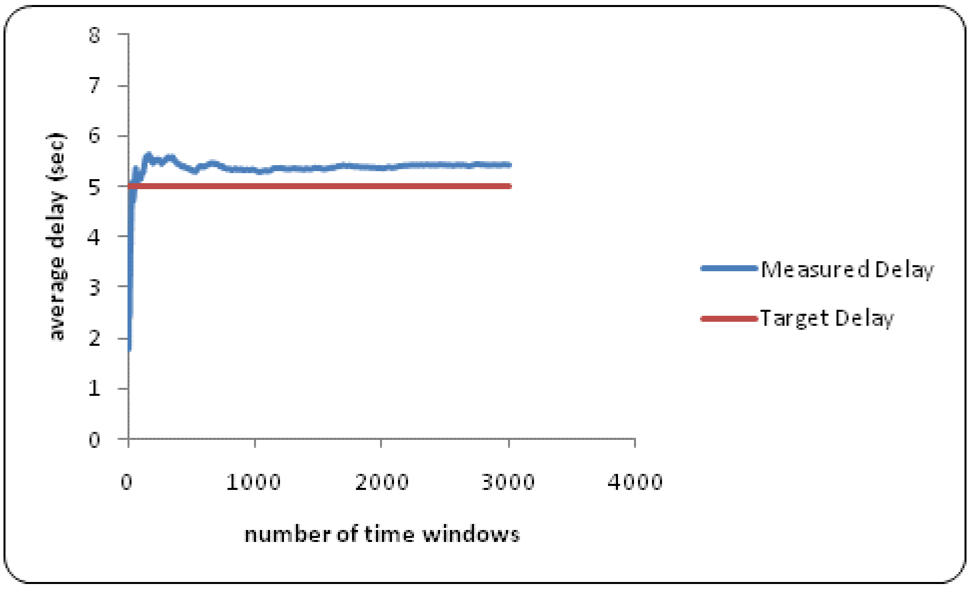


Figure 7.12: Measured mean delay compared with target delay at *TWL*=10

In order to examine the effect of changing the *TWL* on the variance of measurements, different time window lengths have been used. Figure (7.13) shows the delay error variance calculated at different time window lengths with 95% confidence intervals based on ten trials for each value. The delay error variance represents the error in measuring the delay (D_T - D_M) within each time window and is calculated using Equation (7.38):

Delay error variance $= \frac{1}{N}\sum_{W=1}^{N}\left(D_T - D_M\right)^2$ (7.38)

N represents the maximum number of time windows used.

From Figure (7.13) it is noticeable that the delay error variance takes high values if the time window is too short or too long and it reaches its minimum value at *TWL*=7. So *TWL*=7 can be used as the optimum length of the time window. Compared with the results obtained in Figures (7.11) and (7.12) the mean delay achieved in Figure (7.14) was 5.147966 and the variance is 0.166576, which is a smaller value. This also shows that the measured delay can be successfully maintained around its target value if an appropriate length is chosen for the time window.

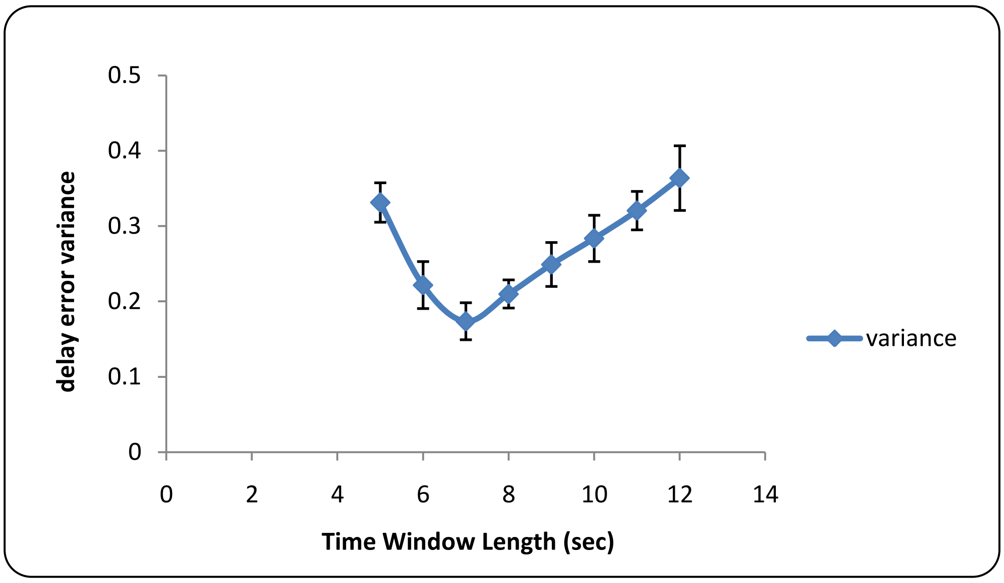


Figure 7.13: Variance of measured mean delay error vs. *TWL*

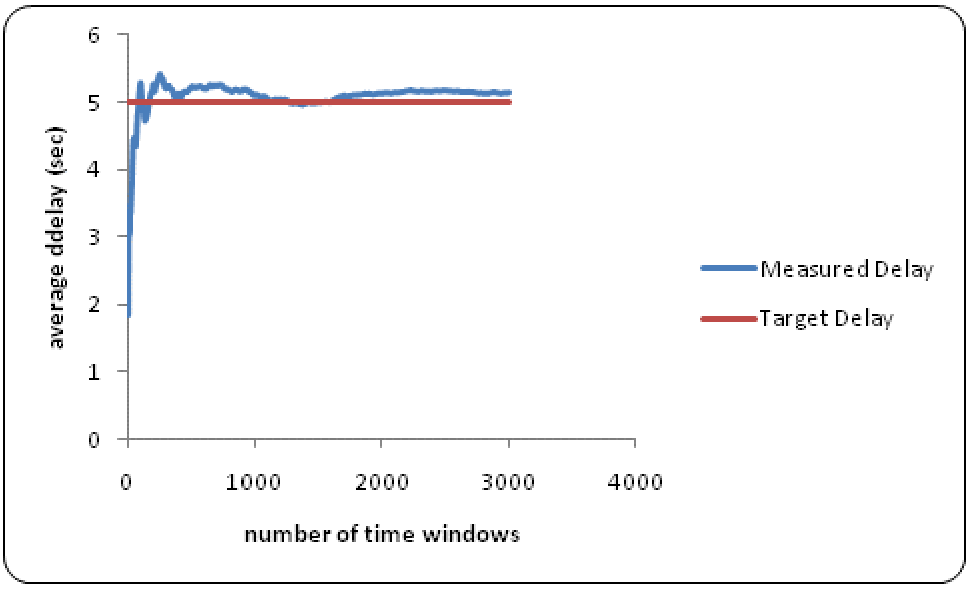


Figure 7.14: Measured mean delay compared with target delay at *TWL*=7

The algorithm has thus been demonstrated to be effective in bounding the average delay at its target value especially in an increasing delay condition when the arrival rates are varying and at high values. To test the performance of the model under lower arrival rate conditions, one of the arrival rates from the MMPP source has been set to a low value λ_{22}=4 while the other rate is kept high at λ_{11}=10 and μ=5. By plotting the delay error variance at different *TWL* values with 95% confidence intervals as presented in Figure (7.15), it is shown that the delay error variance is nearly fixed and is not affected by changing the length of the window. This is likely to be caused by the fact that when $\lambda>\mu$ as with λ_{11}=10 and μ=5, then the delay has been controlled close to the target value which is (5). When $\lambda<\mu$ as with λ_{22}=4 and μ=5, then the delay will inherently be lower than the target. The average delay is therefore expected to be lower than the target delay. This can be seen from the steady state delay results when the Poisson arrival rate is at the λ_{22} value (c.f. Figure (7.7) in Section (7.2.4)). Setting one of the arrival rates to a low value makes the effective arrival rate low most of the time which makes the average delay lower than the target value as it appears from Figure (7.16). Figure (7.16) represents a high variance which equals 3.860855 at *TWL*=9.

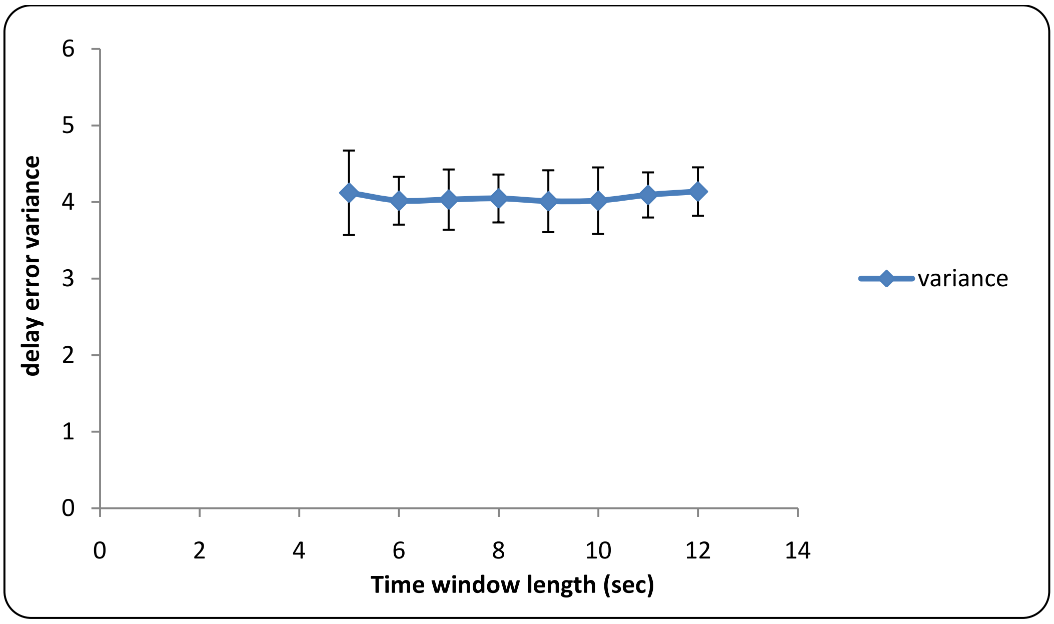


Figure 7.15: Variance of measured mean delay error vs. *TWL* at low arrival rate

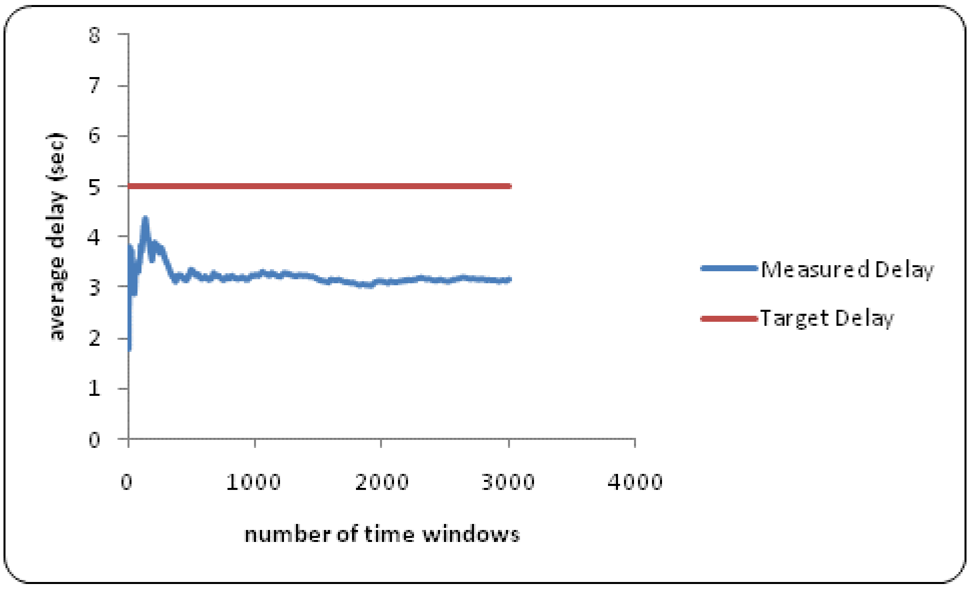


Figure 7.16: Measured mean delay compared with target delay at *TWL*=9

7.3 Impact of Dynamic Threshold Using Pareto Switchover

7.3.1 Motivation

Self-similarity and LRD are the characteristics of some types of modern network traffic. In order to examine the performance of the model under LRD and self-similar conditions. A Pareto swithchover has been used in the former MMPP source to replace the exponential one. This means the transition from state1 to state2 and from state2 to state1 will follow the heavy tailed Pareto distribution. Runing the model under LRD conditions means that the source might spend a much longer time in one of the states than the other. If this is a high arrival rate state then these situations can lead to congestion.

7.3.2 Performance Results

In the simulation model the parameters used are summarized in Table (7.3). Setting the arrival rate at high value in the two states caused high delay error variance. Figure (7.17) represents the delay error variance after using the LRD source. Figure (7.18) shows that at *TWL*=10 the error variance obtained=1.487159 and the measured delay is above the target value.

The lower error variance obtained was at *TWL*=7 and equals 0.983471 as presented in Figure (7.19). It is noticeable that the measured mean delay is still above the target value this is because due to the Pareto distribution the source stays long times in the high rate states and this caused the delay to be higher than the target value. This is because the target delay value was too low to achieve with the high arrival rate. Figure (7.20) represents the relationship between target delay value and the threshold (the measured mean delay).

Table 7.3: The mean delay controller configuration parameters for LRD source

Parameter	Value
max_p	0.1
L_1	5
D_T	5
queue size	40
λ_{11}	10
λ_{22}	6
μ	5
α_{on}	1.2
α_{off}	1.5
$k_{on}= k_{off}$	1

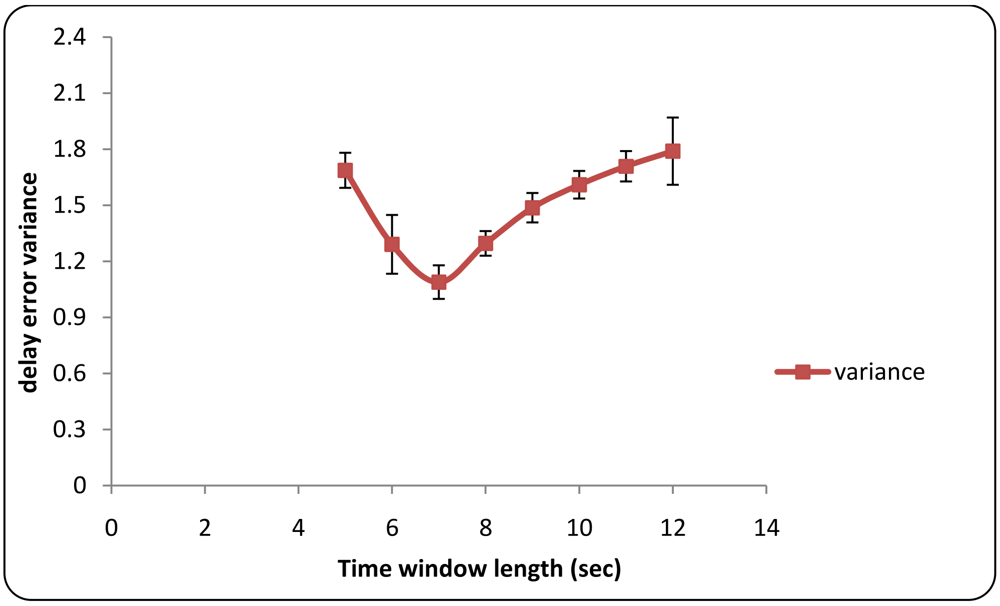


Figure 7.17: Variance of measured mean delay vs. *TWL* using LRD source at high arrival rate

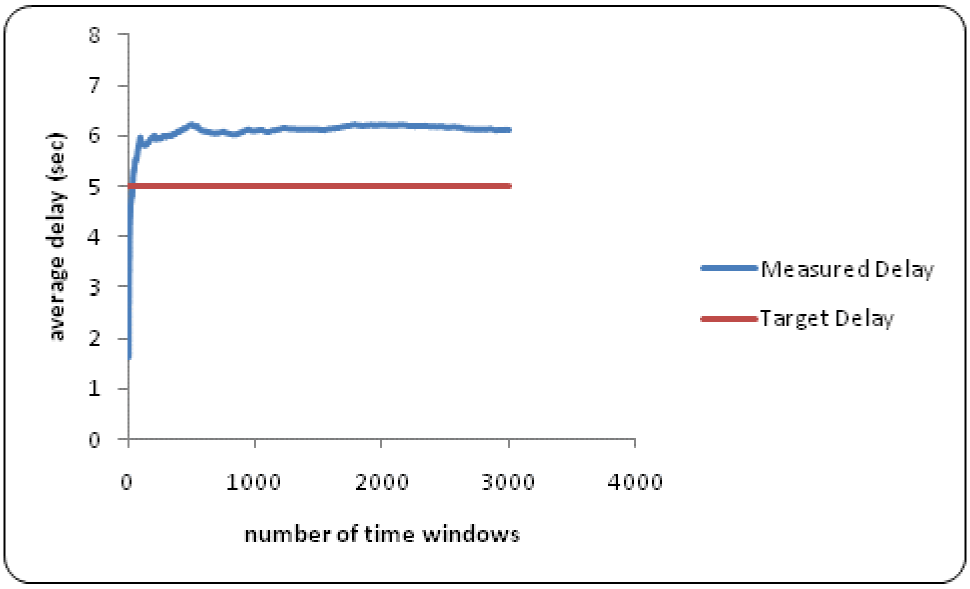


Figure 7.18: Measured mean delay compared with target delay at *TWL*=10 using LRD source

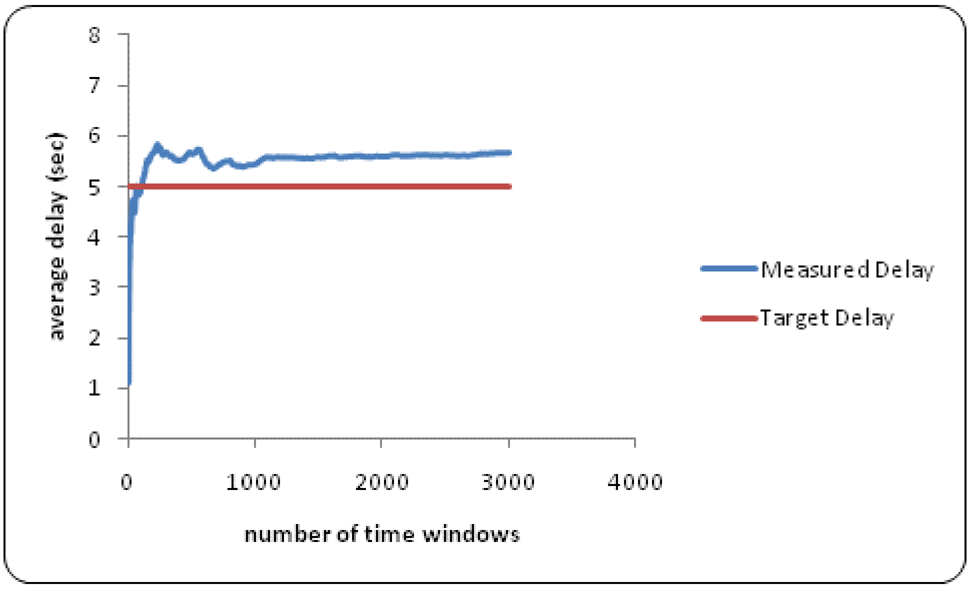


Figure 7.19: Measured mean delay compared with target delay at *TWL*=7 using LRD source

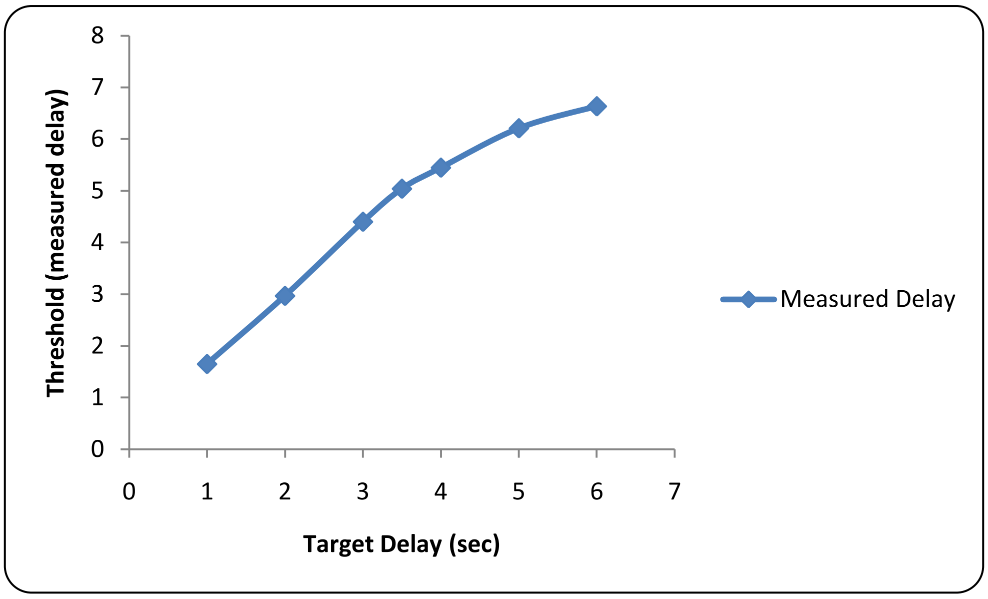


Figure 7.20: Target delay vs. the threshold (measured mean delay)

To test the performance of the model under lower arrival rate conditions, one of the arrival rates of the source has been set to high value λ_{11}=10 and the other rate has been set to low value λ_{22}=4 while the other parameters are kept without change with μ=5. With the Pareto switchover times the results are likely to be biased by the possibility of very long state residence times due to the LRD characteristics of the Pareto distribution. Figure (7.21) shows that the large variations are confirmed by the wide confidence intervals on the delay error variance curve. It shows that the wide intervals of the error bars due to the use of the Pareto swithchover times which causes the source to stay at one of the states longer than the other within a simulation run.

By using *TWL*=5, the variance obtained has high value=0.498559 as in Figure (7.22). The lowest error variance obtained was at *TWL*=9 as shown in Figure (7.23). By using a *TWL*=9 the measured mean delay is close to the target delay value (5) and the variance=0.249665 as shown in Figure (7.23).

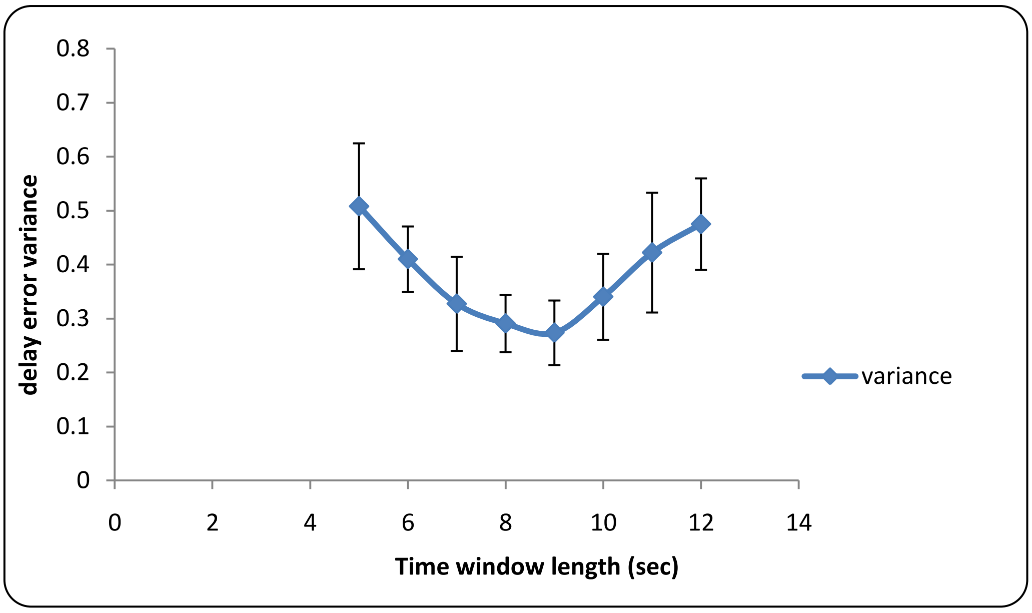


Figure 7.21: Variance of measured mean delay vs. *TWL* using LRD source at low arrival rate

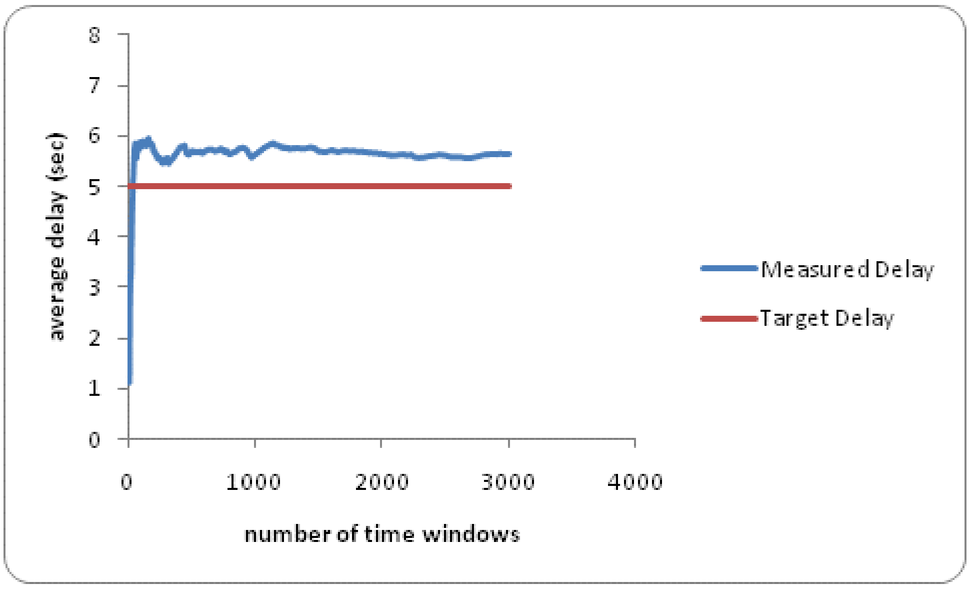


Figure 7.22: Measured mean delay compared with target delay at *TWL*=5 using LRD source at low arrival rate

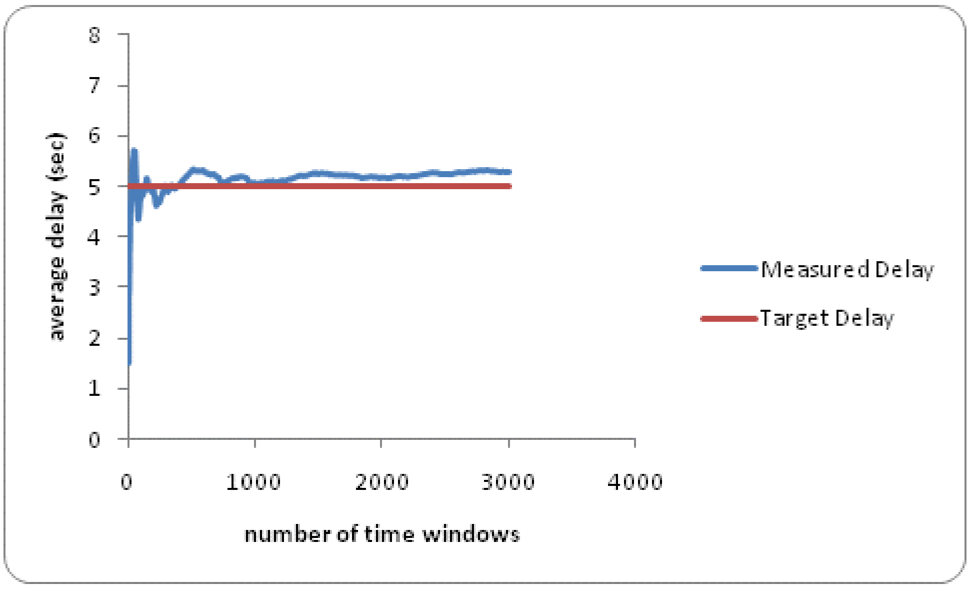


Figure 7.23: Measured mean delay compared with target delay at *TWL*=9 using LRD source at low arrival rate

7.4 Summary

The proposed algorithm is a new algorithm for controlling the mean delay in a buffer with time-varying arrival rate. The model presented can be used to maintain average delay constraints in delay sensitive applications like real-time services for Internet applications. The proposed approach uses a feedback control strategy to adjust the queue threshold dynamically which, in turn, controls the effective arrival rate by randomly dropping packets. In practice, if the system is operating under TCP, then these packet drops will cause the source to slow down as TCP reduces its congestion window size. An equation has been developed that relates the threshold position to the target mean delay over each time window.

The performance of the model depends on the length of the time window and on the ability to accurately measure the changes in the measured mean delay. These issues have been investigated by applying the algorithm under high and low arrival rate conditions. The algorithm has been generalized for other arrival processes, for example the use of Pareto switch over times instead of the exponential ones in order to examine the effects of LRD and self-similarity, which are the characteristics of some types of modern network traffic. The performance of the model has been validated using both simulation and analytical modelling. The results obtained from the simulation matched well with the results obtained from the analytical model.

CHAPTER 8

Conclusions and Future Work

8.1 Conclusions

The main conclusions of this thesis are summarized as follows:

- Due to the rapid growth of communication networks, congestion has become a widespread and persistent problem. This is particularly true of the Internet which has necessitated the need for the deployment of effective congestion control algorithms. Many AQM schemes have been proposed to control congestion but many of these have numerous drawbacks, such as the dependency on parameter settings. Therefore, designing new algorithms to control congestion and predicting the onset of congestion within a network has become an important issue. The algorithms which have been developed and reported within this thesis embody investigations into new methods of controlling congestion and mean delay in communication networks.

- Based on the well known RED algorithm, an algorithm has been developed which sets two thresholds per class in a shared buffer. The model considered is called Dual Class RED (DC-RED) as each arrival class follows the Poisson distribution and the dropping probability of each class is based on the RED algorithm. The effect of varying the parameters of one class on the other class has been investigated, in addition to the effects on the overall performance. The performance analysis has demonstrated the significant impact of the threshold positions on the performance measures of both classes. It has been found that it is very difficult to reach a steady state condition for both classes with the shared buffer.

- Because of the need to identify the onset of congestion at the earliest possible stage, one of the main aims of the thesis has been to develop an algorithm that predicts future traffic levels from past observations. To achieve this aim, a new congestion control algorithm has been developed which makes use of various Internet traffic characteristics, such as self-similarity and LRD, which have not previously been employed in congestion control methods currently used in the Internet. A feedback model with a number of ON/OFF sources has been employed which uses the mean time spent ON for each node as an indicator of which node is causing congestion. The algorithm forced an offending node to the OFF state when the number of packets exceeded a certain limit. The rationale behind this was that any such node that exhibits LRD and which has been ON for an excessively long time is likely to remain ON for a long time in the future. In this context, the algorithm might be considered to incorporate an implicit prediction mechanism. It was found to provide better performance in terms of the average delay, the MQL, the dropping probability and in controlling the number of packets than an equivalent system without the prediction. The drawbacks were that the MQL was high and turning the offending sources OFF caused degradation in the throughput. Also the performance of the model appeared to be dependent on the limit value.

- To remove the system dependency on setting the limit value, the prediction algorithm was modified to incorporate a sliding window mechanism. Modifying the algorithm by the inclusion of a sliding window mechanism was shown to further improve the performance in terms of controlling the total number of packets within the system and improving the throughput. The sliding window mechanism effectively controlled the number of packets in a stable way by avoiding turning OFF sources too often.

- Also considered has been the important problem of maintaining QoS constraints, such as mean delay, which is crucially important in providing satisfactory transmission of real-time services over multi-service networks like the Internet and which were not originally designed for this purpose. The proposed approach used a feedback control strategy to control the mean delay by dynamically adjusting a threshold, which, in turn, controlled the effective arrival rate by randomly dropping packets. Within a TCP environment this would cause the source to reduce its sending rate. The source used was a two state MMPP source in order to model the bursty and correlated traffic and to provide a time-varying arrival rate in the system. This work has been carried out using a mixture of computer simulation and analytical modelling.

- In order to maintain the average delay around its target value, the simulation time was divided into windows of equal length which formed the input to a mean delay control algorithm. An equation was developed that related the threshold position to the target mean delay over each time window. It was found that the performance of the model depended on the length of the time window and on the ability to accurately detect the changes in the arrival rate. These issues were investigated by applying the algorithm under high and low arrival rate conditions and different time window lengths. The simulation results demonstrated that the measured mean delay could be successfully maintained around its target value if an appropriate length was chosen for the time window.

- The mean delay control algorithm was also evaluated for a source with Pareto switchover times instead of the exponential ones in order to examine the effects of LRD and self-similarity. It was found that running the model under LRD conditions resulted in the source spending a much longer time in one of the states than in the other within the relatively limited period of a simulation run. It was also found that because the Pareto switched source stayed for long periods in the high rate states of the source, this caused the delay to be higher than the target value. The measured mean delay reached the target mean delay after setting one of the arrival rates to a low value. The steady state performance of the model was validated by demonstrating a good match between simulation and analytical models.

8.2 Recommendations for Future Work

Further to the work reported in this thesis, several advances are suggested as recommendations for future work as follows:

- For the prediction algorithm, future research should include an investigation of the parameter configurations.
- Future research might also investigate the implementation of target MQL or target mean delay in the prediction algorithm using a moveable threshold or appropriate method.
- The mean delay control algorithm might be modified to be used with multi-class traffic, although this would almost certainly require separate buffers for each type of traffic due to the obvious interdependencies between different traffic classes in a shared buffer environment, which would make an equilibrium state extremely difficult to achieve.
- Modifying the mean delay controller to work as a MQL controller. Also the use of a sliding window instead of a fixed one could be investigated.
- For all the models investigated that make use of the LRD and self-similar traffic, it could be useful to examine the performance under different Hurst parameter values.
- The algorithms might be generalized for other arrival processes and for different queuing disciplines other than the FIFO discipline.
- Future work could also include the development of the simulation models to cater for other network connections such as UDP sources.

References

[1] A. Khan, D. M. Shah, and Z. Xu, "Sync-TCP in high-speed environments," presented at the Proceedings of the 43rd annual Southeast regional conference - Volume 2, Kennesaw, Georgia, 2005.

[2] M. Welzl, *Network Congestion Control: Managing Internet Traffic* Wiley 2005.

[3] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet," RFC 2309 April 1998.

[4] S. Floyd and V. Jacobson., "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Netw.*, vol. 1, pp. 397-413, 1993.

[5] W.-C. Feng, D. D. Kandlur, D. Saha, and K. G. Shin, "A self-configuring RED gateway," in *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, New York, NY, USA, 1999, pp. 1320-1328

[6] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management," *Technical Report,* August 2001.

[7] W.-c. Feng, D. D. Kandlur, D. Saha, and K. G. Shin, "BLUE: A New Class of Active Queue Management Algorithms " In *UMCSE-TR*-387-99, April 1999.

[8] S. Athuraliya, S. H. Low, V. H. Li, and Q. Yin, "REM: active queue management," *Network, IEEE,* vol. 15, pp. 48-53, May 2001.

[9] S. L. Athuraliya, D.   Low, S. , "An enhanced random early marking algorithm for Internet flowcontrol," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies,* 2000, pp. 1425-1434

[10] B. Zheng and M. Atiquzzaman, "DSRED: an active queue management scheme for next generation networks," presented at the Proceedings of the 25th Annual IEEE Conference on Local Computer Networks, 2000.

[11] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of Ethernet traffic (extended version)," *IEEE/ACM Trans. Netw.,* vol. 2, pp. 1-15, 1994.

[12] V. Paxson and S. Floyd., "Wide-area traffic: the failure of Poisson modeling," *SIGCOMM Comput. Commun. Rev.,* vol. 24, pp. 257-268, 1994.

[13] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson, "Self-similarity through high-variability: statistical analysis of ethernet LAN traffic at the source level," *SIGCOMM Comput. Commun. Rev.,* vol. 25, pp. 100-113, 1995.

[14] P. Loiseau, P. Gonçalves, G. Dewaele, P. Borgnat, P. Abry, and P. V.-B. Primet, "Investigating Self-Similarity and Heavy-Tailed Distributions on a Large-Scale Experimental Facility," *IEEE/ACM Transactions on Networking,* vol. pp, pp. 1-14, 2010-03-01 2010.

[15] K. Park and W. Willinger, "Self-Similar Network Traffic: An Overview," in *Self-Similar Network Traffic and Performance Evaluation* K. Park and W. Willinger, Eds., ed New York; Chichester John Wiley & Sons, Inc., 2000, pp. iii-xlix.

[16] Y. Jin, S. Bali, T. E. Duncan, and V. S. Frost, "Predicting properties of congestion events for a queueing system with fBm traffic," *IEEE/ACM Trans. Netw.,* vol. 15, pp. 1098-1108, 2007.

[17] N. G. Duffield and W. Whitt, "A Source traffic model and its transient analysis for network control " *Stochastic Models,* vol. 14, pp. 51 - 78 1998.

[18] N. G. Duffield and W. Whitt, "Network Design and Control Using On/Off and Multilevel Source Traffic Models with Heavy-Tailed Distributions," in *Self-Similar Network Traffic and Performance Evaluation*, K. Park and W. Willinger, Eds., ed: John Wiley & Sons, Inc., 2000.

[19] R. Jain, "Congestion control in computer networks: issues and trends " *Network, IEEE* vol. 4, pp. 24 - 30 May 1990.

[20] W. Stallings, *High-Speed Networks and Internets: Performance and Quality of Service* 2nd ed.: Prentice Hall, 2002.

[21] V. Jacobson, "Congestion avoidance and control," presented at the Symposium proceedings on Communications architectures and protocols, Stanford, California, United States, 1988.

[22] V. Firoiu and M. Borden, "A Study of Active Queue Management for Congestion Control," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* 2000, pp. 1435-1444.

[23] H. J. Fowler and W. E. Leland, "Local area network characteristics, with implications for broadband network congestion management " *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 1139 - 1149 1991.

[24] A. S. Tanenbaum, "Congestion Prevention Policies," in *Computer Networks*, 4th ed: Prentice Hall, 2002, pp. 388 - 389.

[25] S. J. Golestani, "A stop-and-go queueing framework for congestion management," *SIGCOMM Comput. Commun. Rev.*, vol. 20, pp. 8-18, 1990.

[26] R. Jain, K. K. Ramakrishnan, and D.-M. Chiu, "Congestion avoidance in computer networks with a connectionless network layer," in *Innovations in Internetworking*, ed: Artech House, Inc., 1988, pp. 140-156.

[27] Z. Wang and J. Crowcroft, "Analysis of shortest-path routing algorithms in a dynamic network environment," *SIGCOMM Comput. Commun. Rev.*, vol. 22, pp. 63-71, 1992.

[28] S. Keshav, "Congestion control in computer networks," PhD., University of California at Berkeley, 1992.

[29] C.-Q. Yang and A. V. S. Reddy, "A taxonomy for congestion control algorithms in packet switching networks," *Network, IEEE* vol. 9, pp. 34 - 45 Jul/Aug 1995.

[30] J. Nagle, "Congestion Control in IP/TCP Internetworks," RFC 896 January 1984.

[31] R. Jain and K. K. Ramakrishnan, "Congestion avoidance in computer networks with a connectionless network layer: concepts, goals and methodology " in *Proceedings of the IEEE Computer Networking Symposium*, Washington, DC, 1988, pp. 134 - 143.

[32] J. Nagle, "On packet switches with infinite storage," RFC 0970 December 1985.

[33] D. I. Choi, B. D. Choi, and D. K. Sung, "Performance analysis of priority leaky bucket scheme with queue-length-threshold scheduling policy " *Communications, IEE Proceedings-*, vol. 145, pp. 395 - 401 December 1998.

[34] S. J. Golestani, "Congestion-free communication in high-speed packet networks," *IEEE transactions on communications* vol. 39, pp. 1802-1812 December 1991.

[35] N. YIN, S. LI, and T. STERN, "Congestion control for packet voice by selective packet discarding," *IEEE transactions on communications* vol. 38, pp. 674-683 May 1990.

[36] Z. Haas, "Adaptive admission congestion control," *SIGCOMM Comput. Commun. Rev.*, vol. 21, pp. 58-76, 1991.

[37] K. K. Ramakrishnan and R. Jain, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks," *ACM Transactions on Computer Systems*, vol. 8, pp. 158-181, May 1990.

[38] T. Faber, L. H. Landweber, and A. Mukherjee, "Dynamic Time Windows: packet admission control with feedback," presented at the Conference proceedings on Communications architectures & protocols, Baltimore, Maryland, United States, 1992.

[39] G. C. Vinton and E. I. Robert, "A protocol for packet network intercommunication," *SIGCOMM Comput. Commun. Rev.*, vol. 35, pp. 71-82, 2005.

[40] M. Hassan and R. Jain, *High Performance TCP/IP Networking Concepts,Issues, and Solutions*: PEARSON Prentice Hall., 2004.

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

[42] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," RFC 2001 January 1997.

[43] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-down Approach Featuring the Internet* 3rd ed.: Addison Wesley, 2004.

[44] J. Postel, "User Datagram Protocol," RFC 768 August 1980.

[45] C. Bormann, "PPP in a Real-Time Oriented HDLC_like Framing," RFC 2687 September 1999.

[46] W. Simpson, "PPP in HDLC-like Framing," RFC 1662 July 1994.

[47] M. Nabeshima and K. Yata, "Performance improvement of active queue management with per-flow scheduling," *Communications, IEE Proceedings-,* vol. 152, pp. 797- 803, 9 Dec. 2005.

[48] R. Stanojevic, R. N. Shorten, and C. M. Kellett, "Adaptive tuning of drop-tail buffers for reducing queueing delays," *Communications Letters, IEEE,* vol. 10, pp. 570-572, 2006.

[49] A. Guido, K. Isaac, and M. Nick, "Sizing router buffers," *SIGCOMM Comput. Commun. Rev.*, vol. 34, pp. 281-292, 2004.

[50] T. V. Lakshman, A. Neidhardt, and T. J. Ott, "The drop from front strategy in TCP and in TCP over ATM," *INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE,* vol. 3, pp. 1242-1250 24-28 Mar 1996.

[51] A. Mankin, "Random drop congestion control," presented at the Proceedings of the ACM symposium on Communications architectures & protocols, Philadelphia, Pennsylvania, United States, 1990.

[52] E. Hashem, "Analysis of random drop for gateway congestion control," M.S. thesis M.S. , Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1989.

[53] L. Zhang, "A New Architecture for Packet Switching Network Protocols," PhD. Thesis PhD., Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1989.

[54] A. Mankin and K. Ramakrishnan, "Gateway Congestion Control Survey," RFC 1254 August 1991.

[55] C. Mikkel, J. Kevin, O. David, and F. D. Smith, "Tuning RED for Web traffic," *IEEE/ACM Trans. Netw.*, vol. 9, pp. 249-264, 2001.

[56] M. May, J. Bolot, C. Diot, and B. Lyles, "Reasons not to deploy RED," in *IEEE IWQoS'99. Seventh International Workshop on Quality of Service*, London, UK, 1999, pp. 260-262.

[57] M. Vishal, G. Wei-Bo, and T. Don, "Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED," *SIGCOMM Comput. Commun. Rev.*, vol. 30, pp. 151-160, 2000.

[58] T. J. Ott, T. V. Lakshman, and L. H. Wong, "SRED: stabilized RED," in *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies.* , New York, NY, USA, 1999, pp. 1346-1355 vol.3.

[59] L. Dong and M. Robert, "Dynamics of random early detection," presented at the Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication, Cannes, France, 1997.

[60] F. Agharebparast and V. C. M. Leung, "Improving the performance of RED deployment on a class based queue with shared buffers," in *Global Telecommunications Conference, GLOBECOM '01. IEEE* 2001, pp. 2363 - 2367.

[61] M. Parris, K. Jeffay, and F. D. Smith, "Lightweight Active Router-Queue Management for Multimedia Networking," in *Multimedia Computing and Networking, SPIE Proceedings Series*, San Jose, CA, 1999, pp. 162-174.

[62] F. M. Anjum and L. Tassiulas, "Balanced-RED: An Algorithm to Achieve Fairness in the Internet," 1999.

[63] S. Floyd and V. Paxson., "Difficulties in simulating the internet," *IEEE/ACM Trans. Netw.*, vol. 9, pp. 392-403, 2001.

[64] H. M. Deitel and P. J. Deitel, *Java How to Program (7th Edition)*: Prentice-Hall, Inc., 2007.

[65] K. Fall and K. Varadhan. (2000, *The ns Manual* Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.9938

[66] O. C. Site. *OMNeT++*. Available: http://www.omnetpp.org/index.php

[67] V. S. Frost and B. Melamed, "Traffic modeling for telecommunications networks," *Communications Magazine, IEEE*, vol. 32, pp. 70-81, Mar. 1994.

[68] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol, *Discrete Event System Simulation*, 3rd ed.: Prentice Hall, 2000.

[69] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*, 3rd ed.: McGraw-Hill, 2000.

[70] A. O. Allen, *Probability, Statistics, and Queueing Theory with computer science applications*, 1st ed.: Academic Press, 1978.

[71] B. Sumitha, A. L. N. Reddy, Z. Yueping, and L. Dimitri, "Emulating AQM from end hosts," *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 349-360, 2007.

[72] Q. Li and D. L. Mills, "On the long-range dependence of packet round-trip delays in Internet" *Communications, IEEE.*, vol. 2, pp. 1185-1191 7-11 Jun 1998.

[73] S. T. Murad, W. Walter, and S. Robert, "Proof of a fundamental result in self-similar traffic modeling," *SIGCOMM Comput. Commun. Rev.*, vol. 27, pp. 5-23, 1997.

[74] K. Trivedi, *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*, 2nd Edition ed.: JHON WILEY &SONS, INC, 2002.

[75] B. K. Ryu and A. Elwalid, "The importance of long-range dependence of VBR video traffic in ATM traffic engineering: myths and realities," presented at the Conference proceedings on Applications, technologies, architectures, and protocols for computer communications, Palo Alto, California, United States, 1996.

[76] T. Le-Ngoc and S. N. Subramanian, "A Pareto-modulated Poisson process (PMPP) model for long-range dependent traffic," *Computer Communications*, vol. 23, pp. 123-132, 2000.

[77] K. Park and W. Willinger, *Self-Similar Network Traffic and Performance Evaluation*: JOHN WILY & SONS, INC., 2000.

[78] K. S. Trivedi, "Some Important Distributions," in *Probability and Statistics with Reliability, Queuing and Computer Science Applications* 2ed: JOHN WILEY & SONS, INC., 2002, pp. 144-146.

[79] M. E. Crovella and A. Bestavros, "Self-similarity in World Wide Web traffic: evidence and possible causes," *IEEE/ACM Trans. Netw.,* vol. 5, pp. 835-846, 1997.

[80] B. Mandelbrot, "Long-Run Linearity, Locally Gaussian Process, H-Spectra and Infinite Variances," *International Economic Review,* vol. 10, pp. 82-111, 1969.

[81] Cuddlyable3, "Koch snowflake," in *en:Image:Kochsim.gif* vol. 200 x 100 pixels Kochsim.gif, Ed., ed: Wikipedia.

[82] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson, "Self-similarity through high-variability: statistical analysis of Ethernet LAN traffic at the source level," *IEEE/ACM Trans. Netw.,* vol. 5, pp. 71-86, 1997.

[83] T. Tuan and K. Park, "Congestion Control for Self-Similar Network Traffic," in *Self-Similar Network Traffic and Performance Evaluation* K. Park and W. Willinger, Eds., ed: Jhon Wiley & Sons, 2000.

[84] K. S. Trivedi, "Moments and Transforms of Some Distributions," in *Probability and Statistics with Reliability, Queuing and Computer Science Applications* ed: JHON WILEY & SONS, INC., 2002, p. 227.

[85] R. H. Fares and M. E. Woodward, "The Use of Long Range Dependence for Network Congestion Prediction," in *2009 First International Conference on Evolving Internet,* Cannes/La Bocca, France 2009, pp. 119-124.

[86] S. Floyd, R. Gummadi, and S. Shenker. 2000, *Recommendation on using the "gentle_" variant of RED*. Available: http://www.icir.org/floyd/red/gentle.html

[87] S. Athuraliya, V. H. Li, S. H. Low, and Q. Yin., "REM: Active Queue Management," *IEEE Network*, vol. 15, 2001.

[88] M. Welzl, *Network congestion control : managing Internet traffic*: WileyBlackwell 2005.

[89] T. Bonald, M. May, and J. Bolot, "Analytic evaluation of RED performance," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 2000, pp. 1415-1424 vol.3.

[90] P. Hsiao, H. T. Kung, and K. Tan, "Active delay control for TCP," in *Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE*, San Antonio, TX, USA, 2001, pp. 1626-1631

[91] H. Liu and M. El Zarki, "Delay and synchronization control middleware to support real-timemultimedia services over wireless PCS networks," *IEEE Journal on Selected Areas in Communications*, vol. 17, pp. 1660-1672, Sep. 1999.

[92] Z. Wei, T. Liansheng, and P. Gang, "Dynamic queue level control of TCP/RED systems in AQM routers," *Comput. Electr. Eng.*, vol. 35, pp. 59-70, 2009.

[93] W. Fischer and K. Meier-Hellstern, "The Markov-modulated Poisson process (MMPP) cookbook," *Perform. Eval.*, vol. 18, pp. 149-171, 1993.

[94] R. O. Onvural, *Asynchronous Transfer Mode Networks: Performance Issues*, 2nd ed.: Artech House, 1995.

[95] A. Adas, "Traffic models in broadband networks," *Communications Magazine, IEEE,* vol. 35, pp. 82-89, Jul 1997.

[96] L. Guan, M. E. Woodward, and I. U. Awan, "Control of queueing delay in a buffer with time-varying arrival rate," *J. Comput. Syst. Sci.,* vol. 72, pp. 1238-1248, 2006.

[97] J. D. C. Little, "A Proof of the Queueing Formula L = λ W," *Operations Research*, vol. 9, pp. 383-387, 1961.

[98] C. F. Gerald and P. O. Wheatley, *Applied Numerical Analysis,* 6th ed.: Addison Wesley Publishing Company, 1999.