ACTIVE QUEUE MANAGEMENT FOR ASSURED FORWARDING TRAFFIC IN DIFFERENTIATED SERVICES NETWORK

NG ENG SEONG

FACULTY OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2004

ACTIVE QUEUE MANAGEMENT FOR ASSURED FORWARDING TRAFFIC IN DIFFERENTIATED SERVICES NETWORK

NG ENG SEONG

THESIS SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE

FACULTY OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2004

Abstract

Congestion control such as RED is needed to maintain network performance level and to allow efficient usage of the network resources. Generally routers use RED to provide better services than using the traditional tail drop mechanism. In brief, RED work by dropping packets probabilistically when the queue start to pile up. Although RED is capable in performing congestion control, enhancement can be done to RED. Current RED is static and the Internet environment has become more demanding and cannot be met with static RED alone. This dilemma is caused by the fact that network has evolved into a very dynamic state and an idea of developing a more dynamic DiffServ with RED has lead towards the creation of the proposed Active Queue Management for DiffServ Traffic (FuzAQM). FuzAQM works by monitoring the condition of the network in real time and is able change its RED parameters based on the congestion level based on fuzzy logic. The main aim of FuzAQM is to improve network throughput and provide means whereby packets can be treated fairly. This is especially true for lower priority packets where these packets are always choke-out by packets of higher priority. In order to demonstrate the feasibility of the idea, FuzAQM is implemented as an extension to the Ns2 network simulator. Simulations results show that the proposed Active Queue Management for DiffServ Traffic improves the total throughput by 0.79% to 6.46% depending on the traffic load. The proposed mechanism is also able to ensure fair treatment of lower priority packets while maintaining relatively high percentage (87%) of throughput for higher priority packets in the network.

Acknowledgements

I would like to express my utmost gratitude to my supervisor Dr. Phang Keat Keong who has provided me with unlimited support, motivation, time and guidance throughout this research.

Special thanks also goes to Mr Ling Teck Chaw, Mr Ang Tan Fong and Mr Liew Chee Sun for the advice and suggestions to improve this research.

Not the least to Julio Orozco from Universitaire de Beaulieu, France; Timo Viipuri from Helsinki University of Technology, Finland; and Alexander Sayenko from University of Jyvaskyla, Finland who have been generous with their time, expertise and knowledge on ns2 DiffServ.

Finally, special thanks to Soo Wooi King, Chan Siew Yin, Lim Gek Pei, Ngo Foong Kiew, Ng Wai Keat and Vathsala, as my colleagues and peers in the Network Research Lab for the support and help during the entire research. The supports and motivations given are deeply appreciated.

Contents

List of Figures

List of Tables

Abbreviations

ADSL	Asymmetric Digital Subscriber Line
AF	Assured Forwarding
ATM	Asynchronous Transfer Mode
BA Classifier	Behavior Aggregate Classifier
BE	Best Effort
CBWFQ	Class-Based Weighted Fair Queuing
CP	Codepoint
DiffServ	Differentiated Services
Drop prob.	Drop Probability
DS domain	DiffServ Domain
DSCP	DiffServ Codepoint
ECN	Explicit Congestion Notification
edrops	Packets drop by RED early dropping
EF	Expedited Forwarding
FTP	File Transfer Protocol
GUI	graphical user interface
HTTP	Hyper Text Transfer Protocol
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISA	Integrated Services Architecture
ldrops	Packets that are dropped due to link overflow
Max th.	Maximum Threshold
MF Classifier	Multi-field Classifier
Min th.	Minimum Threshold
MRED	Multiple RED
ns2	Network Simulator 2
PHB	Per Hop Behavior
PQ	Priority Queuing
QoS	Quality of Service
RED	Random Early Detection
RFC	Request for Comment
RIO-C	RED with In/Out and Coupled average queues
RIO-D	RED with In/Out and Decoupled average queues
RR	Round Robin
RSVP	Resource Reservation Protocol
SLA	Service Level Agreement
TCP	Transmission Control Protocol
TOS	Type of Service
TotPkts	Packets received
TSW2CM	Time Sliding Window with Two Color Marking
TxPkts	Packets sent
VoIP	Voice over IP
WRED	Weighted RED
WWW	World Wide Web

Chapter 1: Introduction

1.1 Introduction

The Internet world today has evolved into a very high demanding state, which requires high bandwidth and quality of service. The increased demand for the use of time sensitive data in the Internet has set off the need for designing new Internet architecture and total utilization of the Internet resources. These include various architecture and protocol such as more advanced congestion control algorithm based on TCP and Quality of service (QoS) protocols. Such research and demand for new and more advance protocols and architectures has initiated the proposition of Internet services such as Integrated Services Architecture (ISA), Random Early Detection (RED) and Differentiated Services (DiffServ).

The purpose of proposing the use of DiffServ is to deliver an aggregated quality of service in IP networks. Further advancement has been witnessed for DiffServ framework with the integration of Active Queue Management. The intention of the Active queue management, such as RED, in the DiffServ architecture is to preferentially drop packets before any serious congestion begin (Floyd93).

1.2 Thesis Motivation

Although DiffServ combined with RED is an ideal way of service differentiation, but it is still not up to demand, as the recent Internet traffic has evolved into a very dynamic form. Traffic pumped into the network by end users varies a lot from one another. Some end users are still using the traditional dial up method while other users are using Asynchronous Transfer Mode, ADSL or T1 services, which can carry more traffic at higher speed. By using different services, the state in the network has become more volatile. The amount of data exists in the network link at a given moment, varies in a large degree. Thus the existing DiffServ with RED is not capable enough to cater for this ever changing network condition.

Hence the aim of this thesis is to produce a dynamic queue management for DiffServ by incorporating the fuzzy logic. By using fuzzy logic, now the queue management process is more dynamic and responsive towards changes in the network traffic and could be known as FuzAQM Model.

1.3 Thesis Objectives

This thesis undertakes a thorough study of IP QoS models, which stresses on creating a simulation environment for testing and evaluating dynamic queue management model in DiffServ Domain. The objectives of the thesis are summarized as follows:

1. To create a dynamic queue management model for DiffServ network using fuzzy logic
2. To study the differences between different RED parameters settings
3. To explore and study IP QoS models and focus mainly on DiffServ
4. To create a simulation environment for DiffServ
5. To evaluate the effectiveness of the FuzAQM in the created DiffServ simulation environment
6. To study the behaviors and response of the FuzAQM under different parameters and traffic load

1.4 Thesis Scopes

The implementation of this project would be limited to DiffServ AF PHB. This is so as EF PHB is considered as premium service that will get all the special treatment and the BE PHB is considered as best-effort traffic, which will be discarded first, once congestion avoidance takes place or congestion occurs. Furthermore, AF classes have 4 different types of services, which are the AF1, AF2, AF3 and AF4. Between these 4 classes there are different preferential and treatment among them. AF1 is considered more superior than AF2 and so on. Within the same AF class, there will be further

differential among the packets. This difference will be governed by the usage of drop precedence.

1.5 Thesis Significances

Below are the significances of the project.

1 To improve total throughput of traffic

By implementing fuzzy logic into queue management, the total number of successful packets delivered will be boosted.

2 To improve throughput of packets with higher drop precedence

As for packets with the same AF PHB, attention will always be given to packets with lower drop precedence so that it will be delivered within an acceptable throughput percentage. Before the incorporation of fuzzy logic, the traffic conditioner will have a profile, which will maximize the throughput of lower drop precedence that will starve packets with higher drop precedence. With the implementation of fuzzy logic, packets with higher drop precedence will not be starved and at the same time still maintaining an acceptable throughput percentage of the packets with lower drop precedence.

3 To turn the queue management model into a responsive queue management

The queue management model in the DiffServ environment will work according to a set of RED thresholds. As these parameters take effects, it will be set into the queue management. This will make the queue management rigid and unresponsive to network condition changes. But with the help of fuzzy logic, the queue management will now be more responsive to network condition and actively changes the RED thresholds to suits the current condition of the network.

1.6 Thesis Organization

This thesis is organized into 7 chapters as follows:

Chapter 1 Introduction

Chapter 2 Literature Review

Chapter 3 Network Simulator and Simulation

Chapter 4 Proposed Active Queue Management Model

Chapter 5 Coding and Implementation

Chapter 6 Data and Result analysis

Chapter 7 Conclusion and Future Work

Chapter 2: Literature Review

2.1 Quality of Service

Quality of Service (QoS) refers to the network's ability to improve its service for selected network traffic using various technologies. Different type of technologies has been developed for this purpose such as Asynchronous Transfer Mode (ATM), Ethernet and Frame Relay. When trying to provide QoS in the network, some parameters need to be attended. These parameters are keys in providing better or priority services to the traffic and these parameters are as follow:

1. Jitter
2. Delay
3. Packet loss
4. Bandwidth

The major purpose here is to provide a service with a dedicated bandwidth, acceptable jitter and delay; and lesser packet drop. Besides than bandwidth, jitter, delay and loss, there is also an important feature which needs to be addressed in providing QoS which is while providing priority for important traffic, the less important traffic should not be sacrifice or starved.

2.1.1 Concepts of QoS

The primary focus of QoS is to provide a better or special service to those packets which need premium service. This can be done by assigning higher priority to the important packets or by limiting resources for less important traffic. Furthermore, less important traffic may be assigned a value of lower priority from the important ones. As for DiffServ, the concept that will be applied to carry out QoS ability is to have congestion management tools by raising the priority of an aggregated flow, which has same requirements, providing different queues for different priority and servicing those queues differently.

Prioritizing could be done by raising the priority settings for important packets while lowering those unimportant packets, and dropping those lower priority packets prior to higher priority packets. In addition, by manipulating the shaping and queuing mechanism, throughput of those high priority packets could be tuned to surplus those lower priority packets.

QoS tools could help to alleviate most congestion problems, but in reality it couldn't perform efficiently because most of the time, the traffic pumped in is way too much than the bandwidth could support (Cisco00). In this case, QoS tools could only be merely a small relief to the huge problem and the QoS tools couldn't help in providing better QoS for the network.

2.1.2 Architecture of QoS

Three fundamental part have been introduced for QoS implementation (Cisco00), which are:

1. QoS identification and marking
2. QoS within single network element
3. QoS policy, management and accounting functions

2.1.2.1 QoS Identification and Marking

In order to provide different treatment for different packets, firstly the packets must be identified or marked or identified plus marking accordingly. When a packet is identified but not mark, the packets are merely classified by their Per Hop Behavior (PHB) (Cisco00). If the packets are to be marked, the IP precedence field will be modify to reflect the priority of the packets.

2.1.2.2 QoS within Single Network Element

There are several ways to provide QoS in a network. To provide QoS within a single network, queue management, congestion management, link efficiency and shaping or policing is used (Cisco00).

Congestion management is used to provide QoS for those higher priority packets to be sent across the network. This could be done by using different queuing discipline such as

class-based weighted fair queuing (CBWFQ), weighted fair queuing, priority queuing (PQ) or round robin (RR).

Queue management is used to react to initiation of congestion by dropping the packets in the queue. Traditionally, drop tail is implemented where incoming packets are dropped when the queue is full. This will lead to a phenomenon called global synchronization. In order to prevent this phenomenon, there are two things need to be done (Cisco00):

1. Making sure that the queue does not fill up to the maximum limit, so that incoming higher priority packets could have the chance to get into the queue
2. Applying a way to drop lower priority packets before higher priority packets dropped.

Link efficiency refers to ability of the link to be utilized to the fullest whilst avoiding too much delay or segmentation of the packets in the queue. Shaping is used to avoid traffic overflow. The traffic entering the network should be paced so that the network could manage it accordingly to its ability.

2.1.2.3 QoS policy, management and accounting functions

This part of the QoS addresses the importance to evaluate and set the QoS policies and goals (Cisco00). This could be carried out using a probe to monitor the condition of the traffic and initiate proper QoS techniques accordingly. Finally a procedure is needed to evaluate the result of the action taken by getting the feedback from the targeted application to determine whether the QoS goals are met or not.

2.2 Differentiated Services

The Integrated Services Architecture (ISA) and Resource Reservation Protocol (RSVP) (RFC2205) are meant to support QoS in the Internet or private networks. Although both have been useful tools to provided QoS assurance, in reality they are complex to deploy (Stallings02). Another issue, which has been raised by these, two tools, is scalability. Internet nowadays is handling huge amount of traffic, and with the use of ISA and RSVP, it is quite impossible to trace the entire controlling signal required to coordinate integrated QoS offerings and the state information required at routers. This is due to the characteristic of ISA, which treat every single connection distinctively and information need to be stored for each unique flow.

As the network load worsens by the use of variety of applications, immediate attention is required to offer differing levels of QoS to different traffic flow besides ISA. The differentiated services (DiffServ) architecture, based on the Request for Comments: 2475 (RFC 2475) An Architecture for Differentiated Services, is intended to provide uncomplicated, yet easy to implement with low overhead tool to support a variety of network services that are differentiated on the basis performance.

DiffServ efficiency and ease of deployment owe lots to several key characteristics (Stallings02) as listed below:

1. Differing QoS treatment on each IP packets are labeled using the existing IPv4 Type of Service (TOS) field or IPv6 Traffic Class Field. These are all existing field in their respective IP header, so no change is required.
2. Before the customer could really use the DiffServ, a Service Level Agreement (SLA) has to be established between the service provider and the customer. With this feature, there would be no need incorporate DiffServ mechanism in applications. Thus, existing application need not to be modified.
3. DiffServ provides a built-in mechanism where all traffic with the same DiffServ field are aggregated and treated with the same services. This is a vital characteristic, which contributes towards scalability of DiffServ in larger networks and traffic loads.
4. Individual routers in DiffServ domain implement DiffServ by queuing and forwarding packets based on the DiffServ field. Hence routers process each packet individually and need not save state information on packet flows.

2.2.1 Architecture for Differentiated Services

The differentiated services architecture is based on a simple model where traffic entering a network is classified and possibly conditioned at the boundaries of the network, and assigned to different behavior aggregates (RFC2475). Using a specific field in the IP header, Type of Service (TOS) field, behavior of the aggregated traffic could be identified as the DiffServ Codepoint (DSCP) field. Each aggregated behavior or known as per hop behavior (PHB) is being assigned with a unique DSCP value. Within the core of the network, packets are forwarded according to the per-hop behavior associated with the DS codepoint (RFC2475).

2.2.2 DiffServ Codepoint (DSCP)

Each packet that needs to use the DiffServ service will need to be labeled accordingly to its service request by using the DiffServ field. The DiffServ field is located at the TOS field in the IP header. There are two headers in two different versions of IP packets. In IPv4 the TOS field is used and for IPv6 the Traffic class field is used. The format of the DiffServ field (according to the RFC 2474) uses the leftmost 6 bits to form a DiffServ Codepoint (DSCP), while the rightmost 2 bits are left unused at the moment.

With a 6 bit codepoint, there are logically 64 different classes of traffic could be define. But these 64 codepoints are further reallocated into three pools of codepoint as follows (Stallings02):

1. Codepoints of the form XXXXX0, where X is either 1 or 0, are reserved for assignment as standards
2. Codepoints of the form XXXX11, where X is either 1 or 0, are reserved for experimental or local use
3. Codepoints of the form XXXX01, where X is either 1 or 0, are reserved for experimental or local use but might be allocated for future standards action as needed

From the 3 different pools, the first pool has several assignments made. Codepoint 000000 is assigned as default packet class, which is also known as the traditional best effort forwarding behavior (RFC2474). Such packets are forwarded only if they have been received when the link is available. If there is a situation where other higher priority packet exists, this higher priority packet is given preference over those best effort packets.

Codepoints from XXX000 are reserved for backward compatibility with the IPv4 precedence service (RFC2474). The XXX000 DSCP form will be serviced at a minimum equivalent to that IPv4 precedence functionality (Stallings02).

2.2.3 Per Hop Behavior (PHB)

A per-hop behavior (PHB) is a description of the externally observable forwarding behavior of a DS node applied to a particular DS behavior aggregate (RFC2475). This behavior could be seen or obvious if there are few aggregated behavior exist in the same link. For example, if there is only one behavior aggregate in the link, the forwarding activities will depend mainly on the load of the link. Useful behavioral distinctions are mainly observed when multiple behavior aggregates compete for buffer and bandwidth resources on a node (RFC2475). PHB is used by a node to allocate resources depending on the aggregate behavior, and it is on top of this basic hop-by-hop resource allocation mechanism that useful differentiated services may be constructed.

PHBs are implemented in nodes using buffer management and packet scheduling mechanisms. PHBs are defined in terms of behavior characteristics related to service provisioning policies, and not in terms of specific implementation mechanisms. Commonly, a mixture of implementation mechanisms may be fit for implementing a particular PHB group. Furthermore, it is likely that more than one PHB group may be implemented on a node and utilized within a domain (RFC2475). PHB is selected at a node by a mapping of the DS codepoint in a received packet (RFC2474).

To assure that PHB could be implemented clearly and standardized, two RFC have been issued as follows:

1. RFC 2597 Assured Forwarding PHB
2. RFC2598 Expedited Forwarding PHB

2.3.1 Expedited Forwarding PHB

The EF PHB can be used to build a service through DS domains, which will have 5 characteristics as follows:

1. low loss
2. low latency
3. low jitter
4. assured bandwidth
5. end-to-end service

The end users will have the concept of using point-to-point connection or a "virtual leased line". This service has also been described as Premium service (Nichols97).

Loss, latency and jitter are experience when packets travel through network and being queued up for forwarding purpose in each router that they visited. To provide a service, which will not cause loss, jitter and latency will be almost impossible to construct. For this to be realized, packets must be transmitted through indefinite queue size, which will guarantee no packet loss due to drop tail effect, transmitted through an empty queue all the time to avoid latency and the link must be fast enough to transmit the packets t avoid jitter.

However providing low loss, latency and jitter for some traffic aggregate might be more realistic by ensuring that the aggregate sees no or very small queues. It is when traffic

arrives at a router with an arrival speed exceeding its departure speed, queue will be formed. Thus a service that ensures no queues for some aggregate is equivalent to bounding rates such that, at every transit node, the aggregate's maximum arrival rate is less than that aggregate's minimum departure rate (RFC2598). The EF PHB is not a mandatory part of the Differentiated Services architecture (RFC2598).

The creation of the service consists of two parts, which are:

1. Configuring nodes so that the aggregate has a well-defined minimum departure rate.
2. Conditioning the aggregate so that its arrival rate at any node is always less than that node's configured minimum departure rate.

2.3.2 Assured Forwarding PHB

Assured Forwarding (AF) PHB group is a means for a provider DS domain to offer different levels of forwarding assurances for IP packets received from a customer DS domain (RFC2597). AF allows many dropping implementations such as RED (Clark98). There are four AF classes being defined, where in each AF class certain amount of forwarding resources are allocated. The DS domain provider will set the AF class for each IP packet, which would like to use the differentiated services, or the customer himself could set it.

According to the RFC2597, AF PHB group provides forwarding of IP packets in N independent AF classes. Within each AF class, an IP packet is assigned one of M different levels of drop precedence. An IP packet that belongs to an AF class i and has drop precedence j is marked with the AF codepoint AFij, where

and $1 \leq i \leq N$ $1 \leq j \leq M$

For example, there are 2(N=2) independent AF classes which is 1 and 2 ($1 \leq i \leq 2$) and within each AF class, each packet is assigned to 2 (M=2) drop precedence ($1 \leq j \leq 2$). This will create four different preferential treatments for the IP packets, which are

AF1 1 AF1 2 AF2 1 AF2 2

Within each AF class, IP packets are marked with one of three potential drop precedence values. In case of congestion, the drop precedence of a packet determines the relative significance of the packet within the AF class. A congested DS node tries to protect packets with a lower drop precedence value from being lost by preferably discarding packets with a higher drop precedence value (RFC2597).

In the implementation level, the degree of forwarding assurance of an IP packet depends on (RFC2597) as follows:

1. how much forwarding resources has been allocated to the AF class that the packet belongs to
2. what is the current load of the AF class, and, in case of congestion within the class
3. what is the drop precedence of the packet

For example, if traffic conditioning actions at the entrance node of the provider, DS domain make sure that an AF class in the DS nodes is only reasonably loaded by packets with the lowest drop precedence value. AF class therefore can offer a high level of forwarding assurance for packets that are within the subscribed profile (in profile) while assuring a level of forwarding for the lower drop precedence IP.

According to RFC2597, within each AF class, a DS node MUST accept all three drop precedence codepoints and they MUST yield at least two different levels of loss probability. For this project purpose, 2 different levels of drop probability for one class of AF traffic is been used.

An AF implementation MUST attempt to minimize long-term congestion within each class, while allowing short-term congestion resulting from bursts (RFC2597). This requires an active queue management algorithm such as Random Early Drop (RED) (RFC2598).

Recommended codepoints for the four general use AF classes are given below. However, these codepoints do not overlap with any other general use PHB groups.

The table below summarizes the recommended AF codepoint values (RFC2597).

Table 2.1 Recommended AF codepoint values

	Class 1	**Class 2**	**Class 3**	**Class 4**
Low Drop Precedence	001010	010010	011010	100010
Medium Drop Precedence	001100	010100	011100	100100
High Drop Precedence	001110	010110	011110	100110

Drop precedence corresponds to value j and Class corresponds to value i for AF codepoint AFij.

2.2.4 DiffServ Domain

A DS domain is a contiguous set of DS nodes which operate with a common service provisioning policy and set of PHB groups implemented on each node (RFC2475). A DS domain has a well-defined boundary consisting of two characteristics, which are:

1. DS boundary nodes
2. Nodes within the DS domain

These two different types of nodes have their own functions. The boundary node or border router is responsible to classify and possibly condition ingress traffic into the DS domain. This process is vital to ensure that all IP packets are appropriately marked with a Per Hop Behavior supported within the DS domain.

The nodes within the DS Domain have lesser function compared to the border nodes. Their function is to select the forwarding behavior for packets according on their DS codepoint, mapping that value to one of the supported PHBs using either the recommended codepoint to PHB mapping or a locally customized mapping (RFC2474).

2.2.5 Classifiers

Classifiers select packets in a traffic stream based on the content of some portion of the packet header (RFC2475). Two types of classifiers are being defined, which are:

1. Behavior Aggregate Classifier (BA)
2. Multi-field Classifier (MF)

There are differences in the two types of classifiers. The difference could be noticed by the way the classifiers work in classifying the packets. BA Classifier classifies packets based on the DSCP only while MF classifier selects packets based on the value of a combination of one or more header fields. Some of the combinations are as follows:

1. source address
2. destination address
3. DS field
4. protocol ID
5. source port
6. destination port numbers
7. other information such as incoming interface

Classifiers are used to "steer" packets matching some specified rule to an element of a traffic conditioner for further processing (RFC2475). Classifiers must be configured by some management procedure in accordance with the appropriate TCA (RFC2475).

2.2.6 Traffic Classification and Conditioning

Differentiated services are extended across a DS domain boundary by establishing a Service Level Agreement (SLA) between an upstream network and a downstream DS domain (RFC2475). The SLA might contain rules for specifying packet classification and re-marking. Furthermore, it might also specify traffic profiles and actions to traffic streams, which are in or out of profile. The Traffic Conditioning Agreement (TCA) between the domains is derived (explicitly or implicitly) from this SLA (RFC2475).

Traffic conditioning performs functions such as metering, shaping, policing and/or re-marking. These functions are vital to ensure that the traffic entering the DS domain conforms to the rules specified in the TCA. The extent of traffic conditioning required is dependent on the specifics of the service offering, and may range from simple codepoint re-marking to complex policing and shaping operations (RFC2475).

2.2.7 Traffic Conditioners

A traffic conditioner may contain the following elements although it is not necessary to contain all components as follows:

1. Meter
2. Marker
3. Shaper
4. Dropper

A traffic stream is selected by a classifier, which steers the packets to a logical instance of a traffic conditioner. A meter is used to meter the traffic stream against a traffic profile. The state of the meter with respect to a particular packet may be used to affect a marking, dropping, or shaping action (RFC2475).

The diagram below shows an example of how the traffic conditioner components being organized.

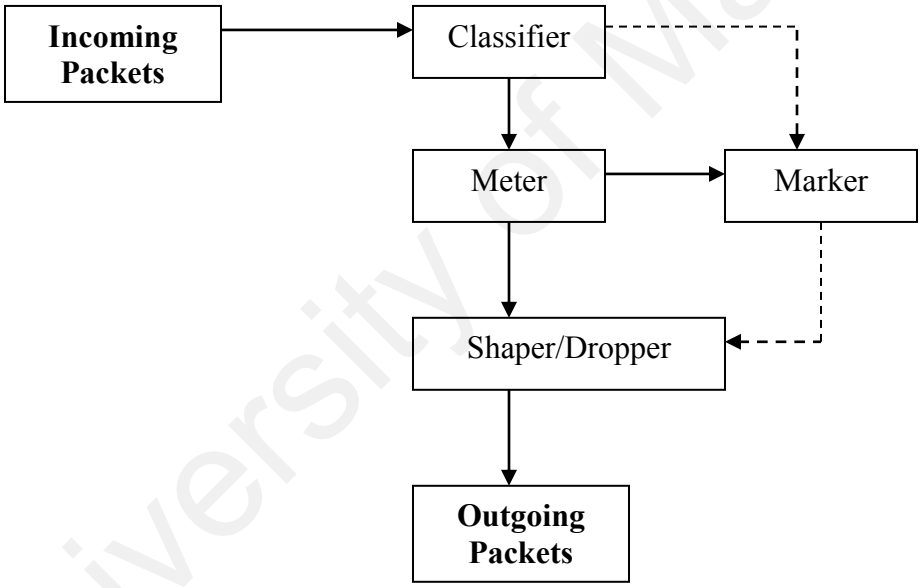


Figure 2.1 Traffic conditioner components

2.2.7.1 Meter

All the incoming packets will be measured by the marker to determine their conformance of a packet with the agreed profile. With the meter, each packet will be determined whether it has exceeded or within the service level guaranteed for the class (Stallings02). Further action will be triggered accordingly by other part of the traffic conditioner if the packets are non-conforming to the agreed profile.

2.2.7.2 Marker

Packet markers set the DS field of a packet to a particular codepoint, adding the marked packet to a particular DS behavior aggregate (RFC2475). Usually the marker is configured to mark a packet to one of a set of codepoints used to select a PHB from PHB group, according to the state of a meter. When the marker changes the codepoint in a packet it is said to have "re-marked" the packet (RFC2475).

2.2.7.3 Shaper

Shapers delay some or all of the packets in a traffic stream in order to bring the stream into compliance with a traffic profile (RFC2475). A shaper usually has a finite-size buffer, and packets may be discarded if there is not sufficient buffer space to hold the delayed packets (RFC2475).

2.2.7.4 Dropper

Droppers discard some or all of the packets in a traffic stream in order to bring the stream into compliance with a traffic profile (RFC2475). This process is known as "policing" the stream.

2.2.8 Traffic Profiles

A traffic profile specifies the temporal properties of a traffic stream selected by a classifier. It provides rules for determining whether a particular packet is in-profile or out-of-profile (RFC2475).

Based on these different profiles, different conditioning actions may be applied to IP packets. For each in profile and out of profile, different actions may be triggered to be applied to the packet. For example in profile packets may be allowed to enter the DS domain without further conditioning; or, alternatively, their DS codepoint may be changed (RFC2475). The latter happens when the DS codepoint is set to a non-Default value for the first time (RFC2474), or when the packets enter a DS domain that uses a different PHB group or codepoint to PHB mapping policy for this traffic stream.

On the other hand, out of profile packets may be queued until they are in-profile (shaped), discarded (policed), marked with a new codepoint (re-marked), or forwarded unchanged while triggering some accounting procedure (RFC2475). Out-of-profile packets may be mapped to one or more behavior aggregates that are "inferior" in some dimension of forwarding performance to the BA into which in-profile packets are mapped (RFC2475).

2.3 Congestion management

A special care must be taken by the end systems to regulate the flow of their data into the network. This is important to maintain network performance level and to allow efficient usage of the network resources. If care is not taken to maintain the network efficiency, congestion might occur or in worse cases, total collapse of the network will occur (Stallings02). This phenomenon is also known as "Internet meltdown" or technically known as congestion collapse was first observed during the early growth phase of the Internet of the mid 1980s (RFC896). In the year of 1986, Jacobson developed congestion avoidance mechanisms, which is implemented in the TCP (Jacobson88, RFC1122). This mechanism will react to congestion and slow down the source transmission rate to avoid further disaster due to network overload. This event can be triggered with the presence of a packet drop and the signal of packet dropped is received by the source. It is acclaim that these TCP congestion avoidance algorithms have contributed towards the prevention of congestion that leads towards collapse of today's Internet (RFC2309).

However, the ever-growing situation of the Internet traffic, has initiated a quest to find better mechanisms than the one developed by Jacobson. In the traditional way, the queue will be filled up until its maximum limit and then the incoming packets will be dropped. This is called the tail drop. There are also two other queuing disciplines could be applied when queue become full (RFC 2309), which are:

1. Random drop on full
2. Drop front on full

Random drop on full drop randomly selected packet from the router queue when the queue is full and a new packet is arriving. On the other hand, the second discipline drops packet at the front of the queue when the queue is full and there are new packet arriving. But both of this discipline does not solve queue full problem (RFC2309).

Active queue management has been a solution to this full-queues problem. Active queue management works by dropping packets even before the queue is full. Such Mechanism is known as Random Early Detection (RED).

2.3.1 Random Early Detection

Random early detection or RED is an active queue management algorithm. Routers use RED to provide better services than using the traditional tail drop mechanism. In general, RED work by dropping packets probabilistically when the queue start to pile up. The probability value increases as the estimated average queue size grow (RFC2309).

RED algorithm consists of two parts (RFC2309) as follows:

1. Average queue size estimator
2. Dropping packet decision maker

2.3.2 Global Synchronization

RED capability to anticipate congestion has saved us experiencing a phenomenon called global synchronization. This happens for TCP (RFC793) traffic where a drop packet will signal the source to enter slow start phase. The purpose of the TCP source to enter slow start phase is to reduce number of packets sent into the network and to ease congestion. The packet, which has been lost, will be retransmitted by the source and this will burden up the network and imposing delay in the forwarding of packets. This can become serious when several sources enter the slow start phase leaving the network under utilized. After some time, sources will start to deliver more packets as it leaves slow start phase. This will cause a surge in the network and again packets will be dropped and the slow start's phase reignite.

2.3.3 RED algorithm

RED general algorithm for each packet arrival is shown below (Stallings02)

Average queue size is calculated as **avg** and the algorithm is as shown in figure 2.2:

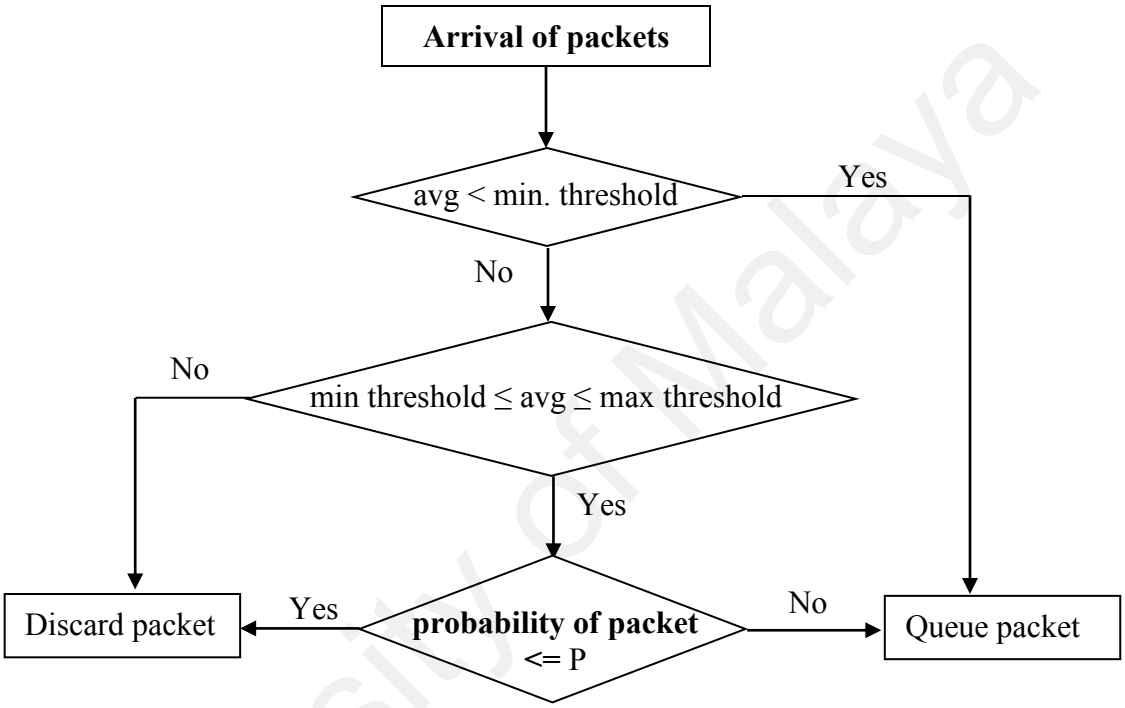


Figure 2.2 RED algorithm

The algorithm starts to work every time when a new packet arrives at the queue. It will calculate the average queue length and to be compare with two RED threshold, the minimum and maximum thresholds. If the average is less than the minimum threshold, the packet will be placed in the queue. If the average is in between the two thresholds value, drop probability will be calculated depending on the average value. The probability will increase as the average value closing up to the maximum threshold value.

Packets with the probability P will be discarded and packets with the probability 1-P will be queued. If the average value exceeded the maximum threshold value, the packet will be discarded at once.

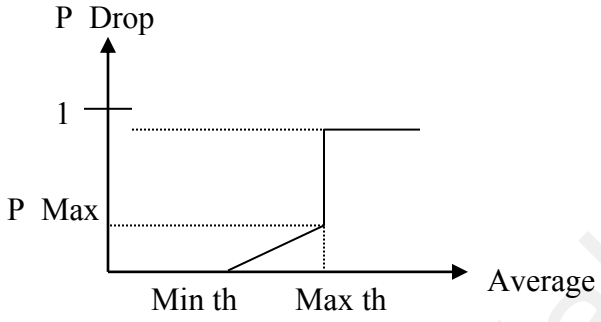


Figure 2.3 RED parameters settings

Figure 2.3 above shows the drop probability P against the Average value. The minimum threshold (Min th) is smaller than the maximum threshold (Max th) value. Packets, which exceed the maximum threshold, will be dropped, indicated by the value of drop probability of 1. Packets below the minimum threshold will be queue, indicated by drop probability of 0. In between the minimum and maximum thresholds, the packet will be dropped with a probability that varies linearly from 0 to P Max.

There are several modifications of the existing RED algorithm and they are called the variants of RED. Some of the variant of RED are Weighted RED (WRED), RED with In/Out and Coupled average queues (RIO-C) and RED with In/Out and Decoupled average queues (RIO-D).

2.4 Fuzzy Logic

The fuzzy logic theory was proposed by Professor L.A. Zadeh in the year 1965. The idea of having fuzzy logic actually begins when there is a thought or demand for a computer to do reasoning using approximation not based on precise input. Computer is very precise and it can't do approximation for a situation unless it is told so and equipped with fuzzy logic. In the real world there are many things, which need heuristic judgment according to situation not by using preplanned or preprogram values. In other words, fuzzy logic is a way to gather human intelligence, knowledge, experience, thinking process and judgment and put into the computer so that it will become intelligent (Asai95).

2.4.1 Fuzzy Set

Fuzzy set is a set, which is used to group objects in it but without a crisp border. The set may have group of things, which precisely belongs to it and group of things, which does not belong to it (Asai95). The word fuzzy itself suggest that it is unclear or blur.

For example, we can use the set hungry to describe a group of hungry people. Logically the group would be divided into two set of values, either they are hungry or not. The two values being mention here are hungry and not hungry. This is a very precise or crisp grouping method. If we used time duration after the last meal to determine whether a person is hungry or not, we can make assumption that 6 hours after the last meal will make a person hungry. So a person, who last consumed a meal 4 hours or 5 hours ago, is considered as not hungry. Perhaps we can also conclude that a person who had just taken

his last meal 5 hours 59 minutes ago is not hungry. Just because of a minute difference, the conclusion is a bit too drastic. Figure 2.4 shows the situation of this example.

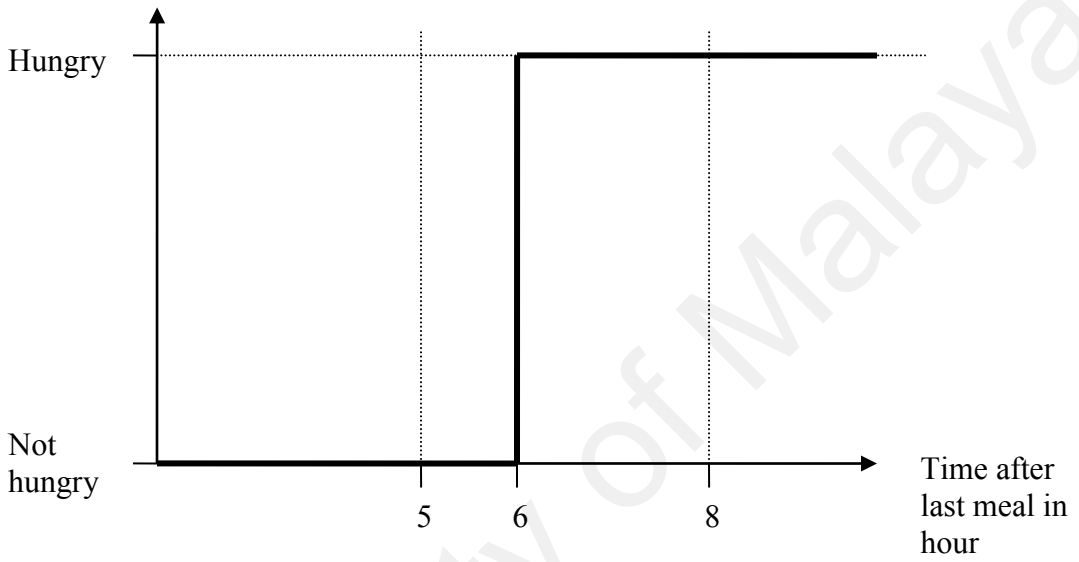


Figure 2.4 Crisp representation of the term of hungry

This drastic changes in decision could be rectified using a degree to represents how hungry is a person. The value between 0 and 1 will be used to measure this degree. By using this degree range, we can now make justification that a person who had his last meal 6 hours ago is consider as hungry with the degree 0.7, 5 hours ago with the degree 0.5 and 8 hours ago is very hungry with the value of 1.0. By using this degree now we can express qualitatively the vague term of hungry quantitatively (Asai95).

2.4.2 Membership Functions

Besides having a degree of 0 to show that the person is not hungry and 1 to show that person is really hungry, we can actually have more than one fuzzy set for how hungry is a person. To further explained this situation, 3 fuzzy sets will be defines for hungry, which are not hungry, relatively hungry and extremely hungry. The figure 2.5 will represent the 3 situations.

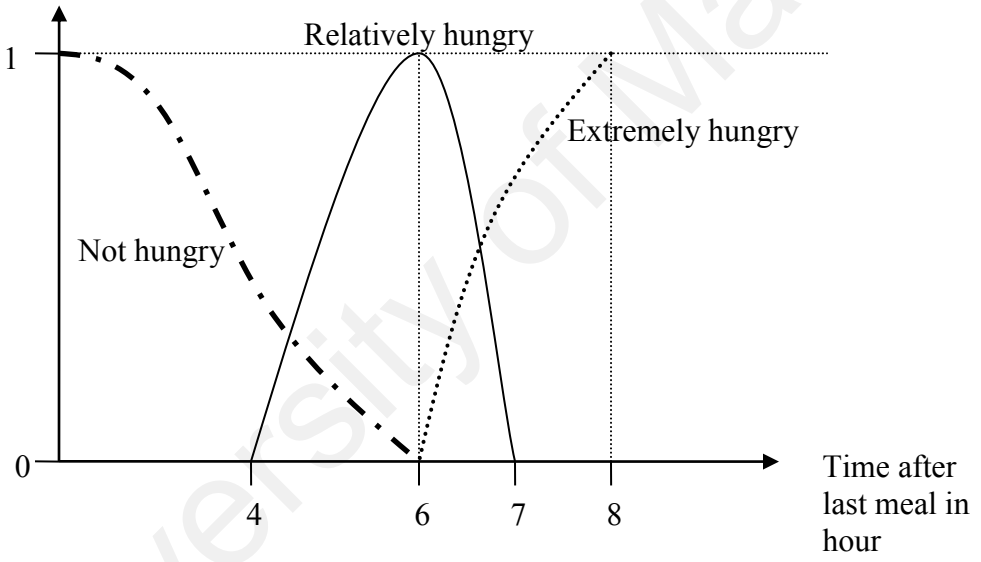


Figure 2.5 Fuzzy representation of the term of hungry

Each curve in the graph is actually called the membership functions. In this example there are only 3 membership functions. It is possible to define more than 3 membership functions. For example, a hungry membership function could be added and the curve can be drawn between 5 hours and 7 hours line.

2.4.2 Fuzzy Operators

Fuzzy sets can be considered as an extension of Boolean logic, so the classical sets of operator could be perform on the fuzzy sets. The classical set operations are as follows:

1. Union
2. Intersection
3. Complement

Union is an operations which selects the maximum of the degrees of memberships from the given sets (Brigette03). For example, given set A and B, with both maximum of the degrees of memberships are 0.9 and 1.0 respectively, the union operator will choose the value 1.0.

Intersection is an operation, which takes the minimum of the degrees of memberships from the given sets (Brigette03). For example, given set G and H, with both minimum degrees of memberships are 0.8 and 0.2 from each set; the intersection operator will choose 0.2 as its output.

Complement is an operator, which takes the degrees of memberships of any given members in a set and minus it with 1 (Brigette03). For example, for a given member in a set, with a degree of membership of 0.45, the complement operator will return the value of 1 – 0.45 which is 0.55.

2.4.3 Fuzzy Reasoning

Reasoning is a process that could be described using normal jargon as interpretation. Fuzzy reasoning or approximately reasoning is a way to get output from several inputs or technically known as fuzzy inference. This method could be explained as the way human does interpretation according for a situation according to several circumstances, with a difference, a computer does it.

2.4.3.1 Fuzzification

Fuzzy inference is a type of logic, which can be, describe with mathematics. The most basic things here are rules, facts and conclusion (Asai95). The inference can be put into a mathematical formulation as shown below.

Rule: IF *x* is A THEN *y* is B

Fact: *x* is A

Conclusion: *y* is B

Or

Rule: IF *you are careless* THEN *you will fall*

Fact: *you are not careless*

Conclusion: *you will not fall*

In order to get inference results, a sets of rules need to be defined. Fuzzification is a method to establish the fact based of the fuzzy system (Brigette03).

2.4.3.2 Inference

The inference process begins when the input is being received by the fuzzy systems. A series of test will be carried of by testing input on every rule or some of the rules. The process will evaluate the if else rules and come out with a result for each if else rule.

2.4.3.3 Composition

Composition is an inference process, which takes into consideration every output from the if else rules, which have been evaluated to, produced a single conclusion.

2.4.3.4 Defuzzification

This is the final process where the single conclusion fuzzy inference value will be mapped into a crisp value. This crisp value is necessary for the main system or the user of the fuzzy system because the system or user can only understand crisp value or act using a crisp value.

Chapter 3: Network Simulator and Simulation

3.1 Network Simulator

The newly developed DiffServ active queue management with fuzzy logic must be tested. There are two main ways to test the model, one is by using real network or the Internet; or we can use a network simulator to simulate the real environment. For this thesis, the testing of the newly developed model will be carried out using simulation. The main reason of using this method is because it is more feasible. If real environment is to be used, it will use up a lot of resources, time and cost which in return is not that practical.

Simulation is important when the case to be studied is very complex with a lot of variables and interacting components, variables relationships are non linear, models to be studied contains a lot of volatile variables or the output must be represented as an animation (Fishwick96).

3.1.1 Computer simulation

Computer simulation is basically about designing a model on a computer and executing it to produce results for analysis. A model can be described as a representation of the real world or situation. It also could mean a theoretical physical representation.

In a computer simulation, there are 3 main parts (Fishwick96), which are:

1. modeling
2. execution
3. analysis

The modeling part concentrates in building a model or few models to be augmented to represent the real situation or problem to be studied. Then, the execution part will need a computer program to steps through time and carry out each event that must be carried out at each tick of the time. At each tick of the time, there might be several variables needed to be updated and initiated or destroyed, depending on the details in the model. The final part, which captures the most attention, is the analysis part. Results generated during or after the simulation has ended served the purpose of understanding the problem to be solved. From this results basically in the form of statistics figure, could be used to make suggestion or feedback some information to improve the model or simulator itself.

A network simulator is actually a computer simulation tool, which could be used to study the network behavior specifically. There are several network simulator tools available such as OPNET, Maisie, PARSEC, INSANE, REAL and ns2.

3.1.2 Advantages and Disadvantages of Computer simulation

Key advantages of computer simulation could be seen when simulation can solve complex situation without incurring high cost, consuming a lot of time or threatening real life and situation. For an example, a simulation for tunneling through a mountain could be carried virtually using computer. The explosion part could be represented in a model so that the outcome of the explosion could be studied without any life harmed and threaten the mountain structure. If there is anything wrong with the simulation due to the use of wrong explosive materials, amount or timing for explosion, they could be reset and done again until a good setting could be compromised. Another few advantages of using simulation are that the environment for the simulation is controllable and it could be animated to help increase understanding of the situation.

There are also disadvantages in a computer simulation. The most crucial part in a computer simulation part is to create a model, which represents the real situation as accurate as possible. Some situation is too complex but the model built to represent it needs to be simplified and maybe some assumptions need to be done. This will cause the simulation to be less accurate. There is also another disadvantage in computer simulation when it needs to simulate situation with unpredictable behavior. For example human behavior is so unpredictable and it is almost too random to be modeled.

3.2 Available Network Simulator

3.2.1 OPNET

OPNET or known as OPtimised Network Engineering Tool Modeler is a well known commercial network simulator which was developed and introduced in the year 1987. OPNET is an object-oriented simulator built on the C/C++ languages (OPNET97). It supports hierarchical network models, object-oriented modeling, finite state machine modeling and wireless, point-to-point and multipoint links.

OPNET has graphical user interface (GUI) support and it is based on a series of hierarchical editors. The organization of these editors is parallel to the structure of the real networks, underlying protocol and equipment (OPNET97). The editors in OPNET are as follows:

1. The Project Editor
2. The Node Editor
3. The Process Editor

OPNET has several advantages such as animation, unlimited sub network nesting, integrated with a debugger, geographical and mobility modeling; and has more than 400 library functions.

Although OPNET is one of the leading network simulator tools available in the market, it is way too expensive to be used.

3.2.2 Maisie

Maisie is a discrete event simulation language, which is developed using C language (Maisie96). The discrete event simulation owes much to its process approach.

Maisie can simulate discrete event simulation model using several different asynchronous parallel simulation protocols (Maisie96). It also has a powerful message receiving constructs that contributes towards natural simulation program and shorter simulation.

Maisie is not equipped with GUI and it is not portable.

3.2.3 PARSEC

PARSEC or known as Parallel Simulation Environment for Complex systems (PARSEC) is a C-based discrete-event simulation language (PARSEC98). Derived from Maisie, PARSEC is now more efficient with several improvements, both in the syntax and the simulation execution environment (PARSEC98).

The key advantage of PARSEC has over other network simulators, is its ability to execute a discrete-event simulation model using several different asynchronous parallel simulation protocols on a variety of parallel architectures. PARSEC is equipped with great message receiving constructs, which result in shorter and more natural simulation programs. Besides than that, it also provides debugging facilities and a front-end for visual specification of the simulation model as well as the runtime output.

A major disadvantage of PARSEC is that in order for a simulation to carry out, it involves the use of the language itself without a GUI. Furthermore, PARSEC is less portable when several platforms are taken into consideration.

3.2.3 INSANE

INSANE or known as Internet Simulated ATM Networking Environment (INSANE) is a network simulator designed to test various IP-over-ATM algorithms with realistic traffic loads derived from empirical traffic measurements and is written using C++ (INSANE96).

INSANE does support a large number of Internet protocol such as IP, TCP and UDP. It also has several applications, which can mimic and simulate real world traffic, which includes telnet, ftp, World Wide Web, real-time audio, and real time video traffic.

INSANE works quite well on distributed computing clusters and it is carried sequentially. It is also capable of handling large simulations.

Although INSANE has several keys advantages, it also has several disadvantages. One of the major disadvantages of INSANE is it does not provide GUI support that facilitate the creation of the environment needed for the simulation. And the second major disadvantage is INSANE is not platform independent. It only operates on UNIX based and the performance output can be only viewed in a text-based form.

3.2.4 REAL

REAL is a network simulator, which has been created for the purpose of studying the dynamic behavior of flow, and congestion control schemes in packet switched data networks (REAL97). REAL has around 30 modules, which emulates the flow control protocols such as TCP and 5 scheduling disciplines such as fair queuing and hierarchical round robin.

REAL provides its source code in C for their users to modify accordingly to their interest. It also has online documentation for distribution and its main strength is its modularity of the codes makes it easy to add new modules.

REAL will run on Digital Unix, SunOS, Solaris, IRIX, BSD4.3, Ultrix, UMIPS systems on VAX, SUN, SPARC, MIPS, Alpha, SGI or DECstation (REAL97). It does not work on Windows based systems. Originally REAL does not come with GUI support but with the newer version it does come with GUI support written in Java.

3.2.5 Ns2

Ns2 or known as Network Simulator 2 is a discrete event network simulator. The idea of ns2 actually derived from the REAL network simulator (NS03). To date ns2 development is supported by DARPA with SAMAN and through NSF with CONSER including ACIRI has witness a series of mass development of the simulator (NS03).

Ns2 is written in C++ language and the scenario interface with the C++ code is written in Tcl scripting or OTcl (for ns2).

The advantage of ns is that it permits simulation with several levels of abstraction, where higher abstraction levels trade off precision for performance. The simulator measurements do not impact the network by adding extra traffic.

The disadvantage of ns2 is that it does not have a GUI for general simulation manipulation and scenario setup.

3.3 Network Simulator Tool

The network simulator used for this thesis is the ns2. The purpose of choosing ns2 is governed by several critical points and the ns2 advantages itself.

Firstly, ns2 is available free online for interested researcher to download and use. Due to financial constraints, ns2 is obviously the best choice for this thesis. Secondly, ns2 is a

well-known simulator, which is widely used in a lot of researches and publications. Furthermore, ns2 has its online forum and mailing list where researchers around the world could communicate and help each other in case of any difficulties or share domain knowledge whenever there is a common area of research interest.

Ns2 is still expanding from time to time where other researchers substantially contribute their code such as the wireless code by UCB Daedelus and CMU Monarch projects and Sun Microsystems. Ns2 is also an open source simulator where the full sets of code could be downloaded and modified accordingly to each researcher needs and interest.

From the ns2 site itself, there are some write up and information that could help a beginner to understand and explore ns2. The site provides link to download the current version of ns2 and also previous version of ns2. It also has documentation for download and several tutorials to help novice user to get started with ns2.

Although ns2 was intentionally developed for platform based on UNIX and Linux, through it development in current years, there are now efforts to port the existing ns2 code over to make it portable and executable in Windows platform.

Chapter 4: Proposed Active Queue Management Model

4.1 The Proposed Active Queue Management Model

Referring to the aim of this thesis and the facts presented in previous chapters, it is obvious that the traditional method of solving traffic problems is no longer effective. Thus there is a need to initiate a more suitable way or a more dynamic method to cater the current trend in network traffic. The proposed active queue management (FuzAQM) is one of the steps taken in answering this challenge.

The proposed model will be using fuzzy logic as a tool to turn the traditional queue management model into a dynamic model. There are two parameters, which will be exploited to realize this model. The two parameters are the RED thresholds and the drop probability. The idea of exploiting these two parameters is because the queue size and the rate of change of queue are related to each other. Furthermore, the rationale here is to create an active queue management so the main concern here is to look into what is happening in the queue rather than other metrics such as queue delay or available bandwidth. Such metrics are useful when the active queue management model expand its perspective into improving end-to-end delay or routing decision.

This proposed FuzAQM model will used the current queue occupancy and the rate of the change in the queue at every moment when a packet arrive at the bottleneck routers as its rules in determining further action. Several researches in networking areas using fuzzy

logic concept have been done and published in IEEE such as (Zhang01), (Chrysostomou03), (Yanfei03) and (Karthik04).

(Zhang01) focuses on developing a new congestion control using fuzzy logic in DiffServ domain. He introduces a drop based congestion control mechanism, which control or drop packets upon its arrival and drop those packets with lower service precedence first before dropping packets with higher service precedence.

(Chrysostomou03) presented an active queue management scheme called Fuzzy Explicit Marking in DiffServ framework. The model uses a fuzzy controller to examine the dynamic network changes and dropped packets or reset packets ECN bit accordingly.

(Yanfei03) defines a congestion index to indicate the degree of network congestion. An intelligent packet dropping mechanism based on fuzzy logic is then being designed which can optimize router performance.

(Karthik04) examines a fuzzy based Connection/Call Admission Control to provide effective congestion control in ATM network. It is proven from the research that fuzzy based Connection/Call Admission Control gives a better control than conventional systems and also maintained the network QoS.

4.2 Fuzzy Logic Implementations

The fuzzy logic codes are inserted in the enqueue() function located in the dsredq.cc file. The code fuzzy_result=fuzzy(q_->length(), prev_20, qlim) will be used to call the fuzzy logic code.

Before the fuzzy() function is called, series of calculation are needed to provide the input for the function. The main concern is to again explicitly specify which direction of traffic to be calculated. This is needed as the enqueue() function is a general function used to queue packets in the physical queue and it is differentiated with the direction of packet or which simplex queue. The action to be taken as whole depends a lot on which part of the simplex link to be used as calculation. Hence there is a necessity to separate the average calculation and separate action on the two different simplex queues.

The differentiation could be witness by the 'if and else' clause that looks as follows:

```
if (src_id == 14 || src_id == 15 || src_id == 16 || src_id == 17 || src_id == 22 || src_id == 23 || src_id == 26 || src_id == 27 || src_id == 30 || src_id == 31 || src_id == 34 || src_id == 35)
{
        ..........
        ..........
} else {
        ..........
        ..........
}
```

The first part will refer to router 6 to router 7 (from left to right) while the else will refer to the traffic traveling from router 7 to router 6 (from right to left). Figure 4.1 below shows the general representation of the above mention directions.

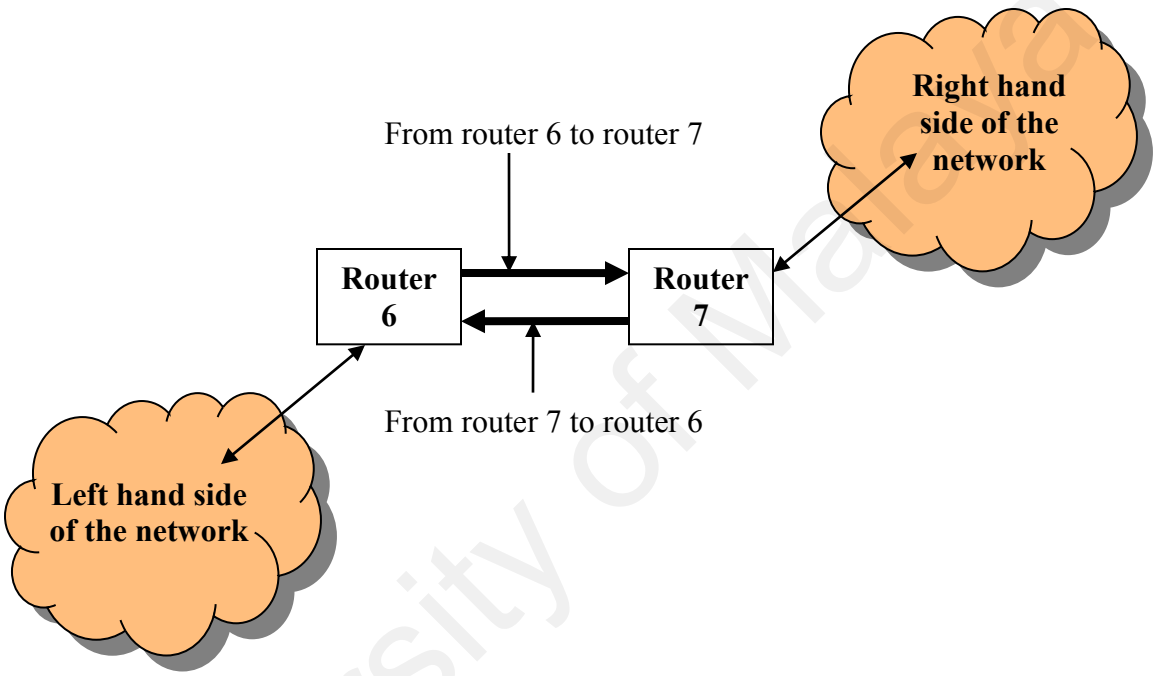


Figure 4.1 General representation of the network topology used

4.2.1 q_->length()

q_->length() is a function which will return the physical queue length for the queue q_. In the constructor, q_ = new PacketQueue() is used create a pointer to a packet queue or the underlying physical queue.

4.2.2 Average Previous 20 Queue Length Calculation

r6prev_20 is a variable of type float. It is used to store the previous 20 physical queue size whenever there is a packet been queued in the physical queue. The value 20 is chosen to makes the average calculation more relevant rather than only 5 or 10 value.

Every time when a packet is been queued, the length of the physical queue length will differ. The time between the last successful packet been queued and the current packet been queued might witness the existing packets in the physical queue being de-queued or emptied. Furthermore, there are a lot of packets being queued and de-queued, therefore the average value of the queue length has to be calculated using previous 20 values maximum. The indicator r6 is referring to the queue, which is responsible to forward packets from router 6 to router 7. The function to calculate the average of the previous 20 queue length is located in the enqueue() function itselt. The calculate average of the previous 20 queue length block of code is as shown below:

```
float r6prev_20;
int counter=1;
r6_travel=new node;
r6_travel=r6_cur;

if (r6_travel!=NULL) {

while((r6_travel->next!=NULL) && (counter!=20))
{

        j=j+r6_travel->num;
        r6_travel=r6_travel->next;
        counter++;
}
r6prev_20=j/counter;
} else {
        r6prev_20=0;
}
```

First a pointer to node is created and name as r6_travel. It is used to travel along the link list to gather up to 20 queue length from the most recent node added. It should be noted that it does not take into account the node that will be added to the link list at the end of the enqueue() function. This is because the previous 20 physical queue length calculations should not taken into the account the one, which will be queued. To further justify this point, it is not relevant to include the present one to calculate the past queue length average.

Counter is used to keep track of the number of nodes been traveled by r6_travel pointer. It will at least have the value of 1. The purpose of having at least a value of 1 is to avoid a division by 0.

The r6_travel counter will be pointing to the node where r6_cur pointer is pointing. The purpose of using if (r6_travel!=NULL) is to make sure that the link list exist. At the very beginning of the simulation, there will be no node in the link list. This is indicated by the head node pointer and current node pointer equal to null. As the enqueue() function being entered the first time, the r6_travel pointer will be pointing to the r6_cur pointer. This means that r6_travel will be pointing to a null node indirectly. For this case, the r6prev_20 variable will be carrying the value 0 instantly with the help of r6prev_20=0 assignment located in the else clause as shown above.

If the above mention case is not true, the while((r6_travel->next!=NULL) && (counter!=20)) loop will be executed. The first argument r6_travel->next!=NULL is to

make sure that the node which r6_travel is pointing is not the first node of the link list. The first node of the link list is named head (where the r6_head pointer is pointing). The second argument (counter!=20) is to make sure if there is more than 20 nodes existed in the link list, the average calculation will only take the most recent 20 nodes queue length value into consideration. If either one of these rules is fulfilled, the while loop will end.

The purpose of having the two arguments "AND" is such as the link list started, it will grow in size. The first time ever, the average value will be 0 because there is no value in the link list to be used. As the link list grows, it will calculate the average using the queue length value of the existing node until most recent 20 nodes the most. For example the (r6_travel->next!=NULL) argument will govern the situation where if there are only 3 nodes in the link list, only 3 nodes will be taken into consideration. If there are only 15 nodes in the link list only 15 nodes will be used. This will be true until the number of node reach 20. The counter variable will be used to calculate the average value. When taken back into consideration the above mention case, if 3 nodes used, the summation of 3 previous physical queue lengths will be divided by the counter value of 3. And for the situation where 15 nodes are used, the summation of 15 nodes physical queues value will be divided with 15.

For traffic traveling from router 7 to router 6, the same type of code and procedure will be used, except this time the variable name will be different (will be identified by the r7 id in front of the variable).

4.3 Fuzzy Sets and Membership Functions

When the average previous 20 queue lengths have been calculated, then the fuzzy logic function could be called. float redQueue::fuzzy(int q_len, float avg, int q_lim) is the fuzzy logic function declaration.

It will receive three arguments from the caller function. The first argument will be used to represent the queue length, the second argument will represent the average queue and the final one will represent the queue limit, each with appropriate variable type.

float q_rate=q_len-avg will be used to calculate the rate of change of queue. The rate of change is actually the difference between the current queue length and the average of previous 20 queue length. The value of this difference will be positive value, zero or negative value. The maximum value of the difference will be 500 and the minimum is -500. This minimum value will occur if the current queue length is zero and the average is 500. The maximum value will occur when the current queue length is 500 and the average queue length is 0.

There are two membership functions in the fuzzy logic code. The first membership function is the current queue length and the second one is the rate of change of queue.

4.3.1 Current Queue Length Fuzzy Set

Current queue length membership function will be represented by an array myLen[2][7] of type float. The dimension of this array is 2 and it has 2 rows and 7 columns as shown below:

{1.0, 0.0, 0.0, q_lim/2, q_lim,0.0,0.0},

{0.0, q_lim/2, q_lim, q_lim, 0.0,0.0,0.0}

Both rows have common column type. The first column represents the shape of the first graph in the membership function. 1.0 will represent a graph with a trapezium shape and 0.0 will represent a graph with a triangle shape.

The following 6 columns will be used to represent 6 different points or areas in the graph. The value shown on each column will represent the x-coordinates in the graph. The corresponding y-coordinates have been set to the value of {X.X, 0, 1, 1, 0, 0, 0} where X.X represent the value which represent the shape of trapezium (which will be 1.0). For the shape of a triangle, the X.X value is 0.0 and the corresponding y-coordinates has been set to the value of {X.X, 0, 1, 0, 0, 0, 0} and the last column value is left unused for this shape.

Column 6th and 7th of the array will always be representing the two extreme ends of the graph. Column 6th will represent negative infinity value or the minimum most x-coordinate value and column 7th will represent positive infinity value or the maximum most x-coordinate value for trapezium graph. For triangle shape graph 5th and 6th column will be used to represent the leftmost and rightmost are of the graph, where the lines stretches towards infinity value of x-axis.

4.3.1.1 Triangle graph

The triangle graph will generally have the shape as shown in Figure 4.2 below.

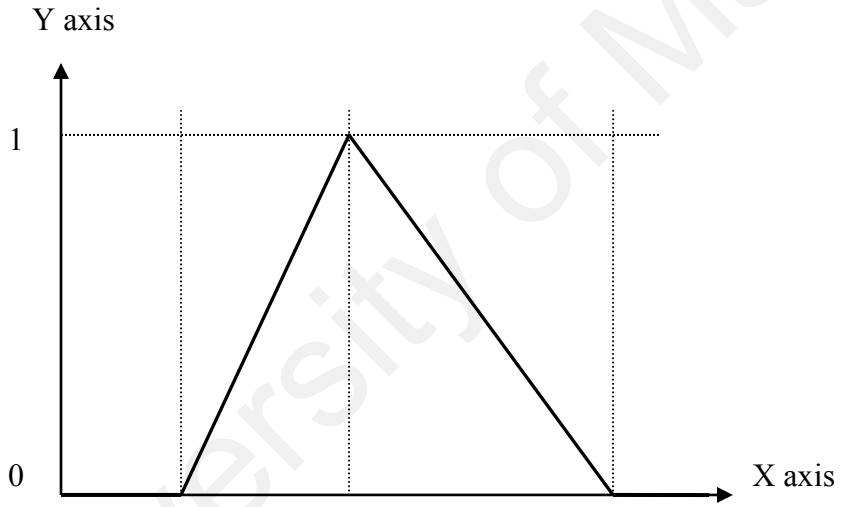


Figure 4.2 Triangle graph

Taking the above given parameters of the triangle graph,

{0.0, q_lim/2, q_lim, q_lim, 0.0, 0.0, 0.0}

where q_lim = 500, the representation of the graph is depicted by Figure 4.3 below.

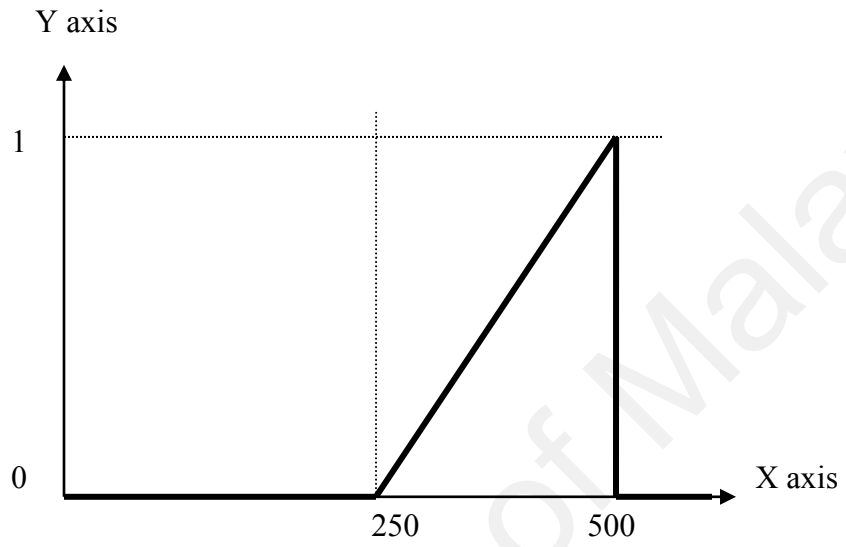


Figure 4.3 Current queue length first array representation using triangle graph

4.3.1.2 Trapezium graph

The trapezium graph will generally have the shape as shown in Figure 4.4 below.

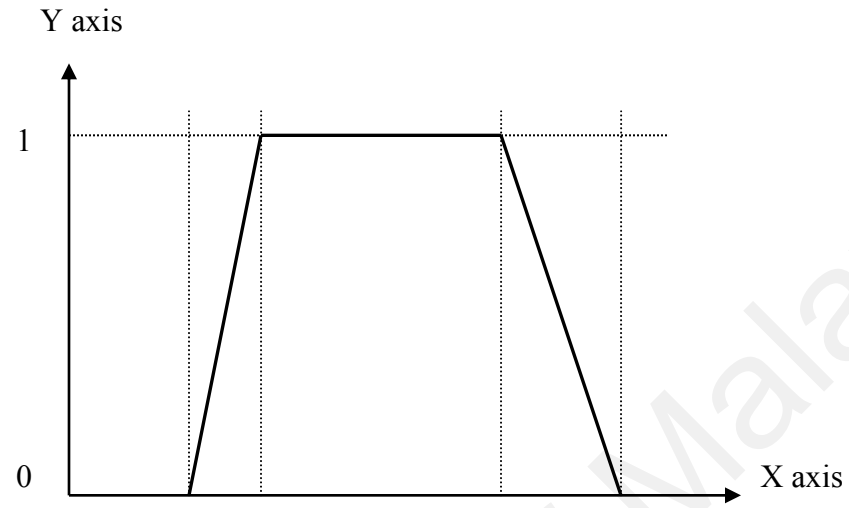


Figure 4.4 Trapezium graph

Taking the above given parameters of the trapezium graph,

{1.0, 0.0, 0.0, q_lim/2, q_lim,0.0,0.0},

where q_lim = 500, the representation of the graph is as shown in Figure 4.6 below:

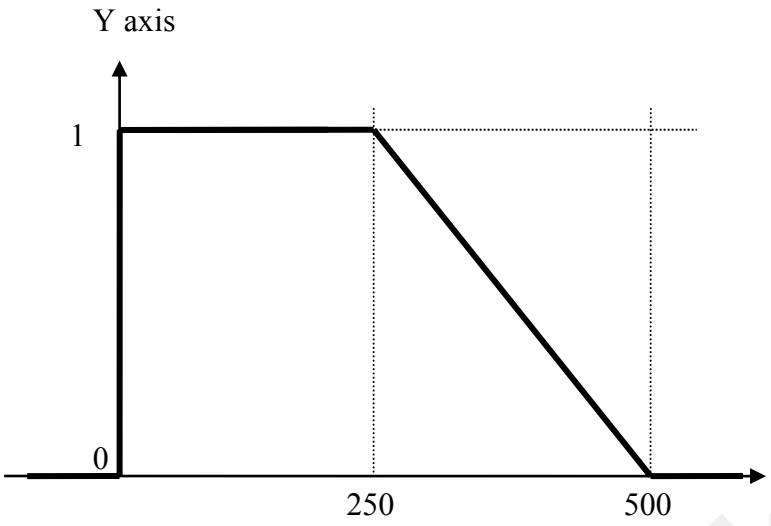


Figure 4.5 Current queue length second array representation using trapezium graph

4.3.2 Current Queue Length Membership Function Graph

To get the idea of the whole current queue length membership function graph, the two existing graph will be merged. The merged graph is as shown in Figure 4.6 below.

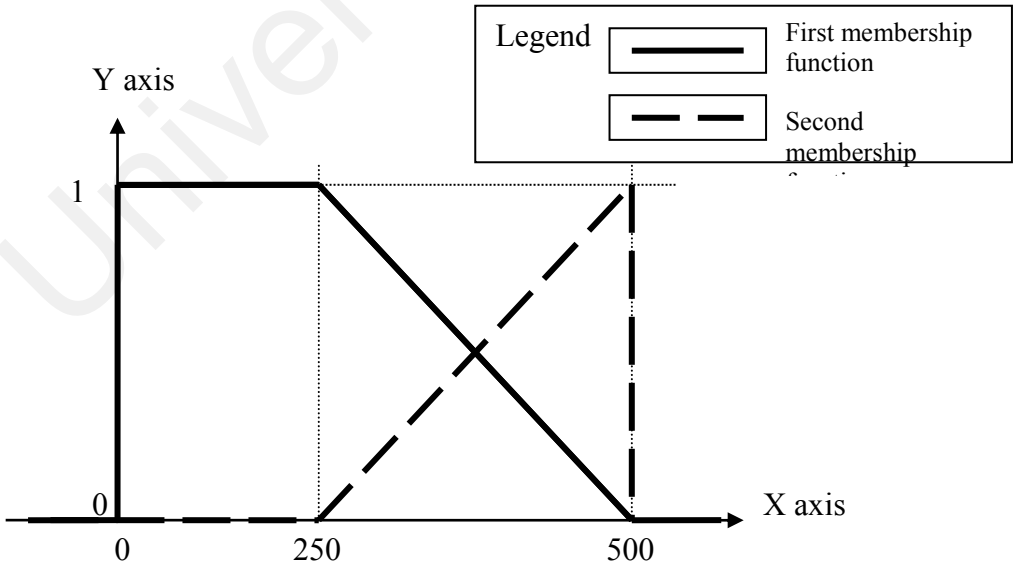


Figure 4.6 Current queue length membership function graph

4.3.2.1 Current Queue Length Rules

The current queue length function will be governed by a set of rules. The queue length will be divided into two zones, 'full' and 'not full'. The first membership function will refer to the function which represents the queue is not full, while the second one refers to the full function.

For the 'not full' function, the output will be 0 if the input falls in zone before 0 or zone after 500 of x-axis. The output will be 1 or referring to true when the queue is empty or until it is 50% full. After 50% threshold, the function will become a liner equation dropping from 1 to 0 linearly. The linear function used to fuzzify this is as shown below:

```
preout1[i]=(myLen[i][4]-q_len)/(myLen[i][4]-myLen[i][3])
```

The degree from 1 to 0 will represent how true or how empty the queue is where 0 represents the queue is full. For the 'full' function, the output will be 0 if the input falls in zone before 250 or zone after 500 of x-axis. The output will be 1 or referring to true when the queue is 100%. Before the 50% queue capacity threshold, the function will be 0 to indicate that the queue is not full at all. After the 50% threshold, the function will increase linearly from 0 to 1. The degree from 0 to 1 will represent how full the queue is. The linear function used to fuzzify this is as shown:

```
preout1[i]=(q_len-myLen[i][1])/(myLen[i][2]-myLen[i][1])
```

4.3.3 Rate of Change of Queue Fuzzy Set

Rate of change of queue membership function will be represented by an array myRate[4][6] of type double. The dimension of this array is 2 and it has 4 rows and 6 columns as shown:

{-q_lim, -q_lim, 20-q_lim, 0.0, 0.0, 0.0},

{-q_lim/2, -10.0, 0.0, 0.0, 0.0, 0.0},

{0.0, 0.0, 10.0, q_lim/2, 0.0, 0.0},

{0.0, q_lim-20, q_lim, q_lim, 0.0, 0.0}

As the array in myRate is used to represent only trapezium graph, there is no need to have the first column to determine the shape of the graph as in Current queue length membership function. All the rows have common column type, which represent 6 different points or areas in the graph. The value shown on each column will represent the x-coordinates in the graph. The corresponding y-coordinates have been set to the value of {0, 1, 1, 0, 0, 0}.

4.3.4 Rate of Change of Queue Membership Function Graph

By plotting all the value of the myRate array into the same graph, the relationship among all the functions could be seen as below.

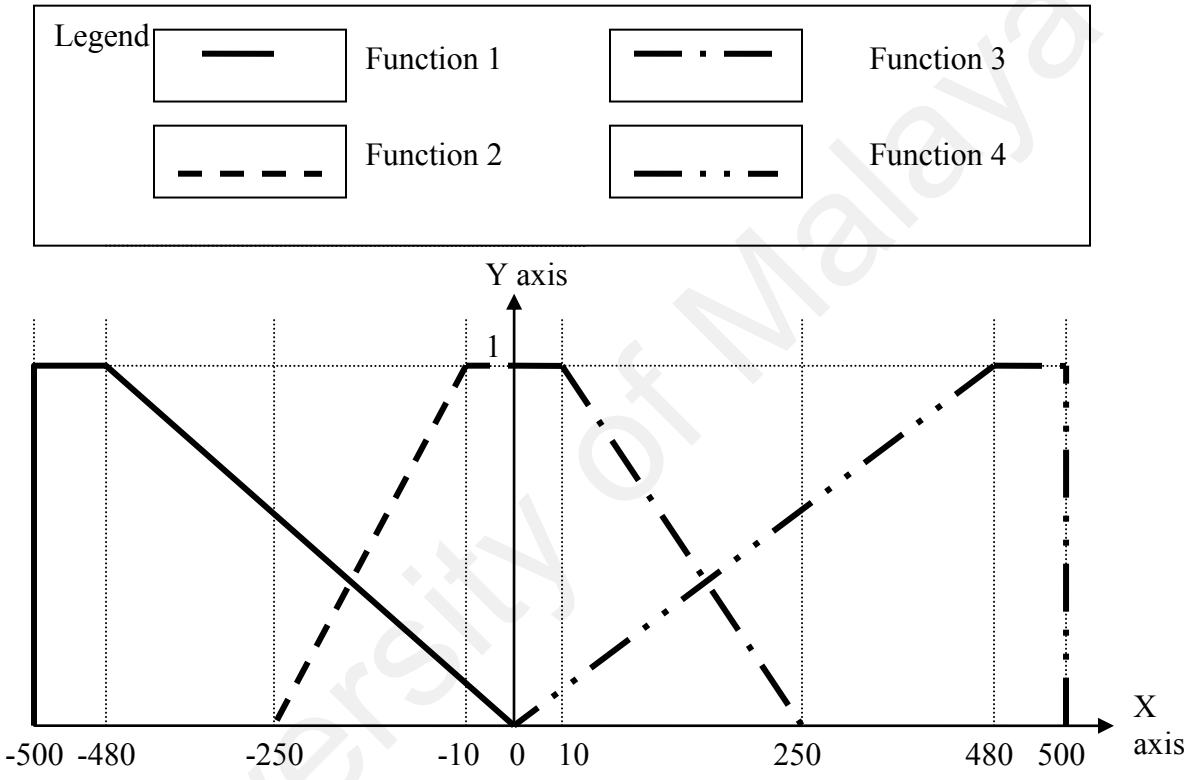


Figure 4.7 Rate of Change of Queue Membership Function Graph

4.3.4.1 Rate of Change of Queue Rules

The rate of change of queue length function will be governed by a set of rules. The rate of change of queue will be divided into four zones, which are decreasing fast, decreasing, increasing and increasing fast. The first membership function will be referring to the function which represents the queue is decreasing fast, while the second one is referring to the rate of change is decreasing. On the other side, the membership function 3 will be used to refer to increasing rate, while function 4 will be used to refer to increasing fast.

For function 1, the rate is said to be decreasing fast (equal to 1) if the difference between current queue length and average length is between -500 and -480 packets. The rate will be decreasing linearly between -480 and 0 packets area. The degree of decreasing fast will become untrue when it reaches 0 packet difference.

For function 2, the rate is said to be decreasing (equal to 1) if the difference between current queue length and average length is between -10 and 0 packets. And the rate will be increasing linearly between -250 and -10 packets area. The degree of decreasing will become untrue when it reaches -250 packet differences. The reason why the degree on decreasing decreased is because the queue is decreasing fast.

For function 3, the rate is said to be increasing (equal to 1) if the difference between current queue length and average length is between 0 and 10 packets. And the rate will be decreasing linearly between 10 and 250 packets area. The degree of increasing will

become untrue when it reaches 250 packet differences. The reason why the degree on increasing decreased is because the queue is entering the increasing fast zone.

For function 4, the rate is said to be increasing fast (equal to 1) if the difference between current queue length and average length is between 480 and 500 packets. And the rate will increase linearly between 0 and 480 packets area. The degree of increasing fast will become untrue if there is 0 packet of difference.

4.4 Fuzzy Inference

The inference process that will be used would differ from the Madami method. The whole process can be divided into two parts, fuzzifying inputs and application of fuzzy operator and output membership function.

The fuzzy inference model has a typical rule, which takes the form as follows:

If Input X and Input Y, then Output is Z

where X and Y will be representing the first and second input.

The output Z_i of each rule will be weighted by the weight value of w_i of each rule. The weight value is predefined according to the situation of the bottleneck link. There are two common fuzzy operator used, but for this thesis only one operator will be used which is

the AND operator. The AND operator is also known as the minimum operator. The operator AND will select the lowest value of two outputs.

```
for (i=0;i<8;i++)
{
        totalY+=myWeight[i]*foutputHeight[i];
        totalH+=foutputHeight[i];
}

if (totalH>0)
        finalOutput=totalY/totalH;
else
        finalOutput=0;

return finalOutput;
}
```

The final output is calculated by the block of code shown above and the value is returned to the caller function. The variable finalOuput is used to store the calculation result. The result will be the summation of all rule output (w_i z_i) and averaged by the summation of all the output strength (z_i). The final output calculation could be represented by the formula (4.1) as shown (Asai95):

$$\text{Final Output} = \frac{\sum_{i=1}^{N} W_i Z_i}{\sum_{i=1}^{N} Z_i} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (4.1)$$

4.4.1 Fuzzy Inference for Current Queue Length and Rate of Change of Queue

The fuzzy inference part for this thesis will be based on a set of rules. The rules are as follows:

If Input X and Input Y, then Output is Z.

Input X will be referring to the current queue length function while input Y will be referring to the rate of change of queue. In order for the fuzzy inference to work, a set of weight has been declared in the fuzzy function. The weight is as shown below

```
float myWeight[8] = {1,2,3,4,5,6,7,8};
```

There are a total of 8 weights used, because there are 8 possible outputs from the inference process. The weight is chosen to represent the degree of critical at the bottle neck queue. The most critical situation will be given weight 8 while the least critical one will be weighted 1. The command line below:

```
float preout1[2]={0};
float preout2[4]={0};
```

is used to declare an array to store inference result before it is used to calculate the final output.

float foutputHeight[8]={0} will be used to keep the result after applying the AND fuzzy operator.

The whole process begins when two parameters presents. The first parameter is the current queue length. The current queue length will be used to find output from the current queue length function. Both graphs from the function will be used to get the output. And then the second input, the rate of change of queue will be used to find out the respective output. Each output will be weighted accordingly to the weight array. For example if the current queue length is 100 and the rate of change is -100 packets output will be as shown. The number besides the figure represents the rules number. Each rule will be weighted accordingly to their weight in the array respectively (example: Rule 1 will be weighted by myWeight[1]=1 and Rule 8 will be weighted by myWeight[8]=8.

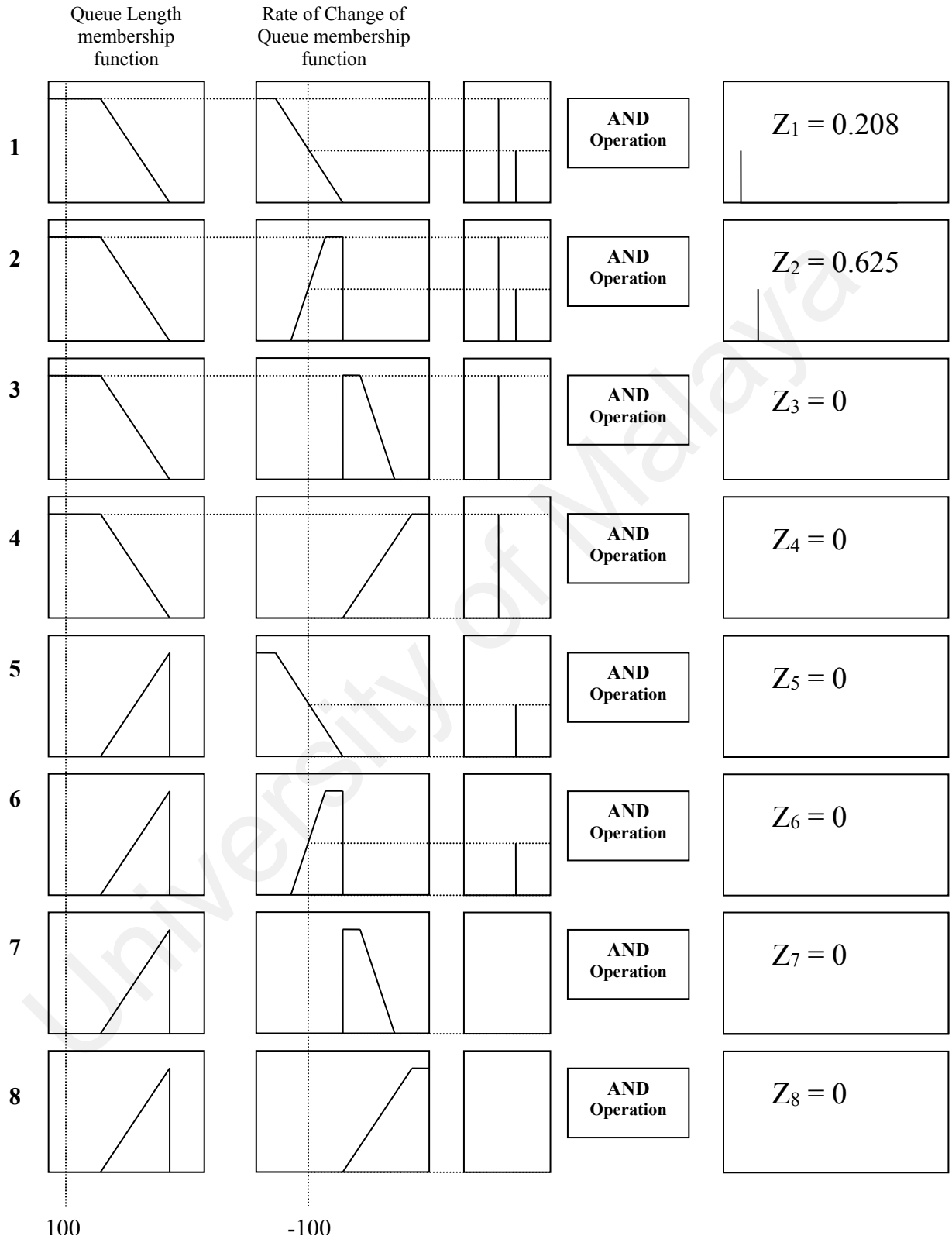


Figure 4.8 Fuzzy inference processes

To get the final required output, the inference results must be defuzzified. The defuzzification process for the above example will be calculated using formula (4.1) as follow:

$$\textbf{Final Output} = \frac{(1x\ 0.208) + (2\ x\ 0.625) + (3\ x\ 0) + (4\ x\ 0) + (5\ x\ 0) + (6\ x\ 0) + (7\ x\ 0) + (8\ x\ 0)}{0.208 + 0.625}$$

$$= \frac{1.458}{0.833}$$

$$= \mathbf{1.75}$$

The value of the final output tells that the congestion is not serious. As a result packet might be queued. Further action will be taken accordingly depending on the value of the final output value.

The following examples will take into consideration two cases, first is the low congestion case and the other, high congestion case.

For the low congestion case, the values which will be used for calculation are current queue length = 10 packets and the rate of change of queue = -480 packets. With these values, we can conclude that the queue is only 2% full and the rate of change is decreasing tremendously.

For the serious congestion case, the values which will be used for calculation are current queue length = 480 packets and the rate of change of queue = 480 packets. With these values, we can conclude that the queue is 96% full and the queue is growing in size very fast.

The following two diagrams will show the fuzzy inference processes for both extreme situation and the corresponding results of both situations.

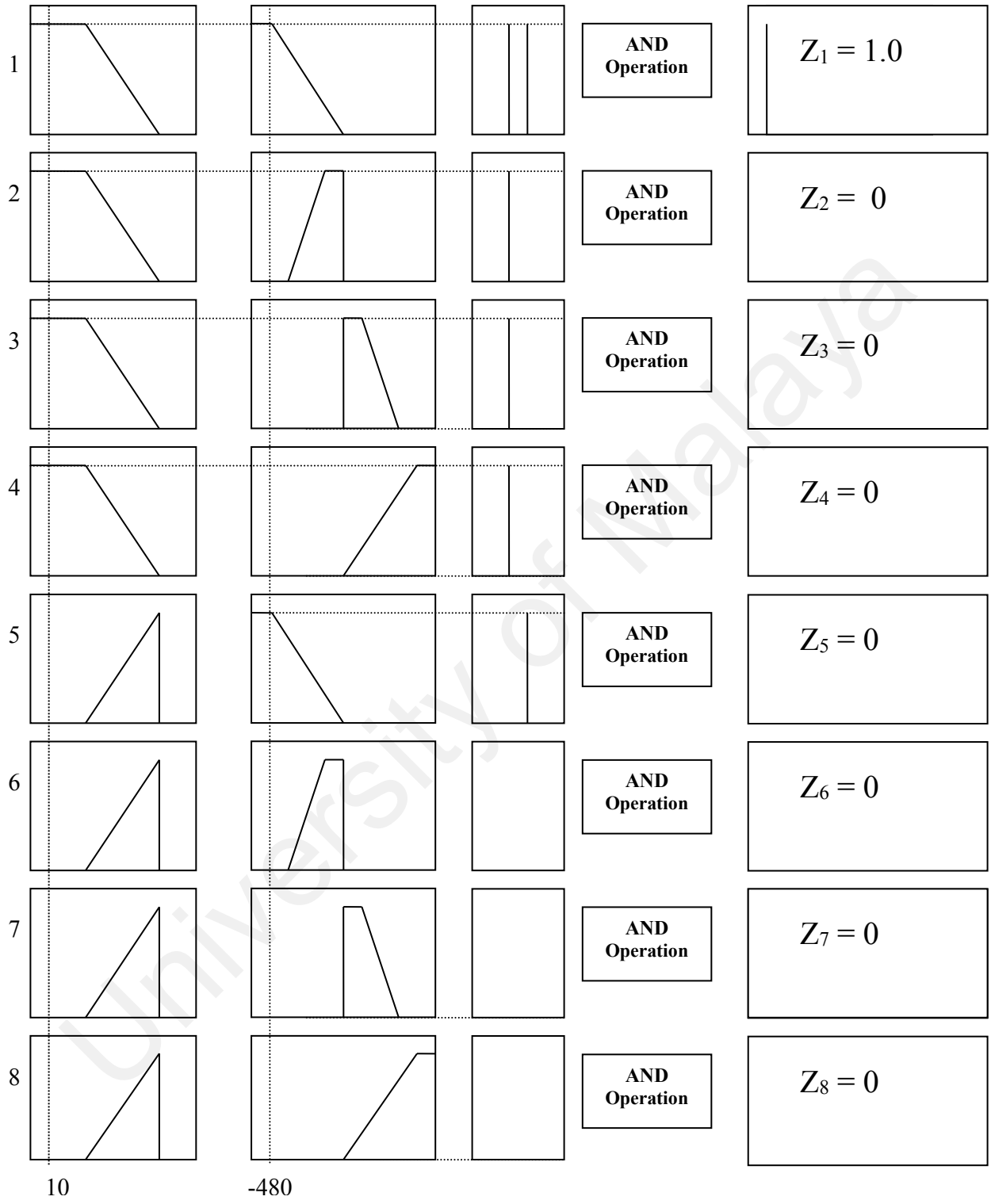


Figure 4.9 Low congestion fuzzy inference processes

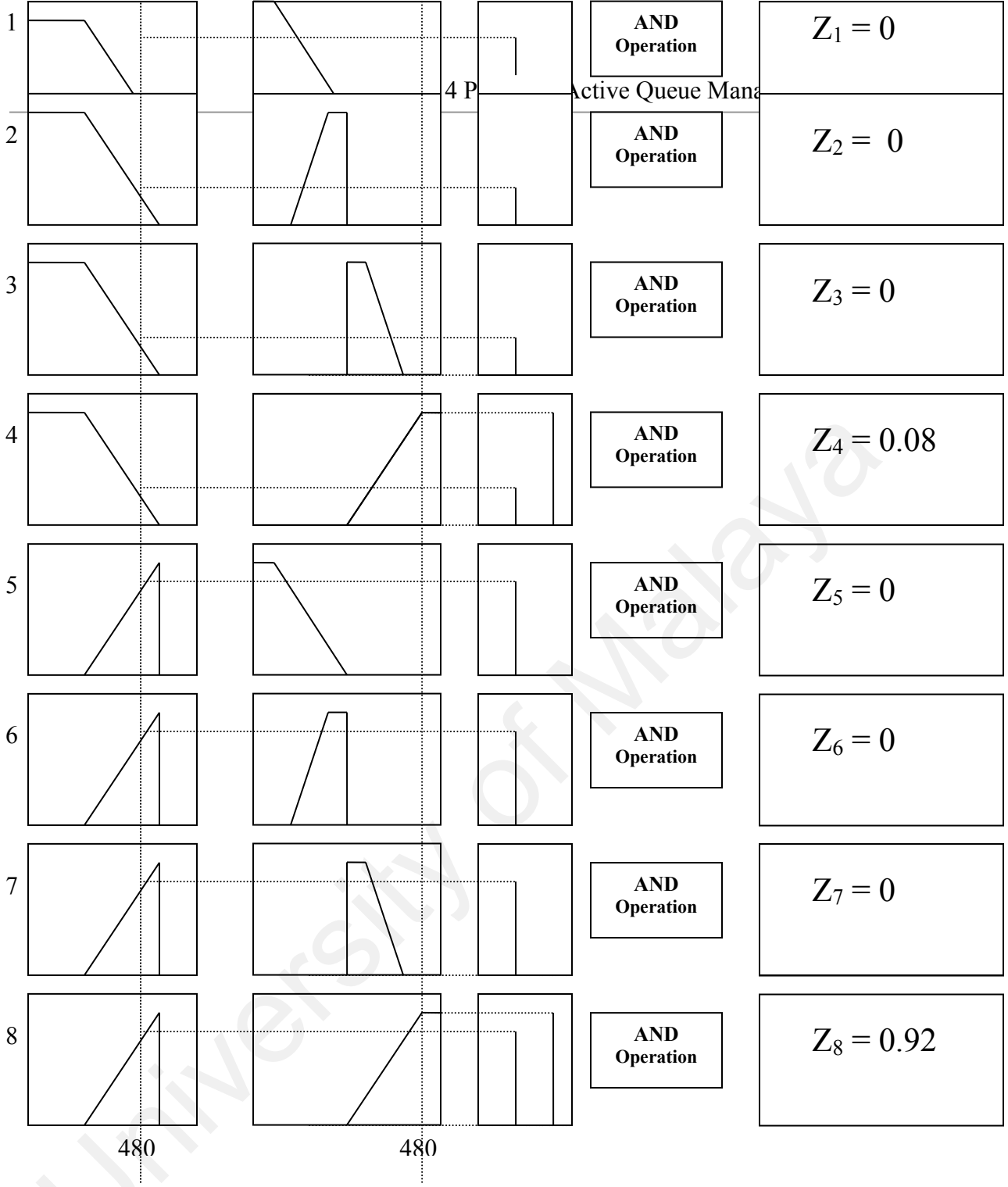


Figure 4.10 Serious congestion fuzzy inference processes

Defuzzification calculation for low congestion scenario is as shown using formula (4.1):

$$\textbf{Final Output} = \frac{(1x\ 1.0) + (2 \times 0) + (3 \times 0) + (4 \times 0) + (5 \times 0) + (6 \times 0) + (7 \times 0) + (8 \times 0)}{1.0}$$

$$= \frac{1.0}{1.0}$$

$$= \mathbf{1.00}$$

Defuzzification calculation for serious congestion scenario is as shown below using formula (4.1):

$$\textbf{Final Output} = \frac{(1x\ 0) + (2 \times 0) + (3 \times 0) + (4 \times 0.08) + (5 \times 0) + (6 \times 0) + (7 \times 0) + (8 \times 0.92)}{0.08 + 0.92}$$

$$= \frac{7.68}{1.0}$$

$$= \mathbf{7.68}$$

4.4.2 Fuzzy Inference result assessment

After the defuzzified results have been calculated, the fuzzy logic function will return a value of type float to the function, which called it. The function which called the fuzzy logic function is fuzzy_result=fuzzy(q_->length(),prev_5,qlim).

The defuzzified value will now be stored in fuzzy_result variable. This value will then be assessed to carry out appropriate action. The first step assessment step taken is to cast the value into 9 categories of seriousness of the bottleneck link. The value used is from 1 to 9 where 9 will be used to represents the most serious bottleneck link condition. In order to classify the fuzzy result, a block code of if else if clause will be used as shown below:

```
if (fuzzy_result>=7.5)
  fuzzy_result=9.0;
else if (fuzzy_result>=7)
  fuzzy_result=8.0;
else if (fuzzy_result>=6)
  fuzzy_result=7.0;
else if (fuzzy_result>=5)
  fuzzy_result=6.0;
else if (fuzzy_result>=4)
  fuzzy_result=5.0;
else if (fuzzy_result>=3)
  fuzzy_result=4.0;
else if (fuzzy_result>=2)
  fuzzy_result=3.0;
else if (fuzzy_result>=1)
  fuzzy_result=2.0;
else if (fuzzy_result>=0)
  fuzzy_result=1.0;
```

Any fuzzy result returned with the value of 7.5 and above will be classified as 9.0 (very serious condition). If it is less than 7.5 but above or equal to 7 it will be classified as 8.0. This process will be continued until the value of class 1.0. Classification process can be summarized using a Figure 4.11 as shown below:

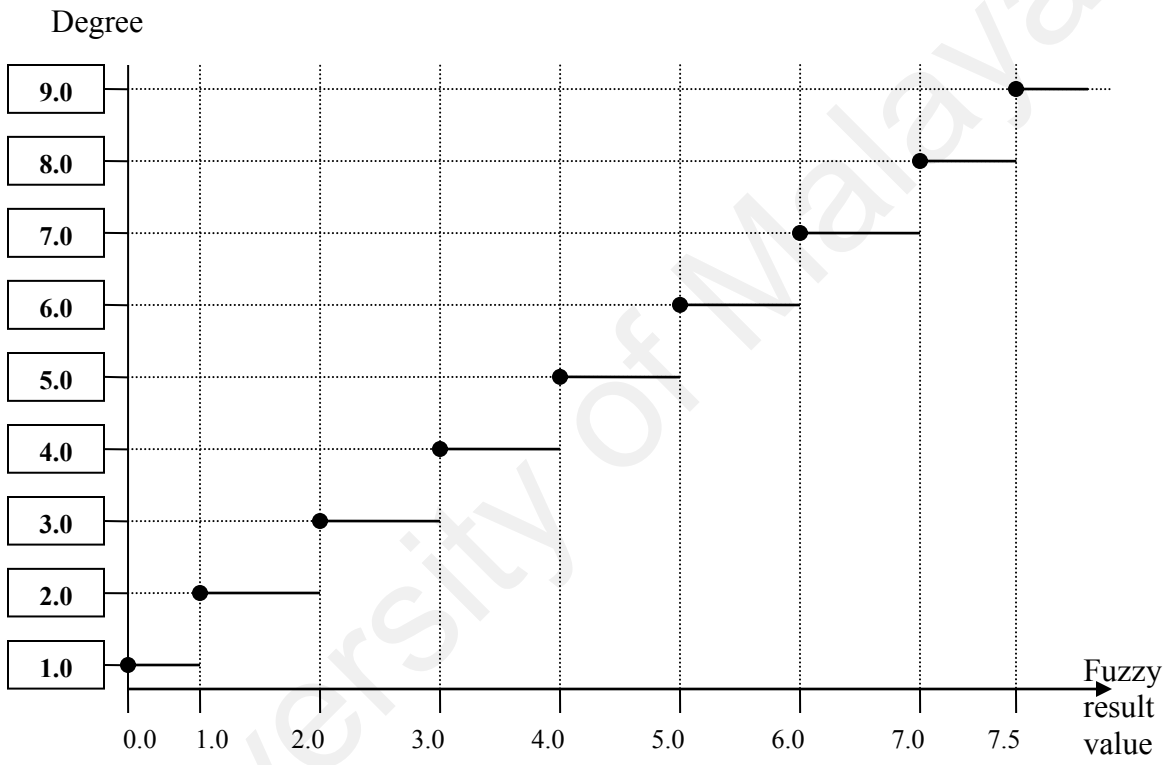


Figure 4.11 Fuzzy inference result casting

Based on these 9 situations or degree of seriousness, the corresponding action will be taken. For each degree, a set of RED thresholds and drop probability will be assigned. This value will then overwrite the existing values, which have been given at the beginning of the simulation, or the value set by previous fuzzy assessment.

The set of thresholds value and drop probability can be further categories into two different categories. One category is the sets of value for packets with precedence 0 and the other set is for packets with precedence 1. In general the two categories represent the code point 20 (or virtual queue 1) and 21 (or virtual queue 2) respectively.

The sets of thresholds and drop probability according to each degree are shown in the Table 4.1 below:

Table 4.1 Sets of thresholds and drop probability according to each cast degree value

Degree	In profile			Out profile		
	Min. th.	Max. th.	Drop prob.	Min. th.	Max. th.	Drop prob.
1	90	100	0.08	88.5	98.5	0.10
2	85	100	0.10	83.5	98.5	0.12
3	80	100	0.12	78.5	98.5	0.14
4	85	90	0.14	83.5	88.5	0.16
5	80	90	0.16	78.5	88.5	0.18
6	75	90	0.18	73.5	88.5	0.20
7	70	80	0.20	68.5	78.5	0.22
8	65	80	0.25	63.5	78.5	0.27
9	60	80	0.30	58.5	78.5	0.32

When the precedence value matched and the degree is known, the minimum threshold will be set according to the value of Min. th. and maximum threshold as Max. th. with the drop probability of Drop prob.

Each set of value will be then set into the simulator before the packet is to be queued. Hence the value of the thresholds and drop probability will be dynamic and the changes are according to the degree of seriousness at the bottleneck link. Therefore the outcome of this whole process will create a dynamic active queue management with the help of fuzzy logic.

Chapter 5: Coding and Implementation

As known, ns2 is written in C++ language and the simulation script is written in Tcl scripting or Otcl. Hence, this chapter will be organized into two main parts. The first part will be explaining the tcl simulation scripts used and the second part will be explaining the C++ codes. Chapter organization is as follows:

5.1 Simulation Script

5.2 Coding and Implementation

5.1 Simulation Script

The simulation script is written in tcl language. The main file for the simulation is the diffnet.tcl file. When the simulation starts, it will call other files as follows:

1. peer_setup.tcl
2. 2q2p.tcl
3. topology.tcl
4. monitoring.tcl

The peer_setup.tcl file consists of tcl script that creates HTTP, FTP and VoIP traffic. The 2q2p.tcl file consists of tcl script that sets all the DiffServ specific procedures. The topology.tcl file consists of tcl script, which creates the topology. The monitoring.tcl file is used to load monitoring functions for the queues and traffic flows.

The set testTime command in the diffnet.tcl file sets the simulation time in seconds. The $defaultRNG seed command is to set the seed value to be used for each time the simulation starts. The default value set for the seed is 0, which means that a random seed will be used every time when the simulation starts. Each time the simulation starts; it will randomly select a seed that will produce random simulation traffic. Therefore each simulation is different from each one except when the random seed is the same.

The launch command will call the launchSession procedure in the peer_setup.tcl file to start creating and generating web traffic. The command launchHttp httpC1 http S4 is used to launch HTTP traffic starting by the http client C1 issuing an http request to http server S4. The command launchFtp ftpC1 ftp S4 is used to launch FTP traffic starting with the ftp client C1 issuing an ftp request to ftp server S4. launchVoip voip1 voip5 0 is used to start VoIP traffic between VoIP 1 and VoIP 5 node with an assigned id of 0.

The policer type used for this simulation is the TSW2CM policer. In order for this policer to function, we need to set the committed information rate (CIR) value. CIR value is used to determine the committed transfer rate for each application. The CIR value chosen for each traffic is different and it is as listed below:

1. set cir_exp 50000 (to set the VoIP CIR rate)
2. set cir_http 70000 (to set the HTTP CIR rate)
3. set cir_ftp 150000 (to set the FTP CIR rate)

The confDSCore command is used to configure the desired queue and policy parameters for the two bottle neck nodes. The details of how or what parameters to be set are located in the 2q2p.tcl file. The command confDSCore r6 r7 is used to set the parameters for traffic flow from router 6 to router 7. The flow of the traffic from router 7 to router 6 is done with another command because the links between the two bottleneck routers are simplex link. Simplex link can only transmit packets in one direction. That is why we need to explicitly set the parameters for the two simplex links.

The confDSEdges command is used to set desired queue and policy parameters for the pair of connection. For example confDSEdges voip1 voip5 $cir_exp 29_app AF will configure the CIR rate of the connection or traffic between as the value set in the set cir_exp. The 29_app is a code to set an ID to determine that traffic with number 29 is VoIP traffic and the initial type of service it require is AF service. 27_app refers to FTP traffic while 31_app refers to HTTP traffic.

The $ns at 50 "timeStats 50" will print the simulation packets statistic on the screen every 50 simulation seconds interval. As the simulation ends, the $ns at $testTime "close-connections" command will try to cleanly close all connection pair by sending finish acknowledgement to connection pair router. The $ns at [expr $testTime + 5] "kill-em-all" will force all remaining connections to close down. [expr $testTime + 5]means that after the simulation time ends, it will then force all remaining connections to close down after another 5 simulation seconds.

At $testTime + 11, the $ns at [expr $testTime + 11] "finish" will call the finish procedure. The 'finish procedure' will finish up the whole simulation. The procedure is listed below:

```
proc finish {} {
   global ns alku trace_on pf q
   $ns flush-trace

   # Print final DiffServ queue statictics from the bottleneck link
   $q(r6r7) printStats
   $q(r7r6) printStats

   #puts "\Simulation seconds: [expr [clock seconds] - $alku] s"

   flush $pf
   close $pf

   exit 0
}
```

It will print all the final DiffServ queue statistics from the bottleneck link and print out the amount of simulation time in seconds and exit the whole simulation.

5.1.1 Seeding

The $defaultRNG refers to the random number generation (RNG) within the mathematical support found in ns2. The RNG class contains implementation of the combined multiple recursive generator MRG32k3a proposed by L'Ecuyer (NS03). Works

done by L'Ecuyer could be found in (L'Ecuyer99), (L'Ecuyer01) and (L'Ecuyer02). MRG32k3a is capable to generate 1.8 X 10^{19} independent streams of random numbers. Each stream will further consist of 2.3 X 10^{15} substreams (NS03). According to the ns Manual, or formerly known as ns Notes and Documentation,

> "each substream has a period of 7.6 X 10^{22}. The period of the entire generator is 3.1 X 10^{57}."

The default value for the seed is 12345. This simulation have set the $defaultRNG 0 which will set the seed based on current time of day counter. There is no guarantee where two random seed will not overlap (NS03). By setting $defaultRNG 0 also means that the simulation will get a non-deterministic behavior.

5.1.2 Simulation topology

The simulation topology is created in the topology.tcl file. The simulation uses a dumb-bell topology with 14 routers and various numbers of source nodes depending on how heavy the traffic that would like to be generated. There are several researches published which favor this topology such as (Clark98), (Chrysostomou03), (DiFata03), (RFC2415), (Liang03) and (Sahu00). For light traffic simulation the number of source node is limited to 24 nodes. For heavy traffic, the number of sources is increased one fold to 48 nodes. Below is the network topology for the simulation with light traffic.

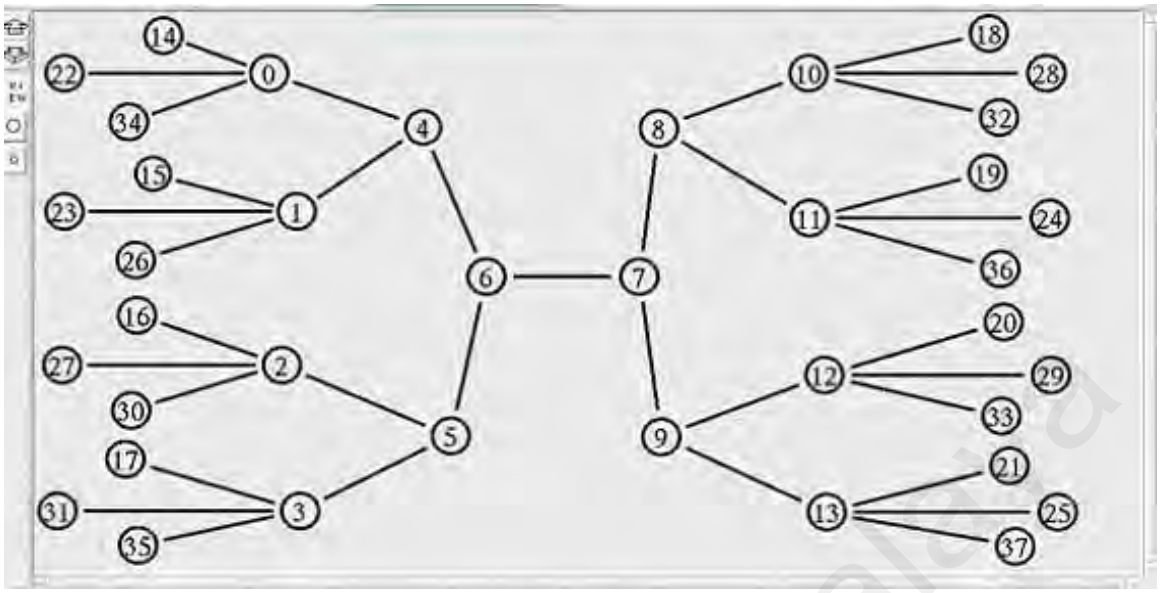


Figure 5.1 Light traffic simulation topology

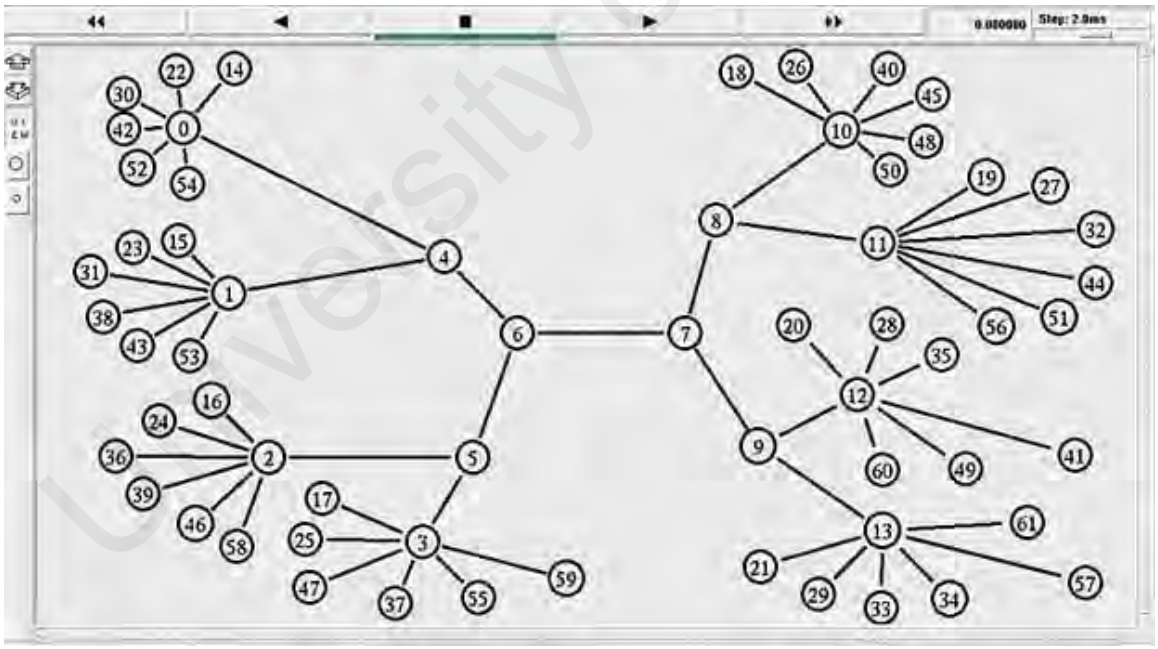


Figure 5.2 Heavy traffic simulation topology

Nodes numbered 4 until 9 represents the core routers while nodes numbered 0, 1, 2, 3, 10, 11, 12, 13 are the edge routers. The rest of the nodes represent the traffic sources. Summary of the number of nodes for light and heavy traffic simulation is as shown in the table 5.1

Table 5.1 Nodes amount for light and heavy simulation

Num.	Type	Quantity	
		Light Traffic	Heavy Traffic
1	Routers	14	14
2	VoIP	8	16
3	FTP Clients	4	8
4	FTP Servers	4	8
5	HTTP Clients	4	8
6	HTTP Servers	4	8
Total		**38**	**62**

The codes used to generate the nodes are as shown below:

```
for {set i 0} {$i < $num_r} {incr i} {
   set n(r$i) [$ns node]
   set r_list "[set r_list] [list r$i]"
}
for {set i 1} {$i <= $num_voip} {incr i} {
   set n(voip$i) [$ns node]
   set voip_list "[set voip_list] [list voip$i]"
}
for {set i 1} {$i <= $num_ftpC} {incr i} {
   set n(ftpC$i) [$ns node]
   set ftpC_list "[set ftpC_list] [list ftpC$i]"
}
for {set i 1} {$i <= $num_ftpS} {incr i} {
   set n(ftpS$i) [$ns node]
   set ftpS_list "[set ftpS_list] [list ftpS$i]"
}
for {set i 1} {$i <= $num_httpC} {incr i} {
   set n(httpC$i) [$ns node]
   set httpC_list "[set httpC_list] [list httpC$i]"
```

```
}
for {set i 1} {$i <= $num_httpS} {incr i} {
   set n(httpS$i) [$ns node]
   set httpS_list "[set httpS_list] [list httpS$i]"
}
```

It uses a for loop to generate the desired number of nodes. All nodes or routers generated using the [$ns node] code. To further classify the nodes, special names will be given. The table below will show the corresponding name used to label the nodes and its type.

Table 5.2 Nodes label

Num.	Code	Type
1	(r$i)	Router
2	(voip$i)	VoIP sources and destinations
3	(ftpC$i)	FTP Clients
4	(ftpS$i)	FTP Servers
5	(httpC$i)	HTTP Clients
6	(httpS$i)	HTTP Servers

5.1.3 Link Bandwidth and Delay

All the traffic sources will be connected to the edge routers using 10.0 Mb simplex links with 10ms of delay. Two simplex links are created for two different directions, one for sending packets from source or destination node to edge router and one from edge router to source or destination node. Node label with the number 0 and 1 represent the bottleneck link of the whole simulation. Between these two nodes, two separate simplex links is created each with bandwidth of 1.5Mb and 15ms delay. This is sufficient enough to create a very small bottle link to force packets to be dropped. The reason of the existence of the bottle link is to test the ability of the active queue management model. From the edge routers to the next inner router (node 4, 5, 8 and 9), a 10.0Mb bandwidth with 10ms delay duplex link is created. From router 4, 5, 8 and 9 to the bottleneck routers, a 4.5Mb bandwidth with 15ms delay duplex link is used. These links are used to create a more focused traffic before entering the bottleneck link. The traffic before entering the bottleneck link will have packets with less distance between them, to create a sudden burst of packets for the bottleneck link. Figure 5.3 shows the existing links and its respective bandwidth and delay for light traffic simulation.

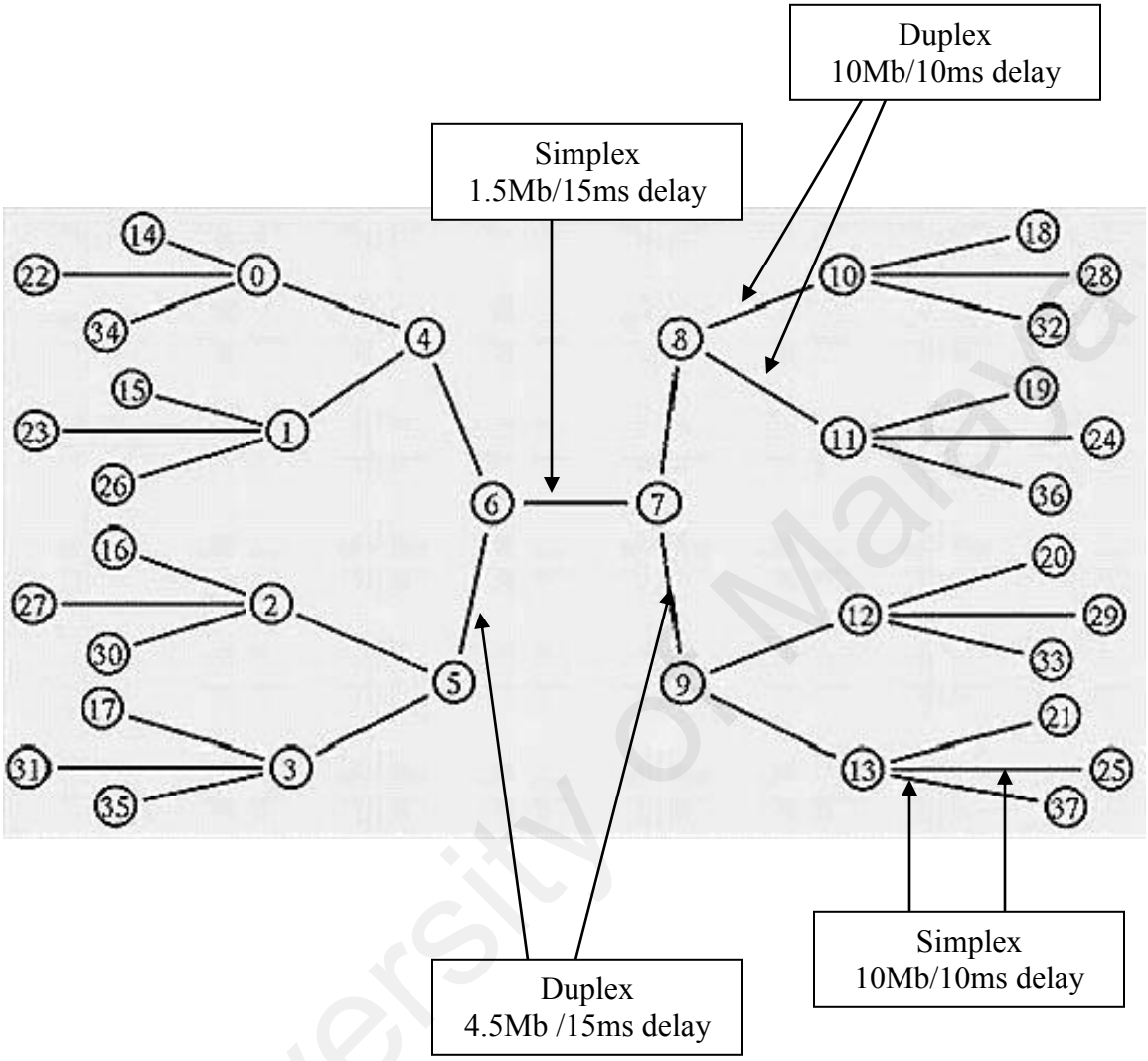


Figure 5.3 Links bandwidth and delay

5.1.4 Traffic source

Traffic sources are divided into three different categories, which are:

1. Hyper Text Transfer Protocol (HTTP)
2. File Transfer Protocol (FTP)
3. Voice over IP (VoIP)

Each type of traffic has per-define parameters and will generate in a random manner a number of packets according to those initial parameters. For further information of the characteristic of the world wide web (WWW) please refer to (Cunha95).

5.1.4.1 HTTP Traffic

The tcl script in the peer_setup.tl file will create an HTTP-page pool, with defined session parameters and call will call by launchSession command in the diffnet.tcl file. The traffic parameters assigned are mean values of an exponential distribution (Luoma00). The parameters listing are as follows:

Table 5.3 Parameters listing for HTTP traffic

Parameter name	Value	Description
sessionSize	5	Number of pages in a session
interSession	5	Time between session arrivals (in seconds)
pageSize	3	Number of objects in a page
interPage	12	Time between page requests (in seconds)
objSize	3300	Object size (in bytes)
interObj	0.1	Time between object requests (in seconds)

5.1.4.2 FTP Traffic

The tcl script in the peer_setup.tl file will create an FTP-pagepool, with defined session parameters and call will call by launchSession command in the diffnet.tcl file. The traffic parameters listing (Luoma00) are as follow

Table 5.4 Parameters listing for FTP traffic

Parameter name	Value	Description
sessionSize	5	Number of pages in a session
interSession	30	Time between session arrivals (in seconds)
pageSize	1	Always 1
interPage	0	Used to determine the first send time
objSize	[expr 1e6]	Object size (in bytes)
interObj	0.1	Time between object requests (in seconds)

In FTP traffic, the page size is always equal to 1 as a file is equivalent to a file.

5.1.4.3 Page Pool

The set pool [new PagePool/WebTraf] for both HTTP and FTP traffic refers to the web traffic model that utilizes PagePool framework (NS03). Page pools are used by server to generate page information by caches to describe which pages are in store. It is also used by the clients to generate a request stream. The data structure can be found in the webcache folder, webtraff.h file. The class WebTrafSession is a class that models web user session. And the WebPage is used to models web page.

5.1.4.4 VoIP Traffic

The tcl script in the peer_setup.tl file will create VoIP traffic; with defined session parameters and call will call by launchSession command in the diffnet.tcl file. The traffic parameters listing (Luoma00) are as follows:

Table 5.5 Parameters listing for VoIP traffic

Parameter name	Value	Description
packetSize	[expr 160+40]	Data + Header (Size of packet generated)
burst_time	180	Burst time (in seconds) which refers to average On time for the generator
Pidle_time	[format %.1f [expr[$expoo set burst_time_]*(1/$load - 1)]]	The average Off time for the generator
rate_	100000	Sending rate during On time
id_	$id	Flow id for each define VoIP traffic
interObj	0.1	Time between object requests (in seconds)
load	0.6	Mean inter-arrival or mean holding time in seconds

The type of traffic for VoIP is UDP. The VoIP traffic will be generated exponentially using the command line set expoo [new Application/Traffic/Exponential].

5.1.4.5 HTTP and FTP Session Launcher

The procedure launchSession is used to set the parameters define for HTTP and FTP above and to start the traffic generation.

```
proc launchSession {pool clnt svr sessionSize_ interSession_ pageSize_ interPage_ objSize_ interObj_} {
   global ns n testTime

   # Maximum number of sessions to be created
   set numSession_  expr round(1.4*$testTime/$interSession_)]
   puts "Creating max $numSession_ sessions"

   # Bind TCL-variable to C-variable
   $pool set testTime_ $testTime

   # Add this session to the session list
   $ns set sessionList "[$ns set sessionList] $pool"

   # Setup servers and clients: add nodes listed in src_ and dst_
   # to pagepools server- and client-pool
   $pool set-num-client [llength $clnt]
   $pool set-num-server [llength $svr]

   set i 0
   foreach s $clnt {
           $pool set-client $i $n($s)
           incr i
   }
   set i 0
   foreach s $svr {
           $pool set-server $i $n($s)
           incr i
   }

   # Number of Sessions
   set numSession $numSession_

   # Inter-session Interval
   set interSession [new RandomVariable/Exponential]
   $interSession set avg_ $interSession_
```

```
    # Number of pages in a session
    set sessionSize [new RandomVariable/Exponential]
    $sessionSize set avg_ $sessionSize_

    # Create sessions
    $pool set-num-session $numSession
    set launchTime [$interSession value]
    for {set i 0} {$i < $numSession} {incr i} {

        ## Number of objects per page
        if {[$pool set application_] == 27 } {

            set pageSize [new RandomVariable/Constant]
            $pageSize set val_ $pageSize_
        } else {
            set pageSize [new RandomVariable/Exponential]
            $pageSize set avg_ $pageSize_
        }

        set interPage [new RandomVariable/Exponential]
        $interPage set avg_ $interPage_

        set interObj [new RandomVariable/Exponential]
        $interObj set avg_ $interObj_

        set objSize [new RandomVariable/Exponential]
        $objSize set avg_ $objSize_

        if { $launchTime < $testTime } {
            $pool create-session $i [$sessionSize value] $launchTime \
            $interPage $pageSize $interObj $objSize
        }
        set launchTime [expr $launchTime + [$interSession value]]
    }

}
```

It is used to create random variable and the start up time for each session (each request). The code block if {[$pool set application_] == 27} will set the page size variable for FTP while the else clause will set for application with number 31 (HTTP) the page size variable. This is needed because for FTP page size is 1 but for HTTP it is not equal to 1

and it will generate an average number for the page size using the RandomVariable/Exponential command. By using RandomVariable/Exponential command, random traffic could be generated using exponential distribution, which reflect real Internet traffic behaves.

5.1.5 Random Variable

The code RandomVariable/Exponential actually refers to a class RandomVariable that provides a thin layer of functionality on top of the base random number generator and the default random number stream and it is define in ranvar.h file (NS03). The abstract classes derived from the class RandomVariable have specific distributions. Each distribution has its own parameter. The defined distributions and their respective parameters are listed below in Table 5.6

Table 5.6 Distribution Parameters

Num	Abstract class name	Parameters associated
1	class UniformRandomVariable	min_ , max_
2	class ExponentialRandomVariable	avg_
3	class ParetoRandomVariable	avg_ , shape_
4	class ParetoIIRandomVariable	avg_ , shape_
5	class ConstantRandomVariable	val_
6	class HyperExponentialRandomVariable	avg_ , cov_
7	class NormalRandomVariable	avg_ , std_
8	class LogNormalRandomVariable	avg_ , std_

From the table, it is clear that RandomVariable/Exponential is the default is used as random number generator. In general, the RandomVariable/<type of random-variable> will create an instance of random object which will create random variables with specific distribution (NS03).

5.1.6 Transport Agent API

The application class is used to provide basic application behavior such as send, receive, resume, start and stop events (NS03). In the real situation, the application will access the network through application programming interface (API) such as sockets. In ns, it is realized using API functions, which are mapped to the appropriate internal agents functions (NS03).

For example the code block for VoIP that is used to do this is as follow

```
set udp [new Agent/UDP]
set null [new Agent/UDPsink]
$ns attach-agent $n($src) $udp
$ns attach-agent $n($dst) $null
$ns connect $udp $null
```

set udp [new Agent/UDP] is used to create UDP agent and the set null [new Agent/UDPsink] is used to create a sink or the place to receive the generated UDP packets. $ns attach-agent $n($src) $udp will then attach the UDP agent to the source and $ns attach-agent $n($dst) $null will attach the sink to the destination. Whenever UDP packets are created by that particular source, it will then travel through the network through the bottle link and to the destination defined for the packet itself.

The block code

```
set expoo [new Application/Traffic/Exponential]
$expoo attach-agent $udp
$expoo set packetSize_  [expr 160+40]; # data + header
$expoo set burst_time_  180
$expoo set idle_time_   [format %.1f [expr [$expoo set burst_time_]*(1/$load - 1)]]
$expoo set rate_        100000
$expoo set id_          $id
```

is used to attach the application to the transport agent. attach-agent will be bind to C++ code in the agent.cc file. agent_ pointer in the class Application will be pointed to the transport agent and it will then call the attachApp( ) functions to set the pointer to point back towards the application. By doing this, a consistency is being assured between the OTcl and C++ (NS03).

5.1.7 Exponential Traffic

The [new Application/Traffic/Exponential] in the 2q2p.tcl file is actually the application and transport agent API explained in chapter 35 of the ns manual (NS03). Application sits on top of a given transport agents. The exponential traffic is a type of traffic generator derived from four C++ class,

1. EXPOO_Traffic (Exponential On/Off Distribution)
2. POO_Traffic (Pareto On/Off Distribution)
3. CBR_Traffic (Constant Bit Rate)
4. TraficTrace (Generate traffic according to trace file)

The EXPOO_Traffic class will generate traffic according to an Exponential On/Off distribution. One of the behavior for this type of traffic is packet of constant size will be sent at a fixed rate during On periods and no packets are sent during Off period. The periods for On and Off are taken from an exponential distribution.

Therefore, the Application/Traffic/Exponential will mean that during On period, packets are generated at a constant burst rate and during Off period, no traffic will be generated. Both time burst time and idle time is taken from exponential distribution.

5.1.8 Connection Pair Configuration

Between two communicating nodes, there are few DiffServ procedures need to be set. Those procedures are found in the 2q2p.tcl file in the confDSEdge procedure

```
proc confDSEdges { svr clnt cir app service_type } {
   global ns q n cir_exp cir_http cir_ftp BE EF AF

   if { $service_type == "AF" } {
         set cp_ $AF(cp)
         set in_min_ $AF(in_min)
         set in_max_ $AF(in_max)
         set in_prob_ $AF(in_prob)
         set out_min_ $AF(out_min)
         set out_max_ $AF(out_max)
         set out_prob_ $AF(out_prob)
         set qlimit_ $AF(qlimit)
   }
   # Edge queues (both in server- and client-side)
   set q($svr$clnt) [[$ns link $n($svr) [$n($svr) neighbors]] queue]
   set q($clnt$svr) [[$ns link $n($clnt) [$n($clnt) neighbors]] queue]

   foreach qe "$svr$clnt $clnt$svr" {
         $q($qe) set numQueues_ 1
         $q($qe) setNumPrec 2
         $q($qe) meanPktSize 300
         #$q($qe) set limit_ $qlimit_
         $q($qe) setPhysQueueSize 0 $qlimit_
         $q($qe) setMREDMode WRED

         $q($qe) addPolicyEntry -1 -1 TSW2CM $cp_ $cir $app
         # Multiple precedences only for AF
```

```
 $q($qe) addPolicerEntry TSW2CM $cp_ [expr $cp_ + 1]

        $q($qe) addPHBEntry $cp_ 0 0
        $q($qe) addPHBEntry [expr $cp_ + 1] 0 1
        $q($qe) configQ 0 0 $in_min_ $in_max_ $in_prob_
        $q($qe) configQ 0 1 $out_min_ $out_max_ $out_prob_
    }
}
```

For each connection pair, the RED threshold is set using

```
        set in_min_ $AF(in_min)
        set in_max_ $AF(in_max)
        set in_prob_ $AF(in_prob)
        set out_min_ $AF(out_min)
        set out_max_ $AF(out_max)
        set out_prob_ $AF(out_prob)
```

block of command. It specifies the minimum and maximum threshold of in profile and out profile packets, stated as in_min, in_max, out_min and out_max. In is for in profile packets or packets with the codepoint 20 and out is for out of profile packet or packets with the codepoint 21.

The in_prob and out_prob are the drop probability of codepoint 20 and 21 respectively.

```
        set q($svr$clnt) [[$ns link $n($svr) [$n($svr) neighbors]] queue]
        set q($clnt$svr) [[$ns link $n($clnt) [$n($clnt) neighbors]] queue]
```

are used to specify the source, destination and the border routers they are connected to.

All of the connection pair packets will be going through one physical queue with two virtual queues, each for codepoint 20 and 21.

```
        $q($qe) set numQueues_ 1
        $q($qe) setNumPrec 2
```

numQueues is referring to the physical queue and setNumPrec will create the virtual queue in the physical queue.

$q($qe) meanPktSize 300 is used to set the packet size value for average queue size calculation. This is important when the queue is empty when the average queue size needs to be calculated.

$q($qe) setPhysQueueSize 0 $qlimit_ set the physical queue limit or queue size. The value of $qlimit_ is being defined earlier in the RED parameter definition of set AF(qlimit) 500

$q($qe) setMREDMode WRED will set the MRED mode. MRED mode is used to calculate the queue size and there are several variants. WRED is one of the variant which is used in this simulation. It will calculate the probability based on single physical queue length.

$q($qe) addPolicyEntry -1 -1 TSW2CM $cp_ $cir $app will define the type of policer that will be used. In this simulation the Time Sliding Window with 2 Color Marking (TSM2CM) policer is used. $cp_ will define the initial codepoint for the type of traffic class stated in the $app. $cir is the committed information rate set in the diffnet.tcl file according to type of traffic (e.g. HTTP, FTP, VoIP).

$q($qe) addPolicerEntry TSW2CM $cp_ [expr $cp_ + 1] is used to add the another entry for the policer. Here it specifies that the next lower precedence for those packets with the initial codepoint 20 plus 1, which is 21.

$q($qe) addPHBEntry $cp_ 0 0 and $q($qe) addPHBEntry [expr $cp_ + 1] 0 1 is used to add an entry into the PHB table and maps the codepoint. Here it defines that packet with the codepoint 20, will be queued in the physical queue's first virtual queue. Where else, packet with the codepoint 21, will be queued in the physical queue's second virtual queue.

For each virtual queue, the corresponding RED parameters are set by using $q($qe) configQ 0 0 $in_min_ $in_max_ $in_prob_ and $q($qe) configQ 0 1 $out_min_ $out_max_ $out_prob_ command. Virtual queue 1 will be accommodating those in profile packet and will be monitoring those packets with $in_min_ $in_max_ $in_prob_ RED parameters. While virtual queue 2 will be housing those out of profile packet and monitoring those packets with $out_min_ $out_max_ $out_prob_ RED parameters.

5.1.9 Bottle Neck Router Configuration

Procedure confDSCore is used to configure the bottleneck routers. It is call in the diffnet.tcl file confDSCore r6 r7 and confDSCore r7 r6 lines. The confDSCore procedure has two arguments, a and b which refers to the node id such as in this case r6 or r7. The whole procedure code block is shown below.

```
proc confDSCore { a b } {
   global ns q n AF

   set q($a$b) [[$ns link $n($a) $n($b)] queue]

   # The number of physical queues and precedences
   $q($a$b) set numQueues_ 1
   $q($a$b) setNumPrec 2
   $q($a$b) meanPktSize 300
   $q($a$b) setMREDMode WRED

   # Set scheduler (and queue weigths)
   $q($a$b) setSchedularMode PRI      ;# Options: RR, WRR, WIRR, PRI

   # Configure PHBs

   # AF
   $q($a$b) addPHBEntry $AF(cp) 0 0 ;
   $q($a$b) addPHBEntry [expr $AF(cp)+1] 0 1
   $q($a$b) configQ 0 0 $AF(in_min) $AF(in_max) $AF(in_prob)
   $q($a$b) configQ 0 1 $AF(out_min) $AF(out_max) $AF(out_prob)
   $q($a$b) setPhysQueueSize 0 $AF(qlimit)
  }
```

The set q(ab) [[$ns link $n($a) $n($b)] queue] command line will name the pair of routers as q(ab). Two different simplex links have been declared earlier and each link will be handling two different traffics from two different directions. The first direction is q(r6r7) which is referring to the traffic from the left hand side of the network to the right hand side (refer to figure 5.1 and figure 5.2, the dumbbell topology). While the second direction is from right to left, known as q(r7r6).

Each direction will have one physical queue with two virtual queues in it. $q($a$b) set numQueues_ 1 and $q($a$b) setNumPrec 2 refer to the physical queue and virtual queue respectively.

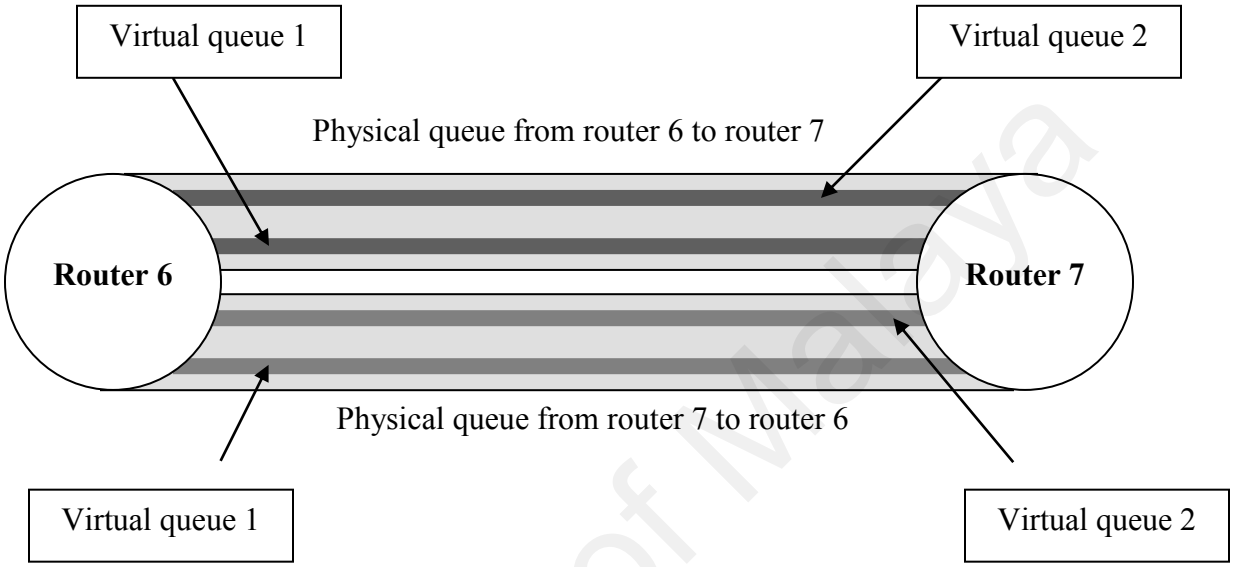


Figure 5.4 Queue organizations in the bottleneck link

$q($a$b) meanPktSize 300 is the value use to calculate average queue size especially when the queue is empty. The $q($a$b) setMREDMode WRED will implement a WRED variant of MRED mode.

$q($a$b) setSchedularMode PRI will declare the type of scheduler mode used at the bottle link routers. The type of scheduler used here is Priority Queuing scheduler. This type of scheduler will first empty higher priority packets (codepoint 20) first before emptying lower priority packets (codepoint 21) from the queue. So, most of the time, forwarding treatment will favor codepoint 20 packets and therefore, most of the time, codepoint 21 packets will remain in the queue longer. This is logic as generally we would want to treat codepoint 20 packets better than those with codepoint 21. Therefore,

codepoint 20 packets will experience less delay and jitter. Only the bottle link routers are equipped with scheduler.

```
$q($a$b) addPHBEntry $AF(cp) 0 0 ;
$q($a$b) addPHBEntry [expr $AF(cp)+1] 0 1
$q($a$b) configQ 0 0 $AF(in_min) $AF(in_max) $AF(in_prob)
$q($a$b) configQ 0 1 $AF(out_min) $AF(out_max) $AF(out_prob)
$q($a$b) setPhysQueueSize 0 $AF(qlimit)
```

will determine or tell the bottle link routers how the DiffServ parameters are to be set. The parameters here are as the same as those parameters with the one applied to each connection pairs at the edge.

5.1.10 Weighted RED

The WRED used for this simulation is for 2 drop precedence queue. Packet precedence will be represented by codepoint 20 and 21, where codepoint 20 is the initial codepoint. There will be a set of threshold applied to each drop precedence practically known as the in and out. Below is the tcl script used to set the value of the threshold and the drop probability.

```
set AF(in_min) 33     ;# ~33% of the queue limit
set AF(in_max) 60     ;# ~60% of the queue limit
set AF(in_prob) 0.65
set AF(out_min) 3     ;# ~3% of the queue limit
set AF(out_max) 33    ;# ~33% of the queue limit
set AF(out_prob) 0.87
set AF(qlimit) 500
```

These limits when plotted on the same graph will look like the graph depicted in Figure 5.5 below.

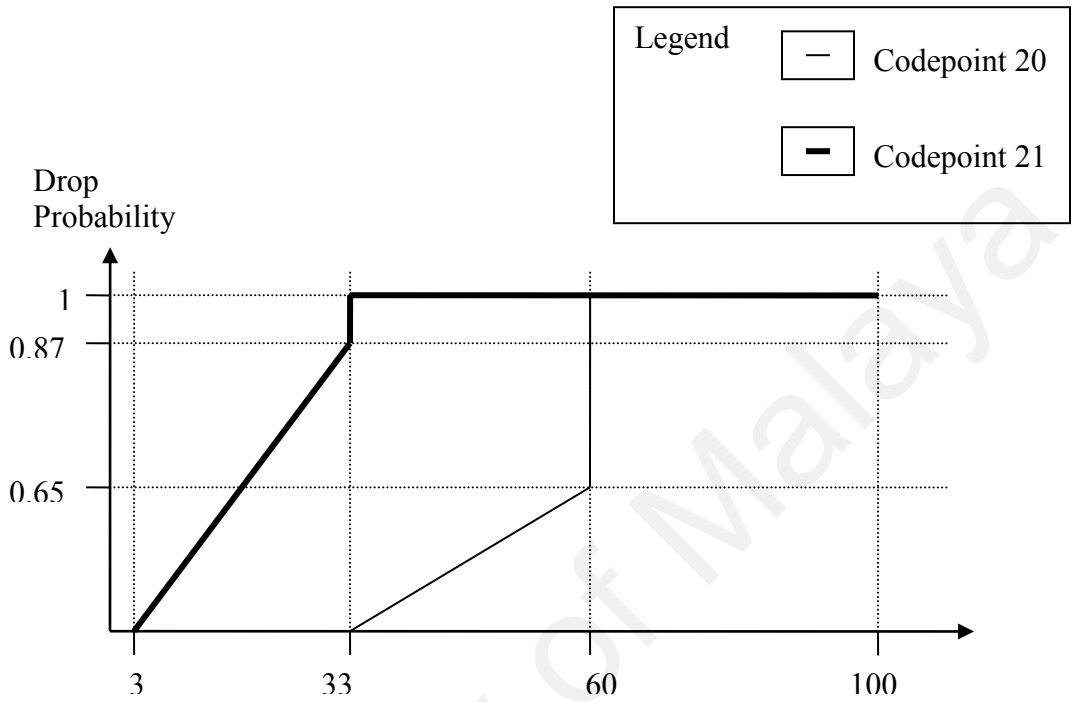


Figure 5.5 RED parameters for codepoint 20 and 21

The type of RED parameters set is known as staggered RED. Practically, staggered RED will provide greater assurance towards packets with codepoint 20 than 21 (Makkar00). The threshold value for codepoint 21 is lower than those for codepoint 20 and the drop probability for codepoint 20 is lower than codepoint 21. By using these settings, a service differentiation could be witness where, codepoint 20 is actually more important than code point 21. This behavior could be manipulated to toggle the packet preference.

5.2 Coding and Implementation

In this chapter, we will go into detail the C++ files that have been modified in order to implement the fuzzy logic engine. All the files needed or related to this thesis are located in the diffserv folder in the ns main folder. The files needed for this thesis are

1. dsredq.h
2. dsredq.cc

Two major functions have been added into the dsredq files which are a data structure (link list) and the fuzzy logic code.

5.2.1 dsredq.h

In this header file, the function signature is added and is regarded as public function. The function signature that have been added is float fuzzy(int q_len, float avg, int q_lim). The data structure code is implemented in the existing int enque(Packet *pkt, int prec, int ecn) function.

5.2.1.1 Data Structure

A data structure has been added in order to keep track of all the queue length value including the physical queue length and each and every physical queue length. The type

of data structure used is a type of link list. The link list starts with a declaration of the data structure in the dsredq.h file. Each node in the link list structure will have the properties as shown below:

```
struct node
{
        int num;
        node *next;
};
```

Each node will have one variable of type integer. The int num is used to store the physical queue size. The last item in the node structure is a pointer to a node. It is used to link the node to the other node. The node structure can be represented by the following Figure 5.6.

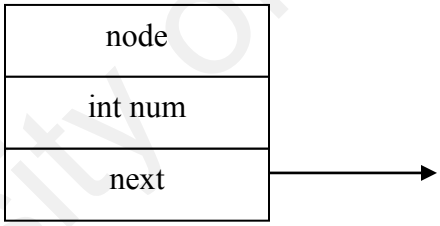


Figure 5.6 Node data structure

Before the link list could be used, the object must be declared every time the simulation starts. This is done by putting the code as shown below:

```
node *r6_head;
node *r6_cur;
node *r6_temp;
node *r6_travel;

node *r7_head;
node *r7_cur;
node *r7_temp;
node *r7_travel;
```

in the class redQueue. The purpose of putting the this block of code is to associate it with the main class of the function (redQueue) such that it could be used in its class. With this block of code, every time when simulation starts, 8 node pointers will be created.

5.2.2 dsredq.cc

The dsredq.cc contains all the procedure or functions, which will be used for simulation. The actual action taken could be seen in the functions in the dsredq.cc. The codes for this thesis have been inserted into the enqueue () function in this file.

5.2.2.1 Constructor

In the dsredq.cc file, 8 nodes will be declared in the constructor shown below:

```
redQueue::redQueue() {
r6_head=NULL;
r6_cur=NULL;
r6_temp=NULL;
r6_travel=NULL;

r7_head=NULL;
r7_cur=NULL;
r7_temp=NULL;
r7_travel=NULL;

  numPrec = MAX_PREC;
  mredMode = rio_c;
  q_ = new PacketQueue();
  }
```

mredMode = rio_c is used to set the default MREDMode if it is specified explicitly by the user.

5.2.2.2 Link List

The link list structure is created starting from the first packet being queued into the physical queue. A new entry will be added to the link list whenever there is a subsequent packets being successfully queued. The link list structure code is divided into two parts using the if else clause, as shown below:

```
if(src_id == 14 || src_id == 15 || src_id == 16 || src_id == 17 || src_id == 22 || src_id == 23
|| src_id == 26 || src_id == 27 || src_id == 30 || src_id == 31 || src_id == 34 || src_id == 35)
{
if(r6_head==NULL)
  {
          r6_head=new node;
          r6_head->num=q_->length();
          r6_head->next=NULL;
          r6_cur=r6_head;
  } else {
         r6_temp=new node;
         r6_temp->num=q_->length();
         r6_temp->next=r6_cur;
         r6_cur=r6_temp;
  } } else {
if(r7_head==NULL)
  {
         r7_head=new node;
         r7_head->num=q_->length();
         r7_head->next=NULL;
         r7_cur=r7_head;
  } else {
         r7_temp=new node;
         r7_temp->num=q_->length();
         r7_temp->next=r7_cur;
         r7_cur=r7_temp;
  }
```

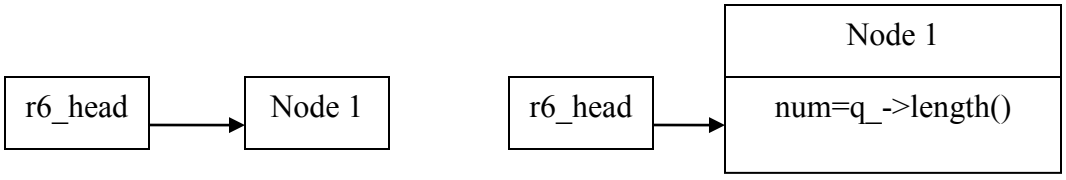
The above mention code block is to create and store a link list for those packets traveling from router 6 to router 7 at the bottleneck link. The list of sources id (14, 15, 16, 17, 22, 23, 26, 27, 30, 31, 34 and 35) is referring to those sources located at the right hand side of

the simulation topology. This is crucial to explicitly make sure that all packets traveling thru the bottleneck link from router 6 to router 7 are been traced into the link list. Each node in the link list will be representing the queue length of the simplex queue from router 6 to router 7 whenever a packet has been queued.

Inside the **if** clause, there is another **if else** clause. This is to differentiate that the link list is empty (when the simulation starts) from the link list is not empty. The first ever packet that is successfully been queued with initiate the if(r6_head==NULL) part. This is because at the very beginning of the simulation when the head pointer is being declared, it is given a value NULL.

Figure 5.7 Head pointer creation and head pointer pointing to first node

r6_head=new node will then create a new node and the subsequent code



r6_head=new node **r6_head->num=q_->length()**

Figure 5.8 Creating new node with head pointer and assigning queue length value

r6_head->num=q_->length() will get the current queue size and put it into the node.

r6_head->next=NULL will declare that the next pointer in the node will be pointing to a NULL value. This will mark the very beginning of the link list. r6_cur=r6_head is used to assign a pointer head to the link list to mark the first node in the link list.

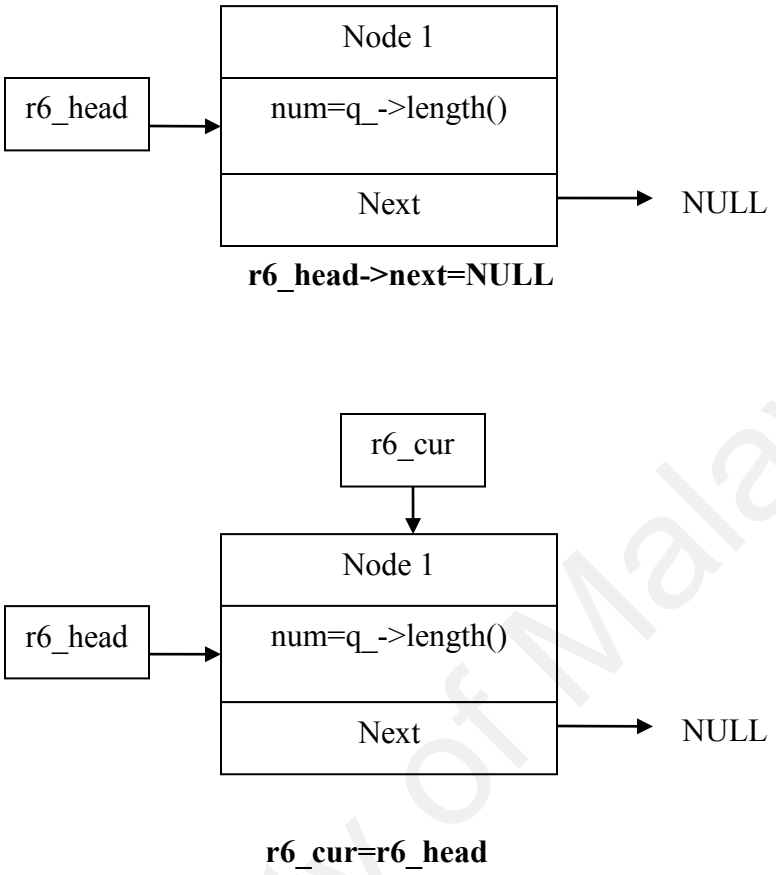


Figure 5.9 Assigning head->next to null and pointing r6_cur to head

If the packet has been successfully queued, and it is not the first packet ever been queued, the else will be triggered.

r6_temp=new node will declare a new node named as r6_temp.

r6_temp->num=q_->length() will then get the queue length value and put it into the temporary node.

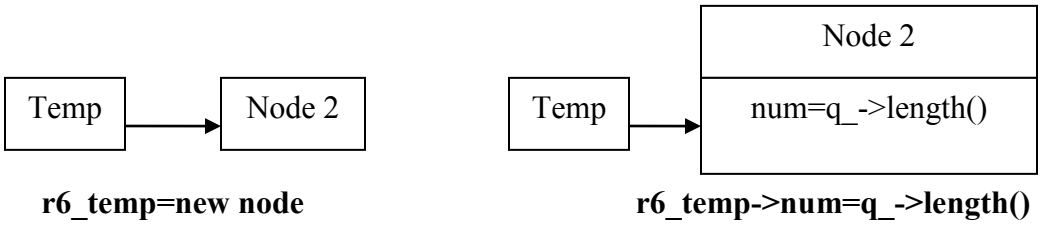


Figure 5.10 Creating temporary node and assigning queue length value

r6_temp->next=r6_cur will then link the temporary node to the existing link list. The second node in the link list will be linked to the head node and if it is the third node onwards, it will be linked to the most current node in the link list.

r6_cur=r6_temp will shift the current pointer to the latest node in such that it will mark the latest node added for next round.

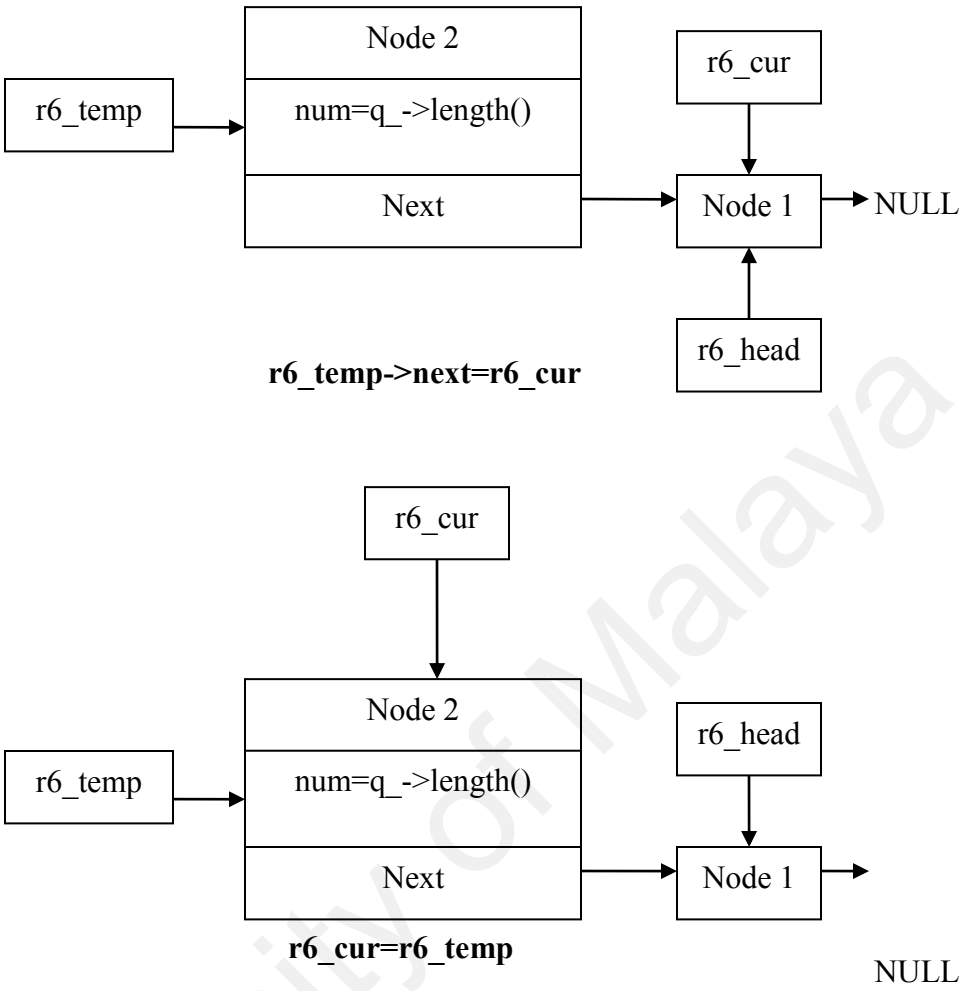


Figure 5.11 Linking temporary node to existing list and shifting current node pointer

The above block of code will be repeated with the condition of source node set to using the else clause. This block of code will do the same action for those traffic queued which comes from the left hand side of the network (node 18, 19, 20, 21, 24, 25, 28, 29, 32, 33, 36 and 37).

5.2.2.3 Fuzzy Logic

The fuzzy logic code is inserted into the enqueue() function. The code

```
fuzzy_result=fuzzy(q_->length(), prev_20, qlim)
```

will be used to call the fuzzy logic code located in the dsreqq.cc file itself.

The fuzzy logic function call has a set of parameters being passed to it. Three parameters are to be passed to the function, the current physical queue length, the average queue length and the physical queue limit. The fuzzy logic function will pass back a variable of type float to be assigned to fuzzy_result variable for further action to be taken as explained earlier in chapter 4.

Chapter 6: Simulation Results and Analysis

6.1 Simulation Organization

The simulations are divided into two big categories, which are as follows:

1. Light Traffic
2. Heavy Traffic

In each category, there will be two sets of data being collected. One set will be the simulation result without fuzzy logic implementation. The other set will be the simulations result with fuzzy logic. Figure 6.1 below describes the organization of the simulations results.

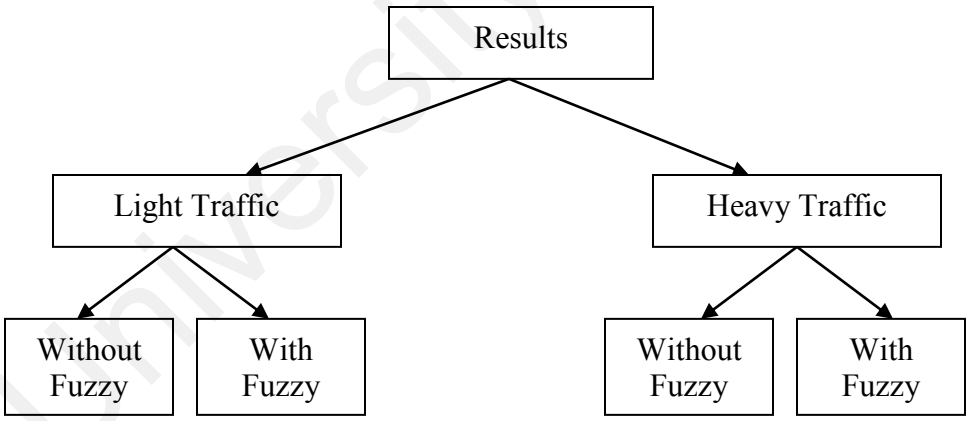


Figure 6.1 Simulation organizations

6.2 Simulation results

Simulations results that will be printed on the screen will represented in a table form, which is also known as packets statistic. The command line printStats will actually print the table on the screen and for this thesis purpose, packets statistic table will be printed at an interval of 50 simulation seconds. Each time when the packets statistic table being printed; there will be two sets of tables. The first set will represents the packets statistic for traffic from router 6 to router 7. While the second set will represents the packets statistic for traffic from router 7 to router 6. For analysis purpose, packets statistic results for each interval seconds of 100 will be recorded starting from 100s. Example of the packets statistic table is as shown below in Table 6.1.

Table 6.1 Packets statistic table

Codepoint	Packets received	Packets sent	Tail dropped packet	RED early dropped packet
All	412893	350867	45926	16100
20	296320	260206	25534	10580
21	116573	90661	20392	5520

The purpose of recording packets statistics at the bottleneck link is because the bottleneck link is simplex link. Hence the data collected can clearly reflects the direction of the packets in the network. Furthermore, the fuzzy logic implementation is done in the two bottleneck routers. Therefore to actually see or determine the successfulness of the FuzAQM, the bottleneck routers packets statistic must be studied. Throughput terms, which will be used onwards, is actually referring to the packets sent. This figure will reflect the amount of packets which has successfully traveled through the bottleneck link.

In this thesis it is assume that for each packet, which has successfully traveled through the bottleneck link, it will reach its destination for sure. The bottleneck link is actually the link with lowest bandwidth and highest delay and the links after or before the bottleneck link are faster and with lower delay.

6.2.1 Light Traffic Simulations and Results

Light traffic simulations are those simulations with 24 sources and destinations nodes. Simulation carried out for light traffic can be as long as 800 simulation seconds. There will be a total amount of 25 sets of simulations being carried out and the average value of each codepoint throughput and the overall throughput for each 100 seconds will be recorded.

6.2.2 Heavy Traffic Simulations and Results

Heavy traffic simulations are those simulations with 48 sources and destinations nodes. Simulation carried out for light traffic will be as long as 800 simulation seconds. There will be a total amount of 25 sets of simulations being carried out. The average value of each codepoint throughput and overall throughput for each 100 seconds will be recorded for all the 25 sets of simulation.

6.3 Light Traffic Simulations Results Analysis

In the light traffic simulations, the fuzzy logic implementation does improve the total throughput. Throughput value is calculated using the formula as shown.

$$\text{Throughput} = \frac{\text{Total of packets successfully travel through the bottleneck link}}{\text{Total of packets received at the bottleneck link}}$$

The throughput calculation is based on one direction (from r6 to r7 or r7 to r6) at the bottleneck link exclusively. The average percentage of amount of packets successfully traveled through the bottleneck link is shown in figure 6.2 and 6.3 below.

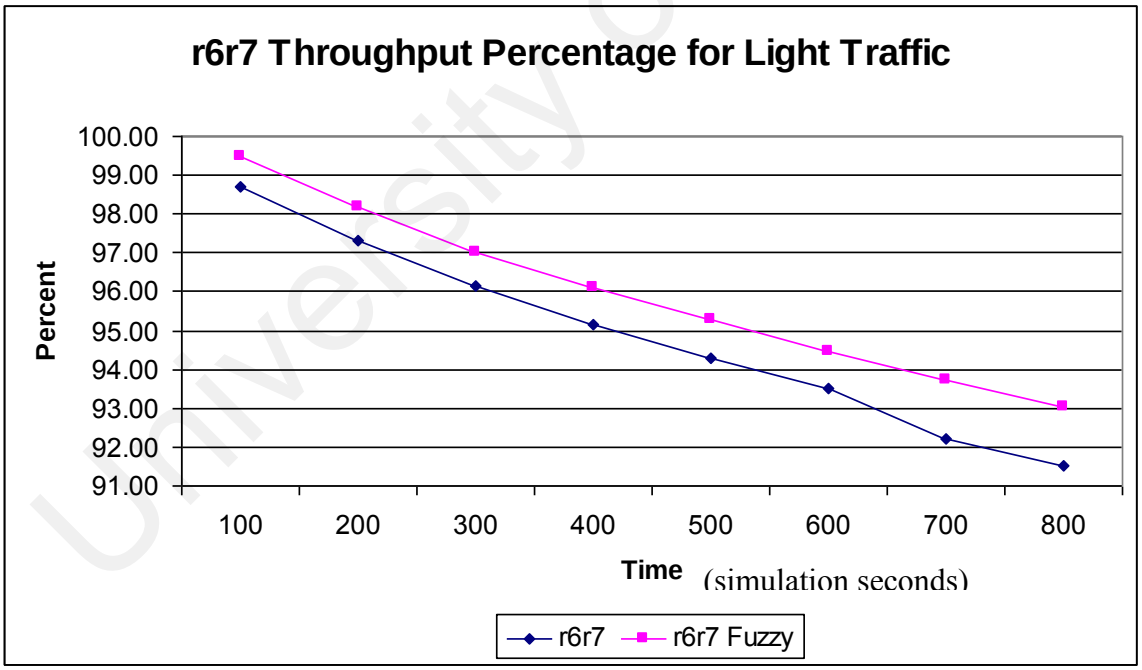


Figure 6.2 r6r7 throughput percentage for light traffic

Table 6.2 r6r7 throughput percentage values for light traffic

	Time (simulation seconds)							
	100	200	300	400	500	600	700	800
r7r6 (in percent)	99.48	99.10	97.03	96.10	95.27	94.40	93.74	93.05
r7r6 Fuzzy (in percent)	98.69	97.30	96.14	95.16	94.31	93.51	92.23	91.52

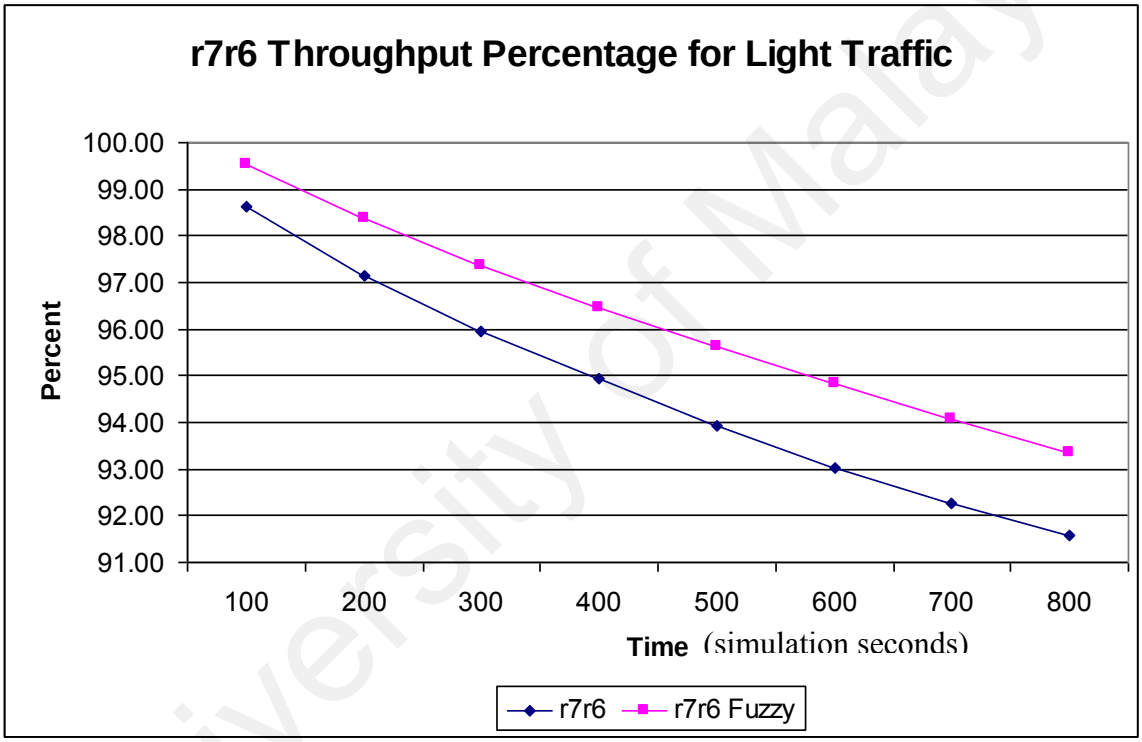


Figure 6.3 r7r6 throughput percentage for light traffic

Table 6.3 r7r6 throughput percentage values for light traffic

	Time (Simulation seconds)							
	100	200	300	400	500	600	700	800
r7r6 (in percent)	99.53	98.38	97.37	96.46	95.63	94.85	94.06	93.38
r7r6 Fuzzy (in percent)	98.61	97.15	95.94	94.95	93.94	93.04	92.26	91.57

The line above the lowest line in both line graphs (Figure 6.2 and Figure 6.3) shows that the fuzzy implementation does improve the average percentage of throughput of packets. Improvement can be seen in between 0.79% till 1.79% throughout 800 seconds of simulation. The graphs below show the average improvement percentage of light traffic simulations.

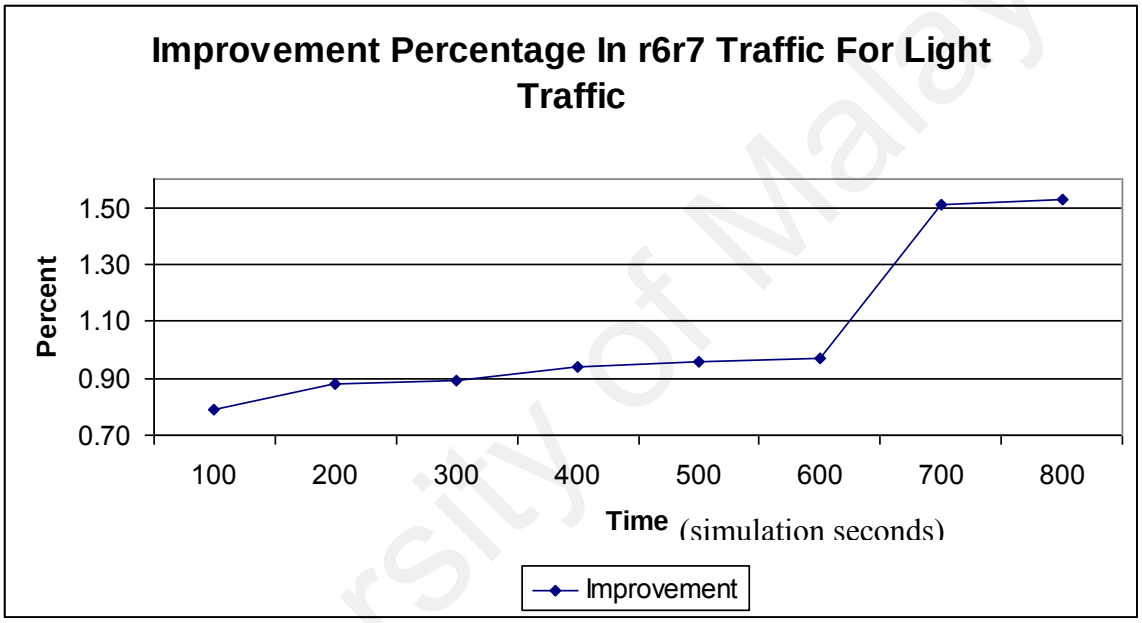


Figure 6.4 Improvement percentage in r6r7 traffic for light traffic

Table 6.4 Improvement percentage values in r6r7 traffic for light traffic

Time (simulation seconds)	100	200	300	400	500	600	700	800
Percent	0.79	0.88	0.89	0.94	0.96	0.97	1.51	1.53

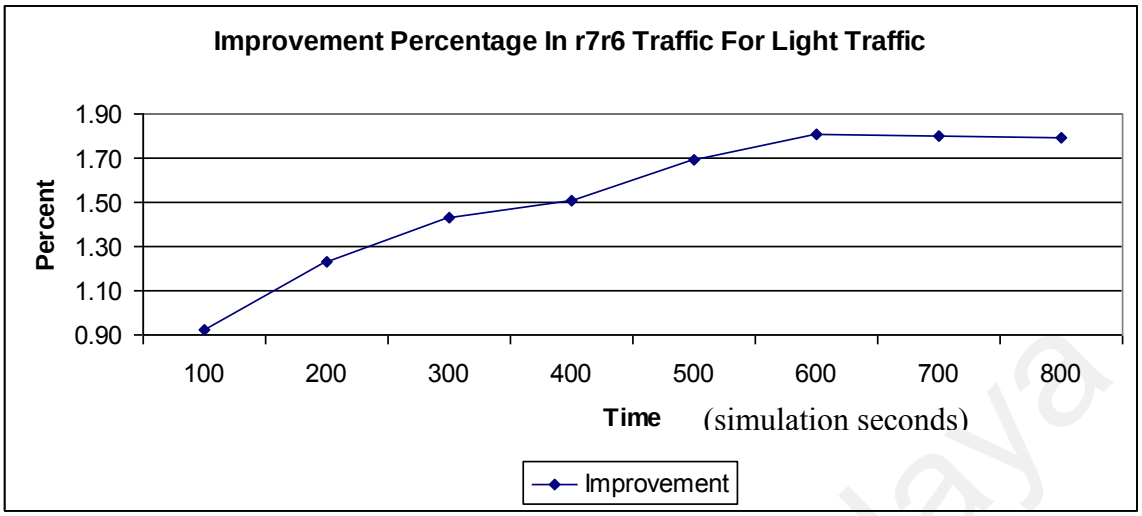


Figure 6.5 Improvement percentage in r7r6 traffic for light traffic

Table 6.5 Improvement percentage values in r7r6 traffic for light traffic

Time (Simulation seconds)	100	200	300	400	500	600	700	800
Percent	0.92	1.23	1.43	1.51	1.69	1.80	1.80	1.79

The total throughputs of packets are the amount of data packets and acknowledgement packets successfully survived through the bottleneck link. The packets distributions between codepoint 20 and 21 for each direction of the bottleneck link are shown in the figure 6.6 and figure 6.7 below. Table 6.2 summarizes the average total packets results for both codepoints.

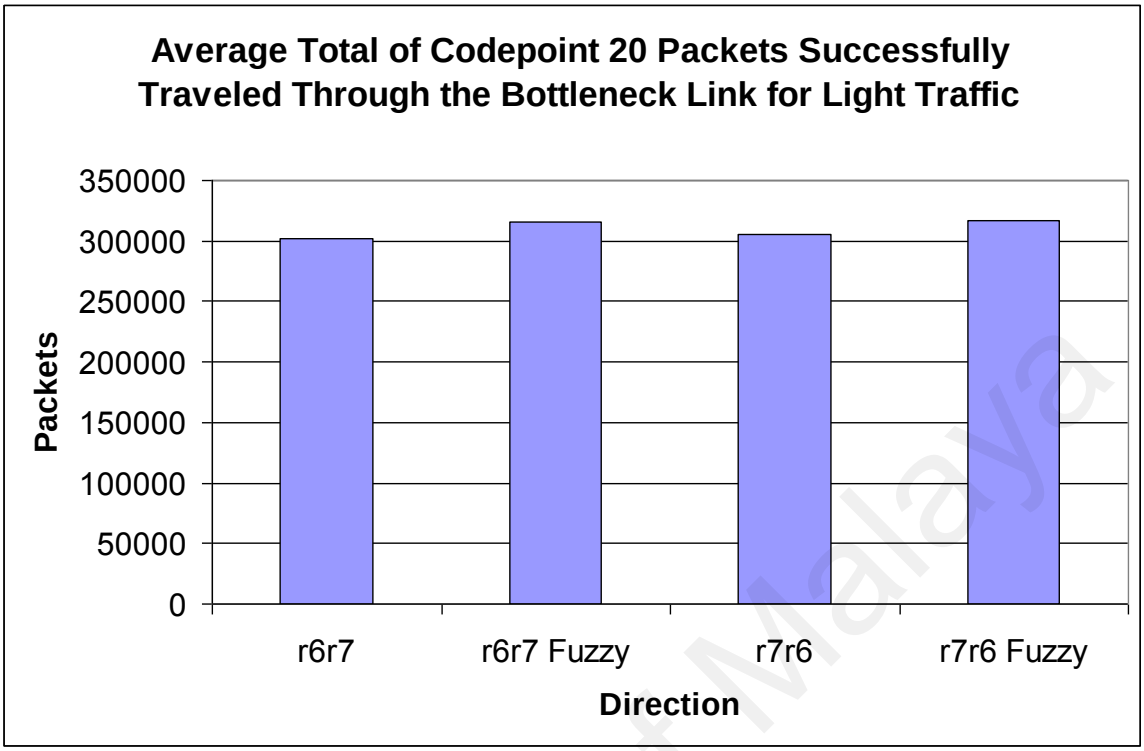


Figure 6.6 Average total of codepoint 20 packets successfully traveled through the bottleneck link for light traffic

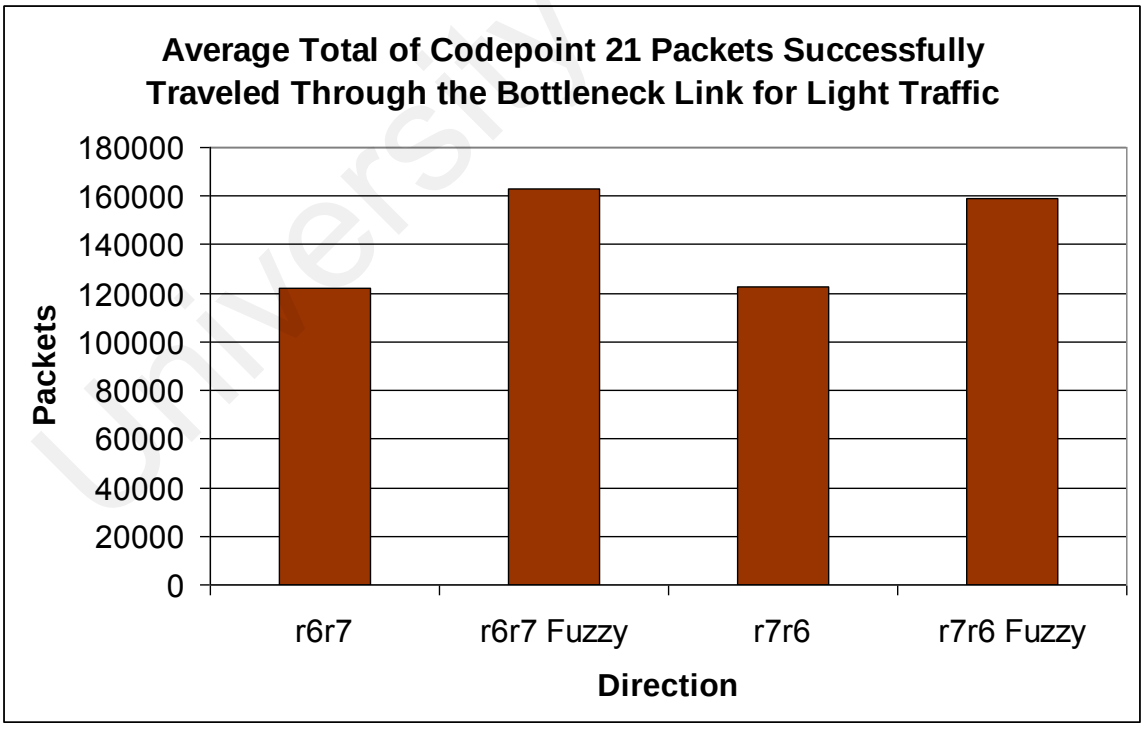


Figure 6.7 Average total of codepoint 21 packets successfully traveled through the bottleneck link for light traffic

Table 6.6 Average packets distributions of codepoint 20 and 21 successfully traveled through the bottleneck link for light traffic

Codepoint	r6r7	r6r7 Fuzzy	r7r6	r7r6 Fuzzy
20	301911	315623	305062	316354
21	122022	162745	1223861	159094

From the graph above, it can be concluded that, with fuzzy logic implementation, total packets with codepoint 20 and 21, which had successfully traveled through the bottleneck link improved.

6.4 Heavy Traffic Simulations Results Analysis

In the heavy traffic simulations, the fuzzy logic implementation does improve the total throughput. The average percentage of amount of packets successfully traveled through the bottleneck link is shown in Figure 6.8 and Figure 6.9 below.

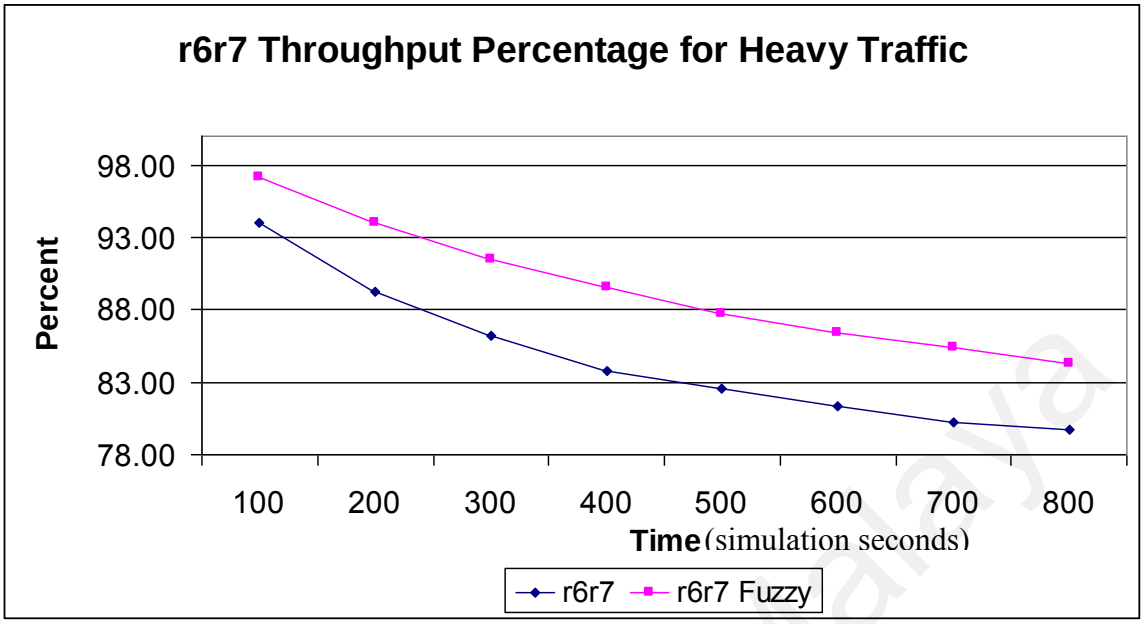


Figure 6.8 r6r7 throughput percentage for heavy traffic

Table 6.7 r6r7 throughput percentage values for heavy traffic

	Time (Simulation seconds)							
	100	200	300	400	500	600	700	800
r7r6 (in percent)	94.01	89.26	86.18	83.81	82.52	81.32	80.20	79.76
r7r6 Fuzzy (in percent)	97.13	93.99	91.52	89.54	87.72	86.40	85.36	84.33

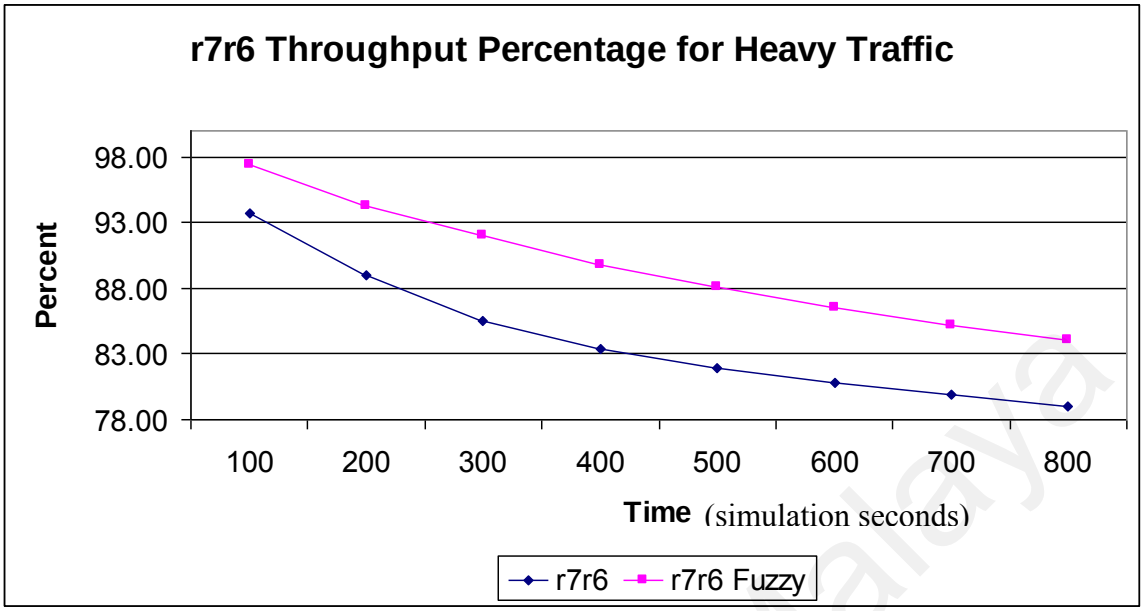


Figure 6.9 r7r6 throughput percentage for heavy traffic

Table 6.8 r7r6 throughput percentage values for heavy traffic

	Time (Simulation seconds)							
	100	200	300	400	500	600	700	800
r7r6 (in percent)	93.76	88.96	85.52	83.37	81.90	80.82	79.94	78.96
r7r6 Fuzzy (in percent)	97.38	94.25	91.98	89.82	88.07	86.58	85.16	94.01

The line above the lowest line in both line graphs shows that the fuzzy implementation does improve the average percentage of throughput of packets. Improvement can be seen in between 3.12% till 6.46% throughout 800 seconds of simulation. The graphs below show the average improvement percentage of light traffic simulations.

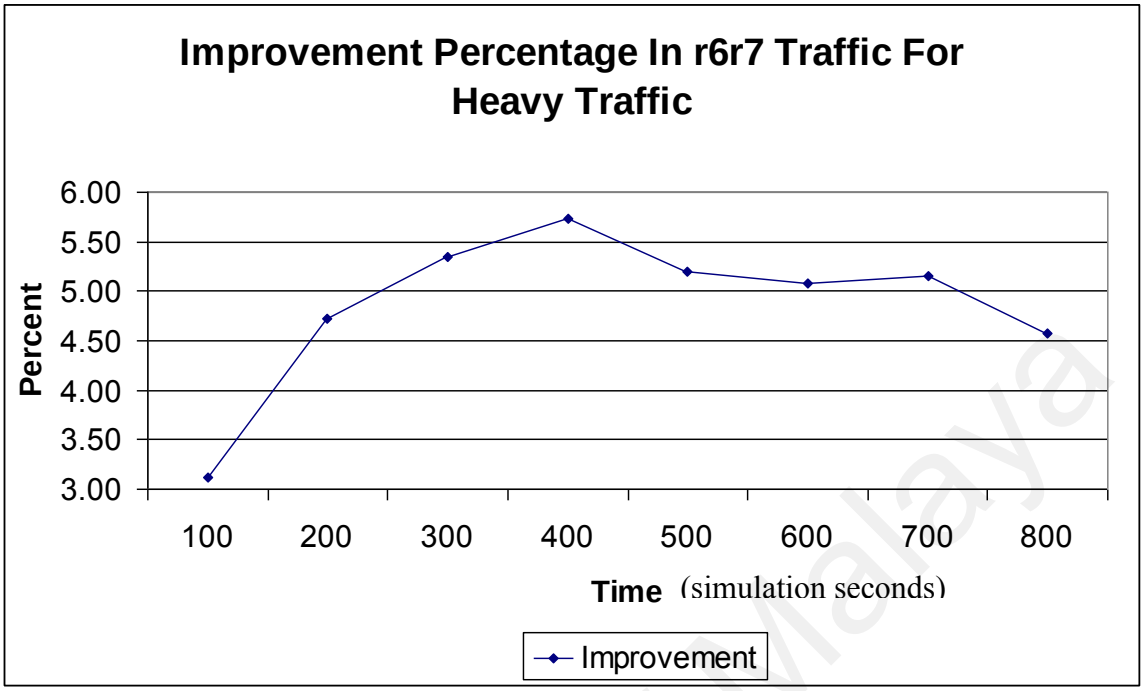


Figure 6.10 Improvement percentage in r6r7 traffic for heavy traffic

Table 6.9 Improvement percentage values in r6r7 traffic for heavy traffic

Time (Simulation seconds)	100	200	300	400	500	600	700	800
Percent	3.12	4.73	5.34	5.73	5.20	5.08	5.16	4.57

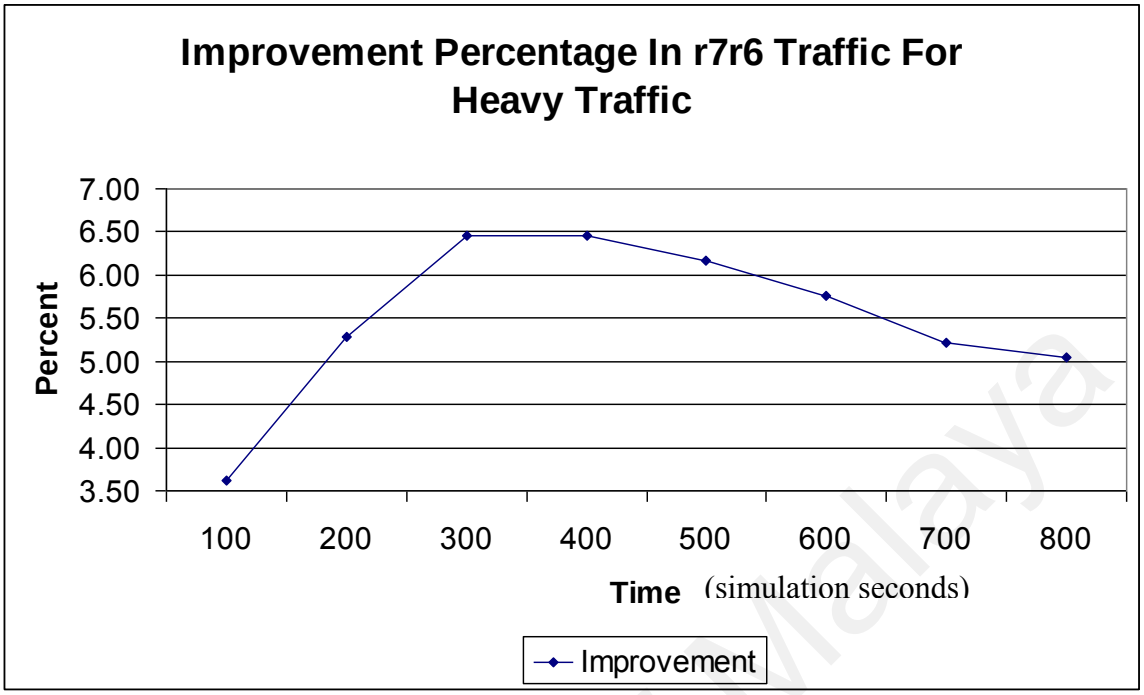


Figure 6.11 Improvement percentage in r7r6 traffic for heavy traffic

Table 6.10 Improvement percentage values in r7r6 traffic for heavy traffic

Time (Simulation seconds)	100	200	300	400	500	600	700	800
Percent	3.62	5.29	6.46	6.45	6.17	5.76	5.22	5.05

The line graphs above (Figure 6.10 and Figure 6.11) shows that the improvement percentage decreases by time. This is caused by the amount of packets trying to travel through the bottleneck link increases and it is way too much from what the buffer in the bottleneck link could handle.

The total packets throughputs of packets are the amount of data packets and acknowledgement packets successfully survived through the bottleneck link. The packets distributions between codepoint 20 and 21 for each direction of the bottleneck link are

shown in the figure 6.12 and figure 6.13 below. Table 6.3 summarizes the average total packets results for both codepoints.

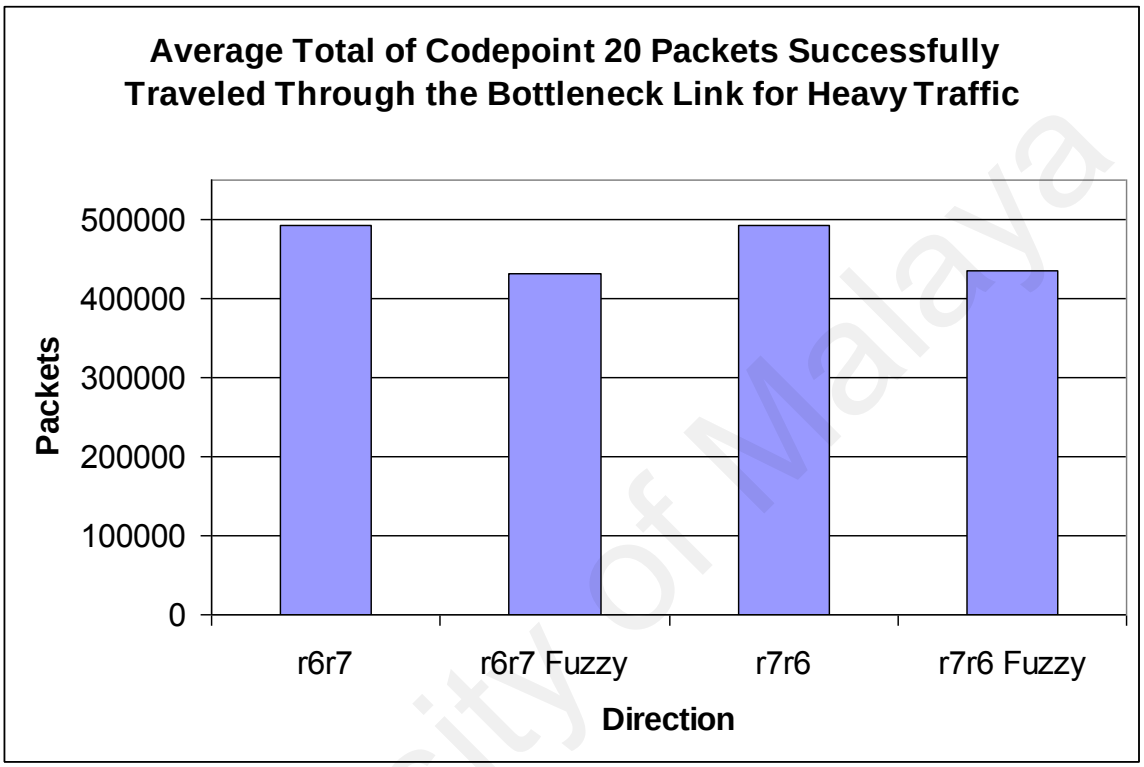


Figure 6.12 Average total of codepoint 20 packets successfully traveled through the bottleneck link for heavy traffic

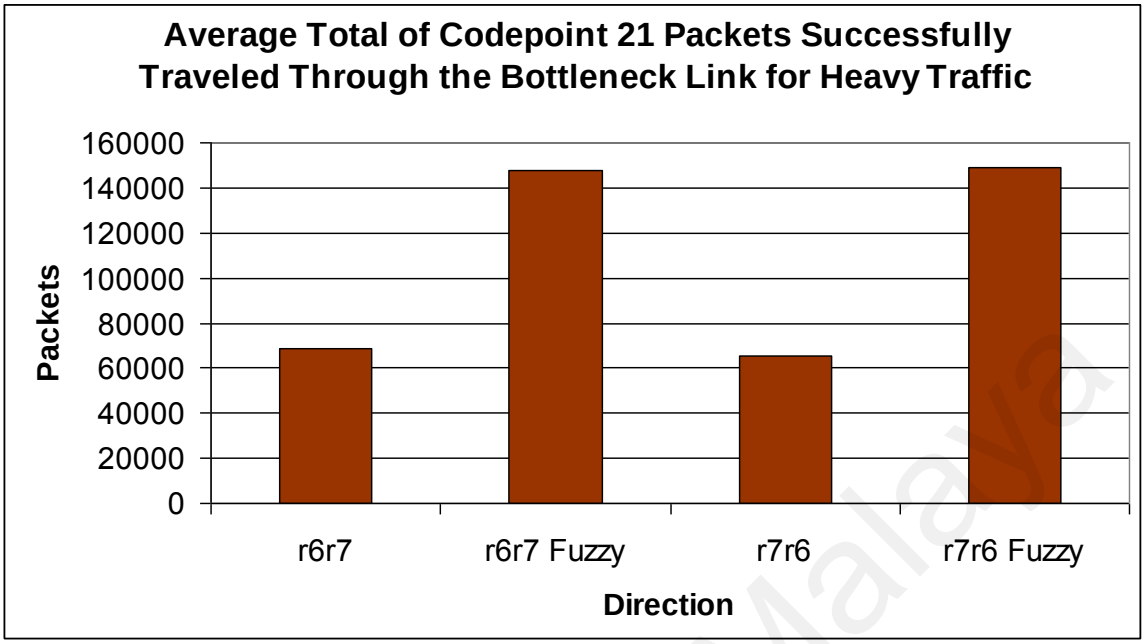


Figure 6.13 Average total of codepoint 21 packets successfully traveled through the bottleneck link for heavy traffic

Table 6.11 Average packets distributions of codepoint 20 and 21 successfully traveled through the bottleneck link for heavy traffic

Codepoint	r6r7	r6r7 Fuzzy	r7r6	r7r6 Fuzzy
20	491941	431260	492692	435888
21	68835	147575	65339	148692

By referring to figure 6.13, for heavy traffic network, the fuzzy logic implementation improves the packets with codepoint 21 throughput by 18% at least compared to the non-fuzzy logic implementation. A huge increment in the throughput for packets with codepoint 21 could be seen by an increment of 45% throughout the 800 seconds of simulations.

In figure 6.12, slight decrement of packets with codepoint 20 throughput by 12.2 % the most throughout 800 seconds of simulations could be witness. Although there is a slight decrement, the fuzzy logic implementation managed to maintain an average total percentage of throughputs for packets with codepoint 20 at a minimum level of 87%.

In a heavy traffic situation, the fuzzy logic implementation is fairer towards packets with codepoint 21 compare to the non-fuzzy implementation. In the normal situation with out fuzzy logic implementation, the queue management model will actually ill treat packets with lower precedence. Packets will higher precedence will gain too much priority and suppress packets with lower precedence from getting through the bottleneck link.

Chapter 7: Conclusion and Future Work

The high demanding state of the Internet requires high bandwidth and quality of service. Several protocols have been developed such as DiffServ to meet this demand. Although DiffServ provide better quality of service if compared to ISA, the increased use of time sensitive data application has set off the need for designing better queue management model to be incorporated into DiffServ domain.

Commonly queue management uses RED to manage the queue occupancy and avoid buffer overflow problem. Although RED can help in queue management, but it is still not up to expectation as the Internet traffic has evolved into a very dynamic form. The amount of data pumped into the network varies from each source and the number of packets in the network at a given moment is unpredictable. Thus the existing DiffServ with RED is not proficient enough to meet this challenging network condition. Therefore the idea of producing a dynamic queue management (FuzAQM) for DiffServ by incorporating fuzzy logic into the architecture has been laid.

This thesis achieved its objectives in creating a dynamic queue management model for DiffServ network using fuzzy logic and to study the behaviors and response of the FuzAQM under different parameters and traffic load. Besides than that, the differences between different RED parameters settings is also been studied and IP QoS models and focus mainly on DiffServ has been explored. This thesis also successfully created a

DiffServ simulation environment for testing the FuzAQM and indirectly effectiveness of the FuzAQM has been evaluated.

The DiffServ environment created to test the FuzAQM has been limited to DiffServ AF PHB. The EF PHB is been excluded because there is no further service differentiation in EF class and it is considered as premium service which will get all resources and treatment it required. BE PHB is also been excluded because it is a best-effort service. Packets with BE codepoint will be discarded first once congestion avoidance take place or congestion happens. But in AF class itself, it has 4 different types of services, namely AF1, AF2, AF3 and AF4. Between these 4 classes there are different preferential and treatment among them. AF1 is considered more superior than AF2 and so on. In depth same AF class, will be further differentiated using drop precedence.

This thesis is significant in its effort to improve the total throughput of traffic especially for packets with higher priority without choking packets with lower priority. Last but not the least, this thesis also successfully turns the queue management model into a responsive queue management model.

7.1 Summary of Work Done

This thesis started off with literature review on QoS, DiffServ, congestion management and fuzzy logic concept. From these reviews, a dynamic queue management model has been proposed based on RED. This dynamic queue management model is known as Active Queue Management for AF Traffic in DiffServ Network (FuzAQM). FuzAQM

manipulate the RED parameters settings based on output from a fuzzy logic inference process. The inference process will output a value that measures the congestion level at the bottleneck link every time a packet arrives. Based on the value that represents the level of the congestion at bottleneck link, appropriate RED parameters would be reset to handle the current situation. By doing this, now the queue management model has now become more responsive to the congestion level.

ns2 network simulator has been chosen for the development and testing FuzAQM. The codes for FuzAQM are written in C++ language located in dsredq.cc and dsredq.h files. The simulation script is written in Otcl which are divided into diffnet.tcl, peer_setup.tcl, 2q2p.tcl, topology.tcl and monitoring.tcl. diffnet.tcl is the main simulation file which will call the rest of the tcl files.

The topology used for the simulations is a dumbbell topology. There are two types of simulated scenarios which are, simulation with light traffic and heavy traffic. A series of simulations has been carried out for each type of scenarios. From each simulation done, packets statistics are recorded at 100 seconds interval.

Based on the statistics results recorded, then the performance of the FuzAQM is verified. To verify the FuzAQM, the network performance for two types of scenarios, one with fuzzy logic implementation and one without, are compared. The main measurement item is the throughput of packets at the bottleneck link.

From the simulation results, it is proven that FuzAQM improves the total throughput by 0.79% to 6.46% depending on the traffic load and maintaining fair treatment for lower priority packets while maintaining at least 87% of throughput for higher priority packets in the network.

7.2 Summary of Contributions

This thesis centers its research in QoS and focus mainly in DiffServ. Works done includes literature review on the QoS and DiffServ motivates the creation a fuzzified active queue management for DiffServ traffic. In this ever-demanding networking world, there is a paradigm shift in its research interest. There is now an urge to search for more advanced and capable algorithm or model, which can cater this need. Thus with these factors as the motivation or a push, a dynamic and responsive model is now getting the spotlight in networking research.

The proposing of FuzAQM model is one of the steps taken in answering the challenge of searching for a better model. Appropriate network simulator has been chosen to help evaluate the effectiveness of the proposed FuzAQM model. And the results from the simulations do prove that the proposed FuzAQM model could actually improved the network performances although the algorithm used is quite basic and simple.

7.3 Objectives Achieved

The simulations analysis has proven that the proposed FuzAQM model achieved its objectives. The main objective achieved was the successfulness of creating active queue management using fuzzy logic. Besides than that, from the fine tuning process of the RED parameters, the impact on setting different RED parameters on the network throughput also been studied.

The simulation environment created is also proven to be sufficient to test out the FuzAQM model. From the initial stage, the environment set was not harsh enough to put the proposed FuzAQM model into stress and test. But in later stage, the environment was successfully set to at least put the proposed FuzAQM model into stress and the effectiveness of the proposed FuzAQM model starts to reflect its effectiveness.

Towards the end of this research, it is clear that an active queue management for DiffServ traffic works better than classic type of non-responsive queue management model. So it can be summarized here that active queue management for AF class traffic DiffServ will be one of the key answer to the research for a better model.

7.4 Future Research Suggestions

The works done and presented in this thesis are actually the ignition of a more serious and advance research works. Due to time and budget constraints, several simplifications and assumptions need to be done. There are several future enhancements and works could be carried out. These enhancements and works include expanding the research to include

various types of packets metering scheme such as time sliding window with 3 color marking (RFC2859), token bucket (Brown03), leaky bucket (Brown03), single rate with 3 color marking (srTCM) (RFC2697) or two rate with 3 color marking (trTCM) (Aboul03, RFC2698) scheme.

This thesis focuses on AF packets type with two drop precedence. Future works could be done to include more than 2 drop precedence and AF types. Future research could be done to include all DiffServ service types and evaluate the way the FuzAQM model could be enhance further to handle a more extended situation.

The fuzzy logic part in this thesis eventually guides the settings of RED and drop parameters. Further work could be done where the output of the fuzzy logic engine is used to guide or change the settings of a wider spectrum of non-constant variables located in any part in the DiffServ network. For example, it could be used to reset the parameters in the token bucket (Sahu00) scheme to regulate the metering process or maybe used to change the type of MRED modes such as RIO (Clark98), Adaptive RIO (Cartas04) or Fair RED (Lin97) in conjunction with the network situation.

References

(Aboul03)	Aboul-Magd, O., and Rabie, S., *Two Rate Three Color Marker for Differentiated Services*, Internet-Draft, October, 2003.
(Asai95)	Asai, K. Ed., (1995) *Fuzzy Systems for Information Processing*, Tokyo, Ohmsha.
(Brigette03)	Krantz B. *A "Crisp" Introduction to Fuzzy Logic* [Internet] Boulder, Colorado University. Available from: < http://www-ugrad.cs.colorado.edu/~cs3202/papers/Brigette_Krantz.html> [Accessed 15 May 2003]
(Brown03)	Brown, M.A. (2003) *Traffic Control - Next Generation* [Internet]. Available from: < http://linux-ip.net/gl/tcng/node1.html> [Accessed 4 July 2004]
(Cartas04)	Mexican International Conference in Computer Science (ENC'04), 5th. Colima, Mexico, (September 2004), *A Fairness Study of the Adaptive RIO Active Queue Management Algorithm.* Cartas, R. et al. IEEE.
(Chrysostomou03)	Chrysostomou, C. et al. (2003) *Fuzzy Explicit Marking for Congestion Control in Differentiated Services Networks*. IN**:** Eighth IEEE International Symposium on Computers and Communications June 30 - July 03, Kemer-Antalya, Turkey, IEEE. pp. 312-319.
(Cisco00)	(December 2000) *Internetworking Technologies Handbook: Quality of Service Networking*, 3rd edition, I.N. USA, Cisco Press
(Clark98)	Clark, D., and W. Fang (1998) Explicit Allocation of Best Effort Packet Delivery Service. *IEEE/ACM Trans. On Networking* August 6(4).

(Cunha95)	Cunha, C.R., Bestavros, A. and Crovella, M.E. (1995) *Characteristics of WWW Client-based Traces* [Internet] Cummington St, Boston University. Available from: < http://cs-www.bu.edu/faculty/crovella/paper-archive/TR-95-010/paper.html> [Accessed 26 February 2003]
(DiFata03)	Di Fata, G. et al. (2003) A Genetic Algorithm for the Design of a Fuzzy Controller for Active Queue Management *IEEE Transactions on Systems, Man, and Cybernetics*. 33(3) August. pp. 313-324.
(Fishwick96)	Fishwick, P.A. (1996) Computer Simulation: The Art and Science of Digital World Construction. *IEEE Pontentials*. February/March, pp. 24-27.
(Floyd93)	Floyd, S., and Jacobson, (1993) V.,Random Early Detection gateways for congestion avoidance *IEEE/ACM Trans. on Networking*, 1(4) August, pp. 397-413.
(INSANE96)	INSANE Users Manual, Comp. Sc. Division, UC Berkeley, May 1996.
(Jacobson88)	Jacobson, V. (1998) *Congestion avoidance and control.* IN: Proceedings of SIGCOMM '88, August, Stanford, Ca, ACM.
(Karthik04)	Karthik, S., Venkatesh, C., and Natarajan, A.M. (2004) *Congestion control in ATM networks using fuzzy logic* IN: Proceedings of the18th International Parallel and Distributed Processing Symposium, 26-30 April, New Mexico, USA, IEEE Computer Society Press. pp.162.
(L'Ecuyer99)	L'Ecuyer, P., (1999) Good parameters and implementations for combined multiple recursive random number generators. *Operation Research*, 47(1) January, pp.159-164.
(L'Ecuyer01)	L'Ecuyer, P., (2001) *Software for uniform random number generation: Distinguish the good and the bad.* IN: Proceedings of the 2001 Winter Simulation Conference, December, pp. 95–105.

(L'Ecuyer02)	L'Ecuyer, P. et al., (2002) An object-oriented random number package with many long streams and substreams. *Operation Research*, 50(6) November, pp. 1073-1075.
(Liang03)	Liang, J., Arvanitis, T.N., and Woolley, S.I. (2003) Fair weighted round robin scheduling scheme for DiffServ networks. *Electronics Letters* 39(3) February, pp. 333-335.
(Lin97)	Lin, D., and Morris, R. (1997) *Dynamics of Random Early Detection*. IN: Proceedings of ACM SIGCOMM, September. Stanford, Ca, ACM, pp. 127-137.
(Luoma00)	Luoma, M..(2000) *Simulation Studies of Differentiated Services Networks* Licentiate thesis, Helsinki University of Technology.
(Maisie96)	Maisie User Manual Release 2.2, UCLA Parallel Computing Lab, Dec. January, 1996.
(Nichols97)	Nichols, K., Jacobson, V., and Zhang, L., "A Two-bit Differentiated Services Architecture for the Internet", Internet-Draft, December 1997.
(NS03)	The ns Manual, The VINT Project, December 2003.
(OPNET97)	OPNET Modeling Manual Vol. 1. OPNET Version 3.5. MIL3 Inc. 1997.
(PARSEC98)	PARSEC User Manual Release 1.1, UCLA Parallel Computing Lab, August 1998.
(REAL97)	REAL 5.0 User Manual, Cornell University, August 1997.
(RFC793)	Information Sciences Institute, University of Southern California *TRANSMISSION CONTROL PROTOCOL*, RFC 793, September 1981.
(RFC896)	Nagle, J., *Congestion Control in IP/TCP*, RFC 896, January 1984.
(RFC1122)	Braden, R., Ed., *Requirements for Internet Hosts --Communication Layers*, RFC 1122, October 1989.

(RFC2205)	Braden, R., Ed. et al., *Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification*, RFC 2205, September 1997.
(RFC2309)	Braden, B., et al., *Recommendations on Queue Management and Congestion Avoidance in the Internet*, RFC 2309, April 1998.
(RFC2415)	Poduri, K., and Nichols, K., *Simulation Studies of Increased Initial TCP Window Size*, RFC 2415, September 1998.
(RFC2474)	Nichols, K., Blake, S., Baker, F., and Black, D., *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*, RFC 2474, December 1998.
(RFC2475)	Blake, S. et al. *An Architecture for Differentiated Services,* RFC 2475, December 1998.
(RFC2597)	Heinanen, J., et al., *Assured Forwarding PHB Group*, RFC 2597, June 1999.
(RFC2598)	Jacobson, V., Nichols, K., and Poduri, K., *An Expedited Forwarding PHB*, RFC2598, June 1999.
(RFC2697)	Heinanen, J., Finland, T., and Guerin, R. *A Single Rate Three Color Marker*, RFC 2697, September 1999.
(RFC2698)	Heinanen, J., Finland, T., and Guerin, R., *A Two Rate Three Color Marker*, RFC 2698, September 1999.
(RFC2859)	Fang, W., Seddigh, N., Nandy B., *A Time Sliding Window Three Colour Marker (TSWTCM)*, RFC 2859, June 2000.
(Sahu00)	Sahu, S. et al. (2000) *On Achievable Service Differentiation with Token Bucket Marking for TCP*. IN: Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, 18 – 21 June, California, United States, ACM Press. pp. 23-33.
(Stallings02)	Stallings, W. (2002) *High-Speed Networks and Internets, Performance and Quality of Service*. 2nd ed., Upper Saddle River, N.J., Prentice Hall.

(Yanfei03)	Yanfei, F., Fengyuan, R., and Chuang, L. (2003) *Design of an active queue management algorithm based fuzzy logic decision.* IN: Communication Technology Proceedings, 2003. International Conference on , 9-11 April, China, IEEE, pp. 286 – 289.
(Zhang01)	Zhang, R. and Ma, J. (2001) *Congestion control using fuzzy logic in differentiated services networks.* IN: Fourth International Conference, Proceedings of the Computational Intelligence and Multimedia Applications, 30 October-1 November 2001, Yokusika City Japan, IEEE. pp. 288 – 292.