Quality of Service for Multimedia and Control System Applications in Mobile Ad-hoc Network

Oche Alexander Egaji

BSC in Electrical\Electronic Engineering, 2008

MSC in Communication Engineering, 2009

A thesis submitted to the Faculty of Computing, Engineering and Sciences, Staffordshire University, in fulfilments of the requirements for the degree of a Ph.D.

February 2015

Abstract

A Mobile Ad-Hoc Network (MANET) is a collection of randomly distributed infrastructure-less mobile nodes that form a wireless network. These Mobile nodes have the capability to act as a host or relay. As a host, the mobile nodes can be the source and/or destination of traffic, and when acting as a relay, they can be an intermediate node that forwards the traffic to its destination. Some of the challenges of a MANET include the dynamic network topology, device discovery, power constraints, wireless channel conditions and limited network resources. These challenges degrade the network performance and thus affect the network stability and robustness. Therefore, it is difficult for a MANET to attain the Quality of Service (QoS) of a wired network. This thesis aims to address the problem of the limited wireless network resources by proposing two adaptive scheduling algorithms that can adapt in real-time to the changes in the network.

To achieve the aim; this thesis first analyses the behaviour of various application profiles in a queue. It models Voice, Email, and Internet Browsing traffic (by specifying packet sizes, and inter-arrival rates based on various distributions) separately and then simultaneously in a common network for uncongested and congested conditions, after which scheduling is applied in order to improve the overall network performance. The Voice traffic profile is then added to the UDP/IP protocol stack and the network performance is compared to a simple node without the UDP/IP protocol stack. A realistic wireless propagation model for the simulation is developed from a point-to-point open-field outdoor experiment.

This thesis proposes two adaptive priority fuzzy based scheduler for a MANET, the priority of packets in the queue are determined based on the real-time available network resources. The methodology for transmitting a live-feed video stream over OPNET to validate the scheduler is also presented. An interface between the simulation and hardware is created to send real-time video traffic through the simulation network.

This thesis concludes by showing that the performance of a MANET network can be improved by applying an adaptive scheduler.

Acknowledgements

I would like to express my gratitude to my Principal supervisor Dr Alison Griffiths for her valuable patience guidance and strong support during the course of this challenging research. I would also like to give a special thanks to my second supervisor Dr. Mohammad S. Hasan for his advice and valuable input towards the development of this thesis and patiently reading my reports. A special thanks to Professor Hongnian Yu, my third supervisor, for his understanding, support, advice and many organised workshop and seminars during the course of this research.

I would like to acknowledge the Faculty of Computing, Engineering, and Science for their sponsorship and financial support during the course of this Ph.D.

A special thanks to friends who have supported me with my outdoor work, which includes my experiment. They have done this diligently without been deterred by the challenging weather conditions at that time.

My thanks go to my colleagues in Mobile Computing research group and the Division of Computing Systems at Staffordshire University with special reference to Patison Palee and Keattikorn Samarnggoon for their interesting comments, encouragement, support, and suggestions.

I also want to thank the OPNET support team for their endless emails spent helping and supporting my many queries.

Finally, I want to express special thanks to my family, for their financial and emotional support during the course of this research and for always being there for me.

Table of Contents

List of Figures

List of Tables

Glossary

Term	**Description**
ABR	Associativity-Based routing
ACL	Asynchronous Connection-Less
AP	Access Points
AODV	Ad-hoc On-Demand Distance Vector
CBR	Constant Bit Rate
CSMA	Carrier Sense Multiple Access
DARPA	Advanced Research Projects Agency
DoD	Department of Defence
DSDV	Destination Sequenced Distance vector routing
DSR	Dynamic Source Routing
DSSS	Direct Sequence Spread Spectrum
ELB ACTD	Extending the Littoral Battle-space Advance Concept Technology Demonstration
ESTREL	Energy Saving on Transmission and Reception based Enhancement Layers
FFD	Full Function Device
FHSS	Frequency Hopping Spread Spectrum
FIFO	First in, First out
FUNET	Finish University and Research Network
GFSK	Gaussian Frequency Shift Keying
IEEE	Institute of Electrical and Electronic Engineers

IGRP	Interior Gateway Routing Protocol
IP	Internet Protocol
IPV6	Internet Protocol version 6
ISM	Industrial, Scientific and Medical
ITU	International Telecommunication Union
LDPC	Low-Density Parity Check
LLC	Logical Link Control
LMR	Lightweight Mobile routing protocol
LOS	Line Of Sight
LPR	Low-cost Packet Radio
MAC	Medium Access Control
MANET	Mobile Ad-hoc Network
MDC	Multiple Descriptions Coding
MIMO	Multiple-Input and Multiple-Output
NCS	Network Control Systems
OFDM	Orthogonal Frequency Division Multiplexer
OPNET	Optimised Network Engineering Tool
OSI	Open Systems Interconnection
OSPF	Open Shorted Path First
PDA	Personal Digital Assistants
P-K	Pollaczek-Khinchin
Pkts	Packets
PQ	Priority Queuing
PRNet	Packet Radio Network
PSMP	Power Save Multi-Poll
QoS	Quality of Service

RCPLF	Routing Control Packets Limited Forwarding
RF	Radio Frequency
RFD	Reduced Function Device
RIP	Routing Information Protocol
RREP	Route Reply Packet
RREQ	Route REQuest
RRS	Round Robin Scheduling
SCO	Synchronous Connection-Oriented
SISO	Single-Input-Single-Output
SURAN	Survivable Radio Network
Src	Source
TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
UDP	User Datagram Protocol
VBR	Variable Bit Rate
VoWLAN	Voice over WLAN
WDQ	Weighted-Distance Scheduling
WFQ	Weighted Fair Queuing
WLANs	Wireless LANs
WNCS	Wireless Network Control Systems

List of Symbols

Symbols	**Description**
B	Bandwidth
C_{queue}	Queue Capacity
C_{link}	Wireless Link Capacity
k	The shape parameter
M	Maximum burst size
n	Number of sources
$N(t)$	Number of packets in the system at time, *t*.
N_Q	The expected number of packets in the queue
N	The number of packets in the system
p	Normal Pareto distribution
$p_n(t)$	The probability that ‘n’ number of packets in the queue at time t
r	The scale parameter (Pareto Distribution)
T_i	Time spent in the system by the i^{th} arriving packet
T_t	The time average of packet delay up to time ‘t’
$\overline{T}_n$	The average delay of the nth packet
T_{sp}	Inter-arrival time
T, τ_{queue}	The average queuing delay
W, $\tau_{waiting}$	The average queue waiting time
$\overline{X} = E\{X\}$	The average service time
$\alpha(t)$	Number of packets that arrived in the interval [0,*t*]
λ_t	The time average or arrival rate
μ	The service rate
∂	Time interval
$\tau_{latency}$	Delay
τ_{tx}	The transmission delay
τ_{prop}	The propagation delay
τ_{proc}	The processing delay
δ	Rate of distribution at $x_0 > 0$
φ	The probability that $x > M$
ω	location parameter
β	scale parameter (Cauchy distribution)

Chapter 1 Introduction

This chapter provides an overview of this thesis; it includes the background and motivation, the aims and objectives, the research challenges, deliverables and the methodology; and finally, a summary of the thesis structure is also provided.

1.1. Background and Motivation

The recent trend of modern commercial and industrial systems is to integrate computing, communication and control into different levels of factory operations and information processes [1]. A Mobile Ad hoc Network (MANET) consists of infrastructure-less nodes that can communicate among themselves to form a self-contained network as shown in Figure 1-1. These nodes are equipped with a microprocessor, radio transceiver and a battery based power supply. They are deployed in an unstructured manner and can communicate with each other through a wireless link, forming an ad hoc network without the requirement of a central control system [2]. A MANET can reconfigure itself when there is a link failure without any external assistance. A pair of mobile nodes can only communicate when they are within radio range of each other. Mobile nodes can act as a host, relay, or both. As a host, a node can function either as a source, a destination or both, and as a relay; it functions as an intermediate node to forward information to destination nodes. This phenomenon describes the multi-hop nature of MANET.

Most MANET operates within the Industrial, Scientific and Medical (ISM) band, and mobile nodes are required to use their available power efficiently because of their limited battery life. The ISM band is a license free international radio band reserved for radio frequencies for industrial, medical and scientific purposes. A list of appliances that operate on similar radio frequencies is the microwave oven, medical diathermy machines and radio frequency process heating. In recent years, there have been a significant development in channel capacity and bandwidth utilisation. However, a number of factors that include multiple node access, signal fading, multipath, noise, and interference affect the quality of the signal received by the destination or relay.

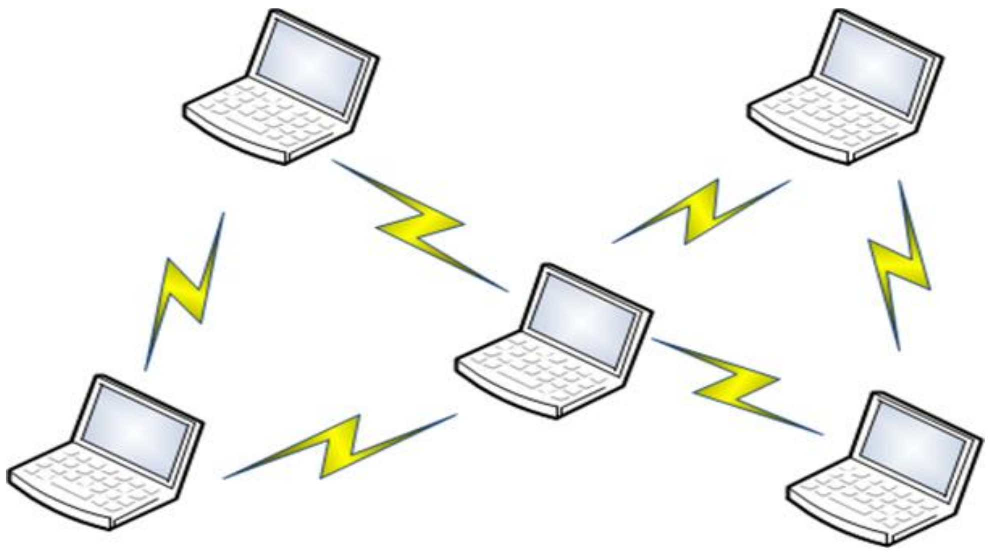


Figure 1-1: Sample MANET

MANET is most suited for an application that requires mobile nodes, where the use of wires will increase their complexity. Examples of such applications are tactical networks, e.g. Military communication and automated battlefields. Emergency services, e.g. search and rescue operations, disaster recovery, supporting doctors and nurses in hospital and supersession of infrastructure in the case of environmental disaster. It is also useful for education, e.g. University and campus settings, virtual classrooms, ad-hoc communication during meetings or lectures. The possibility of deploying a MANET for these applications as well as the lack of preinstalled wired infrastructures justifies the development of such networks. Figure 1-2 shows some of the applications of MANET.

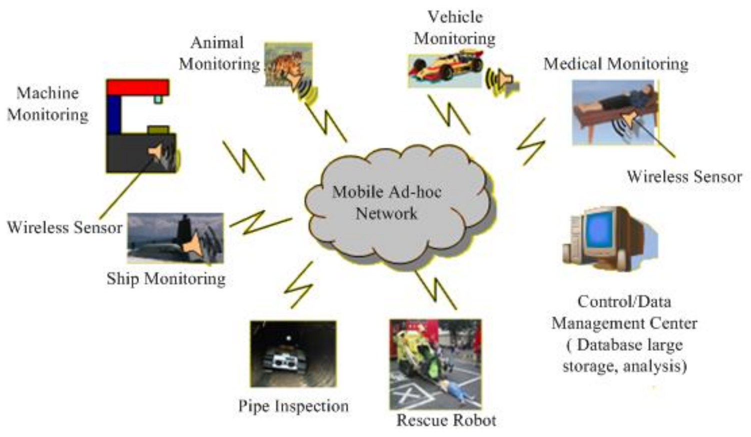


Figure 1-2: MANET Applications

MANET has been proposed for use in other realms such as vehicular communication popularly known as Vehicular Ad-hoc NETwork (VANET) [3] and also been implemented for environmental monitoring. Paper [4] designed and implemented a MANET within a forest to monitor CO_2 movement. It used sensors to estimate the impact of forest-atmosphere CO2 accurately. The PermaSense Project [5] used MANET to collect permafrost related parameters, to understand better, and improve the relationship between climate change and rock fall. Paper [6] implemented wireless sensor network to monitor the condition of a grain silo. The data collected provides the means for effective process control that improves the overall system efficiency by reducing raw material, energy and wastage.

Civil engineering has also benefited from MANET. Paper [7] designed and implemented MANET on the 4200ft long main span and the south tower of the Golden Gate Bridge. It measured the ambient structural vibration reliably without necessarily interfering with the operation of the bridge. Paper [8] implemented an underground structure monitoring system using MANET to ensure a safer mining environment.

Due to the wide range of potential applications of MANET, when fully developed it has the potential to replace some infrastructural networks in the future. For this reason, there is a need to consider both real-time and non-real-time applications in addition to the control applications for performance analysis. These real-time applications are delay sensitive applications. In control systems or telemetry monitoring, ‘real-time’ refers to systems with a strict deadline and it can be catastrophic when these deadlines are not met. However, in multimedia systems, even though the systems have real-time requirements, they can tolerate a certain percentage of loss without necessarily losing meaning or the details of the information. The delivery time can be relaxed since humans are more tolerant to delay as compared to machines. All applications are delay sensitive to some extent, however, the end user can notice real-time multimedia applications that exceed an end-to-end delay bound.

Non-real-time applications are not delay sensitive, but they are sensitive to loss. A small loss can change or corrupt the entire information, in this instance; data integrity is very important.

These real and non-real time applications have Quality of Service (QoS) requirements to guarantee a certain level of acceptable performance. QoS is the collective effect of the service as perceived by the end user. These QoS requirements include delay, loss, jitter and bandwidth [9][10]. The use of MANET for applications such as battlefield, healthcare, and search/rescue; demands a high degree of network predictability.

1.2. Research Challenges

The demands for high QoS have placed a severe constraint in the design of MANET. A MANET is a wireless network, and thus it transmits packets via wireless links. These links are dynamic as compared to wired links because of nodes movements. They are also subject to time and signal attenuation, diffraction, reflection, refraction and interferences and the added problem of limited bandwidth.

The random nature of the node mobility causes frequent route changes, this result in high packet loss, high end-to-end delay and a reduction in the network throughput. The infrastructure-less nature of MANET, in addition to the limited network resources, has made the provision of QoS challenging. Hence, this has increased the need for a QoS solution that is dynamic in nature.

Researchers have focused on using several techniques to improve the QoS in MANET. These techniques include improving or proposing new routing protocols, data coding in the physical layer and link stability (multiple transmitters and receivers). There is little research work on developing a specialised packet scheduler for MANET. Most of the work in MANET relies on the conventional scheduling algorithm designed for a wired network; these algorithms are not suitable for MANET because of its mobile and wireless characteristics.

Hence, using an adaptive scheduling algorithm to determine which queue or packet to serve next should improve the overall network performance. The default scheduling scheme for packets in a MANET is First In, First Out (FIFO).

One of the challenges faced in comprehensively modelling an adaptive scheduling algorithm for traffic in MANET; is that individual packets rather than the entire traffic flow compete for the available network resources. At low load, all packets are forwarded with minimal delay, but as the load increases the delay also increases. Effectively reducing this delay depends on the queuing and scheduling techniques used.

This thesis tackles the problems of limited network resources in MANET by proposing an adaptive scheduling algorithm to improve the network QoS. This algorithm reduces the end-to-end delay, packet loss and also increases the network throughput, thus enhancing the system stability [11][12].

1.3. Aim and Objectives

The aim of this research is to investigate MANET and its research challenges, focusing on the issues of the limited network resources. This thesis aims to develop a mathematical model for real-time traffic e.g. control application and multimedia traffic and also develops an adaptive packet scheduling algorithm to optimise the network performance with increasing network load. The proposed schedulers aim to improve the QoS of MANET, but should be applicable to other wireless networks. The objectives of this research are:

- To investigate MANET, its implementations and challenges.
- To develop models for real and non-real-time applications such as Email, Voice and Internet Browsing in OPNET.
- To develop a queuing delay model for MANET using queuing theory
- To develop a simulation model for real and non-real-time traffic and also investigate its queuing performance under congested and uncongested state using queuing theory.
- To research, analyse and explore the performance of various reactive routing protocols such as Ad hoc On-demand Distance Vector (AODV), Dynamic Source Routing (DSR), Lightweight Mobile routing protocol (LMR) and Associativity-Based Routing (ABR); considering node mobility, packets delivered, jitter, end-to-end delay, and routing load via network simulation. This will help to choose the best routing protocol to use for this thesis.
- To carry out a real-time open-field point-to-point experiment and develop a corresponding simulation model to replicate its channel performance, thus validating the simulation model and the Optimised Network Engineering Tool (OPNET).

- To develop a real-time adaptive packet scheduling algorithm for MANET; this algorithm will optimise network resources to reduce delay, increase throughput and Packet Delivery Ratio (PDR), thus improving the overall system stability.
- To develop a simulation model to analyse the performance (delay and PDR) of the proposed schedulers under various traffic type and network loads.
- To validate the real-time functionality of OPNET by streaming real-time video over a network. This will be achieved by creating an interface between OPNET and an external device (webcam).
- To analyse the performance of the real-time video stream under non-real-time CBR and VBR Background Traffic when scheduling is applied.

1.4. Contribution

The contribution of this thesis is to resolve issues with limited network resources in MANET. This thesis proposed two adaptive fuzzy based schedulers and it develops a real and non-real time traffic model (based on a mathematical formula). An interface between simulation and an external device is created to validate the proposed schedulers. The contributions are broken down into the following stages:

- A detailed analysis and explanation of the advances made e.g. increase of efficiency, reliability, availability, maintainability and performance of MANET - Chapter 2
- A mathematical model for real and non-real-time applications such as Email, Voice and Internet Browsing. Using Cauchy to model Email traffic - Chapter 3
- The development of a queuing delay model for MANET using queuing theory and a detailed simulation analysis of real and non-real-time traffic under the congested and the uncongested condition - Chapter 3
- The development of a realistic simulation model based on an open field experiment, this will provide guidelines for other researchers, to mathematically model a realistic wireless propagation model for a simulation environment - Chapter 3
- The development of an adaptive fuzzy scheduling algorithm to improve the QoS of MANET - Chapter 4

- The development of a real-time video streaming over OPNET, this will interface OPNET with an external hardware device - Chapter 5
- Performance analysis of the proposed schedulers under mixed network traffic, with a live-feed video - Chapter 5

1.5. Research Methodology

The research methodology used is a quantitative approach. The research method consists of a number of sections labelled 1-7 as shown in Figure 1-3. The first section is a literature review presented in Chapter 2. It contains a documented investigation of existing and relevant research work. The second section is presented in section 3.3.1; it contains the design of real and non-real-time traffic; these include Voice, Email and Internet Browsing. The Email is developed with the Cauchy distribution in OPNET. The third section presented in section 3.3.2; it contains the simulation model and performance analysis for staggered and non-staggered start time for traffic under congested and uncongested traffic conditions. These are compared to the theoretical prediction and it formed the baseline simulation model as shown in the fourth stage. The fifth stage is presented in section 3.4; it contains the design and implementation of an outdoor experiment with a corresponding realistic simulation model. These formed the basis for the wireless propagation model used in this thesis. The sixth stage is presented in Chapter 4; it contains the development of an adaptive packet scheduling algorithm along with the performance analysis of various traffic profiles for a single traffic flow under varying level of network load. The seventh and final stage is presented in section 5.4; it contains the design and development of an interface between OPNET and an external hardware device (webcam) for real-time video streaming. The performance of the real-time video stream under various background loads when scheduling is applied is also provided.

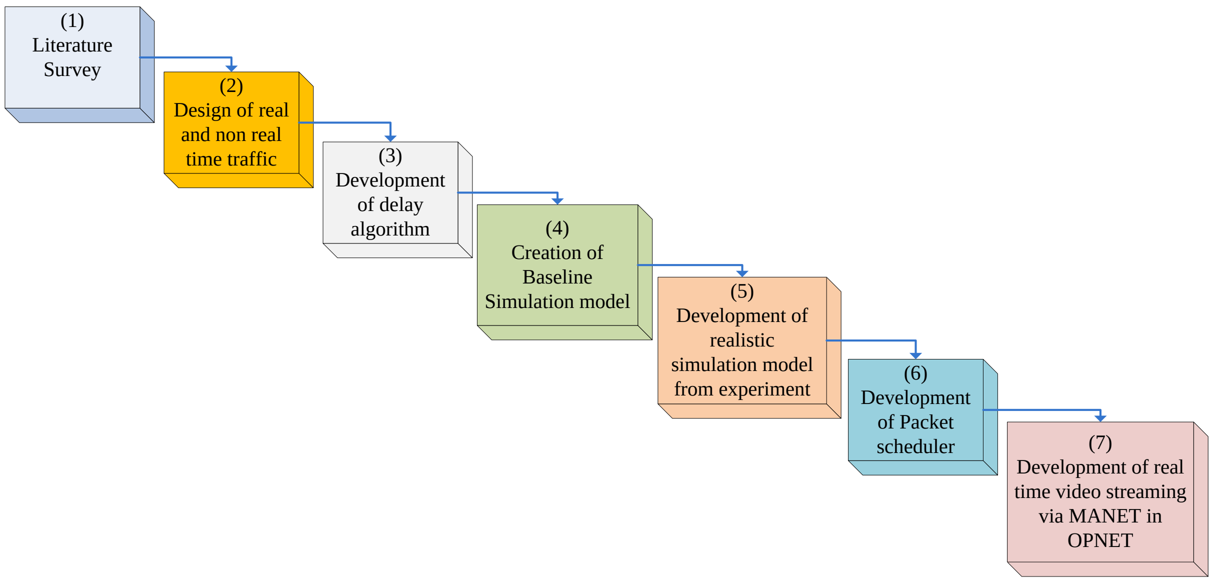

(1)
Literature Survey
(2)
Design of real and non real time traffic
(3)
Development of delay algorithm
(4)
Creation of Baseline Simulation model
(5)
Development of realistic simulation model from experiment
(6)
Development of Packet scheduler
(7)
Development of real time video streaming via MANET in OPNET

Figure 1-3: Research Methodology

1.6. Structure of Thesis

This thesis is structured as follows, Chapter 2 introduces wireless network, and it also explains the evolution of MANET, including some of the main issues. It contains information on the various routing protocols applicable to MANET as well as some of the challenges faced by the network. This chapter also discussed QoS, scheduling algorithm and reviewed existing scheduling algorithm for MANET. Chapter 3 introduces queuing model; a mathematical representation for the queuing system of Voice, Email and Internet Browsing is also presented. The methodology for modelling real and non real-time applications, as well as their behaviour under congested and uncongested states, is presented. Finally, wireless propagation model and a realistic simulation model are presented. Chapter 4 presents details for designing and modelling the proposed adaptive scheduling algorithm along with the performance analysis of the algorithm under various traffic classes and loads. Chapter 5 explains the guidelines for modelling a real-time video streaming in OPNET; this involves developing an interface between OPNET and an external hardware (web camera). The performance analysis of the scheduler under varying traffic classes and background loads for real-time video stream are presented, whilst Chapter 6 presents the conclusion and future work.

Chapter 2 Literature Review

2.1. Introduction

The objective of this literature review is to provide the underlying knowledge of current work being carried out by other researchers in this area and to provide a theoretical basis for this work. Various IEEE specifications, including Bluetooth, ZigBee, and IEEE 802.11 models are described. An explanation of the Open Systems Interconnection (OSI) model and a summary of its layers are included in this chapter. The evolution of MANET, routing and its challenges were also discussed and finally, a critical summary is provided.

2.2. Wireless Network

Wireless Network enables two or more devices to communicate using a standard network protocol, without the need for cabling. There are three commonly used communication standards: IEEE 802.11 standard for Wireless LANs (WLANs), IEEE 802.15.4/ZigBee, IEEE 802.15.1/Bluetooth technology that is the standard for Wireless Personal Area Networks (WPAN). A WLAN transmits and receives data at high radio frequency, whilst a WPAN, is a local area network formed by connected devices placed within a 10 metres radius [13].

2.2.1. IEEE Wireless Networking Specification

The Institute of Electrical and Electronic Engineers (IEEE) released the IEEE 802.11 specification in June 1999. The first IEEE released specification is called the IEEE 802.11; it operates at a frequency band of 2.4 GHz with a data rate of 1 to 2 Mbps using either Frequency Hopping Spread Spectrum (FHSS) or Direct Sequence Spread Spectrum (DSSS). In late 1999, the IEEE released two additional specifications i.e. the IEEE 802.11a and IEEE 802.11b. The IEEE 802.11b data rate is 11 Mbps with an operating frequency of 2.4 GHz, whilst the IEEE 802.11a supported data rate as high as 54 Mbps and utilises a frequency band of 5 GHz.

Unfortunately, both specifications (IEEE 802.11a and IEEE 802.11b) were incompatible because they operated in different frequency bands. In other words, the individual network card interface for both specifications and their corresponding access point cannot communicate with each other. As a result, the IEEE was forced to create a new draft standard known as IEEE 802.11g. The 802.11g can support a data rate of up to 54 Mbps, and it is compatible with the IEEE 802.11b. The emergence of Multiple-Input and Multiple-Output (MIMO) technology inspired the development of a new standard IEEE 802.11n; it supports a much higher data rate of 266 Mbps. Subsequent standards used the Single-Input-Single-Output (SISO) systems; this is an antenna technology in wireless systems, where the source and destination nodes use a single transmitting and receiving antenna. MIMO is an improvement from the SISO technology; it involves the use of multiple antennas at the source and destination. These antennas at both ends of the circuit are combined to minimise error and increase the data speed [14][15].

2.2.1.1. Bluetooth/IEEE 802.15.1

The Bluetooth wireless technology is an open standard specification for Radio Frequency (RF); it is based on short-range connectivity and supports both point-to-point and point-to-multipoint voice and data transfer [16]. Harnessing this technology to its fullest capacity will change the face of computing technology. The design is cheap, with low power requirement. It is now incorporated into a wireless networking system for all classes of portable devices e.g. mobile phones, notebook computer, refrigerator, laptop and Personal Digital Assistants (PDA), etc. It also enables the connection of multiple devices together without the need for a cable such as computers, printers, mouse, and keyboards. When two or more Bluetooth devices are connected together (share a single channel), they form a piconet. Bluetooth devices operate in the 2.4 GHz ISM band, and its communication channel consists of a pseudo-random hopping sequence using 79 RF channels. As the number of devices in the piconet increases, the tendency of two adjacent piconets operating at the same frequency increases; this causes packet collisions and thus retransmission of lost packets [17]. Bluetooth uses frequency hopping spread spectrum, which reduce interference, and fading. It uses Gaussian Frequency Shift Keying (GFSK) and Time Division Multiplexing (TDM) for multiple transmissions. Bluetooth supports Asynchronous Connection-Less (ACL) link for data and the Synchronous Connection-Oriented (SCO) link for voice [18][19].

2.2.1.2. ZigBee/ IEEE 802.15.4

ZigBee operates in three different frequency bands: free 2.4 GHz (ISM) band, 868 MHz in Europe and 915 MHz in America. ZigBee is a standard based on IEEE 802.15.4 for Wireless Personal Area Network (WPANs) [16]. WPAN is a personal area network that wirelessly interconnects devices. Devices in a WPAN can communicate with a speed of up to 250 kbps (at 2.4 GHz), 40 kbps (at 915 MHz) and 20 kbps (at 868 MHz); whilst physically separated by a distance of up to 50 m in typical circumstances and greater distance of 500 m in an ideal environment [20][21].

ZigBee consists of two devices namely: Full Function Device (FFD) and Reduced Function Device (RFD). FFD can be used as a repeater, a router or to co-ordinate the network. FFD can communicate with RFD or another FFD, RFD only communicates with FFD. RFD is only used as a terminal network node i.e. an RFD cannot communicate directly with another RFD because it cannot act as a router or repeater. The hardware for FFD and RFD are the same, but their differences lie in the network layer. ZigBee network supports three network topologies, which are cluster network, the star network and mesh network.

A star network is made up of multi-terminal network nodes along with the co-ordinator node. The co-ordinator node organises the network along with the routing information. Thus, all messages travel via the coordinator node. A sample star network is shown in Figure 2-1a. A Cluster network consists of multiple star networks whose central nodes are connected to a single co-ordinator. A Mesh network is a highly reliable ad hoc network. There are two connections types of a mesh network; these are partial mesh topology and full mesh topology. In a partial mesh topology, some nodes are connected to all the others; whilst some are only connected to those, they exchange the most data. When all the nodes in the network are connected to each other, it forms a full mesh network. A sample partial mesh network is shown in Figure 2-1b; nodes in a network can self-organise to form a mesh network. Mesh networks consist of multiple wireless routing paths, so alternative paths can easily be chosen should the existing one fail [21][22].

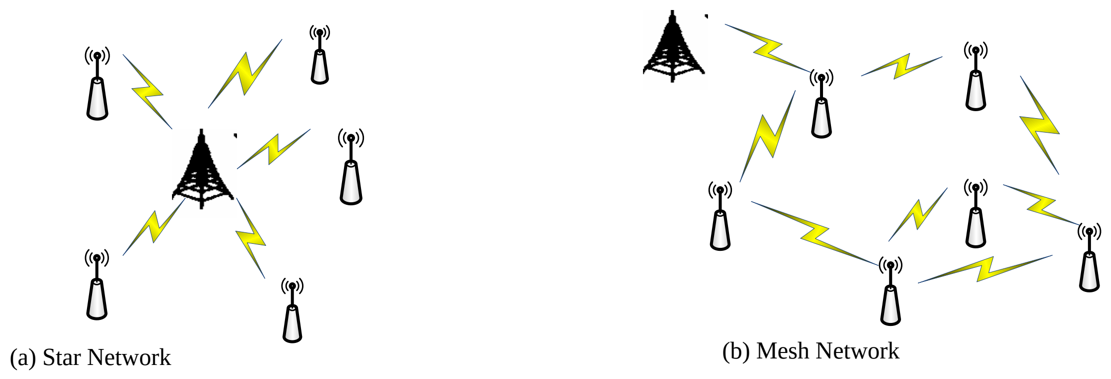


Figure 2-1: (a) and (b) Show the network topology for a Star and Mesh Network, respectively [22]

2.2.1.3. Wireless LAN/IEEE 802.11n

Some amendments were made to the IEEE 802.11g standard to develop the IEEE 802.11n standard. These amendments were made to the physical layer and Medium Access Control (MAC) layer. The modifications of the physical layer include the use of MIMO technology, channel bonding (the use of two 20 MHz bandwidth streams) and advanced coding (this includes Low-Density Parity Check (LDPC) codes). The modifications made to the MAC layer include the QoS features that include supporting delay sensitive application such as multimedia streaming and Voice over WLAN (VoWLAN)). Additional modifications were made to the extended switch announcement (allowing the access point to switch between support for 20 MHz only and 20 MHz/40 MHz) and the support of fast roaming [23]. The IEEE 802.11n uses Orthogonal Frequency Division Multiplexer (OFDM) scheme as its modulation technique. It modulates the data signal in parallel over multiple carriers. The achievable throughput of the IEEE 802.11n is dependent on the number of carrier signals. The number of carrier signals in IEEE 802.11a/b/g is 48 whilst IEEE 802.11n can use a maximum of 52 carrier signals. The IEEE 802.11n operates at an ISM of 2.4GHz and 5 GHz. The IEEE 802.11n consist of four spatial streams, with a maximum data rate of 260 Mbps (guard band interval (GI) = 800ns) and 288.9Mbps (GI = 400ns). The maximum data rate can be achieved when all four spatial streams are used for transmission. Hence, the data rate for the IEEE 802.11n can be varied by varying the number of spatial streams and the guard band interval, the lower the number of spatial streams, the lower the data rate [24]. This is the first time the speed of WLAN has theoretically exceeded the 100 Mbps of cable based fast Ethernet commonly used in workplaces. This standard also significantly improves the signal coverage area and thus maximises bandwidth utilisation. The data throughput is proportional to the distance between the transmitter and receiver; hence as the distance increases the throughput decreases. Therefore, the overall improvement in the throughput of the IEEE 802.11n enables it to cover a longer range with a significantly strong wireless signal at the receiver than the IEEE 802.11a/b/g. The IEEE 802.11n standard is backward compatible with the IEEE 802.11a/b/g standard. However, its full functionality can only be accomplished if the client and the access point support the IEEE 802.11n standard [25].

2.2.2. Open System Inter-connection (OSI) Model

International organisation for standardisation developed the Open System Interconnection (OSI) model. It was the first framework created for sending information through a network. The OSI model consists of seven layers as shown in Figure 2-2 [26][27].

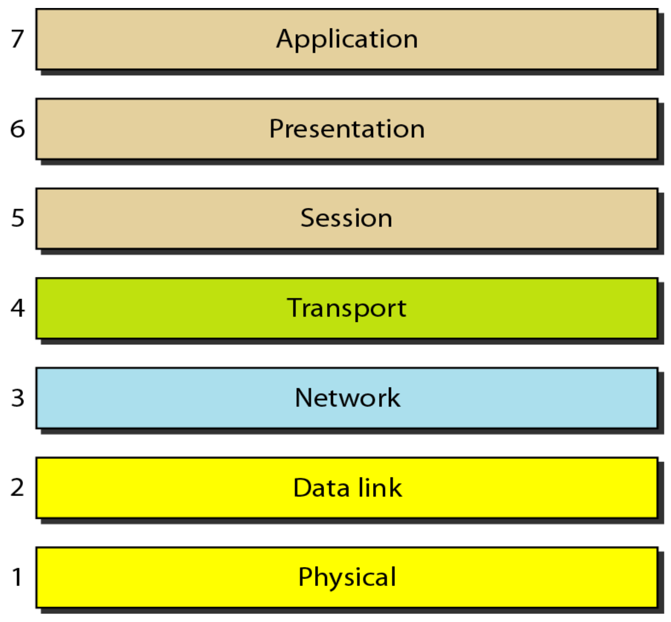


Figure 2-2: OSI Model [28]

2.2.2.1. The Physical Layer

The physical layer is the layer that interacts with the transmission media. It defines the interface specification, needed to establish network connections, and it also defines the medium requirements such as mechanical and all other specification for sending a bit stream in the network. The physical layer is responsible for transmitting and receiving data from one node to the next. One of the major functions of the physical layer is to provide services for the data link layer [28].

2.2.2.2. Data Link Layer

The data link layer is layer 2 of the OSI model. This layer helps devices to access the network in order to transmit and receive messages. It is also the point of the network where a physical address is assigned; thus enabling a device to send its data through the network. The data link layer collaborates with the device software to send and receive messages. Hence, this layer ensures that the initial connection between devices has been set up; messages to be transmitted are divided into frames. It also handles the acknowledgement from the receiver when these messages are successfully received. It also has the capability of error detection for received messages, flow control, etc. by analysing the bit patterns. This layer contains two sub-layers called Medium Access Control (MAC) and the Logical Link Control (LLC) [28].

2.2.2.3. Network Layer

It is layer 3 of the OSI model. This layer provides an end-to-end logical addressing system, thus enabling a transmitted packet to be routed through several layer 2 networks (Frame Relay, Ethernet, Token Ring, etc.). Layer 3 addressing developed in the past does not conform to the required standard. Novell is one of the software manufacturers that developed layer 3 addressing [29]. There has been a tremendous development in the networking industry, hence increasing the need for a common layer 3 addressing system. For example, the current Internet Protocol (IP) has made it easier to set-up a network connection and connects multiple users. It uses an IP addressing system to connect billions of people around the world.

2.2.2.4. Transport Layer

The transport layer of the OSI model is responsible for the point-to-point communication between devices through a network. Depending on the considered application, it does offer either a more reliable, best-effort, connectionless or a more connection oriented communication. Some of the most common transport layers protocol are the Transmission Control Protocol (TCP) which is a connection-oriented and the User Datagram Protocol (UDP) which is connectionless [28].

2.2.2.5. Session Layer

The session layer provides various services, one of such services involves the tracking of the number of received bytes individual sessions have acknowledged receiving from the other end session. The session layer allows either a half-duplex or a full duplex mode of communication between two processes [28].

2.2.2.6. Presentation Layer

The presentation layer is responsible for the format of the data sent through the network. The application can read and understand the message because of the presentation layer. Some of the presentation layer functionalities include formatting of data, formatting of complex data structure (strings, integers, structures etc.) used by the application to bit streams transmitted over the network. It includes data compression, which is the reduction of the size of transmitted data. It also includes security and privacy, which entails encryption; it is scrambling of data sent to the network so that only an authorised personnel can unscramble the received data. Finally, authentication, which is verifying the end user is whom they claim to be and not a fraud [28].

2.2.2.7. The Application Layer

The application layer gives the user (either human or software) access to the network. This is what is visible to the user when the user tries to load applications such as electronic mail, Internet Browsing, file transfer or a remote file access [28].

2.3. MANET Evolution

Historically MANET has been used in battlefields to enhance communication and survivability. The military battlefield operation is dynamic in nature; therefore, a fixed network cannot be relied on for quality communication. Wireless devices do have some added limitations due to the interference from other radio signals operating at similar frequencies and security issues. Radio frequencies, which are higher than 100 MHz cannot propagate beyond its Line Of Sight (LOS) [30]. MANET was uniquely designed to solve these problems with its multi-hop packet transmission capability, without any centralised or fixed infrastructure and to enhance connectivity beyond the LOS.

The first MANET was developed in 1972; it emerged from the Packet Radio Network (PRNet) project of the Defence Advanced Research Projects Agency (DARPA). The inspiration for the PRNet came from the development of packet switching technology, which includes bandwidth optimisation, storage and routing [30]. The architecture of PRNet consists of a network of broadcasting radios with minimal central control; a combination of Aloha and Carrier Sense Multiple Access (CSMA) protocols was used because of the dynamic sharing of broadcast radio. Additionally, the multi-hop nature of packet transfer removed the radio coverage limitation, hence, making it possible for multiple users to communicate within a large area. The Aloha method was initially developed by the University of Hawaii for use in satellite communication. It is a communication scheme that enables each source (transmitter) to send data to its destination whenever there is a frame to send. If the frame successfully arrives at the destination, the source sends the next frame; otherwise, (if unsuccessful) the source resends the frame. CSMA is a part of the MAC layer; it senses or listens for a network signal on the transmission medium before transmitting data. CSMA is implemented in a network with multiple network devices attached.

In 1983, the Survivable Radio Network (SURAN) was developed to combat some of the issues faced by PRNet. These include network security, scalability, energy management, and processing capacity. The objective was to create a network where node density could be in its thousands. The network should be able to survive security attacks and also use low-cost batteries that can meet the power demand of the complex packet radio protocol [30]. These resulted in the development of a Low-cost Packet Radio (LPR) technology in 1987 [31]. The LPR consists of Direct Sequence Spread Spectrum (DSSS) and an integrated packet switch based on the Intel 8086 microprocessor. An advanced network management protocol (bandwidth sharing) was also developed, and hierarchical network topology based on dynamic clustering was used to support network scalability. Other additional improvements to the radio security, adaptability and increased capacity were achieved through the management of the spreading keys [32].

The evolution of microcomputers and the subsequent tremendous growth of the Internet in the late 1980's and early 1990s has made the packet radio idea more applicable and likewise feasible [30]. DARPA Global (GloMo) information systems program was initiated by the Department of Defence (DoD) in the United States in 1994 [32]; it incorporates global information system into a mobile wireless environment. It supports Ethernet-type multimedia connectivity between wireless nodes or devices, within any network environment.

The largest-scale implementation of a mobile multi-hop network was carried out by the United States (US) Army in 1997 [30]. They modified Internet Protocol (IP) to accommodate inter-node communication. DSSS and Time Division Multiple Access (TDMA) were used, and the data rate was in the range of tens of kilobits per second. The discovery from this experiment reinforces the view that commercially available wireless protocols cannot cope with the variable data rates and high bit error rates because of the frequent topology changes [33].

In 1999, the US army carried out an experiment to demonstrate the feasibility of MANET in the Marine Corps war-fighting concept. It requires horizontal communication beyond the LOS between Marines and ships on the sea via an aerial relay. This technology was an advancement of the 'Extending the Littoral Battlespace Advance Concept Technology Demonstration (ELB ACTD)'. The network configuration for this experiment consisted of 20 mobile nodes. The backbone network and access point were built with Lucent's WaveLAN (wireless solution by Lucent) and VRC-99A (radios devices stationed on the ground, ship etc. fitted with antennas, power amplifiers and ad-hoc networking software enabling devices to organise themselves into a backbone network). The ELB ACTD successfully demonstrated the use of an aerial to connect users beyond LOS. In the mid-1990s, definitions of specific wireless standards (e.g. IEEE 802.11) became available, and the commercial wireless kit began to appear in the market. Wireless researchers became aware of the great commercial and research potential of MANET outside the military area.

Most of the existing academic research on MANET is not in the military domain. Recently, some commercial solutions such as Mesh Network and SPANworks are beginning to appear on the market [32]. Some of the applications of the commercially available Mesh Network is the automated meter reading system. The readings of residence are transferred from one residence to another until it gets to the central system without the need of human meter readers [34]. It is also used for post-earthquake site communication to enhance the emergency search and rescue operation [35].

2.3.1. MANET Model

Ad-hoc networks are the key to the evolution of wireless network. MANET inherits problems such as bandwidth optimisation, power control and transmission quality associated with wireless and mobile communication. In addition to the absence of a fixed infrastructure, the multi-hop nature of this network creates some research challenges [36].

Due to the dynamic network topology of MANET, the distribution of nodes and their ability to self-organise plays a major role on the network quality. Some of the major characteristics of a MANET are [37][38]:

- Its topology is very dynamic and difficult to predict.
- It does not require any infrastructure or centralised controller.
- Nodes perform both the role of host (source/destination) hosts and relay.
- MANET is made-up of wireless links, which have a much lower bit rate compared to wired communication systems.
- It can experience greater delay, packet drop, higher loss rate and jitter than fixed network. It is because of the wireless and dynamic network topology of a MANET.
- Mobile nodes rely on batteries for their energy. Thus optimising their use of power is extremely important.

These characteristics that define a MANET add additional challenges to researchers. Many research work is being carried out on network maintenance and discovery, configuration, ad-hoc addressing and self-routing in order to optimise and maintain the network quality. MANET can be deployed where there is little or no network infrastructure. The ad-hoc network maintains device connection to the network; it is also easy to add or remove a device from the network. Nodes have their individual power supply (typically batteries). It is difficult or impossible to replace batteries once the network has been deployed. Hence, energy saving design is a difficult constraint [37][39][40]. The radio transceiver often consumes much more energy than the other components of a node. A node must transmit with low power in order to optimise its battery life. Figure 2-3 shows an example of the multi-hop nature of MANET; for node 1 to transmit data to node 4 (out of its transmission range), the data will traverse node 2 or node 3 or both to reach node 4.

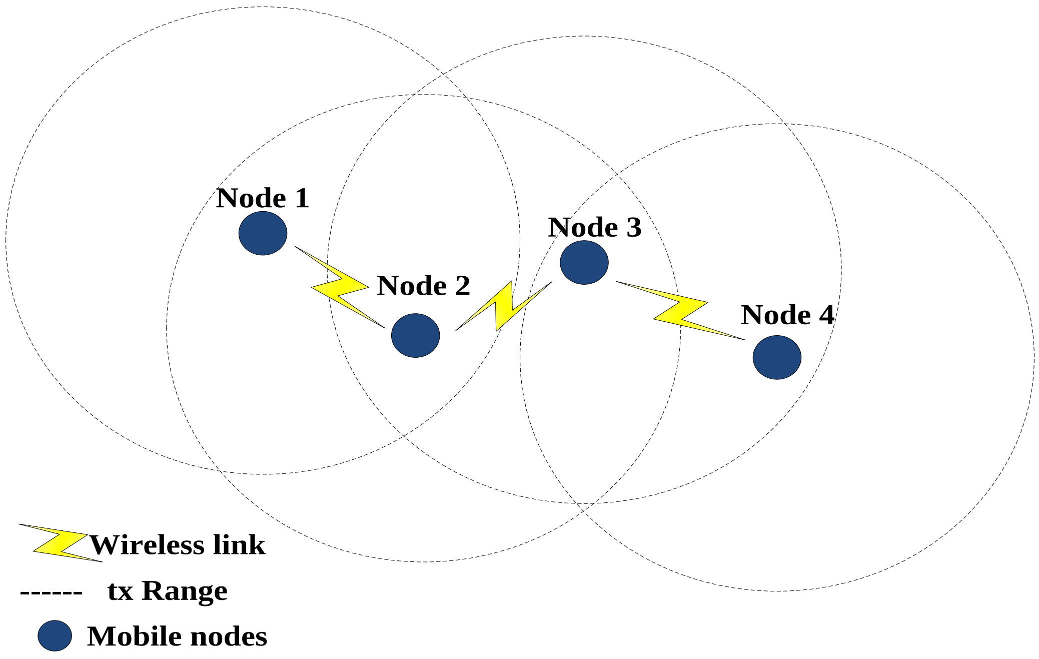


Figure 2-3: Multi-hop Pattern of MANET

2.3.2. MANET Issues

MANET faces a number of challenges, which affect its overall network performance. Some of these challenges are discussed in this section.

2.3.2.1. Dynamically Changing Topologies/Routes

The multi-hop network topology of MANET frequently changes, this results in the frequent network partition (broken communication link) and route changes. This can cause packet loss in the network; Figure 2-4 illustrates the dynamic nature of a MANET topology. Node 1, node 2, node 3 and node 4 are mobile nodes, where node 1 is the source, and node 4 is the destination. These nodes can move around within a specified space to form new links/connections. The initial connections as shown in Figure 2-4 changed because of the mobile nature of the nodes. The nodes can adapt dynamically to its changing network topology i.e. forming new connections.

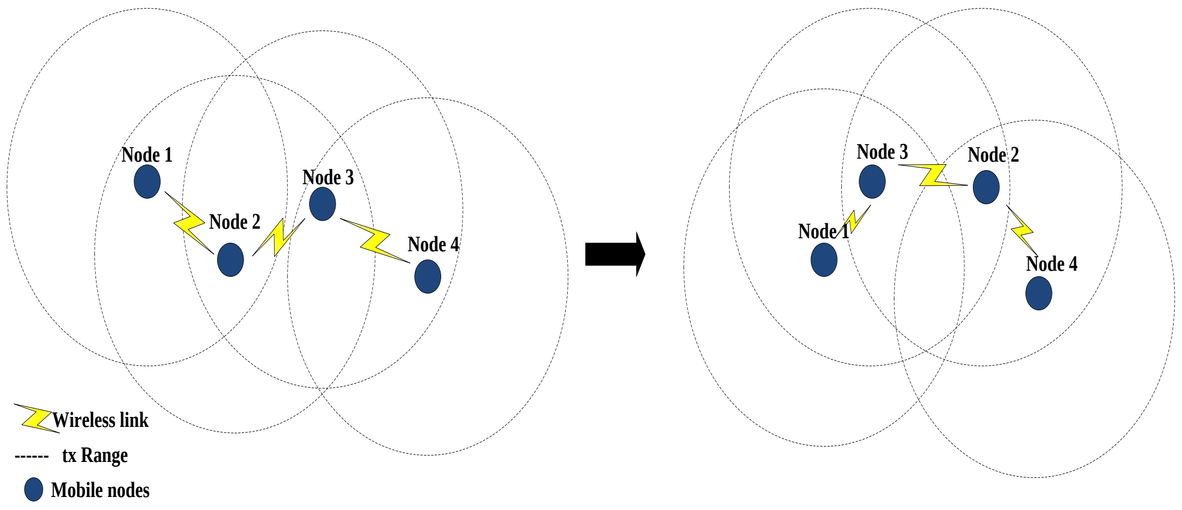


Figure 2-4: Node Re-organisation

2.3.2.2. Multi-hop routing and Lack of Mobility Awareness by application

There is no default available route to forward packets in MANET; individual nodes have the capability to act as both a source and relay. Nodes in MANET share routing and other information among themselves; however, there is a lack of mobility awareness among these nodes, thus routing information already in the routing directory becomes invalid when the nodes re-organise themselves. As a result, the transmitting node initiates a route discovery to discover a new path to the destination [32].

2.3.2.3. The differences in node and link capacities

In a mobile network, individual node might be equipped with different wireless transmission/reception capabilities, and they may also operate on different frequency bands. These differences in nodes capabilities can result in asymmetric links. Mobile nodes can also be equipped with different hardware and software that can result in varying processing capabilities. The network protocols and algorithms for a heterogeneous network are complex. Thus, it is challenging to design. The algorithm has to adapt dynamically to the varying link capabilities, power, channel condition, traffic distribution, congestion, etc. [32].

2.3.2.4. Energy Constrained operation

The processing power of packets in mobile nodes is limited because of the limitation of the batteries being used. This limits the application, and services individual nodes can support because each node requires additional power for forwarding packets to other nodes, processing packet, route discovery, node movement, transmission and reception [32]. Some research works are focused on efficiently managing the energy usage in the MANET. Paper [41] proposed a transmission and reception energy saving model called Energy Saving on Transmission and Reception based Enhancement Layers (ESTREL), which improves battery life of a node by reducing its activities (transmit and receive less packets) when the power falls below a certain threshold. Paper [42] proposed an algorithm that locates wireless network node position in an energy efficient manner. Paper [43] proposed a power minimisation algorithm to enhance the data throughput. Simulation result shows that their proposed algorithm outperforms the existing. Paper [44] proposed two algorithms ‘Select Node’ and ‘Total Minimum Power’ to optimise battery power in MANET. The ‘Select node’ algorithm selects all nodes available for the data transmission, while the ‘Total Minimum Power’ calculates the minimum power required for transmission at any time instance. This algorithm predicts the route with the minimum power requirement, which is then used to transmit data from the source to the destination.

2.3.2.5. Network Scalability

Network management algorithms currently available were mostly designed for infrastructural and small-scale wireless networks. Most mobile network applications consist of a large network area; these consist of nodes in their tens of thousands similar to tactical networks. The issue of scalability must be addressed for the successful deployment of large-scale network. The limited network resources as well as the dynamic network topology present challenges such as configuration management, routing, addressing, interoperability, location management, security and the resulting high capacity wireless network [32].

2.3.3. MANET Routing Protocols

Routing protocols designed for wired networks are not suitable for MANET, because of the mobile nature of the nodes, as routes will expire and change over time [45]. A whole set of different routing protocols has been designed specifically for MANET. This routing falls into two major categories, the proactive and reactive protocols. These protocols will be discussed further below.

2.3.3.1. Proactive Protocols

A proactive routing protocol keeps an updated routing table by regularly requesting and sharing updated information with its neighbouring nodes. Mobile nodes that want to transmit packets already know the packets optimal route to their respective destination. This scheme requires a high number of messages to keep the routing tables up to date; as a result, the limited network resources such as bandwidth and battery life are used. Proactive protocols are often referred to as Distance-Vector or Link State protocol. An example of proactive protocol is the Destination-Sequenced Distance-Vector routing (DSDV) [16].

2.3.3.2. Reactive Protocol

A reactive protocol establishes a route when a node wishes to transmit a packet, and there is no valid route in the routing table. Nodes maintain routes until the destination is not reachable, or the route is no longer needed. The advantage of this type of routing protocol is that, there is less traffic overhead compared to the proactive routing protocol; thus increasing the available bandwidth for sending more information. Examples of reactive protocols are Dynamic Source Routing (DSR) and Ad-hoc On-demand Distance Vector routing protocol (AODV) [46], both protocols are discussed below.

A. Ad hoc On-demand Distance Vector Routing (AODV)

When using Ad hoc On-demand Distance Vector Routing (AODV) protocol, a route is established only when the source node needs it. This protocol was designed to be autonomous and adapt to a variety of different network behaviour such as node mobility, channel changes, link failures, and packet loss. AODV consists of two mechanisms, which are Route Discovery and Route Maintenance. Each node maintains at most one route to a destination. When the available route from a source to a destination fails, AODV protocol invokes a new route discovery. Route discovery is also invoked whenever there is a topology change; however, this can sometimes be inefficient. AODV uses the destination sequence to maintain an up to date path to the destination. Nodes update their routing table when the destination sequence of a received packet is greater than or equal to the stored destination sequence at the node with a smaller hop count. The routing table of nodes contains the most recent available route to the destination as confirmed by the source. AODV uses a broadcast identify number to prevent multiple broadcasts of similar packets. Intermediate nodes only transmit the first received copy of a packet and similar packets received are discarded [47]. Each routing table entry in AODV consists of the following fields:

- Destination IP Address: The IP address of the destination of the packet being transmitted
- Destination Sequence Number
- Next hop: This can be the IP address of the destination or an intermediate node configured to forward the packet to its destination.
- Hop Count: The number of hops required from the source to the destination
- Lifetime: The time, in which nodes receiving the RREP packet, considers a route to be valid and is in milliseconds
- Routing Flags: The current state of the route; down (not valid) or in repair, up (valid)

B. Dynamic Source Routing protocol (DSR)

DSR uses source routing instead of relying on routing tables as in AODV. Individual nodes maintain a routing log accumulated during the process of route discovery, where learned routes are stored. In order to send data to another node, the sender first checks the routing log to determine if there is an available route. If there is an available route, the source appends this routing information to the packet header and sends it to the next available hop on its path. Each intermediate node on receiving the packet examines the header to determine the next available hop and then forwards the packet to the next node indicated in the packet route. When no route is found on the packet, it is sent to a buffer whilst the router tries to obtain a route using a route discovery process [48]. The route discovery and maintenance process are discussed below.

Route Discovery and Maintenance:

- In order to discover a route for a packet, a Route Request (RREQ) packet is broadcast from the source to all nodes within its radio transmission range. A route request contains the address of the source and destination nodes as well as a route record. The route record contains the entire path followed by the route request packet to get to the destination from the source node [48]. The node, which receives the route request does the following:
- It checks if the address matches the destination address of the route if so it is the destination; otherwise it is an intermediate node. If the node is receiving this route request for the first time, it checks its routing table to find a route to the destination. If a route is found, it creates a route reply packet with the route from its routing table and sends it back to the source. This type of reply is called intermediate-node reply. Otherwise, if the route to the destination is not available in the routing table; the node appends its address to the routing table, increment the hop count by one and rebroadcast the request. This process continues until the destination or a node with a valid route is reached. When a node acknowledges being the required destination on receiving the RREQ packet, the node sends a Route Reply packet (RREP) to the node that originated the RREQ message. This RREP follows a reverse route of the RREQ packet. On arriving at the originating node, the route path to the destination node is retrieved from the RREP packet [48][49].

When the RREP packet reaches the source, it adds the routing information to its routing table and then sends any pending data in its buffer to the destination via the new route. If the MAC layer of the transmitting node detects any broken links, a route error packet is generated. This packet is sent back to the source through the reverse path to erase all entries in the routing table that contains the broken link.

2.3.4. Review of previous work on routing protocol for MANET

A number of publications have analysed the performance of proactive and reactive routing protocols with respect to mobility and the network load. Paper [16] compared the effect of mobility on the performance of six routing protocols: four reactive (Associative-Routing protocol (ABR), AODV, DSR, Location-Aided routing Protocol (LAR)) and two proactive (Fisheye State routing Protocols (FSR), Wireless Routing Protocol (WRP)). Their measuring metric was routing overhead, power consumption, PDR and end-to-end delay. They concluded that mobility has a negative effect on network performance of the routing protocols. Reactive protocols are more adaptive to the network changes in MANET as compared to pro-active protocols. As thus reactive protocol outperformed proactive protocols. The network performance for proactive protocols decreases as the network topology changes; thus resulting in high routing overhead and higher power consumption. Paper [50] compared the mobility effect of two on demand protocols DSR and AODV. Their measuring metric are end-to-end delay and throughput. They concluded that DSR performs better than AODV when there are smaller number of nodes with less mobility; while the AODV performs better, when there is a large number of nodes with more mobility. Therefore, based on the literature, reactive protocol is used for all MANET Model in this thesis.

2.4. Congestion Control Mechanism

Congestion occurs in a network when the traffic intensity is greater than the network capacity. Congestion control, are preventive control mechanisms put in place to improve network performance. QoS is the measure of the network performance; its measuring metric includes delay, packet loss and jitter. The network QoS can be improved if appropriate congestion control mechanisms are put in place [51].

2.4.1. Traffic Classification

The intensity of traffic flow to the queue contributes to the congestive state of that queue as well as the network QoS. Hence, to understand the concept of congestion control and QoS, an underlying knowledge of the various traffic classes is required. Traffic flow can be classified into one of the following traffic class: Constant-Bit-Rate (CBR), Variable-Bit-Rate (VBR) and Bursty-bit-rate. These traffic classes are based on the inter-arrival times/distribution of the traffics and are illustrated in Figure 2-5 [52].

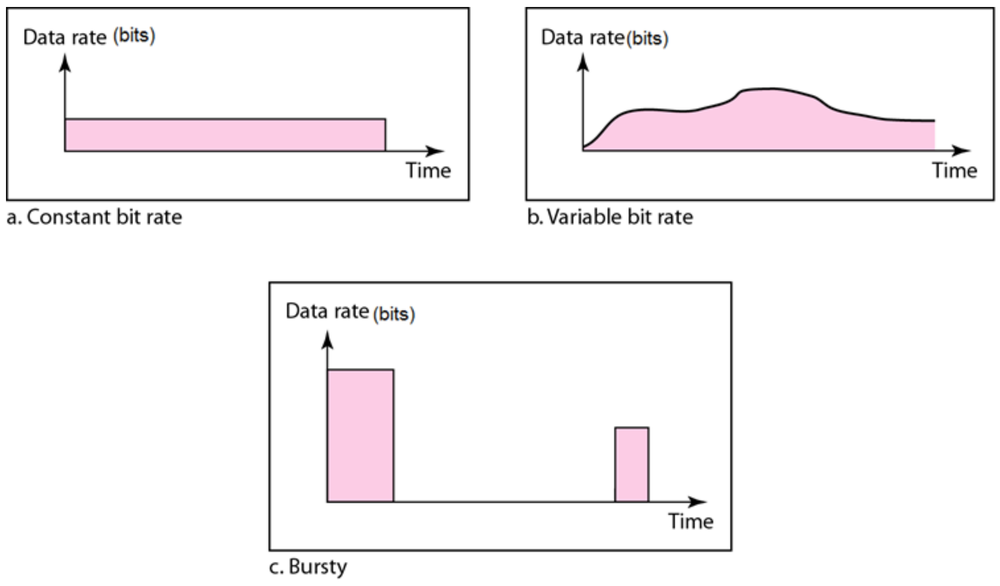


Figure 2-5 Traffic profiles: (a) Constant bit rate (b) Variable bit rate and (c) Bursty[52]

2.4.1.1. Constant Bit Rate (CBR)

The data rate for CBR traffic does not vary over time as shown in Figure 2-5a. The average data rate is constant over time. Theoretically, the QoS requirement for this type of traffic is constant and easily predicted, so the network can allocate the bandwidth needed for a traffic flow when no other traffic is present [82]. This type of traffic is delay sensitive as it consists of real-time traffic. An example of this type of traffic is voice, video, or control application [83].

2.4.1.2. Variable Bit Rate (VBR)

Figure 2-5b is an example of VBR traffic flow, the data rate changes with time. These changes are normally smooth and not sharp or sudden. This traffic type is more difficult for the network to deal with because the network cannot readily predict the resources needed for the traffic flow. Examples of such types of traffic are compressed video and voice streams [82][83].

2.4.1.3. Bursty

Bursty traffic is characterised by the sudden change in the data rate within very short period of time. For example, the data rate might suddenly increase from zero to about 1 Mbps in less than a few microseconds and vice versa. The data rate can also remain at the same value for some time. The maximum burst size is important for traffic flow in a bursty network; because the network uses it to predict the required resources for a particular flow. This traffic type is the most difficult for a network to predict the required network resources because of the behaviour of the traffic. Thus, traffic reshaping is applied to enhance the network performance. An example of a bursty traffic is shown in Figure 2-5c [53].

2.4.2. Network Congestion

Congestion has traditionally been one of the major issues of packet-switched networks. Congestion is caused when the network load (the number of packets sent to the network) is greater than the network capacity, hence waiting or queuing occurs i.e when the rate of packet arrival is greater than the packet-processing rate, there will be a gradual accumulation of packets within the queue; this will eventually cause congestion. Congestion control mechanisms or techniques are used to control congestion, i.e., keep the network load below the network capacity. For example, congestion will occur on the road as a result of abnormalities of the traffic flow caused by an accident or road work during rush hours [53].

2.4.2.1. Network Performance

This is the measure of the QoS of a network. The factors considered when measuring the network performance are delay, throughput, packet loss and jitter. The performance of the throughput is proportional to the packet drops.

A. Delay versus Load

The relationship between packet delay and network load is shown in Figure 2-6. Delay is the total time a packet takes to transverse the network from the source to the destination node. The link capacity is the maximum number of bytes a link can accommodate without being overloaded or congested, and it is given by the Shannon's equation (equation (2-1)) shown below, where ' B ' is the bandwidth in Hz [54]. This is an important index for the network performance.

$$C_{link} = B(1+\frac{SNR}{B})bits/\sec \qquad (2\text{-}1)$$

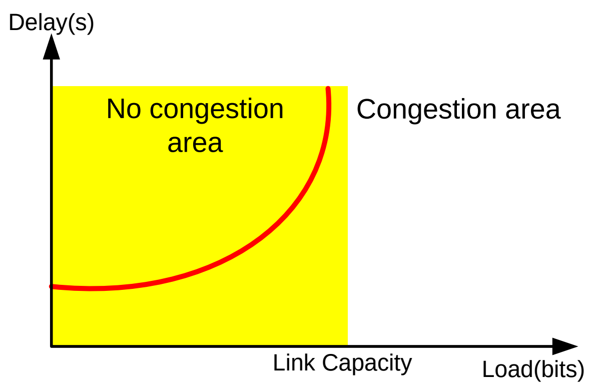


Figure 2-6 Packet Delay versus Network Load [52]

The packet delay is at a minimum when the network load is very small. At low load, the waiting time of a packet in the queue is zero. Hence, the packet delay will only consist of the processing, transmission and propagation delay. As the network load gradually increases, the packet waiting time gradually increases as well. When the network load is equal or greater than the link capacity, the packet waiting time tends towards infinity as shown in Figure 2-6 [53]. The implication of the infinite waiting time is a possibility that no traffic arriving at the queue will reach the destination, and the queue gets longer and longer. The long delays of packets in the queue have an immense effect on the network load. When a packet from the sender is not received at the destination within a stipulated time, a retransmission request is sent, which in turns worsen the delay and congested state of the queue [51].

B. Throughput versus Load

Throughput can be defined as the total number of bits passing through a point per second. This definition can also be extended further for the throughput of a network, which is the total number of load passing through that network in a unit time as shown by equation (2-2). Bearing this in mind, throughput versus network load is depicted in Figure 2-7.

$$Throughput = \frac{amount\ \ of\ \ data\ received}{time}(bits/\sec) \qquad (2\text{-}2)$$

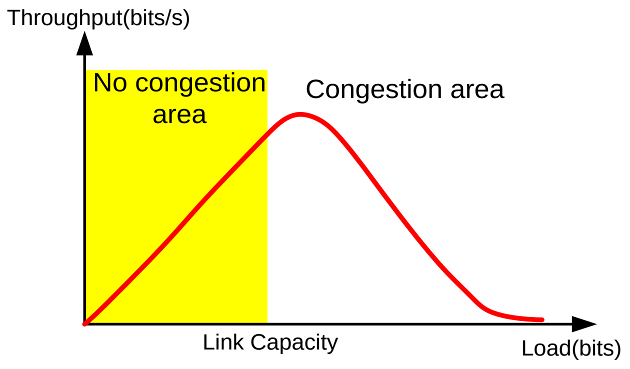


Figure 2-7: Throughput versus Load [52]

As shown in Figure 2-7, the network load increases proportionally with the throughput until it reaches maximum capacity when the throughput starts to drop because of congestion [52]; this is the bottleneck effect. The queue discards packets that have stayed longer than a pre-defined time threshold in the queue. However, this will not leave the queue less congested as the discarded packets are retransmitted by the source when it does not receive an acknowledgement of the packet received from the destination within a specified time.

2.4.2.2. Congestion Control Mechanisms

Congestion Control is the technique or mechanism put in place to prevent congestion from happening or remove congestion after it has already occurred thus improving the QoS. For this reason, congestion control can be divided into two major categories: Open-Loop Congestion Control (prevention) and Closed-Loop Congestion Control (removal) [53]. Open-Loop Congestion Control policies/mechanisms are put in place to prevent the occurrence of congestion at either the source node or the destination node. Whilst Closed-

Loop Congestion Control mechanisms aim to remove or reduce, the effect of congestion in a network after it has already occurred.

2.4.3. Quality of Service

Quality of Service (QoS) is a measure of, how well a network performs and the ability to provide appropriate services for selected traffic. Implementing the QoS requirements for a network involves fine-tuning the network parameters so that the appropriate amount of network resources is provided to different traffic class or profiles according to the requirements. This optimisation is implemented through sets of queues within each mobile node by applying different types of scheduling and traffic condition mechanism. Different applications have different characteristics and as such, their QoS requirements vary. The International Telecommunication Union (ITU) performance targets for voice and video traffic is shown in Table 2-1. Some of the characteristics of a network with an acceptable QoS are reliability, acceptable delay and jitter. Supporting QoS in MANET is a major challenge due to a number of factors; these include limited network resources, frequent changes in the network topology due to node movement, the wireless connectivity, and limited node power. These challenges need to be considered in order to provide the appropriate QoS requirements for applications in MANET.

2.4.3.1. Reliability

Reliability is an attribute associated with a network that consistently conforms to specifications; such specifications are network delay, jitter, and packet loss. One of the important characteristics of a traffic flow is the network reliability; if the network is not reliable, there will be high packet loss and a low number of received packets acknowledgments from the receiver. However, the importance of reliability varies with different applications [52]. For example, non-time-sensitive applications such as Email, file transfer and internet browsing are more sensitive to the network reliability than delay sensitive applications such as audio or video conferencing. Audio and video conferencing can tolerate a certain level of packet loss as shown in Table 2-1, whilst applications such as Email, file transfer and Internet browsing cannot tolerate loss, as the files might end up corrupted, and the end user will find it difficult to make sense of the received information.

Table 2-1: Performance target for audio and video applications [55]

Media	Application	Degree of symmetry	Typical data rates (kb/s)	Key performance parameters and target value		
				One-way delay (ms)	Delay Variation (ms)	packet loss ratio
Audio	Conversational Voice	Two-way	4-64	<150 Preferred*<400 limit*	<1	<3%
Audio	Voice messaging	Primarily one-way	4-32	<1000 for playback <2000 for record	<1	<3%
Audio	High-quality streaming audio	Primarily one-way	16-128	<10000	<1	<1%
Video	Videophone	Two-way	16-384	<150 preferred <400 limit		<1%
Video	One-way	One-way	16-384	<10000		<1%

2.4.3.2. Jitter

Jitter is the variance of the inter-arrival times of packets at the destination and it is caused by network congestion. This variance must be low and constant to maintain a good QoS for real-time applications. Delay sensitive applications such as voice and video cannot tolerate high jitter. When the delay for a real-time voice or video traffic is 2 ms for the first and second packet, and 50 ms delay for the third and fourth; the quality of the transmission will be affected and the audience can notice the delay variation. This is different for non-real-time applications such as file transfer, email or internet browsing, the transport layer will wait for all packets to arrive, before it is being delivered to the application layer [52].

2.4.4. Techniques to Improve QoS

QoS deployment involves the use of either one or a combination of the following components: scheduling, traffic shaping, admission control and resource reservation [52]. The traditional means of communications employ different network for different application. An example is the telecommunication infrastructure used for voice telephony services; there are also separate network/infrastructure for television, radio and general data services that include File Transfer (FTP) and internet browsing. These applications were individually developed and engineered for the specific service they provide. The recent advances in computing and communications technology have created the need to converge these different applications into a single integrated network. These applications have various QoS requirements that are challenging to meet in a single integrated network. QoS improvement techniques such as traffic shaping, admission control and resource reservation are more effective in a network with a single application. Thus, to cater for these multiple QoS requirements, an individual or a flow-based packet scheduler is required.

This work will focus on scheduling, as it is a comprehensive traffic management schemes. It is a congestion prevention technique that efficiently manages how packets are been processed in the queue without necessarily influencing the network activities such as the source transmission rate.

2.4.4.1. Packet Scheduling

Scheduling is an algorithm that determines the order in which a traffic flow can access the available resources. Packets from various flows arrives at the queue, a packet scheduler is used to treat each flow fairly to maximise the network performance. Some of the conventional available packet schedulers are First in First Out (FIFO), Priority Queuing (PQ), Round Robin, Weighted Fair Queuing (WFQ) and Weighted Distance Scheduling (WDS) [56]. These schedulers are briefly discussed below:

First in, First out (FIFO):

FIFO, also known as first come, first served, arriving packets are kept in the buffer until they are ready to be processed by the queue. Each packet is served in the order of their arrival at the queue. If the packet arrival rate is greater than the queue processing rate, the queue will be filled up and newly arriving packets to the queue will be discarded to avoid congestion. The advantage of this queuing scheme is its simplicity to implement, and all packets are treated fairly. However, it is not suitable for a multi-traffic network with real-time applications.

Priority Queuing (PQ):

PQ gives a high priority to important traffic as specified by the user. Packets are assigned various priorities based on their traffic flow and each priority class is assigned a sub-queue. Packets with the highest priority are served first in a FIFO manner, and then lesser priority packets are served afterwards. PQ maintains two or multiple separate queues for multiple traffic flows and these queues are served according to their priority level in a FIFO order. This queuing scheme is suitable for real-time traffic. However, the least prioritised traffic will experience starvation as the traffic intensity of traffic with higher priority increase.

Round Robin Scheduling (RRS):

A Round Robin scheduler identifies each flow by a source and destination pair. It maintains a per-flow queue. Each flow is assigned a small time of equal proportion and is serviced in a circular manner. It is one of the simplest scheduling schemes to be implemented. However, the disadvantage of this scheduling scheme is that all traffic are treated the same and as such it is not suitable for a network with mixed real and non-real time traffic.

Weighted Fair Queuing (WFQ):

Packets are assigned various priority class and admitted to different queues. Each queue is weighed according to the traffic class. This scheme processes packets in a round robin way; the number of packets served at each queue is dependent on the weight of the queue, i.e. the higher priority packets are served more than the lower priority traffic. This is more suitable for real-time traffic; however, the lower priority traffics will suffer from starvation when the traffic intensity of the higher priority traffic is high.

Weighted-Distance Scheduling (WDS):

The WDS is also a round robin scheduler; it gives priority to packets that have the shortest path to their destination. The remaining distance is defined as the distance between a chosen next hop and the destination. The smaller the distance between a source and its destination, the fewer the number of hops it will take to get there.

2.5. Related Work on QoS Improvement in MANET

The conventional schedulers described in the previous section are not enough to guarantee the required QoS in MANET. Paper [57] proposed an efficient scheme for the sharing and dissemination of data files in MANET using tornado coding. It uses several suppliers to transmit tornado code data to the destination. Their protocol reduces file download times of a requesting peer by as much as 75%; however, it is computationally complex and uses much network resources. Paper [58] proposed a Routing Control Packets Limited Forwarding (RCPLF) algorithm. This algorithm limits route discovery to a limited region thus reducing the bandwidth occupied by control packets. However, this is not enough to provide the required QoS service in MANET. Paper [59] significantly improved the video quality by reducing interference, using a multiple point-to-point transmission with Multiple Descriptions Coding (MDC). It uses more network resources and is computationally complex.

Paper [60] proposed two novel control slot scheduling approaches for creating dynamic contention-free Time Division Multiple Access (TDMA) schedulers. The two proposed control approaches are Network Entry Based Scheduling (NEBS) and Virtual Slot Scheduling (VSLOT). Their approach focused on efficiently exchanging the control traffic in the MANET and Mesh network so that the dynamic QoS requirements of the participating nodes can be met. Their algorithms produced lower delay for control information. However, the algorithm is computationally complex and only considered control information.

Paper [61] proposed a congestion aware scheduling algorithm (CARE) that considers the traffic load of its neighbouring nodes when deciding the priority of the passing flow. Their algorithm showed a significant performance improvement when compared to conventional priority and drop-tail algorithm. Considering the load of the neighbouring nodes alone is not enough to guarantee the QoS in MANET because of the dynamic nature of the topology.

Paper [62] combined Hybrid Coordination Function (HCF) with two conventional scheduling algorithms, Low Latency Queuing (LLQ) and Custom Queuing (CQ) to improve the QoS for real-time multimedia voice and video traffic in home network. Their performance analysis showed CQ with HCF performs better than LLQ in terms of delay and jitter. However, as CQ schedules queue cyclically, the delay guarantee for real-time traffic is low at high traffic intensity.

Paper [63] proposed a bottleneck first scheduling scheme for CBR traffic. The scheduler prioritised nodes with higher traffic load to avoid bursty traffic delay. At individual nodes, the traffic with the most amount of hop to the destination is served first. The proposed scheduler showed a satisfactory network delay. This is not suitable for real-time applications as preference is given to the traffic level in a node rather than the traffic class.

Paper [64] proposed a new scheme for multiclass packet scheduling in MANET. They extended the existing Earliest Deadline First (EDF) scheduler to assign priority to the packet. Simulation results showed the extended EDF scheduler performed better than the existing. However, this is not robust enough for multi traffic load in MANET.

Paper [65] developed an urgency-based packet scheduling and routing algorithms effectively to deliver delay-sensitive video data over mobile ad hoc networks. The packet, node and route urgency were determined based on the end-to-end delay requirement and the number of hops over the route. Based on this urgency the queue determines the packet service order. However, determining the number of hops can be challenging, as there is frequent topology change in MANET because of node movement.

These approaches are not robust enough to accommodate the dynamic nature of participating nodes in MANET as thus do not provide appropriate control measures for network with higher load.

Paper [66] proposed a Mamdani fuzzy inference system with two input variables and a single output (priority index), to schedule packets in MANET. The two input variables are channel capacity and data rate, each of which has three linguistic variables (Low, Medium and High). The algorithm consists of nine rules, the priority index of packets to be scheduled are determined. Each node consists of three queues; packet is inserted and served in these queues based on their priority index. Paper [67] also presented some work on Mamdani fuzzy scheduling with MANET (based on buffer size and number of hops suffered by packet). Based on [66], a better way to improve the QoS for MANET was developed.

2.6. Fuzzy Inference System (FIS)

FIS is a system that implements the human experiences and preferences with membership functions and fuzzy rules. It can be used as a general methodology to incorporate knowledge, heuristics or theory into controllers and decision-making [67]. A fuzzy model is made up of four blocks; these blocks consist of a fuzzifier, defuzzifier, the inference engine and fuzzy knowledge base as shown in Figure 2-8. The fuzzifier decides how to convert the crisp input into a fuzzy input to be used by the inference engine. This is achieved by mapping the crisp input to a set of input membership functions stored in the Knowledge Base. The inference engine applies reasoning to compute the fuzzy output using ‘IF-THEN’ type fuzzy rules that are stored in the Knowledge Base, which is used to convert the fuzzy inputs to fuzzy outputs. The Defuzzifier converts the fuzzy outputs into a crisp value using an output membership function stored in the knowledge base [66].

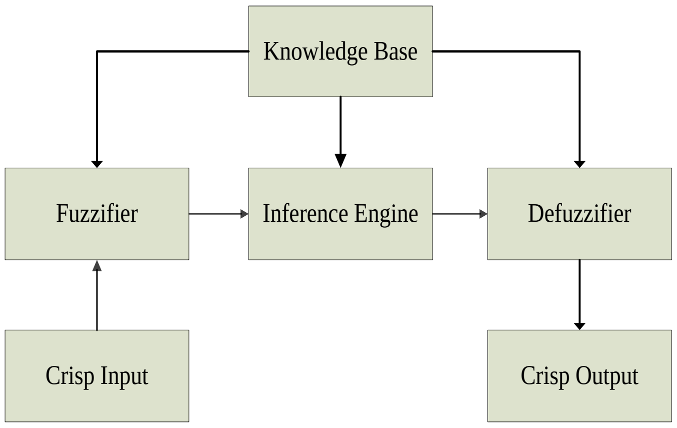


Figure 2-8: Basic Fuzzy System [66]

There are a number of possible membership functions; these include trapezoidal, triangular, piecewise linear, Gaussian and singleton. The most commonly used membership functions are trapezoidal, triangular and Gaussian shapes. The type of the membership function used can be context dependent and is chosen arbitrarily by the user depending on their level of experience [68]. A more detailed explanation is given in Chapter 4.

2.7. Critical Summary

MANET has attracted particular interest in recent years from researchers. This is because of its potential real-time applications, which include search and rescue, environmental monitoring, medical etc. When fully developed MANET has the potential to replace some of the infrastructural network. However, this does come with some research challenges, which include the dynamic network topology and wireless nature of the network. The recent advancement in wireless technology such as the IEEE 802.11a,b,g and n has been a motivating factor for carrying out further research in this area as the effect of channel attenuation (such as fading) is reduced. MANET is mostly considered for real-time applications, however it have limited network resources. These real-time applications have QoS that are difficult to meet. The QoS performance can be improved using an efficient traffic management system such as a scheduler. Some of the conventionally available schedulers are not able to schedule packets efficiently in a MANET, because they were primarily created for a fixed/wired network. The frequent topology change within a MANET has increased the need for an adaptive packet scheduler that can adapt to the real-time changes in the network resources. Many researchers in this area have focused on improving the channel quality or the routing protocol, only a few have considered proposing an adaptive scheduler for MANET as a means of improving the QoS of the network. However, these algorithms are not robust enough and their performance degrades in a network with mixed traffic. Some of the available schedulers have been shown to be computationally complex or not dynamic enough to accommodate the real-time fluctuation in the network resources and deal with mixed traffic. Furthermore, the energy constraint of a MANET has nessesitated the need for a scheduler that is less computationally complex.

According to previous studies, reactive protocols have shown to be well suited for MANET applications because they are on-demand protocols and adapt more quickly to the changing network topology with less traffic overhead as compared to a proactive protocol, thus increasing the available network resources. For this reason, this work will consider reactive protocols as the routing model in the network design of MANET. Previous work in MANET used the IEEE 802.11g or a lower standard (IEEE 802.11a/b). The IEEE 802.11n is a new improved standard with some updated features such as the channel capacity, speed, error correction and detection, etc. The IEEE 802.11n was configured for the wireless model used in this thesis.

The dynamic nature of MANET has an adverse effect on the traffic behaviour and thus affects the QoS performance of individual nodes. This can be modelled as a queue with a known capacity, with packets of different sizes and arrival rates being offered to the queue to be serviced. This can cause unpredictable behaviour that result in congestion. The next chapter builds on this foundation, it aims to explain traffic behaviour in a queue using queuing theory, and this formed the basis of this work.

Chapter 3 Baseline Simulation Model

3.1. Introduction

There is a need to understand the basic principle of queuing theory in order to understand traffic behaviour in a queue. This chapter presents the fundamental knowledge of queuing systems. Section 3.2 introduces the concept of queuing theory; it discusses two queuing systems, the M/M/1 and M/G/1 as they form the basis for this chapter. Queuing theory is used to analyse the various traffic behaviours in the queue and also to proffer a solution that efficiently manages traffic in the queue under congested and uncongested state. This chapter also presents guidelines and parameters for modelling a realistic propagation model based on results collected in an open field experiment.

3.2. Queuing Theory

Queuing Theory is the mathematical study of waiting in lines, it allows the mathematical analysis of several related processes in a queue [69]. It is also seen as a branch of applied probability theory. The applications range from communication networks, computer systems, machine plants and so forth. The basic queuing process involves arrival at the end of the queue, waiting in the queue and being served at the front of the queue [70].

In recent years, the most considered measuring metric for performance analysis for a network is the delay, jitter, throughput, and PDR. These measuring metrics reflect the state of the network at any given time.

Queuing theory requires some simplified assumptions as a more realistic model will be difficult to analyse because it consists of retransmission protocols and wireless links that can cause packet loss. For this reason, it is extremely difficult to be able to predict the actual quantitative delay using queuing models. However, it does provide predictions that are an approximation to the actual delay and also valuable qualitative result that serves as the basis for the network analysis with an important knowledge of the network components [69]. The average end-to-end delay consists of four parts: queuing delay, processing delay, transmission delay and the propagation delay as shown in Figure 3-1.

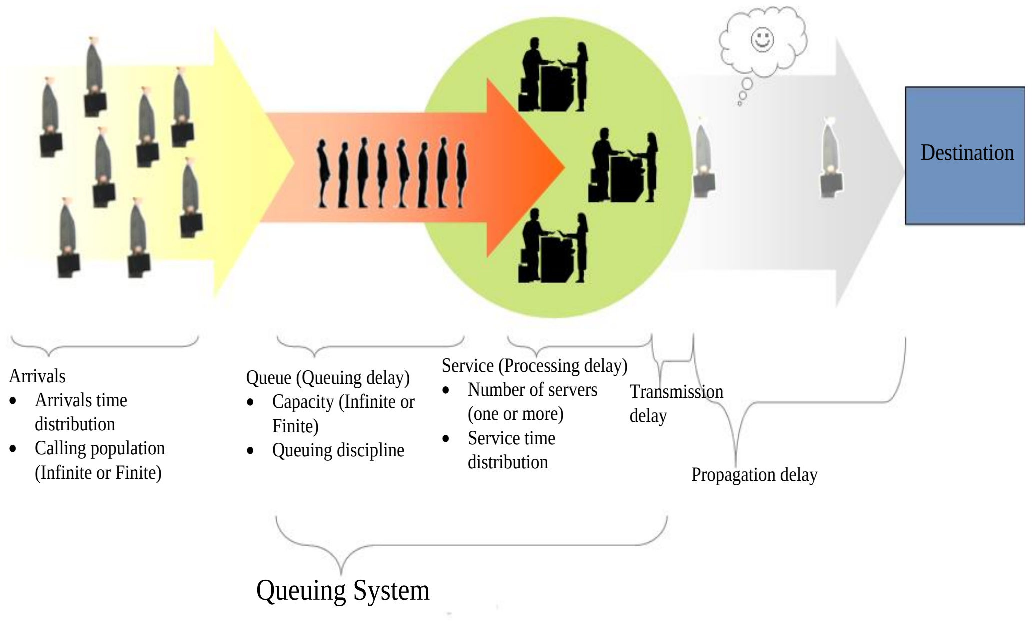

Destination
Arrivals
• Arrivals time distribution
• Calling population (Infinite or Finite)
Queue (Queuing delay)
• Capacity (Infinite or Finite)
• Queuing discipline
Service (Processing delay)
• Number of servers (one or more)
• Service time distribution
Transmission delay
Propagation delay
Queuing System

Figure 3-1: Queuing System [71]

The queuing delay: This is the time spent by a packet in a queue before it is transmitted. It is dependent on the number of packets ahead in the queue and the rate that they are being processed out of the queue.

The processing delay: This is the time interval between a packet arriving at the head of a queue and when the packet is assigned to a particular outgoing link for transmission. The packet processing delay is proportional to the packet size.

The transmission delay: This is the time interval between the transmission of the first and the last bit of a packet; it is constant when the packet sizes are constant.

The propagation delay: This is the time taken for a packet to traverse a particular link. It is the time interval when the receiver receives the first and the last bit of a packet sent by a source. It is proportional to the link distance i.e. the distance between the transmitter and receiver. The propagation delay will remain constant when the propagation speed and the link distance are constant; other factors, which affect the propagation delay of a packet, are the SNR, as well as link capacity.

The delay component does not take into account the re-transmission of packets on a link due to transmission error and various other reasons. The processing and propagation delay are independent of the available traffic in the network [28].

3.2.1. Queuing System

A queuing system can be denoted by a special notation called the Kendall’s notation [72]; the notation has the form: A/S/C/K

Where

‘*A*’ is the probability distribution of the inter-arrival process

‘*S*’ the probability distribution of service times, or the packet sizes

‘*C*’ the numbers of servers or capacity

‘*K*’ the size of the system capacity (which includes the servers)

The commonly used symbols for A and S are:

‘M’ represents a memory-less process similar to Poisson process [73] also referred to as an exponential distribution where M represents the Markov Chain [72].

‘D’ represents a deterministic distribution (also referred to as constant arrival process)

‘G (or G1)’ for general distribution (can also be defined by the mean and variance)

At infinite system capacity ($K=\infty$), the notation commonly used becomes A/S/c. An example of commonly used queuing models are M/M/1, M/G/1 and G/M/1 [72]. However, this section focuses on M/M/1 and the M/G/1 queuing models as they describe the queue behaviour of the applications considered in this chapter.

3.2.1.1. M/M/1

The M/M/1 systems consist of a single queuing system with a single server (in a communication context a single buffer with a single output link) as shown in Figure 3-2. In this queuing model, packets arrive at the queue according to a Poisson process with a rate λ ; the probability distribution of the time required to process a packet (also known as service time) is exponential with a mean of $\frac{1}{\mu}$ (where μ is the service rate).

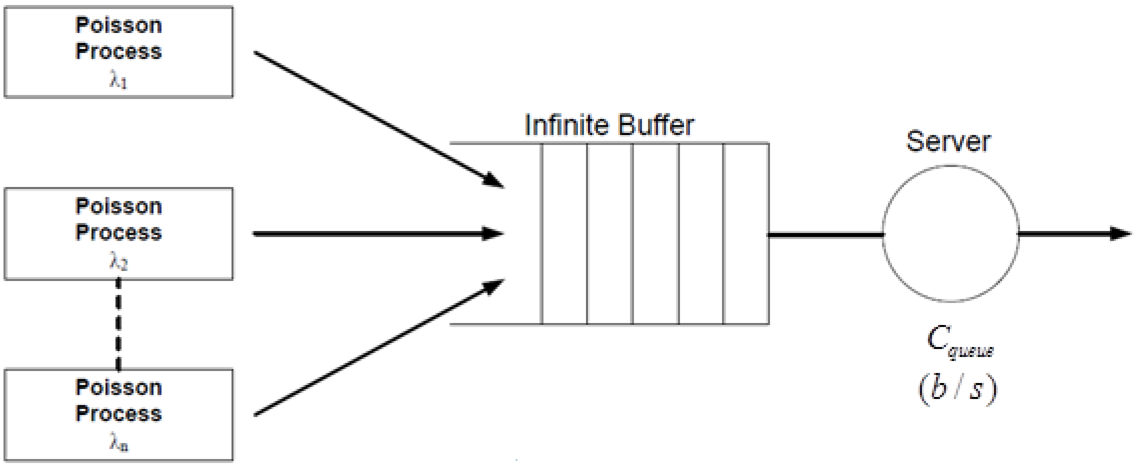


Figure 3-2: M/M/1 Queuing Model [51]

For an M/M/1 queuing system, process $\{N(t)|t \geq 0\}$ can be expressed by a continuous time Markov chain [28]. The process can be analysed using a distance-time Markov chain. The time interval [0, t] is partitioned into time slots $0, \partial, 2\partial, \ldots, k\partial$ and let $N_k = N(k\partial)$ be the number of packets in the queue at time $k\partial$. Then $\{N(t)|t \geq 0\}$ is converted into a process $\{N(t)k = 0,1,2,..\}$, this can be expressed by a discrete-time Markov chain. The flow diagram for the M/M/1 model is shown in Figure 3-3.

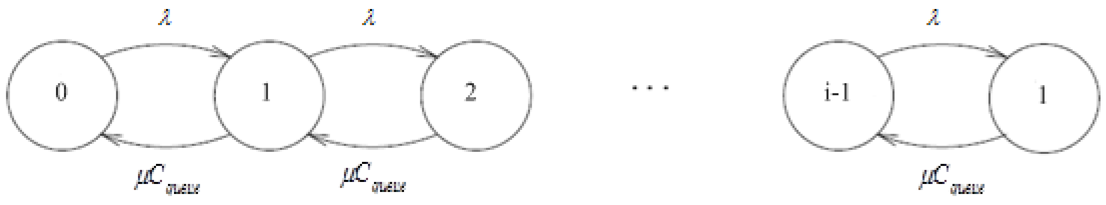


Figure 3-3: Flow diagram for M/M/1 Model

Let 'i' be the state of having i packets in the queue. The probability of the transition from i to $i+1$(an arrival) is λ and from $i+1$ to i (a departure) is μC_{queue}. The number of transition per unit time from i to $i+1$ is equal p_i, this is the fraction of the time the system is in the state times, the rate of arrival occurs when the system is in the state i. The global balance principle states that for each set of states i, the flow out of the set i is equal to the flow of the set. If we apply this principle to state i, it will present a simple relation: $\lambda p_{i-1} = p_i \mu C_{queue}$ for $(i = 1,2,\ldots)$, this can be used recursively so it may be shown that $p_i = \rho^i p_0$ where, $(i = 0,1,\ldots)$, after normalisation $p_i = (1-\rho)\rho^i$ where $(i = 0,1,\ldots)$ and ρ is the traffic intensity and is represented by equation (3-1). The average number 'N' of packets in the system and the average queuing delay can be derived from the equilibrium probabilities. The average number 'N' of packets in the system is given by equation (3-2) [51].

$$\rho = \lambda / \mu C_{queue} \tag{3-1}$$

$$N = \sum_{i=0}^{\infty} i p_i = \frac{\rho}{1-\rho} \tag{3-2}$$

By applying Little's law to equation (3-2), the average queuing delay 'T' of packets in the system is given by equation (3-3) [51].

$$T = \frac{N}{\lambda} = \frac{\rho}{\lambda(1-\rho)} = \frac{1}{\mu C_{queue} - \lambda} \tag{3-3}$$

The average waiting time '*W*' of a packet in the queue is given by equation (3-4).

$$W = T - \frac{1}{\mu C_{queue}} = \frac{1}{\mu C_{queue} - \lambda} - \frac{1}{\mu C_{queue}} = \frac{\rho}{\mu C_{queue} - \lambda} \qquad (3\text{-}4)$$

3.2.1.2. M/G/1

Assuming a Poisson arrival process with an arrival rate of λ message/s, but the packet size is generally distributed and not exponential as in the case of the M/M/1 system (has a Poisson arrival process and an exponential packet size). The M/G/1 system does not have a general closed-form distribution for the number of packets in the queue in steady state, and that is a drawback to the system. Its generalisation is based on average values; meaning for an average number of transaction in the queue, it gives a more general solution and the application of Little's Theorem provides the corresponding result for the average time spent in the queue. Collectively, these results are known as Pollaczek-Khinchin (P-K) formula [28]. Suppose each packet is serviced according to the order it arrives at the head of the queue just as in FIFO system, and the service time is X_i for the i^{th} arrival. Assuming identically distributed random variables ($X_1, X_2, ...$)that are mutually independent with independent arrival times.

Let:

$\overline{X} = E\{X\} = \frac{1}{\mu C_{queue}}$ =average service time

$\overline{X}^2 = E\{X^2\} = \frac{1}{(\mu C_{queue})^2}$ = the Second moment of service time

$$W = \frac{\lambda \overline{X}^2}{2(1-\rho)} \qquad (3\text{-}5)$$

The expected waiting time for the packet in the queue is '*W*' and $\rho = \frac{\lambda}{\mu C_{queue}} = \lambda \overline{X}$ is the utilisation factor of the queue, defined in terms of the packet arrival rate λ and the average service time. Considering the equation (3-5) [51], the total waiting time in the queue is given by equation (3-6):

$$T = \overline{X} + \frac{\lambda \overline{X}^2}{2(1-\rho)} \qquad (3\text{-}6)$$

Applying Little's formula to equation (3-5) and equation (3-6), the expected number of packets in the queue 'N_Q' and the number of packets in the system '*N*', can be calculated using equation (3-7) and equation (3-8) [51], respectively.

$$N_Q = \frac{\lambda^2 \overline{X}^2}{2(1-\rho)} \qquad (3\text{-}7)$$

$$N = \rho + \frac{\lambda^2 \overline{X}^2}{2(1-\rho)} \qquad (3\text{-}8)$$

When the service time is exponentially distributed, as in M/M/1 system, $\overline{X}^2 = \frac{2}{(\mu C_{queue})^2}$, substituting in equation (3-5) will result in equation (3-9) which is similar to equation (3-4):

$$W = \frac{\rho}{\mu C(1-\rho)} \qquad (3\text{-}9)$$

When the service time is identical for all packets as in M/D/1 systems (where D means deterministic), $\overline{X}^2 = \frac{1}{(\mu C_{queue})^2}$, substituting $\overline{X}^2$ in equation (3-5) will give the equation (3-10).

$$W = \frac{\rho}{2\mu C_{queue}(1-\rho)} \qquad (3\text{-}10)$$

3.2.2. Multi-hop MANET delay algorithm

Mobile nodes in MANET can communicate with other nodes outside their transmission region through a relay or intermediate nodes. Assuming T_{sp} is the packet inter-arrival time for a simple case of a point-to-point network as shown in Figure 3-4. Thus a feedback signal is transmitted once every T_{sp}, is exchanged between the system components in the form of information packet through the wireless network with a latency of $\tau_{latency}$ [75].

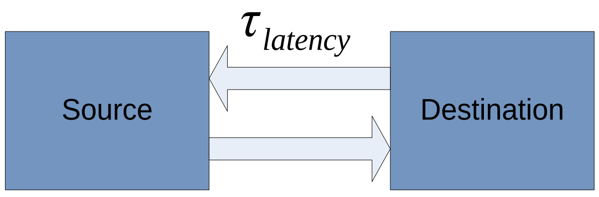


Figure 3-4 Point-to-Point communication with latency [75]

A source node will transmit a packet every T_{sp} and the total time taken to transverse the network is dependent on the queue and link capacity. If '*n*' is the total number of nodes in a network, the maximum number of hops to reach a destination node is *n-1*. Hence, the delay for a single link can be calculated from equation (3-11) [75].

$$\tau_{latency} = \tau_{queue} + \tau_{tx} + \tau_{prop} \qquad (3\text{-}11)$$

Where $\tau_{queue} = \tau_{proc} + \tau_{waiting}$, τ_{proc} is the processing delay, $\tau_{waiting}$ is the waiting time in the queue, τ_{tx} is the transmission delay and τ_{prop} is the propagation delay. A transmitted packet in MANET may transverse the network through intermediate nodes; thus the total delay is the summation of all the delay of individual links as shown by equation (3-12) [75].

$$\tau_{latency} = \sum_{i=0}^{n-2} (\tau_{queue_i} + \tau_{tx_i} + \tau_{prop\,i}) \qquad (3\text{-}12)$$

When the packet size and packet inter-arrival rate are constant, the queuing model is referred to as D/D/1. The queue size for D/D/1 system is given by equation (3-13):

$$\overline{P}_{queue} = \rho \qquad (3\text{-}13)$$

Where ρ is the traffic intensity parameter given by equation (3-1) and $C_{queue} = \frac{d_r}{\mu^{-1}}$ the network capacity; Hence $\rho < 1$, indicates low traffic levels (non-congested network), whilst $\rho \geq 1$, indicates congestion has occurred and $\overline{P}_{queue}$ will increase with a rate that is related to the value of ρ . One packet arrives every T_{sp} at a queue, and it is serviced after one processing time τ_{proc} by the queue. Hence, τ_{proc} is calculated using equation (3-14).

$$\tau_{proc} = \frac{1}{\mu C_{queue}} \qquad (3\text{-}14)$$

Assuming 'n' sources with identical T_{sp}, then 'n' packets with the same packet size will arrive the queue simultaneously every T_{sp}, each packet will be queued and served after a minimum delay of τ_{proc} and multiples thereof, as illustrated in Figure 3-5.

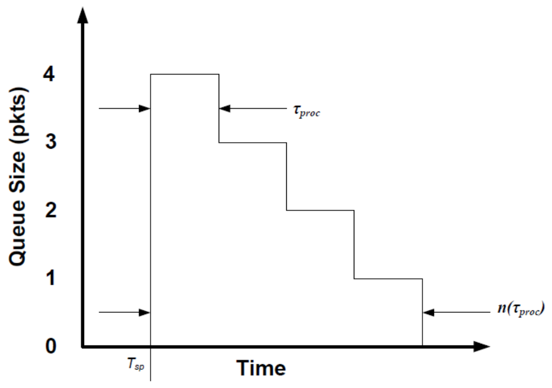


Figure 3-5: Number of packets in a queue over time [49]

The n^{th} packet delay will be $n * \tau_{proc}$; to achieve $\tau_{waiting} = 0$, it is imperative to schedule packets to arrive the queue separately, hence individual packets will only be queued for τ_{proc} . This can be achieved by staggering the start time of the source so that the i^{th} source incurs a start time given by equation (3-15) [75]. This is effective for CBR control traffic.

$$\tau_{start_i} = (i-1)\frac{T_{sp}}{n}, \; for\, i = 1\, to\, n \qquad (3\text{-}15)$$

Assuming all T_{sp} are constant, then equation (3-15) is calculated from $i = 1$ to n. There is a limit to the number of packets the queue can process in time T_{sp}. In other words, if the inter-arrival time T_{sp} is less than the processing time (τ_{proc}), no queuing will occur. The limit can be calculated using equation (3-16) [75].

$$n \leq \frac{T_{sp}}{\tau_{proc}} \tag{3-16}$$

Congestion will occur if the condition given by equation (3-16) is not met. This means there is not enough time to clear the queue before the next arriving packet. The maximum delay suffered by a packet in the queue when equation (3-16) condition is met can be given by equation (3-17). The queue size is given by equation (3-18).

$$\bar{\tau}_{queue} = \frac{1}{n} \lim \sum_{i=0}^{n} [(i)(\tau_{proc})_i] \tag{3-17}$$

$$\bar{p}_{queue} = \frac{1}{n} \lim \sum_{i=0}^{n} [(i)(\rho)_i] \tag{3-18}$$

3.2.3. Simulation Model for CBR Traffic

Consider a D/D/1 queueing model that consists of a single source, queue and sink. The source generates a constant packet of size μ^{-1} 1,024 bit/packet every T_{sp} 0.5s, giving a λ of 2pkts/s per source which is typical for control applications. This models CBR traffic, the service rate 'C_{queue}' is 10,000 bits/s as shown in Table 3-1. The simulation duration is 1,000s and the number of source 'n' was increased from 1 to 4, the corresponding traffic intensity ρ is calculated using equation (3-1) and it is shown in Table 3-2.

Table 3-1: Input Parameters for CBR traffic Simulation Model

$T_{sp}(s)$	μ^{-1}(bits/s)	C_{queue}(bits/s)	μC_{queue}(pkts/s)
0.5	1,024	10,000	9.766

Table 3-2: Traffic intensity for various numbers of sources

N	1	2	3	4
λ(pkts/s)	2	4	6	8
ρ	0.205	0.410	0.614	0.819

The queuing delay 'τ_{queue}' and queue size can be calculated from equation (3-17) and equation (3-18), respectively. The average queuing delay as measured in OPNET and the corresponding theoretical delay from equation (3-17) are shown in Table 3-3 and Figure 3-6, respectively. The delay is constant for all simulation time because of the constant packet size and inter-arrival rate. The theoretical and the measured simulation delay are a match, thus validating equation (3-14) and the OPNET model performance for the D/D/1 model.

Table 3-3: Average Queuing delay for CBR traffic

ρ	Calculated(s) Equation (3-17)	Measured(s) (OPNET)
0.205	0.102	0.102
0.410	0.154	0.154
0.614	0.205	0.205
0.819	0.256	0.256

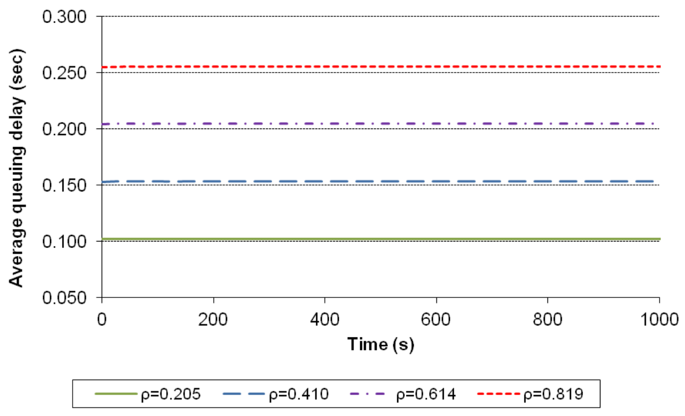


Figure 3-6: Average Queuing delay for CBR traffic at $\boldsymbol{\rho < 1}$

The measured queue size is also identical to the calculated queue size as shown in Table 3-4 and Figure 3-7, thus validating equation (3-18). The network is uncongested as the traffic intensity $\boldsymbol{\rho < 1}$ for all 'n' as shown in Table 3-2.

Table 3-4: Average Queuing size for CBR traffic

ρ	Calculated(pkts) Equation (3-18)	Measured(pkts) (OPNET)

0.205	0.205	0.205
0.410	0.614	0.614
0.614	1.229	1.229
0.819	2.048	2.048

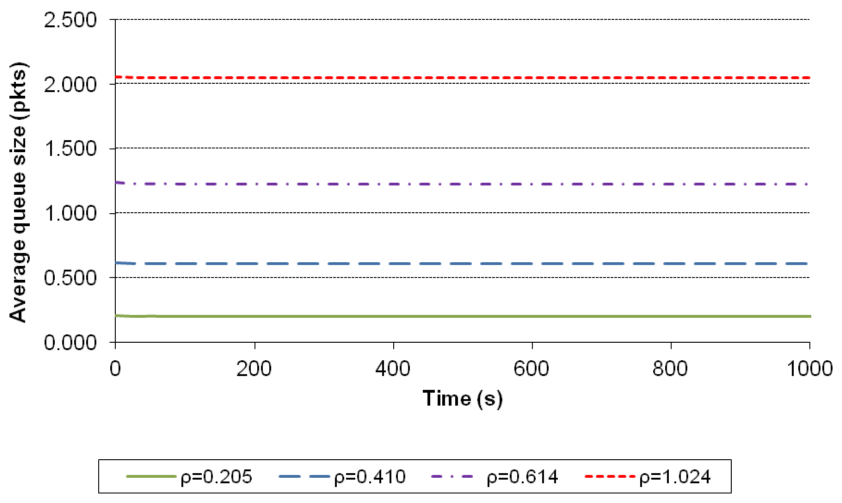


Figure 3-7: Average Queue size for CBR traffic at $\boldsymbol{\rho < 1}$

When 'n' was increased to 5 ($\rho = 1.024$), the network is congested as stated in section 3.2.2. The queue can no longer cope with the intensity of packets arrival, thus there would be a gradual build-up of packets in the queue. The queuing delay and queue size increases linearly with time as shown in Figure 3-8 and Figure 3-9. The gradient increases as the network tends towards congestion and becomes infinity when the network is congested. The theoretical queuing delay and queue size cannot be calculated because the condition for equation (3-17) and equation (3-18) to be valid is not met as $\rho > 1$.

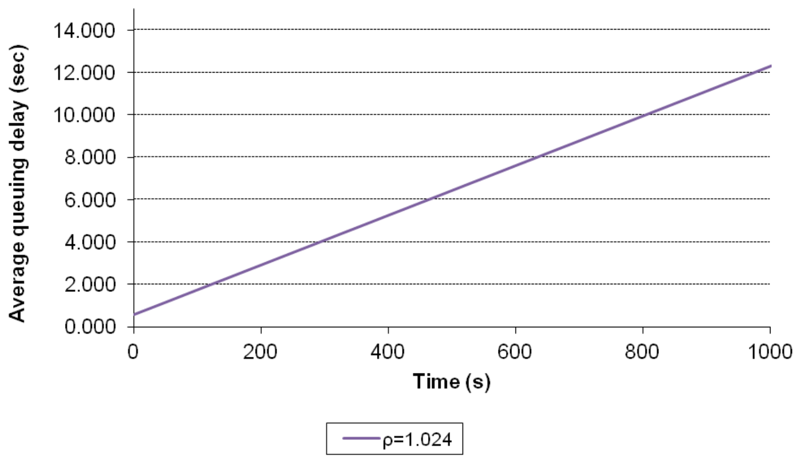


Figure 3-8: Average Queuing delay for CBR traffic at $\boldsymbol{\rho > 1}$

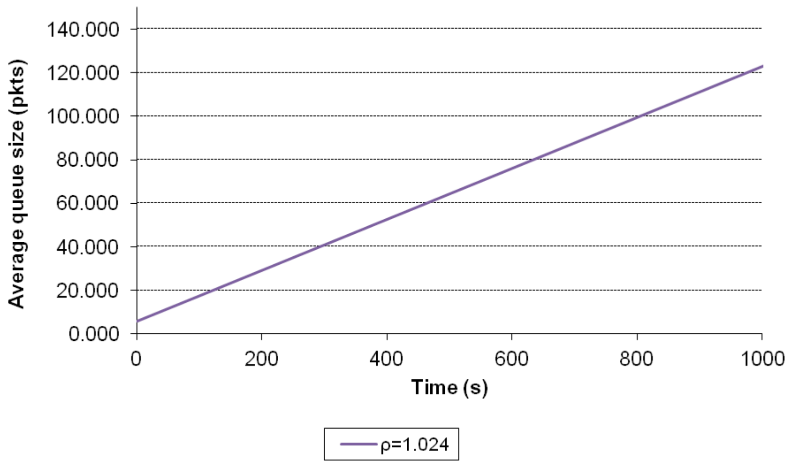


Figure 3-9: Average Queue size for CBR traffic $\boldsymbol{\rho > 1}$

3.3. Simulation model for congested and uncongested queue

Section 3.1 and 3.2 presented the basic concepts of queuing theory, congestion, and congestion control mechanisms. This section builds on this knowledge to model real-time traffic (Voice) and non-real-time traffics (Internet Browsing and Email) in a network under different traffic conditions using OPNET. Initially each type of traffic is modelled separately and finally they are modelled simultaneously in a single network. Initially the network is not congested ($\rho < 1$), the network load is increased until congestion occurs ($\rho \geq 1$) and the behaviour of the traffic is observed, this is compared to the theoretical predictions; and finally, the performance of a baseline model, a MANET model without OSI layer and a MANET model with the OSI layer will be analysed. This will also be compared to the theoretical predictions.

Three applications classified under real and non-real-time were considered in this section. Real and non-real time traffic have been described in section 1.1. These applications were considered because they are the most commonly used and the nature of the different application presents different queuing behaviour, which is important for performance analysis.

The network load is increased until the theoretical prediction ($\rho \geq 1$) where congestion occurred, this is done by increasing the number of sources. Simulation measurements such as queue delay and queue size were analysed and compared to the theoretical prediction in order to justify the simulation model. Other statistics such as traffic received is also measured.

If the arrivals of packets from different sources are scheduled, so the packets arrive at separate times, where the packet inter-arrival time is greater than packet processing time at the queue; individual packets will only be queued for a time equal the packet processing time [75]. The performance of the network is analysed under two conditions; firstly, packets from multiple sources arriving the queue at the same time and secondly scheduling the packets from multiple sources to arrive the queue at separate time. The formal is referred to as the unscheduled, whilst the latter is the scheduled.

3.3.1. Traffic modelling

A brief description of each application and the parameters required to model them is provided in the next section.

3.3.1.1. Voice Calls

This is a real-time application; the acceptable delay targets according to ITU are shown in Table 2-1. According to the ITU the performance target for the one-way delay of a voice call is preferably 150ms but can tolerate a delay of up to 400ms. The delay variation should be less than 1ms whilst the packet loss should be less than 3%. The packet size is exponentially distributed with an average call duration of 120s [76]. The probability density function for exponentially distributed packet with a mean value of $\frac{1}{\delta}$ is shown in equation (3-19), where δ = rate of distribution at $x_0 > 0$. An exponential distribution, models the time until something happens in a continuous time stochastic process. This traffic type is similar to the VBR traffic profile described in Section 3.7.3. The arrival rate of Voice is a Poisson process; this is similar to a M/M/1 queuing system as described in section 3.2.1.1.

$$f_X(x_0) = \begin{cases} \delta e^{\ \delta x_0} \\ 0 \end{cases} \qquad (3\text{-}19)$$

δ = rate of distribution at $x_0 > 0$

3.3.1.2. Internet Browsing

Internet Browsing consists of a sequence of packet calls, during each packet call, several packets may be generated constituting a bursty packet sequence. The burstyness during a packet call is a characteristic feature of packet transmission in the network. Thus, it is vital to consider this phenomenon when modelling this traffic. The modelling of Internet Browsing follows a Pareto burst size distribution (k=1.1, r=81.5 bytes) with a maximum allowable burst size M=66,666 bytes [77]. Where, '*k*' is the shape parameter and '*r*' the location parameter. The behaviour of a bursty traffic profile is explained in section 3.7.3. Equation (3-20) gives the probability distribution function of normal Pareto distribution without a cut-off [78].

$$f(x:k,r) = \begin{cases} \frac{kr^k}{x^{k+1}}, x \geq r \end{cases} \tag{3-20}$$

$mean : E(x) = kr/(k-1) \quad k > 1$ $\qquad$ $Arg\ 1 : location = r > 0$

$Variance : \sigma_x^2 = kr^2/[(k-1)^2(k-2)] \quad k > 2$ $\qquad$ $Arg\ 2 : shape = k > 0$

The packet size for Pareto distributed traffic can be defined as min (P, M), where '*P*' represents the normal Pareto distribution [77]. The packet size distribution is given by equation (3-21) which is a Pareto distribution with a cut-off. The behaviour of this traffic in the queue is similar to the M/G/1 system as described in section 3.2.1.2.

$$f(x:k,r) = \begin{cases} \frac{kr^k}{x^{k+1}}, r \leq x < M \\ \varphi \quad , x > M \end{cases} \tag{3-21}$$

Where φ is the probability that $x > M$ and can be calculated by equation (3-22).

$$\varphi = \int_M^\infty f_X(x)dx = \frac{r^k}{M} , k > 1 \tag{3-22}$$

Therefore, the packet size is given in equation (3-23):

$$\mu = \int_{-\infty}^{\infty} f_n(x)dx = \int_k^{-M} x\frac{kr^k}{x^{k+1}}dx + M\left(\frac{r}{M}\right)^k = \frac{kr - M\left(\frac{r}{M}\right)^k}{k-1} \tag{3-23}$$

Inputting the values for (r=1.1, k=81.5 and M=66,666) given above, the theoretical average burst size for Internet Browsing is calculated using equation (3-23) to give 480 bytes.

3.3.1.3. Email Sessions

According to statistics of Email collected by the Finish University and Research NETwork (FUNET) [79], Email Sessions can be represented by a FUNET model. It can be modelled by a truncated Cauchy Lorenz distribution given by equation (3-24), which is approximated at (ω=0.8, β=1) with a mean value of 830 bytes and a maximum message size of 10kbytes.

The equation (3-24) is simplified to equation (3-25), where ω is the location parameter and β is the scale parameter.

$$f(x:\omega,\beta)=\left(\frac{1}{\pi\beta}\right)\frac{1}{1+\left[\frac{(x-\omega)}{\beta}\right]^2} \qquad (3\text{-}24)$$

This can be simplified to:

$$f(x:\omega,\beta)=\left(\frac{1}{\pi}\right)\left[\frac{\beta}{\beta^2+(x-\omega)^2}\right] \qquad (3\text{-}25)$$

The amplitude or height of the Lorentzian function is given by equation (3-26). The traffic profile of this traffic is bursty as described in section 3.7.3. This can be classified as a M/G/1 queuing model.

$$Amplitude=\frac{1}{\pi\beta} \qquad (3\text{-}26)$$

3.3.2. Simulation parameters

The simulation model is classified into four cases: case 1 to 4, the details of each case is shown in Table 3-5. Each of these cases were sub-divided into '*a*' and '*b*' categories, where '*a*' is unscheduled model whilst '*b*' is the scheduled model. Scheduled models refer to models whose sources start times are staggered according to equation (3-15) (scheduling of multiple sources start time), whilst the unscheduled are non-staggered as explained in Section 3.2.1. Case 1 models Voice traffic, for each source, the packet inter-arrival rate is exponential with a mean outcome of 1s, and the packet size is exponentially distributed with a mean outcome of 1,024 bits. Case 2, Case 3 models Internet Browsing and Email, respectively, whilst case 4 consists of mixed traffic (Voice, Internet Browsing, and Email).

Figure 3-10 shows the queuing model; it consists of a source, queue that models a network of some description and a destination. The source generates packets with a given inter-arrival rate and a given packet size, these packets are processed by the queue and then sent to the destination. The queue uses the OPNET '*acb_fifo'* queuing model, which is the same as FIFO. This model is used because of its capacity to accommodate multiple sources simultaneously. The destination uses the OPNET '*sink*' process model, it receives the packets, records the required statistic before destroying the packet.

Table 3-5: Traffic Modelling (Test Cases)

Case	Application	Distribution: $T_{sp,}$ pkt size (bits)
1 (a,b)	Voice	Exp(1), Exp(1,024)
2 (a,b)	Internet Browsing	Exp(1), Pareto(81.5,1.1)
3 (a,b)	Email	Exp (1), Cauchy (0.8 1)
4 (a,b)	Mixed(one source each): Case 1, Case 2 and Case 3	

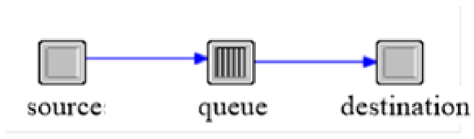


Figure 3-10: A single source queuing model

3.3.2.1. Case 1: Voice Traffic

Consists of a source transmitting an exponentially distributed packet size (μ^{-1}) 1,024 bits with an exponential inter-arrival time 'T_{sp}' of 1s. This model is a Poisson process (M/M/1), therefore each source sends an average of 1 pkts/s. Table 3-6 shows the network parameters Voice, where the data rate (d_r) is 11,000,000 bits/s , thus the service rate 'C_{queue}' is calculated as $C_{queue} = \frac{d_r}{\mu^{-1}}$, which is 10,742.18 bits/s. The queue can only process 10,742.18 bits of traffic in a second.

Table 3-6: Case 1a,b: Network Parameters for Voice

$T_{sp}(s)$	d_r(bits/s)	μ^{-1}(bits/s)	C_{queue}(bits/s)	μC_{queue}(pkts/s)
1	11,000,000	1,024	10,742.18	10.49

The number of sources 'n' is increased to the point where congestion occurred ($\rho \geq 1$). The traffic intensity 'ρ' is equal to 1 when the capacity of the queue is equal to the traffic offered, well below this value there should be no congestion, as ρ reaches 1 congestion will start to occur and above this value congestion increases. Congestion occurred ($\rho \geq 1$) for case 1 when the number of sources becomes greater than 10, the theoretical queuing delay equation (3-4) is undefined at this point as $\mu C_{queue} > \lambda$. The OPNET simulation duration was 7 hours with a start time of 0.02s. The simulation is run for n = 1, 5, 8, 9, 10 and 11 sources, the measured queuing delay from simulation is compared to the theoretical values as shown in Table 3-7. The traffic intensity 'ρ' is calculated for each value of n and is also shown. The network gradually becomes congested as the traffic intensity 'ρ' increases. Thus, the traffic intensity is directly proportional to the network load. At n = 11, the queue is congested as (ρ >1). The queuing delay for the staggered and non-staggered start time (Case 1a,b) is shown in Table 3-7.

According to Table 3-7, there is a close match between the average queuing delays measured from simulation to the theoretical for the non-staggered start time, however, the differences widen as the traffic intensity increases. At n=1, there is a single source, single queue and sink thus packets sent to the sink are passed to the queue with a minimum amount of delay according to equation (3-4). As 'n' increases, the packet waiting time in the queue also increases, thus increasing the overall delay as can be noticed from Table 3-7. The theoretical queue delay for $n = 11$ is undefined because of congestion, ρ is greater than one, thus, the condition of equation (3-16) is no longer satisfied. When the start time of 'n' sources is staggered according to equation (3-15), the resulting delay is shown in Table 3-7.

Table 3-7: Case 1a,b: Average Queue delay for Voice

n	ρ	Calculated (s) Equation (3-4)	Start Time			
			Non Staggered (Case1a)		Staggered (Case1b)	
			Measured (s)	±Diff (%)	Measured (s)	±Diff (%)
1	0.095	0.105	0.106	0.952	0.106	0.952
5	0.477	0.182	0.183	0.559	0.180	1.099
8	0.763	0.402	0.409	1.741	0.402	0.000
9	0.858	0.670	0.693	3.433	0.659	1.642
10	0.953	2.040	2.500	22.549	1.785	12.50
11	0.049	N/A	271.538	N/A	295.311	N/A

The queuing delay for the staggered start time is also shown in Table 3-7, there is a percentage improvement of 1.64, 1.71, 4.91 and 28.6% in the delay for $n = 5, 8, 9$ and 10 respectively. The percentage margin of error between the measured queuing delay and the theory (diff (%)) is also shown in Table 3-7. The staggered model performed worse than the non-staggered at *n*=11 because packet arrives the queue at separate time. The performance of the staggered model improves as the network load increases, until congestion occurred. When the network is congested, the queuing delay tends towards infinity.

The queue size of a network is directly proportional to the network delay, therefore when the queue size is low the delay is also low and vice versa as shown in Table 3-8. The staggered starts showed noticeable improvement (decrease) in the queue size thus resulting in a reduced queuing delay. The measured queue size (OPNET) was close to the theoretical (equation (3-2)) queue size. The margin of error for the measured and theoretical increases as n increases from 5 to 10 for the staggered start time. At $n = 11$, the queue size is theoretically undefined because of the rate of congestion.

Table 3-8: Case 1a,b: Average Queue Size for Voice

n	ρ	Calculated (bits) Equation (3-3)	Start Time			
			Non Staggered (Case1a)		Staggered (Case1b)	
			Measured (bits)	±Diff (%)	Measured (bits)	±Diff (%)
1	0.095	107.52	109.568	1.905	108.544	0.952
5	0.477	933.888	947.2	1.425	931.84	0.219
8	0.763	3,296.256	3,420.16	3.759	3,336.192	1.212
9	0.858	6,187.008	6,573.056	6.240	6,184.96	0.033
10	0.953	20,763.65	26,066.94	25.541	18,545.66	10.68
11	0.049	N/A	3,068,619	N/A	3,321,565	N/A

The traffic received is shown in Table 3-9. The staggered source received more traffic at $n = 1,5,9$ and 11. The variance for the delay, queuing size and traffic received increases as the number of sources increases, thus the queue becomes more unstable and tends towards congestion.

Table 3-9: Case 1a,b: Average Traffic Received for Voice

n	ρ	Start Time	
		Non Staggered (Case1a)	Staggered (Case1b)
		Measured (bits)	Measured (bits)
1	0.095	25798.29	25986.48
5	0.477	129306.5	129571.8
8	0.763	208033.1	206870.1
9	0.858	232406.3	232510.8
10	0.953	259972.1	257263.7
11	0.049	270513	270609.4

3.3.2.2. Case 2: Internet Browsing Traffic

The truncated effect of modelling the Internet Browsing traffic as explained in section 3.3.1.2 is neglected. This traffic is modelled using a scale parameter 'r'of 81.5 and a shape parameter 'k' of 1.1 according to equation (3-20), the average packet size is (E(x)= μ^{-1}) 897 bits with an exponential inter-arrival time T_{sp} of 1s , therefore λ becomes 1 pkts/s for each source. The service rate $C_{queue} = \frac{d_r}{\mu^{-1}}$ is 12,269.94 bits/s as shown in Table 3-10. The number of sources 'n' is increased until the congestion occurs ($\rho \geq 1$), the corresponding value of ρ for all 'n' is shown in Table 3-11. Congestion starts to occur when the number of Internet Browsing sources reaches 13 and 14.

Table 3-10: Case 2: Common Input Parameters for Internet Browsing

$T_{sp}(s)$	d_r(bits/s)	μ^{-1}(bits/s)	C(bits/s)	μc
1	11,000,000	897	12,269.94	13.67

As shown in Table 3-11, the queuing delay improves as the source is staggered for all 'n'. The traffic intensity 'ρ' increases as the number of sources 'n' increases and thus congestion occurred at $n = 14$. The inconsistences in the measured delay as shown in Table 3-11, is caused by the bursty nature of the traffic. As mentioned earlier in section 2.4.1.3, bursty traffic are characterised by sudden change on the data rate within a short period of time. This sudden change is difficult for a queue to handle as it cannot readily predict the structure of the flow. The queue with staggered source time coped better with bursty traffic than the queue with non-staggered source time.

Table 3-11: Case 2a,b: Average Queue delay for Internet Browsing

n	ρ	Calculated (s) Equation (3-6)	Start Time			
			Non Staggered (Case1a)		Staggered (Case1b)	
			Measured (s)	±Diff (%)	Measured (s)	±Diff (%)
1	0.073	0.076	0.365	380.263	0.088	15.789
5	0.366	0.094	5.047	5,269.1	2.792	2870.213
8	0.585	0.096	2.323	2,319.7	0.876	812.500
12	0.877	0.336	14.469	4,206.2	3.501	941.964
13	0.950	0.784	334.608	42,580	6.862	775.255
14	1.022	N/A	15.436	N/A	5.376	N/A

The measured simulation queuing delay does not match the theoretical calculated delay according to equation (3-6). Internet Browsing is very bursty as compared to Voice; thus they exhibit a varying queuing behaviour that is difficult to predict.

The theoretical estimated queue sizes according to equation (3-8) vary from the measured queue size for all values of n as shown in Table 3-12. The traffic build up in the queue decreases when the sources were staggered thus reducing the chances of congestion occurring. The staggered model improves the performance of the network. Mechanisms such as traffic shaping and scheduling can be used to simplify the processing of bursty traffic in the queue in order to improve the queue performance.

The traffic received in bits is shown in Table 3-13; the scheduled model performed better for $n = 1{,}5{,}8$ and 14, the performance degraded as compared to the unscheduled for $n = 12$ and 13.

Table 3-12: Case 2a,b: Average Queue Size for Internet Browsing

n	ρ	Calculated (bits) Equation (3-7)	Start Time			
			Non Staggered (Case1a)		Staggered(3-4) (Case1b)	
			Measured (bits)	±Diff (%)	Measured (bits)	±Diff (%)
1	0.073	68.172	503.403	638.431	92.495	35.679
5	0.366	392.886	45,049.09	11,366.199	28,555.300	7,168.08
8	0.585	571.389	12,662.73	2,116.131	8,174.758	1,330.68
12	0.877	3,619.395	146,547.2	3,948.942	40,128.190	1,008.69
13	0.950	9,128.769	2,755,293.	30,082.525	113,790.500	1,146.50
14	1.022	N/A	163,322.70	N/A	69,133.350	N/A

Table 3-13: Average Traffic received for Internet Browsing

n	ρ	Start Time	
		Non Staggered (Case2a)	Staggered (Case2b)
		Measured (bits)	Measured (bits)
1	0.073	14,323.27	15,016
5	0.366	76,912.9	80,878.13
8	0.585	123,207.9	137,357.5
12	0.877	331,425	279,493.4
13	0.950	212,350.4	195,884.9
14	1.022	228,683.9	244,651.4

3.3.2.3. Case 3: Email Traffic

Case 3 models Email traffic by using a Cauchy distributed packet size with an exponential inter-arrival time T_{sp} of 1s, the theoretical packet size and traffic intensity (ρ) cannot be determined because Cauchy model have no mean. Therefore, simulation results will be relied upon to monitor the network state (congested and non-congested). The implementation of the Cauchy distribution is shown in Appendix A. The service rate C of the queue is 18,000 bits/s. The simulation was carried out for $n = 9 - 12$; the number of source ‘n’ was varied until congestion occurred. The average queuing delay for $n = 9$ and 10 is shown in Figure 3-11. The graph shows that the network is uncongested at $n = 9$ and 10, however, the network tends towards congestion when $n = 11$, it becomes congested at $n = 12$ as shown in Figure 3-12. When the network is congested, the queue is close to or beyond its capacity. The queuing delays increase almost linearly with time and tend towards infinity. The scheduler performed a little better than the non-scheduler for $n = 9$ between the simulation times 0.02 to 8,000s as shown in Figure 3-11, the performance of the scheduled and non-scheduled becomes similar for the remaining simulation duration. The scheduled network show a better performance than the non-scheduled networks at $n = 10$. The average queuing delay before applying scheduling for $n = 9, 10, 11$ and 12 is 0.391, 0.810, 9.033, and 409.511s respectively; after applying scheduling, the average queuing delay becomes 0.368, 0.674, 5.771, and 397.725s respectively. There is an improvement in the queuing delay after the application of scheduling. The percentage improvement for $n = 9, 10, 11$ and 12 after scheduling is approximately 6, 17, 36, and 3% respectively. The performance of the scheduler improves as the network load increases until congestion occurs, after which the performance degrades.

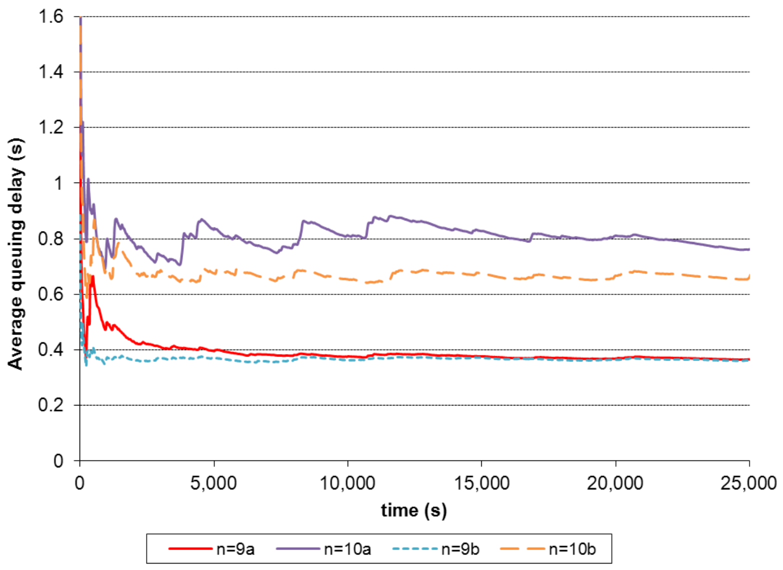


Figure 3-11: Case 3: Average Queue delay for Email (n= 9 and 10)

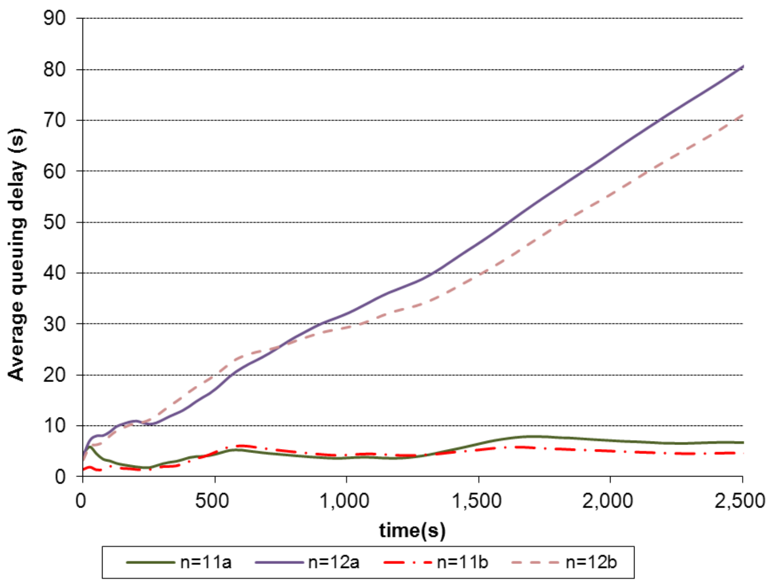


Figure 3-12: Case 3: Average Queue delay for Email (n=11 and 12)

The queue size for $n = 9$ and 10 is shown in Figure 3-13 whilst that for $n = 11$ and 12 is shown in Figure 3-14 for scheduled and non-scheduled conditions. The queue sizes exhibits similar behaviour to the queue delay in Figure 3-11 and Figure 3-12. This is because the queue size is directly proportional to the queuing delay. The network is congested at $n = 12$ as shown in Figure 3-14, thus it increases linearly with the simulation time. The queue size for $n = 11$ is presented on the primary (left) y-axis whilst that of $n = 12$ is on the secondary (right) y-axis because of the large difference in the scale.

The average queue size for $n = 9, 10, 11$ and 12 for scheduled and non-scheduled are shown in Table 3-14. There is a percentage reduction of 7, 17, 36, and 3% in the queue size after scheduling for $n = 9, 10, 11$ and 12 respectively. The performance of the queue improves as the network load increases until congestion occurred.

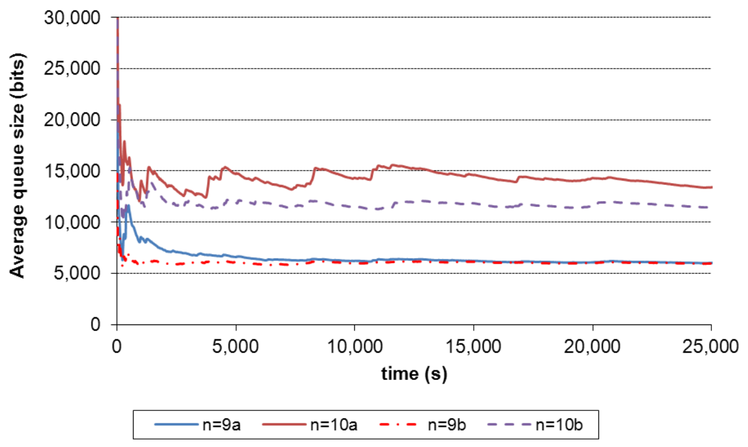


Figure 3-13: Case 3: Average Queue Size for Email (n=9 and 10)

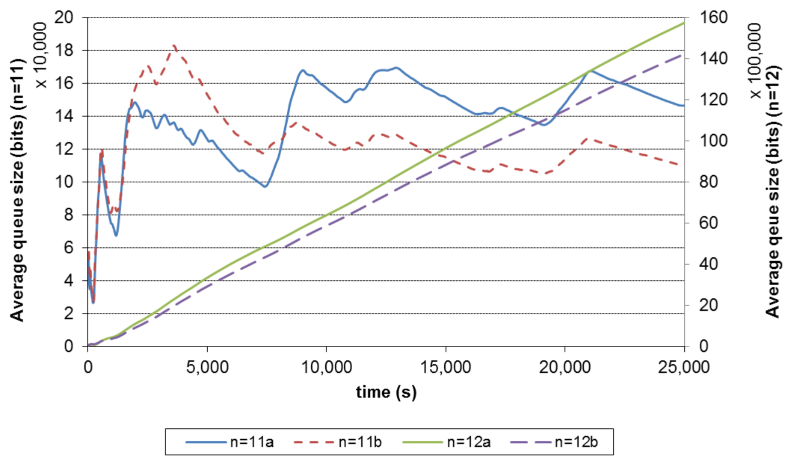


Figure 3-14: Case 3: Average Queue size for Email (n=11 and 12)

Table 3-14: Case 3a,b: Average Queue size for Email

n	Start Time	
	Non Staggered (Case3a)	Staggered (Case3b)
	Measured (bits)	Measured (bits)
9	6,507	6,065
10	14,299	11,826
11	163,083	104,154
12	7,876,150	7,642,714

The average traffic received in bits for the scheduled and non-scheduled network is shown in Table 3-15. The scheduled network received less traffic than the non-scheduled network for most n except at $n = 9$.

Table 3-15: Case 3a,b: Average traffic received for Email

n	Start Time	
	Non Staggered (Case3a)	Staggered (Case3b)
	Measured (bits)	Measured (bits)
9	363,160	363298
10	402,856	402411
11	443,588	442432
12	453,589	453542

3.3.2.4. Case 4: Mixed Traffic

Case 4 combined a single source of Case 1-3 into a single queue as shown in Figure 3-15. A filter is created in C to filter various packets to their respective destinations/sink. The process model and the C code are shown in Appendix B. The service rate 'C_{queue}' is 20,000bits/s, this is arbitrarily chosen because it cannot be theoretically determined.

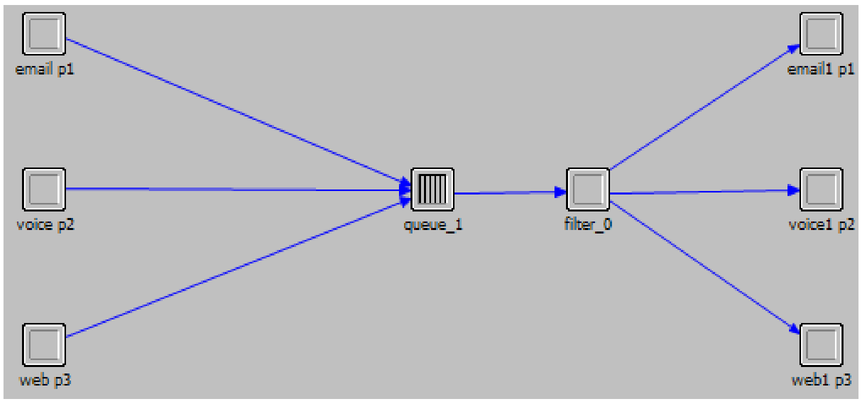


Figure 3-15: Case 4: Simulation model

The queuing delay for Voice, Internet Browsing and Email is shown in Figure 3-16, Figure 3-17 and Figure 3-18 respectively. The results labelled case4Voice_1a, case4web_1a and case4Email_1a are unscheduled whilst those labelled case4web_1b, case4Voice_1b and case4Email_1b are scheduled. The average queuing delay for Voice and Internet Browsing shows a small margin of error when compared to the measured in section 3.3.2.1 and section 3.3.2.2 for $n = 1$. The average end-to-end delay before scheduling for Voice, Internet Browsing and Email traffic are 0.112, 0.079 and 0.141s respectively, after scheduling it becomes 0.084, 0.053 and 0.113s respectively. The scheduling model performs better as it reduces the average end-to-end delay for all traffic. The end-to-end delay did increase significantly after simulation time 3,000s for all traffic as shown in Figure 3-16 to Figure 3-18. The average percentage improvement of the queuing delay for the scheduler as compared to the non-scheduler for Voice, Internet Browsing and Email are 25, 33 and 20% respectively.

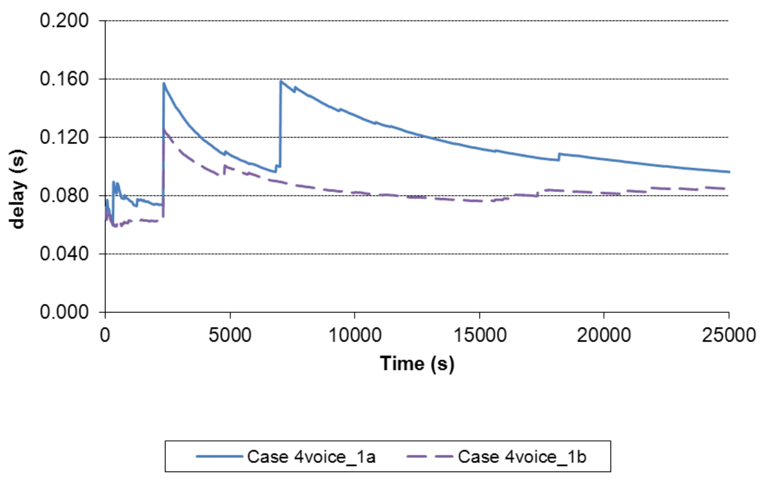


Figure 3-16: Case 4: Average Queue delay for Voice

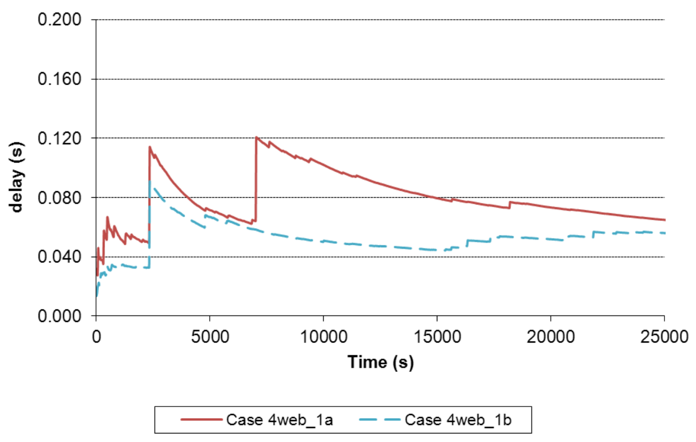


Figure 3-17: Case 4: Average Queue delay for Internet Browsing

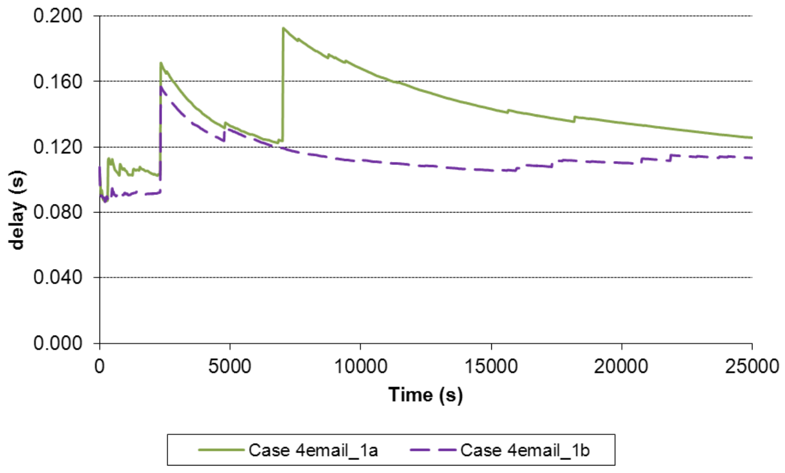


Figure 3-18: Case 4: Average Queue delay for Email

The traffic received before and after scheduling for Voice, Internet Browsing and Email traffic is shown in Figure 3-19. The average traffic received for Voice, Internet Browsing and Email before applying scheduling are 1,018 and 568 and 1,604 bits/s respectively, after applying scheduling the traffic received slightly increased for Voice and Email to 1,040 and 1,617 bits/s respectively. However, the traffic received by Internet Browsing decreases to 491 bits/s. The percentage improvement for Voice and Email is 2 and 0.8% respectively, whilst the traffic received by Internet Browsing decreases by approximately 14%.

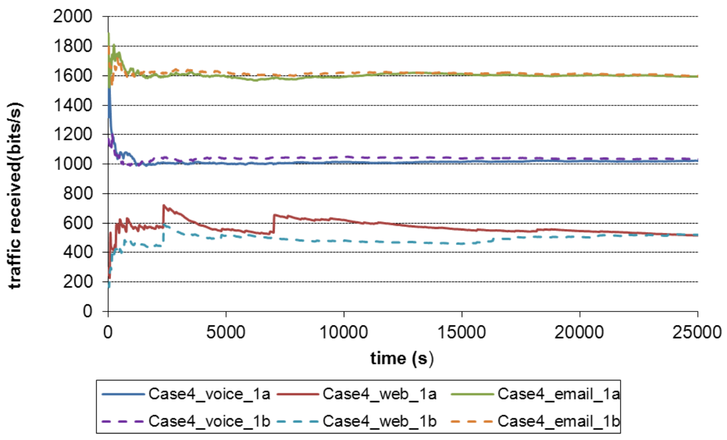


Figure 3-19: Case 4: Average Traffic received for Voice, Internet Browsing and Email

The average queuing delay and queue size for the mixed traffic for scheduled and unscheduled condition are shown in Figure 3-20 and Figure 3-21. The average queuing delay before scheduling is 0.111s; this reduced to 0.083s after scheduling is applied. That is a 25% improvement in the queuing delay after applying scheduling. The average queue size for the mixed traffic is 3,282 bits before applying scheduling; it reduced to 1,351 bits after applying scheduling. This is a 59% decrease in the queue size/network load after applying the scheduler; thus drastically reducing the chances of congestion occurring.

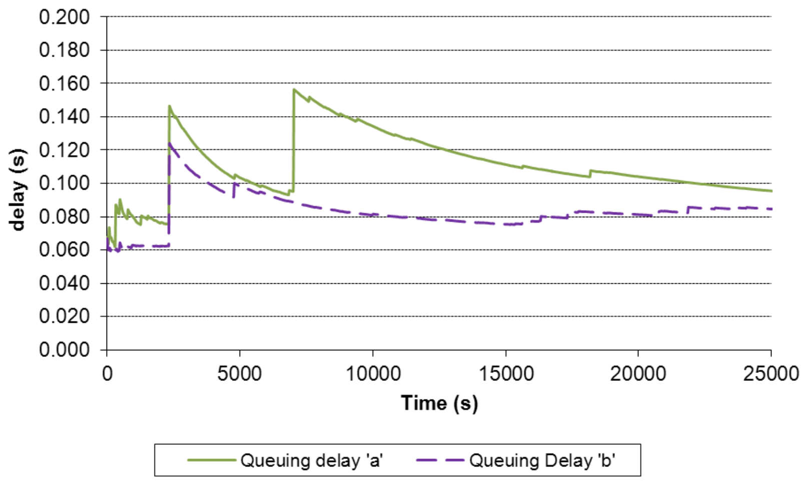


Figure 3-20: Case 4: Average Queue delay for Mixed traffic

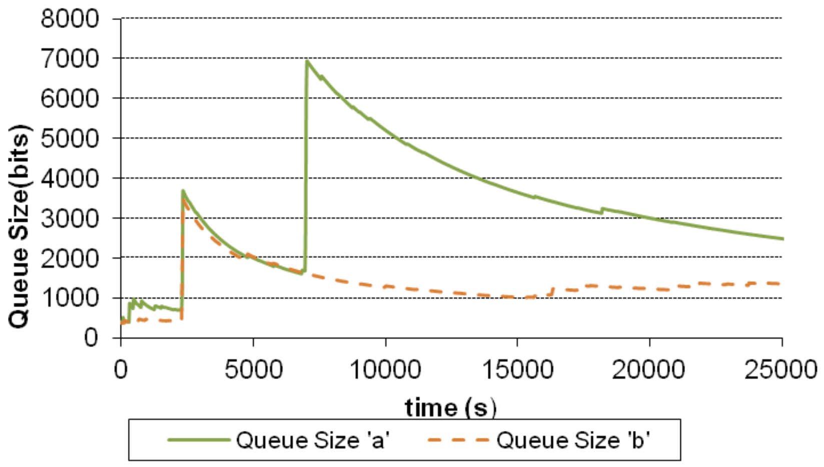
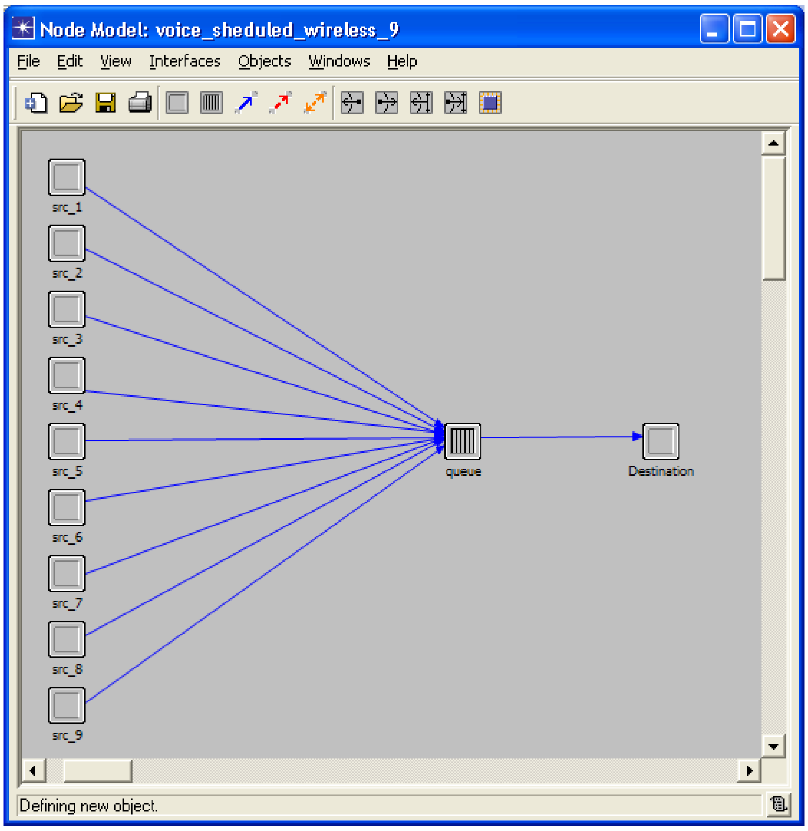


Figure 3-21: Case 4: Average Queue Size for Mixed traffic

3.3.2.5. Performance Analysis of Wired, Wireless and MANET Network

Section 3.3.2 analysed the queuing behaviour of various types of traffic in a wired and wireless node; this will form a baseline model. This section compares simulation performance of three scenarios; A, B and C to the theoretical prediction in section 3.2.1.1. Scenario A consists of baseline wired nodes, it is shown in Figure 3-22; Scenario B consists of baseline wireless nodes without OSI layer, it is shown in Figure 3-23; finally Scenario C consists of MANET nodes with Wireless LAN, which includes the various OSI layer as shown in Figure 3-24.



Defining new object.

Figure 3-22: Scenario A – Wired

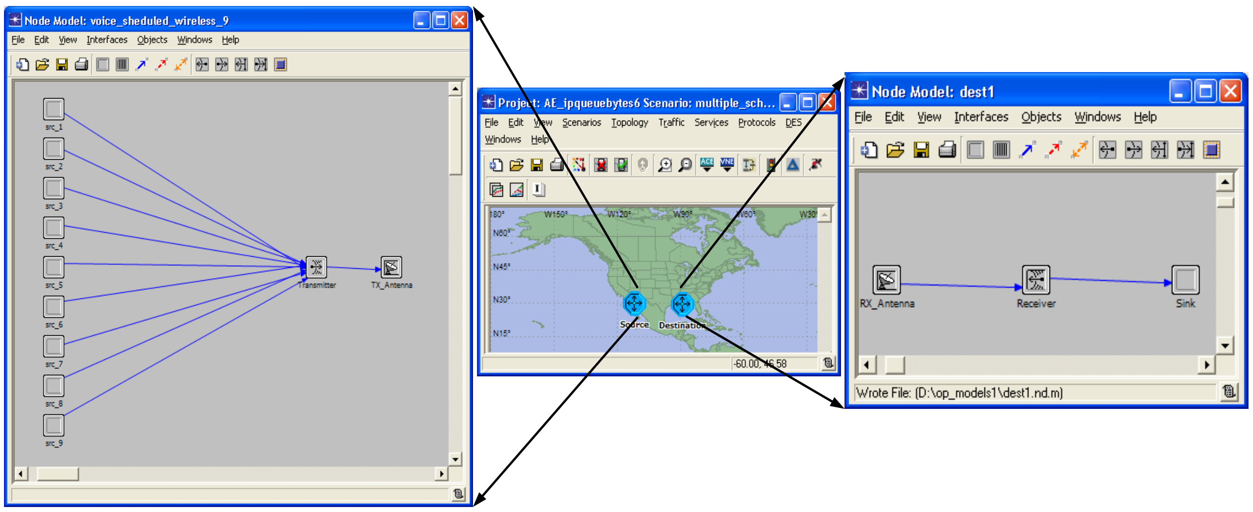

Node Model: voice_sheduled_wireless_9
src_1
src_2
src_3
src_4
src_5
src_6
src_7
src_8
src_9
Transmitter
TX_Antenna
Project: AE_ipqueuebytes6 Scenario: multiple_sch...
Source
Destination
Node Model: dest1
RX_Antenna
Receiver
Sink
Wrote File: (D:\op_models1\dest1.nd.m)

Figure 3-23: Scenario B - Wireless

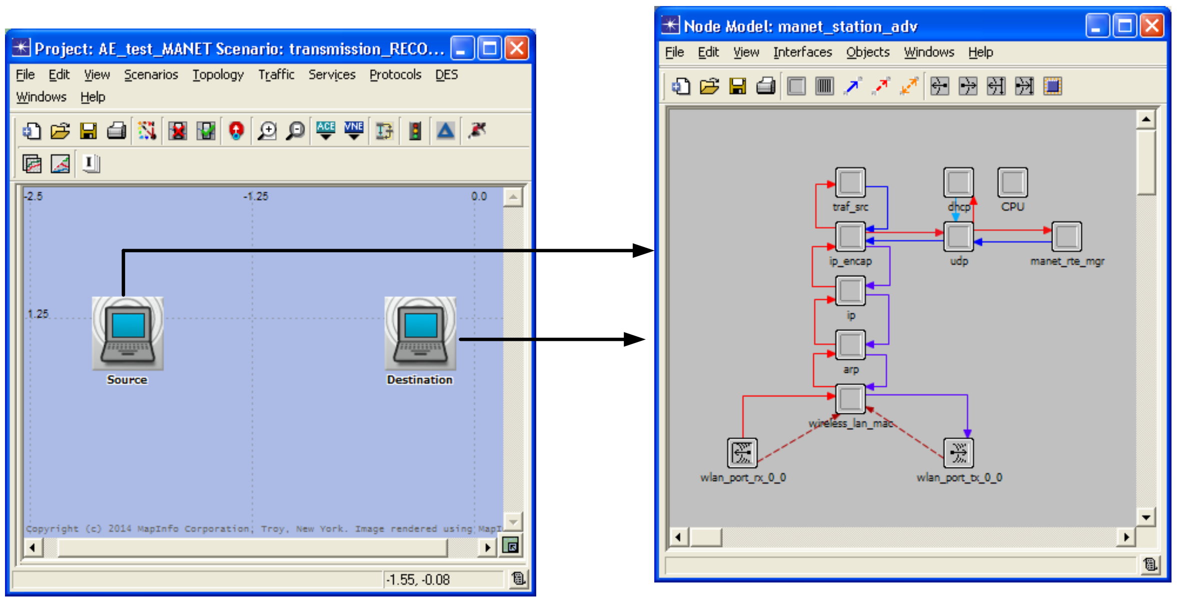


Figure 3-24: Scenario C - WLAN

The M/M/1 queuing system is used for the performance analysis in this section because of the accuracy of the theoretical and the simulation in section 3.3.2.1. Each of these scenarios comprises of a source and destination/sink, and an equal number of sources is used for all scenario (*n*=9) and each source generates the same amount of traffic. The packet size is exponentially distributed with a mean value of 1,024 and an exponential distribution inter-arrival time with a mean value of 1 similar to section 3.3.2.1. The queuing delay, queuing size and traffic received for all three scenarios were analysed. The service rate is 10,742.11

bits/s similar to Case 1 at 'n=9' for all scenarios. The queuing delay for scenarios A, B and C is displayed on a double axis graph in Figure 3-25 because of the scale difference. The queuing delay measurement for scenario A and B are shown on the primary axis whilst that of scenario C is on the secondary axis. The queuing delay for scenario A and B are similar whilst that of scenario C is much higher. The average queuing delay for scenario A, B and C are 0.648, 0.646 and 6.59s, respectively. Scenario A and B are close to the theoretical prediction (0.67s) as shown in section 3.3.2.1., the percentage margin of errors are 3.4 and 3.7%, respectively. Whilst the percentage margin of error for scenario C as compared to the predicted theoretical value is 90%. The measured delay for scenario C is the end-to-end delay, as the simulation tool does not provide a means to measure just the queuing delay. The end-to-end delay consists of queuing, transmission and propagation delay as discussed in Section 3.2. The transmission and propagation delays are very small in the order of milliseconds, hence they cannot have much effect on increasing the end-to-end delay in an ideal channel condition. The model of scenario A and B are less complex as their queue only have to service traffic generated by the source, whilst scenario C queue comprises of both generated traffic and control traffic because of the various functionality of the OSI layer. This additional control traffic, as well as retransmissions of source, increased the network load and, as a result, the queuing delay increases.

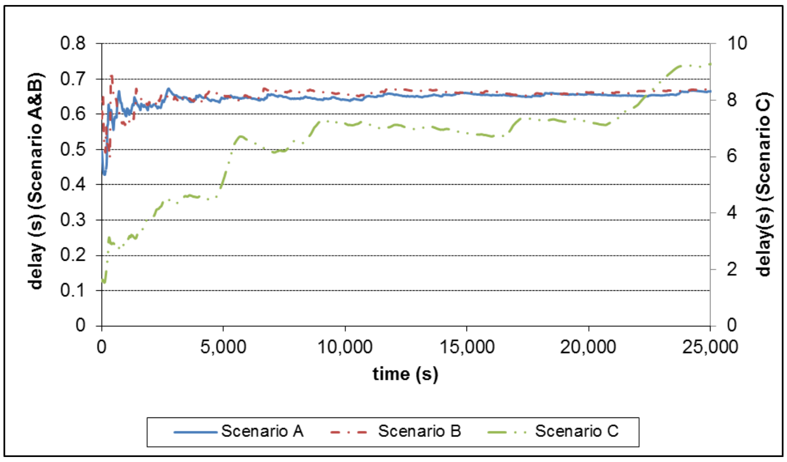


Figure 3-25: Average Queue delay for Scenario A, B and C

The queue size for scenario A and B is shown in Figure 3-26 and that of scenario C could not be measured. The average queue size for scenarios A and B is 6,933 and 7,026 bits respectively, the theoretical predicted queue size according to section 3.3.2.1 is 6,187 bits. Hence, the percentage margin of error for the measured simulated queue size for scenario A and B to that of the theoretical prediction are 12 and 14%, respectively.

The traffic received at the destination in bits/s for scenarios A, B and C are shown in Figure 3-27. The average traffic received is 9,190 and 9,214 and 9,149 bits/s for scenario A, B and C respectively. These values are close to the expected, which is 9,216 bits/s.

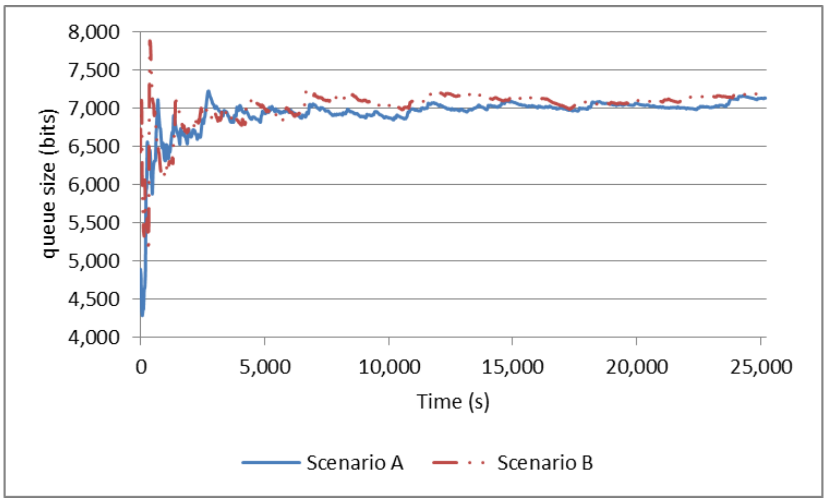


Figure 3-26: Average Queue size for Scenario A and B

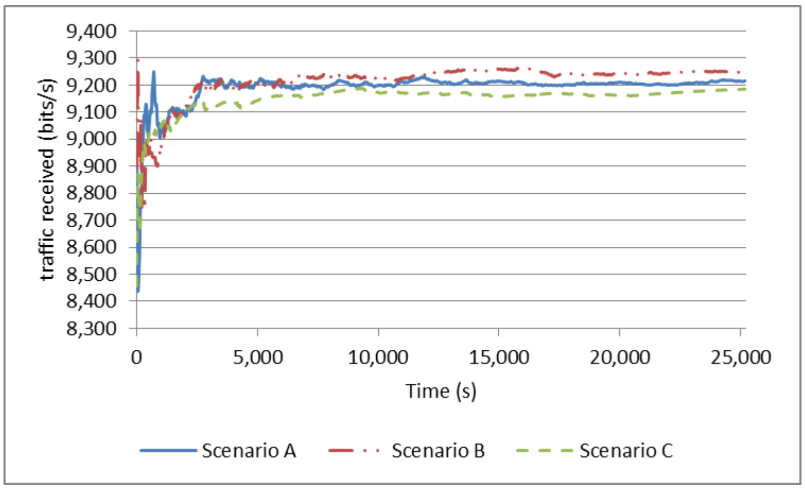


Figure 3-27: Average Traffic Received for Scenario A, B and C

3.4. Development of a Realistic Simulation Model

An open-field experiment is carried out on a peer-to-peer network for constant and exponentially distributed traffic using Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). These are the two available protocols at the transport layer, when using Internet Protocol (IP). The motive of the open field experiment is to establish a baseline propagation model for the IEEE 802.11n standard in OPNET; this should be close to reality. A Fading model (including shadowing and path loss) is applied to the simulation model so it can replicate the outdoor experiment performance. A detailed description of the guidelines and parameters needed to model an outdoor experiment in OPNET is provided.

3.4.1. Background and Previous work

Paper [80],[81] and [82], are some of the currently available MANET simulations with the IEEE 802.11n standard. They focus on channel capacity and security. There is little detail on the overall performance of the network in terms of delay; throughput and traffic received. Most of the previous work with the IEEE 802.11g and earlier standards only considered the received traffic [83],[84]. The end-to-end delay was left out because of the sensitivity of measuring it in real time. The majority of the published papers on MANETs are based on simulation results, and most of these simulation models are based on certain assumptions e.g. the shape of the earth being flat. Paper [85], surveys 2,200 published results pointing out some of the systematic flaws and suggests that the majority of performance evaluation of wireless communication networks is based on stochastic simulation, and they cannot be trusted as they lack credibility.

It is risky to design a simulation model based on simplistic models for radio propagation; thus the simulation models have to be as detailed as possible. The link connectivity graph between two nodes varies over time because of the mobile nature of MANET nodes. However, this change is highly dependent on the differences in the range variation of the antenna, the antenna elevation and also obstacles present (restricting range) [86]. In an ideal case receivers within a transmission range of a source node, receive the entire transmitted packet if there are no obstacles or any adverse channel conditions. However, because of the fading and multipath nature of wireless transmission, in some cases, this ideal assumption significantly deviates from reality. Wireless communication can experience severe fading because of a change in location. It can have a significant effect on the quality of the network performance and likewise invalidates designs (simulation models) that used idealized assumptions. The quality of a wireless link is influenced by its environment [87].

3.4.2. Radio Propagation Model

Some of the characteristics of electromagnetic waves include scattering, reflection, and diffraction. The radio propagation model is classified into two groups, namely; large-scale propagation model (long term fading model), and small-scale propagation model (or short term fading model). The large-scale propagation model predicts the mean signal decay, which is a function of the separation distance, between the transmitter and receiver raised to some power (power law function). The small-scale model is used to predict or characterise the rapid signal strength fluctuation between the transmitter and receiver over a very short distance or within a very short time length. This work will focus on the large scale propagation model, and a brief review of this model is given below [88].

A. *Large Scale Propagation model*

Distance attenuation and shadow fading are two major components that characterise large-scale variations. The distance attenuation is the observed signal power loss over a long period of time, as a function of distance during an experiment. Whilst shadowing is the variation of this trend line over different locations (wireless systems perform differently in different locations). The two most common examples of the large-scale model are free space propagation model and two-ray ground reflection model [89]. The signal power prediction of these two large-scale propagation models is dependent on the distance between the transmitter and receiver; whilst an ideal circle, is used to represent the communication range. This does not consider the shadowing effect. The commonly used model for shadowing effect is the lognormal shadowing model [90]. Theoretical and experimental measurements produced by other researchers show that the average signal power received logarithmically decreases with distance, in both indoor and outdoor environments. The path loss for an arbitrary transmitter-receiver separation can be represented by the log-distance path loss model shown in equation (3-27) [89].

$$\overline{PL}_{dB}(d) = \overline{PL}_{dB}(d_0) + 10\beta log\frac{d}{d_0} \qquad (3\text{-}27)$$

Where, the path loss exponent is represented by β, the corresponding reference distance is represented by d_0, and d represents the distance between the transmitter and the receiver. For a specified value of $\overline{PL}(d_0)$, the average path loss is represented by $\overline{PL}(d)$, it is highly dependent on the propagation environment and it varies between the values 2 and 6. The loss exponent for free space is '2' [90]. However, equation (3-27) does not take into account the environmental variation between two separate locations with the same transmitter and receiver separation distance. Hence, the equation is not sufficient to model this varied performance and therefore cannot model a particular value at specific location. According to empirical studies, at a particular value of $\overline{PL}(d_0)$, the path loss is random at a particular location and log-normally distributed (normally in dB) about the mean distance, this can be represented by equation (3-28)[89].

$$\boldsymbol{PL_{dB}(d) = \overline{PL}_{dB}(d) + X_{\sigma}} \qquad (3\text{-}28)$$

Where X_{σ} represents a zero mean, Gaussian random variable (dB) with a standard deviation (σ_{dB}). The standard deviation varies typically between the values 6-12dB [91]. Therefore, the corresponding received signal power can be written as equation (3-29) [89].

$$\boldsymbol{P_{r_{dBm}}(d) = P_{t_{dBm}} - PL_{dB}(d)} \qquad (3\text{-}29)$$

X_{σ} is dependent on location and distance. The log-normal distribution describes the shadowing effect that occurs over a large number of different measurement locations with the same transmitter and receiver separation distance, but with a varying clutter level in the propagation path. This phenomenon is referred to as log-normal shadowing [84][88].

3.4.3. Experimental Set-up

The specifications and the supported data rate of the WLAN card are shown in Table 3-16. The experimental set-up consists of two laptops with the IEEE 802.11n wireless card; the specifications of both laptops are shown in Table 3-17. The device is configured to use only one spatial stream with a maximum link speed of 54Mbps for this experiment. However, the measured link speed varied between 8-54Mbps during the experiment. A time synchronisation software known as "*dimension 4*" is used to synchronise both laptops [92].

Table 3-16: Wireless card properties and software specification

Parameter	Value
NIC	Node A & B: Realtek RTL 8723AE Wireless LAN 802.11n PCI-E NIC
Network Type	Peer-to-peer
Operating freq.	2.4 GHz
Link Speed	57.8Mbps
Modulation	OFDM(BPSK, QPSK, 16-QAM, 64 QAM)
Receiver Sensitivity	-95dBm
Output Power	-20dBm
IPV4 Address	Node A: 192.168.1.1 Node B: 192.168.1.2
Distance	20, 40, 60, 100, 120, 140, 160, 180, 200, 220, 240, 260, 280, 300, 420 and 460m
Environment	Open field
Transport protocol	UDP and TCP
Time synchronizer	Dimension 4
Nature of Traffic	Constant bit rate (CBR) and Variable bit rate
Traffic Generation	Distributed Internet Traffic Generator (D-ITG D-ITG) [93]
Measurement Metric with reference to distance	Delay, and traffic received

Table 3-17: Laptop Specifications

Toshiba Satellite pro C600-ILD laptop (Node A and B)

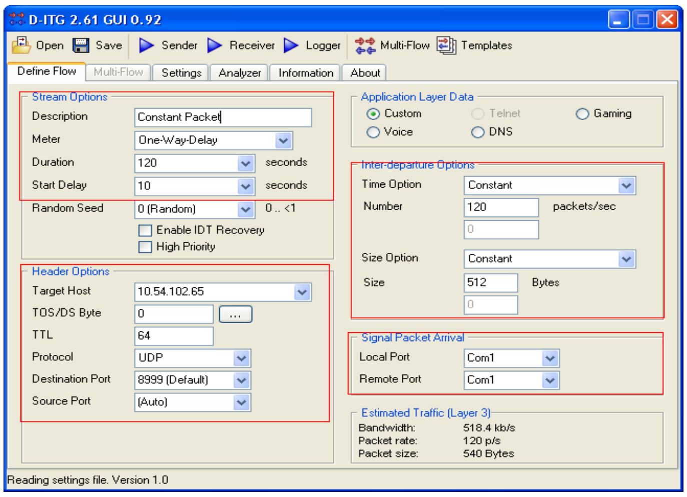
Intel Core (TM) i3-23500M CPU 2.30GHz and equipped with the standard I/O interfaces, and two Express Card slots. Intel Pentium B960 processor and equipped with the standard I/O interfaces, and two Express Card slots
Operating system: window 7 (64-bit operating system)

3.4.4. Software Tools

The traffic generation and data collection for the experiment are implemented using software called Distributed Internet Traffic Generator (D-ITG), whilst the corresponding simulation model, was developed using OPNET. The D-ITG tools are discussed in more details in the following sections.

3.4.4.1. Distributed Internet Traffic Generator (D-ITG)

This software is capable of generating traffic that strictly adheres to the defined Inter-Departure Tme (IDT) between packets and the size of the packet is a stochastic process. Paper [93] compares various traffic generators such as: D-ITG, TG Traffic Generators, Netspec, Netperf, Packet Shell, MGEN, Rude/Crude, UDPgen, Linux Traffic Generator, Traffic Generator (TG), traffic, PacGen and NTgen. According to their findings, D-ITG outperforms the rest; hence it is used for the outdoor experiment. D-ITG has various probability distribution models, which include constant and exponential distributions. Network performance metrics such as throughput, delay, jitter, and traffic received can be measured [94]. Figure 3-28 shows the D-ITG graphical user interface (GUI), and the important settings for this experiment such as stream option, header options, inter-departure option and the signal packet arrival are highlighted.



Stream Options

Description	Constant Packet	
Meter	One-Way-Delay	
Duration	120	seconds
Start Delay	10	seconds
Random Seed	0 (Random)	0 .. <1

☐ Enable IDT Recovery
☐ High Priority

Header Options

Target Host	10.54.102.65
TOS/DS Byte	0
TTL	64
Protocol	UDP
Destination Port	8999 (Default)
Source Port	(Auto)

Application Layer Data

[✓ Custom] Telnet Gaming Voice DNS

Inter-departure Options

Time Option	Constant	
Number	120	packets/sec
	0	
Size Option	Constant	
Size	512	Bytes
	0	

Signal Packet Arrival

Local Port	Com1
Remote Port	Com1

Estimated Traffic (Layer 3)

Bandwidth:	518.4 kb/s
Packet rate:	120 p/s
Packet size:	540 Bytes

Reading settings file. Version 1.0

Figure 3-28: Distributed Internet Traffic Generator (D-ITG) GUI

3.4.4.2. Test Cases

The open-field experiment is classified into four test cases (A, B, C, and D) An average voice call duration is 120s [95], thus the simulation was run for a single call sequence of 120s with a start delay of 10s. The start delay specifies the time the source starts to generate packets; it is the settlement time for individual nodes whilst they establish a connection. The inter-arrival rate is 120 pkts/s, and the packet size is 512 bytes. The 512 bytes/packet size is useful because it is close to the Internet Browsing [96]. The various simulation cases and their respective distributions are shown in Table 3-18.

Table 3-18: Various Simulation and Experiment Cases

Case	UDP/TCP	Pkt size dist.	Pkts arrival rate dist.
A	TCP	constant (512bytes)	constant (120pkts/s)
B	UDP	constant (512bytes)	constant (120pkts/s)
C	TCP	exponential (512 bytes)	exponential(120pkts/s)
D	UDP	exponential (512 bytes)	exponential(120pkts/s)

3.4.5. Simulation Model and Methodology

The OPNET network model contains two MANET WLAN nodes. This node model generates CBR and VBR traffic, which are transmitted over IP. A detailed explanation of CBR and VBR traffic is presented in section 2.4.1. The IEEE 802.11n standard is configured on individual nodes with a maximum interconnection speed of 54Mbps. The IEEE 802.11n parameters of the simulation were modified to match the specifications of the RTL8188CE NIC card used for the experiment. The path loss exponent and shadowing deviation is fine-tuned to match the performance of the simulation to the experiment as closely as possible. This is achieved by keeping σ_{dB} constant and varying β between 1.8 to 3 in steps of 0.01 until the best value for β is reached, then using the best value for β, vary σ_{dB} between values 5 to7.5 in steps of 0.1 until the best value for σ_{dB} is reached. A shadowing propagation model with a path loss exponent (β) of 2.02 and a shadowing deviation (σ_{dB}) of 6.5 is the closest replica of the experimental model. The distance between the nodes for both experiment and simulation is varied between 20 and 460 m. The implementation of the wireless propagation model is shown in Appendix C. The OPNET propagation models have been updated to accommodate the shadowing propagation model. The highlighted parts of the code are the additions to the standard code.

3.4.6. Experiment and Simulation Results

The results of the experiment and the corresponding simulation model are compared in the following sections.

3.4.6.1. Case A

Figure 3-29 and Figure 3-30 show the results of the outdoor experiment and the corresponding simulation model for the average traffic received and end-to-end delay respectively. The average traffic received by the simulation model is slightly higher than that received by the experiment between 20 to 40 m as shown in Figure 3-29. The traffic received by both simulation and experiment is similar (~7.4Mbytes) between 40-420 m, after which the received traffic of the simulation dropped. The percentage error of traffic received between the simulation and experiment is 0.06%, showing a high level of correlation between the simulated and experimental model performance. The average packet delay for the simulated model is much lower than the experimental model, with a percentage error of 95.06%. This is due to the inaccuracy of the time synchronization software.

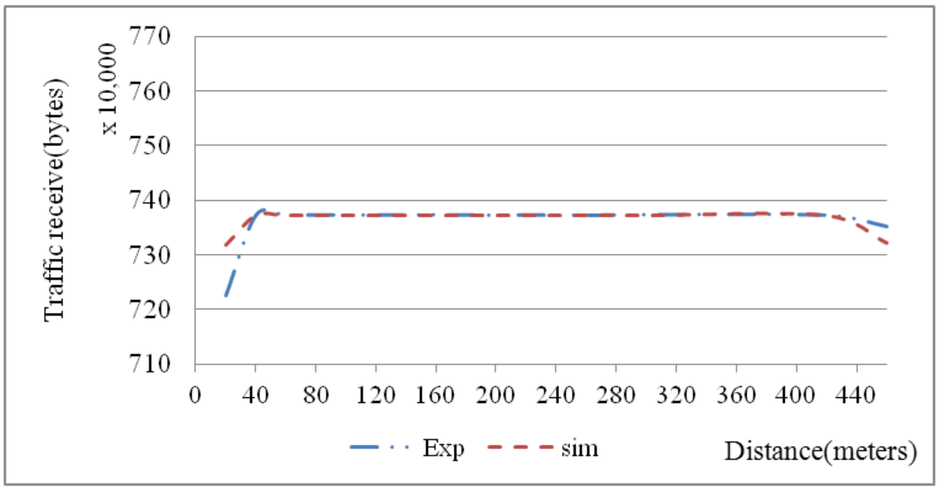


Figure 3-29: Average Traffic Received for Case A

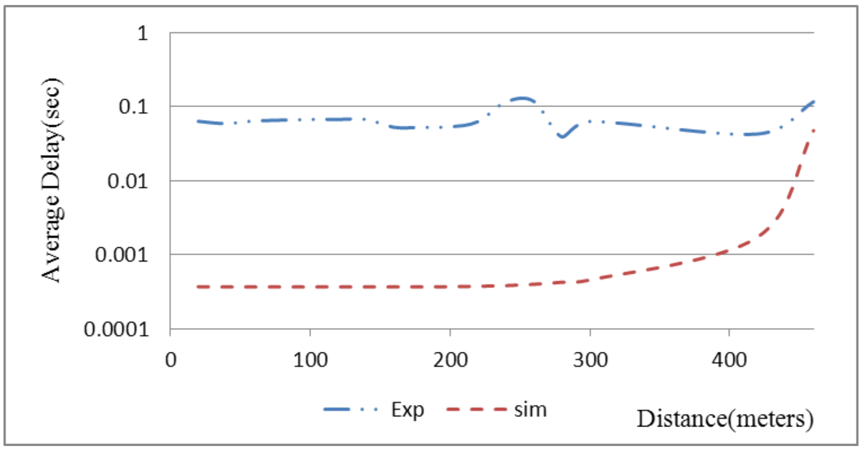


Figure 3-30: Average Delay for Case A

3.4.6.2. Case B

The traffic received by Case B is shown in Figure 3-31, the performance of the simulation model is close to the experiment. The sudden drops in the traffic received for the experiment as noticed at (40m, 180m, and 300m) are due to the environmental conditions on the experiment day; it was drizzling and windy. The percentage margin of error for the traffic received between the experiment and simulation model is 0.978%, signifying an extremely high level of similarity between the experimental and simulation model. The average packet delay for the simulated model is much lower than that of the experiment as shown in Figure 3-32 because of similar reason given in Case A. The percentage margin of error for average delay is 99.04%.

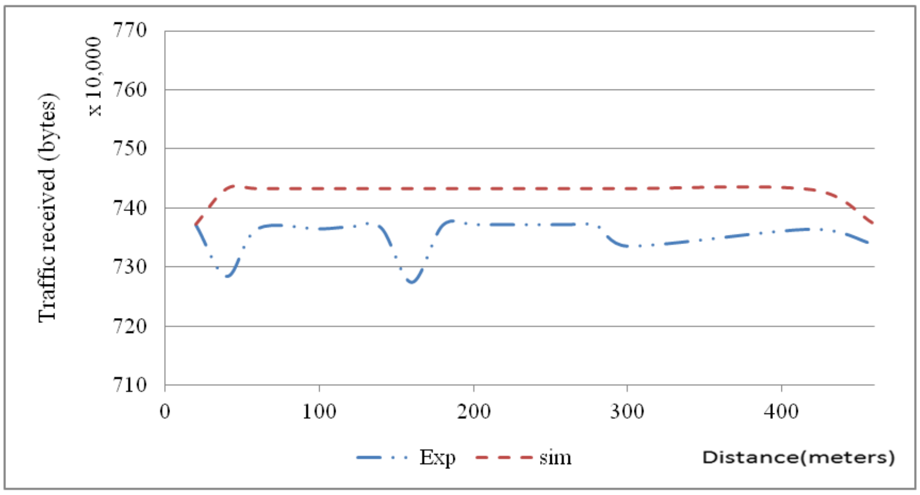


Figure 3-31: Average Traffic Received for Case B

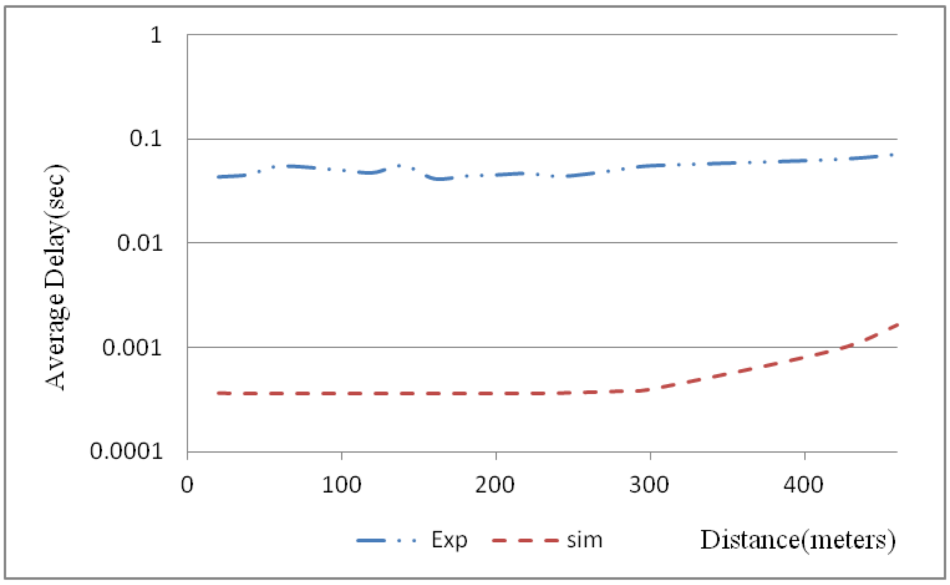


Figure 3-32: Average Delay for Case B

3.4.6.3. Case C

The received traffic for the simulation, as shown in Figure 3-33, is close to the experiment. As expected, due to the exponential nature of the packet size and arrival rate, the behaviour of Case C graph is not as consistent as that of case A and B. There is degradation in the average traffic received by the simulation after 300 m as compared to the experiment; this rapidly worsens after 420 m. The average percentage margin of error between the simulation and experiment for Case C is 0.987%. The average packet delay for the simulated model is lower than that of the experiment as shown in Figure 3-34. The percentage margin of error for average delay is 98.75%. This is as a result of the inaccuracy of the time synchronization software.

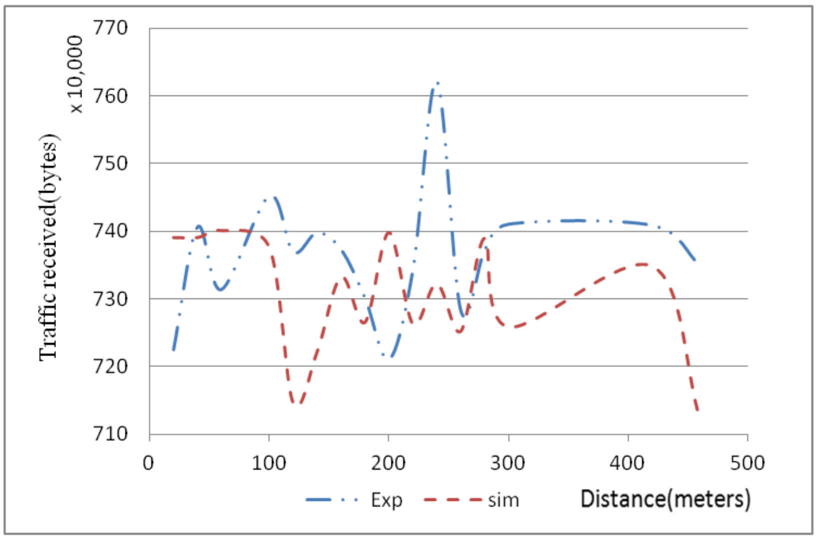


Figure 3-33: Average Traffic received for Case C

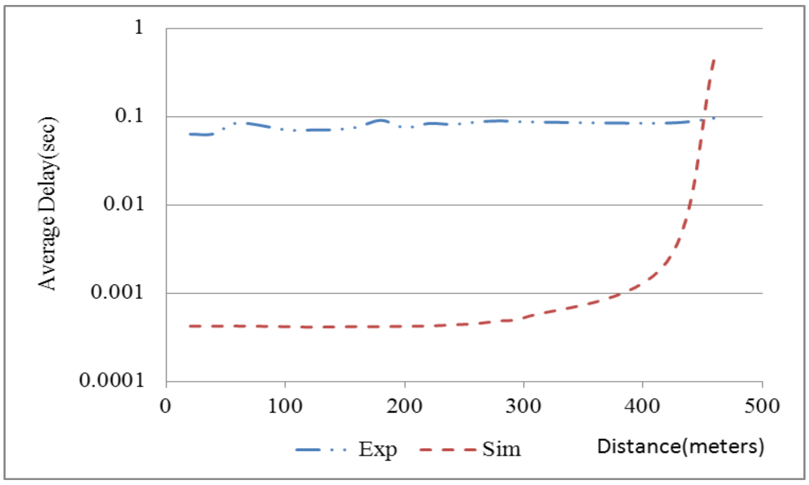


Figure 3-34: Average Delay for Case C

3.4.6.4. Case D

The average traffic received for exponentially distributed packets over UDP is shown in Figure 3-35. The differences between received traffic for experiment and simulation model increased by 188,371 bytes from 290 m and decreases between 300 m and 460 m. The average percentage margin of error between the experiment and simulation for Case D is

0.821%. The performance of the simulation closely replicated the experiment when the traffic received by both models is considered. Thus, OPNET has the capability of modeling realistic models with network performance close to reality. The delay for the experiment and the simulation is significantly far apart as shown in Figure 3-36, for a similar reason to case A. The average percentage margin of error for average delay is 61.38%.

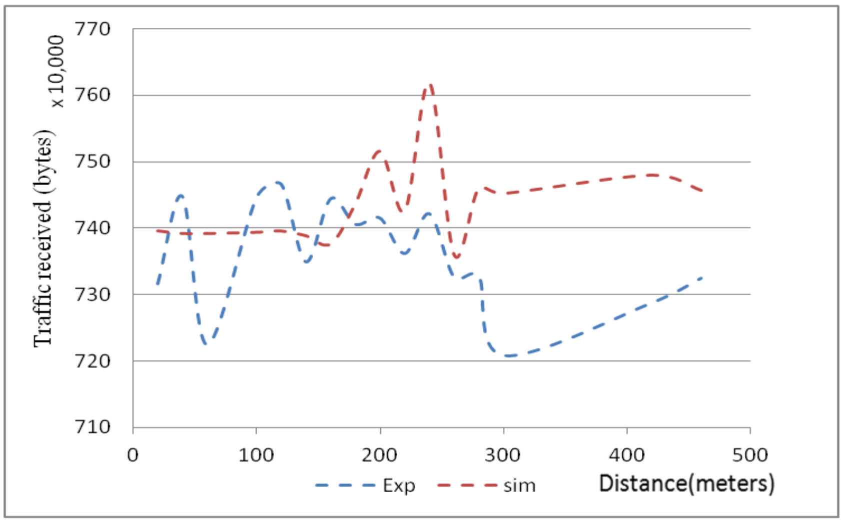


Figure 3-35: Average Traffic Received for Case D

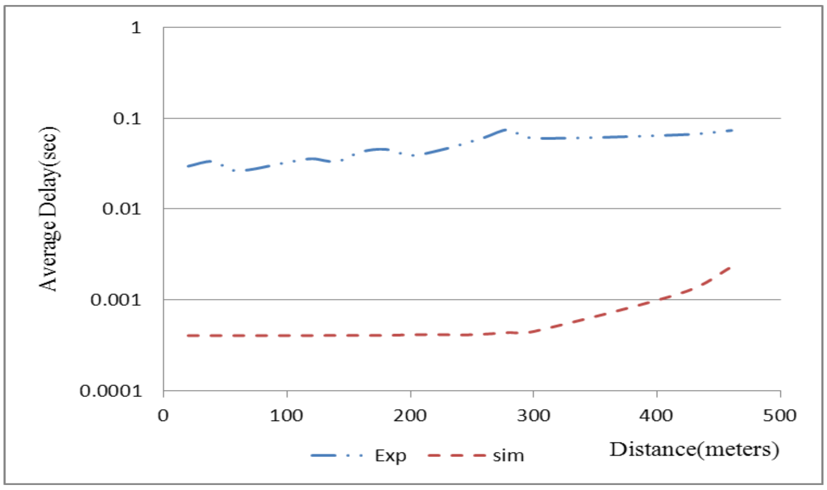


Figure 3-36: Average Delay for Case D

3.4.6.5. Analysis

According to the experiment, the average delay for case B is slightly lower than that of Case A; and also the average delay for Case D is lower than that of Case C. The delay result for the simulation model of Case A and Case B shows similar behaviour to the experiment. This is because of the TCP transport protocol being used; TCP is considered for applications that are not delay sensitive, whilst the delay sensitive applications use UDP. TCP have a slower speed than UDP because in addition to error checking, it also includes recovery options whilst UDP does not. TCP is a connection-oriented protocol; when a connection is established data can be sent bi-directional. If the connection fails, the receiver can request for the entire packet to be retransmitted, in other words, there is minimal file corruption when transmitting with TCP as compared to UDP. This is why the traffic received by TCP in the experiment is slightly higher than that received by the UDP. On the other hand, UDP is connectionless, hence when a packet is sent, it cannot be determined by the source whether the packet is delivered or been corrupted because of the lack of QoS. As a result, UDP has a higher loss rate than TCP even though it outperforms TCP in terms of average delay. Examples of TCP applications include Email, Internet Browsing and File Transfer (FTP) whilst those for UDP include Voice Calling, Video Conferencing, Streaming, etc. A summary of the comparison of UDP and TCP is shown in Table **3-19**. The use of both transport protocols is wholly applications dependent.

There is a high delay variation between the simulation and experimental model because the time synchronization software is not accurate enough. A GPS time synchronization device is needed to measure the delay as accurately as possible with a margin of error in the region of microseconds. However, such devices are very expensive.

OPNET models have the capability of modeling a realistic network in which the measured performance such as, throughput and traffic received is almost identical to real life models with an approximate percentage margin of error in the region of ±1%.

Table 3-19: Summary Comparison of UDP and TCP [97]

Characteristic / Description	**UDP**	**TCP**
General Description	Simple, high-speed, low-functionality “wrapper” that interfaces applications to the network layer and does little else.	Full-featured protocol that allows applications to send data reliably without worrying about network layer issues.
Protocol Connection Setup	Connectionless; data is sent without setup.	Connection-oriented; connection must be established prior to transmission.
Data Interface To Application	Message-based; data is sent in discrete packages by the application.	Stream-based; data is sent by the application with no particular structure.
Reliability and Acknowledgments	Unreliable, best-effort delivery without acknowledgments.	Reliable delivery of messages; all data is acknowledged.
Retransmissions	Not performed. Application must detect lost data and retransmit if needed.	Delivery of all data is managed, and lost data is retransmitted automatically.
Features Provided to Manage Flow of Data	None	Flow control using sliding windows; window size adjustment heuristics; congestion avoidance algorithms.
Overhead	Very low	Low, but higher than UDP
Transmission Speed	Very high	High, but not as high as UDP
Data Quantity Suitability	Small to moderate amounts of data (up to a few hundred bytes)	Small to very large amounts of data (up to gigabytes)
Types of Applications That Use The Protocol	Applications where data delivery speed matters more than completeness, where small amounts of data are sent; or where multicast/broadcast are used.	Most protocols and applications sending data that must be received reliably, including most file and message transfer protocols.
Well-Known Applications and Protocols	Multimedia applications, DNS, BOOTP, DHCP, TFTP, SNMP, RIP, NFS (early versions)	FTP, Telnet, SMTP, DNS, HTTP, POP, NNTP, IMAP, BGP, IRC, NFS (later versions)

3.5. Summary

This chapter introduced Queuing Theory; it is the mathematical study and analysis of waiting in lines or queue. The total delay of a packet in the queue is the summation of the queuing delay and processing delay. In order to calculate the end-to-end delay, the queuing, processing delay, transmission delay and the propagation delay are summed. Queuing theory is used for justification and performance analysis of this thesis. The queuing behaviour of Voice traffic is similar to the M/M/1 queue whilst that of Internet Browsing and Email is similar to the M/G/1 queue. When the finite network resources of a network are stretched to the limit, or traffic arriving at the node or the queue is greater than the capacity, congestion occurs; this leads to packet losses and high delays. If packets arriving the queue are efficiently scheduled, it slows down the chances of congestion occurring.

A brief description of various scheduling algorithms was given; this includes FIFO, PQ, RRS, WFQ and WDS along with the related work. These conventional algorithms are not enough to schedule packets optimally in MANET because of its characteristic. Overall staggering of the source start time has improved the system performance; however, it showed some limitation when dealing with bursty traffic. This scheduling scheme cannot cope in a complex network environment where the nodes are mobile with varying channel conditions. Hence, there is a need to develop an adaptive scheduling algorithm that is comprehensive and can adapt to the changing network topology and various traffic conditions. The traffic behaviour and performance of scheduler (staggered) model will form a baseline model upon which a comprehensive adaptive scheduling algorithm based on fuzzy inference system is developed in the next chapter.

This chapter also provided guidelines for the modelling of a realistic IEEE 802.11n based propagation model for the simulation model in OPNET based on a carried out open field experiment. Shadowing model is applied to the simulation model and the path loss exponent as well as the shadowing deviation was varied to match the performance of the simulation model to the experiment. This formed the baseline propagation model for this thesis.

Chapter 4 Adaptive Fuzzy logic based packet scheduling algorithm

4.1. Introduction

In addition to the band-limited shared network and the error-prone nature of the wireless channel, the infrastructure-less state of MANET makes meeting a specified QoS target more difficult to attain. Hence, it is difficult for any single mobile node to have an accurate and up to date picture of the network topology. For this reason, the network ability to maintain a certain level of QoS is based on an incomplete statistic of the input traffic, queuing behaviour and channel condition; as a result, the decision process has many uncertainties. Fuzzy logic and FIS can be used to model and control complex systems whose available knowledge is limited [98].

This work builds on [66], to propose two adaptive schedulers based on the Mamdani and Sugeno fuzzy logic. The proposed fuzzy schedulers considered some factors that affect the network QoS in real-time to determine a packet priority. In the course of this thesis, paper [66] will be referred to by the first name of the first author ‘Manoj’. The Mamdani method is the widely accepted method for capturing expert knowledge. It allows a more intuitive, more human-like description of the expertise [99]. The proposed Mamdani scheduler is developed based on the Manoj scheduler. It consists of three input variables as compared to the two input variables used by Manoj. Each of these three input variables consists of three linguistic variables; hence the total number of rules is 27 as compared to 9 rules for Manoj.

The proposed Sugeno scheduler is considered because of the simplicity of its design. It is more efficient for optimisation and adaptive techniques of control systems, making it very attractive for network control problems. This section investigates the performance of the proposed Sugeno scheduler as compared to the proposed Mamdani scheduler.

Individual packets were scheduled based on their priority index. The three input variables considered by the scheduler to calculate the packet priority index are Data Rate, Queue Size and Signal-to-Noise Ratio (SNR). The fuzzy scheduler is developed in an OPNET simulation environment using Proto-C language. The Mamdani and Sugeno fuzzy scheduler is optimised so that the algorithm runs quicker which is essential for real-time applications..

4.2. Types of FIS

There are two major types of the fuzzy system named after the founders, the Mamdani [100] and Sugeno [101] fuzzy inference system. The Mamdani FIS was proposed in 1975 by Ebrahim Mamdani in an attempt to control a steam engine and boiler by synthesising a set of linguistic control rules obtained from experienced human operators. The Sugeno FIS was introduced in 1986 by Takagi-Sugeno-Kang. Both the Mamdani and Sugeno FIS for a given system are similar. They both have the same number of inputs and the same membership function. The first two parts of the fuzzy inference process; fuzzification of inputs variables and applying the fuzzy operator are exactly the same. The rules are also the same; the major difference between Mamdani and Sugeno is at the output membership function. The Sugeno output membership functions are either linear or constant whilst the Mamdani output membership function is defined as a fuzzy set. The next section highlights some of the variations between these two FIS systems.

4.2.1. Differences between Mamdani and Sugeno FIS system

The most fundamental difference between the Mamdani and Sugeno FIS is the crispness of the output that is generated from the fuzzy inputs [99]. Some of the most popular Mamdani defuzzification techniques are usually a variation of the max criterion method [21]. These include Smallest-Of-Maxima (SOM), Largest-Of-Maxima (LOM), and the Mean-Of-Maxima (MOM), these methods select the smallest, largest and mean output value for inputs whose membership value reach maximum. MoM is one of the most popular methods; it calculates the final output ‘*Z*’ by averaging the set of output values that have the highest possibility degree ‘*M*’ using the formula given by equation (4-1) [102].

$$Z = \sum_{j=1}^{l} \frac{x_j}{l}, \quad x_j \in M \qquad (4\text{-}1)$$

Two other commonly used defuzzification techniques are the Center Of Gravity (COG) / Centroid and Center Of Area (COA)/Bisector method. The COG/Centroid method determines the crisp output by calculating the center of gravity of the possibility distribution of the outputs. For continuous values, the output '*Z*' is calculated using equation (4-2) [102].

$$Z = \frac{\int_{\mu} \mu(x) x dx}{\int_{\mu} \mu(x) dx} \tag{4-2}$$

The COA is similar to the COG method. However, it calculates the position of the curve where the area of both sides is equal. The COA can be calculated using equation (4-3) [102].

$$\int^{Z} \mu(x)dx = \int_{Z} \mu(x)dx \tag{4-3}$$

Braae et al. [103] presented a detailed analysis of various defuzzification techniques which include COG and MOM, they concluded that COG yields better results and for this reason, the COG / Centroid defuzzification technique is used in this work.

The interpretability and the expressive power of the Mamdani FIS are lost in the Sugeno FIS because the consequent of the rules is not fuzzy [19]. Meaning, when the rules are evaluated the output, will be constant rather than a fuzzy set. Thus, the impact of this on the system performance will be evaluated.

4.3. Adaptive Fuzzy Scheduler

The inputs to the fuzzy scheduler contribute to congestion (both internally and externally). These inputs are SNR, Queue Size and Data Rate as shown in Figure 4-1. This is the Queue Size and Data Rate of the individual nodes the packet is associated with as well as the SNR of the receiver. These inputs were chosen because they best describe the network condition/performance at any time instant. Thus, there is a need to use knowledge gained from this input to determine how packets are scheduled in the network. The inputs are fuzzified, implicated, aggregated and defuzzified to obtain a crisp value that is the output i.e. Priority Index.

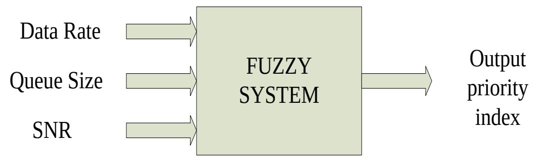


Figure 4-1: Fuzzy Scheduler Inputs and Output

4.3.1. Membership Function

The triangular and trapezoidal membership functions (MFs) are considered in this thesis for their simplicity. They are easy to implement and less computationally complex than the Gaussian membership function. However, the choice of the membership function can change depending on the needs of the designer. The linguistic variables associated with the input variables are Low (L), Medium (M) and High (H). The input membership function for SNR, Queue Size and Data Rate are shown in Figure 4-2 to Figure 4-4, respectively. The x-axis represents the particular fuzzy input and is normalised for all input variables. The y-axis represents the certainty level, and it varies between 0 and 1. There are two ways of mapping MFs, i.e., the number of MFs required for each input variable as well as the baseline. The first is knowledge elicited from experts in the field (manual mapping), and the second is knowledge extracted from trends in empirical data. The range of the fuzzy input on the x-axis is obtained through simple queuing formulas as well as trial and error to maximize the overall system performance. This is carried out by running multiple test simulation models with varying parameters to observe the trends. Paper [66], is also considered in determining the range of the membership functions.

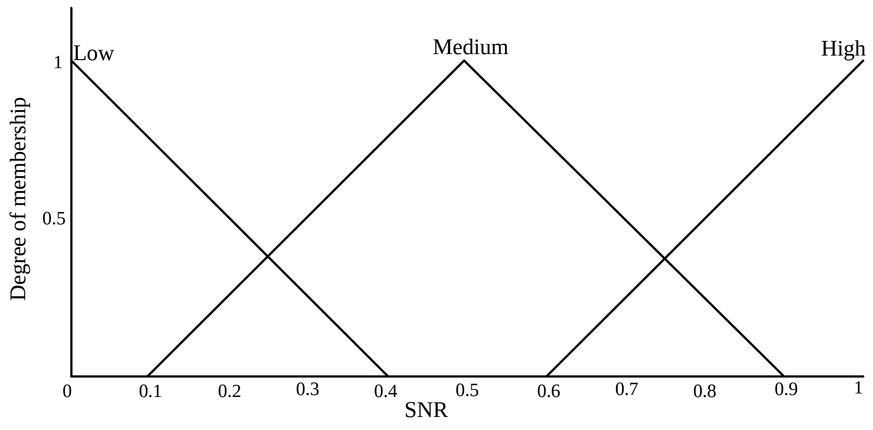


Figure 4-2: Membership function for SNR

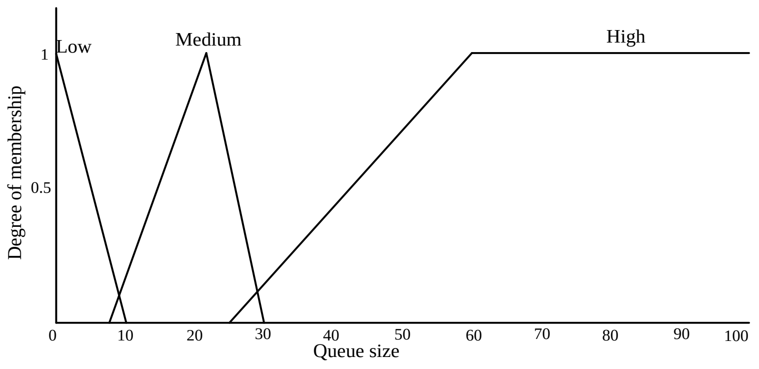


Figure 4-3: Membership function for Queue Size

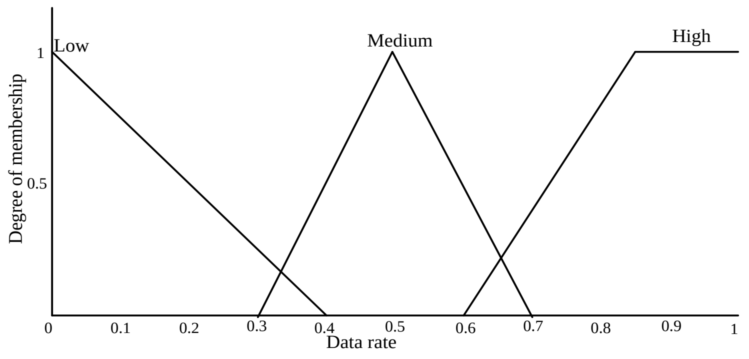


Figure 4-4: Membership function for Data Rate

The fuzzy based output membership function for the Mamdani scheduler is made of a triangular membership function, and it is shown in Figure 4-5. It consists of 5 linguistic variables, namely: Very Low (VL), Low (L), Medium (M), High (H) and Very High (VH).

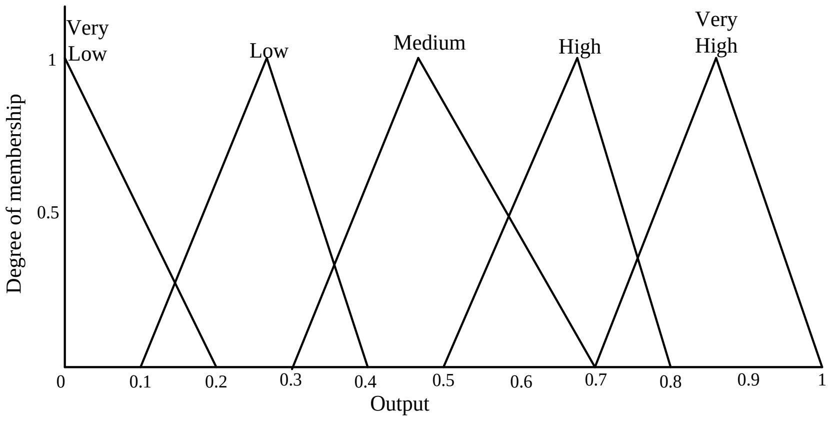


Figure 4-5: Mamdani Output Membership function

Sugeno FIS uses the weighted average to compute the crisp output, and thus the complex iteration process used by Mamdani is bypassed. The Sugeno FIS does not have an output membership function. The output shown in Figure 4-6 is a constant value. It consists of five output constant points that are similar to the number of membership functions for the Mamdani output (Very Low (VL), Low (L), Medium (M), High (H) and Very High (VH)). The Sugeno FIS is a less computationally complex algorithm than the Mamdani equivalent.

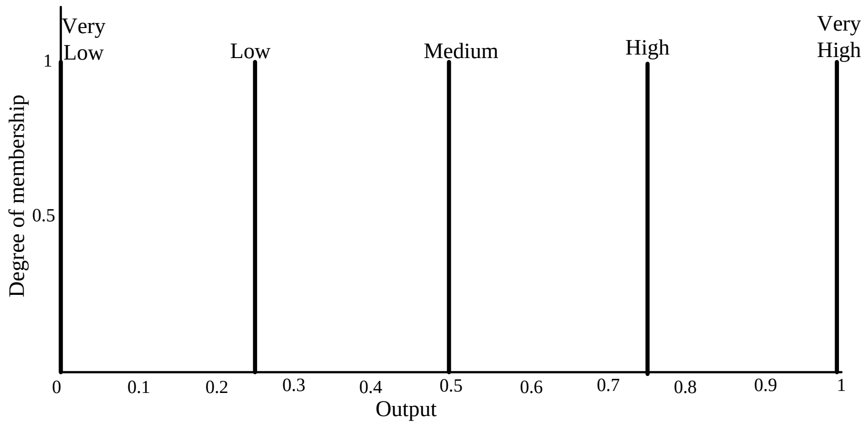


Figure 4-6: Sugeno Output Membership function

The rules are carefully designed based on the relationship between the input variables. The first rule can be interpreted as if (SNR is low) and (Data Rate is Low) and (Queue Size is Low). then the priority index is Medium. The output priority index ranges from 0-1, ‘0’ meaning the highest priority in the queue and ‘1’ the least priority. Thus as the priority index increases from 0-1 the packet priority in the queue drops accordingly. There are three input variables with three associated linguistic variables prompting 3^3 combinations, resulting in the 27 rules shown Table 4-1.

Table 4-1 Rule Base for Fuzzy Inference System

Inputs / Rule	SNR	Data Rate	Queue Size	Output
1	Low	Low	Low	Medium
2	Low	Low	Medium	Medium
3	Low	Low	High	Very Low
4	Low	Medium	Low	Medium
5	Low	Medium	Medium	Low
6	Low	Medium	High	Very Low
7	Low	High	Low	Medium

8	Low	High	Medium	Medium
9	Low	High	High	Very Low
10	Medium	Low	Low	High
11	Medium	Low	Medium	Medium
12	Medium	Low	High	Low
13	Medium	Medium	Low	High
14	Medium	Medium	Medium	Medium
15	Medium	Medium	High	Low
16	Medium	High	Low	High
17	Medium	High	Medium	Medium
18	Medium	High	High	Low
19	High	Low	Low	High
20	High	Low	Medium	Medium
21	High	Low	High	Low
22	High	Medium	Low	Very High
23	High	Medium	Medium	High
24	High	Medium	High	Medium
25	High	High	Low	Very High
26	High	High	Medium	High
27	High	High	High	Low

The graphical representation of Table 4-1 according to the input and output membership functions is shown in Figure 4-7. The surface viewer, which shows the relationship between the inputs and output, is shown in Figure 4-8. When the SNR, Data Rate and Queue Size input are 0.5, 0.5 and 50, respectively, these values will only be viable under rule 15 on Table 4-1 and Figure 4-7. The output fuzzy set is aggregated (this is the minimum of the fuzzy set) for each rule, it will be zero for all except at rule 15 to give the corresponding output. The outputs are aggregated (this is the maximum of the fuzzy set) to the shape similar to that in Rule 15 as all others are zero. This is defuzzified using the Centre of gravity (COG)/Centroid method to give a crisp output of 0.25.

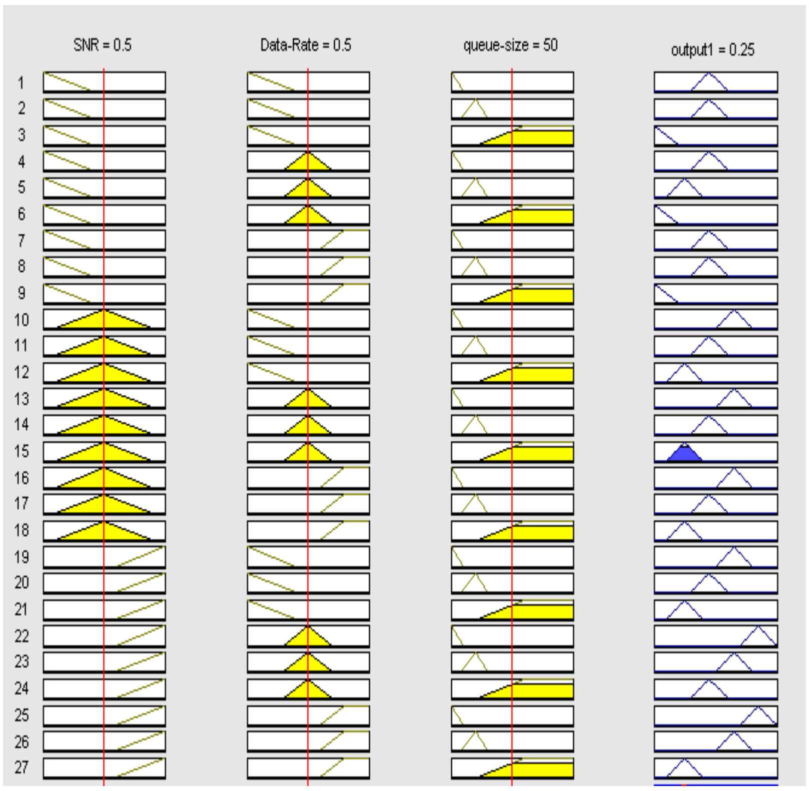
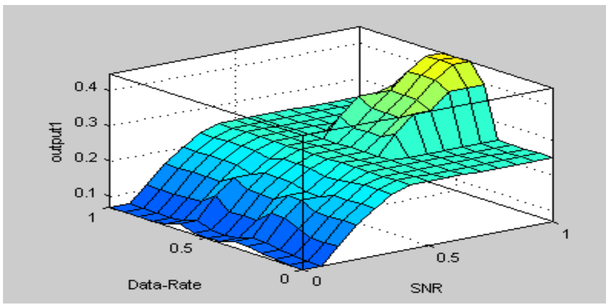


Figure 4-7: Membership Function and fuzzy rule base for the proposed Fuzzy Scheduler



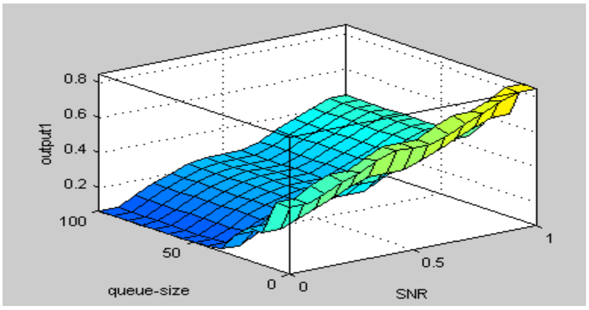
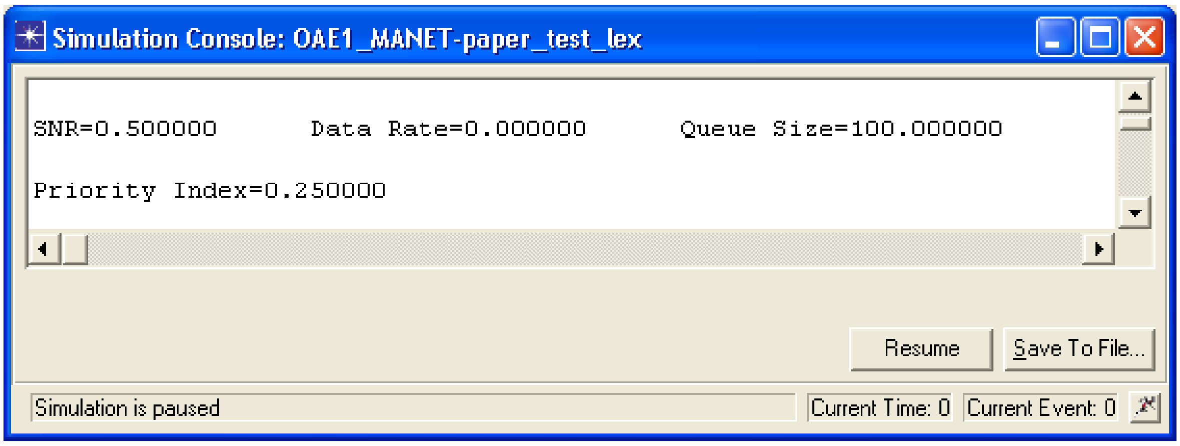


Figure 4-8: Surface viewer for Fuzzy Scheduler

4.3.2. Implementation and Validation of the Fuzzy Scheduler

The Manoj, proposed Mamdani and Sugeno schedulers would be implemented in OPNET. The flow chart is shown in Figure 2-8; the input variables were obtained from the network and the fuzzy rules are evaluated based on these inputs. The inputs are fuzzified; each rule evoked have a corresponding output membership function. This output membership function is then implicated, aggregated, and the crisp value (priority index) is calculated from these aggregated curves by using a Centroid defuzzification technique. The Proto-C algorithm, which implements the Manoj, the proposed Mamdani and Sugeno fuzzy scheduler, is verified using the fuzzy logic toolbox in MATLAB. The implementation of the proposed Mamdani and Sugeno scheduler is shown in Appendix D and Appendix E respectively. The highlighted parts of the codes are the changes made to the standard OPNET model to accommodate the fuzzy algorithm and their inputs. When the SNR, Data Rate and Queue Size input are 0.5, 0.5 and 50, respectively as in Figure 4-7, the Mamdani C algorithm in OPNET calculates a crisp value of 0.25 which is the same as that calculated by MATLAB in Figure 4-9. The OPNET algorithm is run for multiple input values, and it produced the same crisp output with the corresponding MATLAB fuzzy model. The validation of the Manoj scheduler is wholly based on the fuzzy algorithm as paper [66] did not provide enough information to replicate the MANET simulation model.



```
SNR=0.500000       Data Rate=0.000000       Queue Size=100.000000

Priority Index=0.250000
```

Figure 4-9: Mamdani Fuzzy Priority Index

4.3.3. Performance Evaluation using OPNET

The output Priority Index of a packet is used to schedule the packet. By scheduling the packets this way, packets in highly congested queues are scheduled first. This differs from the standard priority scheduler because the packet Priority Index is based on individual packets rather than a traffic flow. If the queue of a node is full, it will cause an increase in the end-to-end delay and packet loss rate, thus newly arriving packets are discarded and packets in the queue that have exceeded the waiting threshold are also discarded. The cause of the degradation of network performance is not limited to the size of the queue; it also relates to the Data Rate and SNR. When the SNR of the receiving node is low, the network will suffer a higher packet loss because of the poor wireless communication link between nodes. The packet priority increases as the SNR decreases, to reduce the packet loss rate and thus improve the end-to-end delay.

The final input is the Data Rate; at higher Data Rates, the end-to-end delay of a packet is low, and the PDR is significantly higher. However, when the reverse is the case, there will be a higher packet loss rate and an increase in the end-to-end delay. Packets are given a higher priority when the Data Rate is low. Packets present in a crowded node will experience a high queue delay and higher packet loss rate. This algorithm monitors these parameters and calculates an appropriate Priority Index in order to optimise the network and improve the QoS performance. When a packet reaches a node, its Priority Index based on the network properties of that node is calculated and attached to its header.

Each node consists of three sub-queues similar to the Manoj scheduler. This will reduce the effect of sorting on the overall network performance; arriving packets are enqueued in these sub-queues based on their Priority Index. The first sub-queue en-queues packets with Priority Index between 0 and 0.33; the second sub-queue en-queues packet with Priority Index greater than 0.33 but less than or equal to 0.66 and finally, the third sub-queue en-queues packet with Priority Index greater than 0.66 but less than or equal to 1. The net result being that packets are sorted in the various sub-queue based on their Priority Index (i.e., packets with the Lowest Priority Index move to the head of the queue and are scheduled first). Appendix F implements the packet insertion and sorting in the various sub-queue, the highlighted part of the codes are the changes and additions made to the standard OPNET model to accommodate packet sorting.

4.3.4. Optimisation of the Mamdani and Sugeno Fuzzy Algorithm

The proposed algorithms are optimised by measuring the number of times each rule is used by the network for CBR and VBR traffic. For CBR traffic, the result shows that only 10 of the 27 rules were used as shown in Table 4-2. This is because CBR traffic consists of constant Data Rate as explained in section 2.4.1. The 17 unused rules are eliminated; further optimisation is carried out with the 10 remaining rules. Two rules are found to have been used less than 200 times (Rule 1 and 23) and therefore are also eliminated, reducing the total number of rules for the scheduler to 8 for CBR traffic. The performance of the optimised scheduler with 8 rules is compared with the performance of the scheduler with 10 rules. This is done by classifying the test into four test cases as shown in Table 4-3.

Table 4-4 shows the average delay, throughput and PDR for all test cases. According to this table, there is no performance degradation for all test cases as all the results are the same. Hence, the final number of rules for the CBR traffic is optimised from 27 to 8 without any performance degradation; thus, Case 4 is implemented.

Table 4-2: Number of FIS Rules used by CBR Traffic profile

Rules	Count
Rule 1	57
Rule 2	0
Rule 3	0
Rule 4	202,551
Rule 5	24,989
Rule 6	186,714
Rule 7	0
Rule 8	0
Rule 9	0
Rule 10	0
Rule 11	0
Rule 12	0
Rule 13	271,724
Rule 14	31,477
Rule 15	310,278
Rule 16	0
Rule 17	0
Rule 18	0
Rule 19	0
Rule 20	0
Rule 21	0
Rule 22	1,512
Rule 23	85
Rule 24	852
Rule 25	0
Rule 26	0
Rule 27	0
Total	**1,030,239**

Table 4-3: Test Cases for CBR Traffic

Case 1	Contains 10 rules
Case 2	Contains 9 rules - only rule 23 is removed
Case 3	Contains 9 rules – only rule 1 is removed
Case 4	Contains 8 rules - rule 1 and 23 are removed

Table 4-4: Rules Optimisation results for CBR Traffic

Cases	Delay (s)	Throughput (bytes/s)	PDR
Case 1	33.603	111,538.9	0.374
Case 2	33.603	111,538.9	0.374
Case 3	33.603	111,538.9	0.374
Case 4	33.603	111,538.9	0.374

A similar optimisation technique used for CBR is applied to VBR. All 27 rules were used, 3 of those rules (rule 20, 23 and 26) were used less than 200 times in the course of the simulation as shown in Table 4-5.

Table 4-5: Number of FIS rules used by VBR Traffic Profile

Rules	Count
Rule 1	102,324
Rule 2	14,632
Rule 3	145,251
Rule 4	81,886
Rule 5	9,626
Rule 6	89,361
Rule 7	80,126
Rule 8	8,076
Rule 9	71,367
Rule 10	130,786
Rule 11	17,515
Rule 12	192,349
Rule 13	100,258
Rule 14	11,380

Rules	Count
Rule 15	117,355
Rule 16	96,985
Rule 17	9,612
Rule 18	93,387
Rule 19	2,881
Rule 20	186
Rule 21	1,032
Rule 22	1,929
Rule 23	117
Rule 24	655
Rule 25	2,149
Rule 26	121
Rule 27	689
Total	**1,382,035**

After the three aforementioned rules are eliminated, a series of simulations were carried out to check the performance degradation resulting from eliminating any or all of these 3 rules. The simulation work is classified into seven test cases as shown in Table 4-6.

Table 4-6: Test Cases for VBR Traffic

Case 1	Contains 27 rules
Case 2	Contains 26 rules – only rule 20 is removed
Case 3	Contains 26 rules – only rule 23 is removed
Case 4	Contains 26 rules – only rule 26 is removed
Case 5	Contains 24 rules - Rule 20, 23 and 26 are removed
Case 6	Contains 25 rules - Rule 20 and 23 are removed
Case 7	Contains 25rules - Rule 20 and 26 are removed

Table 4-7 shows the results for the average delay, throughput and PDR for all test cases. The results showed no significant changes in the performance. Thus, the fuzzy rule for the VBR traffic is optimised from 27 to 24 by eliminating all the rules that are used less than 200 times i.e. case 5 is implemented.

Table 4-7: Rule Optimisation Results VBR Traffic

Cases	Delay (s)	Throughput (bytes/s)	PDR
Case 1	32.93515	82,462.04	0.276353
Case 2	33.10667	82,547.1	0.276452
Case 3	33.00937	82,479.36	0.276411
Case 4	32.93515	82,462.04	0.276353
Case 5	33.09063	82,553.96	0.276419
Case 6	32.93515	82,462.04	0.276353
Case 7	33.00937	82,479.36	0.276411

4.4. Performance Analysis of Fuzzy Schedulers

The performance of the proposed fuzzy scheduler is evaluated by comparing them to “Manoj”. The proposed Sugeno scheduler is faster than the Manoj and the proposed Mamdani because it is less computationally complex, and therefore it should be more appropriate for real-time application. The schedulers have varying degrees of complexity, hence the algorithms were run in Microsoft Visual Studio for a 100 cycles, a timer is inserted at the beginning and end of each cycle to measure the duration, and the average time is calculated. This is added as a constant value to the formulae that calculates packet processing delay in OPNET. The performance of the scheduler is evaluated for CBR, VBR and Bursty traffic class; this traffic profiles were explained in section 2.4.1. The objective of this analysis is to analyse how the proposed schedulers perform under congested conditions. The traffic intensity (ρ) is varied between 1 and 2, and the performance of the scheduler is analysed.

4.4.1. Simulation Environment and Methodology

The simulation models a MANET of 20 mobile nodes, randomly distributed within a 500m x 500m area. The mobile nodes have wireless interfaces, which are configured to the IEEE 802.11n standard with a propagation model as that in section 3.4. Each of the simulations is run for 600s, and multiple runs are carried out with varying seed values and the collected data is then averaged. The seed is used by the simulation's random number generator; multiple seed value will provide multiple instances of the traffic generated. The simulation parameters are shown in Table 4-8, the network topology is shown in Figure 4-10 respectively. The propagation model and link speed is modelled as in section 3.4.5. All mobile nodes served as a transmitter and receiver and the data payload is 1,024 bytes [66]. The performance of the schedulers is evaluated under various load conditions ρ =1, 1.33, 1.66 and 2 for CBR, VBR and bursty traffic profiles. The random waypoint mobility model is used, and the node speed ranges from 0-20m/s with a pause time of 4s.

Table 4-8: Simulation Parameters for MANET

No. of Nodes	20
Area	500m x 500m
Simulation Time	600s
Mobility Model	Random waypoint
Speed	0-20m/s
Traffic Type	CBR, VBR & Bursty
Data Payload	1,024 bytes
MAC protocol	IEEE 802.11n (Buffer Size= 16MB)

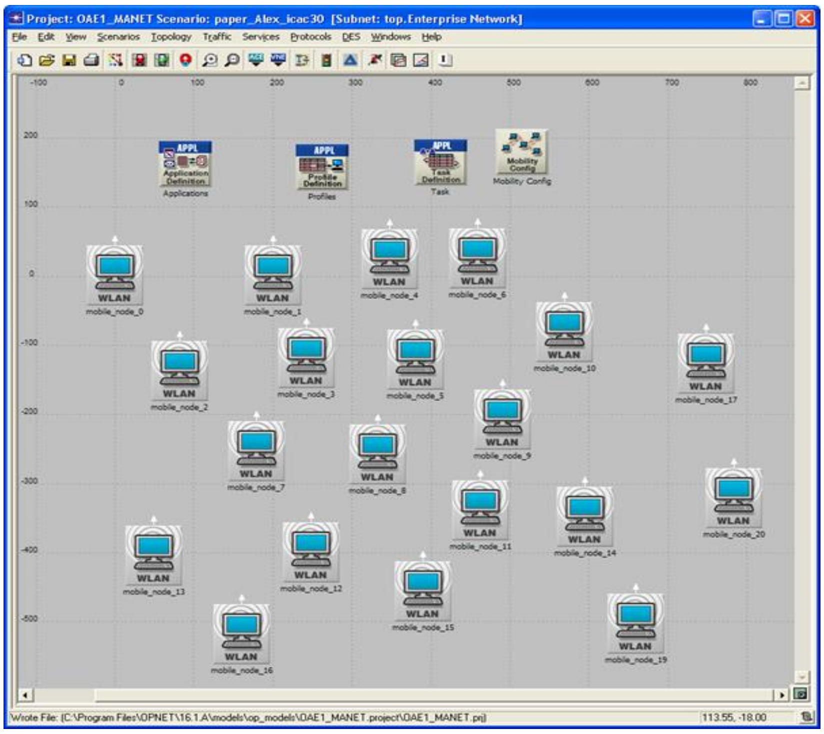

Project: OAE1_MANET Scenario: paper_Alex_icac30 [Subnet: top.Enterprise Network]
File Edit View Scenarios Topology Traffic Services Protocols DES Windows Help
Application Definition
Profile Definition
Task Definition
Mobility Config
WLAN
Wrote File: (C:\Program Files\OPNET\16.1.A\models\op_models\OAE1_MANET.project\OAE1_MANET.prj)
113.55, -18.00

Figure 4-10: Network Topology

4.4.2. Performance Analysis of CBR Traffic

The characteristic of CBR traffic profile has been explained in section 2.4.1.1. The end-to-end delay for CBR traffic at $\rho = 1$ and $\rho = 1.33$ are shown in Figure 4-11 and Figure 4-12 respectively. The proposed schedulers (Mamdani and Sugeno) perform better than the existing (Manoj). According to Figure 4-11, the Sugeno scheduler performs slightly better than the Mamdani scheduler. However, the performance of the Mamdani and Sugeno scheduler are very close to Figure 4-12.

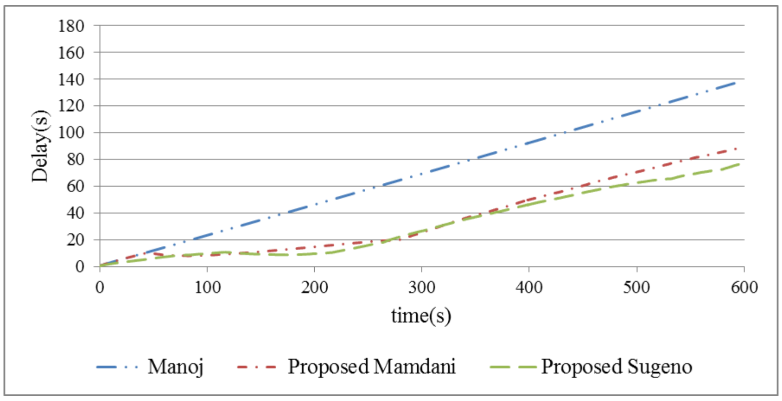


Figure 4-11: End-to-End delay at $\rho = 1$ for CBR Traffic

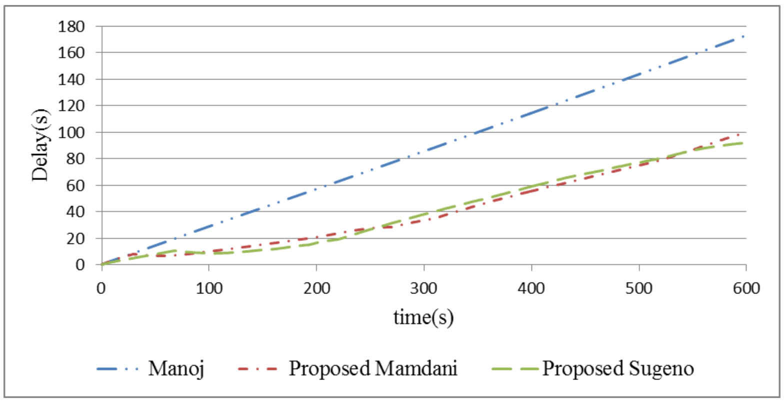


Figure 4-12: End-to-End delay at ρ = 1.33 for CBR Traffic

The graph for the end-to-end delay at ρ = 1.66 and ρ = 2, behave similarly to Figure 4-11 and Figure 4-12. Table 4-9 contains the values for the average end-to-end delay for all traffic loads as well as the percentage improvement of the Mamdani and Sugeno scheduler as compared to Manoj. Overall the end-to-end delay shows a high level of increase when the network is congested; this behavior is similar to section 2.4.2.1.A. According to Table 4-9, it can be noted that the proposed Mamdani and Sugeno scheduler performed better than Manoj. At ρ =1 and ρ =1.66, the proposed Sugeno performed slightly better than Mamdani whilst at ρ =1.33 and ρ =2 the proposed Mamdani performed slightly better.

Table 4-9: Average End-to-End delay for CBR Traffic

Scheduler	Average End-to-End delay (s)			
	ρ =1	ρ =1.33	ρ =*1.66*	ρ =2
Manoj	69.43	86.11	99.75	109.00
Proposed Mamdani	35.92	41.11	46.54	49.96
Proposed Sugeno	32.03	41.36	42.04	49.98
%improvement Mamdani	**48.26**	**52.26**	**53.35**	**54.16**
%improvement Sugeno	**53.87**	**51.97**	**57.86**	**54.15**

When a network is congested, the gradient of its delay graph becomes steeper as more packets arrive at the queue. The gradient of an end-to-end delay graph (Figure 4-11 and Figure 4-12) shows the rate of increase of the network congestion; as the network load increases the gradient also increases. Therefore, to avoid congestion or prevent a severe case of congestion the gradient of the end-to-end delay needs to be prevented from increasing abruptly. The gradient of the end-to-end delay is shown in Table 4-10.

Table 4-10: Gradient of the End-to-End Delay for CBR Traffic

Scheduler	Gradient of the End-to-End delay			
	ρ =1	ρ =1.33	ρ	ρ =2
Manoj	0.230	0.290	0.290	0.360
Proposed Mamdani	0.150	0.160	0.180	0.170
Proposed Sugeno	0.130	0.167	0.174	0.198
%improvement Mamdani	**34.34**	**43.51**	**38.91**	**52.74**
%improvement Sugeno	**41.70**	**41.84**	**39.31**	**45.53**

The Manoj model becomes congested more quickly as shown in Figure 4-11 and Figure 4-12. The proposed fuzzy schedulers (Mamdani and Sugeno) produce a lower gradient than Manoj, this is because incoming packet to the queue are given higher priority when the Queue Size is high, the SNR is low and the Data Rate is also low. These are the characteristics of the input variables when the network tends towards congestion.

The gradient of the Mamdani scheduler is 34.34% less than Manoj at ρ =1, whilst that of Sugeno is 41.70% less than Manoj for a similar load; thus the network congestion is reduced by 34.34 and 41.70% for Mamdani and Sugeno, respectively. At an increased network load ρ =1.33, 1.66 and 2, the gradient of end-to-end delay for the Mamdani scheduler is 43.51, 38.91 and 52.74% lower than Manoj respectively. The proposed schedulers (Mamdani and Sugeno) reduced the level of congestion in the network.

The queue will always forward packet at maximum capacity when the network is congested with CBR traffic. The PDR for ρ =1 and ρ =1.33 are shown in Figure 4-13 and Figure 4-14 respectively.

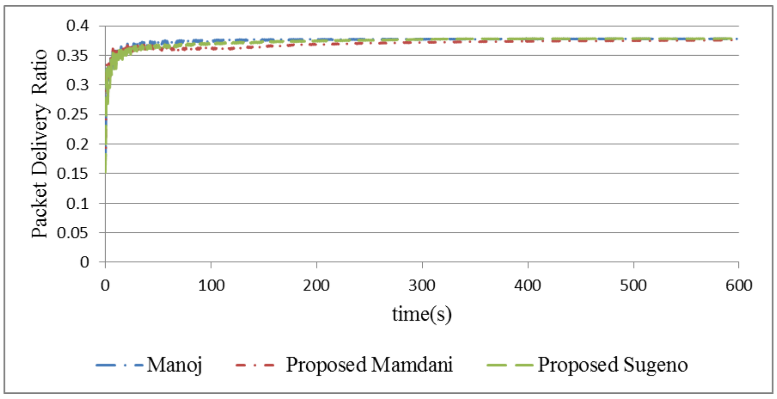


Figure 4-13: Packet delivery ratio at ρ = 1 for CBR Traffic

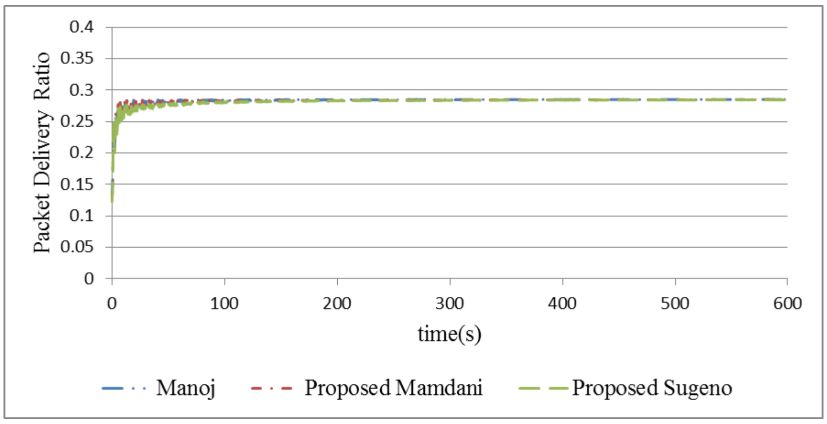


Figure 4-14: Packet delivery ratio at ρ =1.33 for CBR Traffic

The performance of the PDR for all schedulers is similar at the various network loads. The PDR for all network loads is summarised in Table 4-11. These values are low because of congestion, according to Table 4-11, the PDR decreases as the network load increases; this is a bottleneck effect similar to section 2.4.2.1.B. The Mamdani and Sugeno perform slightly better than Manoj.

Table 4-11: Packet Delivery Ratio for CBR Traffic

Scheduler	PDR			
	ρ =1	ρ =1.33	ρ =1.66	ρ =2
Manoj	0.195	0.125	0.093	0.073
Proposed Mamdani	0.276	0.191	0.147	0.116
Proposed Sugeno	0.277	0.188	0.140	0.108
%improvement Mamdani	**42.03**	**53.00**	**57.77**	**58.56**
%improvement Sugeno	**42.53**	**50.12**	**49.64**	**47.38**

4.4.3. Performance Analysis of VBR Traffic

The characteristic of VBR traffic profile has been explained in section 2.4.1.2. The end-to-end delay for the traffic load at ρ =1 is shown in Figure 4-15. The proposed (Mamdani and Sugeno) schedulers perform better than the existing scheduler (Manoj). The performance of the proposed Mamdani scheduler in the first 0-30s is slightly higher than Manoj, whilst Sugeno scheduler is close to Manoj between this simulation times. Thus, Manoj and Sugeno perform better than Mamdani between simulation times 0-30s. The performance of the Mamdani scheduler improves significantly as compared to Manoj from simulation time 30-

600s. The Sugeno scheduler also starts to improve significantly as compared to Manoj between simulation times 90-600s; the Sugeno also performs slightly better than Mamdani between simulation times 120-600s. Since $\rho > 1$, the limited network resources cannot cope with the intensity of packet arrivals at the queue. Thus, the network is congested as a result the average end-to-end delay increases linearly with time as shown in Figure 4-15.

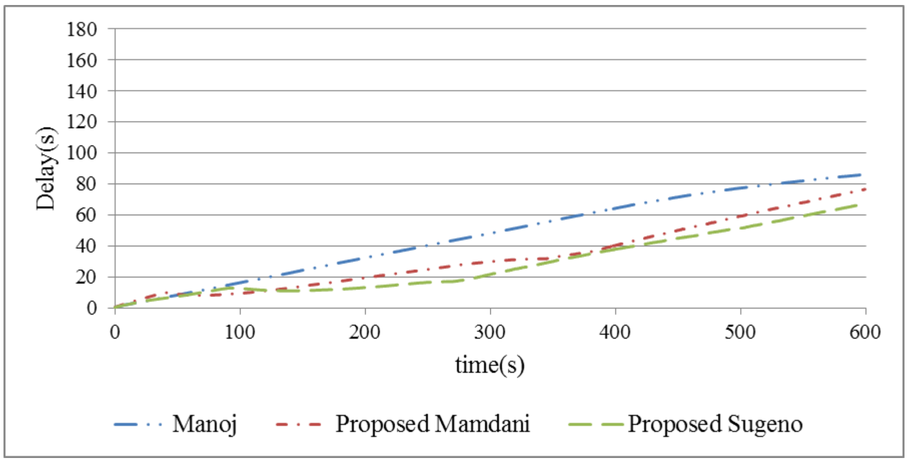


Figure 4-15: End-to-End delay at $\rho = 1$ for VBR Traffic

The behaviour of the network at $\rho = 1.33$ is shown in Figure 4-16; it is similar to Figure 4-15. The end-to-end delay graph at $\rho = 1.66$ and $\rho = 2$ also shows similar traits to Figure 4-15 and Figure 4-16. The values for the average end-to-end delay for all network loads are shown in Table 4-12.

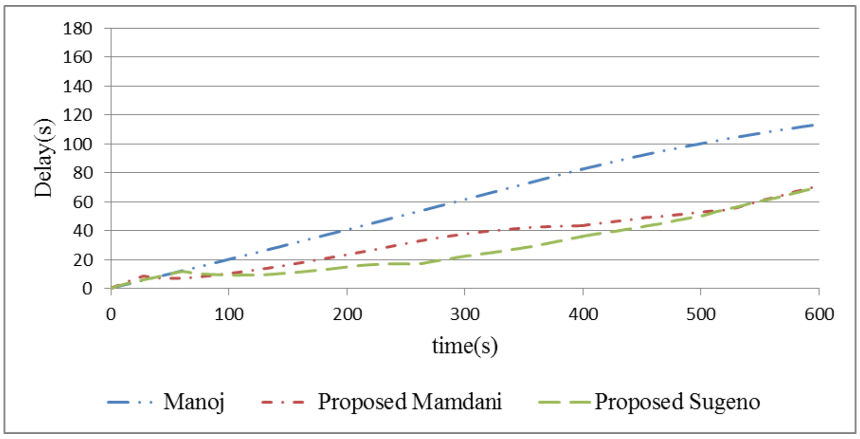


Figure 4-16: End-to-End delay at $\rho = 1.33$ for VBR Traffic

Table 4-12: Average End-to-End Delay for VBR Traffic

Scheduler	Average End-to-End delay (s)			
	ρ =1	ρ =1.33	ρ *=1.66*	ρ =2
Manoj	47.04	60.50	69.14	76.41
Proposed Mamdani	32.94	34.29	40.03	49.78
Proposed Sugeno	28.25	28.20	28.23	32.96
%improvement Mamdani	**29.98**	**43.33**	**42.10**	**34.85**
%improvement Sugeno	**39.94**	**53.39**	**59.16**	**56.87**

According to Table 4-12, the proposed Mamdani scheduler performs 29.98% better than Manoj at ρ =1, 43.33% better at ρ =1.33, 42.10% better at ρ =1.66, and 34.85% better at ρ =2. The performance of the proposed Mamdani scheduler improves more as the network load is increased from ρ =1 to 1.66 but drops slightly at ρ =2, as the network load is twice the network capacity. The proposed Sugeno scheduler also performs better than Manoj, the percentage improvement of the proposed Sugeno scheduler as compared to Manoj is 39.94, 53.39, 59.16 and 56.87% for ρ =1, 1.33, 1.66 and 2 respectively. The performance of the Sugeno scheduler improves as compared to Manoj and Mamdani as the traffic load increases from ρ =1 to 2. Thus by appropriately combining inputs features such as the SNR, Data Rate and Queue Size, the proposed algorithms scheduled packets better than Manoj for VBR traffic.

The gradient of the end-to-end delay is shown in Table 4-13. The gradient of the Mamdani scheduler is 20.02% less than Manoj at ρ =1, whilst that of Sugeno is 28.96% less than Manoj for a similar load; thus the network congestion is reduced by 20.00 and 28.96% for Mamdani and Sugeno, respectively. At an increased network load at ρ =1.33 and ρ =1.66, the gradient for end-to-end delay for the Mamdani scheduler is 45.61 and 40.61% lower than Manoj respectively. The performance slightly dropped to 33.92% when the traffic intensity increased to ρ = 2. The performance improvement of the gradient for the proposed Sugeno scheduler as compared to Manoj at ρ =1.33, 1.66 and 2 is 46.60, 52.47 and 60.60% respectively.

Table 4-13: Gradient of End-to-End delay for VBR traffic

Scheduler	Gradient of the End-to-End delay			
	ρ =1	ρ	ρ	ρ =2
Manoj	0.150	0.200	0.200	0.250
Proposed Mamdani	0.120	0.110	0.120	0.150
Proposed Sugeno	0.110	0.105	0.094	0.098
%improvement Mamdani	**20.02**	**45.61**	**40.61**	**40.30**
%improvement Sugeno	**28.96**	**46.60**	**52.47**	**60.60**

Figure 4-17 and Figure 4-18 show an increase in the PDR for the proposed schedulers (Mamdani and Sugeno) as compared to Manoj at ρ =1 and 1.33. The PDR for all network loads is shown in Table 4-14. The proposed fuzzy schedulers perform better with a higher PDR than Manoj.

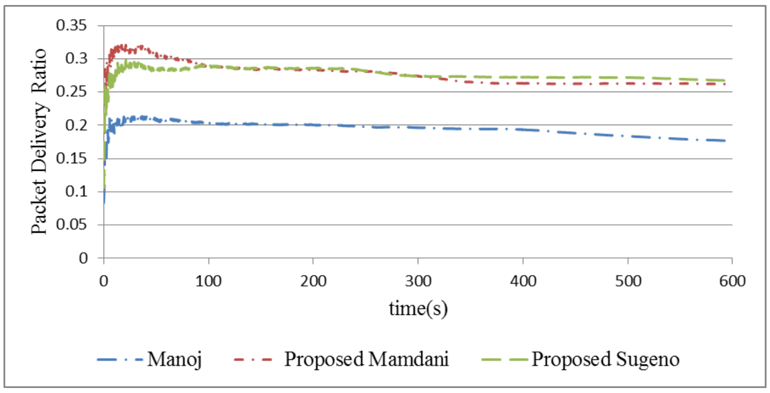


Figure 4-17: Packet delivery ratio at ρ =1 for VBR Traffic

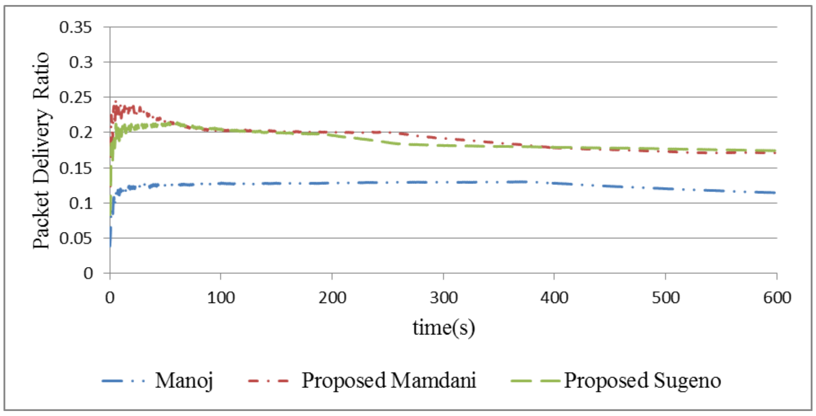


Figure 4-18: Packet delivery ratio at ρ =1.33 for VBR Traffic

Table 4-14: Packet Delivery ratio for VBR Traffic

Scheduler	PDR			
	ρ =1	ρ =1.33	ρ =1.66	ρ =2
Manoj	0.195	0.125	0.093	0.073
Proposed Mamdani	0.276	0.191	0.147	0.116
Proposed Sugeno	0.277	0.188	0.140	0.108
%improvement Mamdani	**42.03**	**53.00**	**57.77**	**58.56**
%improvement Sugeno	**42.53**	**50.12**	**49.64**	**47.38**

The percentage improvements of the PDR for the proposed Mamdani scheduler as compared to Manoj are 42.03, 53.00, 57.77 and 58.56% at ρ =1, 1.33, 1.66 and 2, respectively; whilst the percentage improvement for the proposed Sugeno scheduler are 42.53, 50.12, 49.64 and 47.38% at ρ =1, 1.33, 1.66 and 2, respectively. The proposed Sugeno scheduler delivered more traffic per second than the Manoj scheduler. The PDR of the proposed Mamdani scheduler performs better than the proposed Sugeno scheduler at ρ =1.33, 1.66 and 2.

4.4.4. Performance Analysis for Bursty Traffic

The characteristic of bursty traffic has been explained in section 2.4.1.3. The performance of the schedulers is analysed for bursty traffic. The end-to-end delay for traffic load ρ =1 and ρ =1.33 are shown in Figure 4-19 and Figure 4-20 respectively. On both occasions, the proposed schedulers performed better when compared to Manoj. However, the end-to-end delay for the proposed Sugeno scheduler showed the best performance.

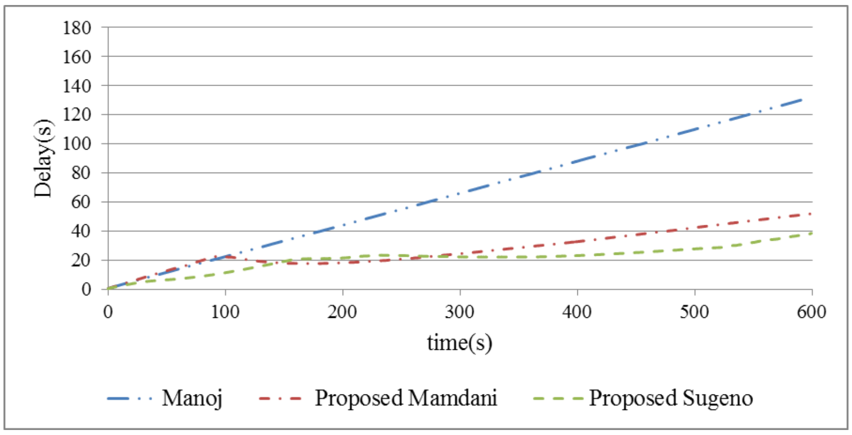


Figure 4-19: End-to-End delay at ρ =1 for Bursty Traffic

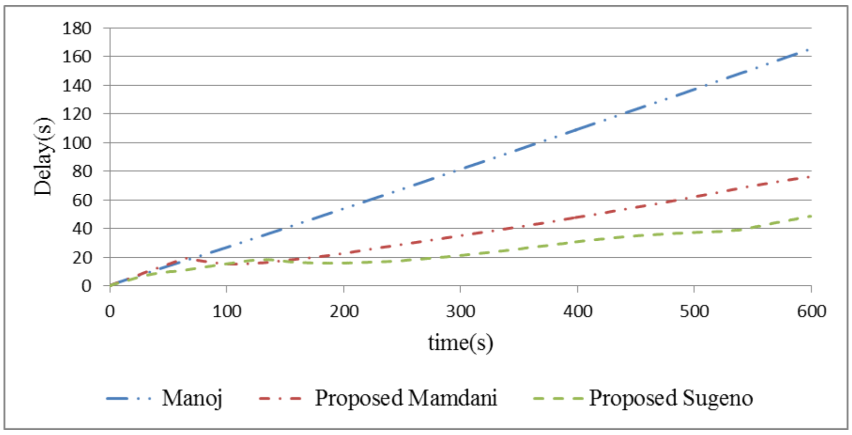


Figure 4-20: End-to-End delay at ρ =1.33 for Bursty Traffic

The average end-to-end delay at ρ = 1, 1.33, 1.66 and 2 are shown in Table 4-15. The percentage improvements of the Mamdani scheduler as compared to Manoj are 58.32, 54.08, 54.48, 52.60% and that of Sugeno scheduler are 68.02, 70.15, 71.36 and 66.88% at ρ =1, 1.33, 1.66 and 2 respectively. The improvement of the Sugeno algorithm increases as the network load increases from ρ =1 to 1.66 and then dropped at the network load ρ =2. This is because of the high level of congestion at ρ =2 as the network load is twice the network capacity. The proposed scheduling algorithms (Mamdani and Sugeno) performed better with bursty traffic as compared to CBR and VBR traffic. Thus, the algorithms adapt better to the changes in the queuing behavior of bursty traffic than Manoj.

Table 4-15: Average End-to-End delay for Bursty Traffic

Scheduler	Average End-to-End delay (s)			
	ρ =1	ρ =1.33	ρ =*1.66*	ρ =2
Manoj	66.08	81.90	95.98	105.84
Proposed Mamdani	27.54	37.61	43.69	50.16
Proposed Sugeno	21.13	24.45	27.49	35.06
% improvement Mamdani	**58.32**	**54.08**	**54.48**	**52.60**
%improvement Sugeno	**68.02**	**70.15**	**71.36**	**66.88**

The gradient of the end-to-end delay shows the rate of increase of congestion in the network, it is shown in Table 4-16. As the gradient increases, the network tends towards congestion. The rate of change of end-to-end delay with simulation time for Manoj scheduler increases linearly as shown in Figure 4-19 and Figure 4-20 respectively. Thus, the network congestion gets worse as the simulation time increases. The percentage decrease in the gradients of the end-to-end delay graph (similar to the percentage decrease in the network congestion) for the proposed Mamdani scheduler are 68.75, 57.48, 44.32 and 49.62%; whilst that for the Sugeno scheduler are 79.96, 76.64, 74.78 and 71.69% at ρ =1, 1.33, 1.66 and 2 respectively.

Table 4-16: Gradient of the End-to-End Delay for Bursty Traffic

Scheduler	Gradient of the End-to-End delay			
	ρ =1	ρ =1.33	ρ	ρ =2
Manoj	0.220	0.280	0.280	0.350
Proposed Mamdani	0.070	0.120	0.150	0.180
Proposed Sugeno	0.040	0.064	0.069	0.100
%improvement Mamdani	**68.75**	**57.48**	**44.32**	**49.62**
%improvement Sugeno	**79.96**	**76.64**	**74.78**	**71.69**

The PDR at ρ =1 and ρ =1.33 are shown in Figure 4-21 and Figure 4-22 respectively. The PDR decreases as the traffic intensity increases (ρ), this has been explained in section 2.4.2.1.B. The proposed schedulers (Mamdani and Sugeno) performed better than the existing scheduler (Manoj). However, the proposed Mamdani scheduler showed better performance than the proposed Sugeno scheduler did.

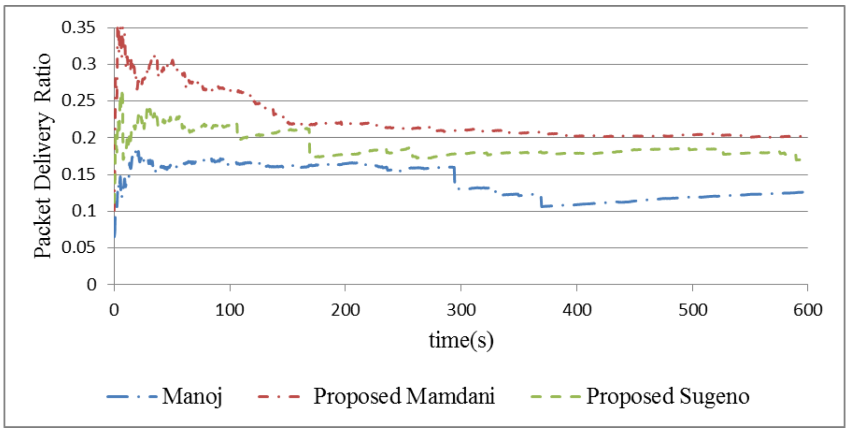


Figure 4-21: Packet delivery ratio at ρ =1 for Bursty Traffic

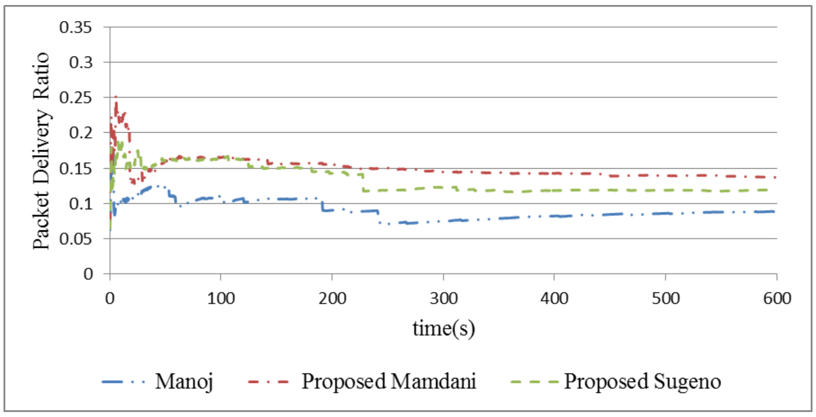


Figure 4-22: Packet delivery ratio at ρ =1.33 for Bursty Traffic

The PDR for all network load is shown in Table 4-17. The percentage improvement in the PDR increased between ρ =1 and 1.66 for both proposed schedulers as compared to Manoj. However, the PDR decreased to 16.92% for the proposed Mamdani scheduler at ρ =2, and also for the proposed Sugeno scheduler for similar network load. The Manoj scheduler performed 17.25% better than the Sugeno at ρ =2, which is the reason for the negative sign in Table 4-17. This traffic load is double the network capacity.

Table 4-17: Packet delivery ratio for Bursty Traffic

Scheduler	PDR			
	ρ =1	ρ	ρ =1.66	ρ =2
Manoj	0.140	0.091	0.047	0.071
Proposed Mamdani	0.224	0.149	0.114	0.083
Proposed Sugeno	0.189	0.132	0.095	0.059
%improvement Mamdani	**60.12**	**64.41**	**140.50**	**16.92**
%improvement Sugeno	**35.05**	**45.83**	**99.99**	**-17.25**

4.5. Summary

This chapter proposed two optimised adaptive fuzzy packet scheduling algorithms based on the Mamdani and Sugeno FIS. These scheduling algorithms were compared to an existing fuzzy scheduler. The proposed schedulers considered three inputs (Data Rate, Queue Size, SNR) as opposed to the existing scheduler, which considered two inputs (Data Rate and Channel Capacity). The inputs to the fuzzy system (Mamdani and Sugeno) are fuzzified, implicated, aggregated and defuzzified to obtain the crisp value. The crisp value ranges from 0-1, and it represents the packet priority index. Zero ‘0’ is the highest priority and 1 is the lowest priority. Thus, packet priority in the queue reduces as the priority index increase from

0-1. The performances of both schedulers were analysed using measuring metric such as end-to-end delay and PDR. The Mamdani scheduler is more computationally complex than the Sugeno scheduler.

The performance of the proposed scheduling algorithms (Mamdani and Sugeno) is analysed for CBR, VBR and Bursty traffic.

The end-to-end delay for the proposed schedulers outperformed the existing for CBR traffic, whilst the PDR is very close. This is as results of the characteristics of CBR traffic that consist of a constant data rate for all simulation time, thus if the network is congested all its available resources are efficiently used.

The proposed Mamdani and Sugeno scheduler performed better than Manoj under all traffic loads for VBR traffic. The proposed Sugeno scheduler performed better than the proposed Mamdani in reducing the end-to-end. The PDR decreases as the network load increases because of congestion.

The end-to-end delay for the proposed schedulers performed better than the existing scheduler for bursty traffic at all network loads. The proposed Sugeno scheduler showed the best end-to-end delay performance when compared to the proposed Mamdani and Manoj. The performance of the PDR for the proposed Mamdani is better than Sugeno and Manoj. At ρ =2, the PDR for the Sugeno scheduler performed worse than the Manoj scheduler by approximately 17%.

The network is congested, thus as the number of packet dropped increases, the level of re-transmission of lost packets will also increase causing high network overhead that will result in high delay and packet loss. Some packets transmitted will transverse one or multiple intermediate nodes to get to the destination; this contributes to the increase in network delay. Some mobile nodes may remain out of the sensing range from other source nodes. This results in a hidden node problem, where some nodes transmit packets without the knowledge that some other nodes might also be involved in the transmission. This can affect the performance of the network by increasing the end-to-end delay and packet loss.

Overall, the proposed schedulers performed better than existing for all traffic class, under varying network loads. The performance of the proposed schedulers improves as the burstiness of the traffic increases.

The simulation results show there is not a significant difference between the performance of the Mamdani and Sugeno scheduler for CBR and VBRtraffic. The proposed Sugeno scheduler showed better performance than the proposed Mamdani scheduler for Bursty traffic. Thus, the proposed Sugeno scheduler will be the better choice for real-time application because of the simplicity of its design and it is less computationally complex.

The next chapter builds on this chapter to create an interface between the OPNET simulation software and external hardware to send real-time video data through the network. The performance of the proposed schedulers will be analysed under mixed traffic condition.

Chapter 5 Real time video streaming over a MANET

5.1. Introduction

A majority of the current research on MANET heavily relies on simulation models because it is cost effective. Thus, simulation software such as NS3, OMNET++ and OPNET are used for MANET designs. In order to validate the performance of MANET, there is the need to combine OPNET with external hardware for a more realistic network analysis. This will provide a cost effective real life validation of research work, thus, reducing simulation errors as simulation assumptions are replaced with more realistic network data. Researchers have the added flexibility to adjust the network parameters as often as needed when analysing a network performance. Research work on applications such as the search and rescue robot, medical surgery, pipe inspections, etc. can be validated.

This chapter creates a real-time video streaming over a simulated wireless network (MANET) in virtual terrain for control purposes. MANET models are expensive to implement, with real life design complexity. For this reason, to validate a complex simulation model in OPNET, there is a need to interface the OPNET model with external devices. OPNET is interfaced with a webcam that streams real-time video through the network. The performance of the video stream is analysed under varying background load. The modelling of MANET is done in OPNET as in the previous chapter, and an interface is created between the source node and a webcam. The source node captures real-time video stream from the webcam and transmits it through the simulated network; this video stream can be viewed at the destination node. OPNET currently have an avenue for connectivity with external device using System-in-the-loop (SITL), but this requires a special expensive license, and an expensive dedicated wireless hardware device, it is also complex to implement. This chapter presents a cheaper alternative and a more simplified design. The performance of the video stream will be analysed after scheduling is applied. Background traffic is gradually introduced into the network at varying traffic intensity, and the performance of the video stream and the background traffic is analysed. The background traffic is non-real-time traffic.

5.2. Related Work on Video Streaming using OPNET

Most of the research work that involves the implementation of a real life system over OPNET uses SITL tool. The SITL module in OPNET provides an avenue/interface for real live connectivity between hardware or software applications or devices and the OPNET discrete event simulation model [104]. Paper [105] presents methodology for implementing a real-time tactical network using physical wireless communication devices over OPNET. A technical unit is connected to a wireless simulated model of tactical radio on a virtual terrain. The paper concludes that, although SITL modules have some deficiencies, as some important devices are not supported (one of which is a wireless bridge), creating an interface between simulation and hardware will have some significant impact in the area of tactical network. Paper [106] also used SITL; it presents a methodology to develop an OPNET simulation with a test network for real video streaming between real transmitter and receiver over a simulated wireless link in a virtual terrain. Paper [107] designs a simulation to real-life to simulation network environment to capture real-time video over a wireless virtual environment. The OPNET SITL is used in the design of the real-time video streaming platform using the Hybrid transport layer protocol. It concludes that, this will provide a new approach for communication network modelling, which will provide the opportunity to evaluate a network property. Paper [108] also implemented and showed the advantage of combining a real network with a simulated OPNET WLAN network.

5.3. System-in-the- loop (SITL)

The SITL module in OPNET provides an avenue/interface for real live connectivity between hardware or software applications or devices and the OPNET discrete event simulation model. OPNET simulation model will be able to exchange packet with a real device or another external network model with the use of SITL. These have made real-time practical testing of a simulated OPNET model with either an external hardware or software easier. The prototype software or hardware, to be tested, can interact with a number of virtual devices present in OPNET, thus, making it easier to carry out real test experiment. The external device, which enables simulation to exchange packets is called SITL gateway. The WinPcaP Window Library helps to route user selected packet/data (user defined filter), from an Ethernet network adaptor to a simulation network [104].

5.4. Simulation Environment and Methodology

A simulation model is designed to analyse the performance of the real-time video stream under varying CBR and VBR background load. The CBR and VBR traffic profile have been discussed in section 2.4.1.1 and section 2.4.1.2 respectively. The simulation environment, methodology, and analysis are presented in this section.

5.4.1. Simulation Parameters

The simulation model for the real-time video stream with CBR and VBR background traffic is considered for stationary nodes and mobile nodes as shown in Table 5-1. Both the stationary and mobile nodes simulation model consists of four wireless nodes over a network simulation area of 500m X 500m. The wireless propagation model in Section 3.4.5 is applied. The overall network architecture is shown in Figure 5-1. The density of nodes will have an effect on the performance of the network, however four nodes have been chosen because it is close to the minimum needed to demonstrate the performance of the scheduler with realistic traffic.

Node 1 is the source node; node 2 and node 3 are the intermediate nodes whilst node 4 is the destination node. The video stream as well as the background traffic is sent from the source (node 1) to the destination (node 4). A Logitech webcam is connected to the computer running OPNET. Node 1 (source) captures the video stream from the webcam connected to the PC; it sends the streams to node 4 (destination) via the intermediate nodes 2 and 3. The video stream can be viewed at the destination node. The video caption, as viewed from the source and destination, is shown in Figure 5-2.

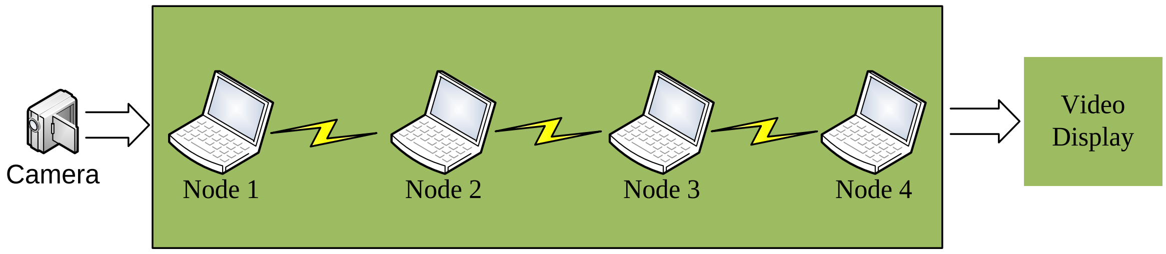


Figure 5-1: Video Streaming Model

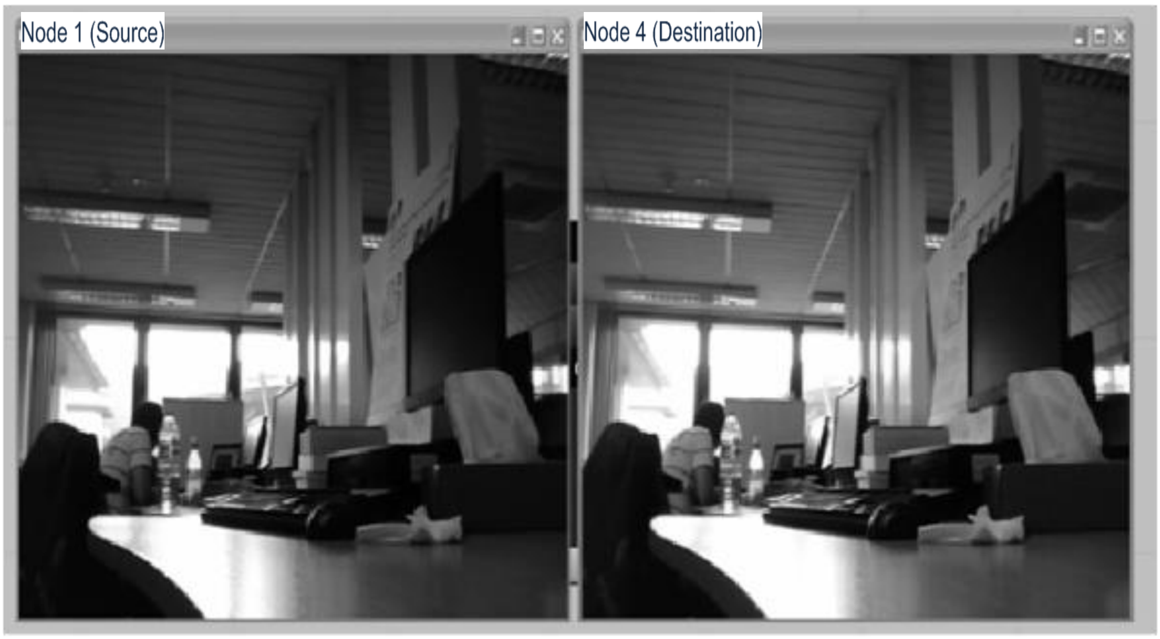


Figure 5-2: Video display for Source and Destination

The interface between OPNET and the webcam is created using the Open source Computer Vision library (OpenCV), a C programming library. The implementation is shown in Appendix G. The highlighted parts of the codes are the changes made to the standard model to accommodate the video streaming in the node 1 and video display at node 4. The captured video frame format is not supported by OPNET. Thus, a custom packet is created within the OPNET simulation model to encapsulate each captured frame from the camera before transmitting it through the network. The packet is sent from node 1 to node 4 via intermediate nodes (node 2 and node 3). At node 4, the packet is de-capsulated, and the video can be viewed.

The resolution of the video frame is 160 x 120 pixels. This is converted to bytes to give the required payload size. Each pixel in a coloured image has three channels (red, green and blue), these colours are represented by 1-byte. Thus, individual pixels in a coloured image consist of 3-byte. A grey scale image consists of a single channel, which is 1-byte. Each byte consists of 8-bits; hence, a 160 x 120 pixels grey scale image would occupy a size of 160 x 120 x 1 bytes, which is 19,200 bytes or 153,600 bits. For simplicity, the video stream is converted to grey scale before being sent through the network. The maximum allowable packet size or Maximum Transmission Unit (MTU) that can be transmitted through OPNET network is 2,304 bytes i.e. 18,432 bits. This is the maximum frame body length defined in IEEE 802.11 standard [109]. Packets greater than the maximum allowable size are fragmented before being sent through the network, and it is re-assembled at the destination node before being displayed. The simulation parameters for stationary and mobile nodes are shown in Table 5-1.

Table 5-1: Simulation Parameters for Stationary and Mobile Nodes

No. of Nodes	4
Area	500m x 500m
Simulation Time	600s
Mobility Model	Stationary Nodes-None
	Mobile Nodes – Random Waypoint
Propagation model	Shadowing model
Data payload	153,600bits / 19,200bytes
Frame rate	10fps
Traffic intensity ‘ρ’	0.33, 0.45, 0.60, 0.75, 0.90, 1.05, 1.55 and 2.05
MAC protocol	IEEE 802.11n (Buffer Size = 16MB)

The OPNET network topology for stationary and mobile nodes is shown in Figure 5-3 and Figure 5-4, respectively. The OPNET node model ‘MANET station fixed’ is used for the stationary nodes whilst the ‘MANET station mobile’ is used for the mobile nodes. According to Figure 5-3, Node_1 is the source node, it transmit the video stream and background traffic; Node_2 and Node_3 are the intermediate nodes, whilst Node_4 is the destination node for the video and background traffic. This is similar to Figure 5-4, where Mobile_1 transmits the video stream and background traffic, Mobile_2 and Mobile_3 are the intermediate nodes, whilst Mobile_4 is the destination.

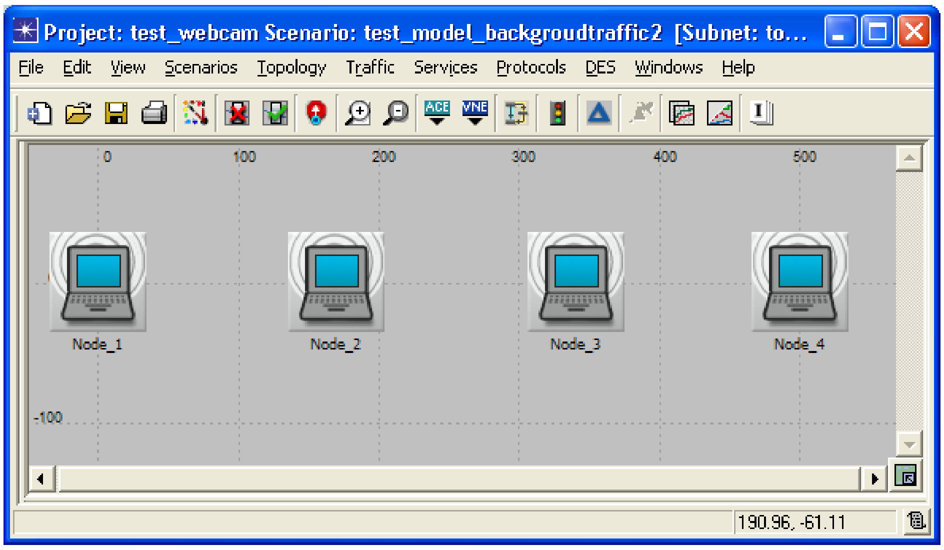

Project: test_webcam Scenario: test_model_backgroudtraffic2 [Subnet: to...
File Edit View Scenarios Topology Traffic Services Protocols DES Windows Help
0
100
200
300
400
500
Node_1
Node_2
Node_3
Node_4
-100
190.96, -61.11

Figure 5-3: Stationary nodes (Ad-hoc network)

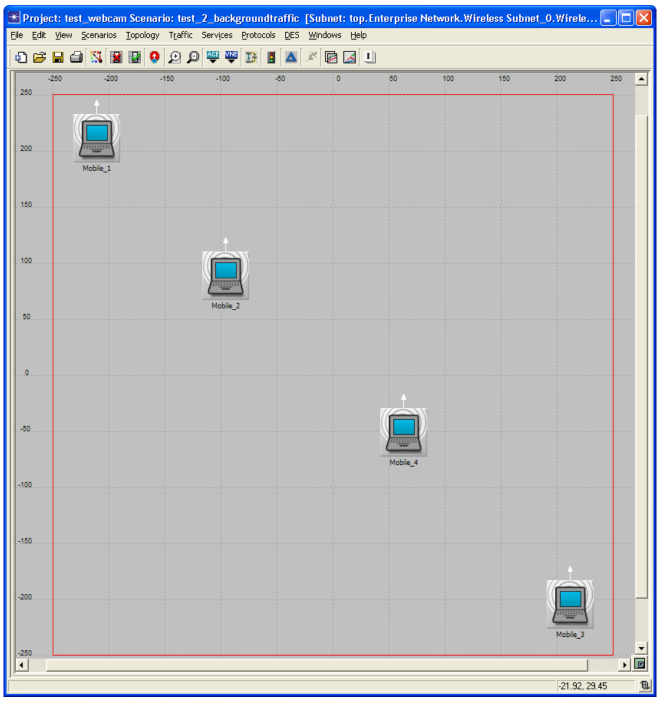

Project: test_webcam Scenario: test_2_backgroundtraffic [Subnet: top.Enterprise Network.Wireless Subnet_0.Wirele...
File Edit View Scenarios Topology Traffic Services Protocols DES Windows Help
Mobile_1
Mobile_2
Mobile_4
Mobile_3
-21.92, 29.45

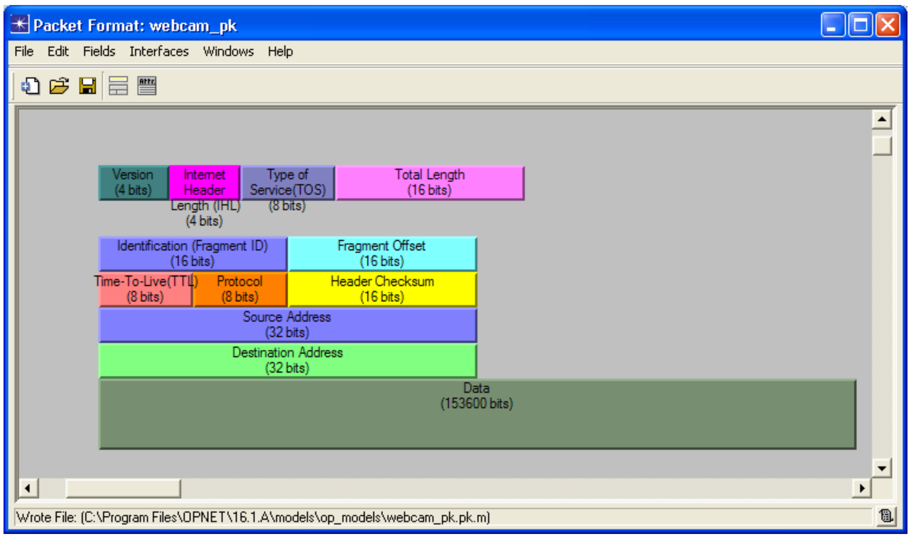
Figure 5-4: Mobile nodes (MANET)

The source nodes transmit video streams at a rate of 10fps, with a frame size of 153,600 bits, the traffic intensity ‘ ρ ’ according to section 3.2.2 with no background load is 0.33;

this will form the baseline upon which other higher traffic intensity will be compared. An example hardware device that can interface the simulation network is the DRK8080. The DRK8080 is an integrated high bandwidth WIFI 802.11 Robot that supports video streaming, audio and sensors information. The system can upload the sensor as well as stream video (up to 4fps) to home PC [110]. The traffic intensity ρ is increased gradually from the baseline (ρ =0.33) by introducing background traffic, and the performance of the video stream when FIFO, Manoj and both proposed schedulers are applied will be analysed. The different level of background traffic introduced as shown by the traffic intensity ρ in Table 5-1 represents 0, 26, 44, 56, 63, 68, 78 and 84% respectively of the total traffic in the network. When the traffic intensity ρ is greater than or equal to 1, the network is congested. The proposed Mamdani and Sugeno schedulers assess individual packets and assign the highest Priority Index to the video traffic because of their sensitivity to delay and loss. The previous simulation in Chapter 4 did not consider a network with mixed traffic. The priority of the background traffic is determined as explained in section 4.3.1.

5.4.2. Packet Format

The packet format is created using the packet format editor in OPNET. This packet format editor enables the design of the internal structure of a packet as a set of fields. The size of the individual box is proportional to the number of bits specified as the field size. A sample packets format is shown in Figure 5-5. The encapsulation and decapsulation of the video frames are shown in Appendix G.


Packet Format: webcam_pk
File Edit Fields Interfaces Windows Help
Version (4 bits)
Internet Header Length (IHL) (4 bits)
Type of Service(TOS) (8 bits)
Total Length (16 bits)
Identification (Fragment ID) (16 bits)
Fragment Offset (16 bits)
Time-To-Live(TTL) (8 bits)
Protocol (8 bits)
Header Checksum (16 bits)
Source Address (32 bits)
Destination Address (32 bits)
Data (153600 bits)

Wrote File: (C:\Program Files\OPNET\16.1.A\models\op_models\webcam_pk.pk.m)

Figure 5-5: Custom OPNET Packet format

5.4.3. Results and Analysis of the Video Stream

The results and analysis of the video stream are presented for all schedulers. The performance of the video stream, when CBR and VBR background traffic is introduced, is presented in this section.

5.4.3.1. Baseline for Stationary and Mobile Nodes Model

The default scheduler in MANET is FIFO. The end-to-end delay for stationary and mobile nodes model at $\rho = 0.33$ (no background traffic) for FIFO, Manoj, proposed Mamdani and Sugeno is shown in Figure 5-6 and Figure 5-7 respectively, these form the baseline ($\rho = 0.33$) for performance analysis. The average end-to-end delay for the stationary nodes is 0.068s for all schedulers as shown in Figure 5-6. The average end-to-end delay for mobile nodes varies among the various schedulers. The average end-to-end delay for FIFO and Manoj scheduler is 0.079 and 0.082s respectively, whilst that for the proposed Mamdani and Sugeno scheduler is 0.068s. The mobile natures of nodes in MANET have added further challenges to the network thus increasing the baseline delay for FIFO and Manoj scheduler. The proposed Mamdani and Sugeno scheduler shows a better result for mobile nodes than FIFO and Manoj as it maintains the same delay (0.068s) as stationary nodes. This is because packets are efficiently scheduled, and the schedulers adapt well to the network topology change. The baseline delay is the minimum average end-to-end delay of the video stream for each scheduler without any background traffic. It is expected that this delay will increase when the queue is not properly scheduled as background traffic is gradually introduced into the network.

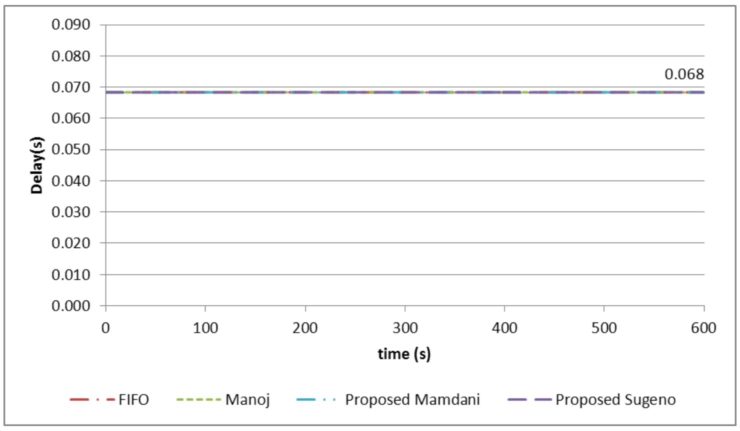


Figure 5-6: Stationary Nodes - Baseline Delay for Video Traffic at ρ =0.33

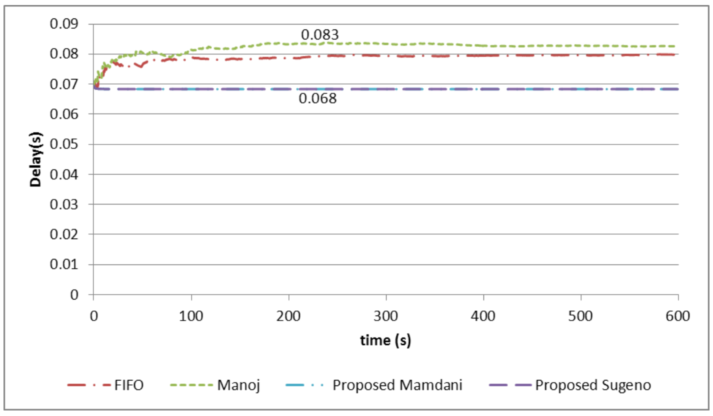


Figure 5-7: Mobile Nodes - Baseline Delay for Video Traffic at ρ =0.33

The PDR for stationary nodes model is shown in Figure 5-8 and Figure 5-9. Both the stationary and mobile nodes model consist only of video traffic and their traffic intensity is low at this traffic level as the network is not congested, thus, all traffic sent by the source node is received at the destination node. This forms the baseline for PDR performance analysis.

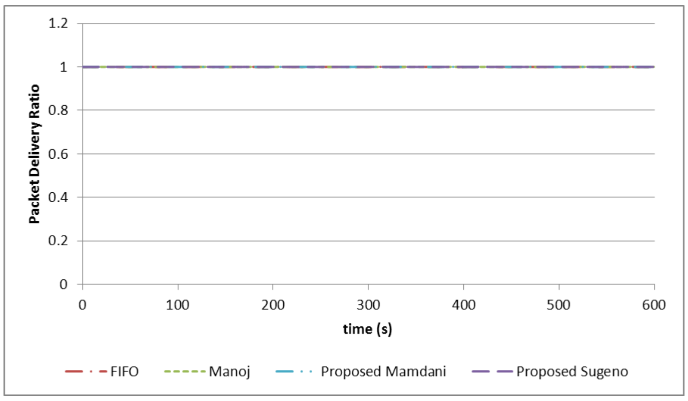


Figure 5-8: Stationary Nodes: Baseline PDR for Video Traffic at ρ =0.33

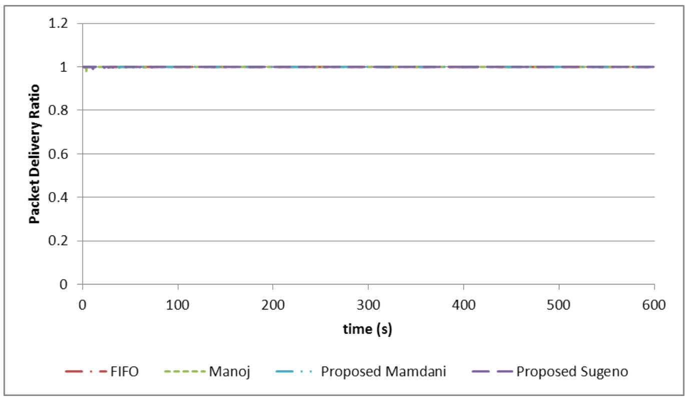


Figure 5-9: Mobile Nodes: Baseline PDR for Video Traffic at ρ =0.33

The performance of the baseline is the best achievable end-to-end delay and PDR by each scheduler for the video stream with no background traffic.

5.4.3.2. Video stream with CBR Background Traffic

Applying the fuzzy scheduler in section 4.3, the performance of the real-time video stream is analysed under increasing CBR background load. The characteristics of CBR traffic have been explained in section 2.4.1.1. The performance of the proposed schedulers (Mamdani and Sugeno) is compared to FIFO and existing scheduler (Manoj). The results and analysis for stationary and mobile nodes are presented in this section.

A. Stationary Nodes

The end-to-end delay for video at ρ =0.45 after the introduction of CBR background traffic is shown in Figure 5-10. The proposed Mamdani and Sugeno scheduler maintains the baseline delay whilst the end-to-end delay for FIFO and Manoj scheduler show a 40% increase from the baseline in Figure 5-6.

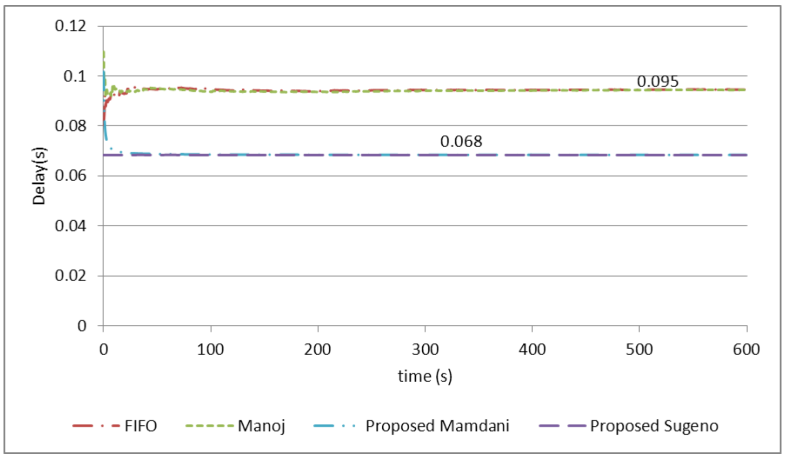


Figure 5-10: Stationary Nodes: Delay for Video Traffic at ρ =0.45 with CBR Background Traffic

The end-to-end delay for the CBR background traffic at ρ =0.45 is shown in Figure 5-11. The FIFO and Manoj scheduler show the same level of performance for the entire simulation duration. The proposed Mamdani and Sugeno scheduler also show the same level of performance for the entire simulation duration. The proposed Mamdani and Sugeno performs approximately 7% better than the FIFO and Manoj.

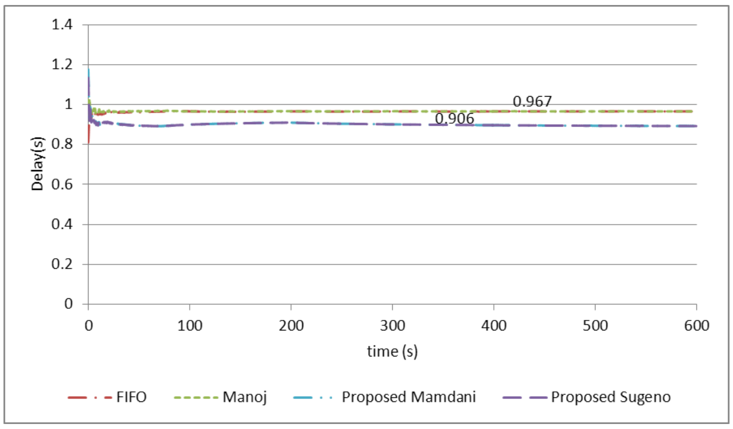


Figure 5-11: Stationary Nodes: Delay for CBR Background Traffic at ρ =0.45

The PDR for video traffic is shown in Figure 5-12. It shows a small amount of packet losses for FIFO and Manoj scheduler between 0-2s simulation time; afterwards it becomes stable at one all through the simulation time. All traffic sent by the source is received at the destination.

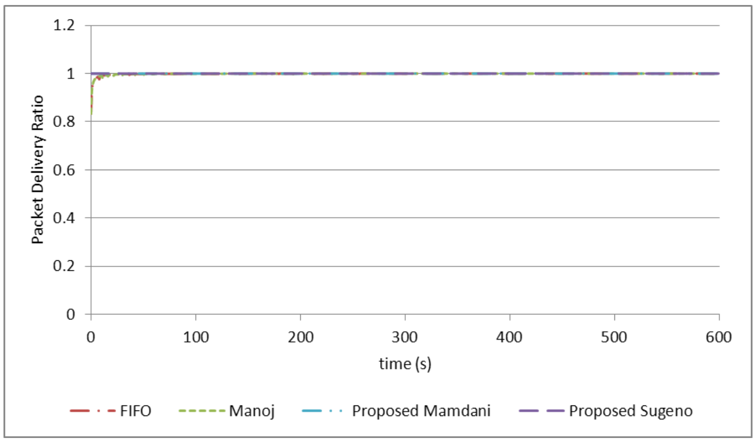


Figure 5-12: Stationary Nodes: PDR for Video Traffic at ρ =0.45 with CBR Background Traffic

The average end-to-end delay for the video traffic for all ρ is shown Table 5-2. The FIFO and Manoj scheduler did not maintain the baseline delay, the end-to-end delay for both schedulers increases as the traffic intensity increases. The only exception is at ρ =0.75 where Manoj shows a very high increase in delay before dropping to 194s at ρ =0.90. The proposed Mamdani and Sugeno maintained the baseline delay until when ρ =1.05, at this stage the network is congested. At ρ =1.05, the proposed Mamdani and Sugeno show a 13 and 106% increase from the baseline as in Figure 5-6, which is much better than 13,144% and 75,906% increase by the FIFO and Manoj scheduler respectively.

The traffic intensity at ρ =1.55 and 2.05 presents a network load that is 55 and 105% more than the network capacity respectively. The end-to-end delay for the video traffic for all schedulers at ρ =1.55 and 2.05 is shown in Figure 5-13 and Figure 5-14 respectively. The

Table 5-2: Stationary Nodes: Average Delay for Video Traffic with CBR Background Traffic

ρ	FIFO (s)	Manoj (s)	Proposed Mamdani (s)	Proposed Sugeno (s)
0.33	0.068	0.068	0.068	0.068
0.45	0.094	0.094	0.069	0.068
0.6	0.118	0.117	0.069	0.068
0.75	0.210	36.579	0.069	0.068
0.90	0.347	0.194	0.069	0.068
1.05	9.006	51.684	0.077	0.140
1.55	22.310	23.396	10.205	16.617
2.05	23.953	51.083	9.934	6.087

FIFO and Manoj schedulers show a higher rate of increase from zero between simulation times 0-100s, the rate of increase of the delay with time is much lower afterwards. The proposed schedulers were stable with low delay between simulation times 0-80s before increasing sharply between simulation times 80-100s; the rate of change of the delay is much lower from around simulation time 100s onwards. The behaviour of the delay graph is caused by high level of network congestion. When the traffic intensity increases to ρ =2.05, the performance of the FIFO and Manoj decreases whilst the proposed schedulers show better performance than ρ =1.55. The queue fills up quicker at ρ =2.05 than ρ =1.55 and remains that way for the entire simulation duration. All arriving packets to the queue when it is full are discarded, the traffic intensity at ρ =2.05 will lose more traffic than ρ =1.55 because it has more packets arriving per second than ρ =1.55. The size of the queue as well as the data rate plays an important role in the calculation of the packet Priority Index. The proposed scheduler shows better performance as compared to FIFO and Manoj at ρ =2.05. The sharp increase in the delay for the proposed schedulers, as shown in Figure 5-14, happens much sooner than in Figure 5-13 because of the much higher traffic intensity causing a high level of network congestion.

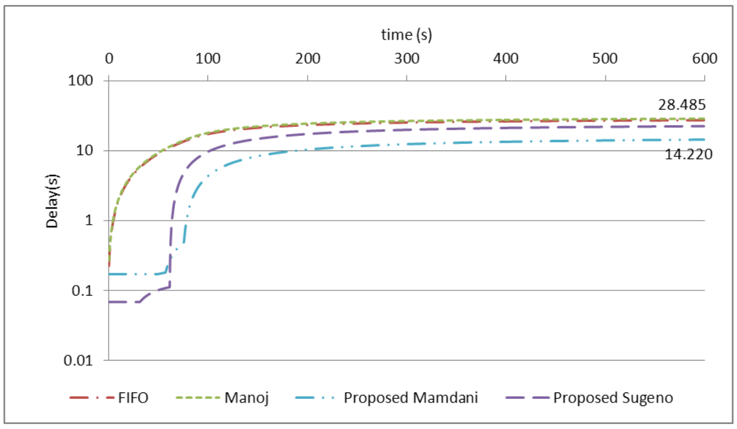


Figure 5-13: Stationary Nodes: Delay for Video Traffic at ρ =1.55 with CBR Background Traffic

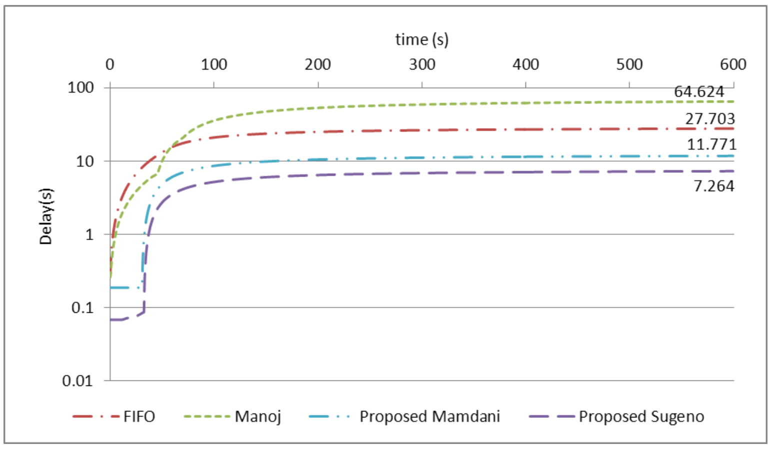


Figure 5-14: Stationary Nodes: Delay for Video Traffic at ρ =2.05 with CBR Background Traffic

The average end-to-end delay for the background traffic for ρ =0.45-2.05 is shown in Table 5-3. The proposed Mamdani and Sugeno scheduler show a better performance as compared to the FIFO and Manoj scheduler for all levels of ρ with an exception at ρ =2.05 where the Manoj scheduler show a better performance than the proposed Sugeno.

The end-to-end delay for the FIFO scheduler increases as the load increases. The Manoj scheduler performs worse because it is not capable of dealing with mixed traffic.

Table 5-3: Stationary Nodes: Average Delay for CBR Background Traffic

ρ	FIFO (s)	Manoj (s)	Proposed Mamdani (s)	Proposed Sugeno (s)
0.45	0.965	0.966	0.900	0.900
0.6	2.900	2.869	2.150	2.150
0.75	5.392	1,146.331	4.010	4.004
0.90	8.620	1,747.292	7.229	7.215
1.05	1,535.220	2,616.989	854.950	446.582
1.55	3606.454	3720.084	2561.122	2778.923
2.05	4301.322	3772.613	2446.599	4185.816

The PDR for the video traffic for the various ρ is shown in Table 5-4. The proposed Mamdani and Sugeno scheduler shows the best performance for all levels of ρ, it slightly dropped at ρ =1.05 as a result of congestion. The Manoj scheduler begins to drop traffic significantly from ρ =0.6.

Table 5-4: Stationary Nodes: Average PDR for Video Traffic with CBR Background Traffic

ρ	FIFO	Manoj	Proposed Mamdani	Proposed Sugeno
0.33	1.000	1.000	1.000	1.000
0.45	0.999	0.999	1.000	1.000
0.6	0.998	0.998	1.000	1.000
0.75	0.998	0.728	1.000	1.000
0.90	0.998	0.623	1.000	1.000
1.05	0.938	0.548	0.998	0.994
1.55	0.432	0.433	0.659	0.604
2.05	0.271	0.276	0.431	0.444

The PDR for ρ =1.55 and 2.05 is also shown in Figure 5-15 and Figure 5-16 respectively. As shown in Figure 5-15, the proposed schedulers start to lose packets after 90s of simulation time. The loss is higher and even occurs much earlier in Figure 5-16 because of the higher traffic intensity resulting in the high level of network congestion. Despite the high packet loss rate, the proposed schedulers perform better than the FIFO, and Manoj scheduler does. The high packet loss rate is due to high level of network congestion.

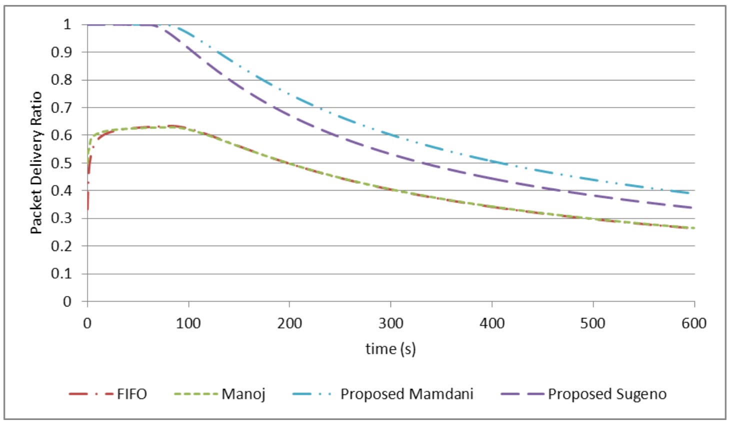


Figure 5-15: Stationary Nodes: PDR for Video Traffic at ρ =1.55 with CBR Background Traffic

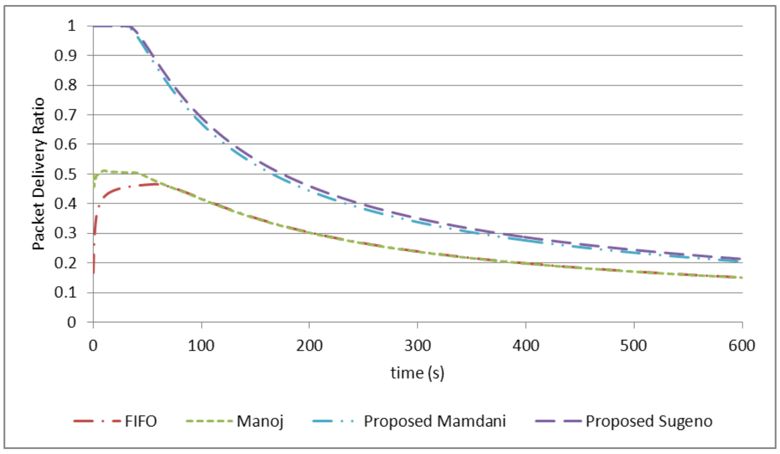


Figure 5-16: Stationary Nodes: PDR for Video Traffic at ρ =2.05 with CBR Background Traffic

The PDR for the background traffic for all values of ρ is shown in Table 5-5. The proposed Mamdani and Sugeno scheduler along with the FIFO scheduler maintains a high PDR for ρ between 0.45 and 1.05. Thus, almost all traffic sent by the source is received at the destination. The PDR of the Manoj scheduler begins to degrade at ρ =0.75.

Table 5-5: Stationary Nodes: Average PDR for CBR Background Traffic

ρ	FIFO	Manoj	Proposed Mamdani	Proposed Sugeno
0.45	0.965	1.000	1.000	1.000
0.6	0.999	0.999	1.000	1.000
0.75	0.999	0.783	0.999	0.999
0.90	0.999	0.729	0.999	0.999
1.05	0.940	0.707	0.912	0.913
1.55	0.693	0.684	0.631	0.648
2.05	0.522	0.517	0.492	0.490

A. Mobile Nodes

The end-to-end delay for the video stream for mobile nodes with CBR background traffic at ρ =0.45 is shown in Figure 5-17. The proposed Mamdani and Sugeno maintain the baseline delay in Figure 5-7 whilst the FIFO and Manoj scheduler shows approximately 22 and 34% degradation from its baseline. It also shows 41 and 62% degradation from the baseline of the proposed schedulers respectively.

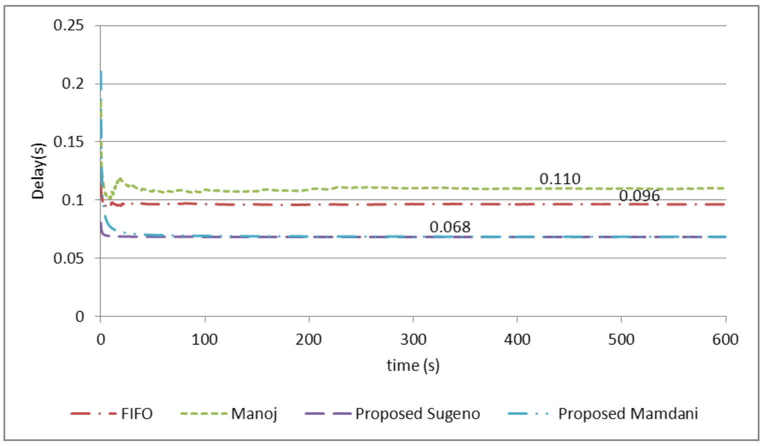


Figure 5-17: Mobile Nodes: Delay for Video traffic at ρ =0.45 with CBR Background Traffic

The end-to-end delay for the background traffic is shown in Figure 5-18. The proposed Mamdani and Sugeno scheduler show similar performance for the entire simulation duration. The FIFO and Manoj scheduler also shows similar performance for the entire simulation duration. The proposed Mamdani and Sugeno show a 10% improvement in performance than the FIFO and Manoj even though preference is given to the video stream.

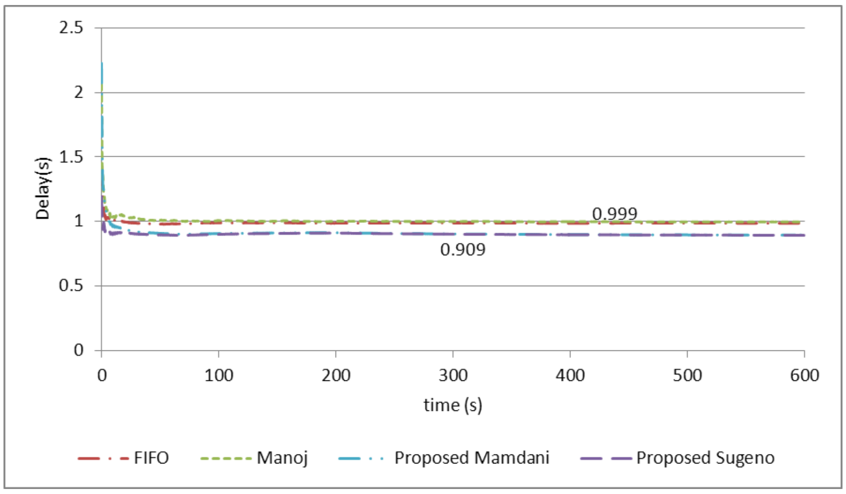


Figure 5-18: Mobile Nodes: Delay for CBR Background Traffic at ρ =0.45

The PDR for the video traffic at ρ =0.45 is shown in Figure 5-19. The total traffic sent by the source is less than the queuing capacity. Thus, all traffic sent is received at the destination at maximum achievable throughput as shown in Figure 5-19.

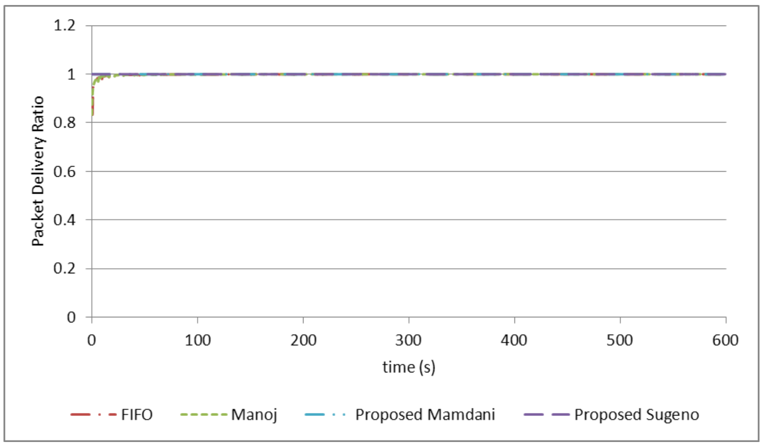


Figure 5-19: Mobile Nodes: PDR for Video Traffic at ρ =0.45 with CBR Background Traffic

The end-to-end delay for the video traffic for all ρ is shown in Table 5-6. The performance of the proposed Mamdani and Sugeno scheduler is close to the baseline in Figure 5-7 as compared to the FIFO and Manoj scheduler. The delay of the FIFO scheduler increases as the load increases whilst the Manoj scheduler shows a high delay at ρ =0.60. The performance of the proposed Mamdani and Sugeno scheduler degrades at ρ =1.05, their delays are 0.229 and 0.247s respectively; which is a 237 and 263% degradation from the baseline delay. The FIFO and Manoj scheduler shows a much higher delay degradation of 13,713 and 70,653% respectively from the baseline of the proposed schedulers shown in Figure 5-7.

Table 5-6: Mobile Nodes: Delay for Video Traffic with CBR Background Traffic

ρ	FIFO (s)	Manoj (s)	Proposed Mamdani (s)	Proposed Sugeno (s)
0.33	0.079	0.082	0.068	0.068
0.45	0.096	0.110	0.069	0.068
0.6	0.119	18.266	0.070	0.069
0.75	0.210	0.225	0.069	0.069
0.90	0.349	50.275	0.070	0.069
1.05	9.393	48.112	0.229	0.247
1.55	39.728	67.582	3.269	1.821
2.05	31.298	49.448	1.428	17.420

The end-to-end delay for ρ =1.55 and 2.05 is shown in Figure 5-20 and Figure 5-21 respectively. As shown in Figure 5-20, the proposed schedulers perform better than the FIFO, and Manoj scheduler did. The proposed schedulers perform better when the nodes are mobile as compared to when they are stationary, whilst the FIFO and Manoj perform worse. The proposed schedulers adapt to the network topology change better than the FIFO and Manoj. The proposed Mamdani scheduler performs better than the proposed Sugeno scheduler did. As the traffic intensity increases to ρ =2.05, the performance of the proposed Sugeno scheduler deteriorates and becomes close to the FIFO scheduler. The proposed Mamdani scheduler maintains a steady delay of 1.428s all through the simulation duration, which is better than the delay at ρ =1.55. The Manoj scheduler shows the worst performance as compared to the other schedulers at this traffic load.

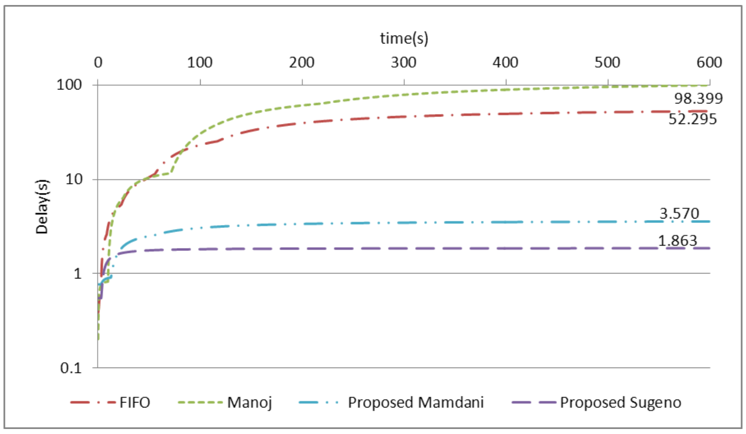


Figure 5-20: Mobile Nodes: Delay for Video Traffic at ρ =1.55 with CBR Background Traffic

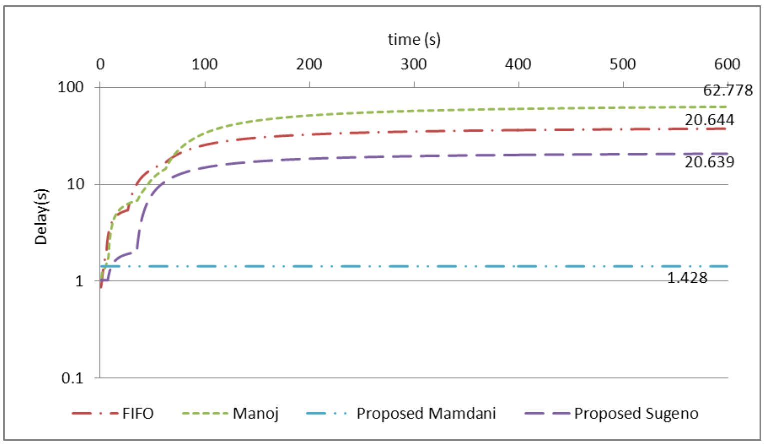


Figure 5-21: Mobile Nodes: Delay for Video Traffic at ρ =2.05 with CBR Background Traffic

The end-to-end delay for the background traffic is shown in Table 5-7. The delay for FIFO, the proposed Mamdani and Sugeno scheduler increases as the network load increases. The Manoj scheduler shows a high increase at ρ =0.60 for background traffic, this was also experienced by the video traffic as shown in Table 5-6. The proposed Mamdani and Sugeno scheduler performs better than the FIFO and Manoj scheduler for all values of ρ .

Table 5-7: Mobile Nodes: Average Delay for CBR Background Traffic

ρ	FIFO (s)	Manoj (s)	Proposed Mamdani (s)	Proposed Sugeno (s)
0.45	0.990	1.005	0.902	0.909
0.6	2.975	384.043	2.153	2.197
0.75	5.400	5.639	4.018	4.052
0.90	8.626	1,771.845	7.284	7.466
1.05	9,453.001	2,461.479	848.585	1,471.257
1.55	2078.020	2363.953	2939.93	1489.299
2.05	1500.576	3508.809	1154.506	2446.599

The PDR for the video traffic is shown in Table 5-8. The proposed Mamdani and Sugeno show better performance than the FIFO and Manoj. All video traffic sent by the source for the proposed Mamdani and Sugeno is received at the destination for ρ <1, the PDR decreases slightly to 0.986 and 0.977 for both proposed schedulers (Mamdani and Sugeno) at ρ =1.05 respectively. The FIFO scheduler shows a high PDR for ρ <1 but decreases to 0.627 at ρ =1.05 because of network congestion.

Table 5-8: Mobile Nodes: Average PDR for Video Traffic with CBR Background Traffic

ρ	FIFO	Manoj	Proposed Mamdani	Proposed Sugeno
0.33	1.000	1.000	1.000	1.000
0.45	0.999	0.999	1.000	1.000
0.6	0.998	0.812	1.000	1.000
0.75	0.998	0.998	1.000	1.000
0.90	0.998	0.607	1.000	1.000
1.05	0.627	0.544	0.977	0.986
1.55	0.245	0.329	0.596	0.431
2.05	0.075	0.235	0.076	0.236

The PDR at ρ =1.55 and 2.05 is shown in Figure 5-22 and Figure 5-23 respectively. The PDR decreases as the traffic intensity increases. As shown in Figure 5-22 the proposed schedulers perform better than the FIFO and Manoj. The FIFO scheduler slightly outperforms the proposed Sugeno scheduler at simulation time 120s onwards. The FIFO and the proposed Sugeno perform better than the proposed Mamdani. The high packet loss at both traffic loads is because of the high level of network congestion.

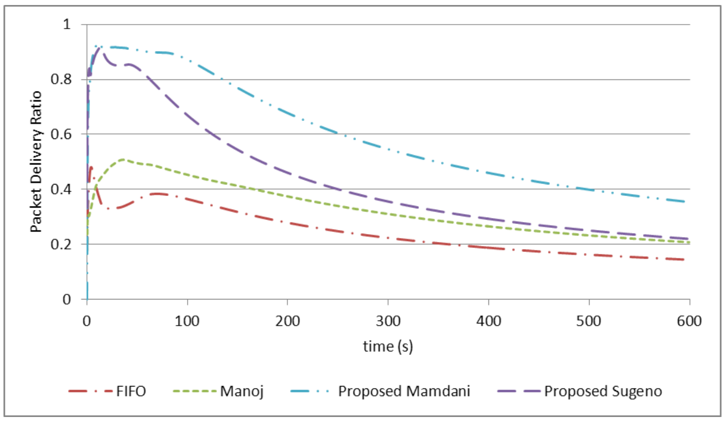


Figure 5-22: Mobile Nodes: PDR for Video Traffic at ρ =1.55 with CBR background Traffic

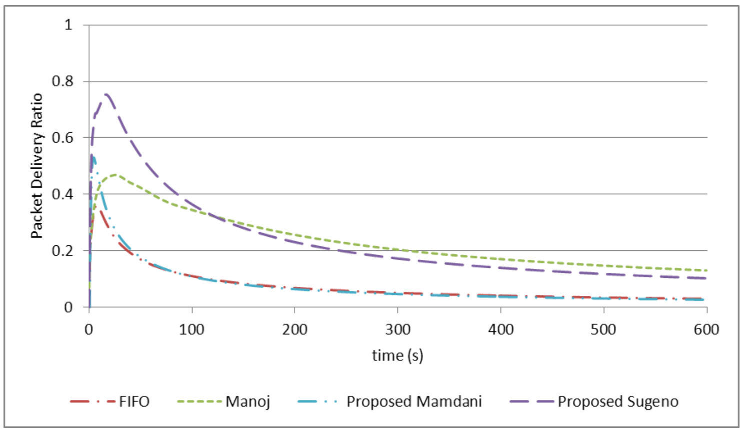


Figure 5-23: Mobile Nodes: PDR for Video Traffic at ρ =2.05 with CBR background Traffic

The PDR for the background traffic at ρ = 0.45-1.05 is shown in Table 5-9. The proposed Mamdani and Sugeno scheduler received 100% of the background traffic sent at ρ <1, this then decreases to 91% and 90% respectively at ρ =1.05. The FIFO scheduler also shows a high PDR for ρ <1 and then decreases to 0.694 at ρ =1.05. The Manoj scheduler shows the worst performance; the results are inconsistent as compared to both proposed and FIFO schedulers.

Table 5-9: Mobile Nodes: Average PDR for CBR Background Traffic

ρ	FIFO	Manoj	Proposed Mamdani	Proposed Sugeno
0.45	1.000	1.000	1.000	1.000
0.60	0.999	0.817	1.000	1.000
0.75	0.999	0.999	0.999	0.999
0.90	0.999	0.722	0.999	0.999
1.05	0.694	0.705	0.911	0.901
1.55	0.408	0.572	0.612	0.348
2.05	0.188	0.448	0.102	0.284

5.4.3.3. Video stream with VBR Background Traffic

The robustness of the schedulers is further analysed by introducing VBR background traffic under varying traffic intensities. The performance analysis of the video stream with VBR background traffic for stationary nodes and mobile nodes is given below.

A. Stationary nodes

The traffic intensity is increased to ρ =0.45 of which 26% consist of background traffic. The end-to-end delay for the video traffic is shown in Figure 5-24. The FIFO scheduler increases to 0.693s whilst the Manoj scheduler increases to 0.966s. This delay variation is due to the nature of the background traffic introduced. The average end-to-end delay according to Figure 5-24 for the FIFO scheduler is 0.535s, and Manoj is 0.776s, this is higher than the baseline at 0.068s shown in Figure 5-6. The average end-to-end delay for the proposed Mamdani and Sugeno scheduler is 0.076s. The proposed schedulers show a better performance than the FIFO and Manoj scheduler did, and their end-to-end delay are closer to the baseline delay (0.068s) in Figure 5-6. The average end-to-end delay for the proposed

Mamdani and Sugeno schedulers show a percentage improvement of 895 and 586% as compared to the FIFO and Manoj scheduler respectively.

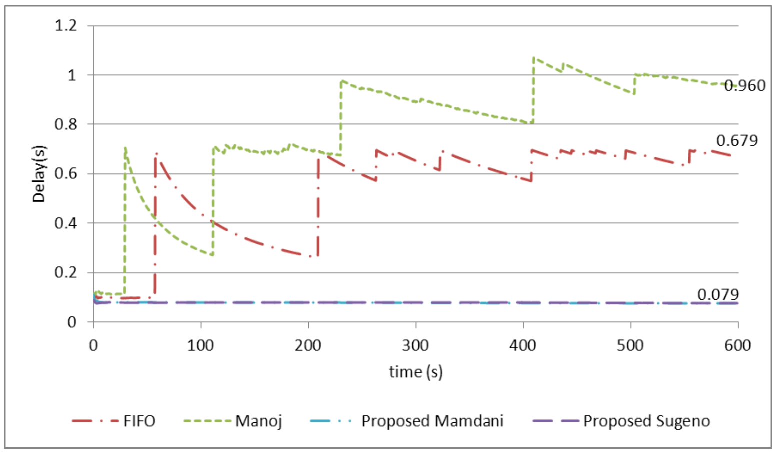


Figure 5-24: Stationary Nodes: Delay for Video traffic at ρ =0.45 with VBR background Traffic

The end-to-end delay for the background traffic is shown in Figure 5-25. The performance of the FIFO, Manoj, proposed Mamdani and Sugeno are close to the simulation time 121s after which Manoj scheduler delay increases. The FIFO, proposed Mamdani and Sugeno scheduler show similar performance after 121s simulation time until 340s where the proposed Mamdani gradually increases. The average end-to-end delay for the FIFO, Manoj, proposed Mamdani and Sugeno scheduler is 0.981, 1.276, 0.993 and 0.966s, respectively. The average end-to-end delay of the background traffic for the proposed Mamdani and Sugeno is close to the FIFO scheduler. However they perform better than the Manoj scheduler did. The proposed schedulers perform better as it maintains the baseline delay in Figure 5-6 for the video stream without adversely affecting the performance of the background traffic.

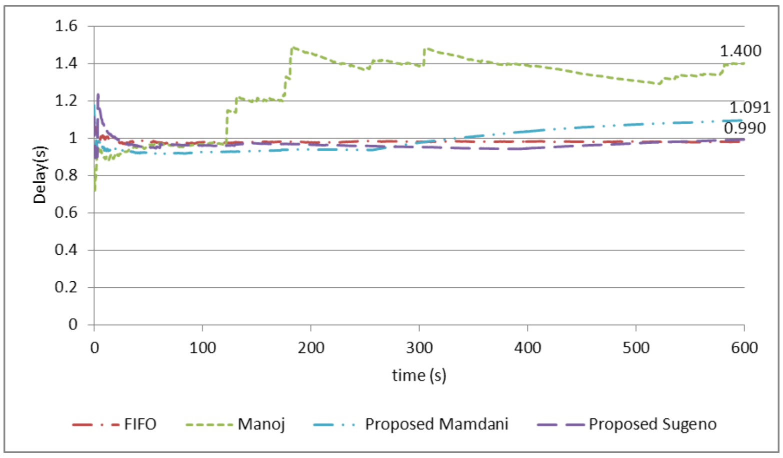


Figure 5-25: Stationary Nodes: Delay for VBR Background Traffic at ρ =0.45

The total traffic sent (video and background) through the network at ρ =0.45 is less than the queuing capacity thus all traffic sent by the source is received at the destination. The PDR for the video traffic at ρ =0.45 is shown in Figure 5-26. According to the figure, all the video and background traffic sent by the source are received at the destination.

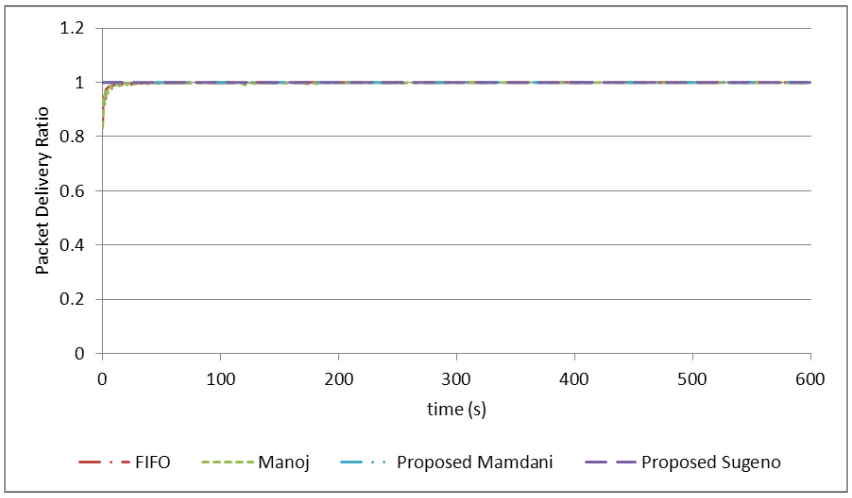


Figure 5-26: Stationary Nodes: PDR for Video Traffic at ρ =0.45 with VBR background Traffic

The average end-to-end delay for various ρ is summarised in Table 5-10. The background traffic make up 44% of the total traffic sent for the traffic intensity ρ =0.60. At this network load, the end-to-end delay of the proposed Mamdani and Sugeno maintained the baseline delay in Figure 5-6 for the video traffic. However, the delay for the FIFO and Manoj scheduler degrades as compared to baseline. The average end-to-end delay for the FIFO and Manoj schedulers at ρ =0.60 is 0.136 and 0.969s, respectively. Overall, the Manoj scheduler performs worse than the FIFO scheduler did. The percentage improvement of the proposed Mamdani and Sugeno scheduler with respect to the FIFO and Manoj is 1,325 and 100% respectively. The end-to-end delay for the background traffic is shown in Table 5-11. The performance of the proposed Mamdani and Sugeno along with the FIFO scheduler at ρ = 0.45 is close. Manoj shows the worst performance with an average end-to-end delay of 1.276s.

Table 5-10: Stationary Nodes: Average Delay for Video Traffic with VBR Background Traffic

ρ	FIFO (s)	Manoj (s)	Proposed Mamdani (s)	Proposed Sugeno (s)
0.33	0.068	0.068	0.068	0.068
0.45	0.535	0.776	0.076	0.076
0.6	0.136	0.969	0.068	0.068
0.75	0.224	41.039	0.068	0.068
0.90	0.456	52.612	0.087	0.076
1.05	8.978	65.290	0.157	0.217
1.55	13.026	109.560	0.218	0.688
2.05	21.516	114.856	10.009	20.053

Table 5-11: Stationary Nodes: Average Delay for VBR Background Traffic

ρ	FIFO (s)	Manoj (s)	Proposed Mamdani (s)	Proposed Sugeno (s)
0.45	0.981	1.276	0.993	0.966
0.6	3.002	152.930	1.946	1.889
0.75	5.634	768.326	6.256	3.967
0.90	9.733	753.346	708.561	33.025
1.05	1,527.843	1,179.640	1,976.869	2,431.538
1.55	1,973.203	962.818	1,482.378	2,740.947
2.05	5,199.538	1,442.211	2,705.961	4,908.652

The traffic intensity is increased to ρ =0.75 as shown in Table 5-10. This implies the background traffic is increased to 55.56% of the total traffic. The average end-to-end delay for the video traffic for the proposed Mamdani and Sugeno is 0.068s; this is the same as the baseline in Figure 5-6. The average end-to-end delay for the FIFO and Manoj is 0.224 and 41.039s respectively. This is a 229 and 60,194% increase in end-to-end delay for the FIFO and Manoj scheduler respectively as compared to the baseline in Figure 5-6.

The average end-to-end delay for the video traffic at ρ =0.90 is shown in Table 5-10. The network is tending towards congestion at this point. The delay for the proposed Mamdani and Sugeno is 0.087 and 0.076s respectively. This is approximately 22 and 11% increase in the baseline shown in Figure 5-6, for the proposed Mamdani and Sugeno scheduler respectively. The FIFO and Manoj scheduler performs worse as compared to the proposed schedulers. Maintaining the baseline delay for a real-time application becomes more challenging as the network tends towards congestion. The average end-to-end delay for the background traffic for the proposed Mamdani and Sugeno becomes higher than the FIFO as shown in Table 5-11; however, the proposed schedulers performed better than the Manoj scheduler did. At higher ρ , the performance of the scheduler for the background traffic degrades, this is a cost effect of prioritising the real-time video stream.

The traffic intensity is then increased to ρ =1.05, at this point, the network is congested as the traffic intensity satisfies the condition of congestion $\rho \geq 1$. The average end-to-end delay for video traffic at all levels of ρ is shown in Table 5-10. The performance of the FIFO and Manoj scheduler significantly degrades because of congestion; however, Manoj scheduler shows the worst performance. The end-to-end delay for the proposed Mamdani and Sugeno scheduler is 0.157 and 0.217s respectively, this shows a 131 and 219% increase from the baseline shown in Figure 5-6, thus outperforming the FIFO and Manoj scheduler. The average end-to-end delay for the background traffic tends towards infinity; the FIFO scheduler shows a better performance, which is a cost effect for maintaining the baseline delay for the time sensitive real-time video traffic at high background load.

The traffic intensity is further increased to ρ =1.55 and 2.05 to analyse the network performance. The end-to-end delay for video traffic at ρ =1.55 and 2.05 is shown in Figure

5-27 and Figure 5-28 respectively. The end-to-end delay for the proposed schedulers performs better than the FIFO and Manoj scheduler for ρ =1.55. The proposed Mamdani scheduler shows the best performance in comparison to all the other schedulers. At ρ =2.05, the performance of the proposed Sugeno scheduler becomes similar to FIFO at simulation time of 150s. The proposed Mamdani scheduler shows the best performance as compared to the other schedulers but overall the performance of the proposed schedulers degrades significantly. The performance of the proposed schedulers degraded because of congestion as the network load is more than double the network capacity.

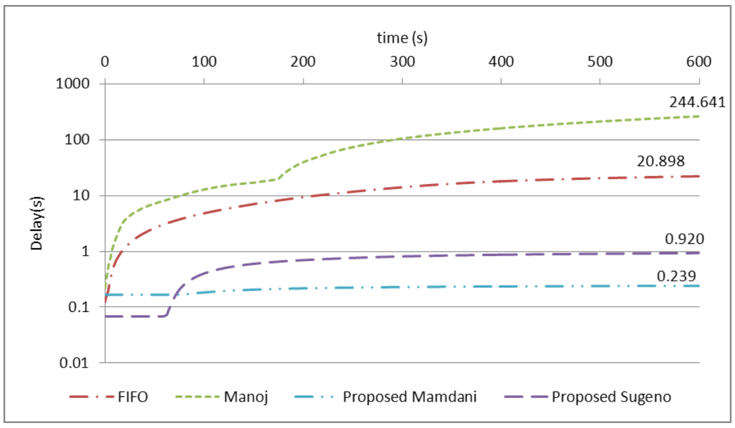


Figure 5-27: Stationary Nodes: Delay for Video Traffic at ρ =1.55 with VBR Background Traffic

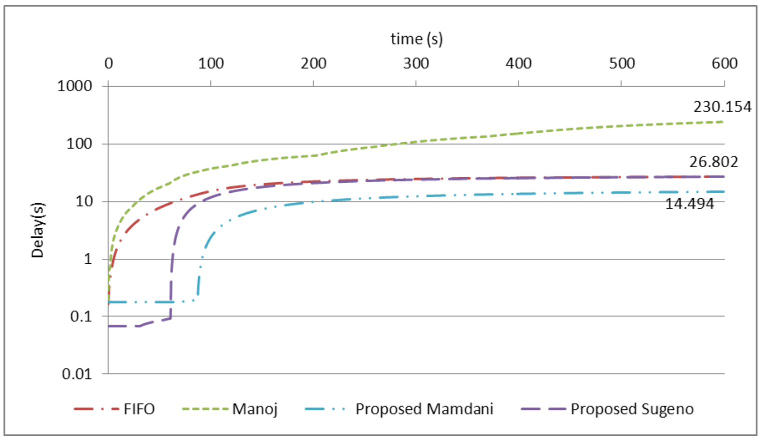


Figure 5-28: Stationary Nodes: Delay for Video Traffic at ρ =2.05 with VBR Background Traffic

The PDR of video traffic for various ρ values is shown in Table 5-12. The proposed Mamdani and Sugeno scheduler maintained a PDR of 1 for ρ =0.33, 0.45. 0.6 and 0.75, i.e., all video frames sent by the source were received at the destination. The PDR for the proposed Mamdani dropped to 0.998 at ρ = 0.90, whilst the proposed Sugeno maintained a PDR of 1. The PDR dropped to 0.883 for the proposed Mamdani scheduler at ρ =1.05, whilst the proposed Sugeno maintained a maximum PDR of 1. The PDR for Manoj scheduler, as shown in Table 5-12, decreases as the network load increases. The FIFO scheduler maintains a high PDR for the entire network load variation.

Table 5-12: Stationary Nodes: Average PDR for Video Traffic with VBR Background Traffic

ρ	FIFO	Manoj	Proposed Mamdani	Proposed Sugeno
0.33	1.000	1.000	1.000	1.000
0.45	0.999	0.998	1.000	1.000
0.6	0.998	0.829	1.000	1.000
0.75	0.998	0.619	1.000	1.000
0.90	0.997	0.488	0.998	1.000
1.05	0.938	0.399	0.883	1.000
1.55	0.894	0.423	0.738	0.825
2.05	0.4014	0.193	0.433	0.341

The PDR for the video traffic at ρ =1.55 and 2.05 is shown in Figure 5-29 and Figure 5-30 respectively. According to Figure 5-29, the PDR at ρ =1.55 for the proposed scheduler (Mamdani and Sugeno) starts to drop after 200 and 180s of simulation time respectively. This is because of the high level of congestion in the network; the scheduler can assign lower Priority Index to the background traffic, which can potentially affect the service rate of the video stream packets in the queue. However, the PDR for the FIFO scheduler shows a better and stable performance at this network load. The Manoj scheduler shows the worst performance when compared to the other schedulers. When the network load is more than double the network capacity at ρ =2.05 as shown in Figure 5-30, the PDR sharply drops. The proposed Mamdani and Sugeno schedulers show a better performance than the FIFO and Manoj between simulation time 0-100 and 0-80s respectively, between this short simulation time span of 0-100s and 0-80s for the proposed Mamdani and Sugeno scheduler respectively, the destination receives all video traffic sent by the source. The performance of the video streams drastically degrades afterwards; thus, the queue is highly congested.

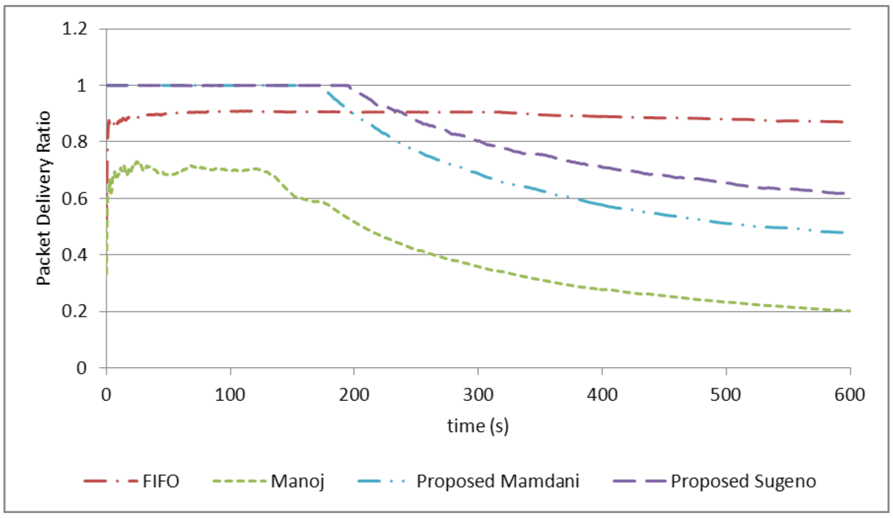


Figure 5-29: Stationary Nodes: PDR for Video Traffic at ρ =1.55 with VBR Background Traffic

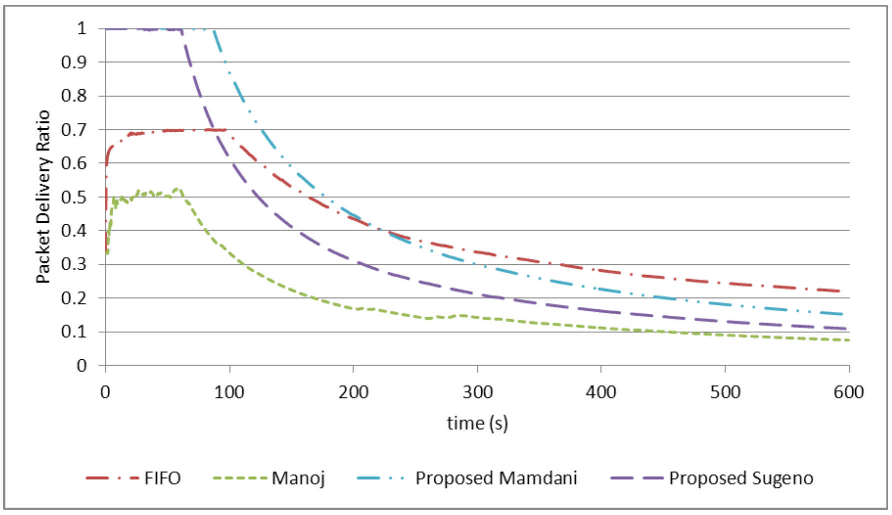


Figure 5-30: Stationary Nodes: PDR for Video Traffic at ρ =2.05 with VBR Background Traffic

The PDR of the background traffic for various level of ρ is shown in Table 5-13, which shows similar behaviour to the video traffic. The proposed Sugeno scheduler maintains a high PDR when ρ <1, however the performance degraded when ρ >1. Maintaining a required QoS for the video traffic on a congested network with mixed traffic is challenging. The performance of the proposed Mamdani scheduler starts to drop at ρ = 0.90, whilst the performance of Manoj scheduler degrades as the network load increases without necessarily improving the video traffic.

Table 5-13: Stationary Nodes: Average PDR for VBR Background Traffic

ρ	FIFO	Manoj	Proposed Mamdani	Proposed Sugeno
0.45	0.950	0.939	0.937	0.953
0.6	0.961	0.776	0.966	0.958
0.75	0.962	0.683	0.959	0.958
0.90	0.968	0.594	0.806	0.971
1.05	0.950	0.539	0.700	0.686
1.55	0.811	0.438	0.605	0.637
2.05	0.572	0.223	0.492	0.487

B. Mobile Nodes

The end-to-end delay, when the traffic intensity is =0.45, is shown in Figure 5-31. The proposed Mamdani and Sugeno scheduler maintains a delay of 0.078 that is approximately 15% higher than the baseline shown in Figure 5-9 because of the added complexity of node movement. The end-to-end delay for the FIFO and Manoj schedulers increase sharply after simulation time of 32s, and it continued to increase further afterwards. The average end-to-end delay for FIFO and Manoj scheduler at ρ =0.45 is 0.603 and 0.845s respectively. This is approximately 787% and 1143% higher than the baseline in Figure 5-9. This is very high as compared to the proposed schedulers, which show only an increase of 15% from the baseline.

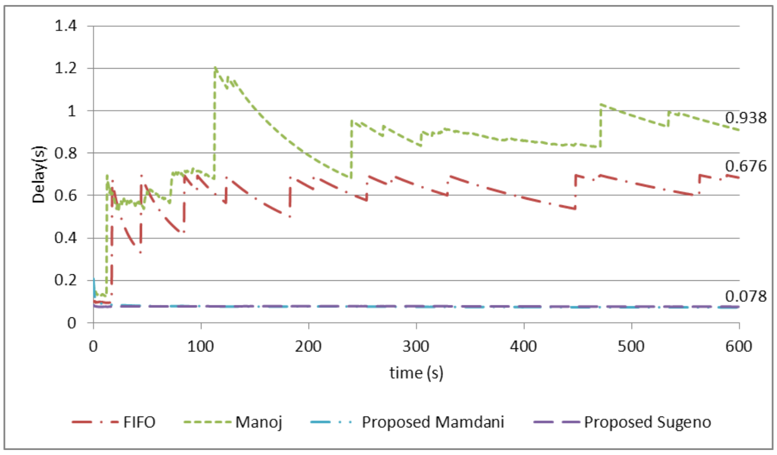


Figure 5-31: Mobile Nodes: Delay for Video Traffic at ρ =0.45 with VBR Background Traffic

The end-to-end delay for the background traffic at ρ =0.45 is shown in Figure 5-32. The performance of the FIFO scheduler along with the proposed Mamdani and Sugeno scheduler are close as shown in Figure 5-32. However, the end-to-end delay for the Manoj scheduler starts to increase after 33s; it shows a higher end-to-end delay as compared to the FIFO, Mamdani and Sugeno scheduler. The average end-to-end delay for the background traffic for FIFO, Manoj, proposed Mamdani and Sugeno at ρ =0.45 are 1, 1.742, 1.095 and 0.974s respectively. The proposed scheduler maintained lower background traffic PDR even though preferences are given to the video stream.

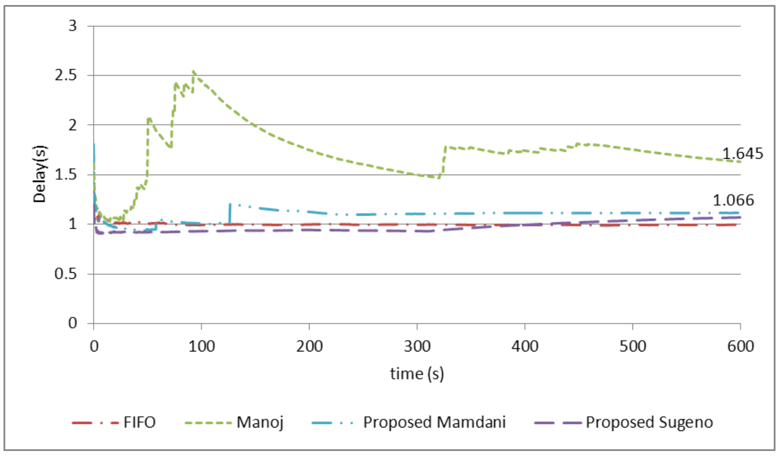


Figure 5-32: Mobile Nodes: Delay for VBR Background Traffic at ρ =0.45

The PDR for the video traffic at ρ =0.45 is shown in Figure 5-33. The PDR for all the schedulers is the same for most of the simulation time. This signifies a very low packet loss for the video traffic at ρ =0.45. All video traffics sent by the source were received at the destination.

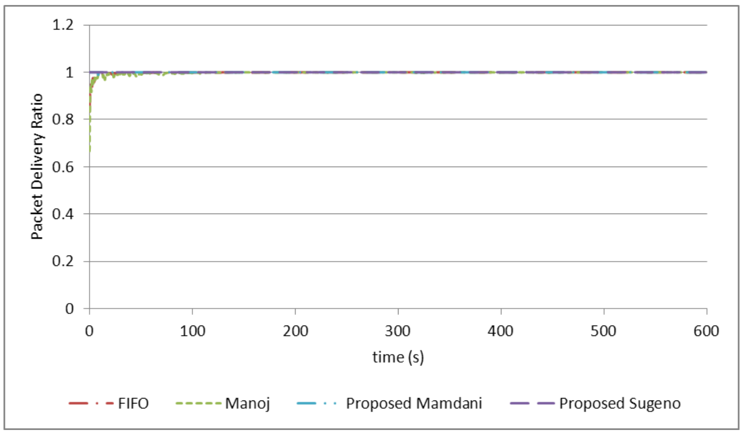


Figure 5-33: Mobile Nodes: PDR for Video Traffic at ρ =0.45 with VBR Background Traffic

The average end-to-end delay for VBR video traffic for various ρ is summarised in Table 5-14. According to Table 5-14, the proposed Mamdani and Sugeno scheduler performs better than the FIFO and Manoj. The proposed schedulers maintain the same delay as the baseline in Figure 5-7 at ρ =0.60. When ρ =0.90, the network is tending towards congestion thus the proposed Sugeno scheduler average end-to-end delay is 0.225s whilst that for the proposed Mamdani is 0.103s. This is about 231 and 51% higher than the baseline for the proposed Mamdani and Sugeno respectively. The performance of the FIFO and Manoj scheduler also degrades.

The traffic intensity for the VBR traffic is increased to ρ =1.55 and 2.05, the end-to-end delay for the video traffic at both value of ρ is shown in Figure 5-34 and Figure 5-35 respectively. The performance of both proposed schedulers is similar as shown in Figure 5-34; they both perform better than the FIFO, and Manoj scheduler did. At ρ =1.55, the proposed Mamdani and Sugeno schedulers perform 1,155% better than FIFO scheduler and 2,997% better than Manoj scheduler. The proposed schedulers also perform better than FIFO and Manoj schedulers at ρ =2.05 as shown in Figure 5-35. The proposed Sugeno scheduler shows the best performance as compared to the three other schedulers.

Table 5-14: Mobile Nodes: Average Delay for Video Traffic with VBR Background Traffic

ρ	FIFO (s)	Manoj (s)	Proposed Mamdani (s)	Proposed Sugeno (s)
0.33	0.079	0.082	0.068	0.068
0.45	0.603	0.845	0.077	0.078
0.6	0.137	0.175	0.068	0.068
0.75	0.220	0.298	0.070	0.072
0.90	0.456	49.413	0.103	0.225
1.05	8.292	57.259	1.604	0.578
1.55	32.077	79.117	2.481	2.555
2.05	33.465	110.789	4.330	1.275

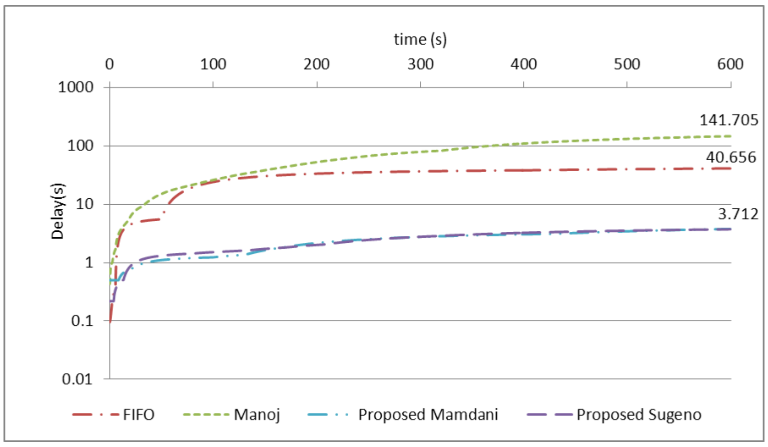


Figure 5-34: Mobile Nodes: Delay for Video Traffic at ρ =1.55 with VBR Background Traffic

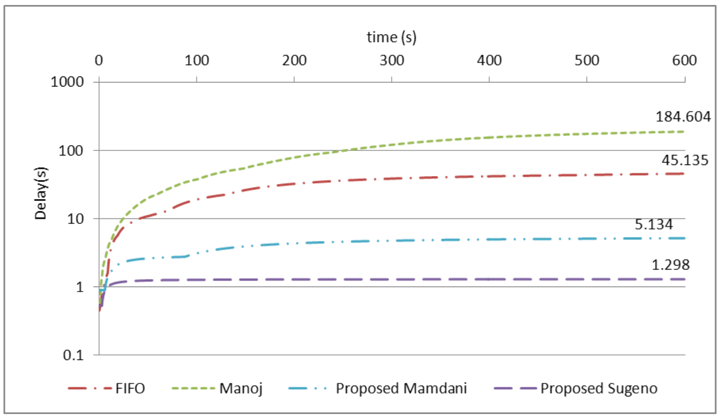


Figure 5-35: Mobile Nodes: Delay for Video Traffic at ρ =2.05 with VBR Background Traffic

The average end-to-end delay for the background traffic for all various ρ is shown in Table 5-15. The proposed Mamdani and Sugeno scheduler performs better as compared to the FIFO and Manoj scheduler for ρ = 0.45, 0.6 and 0.75. The background traffic end-to-end delay increased at ρ =0.90 because the network is tending towards congestion at this point. The proposed Sugeno scheduler shows the best performance at ρ =1.05. Overall the proposed schedulers perform better than the FIFO and Manoj scheduler because the real-time traffic were close to the baseline, and the background traffic has not suffered as much until ρ =0.90 and 1.05.

Table 5-15: Mobile Nodes: Average Delay for VBR Background Traffic

ρ	FIFO (s)	Manoj (s)	Proposed Mamdani (s)	Proposed Sugeno (s)
0.45	1.000	1.742	1.095	0.974
0.6	3.026	3.207	2.124	1.832
0.75	5.652	6.851	4.846	4.598
0.90	9.980	738.898	90.164	803.398
1.05	941.643	1,168.202	2,044.508	63.771
1.55	1,211.793	2,708.476	2,383.617	1,706.897
2.05	2,913.417	2,155.373	3,816.322	1,760.053

The PDR for the video traffic for all ρ values is shown in Table 5-16. The proposed Sugeno scheduler maintained a PDR of 1 at ρ =0.33 to 1.05. The proposed Mamdani scheduler shows a high PDR for all values of ρ . The performance of the proposed Mamdani degrades at ρ =0.90 and 1.05 as congestion occurred. However, this is by a small margin as compared to FIFO and Manoj scheduler. The PDR for Manoj scheduler, as shown in Table 5-16, decreases as the network load increases. The FIFO scheduler maintains a high PDR at ρ =0.33 to 0.90 after which the performance degrades to 0.719 at ρ =1.05.

Table 5-16: Mobile Nodes: Average PDR for Video Traffic with VBR Background Traffic

ρ	FIFO	Manoj	Proposed Mamdani	Proposed Sugeno
0.33	1.000	1.000	1.000	1.000
0.45	0.999	0.998	1.000	1.000
0.6	0.998	0.998	1.000	1.000
0.75	0.998	0.997	1.000	1.000
0.90	0.997	0.440	0.998	1.000
1.05	0.716	0.311	0.873	1.000
1.55	0.579	0.509	0.903	0.924
2.05	0.217	0.199	0.372	0.110

The PDR for ρ =1.55 and 2.05 is also shown in Figure 5-36 and Figure 5-37 respectively. The proposed schedulers show a significantly better performance than Manoj and the FIFO scheduler according to ρ =1.55. The performance for Manoj and FIFO are similar. The proposed scheduler coped well for the real time traffic at this network load. However, the performance of all the schedulers degrades sharply at ρ =2.05 as shown in Figure 5-37. The network load is more than twice the network capacity. Thus the network is highly congested, and the schedulers cannot cope with this level of network load. Thus the high packet drops after the first few seconds of simulation time.

The PDR for the background traffic is shown Table 5-17. It shows similar behaviour to the video traffic. The proposed Sugeno scheduler shows high PDR for ρ = 1 to 1.05. The performance of the proposed Mamdani scheduler drops at ρ = 1.05 because of congestion. The proposed Sugeno performs better because the background traffic is not adversely affected by the priorities real-time video traffic. The performance of Manoj scheduler degraded as the network load increases. The FIFO scheduler slightly degrades when ρ = 1.05.

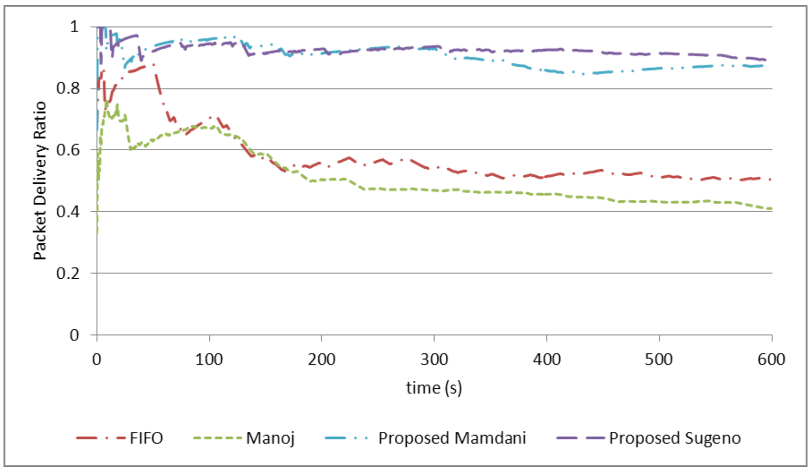


Figure 5-36: Mobile Nodes: PDR for Video Traffic at ρ =1.55 with VBR background Traffic

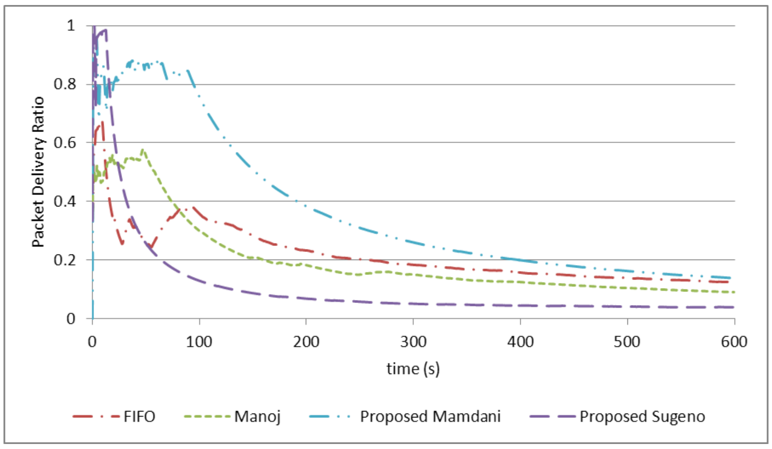


Figure 5-37: Mobile Nodes: PDR for Video Traffic at ρ =2.05 with VBR background Traffic

Table 5-17: Mobile Nodes: Average PDR for VBR Background Traffic

ρ	FIFO	Manoj	Proposed Mamdani	Proposed Sugeno
0.45	0.952	0.944	0.941	0.936
0.60	0.958	0.949	0.974	0.965
0.75	0.957	0.954	0.961	0.953
0.90	0.972	0.563	0.965	0.943
1.05	0.723	0.473	0.749	0.999
1.55	0.525	0.520	0.755	0.779
2.05	0.042	0.251	0.495	0.235

5.5. Summary

This chapter presents the methodology for streaming a real-time video through OPNET. The OpenCV library was used in capturing and displaying the video stream. The captured video frame was encapsulated in a new packet format supported by OPNET before transmitted through the network. The performance of the video stream without background traffic over the network was analysed, and this formed the baseline. The CBR and VBR background traffic was gradually introduced, and the performance of the video stream was analysed when the schedulers are applied.

Both proposed schedulers assigned the highest Priority Index to the video streams whilst the Priority Index of the background traffic was calculated as explained in Chapter 4. The proposed Schedulers showed a better and consistent performance for the video traffic with CBR and VBR background traffic. The performance of the Manoj scheduler was very poor for all ρ as compared to the FIFO and both proposed schedulers. However, it performed better than the FIFO when a single traffic flow was considered in Chapter 4. The performance of Manoj scheduler for CBR and VBR is worse and inconsistent than the FIFO and the proposed schedulers for most values of ρ. The proposed schedulers maintained a delay close to the baseline as the network load increases. The performance of the proposed schedulers deteriorated as the network load increased to ρ =1.55 and 2.05. However, the proposed schedulers maintained lower margin of error to the baseline than the FIFO and

Manoj schedulers at ρ =0.33-1.05, the result is less congested as compared to Manoj and FIFO.

The maximum packet size, an OPNET network can transmit, is 18,432bits. Thus, the packets were fragmented before sent through the network and then re-assembled at the destination. After the first fragmented packet is received, there is a time constraint for how long it can wait for the other parts to arrive before it can be re-assembled. Packets will be discarded if not all its fragments are received at the receiver in a timely manner. This contributed to the poor performance of the Manoj scheduler. When the network was without background traffics all the video frames sent by the source were received at the destination in a timely manner when Manoj scheduler was applied. The introduction of background traffic created additional complexity for the Manoj scheduler because a video frame/packet is fragmented into smaller packets sizes and this fragmented packets are likely to be assigned varying Priority Index based on the input variables to the fuzzy scheduler. Thus, these fragmented packets are spread out around the three sub-queues along with the incoming background traffics as the Manoj scheduler cannot distinguish between the video and background traffic in a queue. Hence, when the traffic intensity of the background traffic increases, it becomes more difficult to receive a complete sequence of packet fragments at the receiver in a timely manner. This will cause an increase in the end-to-end delay and packet loss of the video frames. Dropping of packets in the network will result in a large number of re-transmissions, which will result in huge overhead for the Manoj scheduler, and thus more packets dropped and higher end-to-end delay.

The proposed schedulers assign the highest Priority Index to the video traffic, and they are served in the queue in a FIFO manner, whilst the fuzzy inputs determine the Priority Index of the background traffic. The FIFO scheduler sends the fragmented video packet together with the background traffic in a FIFO manner, thus reassembling these packets at the receiver are made easier because the complete fragmented sequence arrives at the receiver sequentially. The proposed schedulers (Mamdani and Sugeno) are more robust to deal with multiple traffic flows better than the FIFO and Manoj schedulers, because they are dynamic and can adapt to the network characteristic changes in real time.

At higher levels of ρ, the queue gets full. Thus, all arriving packets either real or background traffic, are discarded. This will create a sequence of re-transmission, which will increase the network overhead and affect the delay and packet loss of the video and background traffic. Future work can include discarding some background traffic arriving the queue when the queue length is above a certain threshold to reduce the chances of congestion occurring and reduce the number of dropped video packets at the queue.

In the light of the evidence provided in Chapter 4 and 5, the proposed schedulers performed better than the FIFO and Manoj. The proposed Sugeno scheduler has shown a more consistent performance than the proposed Mamdani and no significant performance degradation as compared to the proposed Mamdani. Thus, the simplicity of the design of the proposed Sugeno scheduler and it being less computationally complex than the proposed Mamdani scheduler has contributed to its suitability for real-time application.

Chapter 6 Conclusion

Providing QoS for MANET is challenging because of the dynamic nature of the network. This thesis focused on the definition, design, and evaluation of a QoS aware solution in a dynamic environment for MANET. Two adaptive fuzzy based schedulers were proposed, and a detailed performance analysis and evaluation was carried out.

This section summarises the research challenges in MANET, the need for a realistic MANET simulation, the need for an adaptive scheduler in MANET and the implementation of a network interface between OPNET and an external device to validate the proposed schedulers.

6.1. Summary of research challenges

The mobile nature of the nodes in MANET causes a frequent topology changes which results to frequent link connections and disconnections. These issues have made the design and modelling of a MANET very complex. Some of the other research challenges in MANET include limited network capacity and the lack of mobility awareness by the nodes. These challenges can result in high network delay and low PDR, thus affecting the stability of the system.

The choice of a routing protocol has a great effect on the performance of a network, especially at high network loads. Hence, based on the literature review the reactive routing protocol has been employed in order to improve the QoS of the MANET. However, this is not enough to provide the needed QoS for real-time application in MANET.

The recent demand to deploy MANET for real time applications has increased, hence the need for a better QoS is paramount. The dynamic nature of the mobile nodes and the limited network resources in MANET makes meeting this QoS demands challenging. These causes the build-up of packets in the queue, resulting in higher packet delay, decrease in throughput and packet loss, this affects the overall system performance.

6.2. Realistic wireless simulation model for MANET

This thesis presents detailed guidelines with the required parameters to create a realistic simulation model based on results from an open field outdoor experiment.

An open field experiment was carried out for two traffic profiles (CBR and VBR) under TCP and UDP transport protocol. Based on literature, shadowing propagation model was applied to the simulation model to replicate the performance of the open field experiment. The path loss exponent (β) and shadowing deviation (α) were varied until the values that best replicate the experiment performance were reached. The values are 2.02 and 6.5 for β and α respectively. The IEEE 802.11n wireless card parameters are configured, so each MANET node replicates the wireless card used in the experiment. The performance of the simulation model matches the experiment in terms of bitrate and received packets, with 99% accuracy. Thus, the modeling of a realistic simulation network for MANET was a success. However, the end-to-end delay could not be accurately measured in the experiment because of the time synchronization device. This can be improved by using a more accurate GPS clock. The UDP protocol is connectionless and it performs better in terms of delay and is mostly considered for delay-sensitive application. The TCP protocol showed a lower packet corruption and packet loss rate because it is connection-oriented. The TCP for the experiment received slightly higher packets than the UDP protocol.

6.3. Requirement for an Adaptive scheduling algorithm for MANET

The increasing popularity of MANET for real-time applications has increased the demand for a better QoS. Previous proposed MANET schedulers are not adaptive/dynamic enough to provide the required QoS. Hence, it is imperative to provide an alternative dynamic solution to improve or guarantee a specific QoS requirement. MANET nodes have a limited bandwidth, hence when packets are not properly scheduled, congestion will occur at high network loads. This will degrade the network delay and the PDR performance. This thesis proposed two adaptive priority based fuzzy schedulers that adapt to the topology changes in MANET and thus improve the network QoS. These proposed schedulers are based on an existing fuzzy scheduler that considered two inputs (data rate and channel capacity). The existing scheduler used the Mamdani FIS. The two proposed schedulers use the Mamdani and Sugeno FIS system respectively. The Sugeno is more compact, easy to implement and computationally efficient than the Mamdani. The major difference between both FIS systems is the output membership function. The output membership function for Mamdani is a fuzzy set whilst that for Sugeno can be either constant or linear as thus the interpretability and expressive power of the Mamdani FIS is lost in Sugeno because the consequent of the rules is not fuzzy.

The proposed schedulers based on three input variables determine the priority of individual packet rather than the entire flow. These three input variables are the data rate, queue size, and SNR; these inputs were considered because their behaviour reflects the state of a network. Each mobile node consists of three sub-queues and packets are enqueued and served based on their Priority Index. The packet priority index varies between 0 and 1.

The performance of the scheduler was analysed for three traffic profiles CBR, VBR and Bursty traffic, under congested network conditions. The measuring metrics, which formed the basis for performance evaluation, are end-to-end delay and PDR.

The proposed Mamdani and Sugeno scheduler performed better than the existing scheduler for CBR traffic; the end-to-end delay for Mamdani and Sugeno scheduler was reduced by an average of 52 and 54% respectively. The performances of the PDR are similar to the existing scheduler because of the characteristic of the CBR traffic; the network was also at full capacity. The proposed Mamdani and Sugeno schedulers showed a better performance for VBR traffic; the end-to-end delay was reduced by an average of 38 and 52%, respectively and the PDR improved by an average of 53 and 47% respectively. The proposed Mamdani and Sugeno schedulers also showed a better performance for Bursty traffic; the end-to-end delay was improved by 55 and 69% respectively, the PDR also improved by 55 and 76% respectively. The proposed scheduler showed the best performance with Bursty traffic, followed by VBR traffic, and then CBR traffic. Thus, the performance of the scheduler increases as the burstyness of the traffic increases.

The proposed schedulers adapt better to the topology changes of the network under congested conditions better than Manoj. The Mamdani scheduler is more computationally complex than the Sugeno scheduler, however it does not significantly outperforms the

Sugeno schedulers. For this reason, the Sugeno scheduler is more suitable for real-time applications.

6.4. Network interface between OPNET and external hardware device

Networks are playing an ever-increasing role in today's the world. The size of the network, as well as the network capacity, has continued to increase over the years and, as a result, it has become more complicated. Hence, it is challenging and expensivee to create in real life for test and validation purpose. The best test method is a multi-simulation approach, where the performance of applications to be tested can be modelled and transmitted through a simulation network. Whilst this may work well, it is still an approximation of the performance with certain margin of error as compared to real-life. Interfacing simulation model with hardware to be tested reduces this margin of errors as real traffic are transmitted through the network. An interface between OPNET and a webcam was developed. Real-time video stream captured from the external webcam device was sent through the network and can be viewed at the destination node. This was achieved by using OPENCV library; the captured video stream is encapsulated in an OPNET supported custom packet and sent through the network. This packet is de-capsulated at the destination and can be viewed. This provided a more realistic hardware – software simulation model to test and validate the considered applications as well as the proposed schedulers. It can also be used by researcher for video surveillance application for the military, search and rescue teams, environmental monitoring and medical surgery that requires live feed videos.

The performance of the proposed scheduler was analysed for the real-time video stream under increasing CBR and VBR background load. The proposed schedulers were compared to the FIFO and an existing scheduler (Manoj) for stationary, and mobile nodes, and the performance metric are delay and PDR.

The proposed scheduler assigned the highest priority Index to the video traffic and they are then serviced in a FIFO manner. The priority Index of the background traffic is determined by considering the fuzzy inputs data rate, SNR and queue size. The proposed scheduler showed better performances for stationary and mobile nodes when CBR and VBR background traffic was introduced into the network.

6.5. Future Work

Some directions for future work are given below.

- Consideration of the multi-hop network in section 3.3 and Table 3-5 and passing traffic through the ISO protocol stack for MANET under various network loads and comparing the results with the expected theoretical values. This will give a wider perspective of the effect of the OSI layer on the received traffics.
- The real-time experiment can be developed to include three (3) or more mobile nodes under congested and non-congested network conditions with different mobility profiles. This will give a more detailed wireless environment modelling for the simulation of various mobility profiles. A GPS time synchronisation device can be used, to accurately measure the end-to-end delay.
- Explore the viability and performance of other wireless devices such as Bluetooth and Zigbee as they are cheaper and easier to power. Most of the available gadgets such as mobile phones and tablets are equipped with Bluetooth device.
- Security is also a vital issue in wireless network, the investigation and implementation of an appropriate security protocol that is reliable and less computationally complex for MANET is needed.
- The use of an adaptive neuro-fuzzy model, this will automate the membership function and fuzzy rule design.

6.6. Epilogue

This thesis investigates MANET and it research challenges. It modelled and analysed real and non-real time traffic in a queue under controlled conditions. The behaviour of traffic in the queue was wholly dependent on its distributions. Traffic with a constant distributed packet size are easy for the network to predict as compared to exponentially or Pareto distributed traffic. This formed the baseline simulation model for this work. Packets with the Cauchy distributed sizes are also difficult for the network to predict. This thesis also developed a realistic model by carrying out a point-to-point open field experiment, using shadowing model in the simulation model to replicate the performance of the wireless channel. Results from this experiment were collected for both TCP and UDP protocols. The simulation model was fine-tuned using shadowing model to match the outdoor experiment in terms of packets delivered and throughput. However, the delay was difficult to measure, as the devices required an accurate time synchronisation device such as a GPS clock. An adaptive fuzzy scheduler was developed, this performed better when compared to the existing scheduler. The simulation network was limited to 20 nodes and the network load was varied, having more nodes will cause the hidden node problem thus degrading the performance of the network. An interface was implemented between OPNET and a Webcam to stream real-time video over the network, and the performance of the scheduler for the real-time traffic was analysed with the introduction of background traffic. The proposed scheduler showed a better performance as compared to the existing when the traffic intensity 'ρ' was between 0-1.05, as the traffic intensity reaches 1.55 the performance of the video stream drastically. This will be improved further by introducing a queuing threshold, when reached; non-real time traffic arriving the queue are discarded (pressed compressed traffic). This will create more room for arriving real-time traffic to be en-queued rather than because the queue is full. The thesis showed that applying an adaptive scheduler in MANET would improve its performance. Creating an efficient membership functions and fuzzy rule is also challenging, this however can be improve with the use of an adaptive neuro-fuzzy model.

References

[1] P. Naghshtabrizi and J. P. Hespanha, ‘Designing an observer-based controller for a network control system’, in 44th IEEE Conference on Decision and Control, European Control Conference. CDC-ECC '05 pp. 848–853, 2005.

[2] X. Liu and A. Goldsmith, ‘Wireless medium access control in networked control systems’, in *American Control Conference*, vol. 4, pp. 3605–3610 vol.4, 2004.

[3] N. S. Nafi and J. Y. Khan, ‘A VANET based Intelligent Road Traffic Signalling System’, in *Australasian Telecommunication Networks and Applications Conference (ATNAC '12),* pp. 1–6, Nov. 2012.

[4] L. Laffea, R. Monson, R. Han, R. Manning, A. Glasser, S. Oncley, J. Sun, S. Burns, S. Semmer, and J. Militzer, ‘Comprehensive monitoring of CO_2 sequestration in subalpine forest ecosystems and its relation to global warming’, in *Proceedings of the 4th international conference on Embedded networked sensor systems*, Boulder, Colorado, USA, pp. 423–424, 2006.

[5] I. Talzi, A. Hasler, S. Gruber, and C. Tschudin, ‘PermaSense: investigating permafrost with a WSN in the Swiss Alps’, in *Proceedings of the 4th workshop on Embedded networked sensors*, Cork, Ireland, 2007, pp. 8–12.

[6] G. Parkinson, D. Crutchley, P. M. Green, M. Antoniou, M. Boon, P. N. Green, P. R. Green, R. Sloan, and T. York, ‘Environmental monitoring in grain’, in *IEEE Instrumentation and Measurement Technology Conference (I2MTC' 10),* pp. 939–943, 2010.

[7] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon, ‘Health monitoring of civil infrastructures using wireless sensor networks’, in *Proceedings of the 6th international conference on Information processing in sensor networks*, Cambridge, Massachusetts, USA, pp. 254–263, 2007.

[8] M. Li and Y. Liu, ‘Underground structure monitoring with wireless sensor networks’, in *Proceedings of the 6th international conference on Information processing in sensor networks*, Cambridge, Massachusetts, USA, pp. 69–78, 2007.

[9] F. Gianfelici, ‘Measurement of quality of service (QoS) for peer-to-peer networks’, in *Proceedings of the IEEE International Conference on Virtual Environments, Human-Computer Interfaces and Measurement Systems. VECIMS '05.*, p. 6 pp, 2005.

[10] D. L. Goldsmith, B. Liebowitz, K. Park, S. Wang, B. Doshi, and J. Kantonides, ‘Precedence and Quality of Service (QoS) Handling in IP Packet Networks’, in *IEE Military Communications Conference. MILCOM '06*, pp. 1–6, 2006.

[11] P. Mannersalo, ‘Gaussian and Multifractal Process in teletraffic Theory’, PhD Thesis, Helsinki University of Technology, 2003.

[12] C. Grimm and G. Schluchtermann, *IP Theory and Performance*. Springer Series on Signal and Communication Technology, 2008.

[13] E. H. Qi, M. Meylemans, and M. Hattig, ‘Augmenting wireless LAN technology for Wi-Fi PAN’, in *Forty-Third Asilomar Conference on Signals, Systems and Computers*, pp. 321–324, 2009.

[14] A. Flatman, ‘Wireless LANs: developments in technology and standards’, *Computing & Control Engineering Journal*, vol. 5, no. 5, pp. 219–224, 1994.

[15] U.S. Robotics, ‘Wireless LAN Networking White Paper’, *Wireless LAN Networking White Paper*. [Online]. Available: http://www.usr.com/download/whitepapers/wireless-wp.pdf. [Accessed: 18-Aug-2010].

[16] D. Djenouri, A. Derhab, and N. Badache, ‘Ad Hoc Network Routing Protocols and Mobility’, in *The Ineternational Arab Journal of International Technology. vol. 3, No. 2*, 2006.

[17] S. Jian-mei and Z. Da-yong, ‘Application of the ZigBee Technology in the Wireless Ordering Dish System’, in *5th International Conference on Wireless Communications, Networking and Mobile Computing. WiCom '09*, pp. 1–4, 2009.

[18] A. C. Davies, 'An overview of Bluetooth Wireless TechnologyTM and some competing LAN standards', in *1st IEEE International Conference on Circuits and Systems for Communications, ICCSC '02,* pp. 206–211, 2002.

[19] P. Bhagwat, 'Bluetooth: technology for short-range wireless apps', *IEEE Internet Computing,* vol. 5, no. 3, pp. 96–103, 2001.

[20] V. Sarkimaki, R. Tiainen, T. Lindh, and J. Ahola, 'Applicability of ZigBee technology to electric motor rotor measurements', in *International Symposium on Power Electronics, Electrical Drives, Automation and Motion, SPEEDAM '06.*, pp. 137–141, 2006.

[21] Qingzhu Wang, Rongchang Liu, Yuquan Ma, Jinchuan Zhao, Lizhen Feng, and Shiguang Liu, 'Application study of mine alarm system based on ZigBee technology', in *IEEE International Conference on Automation and Logistics. ICAL* '08, pp. 2537–2540, 2008.

[22] L. Cao, W. Jiang, and Z. Zhang, 'Networked wireless meter reading system based on ZigBee technology', in *Chinese Control and Decision Conference, CCDC'08.,* pp. 3455–3460, 2008.

[23] T. Paul and T. Ogunfunmi, 'Wireless LAN Comes of Age: Understanding the IEEE 802.11n Amendment', in *IEEE Circuits and Systems Magazine*, no. First Quarter, 2008.

[24] 'Draft STANDARD for Information Technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications Amendment 5: Ehancements for higher throughput', 2009. [Online]. Available: http://standards.ieee.org/getieee802/download/802.11n-2009.pdf. [Accessed: 21-Feb-2012].

[25] 'White Paper IEEE 802.11n Overview WLAN Access Point.' Hirschmann, 2010.

[26] A. E. Conway, 'Performance modelling of multi-layered OSI communication architectures', in *IEEE International Conference on Communications, ICC '89,* pp. 651–657 vol.2, 1989.

[27] M. D. Pardue, ‘Fine-Tuning the OSI Model: Layer Functions and Services’, in *IEEE Military Communications Conference - Crisis Communications: The Promise and Reality, MILCOM '87,* vol. 1, pp. 0199–0203, 1987.

[28] D. Bertsekas and R. Gallager, *Data Networks,* 2nd ed. Prentice Hall, 1991.

[29] P. Simoneau, ‘The OSI Model: Understanding the Seven Layers of Computer Networks.’ Global Knowledge Training LLC, 2006.

[30] J. A. Freebersyser and B. Leiner, ‘A DoD perspective on mobile ad hoc networks, in: Charles E. Perkins (Ed.), Ad Hoc Networking’, *Addison Wesley Read. MA,* pp. 29–51, 2001.

[31] W. C. Fifer and F. J. Bruno, ‘The low-cost packet radio’, *Proceedings of the IEEE,* vol. 75, no. 1, pp. 33– 42, 1987.

[32] I. Chlamtac, M. Conti, and J. J.-N. Liu, ‘Mobile ad hoc networking: imperatives and challenges’, *Ad Hoc Networks*, vol. 1, no. 1, pp. 13–64, Jul. 2003.

[33] J. Strater and B. Wollman, ‘OSPF Modeling and Test Results and Recommendations, Mitre Technical Report 96W0000017’, Xerox Office Products Division, Mar-1996.

[34] Liting Cao, Wei Jiang, and Zhaoli Zhang, ‘Automatic Meter Reading System Based on Wireless Mesh Networks and SOPC Technology’ in *Second International Conference On Intelligent Networks and Intelligent Systems, ICINIS 09,* pp. 142–145, Nov. 2009.

[35] Q. Yong-jun, T. Qing-quan, W. Zhan-ying, Z. Hai-chun, Y. Dong-bing, X. Yan, B. Tao, and L. Fang, ‘Technologies on establishing earthquake site communication network based on wireless mesh network’, *International Conference On Information Systems for Crisis Response and Management*, ISCRAM '11, pp. 548–551, Nov. 2011.

[36] J. V. Capella, A. Bonastre, R. Ors, and J. J. Serrano, ‘Distributed and mobile systems based on wireless networks: definition of a generic control architecture’, in *IEEE International Conference on Industrial Technology, IEEE ICIT ’04,* vol. 2, pp. 830–835 Vol. 2, 2004.

[37] I. Stojmenovic, *Handbook of Wireless Networks and Mobile Computing*. Wiley-Blackwell, 2002.

[38] A. Shrestha and F. Tekiner, ‘On MANET Routing Protocols for Mobility and Scalability’, in *International Conference on Parallel and Distributed Computing, Applications and Technologies*, pp. 451–456, 2009.

[39] S. Han, G. Gu, and J. Ni, ‘A mobile ad hoc network with mobile satellite earth-stations’, in *Second International Multi-Symposiums on Computer and Computational Sciences. IMSCCS '07*. pp. 512–516, 2007.

[40] L. Kant, S. Demers, P. Gopalakrishnan, R. Chadha, L. LaVergne, and S. Newman, ‘Performance modeling and analysis of a mobile ad hoc network management system’, in *IEEE Military Communications Conference. MILCOM '05*, pp. 2816–2822 Vol. 5, 2005.

[41] L. Kaddar and A. Mehaoua, ‘ESTREL: Transmission and Reception Energy Saving Model for Wireless Ad Hoc Networks’, in *32nd IEEE Conference on Local Computer Networks, LCN '07*, pp. 784–785, 2007.

[42] A. Rai, S. Ale, S. S. Rizvi, and A. Riasat, ‘A new methodology for self localization in wireless sensor networks’, in *IEEE International Multitopic Conference, INMIC '08*, pp. 260–265, 2008.

[43] Z. ALfawaer, G. Hua, M. Y. Abdullah, and I. D. Mamady, ‘Power Minimization Algorithm in Wireless Ad-hoc Network Based on PSI’, *Journal of Applied Sciences*, vol. 7, no. 17, pp. 2523–2526, 2007.

[44] P. Gupta, P. Saxena, A. K. Ramani, and R. Mittal, ‘Optimized use of battery power in wireless Ad hoc networks’, in *The 12th International Conference on Advanced Communication Technology, ICACT '10*, vol. 2, pp. 1093–1097, 2010.

[45] M. Conti and S. Giordano, ‘Multihop Ad Hoc Networking: The Theory’, *IEEE Communications Magazine*, vol. 45, no. 4, pp. 88–95, 2007.

[46] M. Abolhasan, T. Wysocki, and E. Dutkiewicz, ‘A review of routing protocols for mobile ad hoc networks’, *Ad Hoc Network*, vol. 2, no. 1, pp. 1–22, Jan. 2004.

[47] A. Pal, J. P. Singh, P. Dutta, P. Basu, and D. Basu, ‘A study on the effect of traffic patterns on Routing protocols in Ad-hoc network following RPGM Mobility model’, in *International Conference on Signal Processing, Communication, Computing and Networking Technologies, ICSCCN '11*, pp. 233–237, 2011.

[48] S. H. Mohammad, ‘Application of Control over MANET’, PhD Thesis, Staffordshire University, Stafford, United Kingdom, 2009.

[49] C. A. Harding, ‘Development of a Delay Algorithm and a Co-Simulation Framework for NCS over MANETs’, PhD Thesis, Staffordshire University, Stafford, United Kingdom, 2009.

[50] C. E. Perkins, E. M. Royer, S. R. Das, and M. K. Marina, ‘Performance comparison of two on-demand routing protocols for ad hoc networks’, IEEE *Personal Communications. v*ol. 8, no. 1, pp. 16–28, 2001.

[51] E. Gelenbe and G. Pujolle, *Introduction to Queuing Networks*, Second Edition. Second Edition, John Wiley & Sons., 1998.

[52] B. A. Forouzan and S. C. Fegan, *Data communications and networking*. New York: McGraw-Hill Higher Education, 2007.

[53] B. A. Forouzan*, Data Communications Networking*, 4th ed. McGraw-Hill Higher Education, 2006.

[54] Zhenyi Chen and Xiaofan Wang, ‘Effects of Network Capacity under Variations of Network Structure and Routing Strategy’, *IEEE International Conference on Networking, Sensing and Control (ICNSC '06)*, pp. 150–155, 2006.

[55] C. Paul, ‘ITU-T Study Group 12 “Multimedia QoS requirements from a user perspective”.’ [Online]. Available: http://www.itu.int/itudoc/itu-t/workshop/qos/s2p1.pdf. [Accessed: 10-Jan-2011].

[56] J. Kim, I. Park, and C. Kim, ‘A study on the performance enhancements of video streaming service based on MPLS network’, in *Proceedings of International Symposium on Intelligent Signal Processing and Communication Systems. ISPACS '04.*, pp. 601–603, 2004.

[57] S. K. Goel, M. Singh, D. Xu, and B. Li, ‘Efficient peer-to-peer data dissemination in mobile ad-hoc networks’, in *Proceedings of the international conference on Parallel Processing Workshops*, pp. 152–158, 2002.

[58] Y. He, Y. Ouyang, C. Li, and Z. Xiong, ‘Routing Optimized Video Transmission over Mobile Ad Hoc Networks’, in *Workshop on Power Electronics and Intelligent Transportation System, PEITS ’08*, pp. 18–22, 2008.

[59] T. Nunome and S. Tasaka, ‘QoE enhancement of audio-video IP transmission with IEEE 802.11e EDCA in mobile ad hoc networks’, in *20th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, pp. 32–36, 2009.

[60] S. Park and D. Sy, ‘Dynamic control slot scheduling algorithms for TDMA based Mobile Ad Hoc Networks’, in *IEEE Military Communications Conference. MILCOM '08.*, pp. 1–7, 2008.

[61] C. Zhi-gang, G. Zhi-hui, and Z. Ming, ‘Congestion Aware Scheduling Algorithm for MANET’, in *International Conference on Wireless Communications, Networking and Mobile Computing. WiCOM '06.*, pp. 1–5, 2006.

[62] R. K. Basukala, K. Han, D. Choi, Y. Kim, and S. Han, ‘QoS Assurance of Multimedia Traffic in Residential Network with Hybrid Co-ordination Function and Queuing Disciplines’, *in Pacific-Asia Conference On Circuits Communications and Systems. PACCS '09*, pp. 767–770, May 2009.

[63] J. Zou and D. Zhao, ‘Real-time CBR traffic scheduling in IEEE 802.16-based wireless mesh networks’, *Wireless Networks*, vol. 15, no. 1, pp. 65–72, 2009.

[64] Y. Dehbi and N. Mikou, ‘Priority Assignment for Multimedia Packets Scheduling in MANET’, *IEEE International Conference On Signal Image Technology and Internet Based Systems, SITIS '08* ,pp. 32–37, Nov. 2008.

[65] H. Joo, K. J. An, and H. Song, ‘Urgency-based packet scheduling and routing algorithms for video transmission over MANETs’, in *IET International Communication Conference on Wireless Mobile and Computing. CCWMC 2011*, pp. 403–408, Nov. 2011.

[66] K. Manoj, S. C. Sharma, and L. Arya, ‘Fuzzy Based QoS Analysis in Wireless Ad hoc Network for DSR Protocol’, in *IEEE International Advance Computing Conference, IACC '09*, pp. 1357–1361, 2009.

[67] C. Gomathy and S. Shanmugavel, ‘An efficient fuzzy based priority scheduler for mobile ad hoc networks and performance analysis for various mobility models’, in *IEEE Wireless Communications and Networking Conference. WCNC '04*, vol. 2, pp. 1087– 1092, 2004.

[68] J. M. Mendel, ‘Fuzzy logic systems for engineering: a tutorial’, *Proceedings of the IEEE*, vol. 83, no. 3, pp. 345–377, Mar. 1995.

[69] M. Marsudi and N. bin Azman, ‘Queuing theory application to analyze production capacity’, in *Second International Conference on Engineering Systems Management and Its Applications (ICESMA '10),* pp. 1–6, 2010.

[70] L. Tadj, ‘Waiting in line [queuing theory]’, *Potentials IEEE*, vol. 14, no. 5, pp. 11–13, 1995.

[71] ‘Queueing Theory.’ [Online]. Available: http://www.google.co.uk/url?sa=t&rct=j&q=&esrc=s&source=web&cd=24&cad=rja&ved=0CJQBEBYwFw&url=http%3A%2F%2Faini.staff.ugm.ac.id%2Fwp-content%2Fqueuing-theory.pptx&ei=MPzkUejUIuqf0QWZ1YD4AQ&usg=AFQjCNGvd5OrOKE_xtexooE5ugZue-3OZw&sig2=P6A1pBsf0rSpPb7CaxgO7Q&bvm=bv.48705608,d.d2k. [Accessed: 16-Jul-2013].

[72] L. M. Surhone, M. T. Timpledon, and S. F. Marseken, *Queing Model: Queueing Theory, Kendall’s Notation, Markov Property, Erlang Distribution, Degenerate Distribution, Phase-Type Distribution, Markov Chain, Stochastic*. Betascript Publishing, 2010.

[73] G. Lindgren and U. Hoist, ‘Recursive estimation of parameters in Markov-modulated Poisson processes’, *IEEE Transactions on Communications*, vol. 43, no. 11, pp. 2812–2820, Nov. 1995.

[74] H. Yu, C. Harding, M. Hasan Shahidul, and A. Griffiths, ‘Modeling Delay and Packet Drop in Networked Control Systems Using Network Simulator NS2’, *International Journal of Automation and Computing 2*, 2005.

[75] C. A. Harding, A. C. Griffiths, and H. Yu, ‘Optimising Wireless Network Control System Traffic – Using Queuing Theory’, in *Proceedings of the 14th International Conference on Automation & Computing, Brunel University, West London, UK,* London, , GBR, 2008.

[76] L. Xi, *Radio Access Network Dimensioning for 3G UMTS*. Springer, 2011.

[77] Gou-feng Zhao, Qing Shan, Shasha Xiao, and Chuan Xu, ‘Modeling Web Browsing on Mobile Internet’, *IEEE Communication Letter*, vol. 15, no. 10, pp. 1081–1083, 2011.

[78] E. Chlebus and G. Divgi, ‘The Pareto or Truncated Pareto Distribution Measurement-Based Modeling of Session Traffic for Wi-Fi Wireless Internet Access’, in *IEEE Wireless Communications and Networking Conference, WCNC '07*. pp. 3625–3630, 2007

[79] S. Y. Tang, S. Thilakawardana, and R. Tafazolli, ‘Dynamic radio resource management in GSM/GPRS using scalable resource allocation technique’, in *15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications. PIMRC '04.*, vol. 1, pp. 545– 550 Vol.1, 2004.

[80] Y. Qin, Y. Yin, D. Huang, and N. Shah, ‘A comparative study on anonymous 802.11n protocols’, *IEEE Military Communications Conference, MILCOM '08*, pp. 1–7, 16, 2008.

[81] W.-S. Kim and S.-H. Chung, ‘Design and Implementation of IEEE 802.11n in Multi-hop over Wireless Mesh Networks with Multi-Channel Multi-Interface’, in *14th International Conference on High Performance Computing and Communication & IEEE 9th International Conference on Embedded Software and Systems, HPCC-ICESS ‘12,* pp. 707–713, 25, 2012.

[82] M. Lin, J. Tzu, L. Lin, and H. Lee, ‘The IEEE802.11n capability analysis model based on mobile networking architecture’, in *IEEE International Conference on Systems, Man and Cybernetics, SMC '09*, pp. 1857–1860, 2009.

[83] J. M. M. Kamal, M. S. Hasan, A. L. Carrington, and H. Yu, ‘Lessons learned from real MANET experiments and simulation-based evaluation of UDP and TCP’, *13th International Conference On Computing and Information Technology, ICCIT '10*, pp. 511–515, 23, 2010.

[84] D. Kotz, C. Newport, R. S. Gray, J. Liu, Y. Yuan, and C. Elliott, ‘Experimental evaluation of wireless simulation assumptions’, in *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pp. 78–82, Venice, Italy, 2004.

[85] K. Pawlikowski, H.-D. J. Jeong, and J.-S. R. Lee, ‘On credibility of simulation studies of telecommunication networks’, IEEE *Communications Magazine*, vol. 40, no. 1, pp. 132–139, 2002.

[86] M. Cardei, J. Wu, and S. Yang, ‘Topology control in ad hoc wireless networks using cooperative communication’, *IEEE Transactions On Mobile Computing*. vol. 5, no. 6, pp. 711– 724, 2006.

[87] Q. Guan, F. R. Yu, S. Jiang, and V. C. M. Leung, ‘Capacity-Optimized Topology Control for MANETs with Cooperative Communications’, in *IEEE Transaction On Wireless Communication*. vol. 10, no. 7, pp. 2162–2170, 2011.

[88] S. Y. Han and N. B. Abu-Ghazaleh, ‘On the Effect of Fading on Ad-hoc Networks’, *Comput. Sci. Dept State Univ. N. Y. Binghamt.*, Apr. 2005.

[89] R. N. Pupala, ‘Introduction to Wireless Electromagnetic Channels & Large Scale Fading’, *Dep. Electr. Eng. – Rutgers Univ. Piscataway NJ 08854*, 2005.

[90] Z. Ren, Y. Huang, Q. Chen, and H. Li, ‘Modeling and simulation of fading, pathloss, and shadowing in wireless networks’, in *IEEE International Conference on Communications Technology and Applications. ICCTA ’09*, pp. 335–343, 2009.

[91] L. J. Greenstein, V. Erceg, Y. S. Yeh, and M. V. Clark, ‘A new path-gain/delay-spread propagation model for digital cellular channels’, IEEE Transactions on Vehicular Technology, vol. 46, no. 2, pp. 477–485, May 1997.

[92] ‘Thinking Man Software - Dimension 4 v5.0.’ [Online]. Available: http://www.thinkman.com/dimension4/. [Accessed: 23-Oct-2012].

[93] S. Avallone, S. Guadagno, D. Emma, A. Pescape, and G. Ventre, ‘D-ITG distributed Internet traffic generator’, in *First International Conference on the Quantitative Evaluation of Systems, QEST '04.*, pp. 316– 317, 2004.

[94] D. Emma, A. Pescape, and G. Ventre, ‘Analysis and experimentation of an open distributed platform for synthetic traffic generation’, *in 10th IEEE International Workshop on Future Trends of Distributed Computing Systems, FTDCS '04*, pp. 277– 283, 2004.

[95] X. Li, *Radio Access Network Dimensioning for 3G UMTS*. Springer Science & Business Media, 2011.

[96] V. Karthikeyani and T. Thiruvenkadam, ‘Packet size based performance analysis of IEEE 802.11 WLAN comprising virtual server arrays’, *International Conference on Pattern Recognition, Informatics and Mobile Engineering, PRIME '13,* pp. 43–48, 21, 2013.

[97] C. M. Kozierok, *The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference*. No Starch Press, 1 edition 14 oct. 2005.

[98] A. Ferrero, A. Federici, and S. Salicone, ‘Instrumental Uncertainty and Model Uncertainty Unified in a Modified Fuzzy Inference System’, *IEEE Transactions on Instrumentation and Measurement,* vol. 59, no. 5, pp. 1149–1157, May 2010.

[99] A. Hamam and N. D. Georganas, ‘A comparison of Mamdani and Sugeno fuzzy inference systems for evaluating the quality of experience of Hapto-Audio-Visual applications’, in *IEEE International Workshop on Haptic Audio visual Environments and Games, HAVE '08,* pp. 87–92, Oct. 2008.

[100] E. H. Mamdani and S. Assilian, ‘An experiment in linguistic synthesis with a fuzzy logic controller’, *International Journal of Man-Machine Studies,* vol. 7, no. 1, pp. 1–13, Jan. 1975.

[101] T. Takagi and M. Sugeno, ‘Fuzzy identification of systems and its applications to modeling and control’, in IEEE Transactions on Systems, Man and Cybernetics, vol. SMC-15, no. 1, pp. 116–132, Feb. 1985.

[102] N. Mogharreban and L. F. DiLalla, ‘Comparison of Defuzzification Techniques for Analysis of Non-interval Data’, in *Annual meeting of the North American Fuzzy Information Processing Society, NAFIPS '06,* pp. 257–260, Jun. 2006.

[103] M. Braae and D. A. Rutherford, ‘Fuzzy Relations in a Control Setting’, *Kybernetes,* vol. 7, no. 3, pp. 185–188, Dec. 1978.

[104] ‘wireless brochure - System-In-The-Loop (SITL).’ [Online]. Available: http://www.opnet.com/solutions/brochures/SITL.pdf. [Accessed: 14-Oct-2013].

[105] M. Fras, J. Mohorko, and Ž. Čučej, ‘Real time “system-in-the-loop” simulation of tactical networks’, 16th International Conference on Software, Telecommunications and Computer Networks, SoftCOM '08, pp. 105 - 108 , Sept. 2008.

[106] J. Mohorko, M. Fras, and Ž. Čučej, ‘Real video stream transmission over simulated wireless link’, in *International Conference on Advanced Technologies for Communications, ATC 2008,* pp. 231 - 234, oct. 2008

[107] S. Matilda, B. Palaniapp, and J. Thambidurai, ‘Streaming Real-time Video Using Hybrid Transport Layer Protocols in Sim-real-sim Environment’, in The International Journal of Computer Science & Applications, TIJCSA '13, vol. 2, Mar. 2013.

[108] L. Sendre, J. Valiska, and S. Marchevsky, ‘H.264 VIDEO TRANSMISSION OVER WLAN IN OPNET MODELLER’, in *Journal of Electrical Engineering* , vol. 64, no 2, 2013.

[109] L. Zhang, Y.-J. Cheng, and X. Zhou, ‘Rate avalanche: Effects on the performance of multi-rate 802.11 wireless networks’, *Simulation Modelling Practice and Theory*, vol. 17, no. 2, pp. 487–503, Mar. 2009.

[110] Dr Robot, ‘WiRobot DRK6080/8080 User Manuel’, Jun-2006. [Online]. Available: http://www.drrobot.com/products/item_downloads/DRK8080_1.pdf. [Accessed: 01-Sep-2014].

Appendix A : Implementation of Cauchy Distribution Model in OPNET

OPNET PDF has an internal binary format. A PDF can be created by either using the PDF Graphical User Interface (GUI) or the External Model Access (EMA) Application Programming Interface (API). However, the accuracy of using the PDF GUI cannot be guaranteed because it requires the distribution function to be manually drawn. EMA is the best option when accuracy is important. It can programmatically create a PDF file (for example, use EMA to create a PDF file based on a text file or an equation).

Creating of a PDF using EMA involves the following steps:

- Create from scratch a PDF <pdf_model> in the PDF editor; the range of the x-axis outcome can be varied. After the PDF has been created as shown in Figure A-1, then select File->Generate EMA Code.

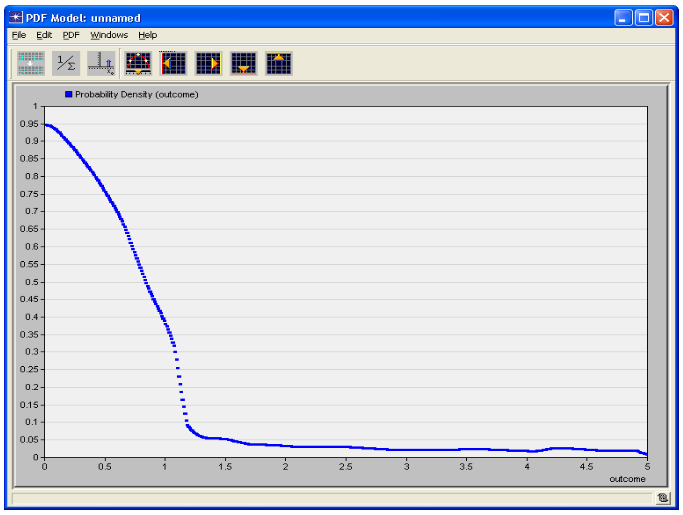


Figure A-1: Sample PDF

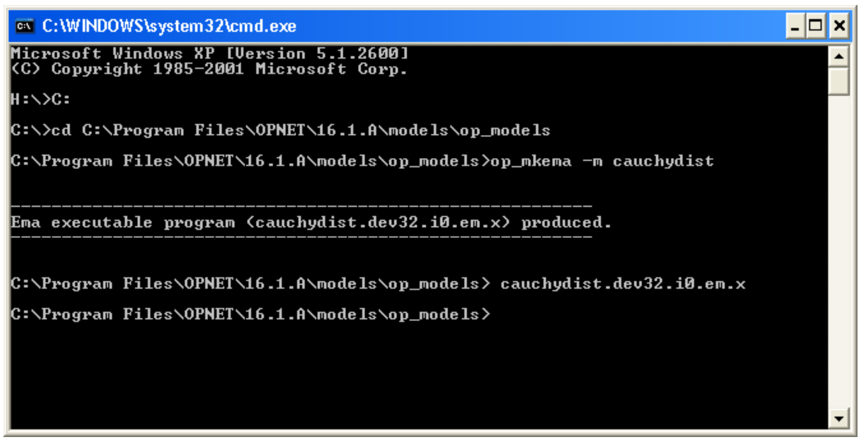
- A ‘c’ file is created; this file should be named <pdf_model.em.c>, at this stage it is important to include the ‘..c’ when naming the file. This file is usually created in the user directory by default or where ever else is top of the top directory in the model directory. A sample of the generated EMA code is shown below.

- The generated EMA code is used as a template, each point in the PDF is listed as a vector in the EMA file. The code is modified to use data points generated by iterative calls to the Cauchy distribution PDF formula or data points from a text file can also be used instead of the vector points defined in the EMA program. Once done, save in a new file name ‘cauchydist.em.c’.

- op_mkema program is used to compile and generate an EMA executable.

To do this on a window: an OPNET console window needs to be started, afterwards the command ‘op_mkema -m cauchydist’ is entered to compile and generate an executable EMA (the ‘.em.c’ extension is not included when compiling the EMA code). An executable called ‘cauchydist.dev32.i0.em.x’ is produced.

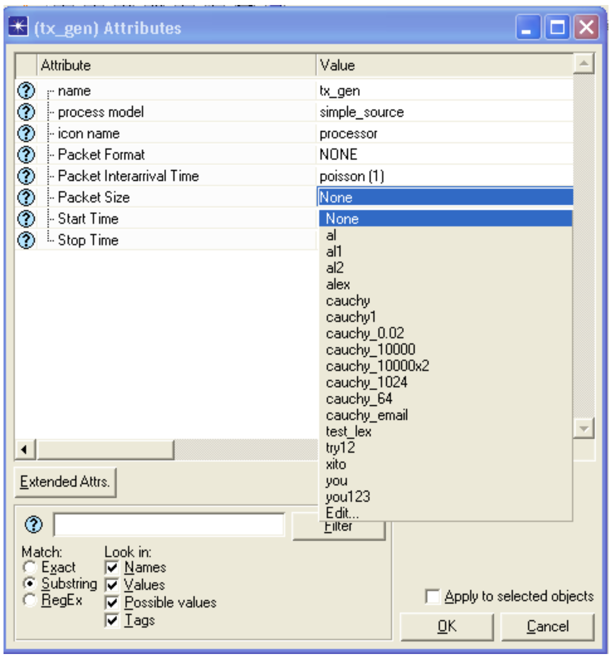
- Run the ‘cauchydist.dev32.i0.em.x’ executable to create the PDF to be used in OPNET. A new files ‘unnamed.pd.s’ is created in the primary user directory. This can be renamed to ‘cauchy.pd.s’ so that it reflects the PDF model created. The execution of the steps in OPNET console can be seen in Figure A-2.



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

H:\>C:

C:\>cd C:\Program Files\OPNET\16.1.A\models\op_models

C:\Program Files\OPNET\16.1.A\models\op_models>op_mkema -m cauchydist


----------------------------------------------------------
Ema executable program (cauchydist.dev32.i0.em.x) produced.
----------------------------------------------------------

C:\Program Files\OPNET\16.1.A\models\op_models> cauchydist.dev32.i0.em.x

C:\Program Files\OPNET\16.1.A\models\op_models>
```

Figure A-2: OPNET Console

- OPNET needs to be restarted or the model directory needs to be refreshed before the new PDF model can be viewed as shown in Figure A-3. This can be done by navigating ‘File->Model Files->Refresh Model Directories’.



(tx_gen) Attributes

Attribute	Value
name	tx_gen
process model	simple_source
icon name	processor
Packet Format	NONE
Packet Interarrival Time	poisson (1)
Packet Size	None
Start Time	
Stop Time	

None
al
al1
al2
alex
cauchy
cauchy1
cauchy_0.02
cauchy_10000
cauchy_10000x2
cauchy_1024
cauchy_64
cauchy_email
test_lex
try12
xito
you
you123
Edit...

Figure A-3: Source node Attributes

EMA CODE

```
#include <opnet.h>
#include <ema.h>
#include <opnet_emadefs.h>
#include <opnet_constants.h>

double                          dvec_0 [] =
                {
                0.947407407407, 0.947407407407, 0.930705882353, 0.930705882353, 0.894183006536,
                0.894183006536, 0.894183006536, 0.853333333333, 0.808539682540, 0.808539682540,
                0.755714285714, 0.755714285714, 0.755714285714, 0.699283068783, 0.621018518519,
                0.621018518519, 0.531111111111, 0.531111111111, 0.451452991453, 0.451452991453,
                0.380769230769, 0.380769230769, 0.254657407407, 0.254657407407, 0.254657407407,
                0.083350000000, 0.059214814815, 0.059214814815, 0.054506172840, 0.054506172840,
                0.052228719948, 0.052228719948, 0.044736842105, 0.044736842105, 0.038056884636,
                0.038056884636, 0.036616161616, 0.036616161616, 0.035429292929, 0.035429292929,
                0.032916666667, 0.032916666667, 0.031138888889, 0.031138888889, 0.031111111111,
                0.031111111111, 0.031111111111, 0.031111111111, 0.031111111111, 0.031111111111,
                0.030330623306, 0.030330623306, 0.028184281843, 0.028184281843, 0.026016260163,
                0.026016260163, 0.023848238482, 0.023848238482, 0.023848238482, 0.022287262873,
                0.022222222222, 0.022222222222, 0.022222222222, 0.022222222222, 0.022222222222,
                0.022222222222, 0.022230769231, 0.022230769231, 0.022777777778, 0.022777777778,
                0.023632478632, 0.023632478632, 0.024136752137, 0.024136752137, 0.022888888889,
                0.022888888889, 0.021407407407, 0.021407407407, 0.019962962963, 0.019962962963,
                0.018703703704, 0.018703703704, 0.018703703704, 0.019185185185, 0.025333333333,
                0.025333333333, 0.025333333333, 0.026533333333, 0.024777777778, 0.024777777778,
                0.022555555556, 0.022555555556, 0.020466666667, 0.020466666667, 0.020000000000,
                0.020000000000, 0.020000000000, 0.020000000000, 0.020000000000, 0.020000000000,
                };
int
main (int argc, char* argv [])
                {
                EmaT_Model_Id                                           model_id;
                int                                                                 i;

                /* initialize EMA package */
                Ema_Init (EMAC_MODE_ERR_PRINT | EMAC_MODE_REL_60, argc, argv);

                /* create an empty model */
                model_id = Ema_Model_Create (MOD_PDF);


                /* set the model level attributes */
                Ema_Model_Attr_Set (model_id,
                                "outcome min",          COMP_CONTENTS, (double) 0,
                                "outcome spacing",      COMP_CONTENTS, (double) 0.05,
                                "prob density",         COMP_INTENDED, EMAC_DISABLED,
                                EMAC_EOL);

                Ema_Model_Attr_Set (model_id, "prob density", COMP_DVEC_SIZE, 100, EMAC_EOL);

                for (i = 0; i < 100; i++)
                                {
                                Ema_Model_Attr_Set (model_id, "prob density", COMP_DVEC_CONTENTS(i), dvec_0 [i], EMAC_EOL);
                                }

                /* write the model to application-readable form */
                Ema_Model_Write (model_id, "unnamed");

                return 0;
                }
```

Appendix B : Implementation of Packet Filter for Mixed Traffic

The process model for the filter is shown in Figure B-4. The filter separates the various traffic profiles to their respective destination in section 3.3.2.4.

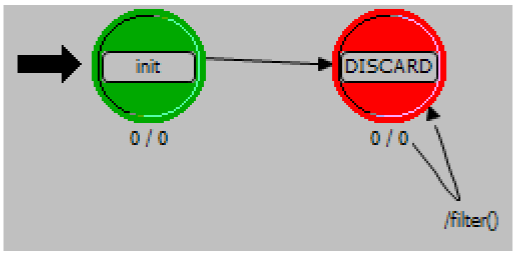
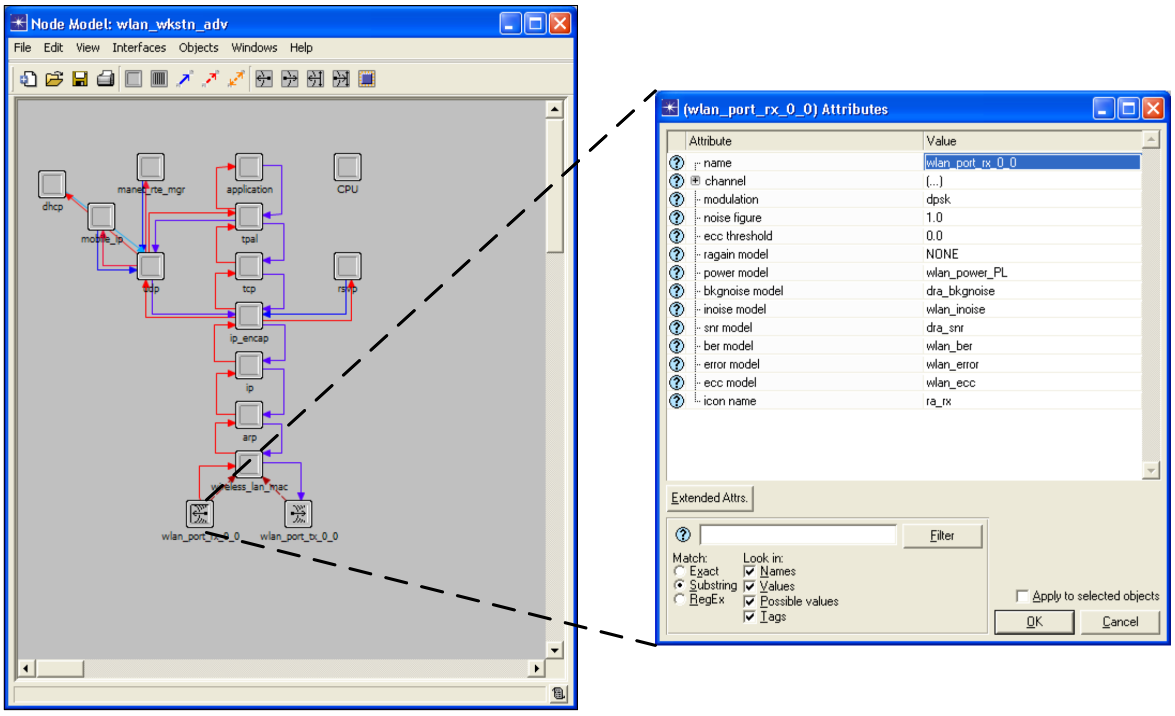


Figure B-4: Process model for Filter

```
static void
filter (void)
          {
          Packet *pkptr;
          int type;
          FIN (filter ());
          //printf("filter: %.8f/n",op_sim_time());
          /*          filters the packets to the relevant sink                    */

          pkptr = op_pk_get (op_intrpt_strm ());
op_pk_nfd_get_int32(pkptr,"type",&type);
switch(type)
          {
          case 1:
          //p1 controller traffic
                    op_pk_send_forced(pkptr,1);
          break;
          case 2:
          //p2 controller traffic
                    op_pk_send_forced(pkptr,2);
          break;
          case 3:
          //p2 controller traffic
                    op_pk_send_forced(pkptr,3);
          break;
          case 4:
          //p2 controller traffic
                    op_pk_send_forced(pkptr,4);
          break;
          default:
                    //external traffic
                    op_pk_send_forced(pkptr,0);
          }
          FOUT;
          }
```

Appendix C :Implementation of the Wireless Propagation Model



(wlan_port_rx_0_0) Attributes

Attribute	Value
name	wlan_port_rx_0_0
channel	(...)
modulation	dpsk
noise figure	1.0
ecc threshold	0.0
ragain model	NONE
power model	wlan_power_PL
bkgnoise model	dra_bkgnoise
inoise model	wlan_inoise
snr model	dra_snr
ber model	wlan_ber
error model	wlan_error
ecc model	wlan_ecc
icon name	ra_rx

Figure C-5: Implementation of the Wireless Propagation Model

```
/* wlan_power.ps.c */
/* Received power model for WLAN suite physical layer modeling.          */

/*****************************************/
/*                  Copyright (c) 1986-2011                    */
/*      by OPNET Technologies, Inc.      */
/*                          (A Delaware Corporation)                    */
/*                7255 Woodmont Av., Suite 250                  */
/*     Bethesda, MD 20814, U.S.A.      */
/*                                   All Rights Reserved.                */
/*****************************************/

#include "opnet.h"
#include "wlan_support.h"
#include "jammers.h"
#include "oms_dist_support_base.h"
#include <math.h>
#include <oms_fhss_support.h>
#include <oms_dist_support.h>
#include <oms_string_support.h>


/***** constants *****/
```

```
#define C                                                       3.0E+08
        /* speed of light (m/s) */
#define SIXTEEN_PI_SQ                  (16.0 * VOSC_NA_PI * VOSC_NA_PI)        /* 16 times pi-squared */
#define L           1
#define DIST0       1
#define PATHLOSSEXP    2.02
#define STD_DB      6.5
#if 0
static const char*      PowI_Err_Hdr = "Error in radio power computation pipeline stage (wlan_power):";
#endif


/***** pipeline procedure *****/

#if defined (__cplusplus)
extern "C"
#endif
void
wlan_power_PL_mt (OP_SIM_CONTEXT_ARG_OPT_COMMA Packet* pkptr)
        {
        double                                  prop_distance, rcvd_power, path_loss;
        double                                  tx_power, tx_base_freq, tx_bandwidth, tx_center_freq;
        double                                  lambda, rx_ant_gain, tx_ant_gain;
        Objid                                   rx_ch_obid;
        double                                  in_band_tx_power, band_max, band_min;
        double                                  rx_base_freq, rx_bandwidth;
        double                                  pk_reception_end;
        int                                             rcvd_frame_type;
        int                                             chanmatch_status;
        WlanT_Rx_State_Info *  rx_state_ptr;
/*##################### Alex Egaji #######################*/
        double rcvd_power0, avg_db, powerLoss_db; // std_db,rcvd_power1;
        Distribution *fading;
/*#########################################################*/
        Boolean                                 is_jammer;
        double                                  current_time;
        Boolean                                 is_receiving_jammer;
        double                                  channel_overlap;

        /** Compute the average received power in Watts of the          **/
        /** signal associated with a transmitted packet.                **/
        FIN_MT (wlan_power_PL (pkptr));

        /* Get power allotted to transmitter channel.                           */
        tx_power = op_td_get_dbl (pkptr, OPC_TDA_RA_TX_POWER);

        /* Get transmission frequency in Hz.                                            */
        tx_base_freq = op_td_get_dbl (pkptr, OPC_TDA_RA_TX_FREQ);
        tx_bandwidth = op_td_get_dbl (pkptr, OPC_TDA_RA_TX_BW);
        tx_center_freq = tx_base_freq + (tx_bandwidth / 2.0);

        /* When using TMM, the TDA OPC_TDA_RA_RCVD_POWER will                   */
        /* already have a raw value for the path loss.                          */
        if (op_td_is_set (pkptr, OPC_TDA_RA_RCVD_POWER))
                {
                path_loss = op_td_get_dbl (pkptr, OPC_TDA_RA_RCVD_POWER);
                }
        else
                {
                /* Caclculate wavelength (in meters).                                   */
                lambda = C / tx_center_freq;

                /* Get distance between transmitter and receiver (in            */
                /* meters).
        */
                prop_distance = op_td_get_dbl (pkptr, OPC_TDA_RA_START_DIST);

                /* Compute the path loss for this distance and                  */
                /* wavelength.
                */
                if (prop_distance > 0.0)
                        {
/*####################Alex Egaji##########################*/
                        path_loss = PATHLOSSEXP;

/*####################################################*/
                        }
                else
                        path_loss = 1.0;
                }

        /* Compute the in-band transmission power by multiplying        */
        /* the transmission power with the overlap ratio between          */
        /* the frequency bands of the transmitter and receiver.                   */
        chanmatch_status = op_td_get_int (pkptr, OPC_TDA_RA_MATCH_STATUS);
        if (chanmatch_status == OPC_TDA_RA_MATCH_VALID)
                {
                /* Since the packet status is not set to noise, there   */
                /* must be a exact match between the bands (i.e.                  */
                /* overlap ratio = 1.0).                                                                    */
                in_band_tx_power = tx_power;
                }
        else
                {
                /* Determine the receiver bandwidth and base frequency.          */
                rx_base_freq = op_td_get_dbl (pkptr, OPC_TDA_RA_RX_FREQ);
                rx_bandwidth = op_td_get_dbl (pkptr, OPC_TDA_RA_RX_BW);

                /* Use these values to determine the band overlap with             */
                /* the transmitter. Note that if there were no overlap              */
                /* at all, the packet would already have been filtered              */
                /* by the channel match stage.                                                              */

                /* The base of the overlap band is the highest base                */
                /* frequency.
                */
                if (rx_base_freq > tx_base_freq)
                        band_min = rx_base_freq;
                else
                        band_min = tx_base_freq;

                /* The top of the overlap band is the lowest end                    */
                /* frequency.
                */
                if (rx_base_freq + rx_bandwidth > tx_base_freq + tx_bandwidth)
                        band_max = tx_base_freq + tx_bandwidth;
                else
                        band_max = rx_base_freq + rx_bandwidth;

                /* Compute the channel overlap */
                channel_overlap = Oms_FHSS_Channel_Overlap_Get (pkptr);
                if (channel_overlap == OMSC_FHSS_NOT_USED)
                        channel_overlap = (band_max - band_min) / tx_bandwidth;

                /* Compute the amount of in-band transmitter power.             */
                in_band_tx_power = tx_power * channel_overlap;
                }

        /* Get antenna gains (raw form, not in dB).                                        */
        tx_ant_gain = pow (10.0, op_td_get_dbl (pkptr, OPC_TDA_RA_TX_GAIN) / 10.0);
        rx_ant_gain = pow (10.0, op_td_get_dbl (pkptr, OPC_TDA_RA_RX_GAIN) / 10.0);

/*############################## Alex Egaji ##############################*/

        /* Calculate received power levelby friis model at referece DIST0.              */


        rcvd_power0 = in_band_tx_power * tx_ant_gain * rx_ant_gain * pow(lambda, 2); //ED

        rcvd_power0 = rcvd_power0 / ( L * 16 * pow(3.14159265, 2) * pow(DIST0, 2));//ED


        avg_db = -10.0 * path_loss* log10 (prop_distance / DIST0); //ED

        /*get power loss by adding a log-normal random variable (fading)*/

        fading = op_dist_load ("normal",0, (STD_DB)); // ED

        powerLoss_db = avg_db +  op_dist_outcome(fading); //ED


        /* calculate the received power at distance */

        rcvd_power = rcvd_power0 * pow(10.0, powerLoss_db/10.0); // ED
/*########################################################################*/

        /* Assign the received power level (in Watts) to the packet        */
        /* transmission data attribute.                                                               */
        op_td_set_dbl (pkptr, OPC_TDA_RA_RCVD_POWER, rcvd_power);

        /* Now we need to check whether the receiver will lock onto       */
        /* this signal, which depends on the receiver's current               */
        /* status and the reception power of the signal.                          */
        /* First check the status of the receiver node.                                    */
```

```
if (op_td_is_set (pkptr, OPC_TDA_RA_ND_FAIL))
        {
        /* The receiving node is disabled. Change the channel         */
        /* match status to noise.                                                              */
        op_td_set_int (pkptr, OPC_TDA_RA_MATCH_STATUS, OPC_TDA_RA_MATCH_NOISE);
        }
else
        {
        /* Get the current simulation time.                                              */
        current_time = op_sim_time ();

        /* Decide if the incoming pakcet is a jammer packet.          */
        is_jammer = op_pk_encap_flag_is_set (pkptr, OMSC_JAMMER_ENCAP_FLAG_INDEX);

        /* The receiving node is enabled. Retrieve the state */
        /* information of the receiver channel.                                  */
        rx_ch_obid   = op_td_get_int (pkptr, OPC_TDA_RA_RX_CH_OBJID);

        /* The FHSS package provides an API to retrieve the channel state information.  This API will return the default channel        */
        /* object state if FHSS is not used or disabled. If FHSS is enabled, then the API will retrieve the user state information        */
        /* from the FHSS state information. All access to channel state information has to be performed via this API to ensure
*/
        /* consistent operation.

                        */
        rx_state_ptr = (WlanT_Rx_State_Info *) Oms_FHSS_User_Channel_State_Get (rx_ch_obid);

        /* Decide if the receiver is already receving a jammer  */
        /* or there is a jammer in the background.                             */
        if (rx_state_ptr->jammer_rx_end_time >= current_time)
                is_receiving_jammer = OPC_TRUE;
        else
                is_receiving_jammer = OPC_FALSE;

        /* If the receiver is already receiving another WLAN    */
        /* packet, then this packet will be considered as noise.*/
        /* This         prevents simultaneous reception of multiple           */
        /* valid packets.  If the receiver is already receiving */
        /* a jammer, don't let jammer causes WLAN frame to be           */
        /* marked as noise. If the receiver is receiving a WLAN          */
        /* frame with a jammer at the background, the following */
        /* WLAN frame will be marked as noise.                                       */
        if (rx_state_ptr->rx_end_time <= current_time ||
                (is_receiving_jammer == OPC_TRUE && is_jammer == OPC_FALSE && rx_state_ptr->wlan_pk_rx_end_time <=
current_time))
                {
                /* The receiver is idle. Check the packet's                    */
                /* reception power value.                                                    */
                if (rcvd_power > rx_state_ptr->rx_power_thresh)
                        {
                        /* Check the status of the packet. It could be          */
                        /* set to noise in channel match stage because        */
                        /* of imperfect match (interference from                       */
                        /* another BSS). If this is the case, since it            */
                        /* is still a powerful packet and arriving the           */
                        /* receiver while it is idling, change its                       */
                        /* status to "VALID" and set its "Accept"                     */
                        /* field appropriately.                                                          */
                        if (chanmatch_status == OPC_TDA_RA_MATCH_NOISE)
                                {

                                op_td_set_int (pkptr, OPC_TDA_RA_MATCH_STATUS,
OPC_TDA_RA_MATCH_VALID);

                                /* Make sure that this a WLAN packet, not a          */
                                /* jammer packet.
*/
                                if (is_jammer == OPC_FALSE)
                                        op_pk_nfd_set_int32 (pkptr, "Accept",
WLANC_NEIGHBOR_BSS_PKT_REJECT);

                                }

                        /* Update the WLAN receiving end time for WLAN  */
                        /* frame.
        */
                        if (is_jammer == OPC_FALSE)
                                {
                                pk_reception_end = op_td_get_dbl (pkptr, OPC_TDA_RA_END_RX);
                                if (pk_reception_end > rx_state_ptr->wlan_pk_rx_end_time)
                                        rx_state_ptr->wlan_pk_rx_end_time = pk_reception_end;

                                /* Receiver is busy due to a valid WLAN                        */
```

```
                                        /* packet.
                    */
                                        rx_state_ptr->busy_due_to_jammer = OPC_FALSE;
                                        }
                                else
                                        {
                                        /* Receiver is busy due to a valid jammer          */
                                        /* packet.
                    */
                                        rx_state_ptr->busy_due_to_jammer = OPC_TRUE;
                                        }
                                }
                        else
                                {
                                /* Packet's power is too weak to turn on the        */
                                /* signal lock and to remain a valid packet.        */
                                /* Change its status to noise.                                          */
                                op_td_set_int (pkptr, OPC_TDA_RA_MATCH_STATUS, OPC_TDA_RA_MATCH_NOISE);

                                /* Beacons are monitored for evaluating                          */
                                /* connectivity to the AP. One method to model     */
                                /* Beacon monitoring is to periodically check      */
                                /* for Beacon reception, by setting periodic       */
                                /* self-interrupts. However, this method is not    */
                                /* very efficient. Therefore, we will rig the      */
                                /* physical layer so that all Beacons are                  */
                                /* processed for reliability. Those that can't     */
                                /* be correctly demodulated will be used to                */
                                /* model missing beacons without actually    */
                                /* using an inefficient self-timer mechanism.      */
                                /* Only consider the Beacons that are                      */
                                /* transmitted at the channel currently used by    */
                                /* the receiver (i.e. if the initial status is     */
                                /* VALID).
                    */
                                if (rx_state_ptr->roaming_info_ptr->enable_roaming
                    &&
                                        rx_state_ptr->roaming_info_ptr->scan_type == WlanC_Scan_Type_Beacon &&
                                        chanmatch_status
== OPC_TDA_RA_MATCH_VALID   )
                                        {
                                        op_pk_nfd_get_int32 (pkptr, "Type", (int *) &rcvd_frame_type) ;
                                        if (rcvd_frame_type == WlanC_Beac)
                                                {
                                                /* Serialize the execution of AP                  */
                                                /* reliability evaluation, since the      */
                                                /* MAC of the receiver can be reading/            */
                                                /* writing the same information.                  */
                                                op_prg_mt_mutex_lock (rx_state_ptr->roaming_info_ptr->roam_info_mutex,
0);
                                                wlan_ap_reliability_eval (pkptr, OPC_FALSE, rx_state_ptr-
>roaming_info_ptr);
                                                op_prg_mt_mutex_unlock (rx_state_ptr->roaming_info_ptr-
>roam_info_mutex);
                                                }
                                        }
                                }
                        }
                else
                        {
                        /* The channel is already busy. We will treat the     */
                        /* current packet as noise.                                                                  */
                        op_td_set_int (pkptr, OPC_TDA_RA_MATCH_STATUS, OPC_TDA_RA_MATCH_NOISE);

                        /* See comment in above that explains special                 */
                        /* treatment of Beacons.                                                      */
                        if (rx_state_ptr->roaming_info_ptr->enable_roaming && rx_state_ptr->roaming_info_ptr->scan_type ==
WlanC_Scan_Type_Beacon &&
                                chanmatch_status == OPC_TDA_RA_MATCH_VALID)
                                {
                                op_pk_nfd_get_int32 (pkptr, "Type", (int *) &rcvd_frame_type) ;
                                if (rcvd_frame_type == WlanC_Beac)
                                        {
                                        /* Serialize the execution of AP                          */
                                        /* reliability evaluation, since the MAC of           */
                                        /* the receiver can be reading/writing the            */
                                        /* same information.                                                          */
                                        op_prg_mt_mutex_lock (rx_state_ptr->roaming_info_ptr->roam_info_mutex, 0);
                                        wlan_ap_reliability_eval (pkptr, OPC_FALSE, rx_state_ptr->roaming_info_ptr);
                                        op_prg_mt_mutex_unlock (rx_state_ptr->roaming_info_ptr->roam_info_mutex);
                                        }
                                }
                        }
```

```
		/* Update the reception end time for the receiver based		*/
		/* on the new packet if its power is exceeding the			*/
		/* threshold.
*/
		if (rcvd_power > rx_state_ptr->rx_power_thresh)
			{
			pk_reception_end = op_td_get_dbl (pkptr, OPC_TDA_RA_END_RX);
			if (pk_reception_end > rx_state_ptr->rx_end_time)
				rx_state_ptr->rx_end_time = pk_reception_end;

			/* Update the jammer reception end time since a new */
			/* valid jammer comes.											*/
			if (is_jammer == OPC_TRUE && pk_reception_end > rx_state_ptr->jammer_rx_end_time)
				rx_state_ptr->jammer_rx_end_time = pk_reception_end;
			else if (pk_reception_end > rx_state_ptr->wlan_pk_rx_end_time)
				rx_state_ptr->wlan_pk_rx_end_time = pk_reception_end;
			}
		}

FOUT;
}
```

Appendix D : Implementation of the Mamdani Fuzzy Scheduler

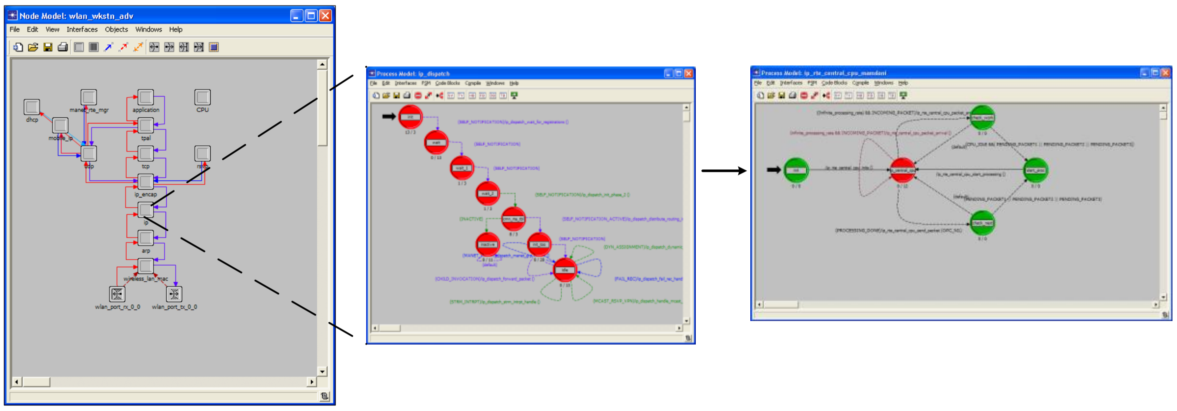


Figure D-6: Implementation of Mamdani

```
/* Transition executives */

static void
ip_rte_central_cpu_inits (void)
        {
        IpT_Interface_Info *    iface_info_ptr;
        int                                             i;
        int                                             num_if;
        IpT_Group_Intf_Info*    group_info_ptr;
        int                                             ith_member, num_member_intfs;
        IpT_Member_Intf_Info* member_intf_ptr;


        /** Perform appropriate initializations before reaching the wait state **/
        FIN (ip_rte_central_cpu_inits ());

        /* Obtain module memory and set up for ip_rte_int routines */
        module_data_ptr = (IpT_Rte_Module_Data *)op_pro_modmem_access ();
        ip_rte_set_procs (module_data_ptr, ip_rte_cpu_error, ip_rte_cpu_warn);

        own_id = op_id_self ();
        own_prohandle = op_pro_self ();

        /* Create pool */
        ip_rte_routing_processor_info_init ();

        /* Unless the processing speed is set to infinity, obtain          */
        /* CPU resource handle to schedule shared processing                       */
        /* throughout the node.                                                                                              */
        if (! infinite_processing_rate)
                {
                cpu_resource_handle = Oms_Resource_Handle_Get (module_data_ptr->node_id,
                        module_data_ptr->module_id, "CPU");
                cpu_is_idle = OPC_TRUE;
                }

        /* If QoS is in place, invoke interface process model to         */
        /* let it create its buffers using the central memory pool       */
        /* Also take control of any stream interrupts coming on                    */
        /* each interface.
        */
        num_if = inet_rte_num_interfaces_get (module_data_ptr);
        for (i = 0; i < num_if; ++i)
                {
                iface_info_ptr = inet_rte_intf_tbl_access (module_data_ptr, i);

                /* Unless the stream index is invalid, register for all          */
                /* stream interrupts on that stream. Logical interfaces          */
                /* like loopback interfaces and tunnel interfaces                */
                /* might have invalid stream indices.                                              */

                /* In the case of interface groups, register for stream          */
                /* interrupts on the stream indices of all member                */
                /* interfaces.
                */
                if (ip_rte_intf_is_group (iface_info_ptr))
                        {
                        /* Get a handle to structure that stores group                 */
                        /* related parameters.                                                                   */
                        group_info_ptr = iface_info_ptr->phys_intf_info_ptr->group_info_ptr;

                        /* Get the number of member interfaces.                                  */
                        num_member_intfs = group_info_ptr->num_members;

                        /* Loop through each member interface                                    */
                        for (ith_member = 0; ith_member < num_member_intfs; ith_member++)
                                {
                                /* Get a handle to the ith member structure                */
                                member_intf_ptr = &(group_info_ptr->member_intf_array[ith_member]);

                                /* Register for the stream interface.                      */
                                op_intrpt_port_register (OPC_PORT_TYPE_STRM,
                                        member_intf_ptr->instrm, own_prohandle);
                                }
                        }
                else if (ip_rte_intf_in_port_num_get (iface_info_ptr) != IPC_PORT_NUM_INVALID)
                        {
                        op_intrpt_port_register (OPC_PORT_TYPE_STRM,
                                ip_rte_intf_in_port_num_get (iface_info_ptr), own_prohandle);
                        }

                if (iface_info_ptr->queuing_scheme != IpC_No_Queuing)
                        {
                        op_pro_invoke (iface_info_ptr->output_iface_prohandle, routing_buffer_pool);
                        }
                }

        /* Take control of stream interrupts coming from the upper layers */
        op_intrpt_port_register (OPC_PORT_TYPE_STRM,
                module_data_ptr->instrm_from_ip_encap, own_prohandle);

        /* Check to see if there is the possibility of background utilization          */
        /* traffic.
                                                */
        if ((module_data_ptr->do_bgutil) && (! infinite_processing_rate))
                {
                /* Initialize the background utilization state ptr for the central cpu*/
                module_data_ptr->bgutil_routed_state_ptr = OPC_NIL;

                /* Initialize the background utilization state ptr for keeping track */
                /* of sent and received background utilization traffic.                          */
                module_data_ptr->received_bgutil_routed_state_ptr = oms_bgutil_routed_state_create (
                        !service_rate_pps, DO_NOT_SCALE);
                module_data_ptr->sent_bgutil_routed_state_ptr = oms_bgutil_routed_state_create (
                        !service_rate_pps, DO_NOT_SCALE);

                /* Init the time at which background utilization statistics      */
                /* generation begins to time 0.
        */
                module_data_ptr->received_last_stat_update_time = 0.0;
                module_data_ptr->sent_last_stat_update_time = 0.0;

                /* Schedule a procedure which will be called for this process at the*/
                /* end of simulation to update background utilization statistics if            */
                /* needed.
                                                */
                op_intrpt_schedule_call (OPC_INTRPT_SCHED_CALL_ENDSIM, 0,
                        ip_rte_pk_stats_update_endsim, module_data_ptr);
                }

        FOUT;

        }
```

```
static void
ip_rte_central_cpu_packet_arrival (void)
	{
	Packet *							pkptr = OPC_NIL;
	int								instrm;
	double			PQ,PI;
	Ici *		iciptr;
	IpT_Rte_Ind_Ici_Fields *		intf_ici_fdstruct_ptr = OPC_NIL;
	IpT_Interface_Info *		rcvd_iface_info_ptr = OPC_NIL;
	int								result;

	double Low[3],Medium[3],High1[3],y[27],x1,x2,x3;		//AE

	double b10,c10,a11,b11,c11,a12,b12,b20,c20,a21,b21,c21,a22,b22,c22,b30,c30,a31,b31,c31,a32,b32,c32; //AE
	int i;	//AE
	char mf1[3],mf2[3],mf3[3];

	double  x[N],a,totalsum, multiplesum,output,g;  //AE
	double yout[N],yout_1,yout_2,yout1[N],yout2[N],yout3[N],yout4[N],yout5[N],yout6[N],yout7[N],yout8[N],yout9[N],yout10[N],yout11[N];
	double  yout12[N],yout13[N],yout14[N],yout15[N];//AE
	double
yout16[N],yout17[N],yout18[N],yout19[N],yout20[N],yout21[N],yout22[N],yout23[N],yout24[N],yout25[N],yout26[N],yout27[N];//AE



	/** An incoming packet has arrived.  It might	**/
	/** be from an "upper layer", a "lower layer",	**/
	/** or generated from within ip.						**/
	FIN (ip_rte_central_cpu_packet_arrival ());

	if (invoke_mode == OPC_PROINV_INDIRECT)
		{
		/* Packet generated from withing IP and forwarded by our	*/
		/* parent process.
		*/
		pkptr = (Packet *)op_pro_argmem_access ();
		instrm = IpC_Pk_Instrm_Child;
		}
	else
		{
		/* Packet coming from some stream */
		instrm = op_intrpt_strm ();
		pkptr = op_pk_get (instrm);
		if (pkptr == OPC_NIL)
			ip_rte_cpu_error ("Unable to get packet from input stream.");
		}

	/* Perform standard IP processing of incoming packet		*/
	/*		1. Perform forwarding decision and populate the	ICI		*/
	/*		2. Populate rcvd_iface_info_ptr. It is set to NIL	*/
	/*		   if packet arrives from higher layer.					*/
	result = ip_rte_packet_arrival (module_data_ptr,
		&pkptr, instrm, &intf_ici_fdstruct_ptr, &rcvd_iface_info_ptr);

	if (result == OPC_FALSE)
		{
		/* Packet was dropped in call */
		FOUT;
		}


	/* If the processing speed is set to infinity, forward */
	/* the packet immediately.														*/
	if (infinite_processing_rate)
		{

			ip_rte_central_cpu_send_packet (pkptr);
		}
	else
		{


	 /*x1=SNR(signal to noise ratio), x2= Data rate, x3= Queue */


 /********Fuzzification*********/


		if (x1>1)
		{
		x1=1;
		}
```

```
        else ;
        if (x2>1) // data rate
                {
                x2=1;
                }
        else ;
        if (x3>100)  //queue length
                {
                x3=100;
                }
        else ;



        b10 = 0; c10 = 0.4;     a11 = 0.1; b11 = 0.5; c11 = 0.9;        a12 = 0.6; b12 = 1; //SNR
        b20 = 0; c20 = 0.4;     a21 = 0.3; b21 = 0.5; c21 = 0.7;        a22 = 0.6; b22 = 0.8; c22=1; // Data rate
        b30 = 0; c30 = 10;     a31 = 8; b31 = 20; c31 = 30;        a32 = 22; b32 = 60; c32=100; //queue length



       //SNR/

        Low[1] = max((c10-x1)/(c10-b10),0);
        Medium[1] = max(min((x1-a11)/(b11-a11),(c11-x1)/(c11-b11)),0);
        High1[1] = max((x1-a12)/(b12-a12),0);


        if (Low[1]>0)
                mf1[1]='l';
        else mf1[1]='N';
        if (Medium[1]>0)
                mf1[2]='m';
        else mf1[2]='N';
        if (High1[1]>0)
                mf1[3]='h';
        else mf1[3]='N';

        //Data rate/

        Low[2] = max((c20-x2)/(c20-b20),0);

        Medium[2] = max(min((x2-a21)/(b21-a21),(c21-x2)/(c21-b21)),0);

        if (x2<0.8){

        High1[2] = max((x2-a22)/(b22-a22),0);
        }
        else {
                High1[2]=1;
                }



        if (Low[2]>0)
                mf2[1]='l';

        else mf2[1]='N';
        if (Medium[2]>0)
                mf2[2]='m';
        else mf2[2]='N';
        if (High1[2]>0)
                mf2[3]='h';
        else mf2[3]='N';


        //Queue length/

        Low[3] = max((c30-x3)/(c30-b30),0);

        Medium[3] = max(min((x3-a31)/(b31-a31),(c31-x3)/(c31-b31)),0);

        if (x3<60){
        High1[3] = max((x3-a32)/(b32-a32),0);
        }
        else {
                High1[3]=1;
                }

        if (Low[3]>0)
                mf3[1]='l';

        else mf3[1]='N';
```

```
            if (Medium[3]>0)
                        mf3[2]='m';
            else mf3[2]='N';
            if (High1[3]>0)
                        mf3[3]='h';
            else mf3[3]='N';



            /***********Implication***********/
            for(i=0;i<N;i++)
{
            yout1[i]=0;

            }

if (mf1[1]=='l' && mf2[1]=='l' &&  mf3[1]=='l')          // 1
                         {

y[1]= min( min(Low[1],Low[2]),Low[3]);    //m

yout1[2]= y[1];

}

 else;
//****************************************************************************************/
 for(i=0;i<N;i++)
{
            yout2[i]=0;

            }

if (mf1[1]=='l' && mf2[1]=='l' &&  mf3[2]=='m')          // 2
                         {

y[2]= min( min(Low[1],Low[2]),Medium[3]);    //m

yout2[2]= y[2];

}

 else ;
//****************************************************************************************/
 for(i=0;i<N;i++)
{
            yout3[i]=0;

            }
if (mf1[1]=='l' && mf2[1]=='l' &&  mf3[3]=='h')          // 3
                         {

y[3]= min( min(Low[1],Low[2]),High1[3]);    //vl


yout3[0]= y[3];
}

 else ;

//****************************************************************************************/
 for(i=0;i<N;i++)
{
            yout4[i]=0;

            }
if (mf1[1]=='l' && mf2[2]=='m' &&  mf3[1]=='l')          // 4
                         {

y[4]= min( min(Low[1],Medium[2]),Low[3]);    //m


yout4[2]=y[4];

}


 else ;
//****************************************************************************************/
for(i=0;i<N;i++)
{
            yout5[i]=0;
            }

 if (mf1[1]=='l' && mf2[2]=='m' && mf3[2]=='m')     // 5

             {

y[5]=min(min(Low[1],Medium[2]), Medium[3]);     //l


yout5[1]= y[5];


}

 else ;
//*****************************************************************/
for(i=0;i<N;i++)
{
            yout6[i]=0;

            }

 if (mf1[1]=='l' && mf2[2]=='m' && mf3[3]=='h')     // 6

             {
             for(i=0;i<N;i++)
{
            yout6[i]=0;

            }

y[6]=min(min(Low[1],Medium[2]), High1[3]);     //vl


yout6[0]= y[6];

}

 else ;
//******************************************************************/
for(i=0;i<N;i++)
{
            yout7[i]=0;

            }

 if (mf1[1]=='l' && mf2[3]=='h' && mf3[1]=='l')      // 7

            {

y[7]=min(min(Low[1],High1[2]),Low[3]);     //m


yout7[2]= y[7];

}


 else ;
//*******************************************************************/
for(i=0;i<N;i++)
{
            yout8[i]=0;

            }
 if (mf1[1]=='l' && mf2[3]=='h' && mf3[2]=='m')      // 8

            {

y[8]=min(min(Low[1],High1[2]),Medium[3]);     //m


yout8[2]= y[8];

}


 else ;
//********************************************************************/
for(i=0;i<N;i++)
{
            yout9[i]=0;
```

```
                    }

 if (mf1[1]=='l' && mf2[3]=='h' && mf3[3]=='h')       // 9

                    {

y[9]=min(min(Low[1],High1[2]),High1[3]);     //vl


yout9[0]= y[9];

}


 else ;
//***********************************************************************/
for(i=0;i<N;i++)
{
                    yout10[i]=0;

                    }
 if (mf1[2]=='m' && mf2[1]=='l' && mf3[1]=='l')            // 10

                    {

y[10]= min(min(Medium[1],Low[2]),Low[3]);                //h



yout10[3]= y[10];
}

 else ;
//**********************************************************************/
for(i=0;i<N;i++)
{
                    yout11[i]=0;

                    }

 if (mf1[2]=='m' && mf2[1]=='l' && mf3[2]=='m')          //11

                     {

y[11] = min(min(Medium[1],Low[2]), Medium[3]) ;          //m



yout11[2]= y[11];

}

 else;
//**********************************************************************/
for(i=0;i<N;i++)
{
                    yout12[i]=0;

                    }

 if (mf1[2]=='m' && mf2[1]=='l' && mf3[3]=='h')         //12

                     {

y[12]=min(min(Medium[1], Low[2]),High1[3]);      //l


yout12[1]=y[12];

}

 else;
//***********************************************************************/
for(i=0;i<N;i++)
{
                    yout13[i]=0;

                    }

 if (mf1[2]=='m' && mf2[2]=='m' && mf3[1]=='l')                  //13

                     {
```

```
y[13]=min(min(Medium[1], Medium[2]),Low[3]);       //h

yout13[3]= y[13];

}
 else;
//************************************************************************/
for(i=0;i<N;i++)
{
                    yout14[i]=0;

                    }

 if (mf1[2]=='m' && mf2[2]=='m' && mf3[2]=='m')                      //14

                     {

y[14]=min(min(Medium[1], Medium[2]),Medium[3]);       //m

yout14[2]=y[14];

}

 else;
//**************************************************************************/
for(i=0;i<N;i++)
{
                    yout15[i]=0;

                    }
 if (mf1[2]=='m' && mf2[2]=='m' && mf3[3]=='h')                      //15

                     {

y[15]=min(min(Medium[1], Medium[2]),High1[3]);       //l

yout15[1]= y[15];
}

 else;
//***************************************************************************/
for(i=0;i<N;i++)
{
                    yout16[i]=0;

                    }

 if (mf1[2]=='m' && mf2[3]=='h' && mf3[1]=='l')                      //16

                     {

y[16]=min(min(Medium[1], High1[2]),Low[3]);       //h

yout16[3]= y[16];

}
 else;
//***************************************************************************/
for(i=0;i<N;i++)
{
                    yout17[i]=0;

                    }

 if (mf1[2]=='m' && mf2[3]=='h' && mf3[2]=='m')                      //17

                     {

y[17]=min(min(Medium[1], High1[2]),Medium[3]);       //m

yout17[2]= y[17];

}
```

```
 else;
//*******************************************************************************/

 for(i=0;i<N;i++)
{
            yout18[i]=0;

            }

 if (mf1[2]=='m' && mf2[3]=='h' && mf3[3]=='h')              //18

             {

y[18]=min(min(Medium[1],High1[2]),High1[3]);     //l


yout18[1]= y[18];

}

 else;
//*******************************************************************************/

 for(i=0;i<N;i++)
{
            yout19[i]=0;

            }

 if (mf1[3]=='h' && mf2[1]=='l' && mf3[1]=='l')              //19

             {

y[19]=min(min(High1[1], Low[2]),Low[3]);     //h


yout19[3]= y[19];

}

 else;
//*******************************************************************************/
for(i=0;i<N;i++)
{
            yout20[i]=0;

            }

 if (mf1[3]=='h' && mf2[1]=='l' && mf3[2]=='m')              //20

             {

y[20]=min(min(High1[1], Low[2]),Medium[3]);     //m



yout20[2]= y[20];

}


 else;
//*******************************************************************************/
for(i=0;i<N;i++)
{
            yout21[i]=0;

            }

 if (mf1[3]=='h' && mf2[1]=='l' && mf3[3]=='h')              //21

             {

y[21]=min(min(High1[1], Low[2]),High1[3]);     //l


yout21[1]= y[21];

}
```

```
 else;
//***********************************************************************************/
for(i=0;i<N;i++)
{
                yout22[i]=0;

                }

 if (mf1[3]=='h' && mf2[2]=='m' && mf3[1]=='l')                  //22

                 {

y[22]=min(min(High1[1], Medium[2]),Low[3]);       //VH


yout22[4]= y[22];

}

 else;
//*********************************************************************************/
for(i=0;i<N;i++)
{
                yout23[i]=0;

                }

 if (mf1[3]=='h' && mf2[2]=='m' && mf3[2]=='m')                   //23

                 {

y[23]=min(min(High1[1], Medium[2]),Medium[3]);       //H


yout23[3]= y[23];

}


 else;
//*********************************************************************************/
for(i=0;i<N;i++)
{
                yout24[i]=0;

                }

 if (mf1[3]=='h' && mf2[2]=='m' && mf3[3]=='h')                  //24

                 {

y[24]=min(min(High1[1], Medium[2]),High1[3]);       //m


yout24[2]= y[24];

}


 else;
//*********************************************************************************/

 for(i=0;i<N;i++)
{
                yout25[i]=0;

                }

 if (mf1[3]=='h' && mf2[3]=='h' && mf3[1]=='l')                  //25

                 {

y[25]=min(min(High1[1], High1[2]),Low[3]);       //VH


yout25[4]= y[25];
}


 else;
//*********************************************************************************/
for(i=0;i<N;i++)
{
```

```
                yout26[i]=0;

                }

 if (mf1[3]=='h' && mf2[3]=='h' && mf3[2]=='m')                //26

                 {

y[26]=min(min(High1[1],High1[2]),Medium[3]);       //H


yout26[3]= y[26];
}


 else;
//*****************************************************************************/

 for(i=0;i<N;i++)
{
                yout27[i]=0;

                }

 if (mf1[3]=='h' && mf2[3]=='h' && mf3[3]=='h')           //27

                 {

y[27]=min(min(High1[1],High1[2]),High1[3]);       //l


yout27[1]= y[27];
}


 else;
 //*********************************AGGREDATION*******************************//

for(i=0;i<N;i++)
                {

  yout_1= max(max(max(max(max(max(max(max(max(yout1[i],yout2[i]),yout3[i]),yout4[i]),yout5[i]),yout6[i]),yout7[i]),yout8[i]),yout9[i]),yout10[i]);
  yout_2=
max(max(max(max(max(max(max(max(max(yout_1,yout11[i]),yout12[i]),yout13[i]),yout14[i]),yout15[i]),yout16[i]),yout17[i]),yout18[i]),yout19[i]);
  yout[i]= max(max(max(max(max(max(max(max(yout_2,yout20[i]),yout21[i]),yout22[i]),yout23[i]),yout24[i]),yout25[i]),yout26[i]),yout27[i]);


  }


 /***********************DEFUZZIFICATION**********************************/

 totalsum=0;multiplesum=0;

                for (i=0;i<N;i++)
{
totalsum=totalsum+yout[i];

}
                a=0;
for (i=0;i<N;i++)
{
x[i]=a;
a+=0.25;
};

for (i=0; i<N;i++)
{
g = yout[i]*x[i];
multiplesum = multiplesum + g;

}

output = multiplesum/totalsum;
PQ=output;

/*********************************************************/

iciptr=op_pk_ici_get(pkptr);

op_ici_attr_set (iciptr, "Tos", output);// Setting packet priority index
```

```
//op_ici_attr_get(iciptr, "Tos",&PI);

PQ=output;

if(PQ >= 0.0 && PQ <= 0.33)

{

queue_id=1;


    /* Attempt to place new packet in pending queue */
                              if (oms_buffer_bgutil_enqueue (routing_buffer1, pkptr)
                                             != OmsC_Buffer_Enqueue_Success)
                                      {


                                      /*            The insertion failed (due to a full buffer).            */
                                      char intf_addr_str [IPC_ADDR_STR_LEN];

                                      /*  Get a printable version of the interface addr.     */
                                      if (rcvd_iface_info_ptr == OPC_NIL)

                                             sprintf (intf_addr_str, "Higher Layer");
                                      else

                                             ip_address_print (intf_addr_str,
                                                    rcvd_iface_info_ptr->addr_range_ptr->address);

                                      /*            Issue a warning message to the sim. log.                    */
                                      ipnl_reswarn_pktinsert (op_pk_id (pkptr),
                                             op_pk_tree_id (pkptr), intf_addr_str);
                                      /* Update packets dropped statistics and destroy     */
                                      /* the IP datagram.
              */
                                      ip_rte_dgram_discard (module_data_ptr, pkptr, op_pk_ici_get (pkptr), "Buffer overflow");
                                      }


}
              else if (PQ > 0.33 && PQ<=0.66)
                              {
                                      queue_id=2;

                              /* Attempt to place new packet in pending queue */

                                      if (oms_buffer_bgutil_enqueue (routing_buffer2, pkptr)
                                             != OmsC_Buffer_Enqueue_Success)
                                      {

                                      /*            The insertion failed (due to a full buffer).            */
                                      char intf_addr_str [IPC_ADDR_STR_LEN];

                                      /*  Get a printable version of the interface addr.     */
                                      if (rcvd_iface_info_ptr == OPC_NIL)

                                             sprintf (intf_addr_str, "Higher Layer");
                                      else

                                             ip_address_print (intf_addr_str,
                                                    rcvd_iface_info_ptr->addr_range_ptr->address);

                                      /*            Issue a warning message to the sim. log.                    */
                                      ipnl_reswarn_pktinsert (op_pk_id (pkptr),
                                             op_pk_tree_id (pkptr), intf_addr_str);

                                      /* Update packets dropped statistics and destroy     */
                                      /* the IP datagram.
              */
                                      ip_rte_dgram_discard (module_data_ptr, pkptr, op_pk_ici_get (pkptr), "Buffer overflow");
                                      }

                                      }


              else {   //
                           queue_id=3;

                              /* Attempt to place new packet in pending queue */
                              if (oms_buffer_bgutil_enqueue (routing_buffer3, pkptr)
                                             != OmsC_Buffer_Enqueue_Success)
                                      {
```

```
                              /*          The insertion failed (due to a full buffer).          */
                              char intf_addr_str [IPC_ADDR_STR_LEN];

                              /*  Get a printable version of the interface addr.      */
                              if (rcvd_iface_info_ptr == OPC_NIL)

                                        sprintf (intf_addr_str, "Higher Layer");
                              else

                                        ip_address_print (intf_addr_str,
                                                  rcvd_iface_info_ptr->addr_range_ptr->address);

                              /*          Issue a warning message to the sim. log.                    */
                              ipnl_reswarn_pktinsert (op_pk_id (pkptr),
                                        op_pk_tree_id (pkptr), intf_addr_str);

                              /* Update packets dropped statistics and destroy      */
                              /* the IP datagram.
          */
                              ip_rte_dgram_discard (module_data_ptr, pkptr, op_pk_ici_get (pkptr), "Buffer overflow");
                              }
}//

          /*****************************************Alex Egaji***************************************/

                    }

          FOUT;

          }


static void
ip_rte_central_cpu_start_processing (void)
          {
          Packet *       pkptr;
          Ici *                         pk_ici_ptr;
          double                        comp_decomp_delay = 0.0;
          double                        pk_svc_time = 0.0;
          double                        bgutil_packets = 0.0;
          double                        bgutil_bytes = 0.0;
          double      bgutil_delay = 0.0;
          IpT_Rte_Ind_Ici_Fields *                intf_ici_fdstruct_ptr;
          IpT_Rte_Module_Data*                    iprmd_ptr;

          /** Initiate the processing of a received packet        **/
          FIN (ip_rte_central_cpu_start_processing (void));

          /* Get a handle to the packet at the top of the buffer.This           */
          /* is the packet whose service is about to start. The                 */
          /* packet is in the central processor's buffer.                                 */


          /***************************************Alex Egaji*****************************************/


if (oms_buffer_bgutil_num_packets_get (routing_buffer1, OPC_NIL) > 0.0){

          pkptr = oms_buffer_bgutil_access (routing_buffer1, 0, OPC_NIL,
                    &bgutil_bytes, &bgutil_packets, &bgutil_delay);



          if (pkptr == OPC_NIL)
                    {
                    ip_rte_cpu_error ("Unable to get packet from head of central buffer.");
                    }


          /* Retrive the ICI associated with the packet */
          pk_ici_ptr = op_pk_ici_get (pkptr);

          /*          Get the rte_info_fields from the ici.     */
          op_ici_attr_get (pk_ici_ptr, "rte_info_fields", &intf_ici_fdstruct_ptr);

          /*          Determine the time required to complete service of the             */
          /*          enqueued packet, including any bgutil in front of it.               */
          if (intf_ici_fdstruct_ptr->packet_is_labeled)
                    {
                    /* This is a labeled packet and the service time is    */
                    /* based on the packet forwarding rate                                */
                    pk_svc_time = one_over_forwarding_rate + bgutil_delay;
                    }
        else if (service_rate_pps)
                    {
                    /* rate in packets per second */
                    pk_svc_time = one_over_service_rate + bgutil_delay + 0.000165;

                    }
        else
                    {
                    /* rate in bits per second */
                    pk_svc_time = (one_over_service_rate * op_pk_total_size_get (pkptr)) + bgutil_delay;

                    }

        /* Retrieve the decompression and/or compression delay                  */
        /* associated with the packet. This delay will be added to         */
        /* the service time when scheduling the end of the service. */
        /* If the packet will not be decompressed or compressed at        */
        /* this processing, then the value of the delay is zero.             */
        comp_decomp_delay = intf_ici_fdstruct_ptr->comp_decomp_delay;

        /* Reset the attribute.                                                                                  */
        if (comp_decomp_delay)
                    {
                    intf_ici_fdstruct_ptr->comp_decomp_delay = 0.0;
                    }

        /* If the node is configured as a firewall, there can be              */
        /* also a proxy server delay associated with the datagram.         */
        /* Retrieve this delay, too.                                                                     */
        if (ip_node_is_firewall (module_data_ptr))
                    {
                    /* We can safely add the proxy delay to the compression          */
                    /* and decompression delay, since all of them are                 */
                    /* additional delays besides the routing delay and they           */
                    /* all will be summed up when the service completion             */
                    /* is scheduled.
        */
                    comp_decomp_delay += intf_ici_fdstruct_ptr->proxy_delay;

                    /* Reset the ICI field.                                                                  */
                    intf_ici_fdstruct_ptr->proxy_delay = 0.0;
                    }

        /* If the node is tunneling packets, there can also    be a vpn*/
        /* tunnel delay associated with the datagram. Retrieve this */
        /* delay, too.
                    */

        /* State variables cannot be passed to macros.        */
        iprmd_ptr = module_data_ptr;
        if (ip_l2tp_vpn_is_enabled (iprmd_ptr))
                    {
                    /* The vpn delay can be added to the compression and    */
                    /* decompression delay the same way as proxy delay.                      */
                    comp_decomp_delay += intf_ici_fdstruct_ptr->vpn_delay;

                    /* Reset the ICI field.                                                                  */
                    intf_ici_fdstruct_ptr->vpn_delay = 0.0;
                    }

        /* Use the resource management package to let it decide                  */
        /* how much CPU time it takes to do the requested                                */
        /* processing. End of processing is indicated with a                     */
        /* process interrupt.                                                                                    */
        /* Since we are dealing with one packet at a time, no need          */
        /* to provide any structure.                                                                      */
        job_id = Oms_Resource_Job_Insert (cpu_resource_handle, own_prohandle,
                    OPC_NIL, pk_svc_time + comp_decomp_delay);

        if (Oms_Resource_Job_Id_Equal (job_id, OMSC_JOB_ID_INVALID))
                    {
                    ip_rte_cpu_warn ("Unable to schedule service completion.");
                    }
        else
                    {
                    /*              Mark the server to be busy now.                                  */
                    cpu_is_idle = OPC_FALSE;
        }
        }

else if (oms_buffer_bgutil_num_packets_get (routing_buffer2, OPC_NIL) > 0.0){
			pkptr = oms_buffer_bgutil_access (routing_buffer2, 0, OPC_NIL,
			&bgutil_bytes, &bgutil_packets, &bgutil_delay);


		if (pkptr == OPC_NIL)
			{
			ip_rte_cpu_error ("Unable to get packet from head of central buffer.");
			}


		/* Retrive the ICI associated with the packet */
		pk_ici_ptr = op_pk_ici_get (pkptr);

		/*	Get the rte_info_fields from the ici.	*/
		op_ici_attr_get (pk_ici_ptr, "rte_info_fields", &intf_ici_fdstruct_ptr);

		/*	Determine the time required to complete service of the		*/
		/*	enqueued packet, including any bgutil in front of it.		*/
		if (intf_ici_fdstruct_ptr->packet_is_labeled)
			{
			/* This is a labeled packet and the service time is	*/
			/* based on the packet forwarding rate						*/
			pk_svc_time = one_over_forwarding_rate + bgutil_delay;


			}
		else if (service_rate_pps)
			{
			/* rate in packets per second */
			pk_svc_time = one_over_service_rate + bgutil_delay + 0.000165;


			}
		else
			{
			/* rate in bits per second */
			pk_svc_time = (one_over_service_rate * op_pk_total_size_get (pkptr)) + bgutil_delay;


			}

		/* Retrieve the decompression and/or compression delay		*/
		/* associated with the packet. This delay will be added to	*/
		/* the service time when scheduling the end of the service. */
		/* If the packet will not be decompressed or compressed at	*/
		/* this processing, then the value of the delay is zero.	*/
		comp_decomp_delay = intf_ici_fdstruct_ptr->comp_decomp_delay;

		/* Reset the attribute.									*/
		if (comp_decomp_delay)
			{
			intf_ici_fdstruct_ptr->comp_decomp_delay = 0.0;
			}

		/* If the node is configured as a firewall, there can be		*/
		/* also a proxy server delay associated with the datagram.	*/
		/* Retrieve this delay, too.								*/
		if (ip_node_is_firewall (module_data_ptr))
			{
			/* We can safely add the proxy delay to the compression		*/
			/* and decompression delay, since all of them are			*/
			/* additional delays besides the routing delay and they		*/
			/* all will be summed up when the service completion		*/
			/* is scheduled.
		*/
			comp_decomp_delay += intf_ici_fdstruct_ptr->proxy_delay;

			/* Reset the ICI field.								*/
			intf_ici_fdstruct_ptr->proxy_delay = 0.0;
			}

		/* If the node is tunneling packets, there can also	be a vpn*/
		/* tunnel delay associated with the datagram. Retrieve this */
		/* delay, too.
			*/

		/* State variables cannot be passed to macros.	*/
		iprmd_ptr = module_data_ptr;
		if (ip_l2tp_vpn_is_enabled (iprmd_ptr))
			{
			/* The vpn delay can be added to the compression and	*/
			/* decompression delay the same way as proxy delay.			*/
```

```
                    comp_decomp_delay += intf_ici_fdstruct_ptr->vpn_delay;

                    /* Reset the ICI field.                                                  */
                    intf_ici_fdstruct_ptr->vpn_delay = 0.0;
                    }

          /* Use the resource management package to let it decide                  */
          /* how much CPU time it takes to do the requested                                  */
          /* processing. End of processing is indicated with a                     */
          /* process interrupt.                                                              */
          /* Since we are dealing with one packet at a time, no need     */
          /* to provide any structure.                                                      */
          job_id = Oms_Resource_Job_Insert (cpu_resource_handle, own_prohandle,
                    OPC_NIL, pk_svc_time + comp_decomp_delay);

          if (Oms_Resource_Job_Id_Equal (job_id, OMSC_JOB_ID_INVALID))
                    {
                    ip_rte_cpu_warn ("Unable to schedule service completion.");
                    }
          else
                    {
                    /*          Mark the server to be busy now.                                    */
                    cpu_is_idle = OPC_FALSE;
          }

          }
else if (oms_buffer_bgutil_num_packets_get (routing_buffer3, OPC_NIL) > 0.0){

          pkptr = oms_buffer_bgutil_access (routing_buffer3, 0, OPC_NIL,
                    &bgutil_bytes, &bgutil_packets, &bgutil_delay);


          if (pkptr == OPC_NIL)
                    {
                    ip_rte_cpu_error ("Unable to get packet from head of central buffer.");
                    }


          /* Retrive the ICI associated with the packet */
          pk_ici_ptr = op_pk_ici_get (pkptr);

          /*        Get the rte_info_fields from the ici.     */
          op_ici_attr_get (pk_ici_ptr, "rte_info_fields", &intf_ici_fdstruct_ptr);

          /*        Determine the time required to complete service of the          */
          /*        enqueued packet, including any bgutil in front of it.           */
          if (intf_ici_fdstruct_ptr->packet_is_labeled)
                    {
                    /* This is a labeled packet and the service time is   */
                    /* based on the packet forwarding rate                               */
                    pk_svc_time = one_over_forwarding_rate + bgutil_delay;


                    }
          else if (service_rate_pps)
                    {
                    /* rate in packets per second */
                    pk_svc_time = one_over_service_rate + bgutil_delay;


                    }
          else
                    {
                    /* rate in bits per second */
                    pk_svc_time = (one_over_service_rate * op_pk_total_size_get (pkptr)) + bgutil_delay;


                    }

          /* Retrieve the decompression and/or compression delay                   */
          /* associated with the packet. This delay will be added to      */
          /* the service time when scheduling the end of the service. */
          /* If the packet will not be decompressed or compressed at      */
          /* this processing, then the value of the delay is zero.        */
          comp_decomp_delay = intf_ici_fdstruct_ptr->comp_decomp_delay;

          /* Reset the attribute.                                                            */
          if (comp_decomp_delay)
                    {
                    intf_ici_fdstruct_ptr->comp_decomp_delay = 0.0;
                    }

          /* If the node is configured as a firewall, there can be        */
```

```
		/* also a proxy server delay associated with the datagram.	*/
		/* Retrieve this delay, too.					*/
		if (ip_node_is_firewall (module_data_ptr))
			{
			/* We can safely add the proxy delay to the compression	*/
			/* and decompression delay, since all of them are		*/
			/* additional delays besides the routing delay and they	*/
			/* all will be summed up when the service completion	*/
			/* is scheduled.
		*/
			comp_decomp_delay += intf_ici_fdstruct_ptr->proxy_delay;

			/* Reset the ICI field.						*/
			intf_ici_fdstruct_ptr->proxy_delay = 0.0;
			}

		/* If the node is tunneling packets, there can also	be a vpn*/
		/* tunnel delay associated with the datagram. Retrieve this */
		/* delay, too.
			*/

		/* State variables cannot be passed to macros.	*/
		iprmd_ptr = module_data_ptr;
		if (ip_l2tp_vpn_is_enabled (iprmd_ptr))
			{
			/* The vpn delay can be added to the compression and	*/
			/* decompression delay the same way as proxy delay.		*/
			comp_decomp_delay += intf_ici_fdstruct_ptr->vpn_delay;

			/* Reset the ICI field.						*/
			intf_ici_fdstruct_ptr->vpn_delay = 0.0;
			}

		/* Use the resource management package to let it decide		*/
		/* how much CPU time it takes to do the requested			*/
		/* processing. End of processing is indicated with a		*/
		/* process interrupt.							*/
		/* Since we are dealing with one packet at a time, no need	*/
		/* to provide any structure.						*/
		job_id = Oms_Resource_Job_Insert (cpu_resource_handle, own_prohandle,
			OPC_NIL, pk_svc_time + comp_decomp_delay);

		if (Oms_Resource_Job_Id_Equal (job_id, OMSC_JOB_ID_INVALID))
			{
			ip_rte_cpu_warn ("Unable to schedule service completion.");
			}
		else
			{
			/*		Mark the server to be busy now.				*/
			cpu_is_idle = OPC_FALSE;
		}
		}

		FOUT;

		}


static void
ip_rte_central_cpu_send_packet (Packet* pkptr)
		{
		OpT_Packet_Size				packet_size;
		Ici						*pk_ici_ptr;
		IpT_Dgram_Fields*			pk_fd_ptr;
		IpT_Rte_Ind_Ici_Fields  *intf_ici_fdstruct_ptr;
		double size1;

		/** Send processed packet to appropriate level	**/
		FIN (ip_rte_central_cpu_send_packet (pkptr));

		/* Unless the processing speed is set to infinity,	*/
		/* we need to extract the packet from the buffer.	*/


		if (! infinite_processing_rate)
			{

/***************************************Alex Egaji*******************************************/


if (oms_buffer_bgutil_num_packets_get (routing_buffer1, OPC_NIL) > 0.0){

		/* Destroy job ID. Resource package does not		*/
```

```
                    /* destroy the memory allocated to store this info.   */
                    /* if there is no ici expected , the job id is      */
                    /* destroyed by resource package                                                    */
                    op_prg_mem_free (job_id);

                    /* Block outgoing background traffic during the service of this packet. */
                    oms_buffer_bgutil_outgoing_flow_control(routing_buffer1, OPC_FALSE);

                    /* Extract packet at head of queue - this is the packet just           */
                    /* finishing service.
                    */
                    pkptr = oms_buffer_bgutil_dequeue (routing_buffer1);

                    /* Reset the outgoing background traffic flag. */
                    oms_buffer_bgutil_outgoing_flow_control (routing_buffer1, OPC_TRUE);



                    if (pkptr == OPC_NIL)
                              ip_rte_cpu_error ("Unable to get packet from head of central buffer.");

                    /* The central cpu becomes available again.           */
                    cpu_is_idle = OPC_TRUE;


}

else if (oms_buffer_bgutil_num_packets_get (routing_buffer2, OPC_NIL) > 0.0){


          /* Destroy job ID. Resource package does not                        */
                    /* destroy the memory allocated to store this info.   */
                    /* if there is no ici expected , the job id is      */
                    /* destroyed by resource package                                                    */
                    op_prg_mem_free (job_id);

                    /* Block outgoing background traffic during the service of this packet. */
                    oms_buffer_bgutil_outgoing_flow_control(routing_buffer2, OPC_FALSE);

                    /* Extract packet at head of queue - this is the packet just           */
                    /* finishing service.
                    */
                    pkptr = oms_buffer_bgutil_dequeue (routing_buffer2);

                    /* Reset the outgoing background traffic flag. */
                    oms_buffer_bgutil_outgoing_flow_control (routing_buffer2, OPC_TRUE);



                    if (pkptr == OPC_NIL)
                              ip_rte_cpu_error ("Unable to get packet from head of central buffer.");

                    /* The central cpu becomes available again.           */
                    cpu_is_idle = OPC_TRUE;


}


else{

          /* Destroy job ID. Resource package does not                        */
                    /* destroy the memory allocated to store this info.   */
                    /* if there is no ici expected , the job id is      */
                    /* destroyed by resource package                                                    */
                    op_prg_mem_free (job_id);

                    /* Block outgoing background traffic during the service of this packet. */
                    oms_buffer_bgutil_outgoing_flow_control(routing_buffer3, OPC_FALSE);

                    /* Extract packet at head of queue - this is the packet just           */
                    /* finishing service.
                    */
                    pkptr = oms_buffer_bgutil_dequeue (routing_buffer3);

                    /* Reset the outgoing background traffic flag. */
                    oms_buffer_bgutil_outgoing_flow_control (routing_buffer3, OPC_TRUE);


                    if (pkptr == OPC_NIL)
                              ip_rte_cpu_error ("Unable to get packet from head of central buffer.");

                    /* The central cpu becomes available again.           */
```

```
                        cpu_is_idle = OPC_TRUE;

}
    }
            /****************************************Alex Egaji*******************************************/

            /* Retrieve the ICI associated with the packet.        */
            pk_ici_ptr = op_pk_ici_get (pkptr);

            /*          Get the data structure stored in the "rte_info_fields" field.        */
            if (op_ici_attr_exists (pk_ici_ptr, "rte_info_fields") == OPC_TRUE)
                        {
                        op_ici_attr_get (pk_ici_ptr, "rte_info_fields", &intf_ici_fdstruct_ptr);
                        }
            else
                        {
                        ip_rte_cpu_error ("Unable to obtain routing information from packet.");
                        }

            /* Get a handle to the data structure stored in "fields" field. */
            /* This structure contains all the protocol header information     */
            /* as well as some internal information attached to each IP        */
            /* datagram for simulation efficiency.                                                  */
            op_pk_nfd_access (pkptr, "fields", &pk_fd_ptr);

            /* Decompress the packet if necessary.                 */
            ip_rte_decompress (pkptr, intf_ici_fdstruct_ptr, pk_fd_ptr);

            /* Update the packet size. */
            packet_size = op_pk_total_size_get (pkptr);

            /* Compress datagram if specified */
            module_data_ptr->dgram_compressed_size = ip_rte_compress (
                        module_data_ptr, pkptr, packet_size, intf_ici_fdstruct_ptr, pk_fd_ptr);

            /* Call the function that would forward the packet appropriately*/
            ip_rte_packet_send (module_data_ptr, pkptr, pk_ici_ptr,
                        intf_ici_fdstruct_ptr, IpC_Rte_Process_Type_Central_Cpu, OPC_NIL);

            FOUT;

            }


/* Other local routines */

static void
ip_rte_cpu_error (const char *msg)
            {
            /** Print an error message and exit the simulation. **/
            FIN (ip_rte_cpu_error (msg));

            op_sim_end (
                        "Error in IP central forwarding process model (ip_rte_central_cpu):",
                        msg,
                        "Check simulation log messages from the IP model for more information.",
                        "Simulation Logging can be enabled/disabled in the Simulation Editor.");
            FOUT;

            }

static void
ip_rte_cpu_warn (const char *msg)
            {
            /** Print a warning message and resume. **/
            FIN (ip_rte_cpu_warn (msg));

            op_sim_message ("Warning from IP central forwarding process model (ip_rte_central_cpu):", msg);

            FOUT;

            }

static void
ip_rte_routing_processor_info_init (void)
            {
            Objid                         ip_proc_info_objid;
            Objid                         compound_objid;
            Compcode                      return_status;
            double                                  buffer_size;
            OmsT_Buffer_Create      status;
            OmsT_Buffer_Stats       stat_array;
            int                                               i;
```

```
	double						service_rate;
	double						forwarding_rate;

	const char * stat_names [OmsC_Buffer_Stat_Number] =
			{
			"IP Processor.Forwarding Memory Overflows",
			"IP Processor.Forwarding Memory Free Size (bytes)",
			"", /* ignored */
			"IP Processor.Forwarding Memory Queue Size (bytes)",
			"IP Processor.Forwarding Memory Queue Size (packets)",
			"IP Processor.Forwarding Memory Queue Size (incoming bytes)",
			"IP Processor.Forwarding Memory Queue Size (incoming packets)",
			"IP Processor.Forwarding Memory Queuing Delay",
			"", /* delay variation ignored */
			};

	/** Extract routing processor information and perform appropriate initializtions **/
	FIN (ip_rte_routing_processor_info_init ());

	return_status = op_ima_obj_attr_get (own_id,
			"ip processing information", &ip_proc_info_objid);
	if (return_status == OPC_COMPCODE_FAILURE)
			ip_rte_cpu_error ("Unable to get forwarding information from attribute.");

	compound_objid = op_topo_child (ip_proc_info_objid, OPC_OBJTYPE_GENERIC, 0);

	/*		Get assigned value of server processing rate.				*/
	return_status = op_ima_obj_attr_get (compound_objid,
			"Datagram Forwarding Rate", &service_rate);
	if (return_status == OPC_COMPCODE_FAILURE)
			ip_rte_cpu_error ("Unable to get server datagram forwarding rate from attribute.");

	/* If the service rate is set to infinity, we don't need*/
	/* to create a buffer to hold the packets while they are		*/
	/* being serviced.
	*/
	if (IPC_EFFICIENCY_MODE_FORWARDING_RATE == service_rate)
			{
			/* Set the flag indicating that the processing speed is		*/
			/* infinite.
	*/
			infinite_processing_rate = OPC_TRUE;

			/* Write a log message warning the user that CPU stats		*/
			/* will not be recorded because the processing speed		*/
			/* is set to infinity.								*/
			if (OPC_FALSE == infinite_processing_log_written)
					{
					/* The log message has not been written already.	*/
					ipnl_cpu_stats_not_written_log_write ();

					/* Do not write the log message again.			*/
					infinite_processing_log_written = OPC_TRUE;
					}

			FOUT;
			}

	/* Finite processing speed.								*/
	infinite_processing_rate = OPC_FALSE;

	/* Store 1/service rate to avoid doing division each time */
	one_over_service_rate = 1.0 / service_rate;
module_data_ptr->service_rate = service_rate;

	/* MPLS forwarding rate							*/
	return_status = op_ima_obj_attr_get (compound_objid,
					"Datagram Switching Rate", &forwarding_rate);
	if (return_status == OPC_COMPCODE_FAILURE)
			ip_rte_cpu_error ("Unable to get Datagram Switching rate from the attribute.");
	one_over_forwarding_rate = 1.0/forwarding_rate;

	return_status = op_ima_obj_attr_get (compound_objid,
			"Forwarding Rate Units", &service_rate_pps);
	if (return_status == OPC_COMPCODE_FAILURE)
			ip_rte_cpu_error ("Unable to get units of forwarding rate from attribute.");

	/* Find out routing processing buffer size and create corresponding buffer	*/
	return_status = op_ima_obj_attr_get (compound_objid,
			"Memory Size", &buffer_size);
	if (return_status == OPC_COMPCODE_FAILURE)
			ip_rte_cpu_error ("Unable to get forwarding memory size from attribute.");
```

```
/* The buffer pool will be used for the routing queue, as well      */
/* as the QoS buffers, if no slots are used.                                                    */
/* The pool size is based on the routing processor memory size  */
/* The pool does not have any size limitations.        */
/* The various buffers will have their own.                          */

//routing_buffer_pool


routing_buffer_pool = oms_buffer_bgutil_pool_create (
        buffer_size,                              /* number of bytes */
        OmsC_Buffer_Infinite,    /* number of packets */
        OPC_NIL, "Routing Processor Pool", OPC_NIL, &status);

if (routing_buffer_pool == OPC_NIL)
        {
        /* Something went wrong, report and terminate */
        switch (status)
                {
                case OmsC_Buffer_Create_Invalid_Byte_Size:
                        op_sim_end (
                                "Unable to create IP shared memory pool due to an invalid byte size",
                                "", "", "");
                        break;
                case OmsC_Buffer_Create_Invalid_Packet_Size:
                        op_sim_end (
                                "Unable to create IP shared memory pool due to an invalid packet size",
                                "", "", "");
                        break;
                case OmsC_Buffer_Create_Memory_Alloc_Failed:
                        op_sim_end (
                                "Unable to create IP shared memory pool due to an internal memory allocation failure",
                                "", "", "");
                        break;
                }
        }

/* All stats but free size in packets (since no upper limit) */
for (i = 0; i < OmsC_Buffer_Stat_Number; i++)
        {
        if (stat_names [i][0] == '\0')
        stat_array [i].used = OPC_FALSE;
        else
                {
        stat_array [i].used = OPC_TRUE;
        stat_array [i].shandle = op_stat_reg (stat_names [i],
                        OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);

                }
        }
/******************************************ALEX EGAJI**************************************************/

routing_buffer1 = oms_buffer_bgutil_create (routing_buffer_pool,
        OmsC_Buffer_Parent_Limit, OmsC_Buffer_Parent_Limit,
        module_data_ptr->do_bgutil,
        service_rate, service_rate_pps,
        &stat_array, "Routing Processor Buffer", OPC_NIL, /* No user data */
        &status);
routing_buffer2 = oms_buffer_bgutil_create (routing_buffer_pool,
        OmsC_Buffer_Parent_Limit, OmsC_Buffer_Parent_Limit,
        module_data_ptr->do_bgutil,
        service_rate, service_rate_pps,
        &stat_array, "Routing Processor Buffer", OPC_NIL, /* No user data */
        &status);
routing_buffer3 = oms_buffer_bgutil_create (routing_buffer_pool,
        OmsC_Buffer_Parent_Limit, OmsC_Buffer_Parent_Limit,
        module_data_ptr->do_bgutil,
        service_rate, service_rate_pps,
        &stat_array, "Routing Processor Buffer", OPC_NIL, /* No user data */
        &status);

/******************************************ALEX EGAJI*************************************************/

if ((routing_buffer1 == OPC_NIL)||(routing_buffer2 == OPC_NIL)||(routing_buffer3 == OPC_NIL))
        ip_rte_cpu_error ("Unable to create routing processor memory buffer.");

FOUT;



}
```

Appendix E : Implementation of the Sugeno Fuzzy Scheduler

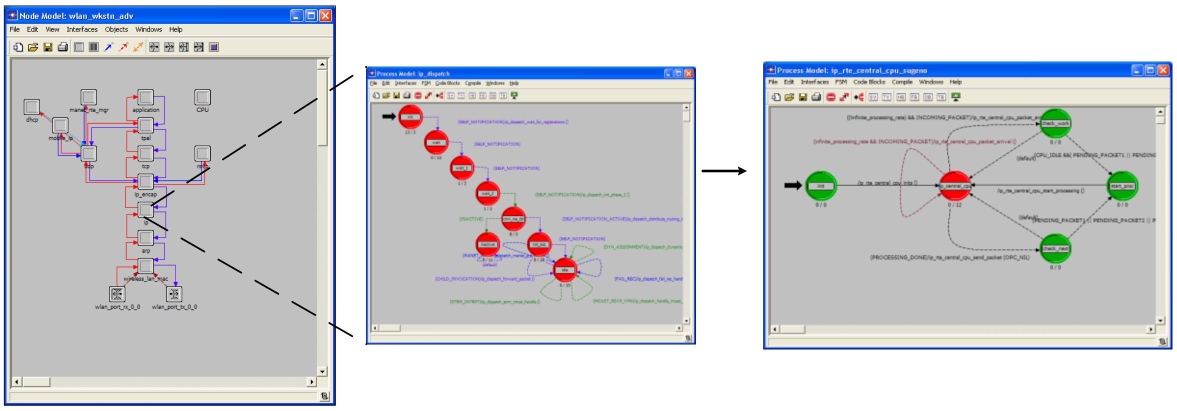


Figure E-7: Implementation of Sugeno Scheduler

```
/* Transition executives */

static void
ip_rte_central_cpu_inits (void)
        {
        IpT_Interface_Info *     iface_info_ptr;
        int                                          i;
        int                                          num_if;
        IpT_Group_Intf_Info*     group_info_ptr;
        int                                          ith_member, num_member_intfs;
        IpT_Member_Intf_Info*  member_intf_ptr;


        /** Perform appropriate initializations before reaching the wait state **/
        FIN (ip_rte_central_cpu_inits ());

        /* Obtain module memory and set up for ip_rte_int routines */
        module_data_ptr = (IpT_Rte_Module_Data *)op_pro_modmem_access ();
        ip_rte_set_procs (module_data_ptr, ip_rte_cpu_error, ip_rte_cpu_warn);

        own_id = op_id_self ();
        own_prohandle = op_pro_self ();

        /* Create pool */
        ip_rte_routing_processor_info_init ();

        /* Unless the processing speed is set to infinity, obtain      */
        /* CPU resource handle to schedule shared processing                */
        /* throughout the node.                                                                          */
        if (! infinite_processing_rate)
                {
                cpu_resource_handle = Oms_Resource_Handle_Get (module_data_ptr->node_id,
                        module_data_ptr->module_id, "CPU");
                cpu_is_idle = OPC_TRUE;
                }

        /* If QoS is in place, invoke interface process model to      */
        /* let it create its buffers using the central memory pool    */
        /* Also take control of any stream interrupts coming on            */
        /* each interface.
        */
        num_if = inet_rte_num_interfaces_get (module_data_ptr);
        for (i = 0; i < num_if; ++i)
```

```
                {
                iface_info_ptr = inet_rte_intf_tbl_access (module_data_ptr, i);

                /* Unless the stream index is invalid, register for all          */
                /* stream interrupts on that stream. Logical interfaces           */
                /* like loopback interfaces and tunnel interfaces                 */
                /* might have invalid stream indices.                                                    */

                /* In the case of interface groups, register for stream          */
                /* interrupts on the stream indices of all member                 */
                /* interfaces.
                */
                if (ip_rte_intf_is_group (iface_info_ptr))
                        {
                        /* Get a handle to structure that stores group                 */
                        /* related parameters.                                                                                       */
                        group_info_ptr = iface_info_ptr->phys_intf_info_ptr->group_info_ptr;

                        /* Get the number of member interfaces.                                        */
                        num_member_intfs = group_info_ptr->num_members;

                        /* Loop through each member interface                                          */
                        for (ith_member = 0; ith_member < num_member_intfs; ith_member++)
                                {
                                /* Get a handle to the ith member structure                    */
                                member_intf_ptr = &(group_info_ptr->member_intf_array[ith_member]);

                                /* Register for the stream interface.                          */
                                op_intrpt_port_register (OPC_PORT_TYPE_STRM,
                                        member_intf_ptr->instrm, own_prohandle);
                                }
                        }
                else if (ip_rte_intf_in_port_num_get (iface_info_ptr) != IPC_PORT_NUM_INVALID)
                        {
                        op_intrpt_port_register (OPC_PORT_TYPE_STRM,
                                ip_rte_intf_in_port_num_get (iface_info_ptr), own_prohandle);
                        }

                if (iface_info_ptr->queuing_scheme != IpC_No_Queuing)
                        {
                        op_pro_invoke (iface_info_ptr->output_iface_prohandle, routing_buffer_pool);
                        }
                }

        /* Take control of stream interrupts coming from the upper layers */
        op_intrpt_port_register (OPC_PORT_TYPE_STRM,
                module_data_ptr->instrm_from_ip_encap, own_prohandle);

        /* Check to see if there is the possibility of background utilization          */
        /* traffic.
                                                */
        if ((module_data_ptr->do_bgutil) && (! infinite_processing_rate))
                {
                /* Initialize the background utilization state ptr for the central cpu*/
                module_data_ptr->bgutil_routed_state_ptr = OPC_NIL;

                /* Initialize the background utilization state ptr for keeping track */
                /* of sent and received background utilization traffic.                                        */
                module_data_ptr->received_bgutil_routed_state_ptr = oms_bgutil_routed_state_create (
                        !service_rate_pps, DO_NOT_SCALE);
                module_data_ptr->sent_bgutil_routed_state_ptr = oms_bgutil_routed_state_create (
                        !service_rate_pps, DO_NOT_SCALE);

                /* Init the time at which background utilization statistics          */
                /* generation begins to time 0.
        */
                module_data_ptr->received_last_stat_update_time = 0.0;
                module_data_ptr->sent_last_stat_update_time = 0.0;

                /* Schedule a procedure which will be called for this process at the*/
                /* end of simulation to update background utilization statistics if           */
                /* needed.
                                                */
                op_intrpt_schedule_call (OPC_INTRPT_SCHED_CALL_ENDSIM, 0,
                        ip_rte_pk_stats_update_endsim, module_data_ptr);
                }

        FOUT;
        }

static void
ip_rte_central_cpu_packet_arrival (void)
        {
        Packet *                                                pkptr = OPC_NIL;
        int                                                     instrm;
```

```
double                      PQ,PI;
  Ici *         iciptr;
  IpT_Rte_Ind_Ici_Fields *                  intf_ici_fdstruct_ptr = OPC_NIL;
  IpT_Interface_Info *                      rcvd_iface_info_ptr = OPC_NIL;
  int                                                                    result;

  double Low[3],Medium[3],High1[3],y[27],x1,x2,x3;          //AE

  double b10,c10,a11,b11,c11,a12,b12,b20,c20,a21,b21,c21,a22,b22,c22,b30,c30,a31,b31,c31,a32,b32,c32; //AE
  int i;       //AE
  char mf1[3],mf2[3],mf3[3];

  double   x[N],a,totalsum, multiplesum,output,g;  //AE
  double  yout[N],yout_1,yout_2,yout1[N],yout2[N],yout3[N],yout4[N],yout5[N],yout6[N],yout7[N],yout8[N],yout9[N];
   double  yout10[N],yout11[N],yout12[N],yout13[N],yout14[N],yout15[N];//AE
  double  yout16[N],yout17[N],yout18[N],yout19[N],yout20[N],yout21[N],yout22[N],yout23[N],yout24[N],yout25[N] ;
   double   yout26[N],yout27[N];//AE




  /** An incoming packet has arrived.  It might         **/
  /** be from an "upper layer", a "lower layer",        **/
  /** or generated from within ip.                                          **/
  FIN (ip_rte_central_cpu_packet_arrival ());

  if (invoke_mode == OPC_PROINV_INDIRECT)
          {
          /* Packet generated from withing IP and forwarded by our        */
          /* parent process.
          */
          pkptr = (Packet *)op_pro_argmem_access ();
          instrm = IpC_Pk_Instrm_Child;
          }
  else
          {
          /* Packet coming from some stream */
          instrm = op_intrpt_strm ();
          pkptr = op_pk_get (instrm);
          if (pkptr == OPC_NIL)
                  ip_rte_cpu_error ("Unable to get packet from input stream.");
          }

  /* Perform standard IP processing of incoming packet              */
  /*             1. Perform forwarding decision and populate the    ICI           */
  /*             2. Populate rcvd_iface_info_ptr. It is set to NIL    */
  /*               if packet arrives from higher layer.                                  */
  result = ip_rte_packet_arrival (module_data_ptr,
          &pkptr, instrm, &intf_ici_fdstruct_ptr, &rcvd_iface_info_ptr);

  if (result == OPC_FALSE)
          {
          /* Packet was dropped in call */
          FOUT;
          }


  /* If the processing speed is set to infinity, forward */
  /* the packet immediately.                                                                                  */
  if (infinite_processing_rate)
          {

          ip_rte_central_cpu_send_packet (pkptr);
          }
  else
          {
          /* Finite processing speed.                                                                          */


          /**********************************Alex Egaji*************************************/

    /*x1=SNR(signal to noise ratio), x2= Data rate, x3= Queue */

  /********Fuzzification*********/

          if (x1>1)
          {
          x1=1;
          }


  else ;
  if (x2>1) // data rate
          {
                x2=1;
                }
        else ;
        if (x3>100)  //queue length
                {
                x3=100;
                }
        else ;



        b10 = 0; c10 = 0.4;     a11 = 0.1; b11 = 0.5; c11 = 0.9;        a12 = 0.6; b12 = 1; //SNR
        b20 = 0; c20 = 0.4;     a21 = 0.3; b21 = 0.5; c21 = 0.7;        a22 = 0.6; b22 = 0.8; c22=1;//trapoziod //Data rate
        b30 = 0; c30 = 10;     a31 = 8; b31 = 20; c31 = 30;        a32 = 22; b32 = 60; c32=100; //trapoziod //queue length



    //SNR/

        Low[1] = max((c10-x1)/(c10-b10),0);

        Medium[1] = max(min((x1-a11)/(b11-a11),(c11-x1)/(c11-b11)),0);

        High1[1] = max((x1-a12)/(b12-a12),0);


        if (Low[1]>0)
                mf1[1]='l';

        else mf1[1]='N';
        if (Medium[1]>0)
                mf1[2]='m';
        else mf1[2]='N';
        if (High1[1]>0)
                mf1[3]='h';
        else mf1[3]='N';

        //Data rate/

        Low[2] = max((c20-x2)/(c20-b20),0);

        Medium[2] = max(min((x2-a21)/(b21-a21),(c21-x2)/(c21-b21)),0);

        if (x2<0.8){
        High1[2] = max((x2-a22)/(b22-a22),0);
        }
        else {
                High1[2]=1;
                }


        if (Low[2]>0)
                mf2[1]='l';

        else mf2[1]='N';
        if (Medium[2]>0)
                mf2[2]='m';
        else mf2[2]='N';
        if (High1[2]>0)
                mf2[3]='h';
        else mf2[3]='N';


        //Queue length/

        Low[3] = max((c30-x3)/(c30-b30),0);

        Medium[3] = max(min((x3-a31)/(b31-a31),(c31-x3)/(c31-b31)),0);


        if (x3<60){
        High1[3] = max((x3-a32)/(b32-a32),0);
        }
        else {
                High1[3]=1;
                }


        if (Low[3]>0)
                mf3[1]='l';

        else mf3[1]='N';
        if (Medium[3]>0)
                mf3[2]='m';
        else mf3[2]='N';
```

```
                if (High1[3]>0)
                                mf3[3]='h';
                else mf3[3]='N';

/*****************Implication*****************************/

                for(i=0;i<N;i++)
{
                yout1[i]=0;

                }

if (mf1[1]=='l' && mf2[1]=='l' &&  mf3[1]=='l')          // 1
                                 {

y[1]= min( min(Low[1],Low[2]),Low[3]);    //m

yout1[2]= y[1];

}

 else;
//****************************************************************************************/
 for(i=0;i<N;i++)
{
                yout2[i]=0;

                }

if (mf1[1]=='l' && mf2[1]=='l' &&  mf3[2]=='m')          // 2
                                 {

y[2]= min( min(Low[1],Low[2]),Medium[3]);    //m

yout2[2]= y[2];

}

 else ;
//****************************************************************************************/
 for(i=0;i<N;i++)
{
                yout3[i]=0;

                }
if (mf1[1]=='l' && mf2[1]=='l' &&  mf3[3]=='h')          // 3
                                 {

y[3]= min( min(Low[1],Low[2]),High1[3]);    //vl

yout3[0]= y[3];
}

 else ;
//****************************************************************************************/
 for(i=0;i<N;i++)
{
                yout4[i]=0;

                }
if (mf1[1]=='l' && mf2[2]=='m' &&  mf3[1]=='l')          // 4
                                 {

y[4]= min( min(Low[1],Medium[2]),Low[3]);    //m


yout4[2]=y[4];

}

 else ;
//****************************************************************************************/
for(i=0;i<N;i++)
{
                yout5[i]=0;

                }

 if (mf1[1]=='l' && mf2[2]=='m' && mf3[2]=='m')      // 5

                 {

y[5]=min(min(Low[1],Medium[2]), Medium[3]);       //l
```

```
yout5[1]= y[5];

}

 else ;
//*****************************************************************************************/
for(i=0;i<N;i++)
{
                yout6[i]=0;

                }

 if (mf1[1]=='l' && mf2[2]=='m' && mf3[3]=='h')      // 6

                 {
                 for(i=0;i<N;i++)
{
                yout6[i]=0;

                }

y[6]=min(min(Low[1],Medium[2]), High1[3]);       //vl


yout6[0]= y[6];

}
 else ;
//*********************************************************************************/
for(i=0;i<N;i++)
{
                yout7[i]=0;

                }

 if (mf1[1]=='l' && mf2[3]=='h' && mf3[1]=='l')        // 7

                {

y[7]=min(min(Low[1],High1[2]),Low[3]);      //m


yout7[2]= y[7];

}

 else ;
//***************************************************************************/
for(i=0;i<N;i++)
{
                yout8[i]=0;

                }
 if (mf1[1]=='l' && mf2[3]=='h' && mf3[2]=='m')        // 8

                {

y[8]=min(min(Low[1],High1[2]),Medium[3]);      //m

yout8[2]= y[8];

}

 else ;
//*****************************************************************************/
for(i=0;i<N;i++)
{
                yout9[i]=0;

                }

 if (mf1[1]=='l' && mf2[3]=='h' && mf3[3]=='h')        // 9

                {

y[9]=min(min(Low[1],High1[2]),High1[3]);      //vl

yout9[0]= y[9];

}

 else ;
//*****************************************************************************/
for(i=0;i<N;i++)
{
```

```
                yout10[i]=0;

                }
 if (mf1[2]=='m' && mf2[1]=='l' && mf3[1]=='l')                    // 10

                {

y[10]= min(min(Medium[1],Low[2]),Low[3]);                    //h

yout10[3]= y[10];
}

 else ;
//*******************************************************************************/
for(i=0;i<N;i++)
{
                yout11[i]=0;

                }

 if (mf1[2]=='m' && mf2[1]=='l' && mf3[2]=='m')                    //11

                 {

y[11] = min(min(Medium[1],Low[2]), Medium[3]) ;           //m

yout11[2]= y[11];

}

 else;
//*********************************************************************************/
for(i=0;i<N;i++)
{
                yout12[i]=0;

                }

 if (mf1[2]=='m' && mf2[1]=='l' && mf3[3]=='h')                 //12

                 {

y[12]=min(min(Medium[1], Low[2]),High1[3]);      //l

yout12[1]=y[12];

}

 else;
//*********************************************************************************/
for(i=0;i<N;i++)
{
                yout13[i]=0;

                }

 if (mf1[2]=='m' && mf2[2]=='m' && mf3[1]=='l')                 //13

                 {

y[13]=min(min(Medium[1], Medium[2]),Low[3]);      //h

yout13[3]= y[13];

}

 else;
//*****************************************************************************/
for(i=0;i<N;i++)
{
                yout14[i]=0;

                }

 if (mf1[2]=='m' && mf2[2]=='m' && mf3[2]=='m')                  //14

                 {

y[14]=min(min(Medium[1], Medium[2]),Medium[3]);      //m

yout14[2]=y[14];

}

 else;
```

```
//******************************************************************************/
for(i=0;i<N;i++)
{
            yout15[i]=0;

            }
 if (mf1[2]=='m' && mf2[2]=='m' && mf3[3]=='h')             //15

             {

y[15]=min(min(Medium[1], Medium[2]),High1[3]);      //l


yout15[1]= y[15];
}

 else;
//*********************************************************************************/
for(i=0;i<N;i++)
{
            yout16[i]=0;

            }

 if (mf1[2]=='m' && mf2[3]=='h' && mf3[1]=='l')             //16

             {

y[16]=min(min(Medium[1], High1[2]),Low[3]);      //h


yout16[3]= y[16];

}

 else;
//**********************************************************************************/
for(i=0;i<N;i++)
{
            yout17[i]=0;

            }

 if (mf1[2]=='m' && mf2[3]=='h' && mf3[2]=='m')             //17

             {

y[17]=min(min(Medium[1], High1[2]),Medium[3]);      //m

yout17[2]= y[17];

}

 else;
//***************************************************************************/

 for(i=0;i<N;i++)
{
            yout18[i]=0;

            }

 if (mf1[2]=='m' && mf2[3]=='h' && mf3[3]=='h')             //18

             {

y[18]=min(min(Medium[1],High1[2]),High1[3]);      //l

yout18[1]= y[18];

}

 else;
//****************************************************************************/

 for(i=0;i<N;i++)
{
            yout19[i]=0;

            }

 if (mf1[3]=='h' && mf2[1]=='l' && mf3[1]=='l')             //19

             {
```

```
y[19]=min(min(High1[1], Low[2]),Low[3]);       //h

yout19[3]= y[19];

}

 else;
//****************************************************************************/
for(i=0;i<N;i++)
{
                yout20[i]=0;

                }

 if (mf1[3]=='h' && mf2[1]=='l' && mf3[2]=='m')                    //20

                 {

y[20]=min(min(High1[1], Low[2]),Medium[3]);       //m

yout20[2]= y[20];

}

 else;
//****************************************************************************/
for(i=0;i<N;i++)
{
                yout21[i]=0;

                }

 if (mf1[3]=='h' && mf2[1]=='l' && mf3[3]=='h')                    //21

                 {

y[21]=min(min(High1[1], Low[2]),High1[3]);       //l

yout21[1]= y[21];

}

 else;
//***********************************************************************/
for(i=0;i<N;i++)
{
                yout22[i]=0;

                }

 if (mf1[3]=='h' && mf2[2]=='m' && mf3[1]=='l')                    //22

                 {

y[22]=min(min(High1[1], Medium[2]),Low[3]);       //VH

yout22[4]= y[22];

}

 else;
//***********************************************************************/
for(i=0;i<N;i++)
{
                yout23[i]=0;

                }

 if (mf1[3]=='h' && mf2[2]=='m' && mf3[2]=='m')                    //23

                 {

y[23]=min(min(High1[1], Medium[2]),Medium[3]);       //H

yout23[3]= y[23];

}

 else;
//*************************************************************************/
for(i=0;i<N;i++)
{
                yout24[i]=0;

                }
 if (mf1[3]=='h' && mf2[2]=='m' && mf3[3]=='h')            //24

             {

y[24]=min(min(High1[1], Medium[2]),High1[3]);    //m

yout24[2]= y[24];

}

 else;
//*************************************************************************/

 for(i=0;i<N;i++)
{
            yout25[i]=0;

            }

 if (mf1[3]=='h' && mf2[3]=='h' && mf3[1]=='l')            //25

             {

y[25]=min(min(High1[1], High1[2]),Low[3]);    //VH

yout25[4]= y[25];
}

 else;
//************************************************************************/
for(i=0;i<N;i++)
{
            yout26[i]=0;

            }

 if (mf1[3]=='h' && mf2[3]=='h' && mf3[2]=='m')            //26

             {

y[26]=min(min(High1[1],High1[2]),Medium[3]);    //H

yout26[3]= y[26];
}

 else;
//**************************************************************************/

 for(i=0;i<N;i++)
{
            yout27[i]=0;

            }

 if (mf1[3]=='h' && mf2[3]=='h' && mf3[3]=='h')            //27

             {

y[27]=min(min(High1[1],High1[2]),High1[3]);    //l

yout27[1]= y[27];
}

 else;
//******************************Aggregation***********************************************/
for(i=0;i<N;i++)
            {

yout_1 =  max(max(max(max(max(max(max(max(max(yout1[i],yout2[i]),yout3[i]),yout4[i]),yout5[i]),yout6[i]),yout7[i]),yout8[i]), yout9[i]),yout10[i]);
yout_2= max(max(max(max(max(max(max(max(max(yout_1,yout11[i]),yout12[i]),yout13[i]),yout14[i]),yout15[i]),yout16[i]),yout17[i]),
yout18[i]),yout19[i]);
yout[i]= max(max(max(max(max(max(max(max(yout_2,yout20[i]),yout21[i]),yout22[i]),yout23[i]),yout24[i]),yout25[i]), yout26[i]),yout27[i]);

   }

//*********************************DEFFUZZIFICATION**************************************//
totalsum=0;multiplesum=0;

            for (i=0;i<N;i++)
{
totalsum=totalsum+yout[i];

}
            a=0;
```

```
for (i=0;i<N;i++)
{
x[i]=a;
a+=0.25;
}

for (i=0; i<N;i++)
{
g = yout[i]*x[i];
multiplesum = multiplesum + g;

}

output = multiplesum/totalsum; //PRIORITY INDEX

//**********************************************************************************************//

iciptr=op_pk_ici_get(pkptr);
op_ici_attr_set (iciptr, "Tos", output); //Setting Packet Priority Index

op_ici_attr_get(iciptr, "Tos",&PI);

PQ=PI;


//*******************************************Packet insertion in the Sub-queue************************//
if(PQ >= 0.0 && PQ <= 0.33)

{
queue_id=1;


  /* Attempt to place new packet in pending queue */
                    if (oms_buffer_bgutil_enqueue (routing_buffer1, pkptr)
                                    != OmsC_Buffer_Enqueue_Success)
                              {


                              /*          The insertion failed (due to a full buffer).          */
                              char intf_addr_str [IPC_ADDR_STR_LEN];

                              /*  Get a printable version of the interface addr.      */
                              if (rcvd_iface_info_ptr == OPC_NIL)

                                        sprintf (intf_addr_str, "Higher Layer");
                              else

                                        ip_address_print (intf_addr_str,
                                                  rcvd_iface_info_ptr->addr_range_ptr->address);

                              /*          Issue a warning message to the sim. log.                    */
                              ipnl_reswarn_pktinsert (op_pk_id (pkptr),
                                        op_pk_tree_id (pkptr), intf_addr_str);
                              /* Update packets dropped statistics and destroy      */
                              /* the IP datagram.
          */
                              ip_rte_dgram_discard (module_data_ptr, pkptr, op_pk_ici_get (pkptr), "Buffer overflow");
                    }


}
          else if (PQ > 0.33 && PQ<=0.66)
                    {
                              queue_id=2;

                    /* Attempt to place new packet in pending queue */

                              if (oms_buffer_bgutil_enqueue (routing_buffer2, pkptr)
                                        != OmsC_Buffer_Enqueue_Success)
                              {

                              /*          The insertion failed (due to a full buffer).          */
                              char intf_addr_str [IPC_ADDR_STR_LEN];

                              /*  Get a printable version of the interface addr.      */
                              if (rcvd_iface_info_ptr == OPC_NIL)

                                        sprintf (intf_addr_str, "Higher Layer");
                              else

                                        ip_address_print (intf_addr_str,
                                                  rcvd_iface_info_ptr->addr_range_ptr->address);

                              /*          Issue a warning message to the sim. log.                    */
                              ipnl_reswarn_pktinsert (op_pk_id (pkptr),
                                        op_pk_tree_id (pkptr), intf_addr_str);

                              /* Update packets dropped statistics and destroy     */
                              /* the IP datagram.
          */
                              ip_rte_dgram_discard (module_data_ptr, pkptr, op_pk_ici_get (pkptr), "Buffer overflow");
                              }

                              }

          else {   //
                  queue_id=3;

                    /* Attempt to place new packet in pending queue */
                    if (oms_buffer_bgutil_enqueue (routing_buffer3, pkptr)
                                        != OmsC_Buffer_Enqueue_Success)
                              {

                              /*        The insertion failed (due to a full buffer).         */
                              char intf_addr_str [IPC_ADDR_STR_LEN];

                              /*  Get a printable version of the interface addr.      */
                              if (rcvd_iface_info_ptr == OPC_NIL)

                                        sprintf (intf_addr_str, "Higher Layer");
                              else

                                        ip_address_print (intf_addr_str,
                                                  rcvd_iface_info_ptr->addr_range_ptr->address);

                              /*        Issue a warning message to the sim. log.                    */
                              ipnl_reswarn_pktinsert (op_pk_id (pkptr),
                                        op_pk_tree_id (pkptr), intf_addr_str);

                              /* Update packets dropped statistics and destroy     */
                              /* the IP datagram.
          */
                              ip_rte_dgram_discard (module_data_ptr, pkptr, op_pk_ici_get (pkptr), "Buffer overflow");
                              }
}

/*********************Alex Egaji************************/

                    }

          FOUT;
          }

static void
ip_rte_central_cpu_start_processing (void)
          {
          Packet *       pkptr;
          Ici *                    pk_ici_ptr;
          double                   comp_decomp_delay = 0.0;
          double                   pk_svc_time = 0.0;
          double                   bgutil_packets = 0.0;
          double                   bgutil_bytes = 0.0;
          double      bgutil_delay = 0.0;
          IpT_Rte_Ind_Ici_Fields *          intf_ici_fdstruct_ptr;
          IpT_Rte_Module_Data*              iprmd_ptr;

          /** Initiate the processing of a received packet       **/
          FIN (ip_rte_central_cpu_start_processing (void));

          /* Get a handle to the packet at the top of the buffer.This          */
          /* is the packet whose service is about to start. The                */
          /* packet is in the central processor's buffer.                                */


          /***************************************Alex Egaji*********************************************/

if (oms_buffer_bgutil_num_packets_get (routing_buffer1, OPC_NIL) > 0.0){

          pkptr = oms_buffer_bgutil_access (routing_buffer1, 0, OPC_NIL,
                    &bgutil_bytes, &bgutil_packets, &bgutil_delay);



          if (pkptr == OPC_NIL)
                    {
                    ip_rte_cpu_error ("Unable to get packet from head of central buffer.");
                    }


          /* Retrive the ICI associated with the packet */
```

```
		pk_ici_ptr = op_pk_ici_get (pkptr);

		/*		Get the rte_info_fields from the ici.	*/
		op_ici_attr_get (pk_ici_ptr, "rte_info_fields", &intf_ici_fdstruct_ptr);

		/*		Determine the time required to complete service of the		*/
		/*		enqueued packet, including any bgutil in front of it.		*/
		if (intf_ici_fdstruct_ptr->packet_is_labeled)
			{
			/* This is a labeled packet and the service time is	*/
			/* based on the packet forwarding rate					*/
			pk_svc_time = one_over_forwarding_rate + bgutil_delay;

			}
		else if (service_rate_pps)
			{
			/* rate in packets per second */
			pk_svc_time = one_over_service_rate + bgutil_delay + 0.000165;

			}
		else
			{
			/* rate in bits per second */
			pk_svc_time = (one_over_service_rate * op_pk_total_size_get (pkptr)) + bgutil_delay;

			}

		/* Retrieve the decompression and/or compression delay			*/
		/* associated with the packet. This delay will be added to		*/
		/* the service time when scheduling the end of the service. */
		/* If the packet will not be decompressed or compressed at	*/
		/* this processing, then the value of the delay is zero.		*/
		comp_decomp_delay = intf_ici_fdstruct_ptr->comp_decomp_delay;

		/* Reset the attribute.													*/
		if (comp_decomp_delay)
			{
			intf_ici_fdstruct_ptr->comp_decomp_delay = 0.0;
			}

		/* If the node is configured as a firewall, there can be		*/
		/* also a proxy server delay associated with the datagram.	*/
		/* Retrieve this delay, too.											*/
		if (ip_node_is_firewall (module_data_ptr))
			{
			/* We can safely add the proxy delay to the compression		*/
			/* and decompression delay, since all of them are			*/
			/* additional delays besides the routing delay and they		*/
			/* all will be summed up when the service completion		*/
			/* is scheduled.
		*/
			comp_decomp_delay += intf_ici_fdstruct_ptr->proxy_delay;

			/* Reset the ICI field.													*/
			intf_ici_fdstruct_ptr->proxy_delay = 0.0;
			}

		/* If the node is tunneling packets, there can also	be a vpn*/
		/* tunnel delay associated with the datagram. Retrieve this */
		/* delay, too.*/

		/* State variables cannot be passed to macros.	*/
		iprmd_ptr = module_data_ptr;
		if (ip_l2tp_vpn_is_enabled (iprmd_ptr))
			{
			/* The vpn delay can be added to the compression and	*/
			/* decompression delay the same way as proxy delay.			*/
			comp_decomp_delay += intf_ici_fdstruct_ptr->vpn_delay;

			/* Reset the ICI field.													*/
			intf_ici_fdstruct_ptr->vpn_delay = 0.0;
			}

		/* Use the resource management package to let it decide			*/
		/* how much CPU time it takes to do the requested				*/
		/* processing. End of processing is indicated with a				*/
		/* process interrupt.														*/
		/* Since we are dealing with one packet at a time, no need	*/
		/* to provide any structure.											*/
		job_id = Oms_Resource_Job_Insert (cpu_resource_handle, own_prohandle,
			OPC_NIL, pk_svc_time + comp_decomp_delay);

		if (Oms_Resource_Job_Id_Equal (job_id, OMSC_JOB_ID_INVALID))
			{
			ip_rte_cpu_warn ("Unable to schedule service completion.");
```

```
                }
        else
                {
                /*              Mark the server to be busy now.                                                 */
                cpu_is_idle = OPC_FALSE;
        }
        }

else if (oms_buffer_bgutil_num_packets_get (routing_buffer2, OPC_NIL) > 0.0){


                pkptr = oms_buffer_bgutil_access (routing_buffer2, 0, OPC_NIL,
                &bgutil_bytes, &bgutil_packets, &bgutil_delay);


        if (pkptr == OPC_NIL)
                {
                ip_rte_cpu_error ("Unable to get packet from head of central buffer.");
                }


        /* Retrive the ICI associated with the packet */
        pk_ici_ptr = op_pk_ici_get (pkptr);

        /*              Get the rte_info_fields from the ici.   */
        op_ici_attr_get (pk_ici_ptr, "rte_info_fields", &intf_ici_fdstruct_ptr);

        /*              Determine the time required to complete service of the          */
        /*              enqueued packet, including any bgutil in front of it.           */
        if (intf_ici_fdstruct_ptr->packet_is_labeled)
                {
                /* This is a labeled packet and the service time is     */
                /* based on the packet forwarding rate                                          */
                pk_svc_time = one_over_forwarding_rate + bgutil_delay;

                }
        else if (service_rate_pps)
                {
                /* rate in packets per second */
                pk_svc_time = one_over_service_rate + bgutil_delay + 0.000165;

                }
        else
                {
                /* rate in bits per second */
                pk_svc_time = (one_over_service_rate * op_pk_total_size_get (pkptr)) + bgutil_delay;

                }

        /* Retrieve the decompression and/or compression delay                  */
        /* associated with the packet. This delay will be added to      */
        /* the service time when scheduling the end of the service. */
        /* If the packet will not be decompressed or compressed at      */
        /* this processing, then the value of the delay is zero.        */
        comp_decomp_delay = intf_ici_fdstruct_ptr->comp_decomp_delay;

        /* Reset the attribute.                                                                                                 */
        if (comp_decomp_delay)
                {
                intf_ici_fdstruct_ptr->comp_decomp_delay = 0.0;
                }

        /* If the node is configured as a firewall, there can be        */
        /* also a proxy server delay associated with the datagram.      */
        /* Retrieve this delay, too.                                                                            */
        if (ip_node_is_firewall (module_data_ptr))
                {
                /* We can safely add the proxy delay to the compression         */
                /* and decompression delay, since all of them are               */
                /* additional delays besides the routing delay and they         */
                /* all will be summed up when the service completion            */
                /* is scheduled.
        */
                comp_decomp_delay += intf_ici_fdstruct_ptr->proxy_delay;

                /* Reset the ICI field.                                                                                 */
                intf_ici_fdstruct_ptr->proxy_delay = 0.0;
                }

        /* If the node is tunneling packets, there can also     be a vpn*/
        /* tunnel delay associated with the datagram. Retrieve this */
        /* delay, too.
                */

        /* State variables cannot be passed to macros.  */
```

```
		iprmd_ptr = module_data_ptr;
		if (ip_l2tp_vpn_is_enabled (iprmd_ptr))
			{
			/* The vpn delay can be added to the compression and	*/
			/* decompression delay the same way as proxy delay.			*/
			comp_decomp_delay += intf_ici_fdstruct_ptr->vpn_delay;

			/* Reset the ICI field.								*/
			intf_ici_fdstruct_ptr->vpn_delay = 0.0;
			}

		/* Use the resource management package to let it decide			*/
		/* how much CPU time it takes to do the requested				*/
		/* processing. End of processing is indicated with a			*/
		/* process interrupt.								*/
		/* Since we are dealing with one packet at a time, no need		*/
		/* to provide any structure.							*/
		job_id = Oms_Resource_Job_Insert (cpu_resource_handle, own_prohandle,
			OPC_NIL, pk_svc_time + comp_decomp_delay);

		if (Oms_Resource_Job_Id_Equal (job_id, OMSC_JOB_ID_INVALID))
			{
			ip_rte_cpu_warn ("Unable to schedule service completion.");
			}
		else
			{
			/*		Mark the server to be busy now.				*/
			cpu_is_idle = OPC_FALSE;
		}

		}
else if (oms_buffer_bgutil_num_packets_get (routing_buffer3, OPC_NIL) > 0.0){

		pkptr = oms_buffer_bgutil_access (routing_buffer3, 0, OPC_NIL,
			&bgutil_bytes, &bgutil_packets, &bgutil_delay);


		if (pkptr == OPC_NIL)
			{
			ip_rte_cpu_error ("Unable to get packet from head of central buffer.");
			}


		/* Retrive the ICI associated with the packet */
		pk_ici_ptr = op_pk_ici_get (pkptr);

		/*		Get the rte_info_fields from the ici.	*/
		op_ici_attr_get (pk_ici_ptr, "rte_info_fields", &intf_ici_fdstruct_ptr);

		/*		Determine the time required to complete service of the		*/
		/*		enqueued packet, including any bgutil in front of it.		*/
		if (intf_ici_fdstruct_ptr->packet_is_labeled)
			{
			/* This is a labeled packet and the service time is	*/
			/* based on the packet forwarding rate					*/
			pk_svc_time = one_over_forwarding_rate + bgutil_delay;

				}
		else if (service_rate_pps)
			{
			/* rate in packets per second */
			pk_svc_time = one_over_service_rate + bgutil_delay;

			}
		else
			{
			/* rate in bits per second */
			pk_svc_time = (one_over_service_rate * op_pk_total_size_get (pkptr)) + bgutil_delay;

			}

		/* Retrieve the decompression and/or compression delay		*/
		/* associated with the packet. This delay will be added to	*/
		/* the service time when scheduling the end of the service. */
		/* If the packet will not be decompressed or compressed at	*/
		/* this processing, then the value of the delay is zero.	*/
		comp_decomp_delay = intf_ici_fdstruct_ptr->comp_decomp_delay;

		/* Reset the attribute.								*/
		if (comp_decomp_delay)
			{
			intf_ici_fdstruct_ptr->comp_decomp_delay = 0.0;
			}

		/* If the node is configured as a firewall, there can be	*/
```

```
        /* also a proxy server delay associated with the datagram.     */
        /* Retrieve this delay, too.                                                          */
        if (ip_node_is_firewall (module_data_ptr))
                {
                /* We can safely add the proxy delay to the compression        */
                /* and decompression delay, since all of them are              */
                /* additional delays besides the routing delay and they        */
                /* all will be summed up when the service completion           */
                /* is scheduled.
        */
                comp_decomp_delay += intf_ici_fdstruct_ptr->proxy_delay;

                /* Reset the ICI field.                                                           */
                intf_ici_fdstruct_ptr->proxy_delay = 0.0;
                }

        /* If the node is tunneling packets, there can also    be a vpn*/
        /* tunnel delay associated with the datagram. Retrieve this */
        /* delay, too.
                */

        /* State variables cannot be passed to macros.      */
        iprmd_ptr = module_data_ptr;
        if (ip_l2tp_vpn_is_enabled (iprmd_ptr))
                {
                /* The vpn delay can be added to the compression and    */
                /* decompression delay the same way as proxy delay.                 */
                comp_decomp_delay += intf_ici_fdstruct_ptr->vpn_delay;

                /* Reset the ICI field.                                                           */
                intf_ici_fdstruct_ptr->vpn_delay = 0.0;
                }

        /* Use the resource management package to let it decide                 */
        /* how much CPU time it takes to do the requested                                 */
        /* processing. End of processing is indicated with a                     */
        /* process interrupt.                                                                 */
        /* Since we are dealing with one packet at a time, no need      */
        /* to provide any structure.                                                          */
        job_id = Oms_Resource_Job_Insert (cpu_resource_handle, own_prohandle,
                OPC_NIL, pk_svc_time + comp_decomp_delay);

        if (Oms_Resource_Job_Id_Equal (job_id, OMSC_JOB_ID_INVALID))
                {
                ip_rte_cpu_warn ("Unable to schedule service completion.");
                }
        else
                {
                /*          Mark the server to be busy now.                                  */
                cpu_is_idle = OPC_FALSE;
        }
        }

        FOUT;
        }

static void
ip_rte_central_cpu_send_packet (Packet* pkptr)
        {
        OpT_Packet_Size                         packet_size;
        Ici                                                     *pk_ici_ptr;
        IpT_Dgram_Fields*               pk_fd_ptr;
        IpT_Rte_Ind_Ici_Fields  *intf_ici_fdstruct_ptr;
        double size1;

        /** Send processed packet to appropriate level      **/
        FIN (ip_rte_central_cpu_send_packet (pkptr));

        /* Unless the processing speed is set to infinity,      */
        /* we need to extract the packet from the buffer.       */


        if (! infinite_processing_rate)
                {

/***************************************Alex Egaji*******************************************/

if (oms_buffer_bgutil_num_packets_get (routing_buffer1, OPC_NIL) > 0.0){

        /* Destroy job ID. Resource package does not                 */
                /* destroy the memory allocated to store this info.  */
                /* if there is no ici expected , the job id is      */
                /* destroyed by resource package                                  */
                op_prg_mem_free (job_id);
```

```
                    /* Block outgoing background traffic during the service of this packet. */
                    oms_buffer_bgutil_outgoing_flow_control(routing_buffer1, OPC_FALSE);

                    /* Extract packet at head of queue - this is the packet just        */
                    /* finishing service.
                    */
                    pkptr = oms_buffer_bgutil_dequeue (routing_buffer1);

                    /* Reset the outgoing background traffic flag. */
                    oms_buffer_bgutil_outgoing_flow_control (routing_buffer1, OPC_TRUE);



                    if (pkptr == OPC_NIL)
                              ip_rte_cpu_error ("Unable to get packet from head of central buffer.");

                    /* The central cpu becomes available again.        */
                    cpu_is_idle = OPC_TRUE;


}
else if (oms_buffer_bgutil_num_packets_get (routing_buffer2, OPC_NIL) > 0.0){

          /* Destroy job ID. Resource package does not                */
                    /* destroy the memory allocated to store this info.   */
                    /* if there is no ici expected , the job id is    */
                    /* destroyed by resource package                                              */
                    op_prg_mem_free (job_id);

                    /* Block outgoing background traffic during the service of this packet. */
                    oms_buffer_bgutil_outgoing_flow_control(routing_buffer2, OPC_FALSE);

                    /* Extract packet at head of queue - this is the packet just        */
                    /* finishing service.
                    */
                    pkptr = oms_buffer_bgutil_dequeue (routing_buffer2);

                    /* Reset the outgoing background traffic flag. */
                    oms_buffer_bgutil_outgoing_flow_control (routing_buffer2, OPC_TRUE);

                    if (pkptr == OPC_NIL)
                              ip_rte_cpu_error ("Unable to get packet from head of central buffer.");

                    /* The central cpu becomes available again.        */
                    cpu_is_idle = OPC_TRUE;


}

else{

          /* Destroy job ID. Resource package does not                */
                    /* destroy the memory allocated to store this info.   */
                    /* if there is no ici expected , the job id is    */
                    /* destroyed by resource package                                              */
                    op_prg_mem_free (job_id);

                    /* Block outgoing background traffic during the service of this packet. */
                    oms_buffer_bgutil_outgoing_flow_control(routing_buffer3, OPC_FALSE);

                    /* Extract packet at head of queue - this is the packet just        */
                    /* finishing service.
                    */
                    pkptr = oms_buffer_bgutil_dequeue (routing_buffer3);

                    /* Reset the outgoing background traffic flag. */
                    oms_buffer_bgutil_outgoing_flow_control (routing_buffer3, OPC_TRUE);

                    if (pkptr == OPC_NIL)
                              ip_rte_cpu_error ("Unable to get packet from head of central buffer.");

                    /* The central cpu becomes available again.        */
                    cpu_is_idle = OPC_TRUE;

}
 }
          /***************************************Alex Egaji******************************************/

          /* Retrieve the ICI associated with the packet.        */
          pk_ici_ptr = op_pk_ici_get (pkptr);

          /*        Get the data structure stored in the "rte_info_fields" field.        */
          if (op_ici_attr_exists (pk_ici_ptr, "rte_info_fields") == OPC_TRUE)
                    {
                    op_ici_attr_get (pk_ici_ptr, "rte_info_fields", &intf_ici_fdstruct_ptr);
```

```
					}
			else
					{
					ip_rte_cpu_error ("Unable to obtain routing information from packet.");
					}

			/* Get a handle to the data structure stored in "fields" field. */
			/* This structure contains all the protocol header information	*/
			/* as well as some internal information attached to each IP		*/
			/* datagram for simulation efficiency.								*/
			op_pk_nfd_access (pkptr, "fields", &pk_fd_ptr);

			/* Decompress the packet if necessary.				*/
			ip_rte_decompress (pkptr, intf_ici_fdstruct_ptr, pk_fd_ptr);

			/* Update the packet size. */
			packet_size = op_pk_total_size_get (pkptr);

			/* Compress datagram if specified */
			module_data_ptr->dgram_compressed_size = ip_rte_compress (
					module_data_ptr, pkptr, packet_size, intf_ici_fdstruct_ptr, pk_fd_ptr);

			/* Call the function that would forward the packet appropriately*/
			ip_rte_packet_send (module_data_ptr, pkptr, pk_ici_ptr,
					intf_ici_fdstruct_ptr, IpC_Rte_Process_Type_Central_Cpu, OPC_NIL);

			FOUT;
			}

/* Other local routines */

static void
ip_rte_cpu_error (const char *msg)
			{
			/** Print an error message and exit the simulation. **/
			FIN (ip_rte_cpu_error (msg));

			op_sim_end (
					"Error in IP central forwarding process model (ip_rte_central_cpu):",
					msg,
					"Check simulation log messages from the IP model for more information.",
					"Simulation Logging can be enabled/disabled in the Simulation Editor.");
			FOUT;
			}

static void
ip_rte_cpu_warn (const char *msg)
			{
			/** Print a warning message and resume. **/
			FIN (ip_rte_cpu_warn (msg));

			op_sim_message ("Warning from IP central forwarding process model (ip_rte_central_cpu):", msg);

			FOUT;
			}

static void
ip_rte_routing_processor_info_init (void)
			{
			Objid						ip_proc_info_objid;
			Objid						compound_objid;
			Compcode				return_status;
			double						buffer_size;
			OmsT_Buffer_Create	status;
			OmsT_Buffer_Stats	stat_array;
			int								i;
			double						service_rate;
			double						forwarding_rate;

			const char * stat_names [OmsC_Buffer_Stat_Number] =
					{
					"IP Processor.Forwarding Memory Overflows",
					"IP Processor.Forwarding Memory Free Size (bytes)",
					"", /* ignored */
					"IP Processor.Forwarding Memory Queue Size (bytes)",
					"IP Processor.Forwarding Memory Queue Size (packets)",
					"IP Processor.Forwarding Memory Queue Size (incoming bytes)",
					"IP Processor.Forwarding Memory Queue Size (incoming packets)",
					"IP Processor.Forwarding Memory Queuing Delay",
					"", /* delay variation ignored */
					};

			/** Extract routing processor information and perform appropriate initializtions **/
			FIN (ip_rte_routing_processor_info_init ());
```

```
	return_status = op_ima_obj_attr_get (own_id,
		"ip processing information", &ip_proc_info_objid);
	if (return_status == OPC_COMPCODE_FAILURE)
		ip_rte_cpu_error ("Unable to get forwarding information from attribute.");

	compound_objid = op_topo_child (ip_proc_info_objid, OPC_OBJTYPE_GENERIC, 0);

	/*	Get assigned value of server processing rate.				*/
	return_status = op_ima_obj_attr_get (compound_objid,
		"Datagram Forwarding Rate", &service_rate);
	if (return_status == OPC_COMPCODE_FAILURE)
		ip_rte_cpu_error ("Unable to get server datagram forwarding rate from attribute.");

	/* If the service rate is set to infinity, we don't need*/
	/* to create a buffer to hold the packets while they are	*/
	/* being serviced.
	*/
	if (IPC_EFFICIENCY_MODE_FORWARDING_RATE == service_rate)
		{
		/* Set the flag indicating that the processing speed is	*/
		/* infinite.
	*/
		infinite_processing_rate = OPC_TRUE;

		/* Write a log message warning the user that CPU stats	*/
		/* will not be recorded because the processing speed	*/
		/* is set to infinity.							*/
		if (OPC_FALSE == infinite_processing_log_written)
			{
			/* The log message has not been written already.	*/
			ipnl_cpu_stats_not_written_log_write ();

			/* Do not write the log message again.			*/
			infinite_processing_log_written = OPC_TRUE;
			}

		FOUT;
		}

	/* Finite processing speed.							*/
	infinite_processing_rate = OPC_FALSE;

	/* Store 1/service rate to avoid doing division each time */
	one_over_service_rate = 1.0 / service_rate;
module_data_ptr->service_rate = service_rate;

	/* MPLS forwarding rate						*/
	return_status = op_ima_obj_attr_get (compound_objid,
			"Datagram Switching Rate", &forwarding_rate);
	if (return_status == OPC_COMPCODE_FAILURE)
		ip_rte_cpu_error ("Unable to get Datagram Switching rate from the attribute.");
	one_over_forwarding_rate = 1.0/forwarding_rate;

	return_status = op_ima_obj_attr_get (compound_objid,
		"Forwarding Rate Units", &service_rate_pps);
	if (return_status == OPC_COMPCODE_FAILURE)
		ip_rte_cpu_error ("Unable to get units of forwarding rate from attribute.");

	/* Find out routing processing buffer size and create corresponding buffer	*/
	return_status = op_ima_obj_attr_get (compound_objid,
		"Memory Size", &buffer_size);
	if (return_status == OPC_COMPCODE_FAILURE)
		ip_rte_cpu_error ("Unable to get forwarding memory size from attribute.");

	/* The buffer pool will be used for the routing queue, as well	*/
	/* as the QoS buffers, if no slots are used.				*/
	/* The pool size is based on the routing processor memory size	*/
	/* The pool does not have any size limitations.	*/
	/* The various buffers will have their own.		*/

	//routing_buffer_pool


	routing_buffer_pool = oms_buffer_bgutil_pool_create (
		buffer_size,			/* number of bytes */
		OmsC_Buffer_Infinite,	/* number of packets */
		OPC_NIL, "Routing Processor Pool", OPC_NIL, &status);

	if (routing_buffer_pool == OPC_NIL)
		{
		/* Something went wrong, report and terminate */
		switch (status)
			{
			case OmsC_Buffer_Create_Invalid_Byte_Size:
				op_sim_end (
                                        "Unable to create IP shared memory pool due to an invalid byte size",
                                        "", "", "");
                                break;
                        case OmsC_Buffer_Create_Invalid_Packet_Size:
                                op_sim_end (
                                        "Unable to create IP shared memory pool due to an invalid packet size",
                                        "", "", "");
                                break;
                        case OmsC_Buffer_Create_Memory_Alloc_Failed:
                                op_sim_end (
                                        "Unable to create IP shared memory pool due to an internal memory allocation failure",
                                        "", "", "");
                                break;
                        }
                }

        /* All stats but free size in packets (since no upper limit) */
        for (i = 0; i < OmsC_Buffer_Stat_Number; i++)
                {
                if (stat_names [i][0] == '\0')
                stat_array [i].used = OPC_FALSE;
                else
                        {
                stat_array [i].used = OPC_TRUE;
                stat_array [i].shandle = op_stat_reg (stat_names [i],
                                OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
                        }
                }
        /**************************ALEX EGAJI******************************/

        routing_buffer1 = oms_buffer_bgutil_create (routing_buffer_pool,
                OmsC_Buffer_Parent_Limit, OmsC_Buffer_Parent_Limit,
                module_data_ptr->do_bgutil,
                service_rate, service_rate_pps,
                &stat_array, "Routing Processor Buffer", OPC_NIL, /* No user data */
                &status);
        routing_buffer2 = oms_buffer_bgutil_create (routing_buffer_pool,
                OmsC_Buffer_Parent_Limit, OmsC_Buffer_Parent_Limit,
                module_data_ptr->do_bgutil,
                service_rate, service_rate_pps,
                &stat_array, "Routing Processor Buffer", OPC_NIL, /* No user data */
                &status);
        routing_buffer3 = oms_buffer_bgutil_create (routing_buffer_pool,
                OmsC_Buffer_Parent_Limit, OmsC_Buffer_Parent_Limit,
                module_data_ptr->do_bgutil,
                service_rate, service_rate_pps,
                &stat_array, "Routing Processor Buffer", OPC_NIL, /* No user data */
                &status);

        /********************************ALEX EGAJI*************************/
                if ((routing_buffer1 == OPC_NIL)||(routing_buffer2 == OPC_NIL)||(routing_buffer3 == OPC_NIL))


                ip_rte_cpu_error ("Unable to create routing processor memory buffer.");


        FOUT;}
```

Appendix F :Packet Insertion in the Queue

```
/* Opnet Model Support package	*/
/* Buffer sub-package			*/
/* Specialized version for packets and bytes */
/* also aware of background utilization			*/

/****************************************/
/*		Copyright (c) 1986-2011		*/
/*	by OPNET Technologies, Inc.	*/
/*	  (A Delaware Corporation)	*/
/*	7255 Woodmont Av., Suite 250		*/
/*	Bethesda, MD 20814, U.S.A.	*/
/*	  All Rights Reserved.		*/
/****************************************/

/* Define symbolic constant for export. */
#define OMSC_EXPORT	DLLEXPORT

/* ---- (SECTION_INC) Include Directives ---- */

#include "oms_buffer_bgutil.h"
#include "oms_bgutil.h"
#include "oms_buffer_int.h"
#include <string.h>

/* ---- (SECTION_LSC) Local Symbolic Constants ---- */

/* ---- (SECTION_MD) Macro Definitions ---- */

/* ---- (SECTION_LDT) Local Datatypes ---- */

struct OmsT_Buffer_Bgutil_Pool
		{
		OmsT_Buffer_Pool	pool;
		double						bgutil_bytes;
		double						bgutil_packets;
		};

typedef struct OmsT_Buffer_Bgutil_Pool		OmsT_Buffer_Bgutil_Pool;

struct OmsT_Buffer_Bgutil
		{
		OmsT_Buffer								buffer;
		Boolean									do_bgutil;
		Boolean									allow_outgoing;
		OmsT_Bgutil_Routed_State *		bgutil_routed_state_ptr;
		double									bgutil_last_update_time;
		/* These 3 represent bgutil scattered throughout the buffer */
		double									total_bgutil_bytes;
		double									total_bgutil_packets;
		double									total_bgutil_seconds;
		/* These 3 represent bgutil after last element */
		double									trailing_bgutil_bytes;
		double									trailing_bgutil_packets;
		double									trailing_bgutil_seconds;
		/* Information passed to bgutil when updating */
		double									bgutil_average_rate;
		Boolean									bgutil_rate_pps;
		};

typedef struct OmsT_Buffer_Bgutil			OmsT_Buffer_Bgutil;

/* Each element is not just a packet but a datastructure that	*/
/* also keeps track of bgutil data.												*/
typedef struct OmsT_Buffer_Bgutil_Element
		{
		Packet *	pkptr;
		double			size;
```

```
          /* bgutil before the packet in this element */
          double                  bgutil_bytes;
          double                  bgutil_packets;
          double                  bgutil_seconds;
          } OmsT_Buffer_Bgutil_Element;

/* ---- (SECTION_GD) Global Data ---- */

static    Pmohandle  omsi_buffer_bgutil_pmohandle;
static    Pmohandle  omsi_buffer_bgutil_pool_pmohandle;
static    Pmohandle  omsi_buffer_bgutil_element_pmohandle;
static    int                        omsi_buffer_bgutil_pmohandles_defined = OPC_FALSE;

/* ---- (SECTION LP) Local Prototypes ---- */

#define oms_buffer_bgutil_update(buffer_ptr)        \
          oms_buffer_bgutil_update_core (buffer_ptr, op_sim_time(), OPC_NIL, 0.0, OPC_TRUE, OPC_NIL, OPC_NIL)

#define oms_buffer_bgutil_pool_update(pool_ptr)  \
          oms_buffer_bgutil_pool_update_core (pool_ptr, op_sim_time(), \
                    OPC_NIL, OPC_NIL, 0.0, 0.0, OPC_NIL, OPC_NIL)

static void         oms_buffer_bgutil_update_core (OmsT_Buffer_Bgutil * buffer_ptr,
          double cur_time, Packet * pkptr, double processed_bgutil,
          Boolean update_pool, double * delta_bytes_ptr, double * delta_packets_ptr);
static void                   oms_buffer_bgutil_pool_update_core (OmsT_Buffer_Bgutil_Pool * pool_ptr,
          double cur_time, OmsT_Buffer_Bgutil_Pool * invoker_pool,
          OmsT_Buffer_Bgutil * invoker_buffer,
          double delta_bytes, double delta_packets,
          double * delta_bytes_ptr, double * delta_packets_ptr);
static double       oms_buffer_bgutil_element_byte_size_get (void * elemptr);
static void         oms_buffer_bgutil_element_destroy (void * elemptr);
static const char * oms_buffer_bgutil_element_trace (void * elemptr);

/* ---- (SECTION_ECP) Externally Callable Procedures ---- */

OmsT_Buffer_Bgutil_Pool_Handle
oms_buffer_bgutil_pool_create (
          double    max_bytes_v,
          double    max_packets_v,
          OmsT_Buffer_Stats *   buffer_stat_array,
          const char *          buffer_name_str,
          void *                                   user_data_ptr_v,
          OmsT_Buffer_Create * status_ptr)
          {
          OmsT_Buffer_Bgutil_Pool_Handle    buffer_pool_handle;

          /** Create and initialize a buffer pool for packets and bgutil **/
          FIN (oms_buffer_bgutil_pool_create (max_bytes_size, max_packet_size,
                    buffer_stat_ptr, user_data_ptr));

          if (omsi_buffer_bgutil_pmohandles_defined == OPC_FALSE)
                    {
                    omsi_buffer_bgutil_pool_pmohandle =
                              prg_pmo_define ((char *)"OmsT_Buffer_Bgutil_Pool",
                                        sizeof (OmsT_Buffer_Bgutil_Pool), 4);
                    omsi_buffer_bgutil_pmohandle =
                              prg_pmo_define ((char *)"OmsT_Buffer_Bgutil",
                                        sizeof (OmsT_Buffer_Bgutil), 10);
                    omsi_buffer_bgutil_element_pmohandle =
                              prg_pmo_define ((char *)"OmsT_Buffer_Bgutil_Element",
                                        sizeof (OmsT_Buffer_Bgutil_Element), 50);
                    omsi_buffer_bgutil_pmohandles_defined = OPC_TRUE;
                    }

          buffer_pool_handle = (OmsT_Buffer_Bgutil_Pool_Handle)
                    prg_pmo_alloc (omsi_buffer_bgutil_pool_pmohandle);
          if (buffer_pool_handle == OPC_NIL)
                    {
                    if (status_ptr != OPC_NIL)
                              *status_ptr = OmsC_Buffer_Create_Memory_Alloc_Failed;
                    FRET (OPC_NIL);
                    }

          oms_buffer_pool_create_storage (&buffer_pool_handle->pool,
                    max_bytes_v, max_packets_v,
                    (OmsT_Buffer_Insertion_Criteria_Proc)OPC_NIL,
                    (OmsT_Buffer_Element_Size_Proc)oms_buffer_bgutil_element_byte_size_get,
                    (OmsT_Buffer_Element_Destroy_Proc)oms_buffer_bgutil_element_destroy,
                    (OmsT_Buffer_Element_Trace_Proc)oms_buffer_bgutil_element_trace,
                    buffer_stat_array, buffer_name_str,
                    "packet", "packets", "byte", "bytes",
                    user_data_ptr_v, status_ptr);

          buffer_pool_handle->bgutil_bytes = 0.0;
```

```
		buffer_pool_handle->bgutil_packets = 0.0;

		FRET (buffer_pool_handle);
		}

OmsT_Buffer_Bgutil_Pool_Handle
oms_buffer_bgutil_subpool_create (
		OmsT_Buffer_Bgutil_Pool_Handle parent_pool_handle,
		double							max_bytes,
		double							max_packets,
		OmsT_Buffer_Stats *	buffer_stat_array,
		const char *			buffer_pool_name,
		void *							user_data_ptr,
		OmsT_Buffer_Create * status_ptr)
		{
		OmsT_Buffer_Bgutil_Pool_Handle	subpool_handle;

		/** Create subpool, which is identical to the default process **/
		FIN (oms_buffer_bgutil_subpool_create (parent_pool_handle, ...));

		subpool_handle = (OmsT_Buffer_Bgutil_Pool_Handle)
				prg_pmo_alloc (omsi_buffer_bgutil_pool_pmohandle);
		if (subpool_handle == OPC_NIL)
				{
				if (status_ptr != OPC_NIL)
						*status_ptr = OmsC_Buffer_Create_Memory_Alloc_Failed;
				FRET (OPC_NIL);
				}

		oms_buffer_subpool_create_storage (&subpool_handle->pool,
				(OmsT_Buffer_Pool_Handle)parent_pool_handle,
				max_bytes, max_packets,
				(OmsT_Buffer_Insertion_Criteria_Proc)OPC_NIL,
				buffer_stat_array, buffer_pool_name,
				user_data_ptr, status_ptr);

		subpool_handle->bgutil_bytes = 0.0;
		subpool_handle->bgutil_packets = 0.0;

		FRET (subpool_handle);
		}

/* Destroy a buffer pool	*/
/* Returns the data pointer set at creation time */
void *
oms_buffer_bgutil_pool_destroy (
		OmsT_Buffer_Bgutil_Pool_Handle buffer_bgutil_pool_handle)
		{
		void *		user_data_ptr;
		double		delta_bytes;
		double		delta_packets;
		OmsT_Buffer_Bgutil_Pool *		parent_bgutil;
		OmsT_Buffer_Pool *	parent_pool;

		FIN (oms_buffer_bgutil_pool_destroy (buffer_bgutil_pool_handle));

		if (buffer_bgutil_pool_handle == OPC_NIL)
				FRET (OPC_NIL);

		oms_buffer_bgutil_pool_update (buffer_bgutil_pool_handle);
		delta_bytes = buffer_bgutil_pool_handle->bgutil_bytes;
		delta_packets = buffer_bgutil_pool_handle->bgutil_packets;
		parent_pool = buffer_bgutil_pool_handle->pool.parent_pool;
		while (parent_pool != OPC_NIL)
				{
				parent_bgutil = (OmsT_Buffer_Bgutil_Pool *)parent_pool;
				parent_bgutil->bgutil_bytes -= delta_bytes;
				parent_bgutil->bgutil_packets -= delta_packets;
				parent_pool = parent_pool->parent_pool;
				}

		user_data_ptr = oms_buffer_pool_destroy (
				(OmsT_Buffer_Pool_Handle)buffer_bgutil_pool_handle);

		FRET (user_data_ptr);
		}

double
oms_buffer_bgutil_pool_num_packets_get (
		OmsT_Buffer_Bgutil_Pool_Handle buffer_bgutil_pool_handle,
		double * bgutil_packets_ptr)
		{
		double		num;

		/** Obtain current number of packets in buffer pool, include bgutil		**/
```

```
	/** bgutil_packets_ptr, if not OPC_NIL, is set with just the bgutil count**/
	FIN (oms_buffer_bgutil_pool_num_packets_get (buffer_bgutil_pool_handle, bgutil_packets_ptr));

	if (buffer_bgutil_pool_handle == OPC_NIL)
		{
		if (bgutil_packets_ptr != OPC_NIL)
			*bgutil_packets_ptr = 0.0;
		FRET (0.0);
		}

	oms_buffer_bgutil_pool_update (buffer_bgutil_pool_handle);

	num = buffer_bgutil_pool_handle->pool.num_elements;
	if (bgutil_packets_ptr != OPC_NIL)
		*bgutil_packets_ptr = buffer_bgutil_pool_handle->bgutil_packets;
	FRET (num);
	}

double
oms_buffer_bgutil_pool_free_packets_get (
	OmsT_Buffer_Bgutil_Pool_Handle buffer_bgutil_pool_handle)
	{
	/** Obtain available number of element slots in buffer pool **/
	/** Returns OmsC_Buffer_Infinite or positive number				**/
	FIN (oms_buffer_bgutil_pool_free_packets_get (buffer_bgutil_pool_handle));

	if (buffer_bgutil_pool_handle == OPC_NIL)
		FRET (0.0);

	oms_buffer_bgutil_pool_update (buffer_bgutil_pool_handle);

	FRET (oms_buffer_pool_free_elements_get (&buffer_bgutil_pool_handle->pool));
	}

double
oms_buffer_bgutil_pool_num_bytes_get (
	OmsT_Buffer_Bgutil_Pool_Handle buffer_bgutil_pool_handle,
	double * bgutil_bytes_ptr)
	{
	double	num;

	/** Obtain current number of units in buffer pool, including bgutil		**/
	/** bgutil_bytes_ptr, if not OPC_NIL, is set to just tbe bgutil count		**/
	FIN (oms_buffer_bgutil_pool_num_bytes_get (buffer_bgutil_pool_handle, bgutil_bytes_ptr));

	if (buffer_bgutil_pool_handle == OPC_NIL)
		{
		if (bgutil_bytes_ptr != OPC_NIL)
			*bgutil_bytes_ptr = 0.0;
		FRET (0.0);
		}

	oms_buffer_bgutil_pool_update (buffer_bgutil_pool_handle);

	num = buffer_bgutil_pool_handle->pool.num_units;
	if (bgutil_bytes_ptr != OPC_NIL)
		*bgutil_bytes_ptr = buffer_bgutil_pool_handle->bgutil_bytes;
	FRET (num);
	}

double
oms_buffer_bgutil_pool_free_bytes_get (
	OmsT_Buffer_Bgutil_Pool_Handle buffer_bgutil_pool_handle)
	{
	/** Obtain available number of units in buffer pool **/
	/** Returns OmsC_Buffer_Infinite or positive number		**/
	FIN (oms_buffer_bgutil_pool_free_bytes_get (buffer_bgutil_pool_handle));

	if (buffer_bgutil_pool_handle == OPC_NIL)
		FRET (0.0);

	oms_buffer_bgutil_pool_update (buffer_bgutil_pool_handle);

	FRET (oms_buffer_pool_free_units_get (&buffer_bgutil_pool_handle->pool));
	}

void
oms_buffer_bgutil_pool_report (
	OmsT_Buffer_Bgutil_Pool_Handle	buffer_bgutil_pool_handle,
	const char *			header_str)
	{
	/** Generate ODB detailed report on the specified pool **/
	char		line1 [OMSC_BUFFER_TRACE_STR_LEN];
	char		line2 [OMSC_BUFFER_TRACE_STR_LEN];
	char		line3 [OMSC_BUFFER_TRACE_STR_LEN];
	char		line4 [OMSC_BUFFER_TRACE_STR_LEN];
	char		str [100];
	int			num_subpools, num_buffers;
	OmsT_Buffer_Pool * buffer_pool_ptr;

	/** Generate detailed report on the pool for ODB output		**/
	FIN (oms_buffer_bgutil_pool_report (buffer_pool_handle, header_str));

	if (buffer_bgutil_pool_handle == OPC_NIL)
		FOUT;

	buffer_pool_ptr = &buffer_bgutil_pool_handle->pool;

	if (buffer_pool_ptr->parent_pool == OPC_NIL)
		{
		sprintf (line1, "Oms Buffer Pool %s (no parent pool)",
			oms_buffer_pool_trace_name (buffer_pool_ptr));
		}
	else
		{
		sprintf (line1, "Oms Buffer Subpool %s",
			oms_buffer_pool_trace_name (buffer_pool_ptr));
		if (buffer_pool_ptr->parent_pool->name == OPC_NIL)
			{
			strcat (line1, "(unamed parent pool)");
			}
		else
			{
			strcat (line1, "(parent pool");
			strcat (line1, buffer_pool_ptr->parent_pool->name);
			strcat (line1, ")");
			}
		}
	strcpy (line2, "Capacity: ");
	strcat (line2, oms_buffer_pool_trace_elements (buffer_pool_ptr,
		buffer_pool_ptr->max_elements));
	strcat (line2, ", ");
	strcat (line2, oms_buffer_pool_trace_units (buffer_pool_ptr,
		buffer_pool_ptr->max_units));
	strcpy (line3, "Content: ");
	strcat (line3, oms_buffer_pool_trace_elements (buffer_pool_ptr,
		buffer_pool_ptr->num_elements));
	if (buffer_bgutil_pool_handle->bgutil_packets < OMSC_BUFFER_TRACE_MAX_ULONG)
		sprintf (str, "(%f background)", buffer_bgutil_pool_handle->bgutil_packets);
	else
		sprintf (str, "(%.g background)", buffer_bgutil_pool_handle->bgutil_packets);
	strcat (line3, str);
	strcat (line3, ", ");
	strcat (line3, oms_buffer_pool_trace_units (buffer_pool_ptr,
		buffer_pool_ptr->num_units));
	if (buffer_bgutil_pool_handle->bgutil_bytes < OMSC_BUFFER_TRACE_MAX_ULONG)
		sprintf (str, "(%f background)", buffer_bgutil_pool_handle->bgutil_bytes);
	else
		sprintf (str, "(%.g background)", buffer_bgutil_pool_handle->bgutil_bytes);
	strcat (line3, str);

	num_subpools = op_prg_list_size (buffer_pool_ptr->subpool_list);
	num_buffers = op_prg_list_size (buffer_pool_ptr->buffer_list);
	sprintf (line4, "%d subpool%s, %d buffer%s",
		num_subpools, (num_subpools == 1 ? "" : "s"),
		num_buffers, (num_buffers == 1 ? "" : "s"));

	if (header_str != OPC_NIL)
		op_prg_odb_print_major ((char *)header_str, line1, line2, line3, line4, OPC_NIL);
		else
		op_prg_odb_print_major (line1, line2, line3, line4, OPC_NIL);

	FOUT;
	}

OmsT_Buffer_Bgutil_Handle
oms_buffer_bgutil_create (
	OmsT_Buffer_Bgutil_Pool_Handle	buffer_pool,
	/* Maximum number (>0) or OmsC_Buffer_Infinite */
	double					max_packets,
	double					max_bytes,
	Boolean					use_bgutil,
	double					average_bgutil_rate, /* if 'use_bgutil' is true */
	Boolean					packets_per_second,  /* if 'use_bgutil' is true */
	OmsT_Buffer_Stats *  buffer_stats,
	/* For tracing information. If not OPC_NIL, copied internally */
	const char *			buffer_name,
	void *					user_data_ptr,
	OmsT_Buffer_Create * status_ptr)
	{
        OmsT_Buffer_Bgutil_Handle        buffer_ptr;
        OmsT_Buffer_Create                      status;

        /** Create a bgutil-aware buffer for packets **/
        FIN (oms_buffer_bgutil_create (buffer_pool, max_packets, max_bytes,
                use_bgutil, buffer_stats, buffer_name, user_data_ptr, status_ptr));

        buffer_ptr = (OmsT_Buffer_Bgutil_Handle)
                prg_pmo_alloc (omsi_buffer_bgutil_pmohandle);
        if (buffer_ptr == OPC_NIL)
                {
                if (status_ptr != OPC_NIL)
                        *status_ptr = OmsC_Buffer_Create_Memory_Alloc_Failed;
                FRET (OPC_NIL);
                }

        /* Perform underlying oms_buffer initialization */
        oms_buffer_create_storage (&buffer_ptr->buffer,
                (OmsT_Buffer_Pool *)buffer_pool,
                max_packets, max_bytes, (OmsT_Buffer_Insertion_Criteria_Proc)OPC_NIL,
                buffer_stats, buffer_name, user_data_ptr, &status, OPC_NIL);

        /* Set the status ptr for the benefit of the        */
        /* calling proc                                                                                        */
        if (status_ptr != OPC_NIL)
                *status_ptr = status;

                        *status_ptr = status;

        if (status != OmsC_Buffer_Create_Success)
                {
                prg_mem_free (buffer_ptr);
                FRET (OPC_NIL);
                }


        /* State is initially nil, to be created by bgutil later on */
        buffer_ptr->bgutil_routed_state_ptr = OPC_NIL;
        buffer_ptr->do_bgutil = use_bgutil;
        buffer_ptr->allow_outgoing = OPC_TRUE;
        buffer_ptr->bgutil_rate_pps = packets_per_second;
        buffer_ptr->bgutil_average_rate = average_bgutil_rate;
        buffer_ptr->trailing_bgutil_bytes = 0.0;
        buffer_ptr->trailing_bgutil_packets = 0.0;
        buffer_ptr->trailing_bgutil_seconds = 0.0;
        buffer_ptr->total_bgutil_bytes = 0.0;
        buffer_ptr->total_bgutil_packets = 0.0;
        buffer_ptr->total_bgutil_seconds = 0.0;
        buffer_ptr->bgutil_last_update_time = -1.0;

        FRET (buffer_ptr);
        }

void *
oms_buffer_bgutil_destroy (OmsT_Buffer_Bgutil_Handle buffer_bgutil_handle)
        {
        void *      user_data_ptr;

        /** Destroy a buffer                                                                           **/
        /** Returns the data pointer set at creation time       **/
        FIN (oms_buffer_bgutil_destroy (buffer_bgutil_handle));

        user_data_ptr = oms_buffer_destroy (&buffer_bgutil_handle->buffer);

        /* ### Don't seem to have any way of destroying bgutil */

        prg_mem_free (buffer_bgutil_handle);

        FRET (user_data_ptr);
        }


void
oms_buffer_bgutil_outgoing_flow_control (OmsT_Buffer_Bgutil_Handle buffer_bgutil_handle,
        Boolean allow_flow)
        {
        /** Block/allow outgoing bgutil flow if no explicit packet **/
        FIN (oms_buffer_bgutil_outgoing_flow_control (buffer_bgutil_handle, allow_flow));

        if (buffer_bgutil_handle != OPC_NIL)
                buffer_bgutil_handle->allow_outgoing = allow_flow;

        FOUT;
        }
```

```
OmsT_Buffer_Enqueue
oms_buffer_bgutil_enqueue (OmsT_Buffer_Bgutil_Handle buffer_bgutil_handle,
        Packet * pkptr)
        {
        Ici*                iciptr_q; //OAE
        Ici*                iciptr;   //OAE
        OmsT_Buffer_Bgutil_Element *     element_ptr; //OAE
        OmsT_Buffer_Bgutil_Element *     element1;  //OAE
        OmsT_Buffer_Enqueue                          enqueue_result;
        Boolean                                                              tracer_pkt;
        double              k,p; /* where k=pq for current packet and p=pq for packet already in queue.*/OAE
        int                 index,i; //OAE

        /** Attempt to enqueue a packet                                      **/
        /** Returns appropriate value to indicate result       **/
        FIN (oms_buffer_bgutil_enqueue (buffer_bgutil_handle));


        if (buffer_bgutil_handle == OPC_NIL)
                FRET (OmsC_Buffer_Enqueue_Null_Handle);
        if (pkptr == OPC_NIL)
                FRET (OmsC_Buffer_Enqueue_Null_Element);

        oms_buffer_bgutil_update_core (buffer_bgutil_handle, op_sim_time(),
                pkptr, 0.0, OPC_TRUE, OPC_NIL, OPC_NIL);

        /* At this point, the "trailing" bgutil in the buffer     */
        /* represents what needs to be processed before this              */
        /* new packet can get its own processing.                                */
        element_ptr = (OmsT_Buffer_Bgutil_Element *)
                prg_pmo_alloc (omsi_buffer_bgutil_element_pmohandle);
        if (element_ptr == OPC_NIL)
                {

                FRET (OmsC_Buffer_Enqueue_Memory_Alloc_Failed);
                }

        element_ptr->pkptr = pkptr;
        element_ptr->bgutil_bytes = buffer_bgutil_handle->trailing_bgutil_bytes;
        element_ptr->bgutil_packets = buffer_bgutil_handle->trailing_bgutil_packets;
        element_ptr->bgutil_seconds = buffer_bgutil_handle->trailing_bgutil_seconds;
        /* Tracer packets are considered 0 size since they do not            */
        /* exist in real life and thus should not cause buffer               */
        /* overflow when used in a simulation.                                                           */
        tracer_pkt = op_pk_encap_flag_is_set (pkptr, OMSC_BGUTIL_ENCAP_FLAG_INDEX);
        if (tracer_pkt){
                element_ptr->size = 0;
                        }
        else
                element_ptr->size = (double) op_pk_total_size_get (pkptr)/8;

/***********************************Alex Egaji**********************************/


        if(oms_buffer_num_elements_get(&buffer_bgutil_handle->buffer)>0)

        {
        for (i=oms_buffer_num_elements_get(&buffer_bgutil_handle->buffer)-1;i>=0;i--)
                {

                element1=(OmsT_Buffer_Bgutil_Element *)
                oms_buffer_access (&buffer_bgutil_handle->buffer, i, OPC_NIL);

                iciptr=op_pk_ici_get(pkptr);  // New arriving Packet
                iciptr_q=op_pk_ici_get(element1->pkptr);   //Existing packet in the queue
                op_ici_attr_get(iciptr_q, "Tos",&k); //Priority index for Existing packets in the queue

                  op_ici_attr_get(iciptr, "Tos",&p); //Priority index for New arriving Packet to the queue

                if (p>k || p==k) {

                        break;
                }


                }

                 index=i;

                 }
        ql=oms_buffer_num_units_get (&buffer_bgutil_handle->buffer);// current number of bytes in buffer

        /* Attempt to enqueue element */
```

```
  enqueue_result = oms_buffer_enqueue_with_index (&buffer_bgutil_handle->buffer,element_ptr, VOSC_NIL, op_sim_time (), index); //

/*****************************************************************************************/


        if (enqueue_result != OmsC_Buffer_Enqueue_Success)
                {

                /* If unsuccessful, delete element                 */
                /* No changes to the trailing bgutil      */
                prg_mem_free (element_ptr);

                }
        else
                {

                /* The packet and its preceeding bgutil have been enqueued        */
                /* There is no longer any trailing bgutil.                                              */
                buffer_bgutil_handle->trailing_bgutil_bytes = 0;
                buffer_bgutil_handle->trailing_bgutil_packets = 0;
                buffer_bgutil_handle->trailing_bgutil_seconds = 0;
                }

        FRET (enqueue_result);
        }

Packet *
oms_buffer_bgutil_access (OmsT_Buffer_Bgutil_Handle buffer_bgutil_handle,
        int             index, double *              insertion_time_ptr, double * bgutil_bytes_ptr,
        double * bgutil_packets_ptr, double * bgutil_delay_ptr)
        {
        OmsT_Buffer_Bgutil_Element *        element;

        /** Returns address of packet 'index' positions from the head      **/
        /** of the buffer but while leaving the element in the buffer.       **/
        /** Returns OPC_NIL if empty, otherwise the packet ptr and                   **/
        /** fill in the packet insertion time and the number of bgutil       **/
        /** bytes and packets in front of the explicit packet.                                   **/
        /** The last 3 pointer can be OPC_NIL in which case no values **/
        /** will be filled in.
        **/
        FIN (oms_buffer_bgutil_access (buffer_bgutil_handle, index,
                insertion_time_ptr, bgutil_bytes_ptr, bgutil_packets_ptr));

        element = (OmsT_Buffer_Bgutil_Element *)
                oms_buffer_access (&buffer_bgutil_handle->buffer, index,
                        insertion_time_ptr);

        if (element == OPC_NIL)
                FRET (OPC_NIL);

        if (bgutil_bytes_ptr != OPC_NIL)
                *bgutil_bytes_ptr = element->bgutil_bytes;
        if (bgutil_packets_ptr != OPC_NIL)
                *bgutil_packets_ptr = element->bgutil_packets;
        if (bgutil_delay_ptr != OPC_NIL)
                *bgutil_delay_ptr = element->bgutil_seconds;

        FRET (element->pkptr);
        }

Packet *
oms_buffer_bgutil_dequeue (OmsT_Buffer_Bgutil_Handle buffer_bgutil_handle)
        {
        OmsT_Buffer_Bgutil_Element *        element;
        OmsT_Buffer_Bgutil_Pool *                       parent_pool;
        Packet *                                                        pkptr;

        /** Remove packet at the head of the buffer.         **/
        /** Returns OPC_NIL if empty                                            **/
        FIN (oms_buffer_bgutil_dequeue (buffer_bgutil_handle));

        if (buffer_bgutil_handle == OPC_NIL)
                FRET (OPC_NIL);

        element = (OmsT_Buffer_Bgutil_Element *)
                oms_buffer_dequeue (&buffer_bgutil_handle->buffer, 0, op_sim_time ());

        if (element == OPC_NIL)
                FRET (OPC_NIL);

        /* Have bgutil before packet considered processed  */
        /* Deal with potential computation rounding errors                 */
        buffer_bgutil_handle->total_bgutil_bytes -= element->bgutil_bytes;
        if (buffer_bgutil_handle->total_bgutil_bytes < 0.0)
```

```
			buffer_bgutil_handle->total_bgutil_bytes = 0.0;
		buffer_bgutil_handle->total_bgutil_packets -= element->bgutil_packets;
		if (buffer_bgutil_handle->total_bgutil_packets < 0.0)
			buffer_bgutil_handle->total_bgutil_packets = 0.0;
		buffer_bgutil_handle->total_bgutil_seconds -= element->bgutil_seconds;
		if (buffer_bgutil_handle->total_bgutil_seconds < 0.0)
			buffer_bgutil_handle->total_bgutil_seconds = 0.0;
		/* update parent pools as well */
		parent_pool = (OmsT_Buffer_Bgutil_Pool *)
			buffer_bgutil_handle->buffer.parent_pool;
		while (parent_pool != OPC_NIL)
			{
			parent_pool->bgutil_bytes -= element->bgutil_bytes;
			if (parent_pool->bgutil_bytes < 0.0)
				parent_pool->bgutil_bytes = 0.0;
			parent_pool->bgutil_packets -= element->bgutil_packets;
			if (parent_pool->bgutil_packets < 0.0)
				parent_pool->bgutil_packets = 0.0;
			parent_pool = (OmsT_Buffer_Bgutil_Pool *)
				parent_pool->pool.parent_pool;
			}

		oms_buffer_bgutil_update_core (buffer_bgutil_handle, op_sim_time(),
			OPC_NIL, element->bgutil_seconds, OPC_TRUE, OPC_NIL, OPC_NIL);
		/* Now the buffer correctly contains the total amount of pending		*/
		/* bgutil and the trailing amount correctly represents what has		*/
		/* been received after the last enqueued packet (if any)				*/

		pkptr = element->pkptr;
		prg_mem_free (element);

		FRET (pkptr);
		}

void
oms_buffer_bgutil_head_trim (OmsT_Buffer_Bgutil_Handle	buffer_bgutil_handle,
		double bgutil_packets, double bgutil_bytes)
		{
		OmsT_Buffer_Bgutil_Element *	element;
		double		max_packets, max_bytes, max_seconds;
		double		delta_seconds;

		/** Remove 'bgutil_packets' or 'bgutil_bytes' from the bgutil		**/
		/** in front of the explicit packet at the head of the queue.		**/
		/** If no explicit packets are present, remove from overall bgutil	**/
		/** bgutil_packets or bgutil_bytes can be 0, in which case the		**/
		/** amount of bgutil being removed is based on the other value		**/
		FIN (oms_buffer_bgutil_head_trim (buffer_bgutil_handle, bgutil_packets, bgutil_bytes));

		if (buffer_bgutil_handle == OPC_NIL)
			FOUT;

		/* Can't be used to add bgutil! */
		if ((bgutil_packets < 0) || (bgutil_bytes < 0))
			FOUT;

		if ((bgutil_packets == 0) && (bgutil_bytes == 0))
			FOUT;

		element = (OmsT_Buffer_Bgutil_Element *)
			oms_buffer_access (&buffer_bgutil_handle->buffer, 0, OPC_NIL);

		if (element == OPC_NIL)
			{
			/* Remove from trailing bgutil */
			max_bytes = buffer_bgutil_handle->trailing_bgutil_bytes;
			max_packets = buffer_bgutil_handle->trailing_bgutil_packets;
			max_seconds = buffer_bgutil_handle->trailing_bgutil_seconds;
			}
		else
			{
			/* Remove from preceding bgutil in element */
			max_bytes = element->bgutil_bytes;
			max_packets = element->bgutil_packets;
			max_seconds = element->bgutil_seconds;
			}

		if ((bgutil_bytes >= max_bytes) || (bgutil_packets >= max_seconds))
			{
			/* Can only remove up to what is there */
			bgutil_bytes = max_bytes;
			bgutil_packets = max_seconds;
			}

		if (bgutil_bytes == 0)
```

```
			{
			/* Appropriate byte fraction */
			bgutil_bytes = (bgutil_packets * max_bytes) / max_packets;
			}
		else if (bgutil_packets == 0)
			{
			/* A bit trickier, get an appropriate number of packets */
			bgutil_packets = (long)((bgutil_bytes * max_packets ) / max_bytes);
			}

		if (element == OPC_NIL)
			{
			/* Remove from trailing bgutil */
			buffer_bgutil_handle->trailing_bgutil_bytes -= bgutil_bytes;
			buffer_bgutil_handle->trailing_bgutil_packets -= bgutil_packets;
			}
		else
			{
			/* Remove from preceding bgutil in element */
			element->bgutil_bytes -= bgutil_bytes;
			element->bgutil_packets -= bgutil_packets;
			}

		/* The corresponding variation is expressed in second		*/
		/* for the bgutil package.								*/
		delta_seconds = (bgutil_packets * max_seconds) / max_packets;

		oms_buffer_bgutil_update_core (buffer_bgutil_handle, op_sim_time (),
			OPC_NIL, delta_seconds, OPC_TRUE, OPC_NIL, OPC_NIL);

		FOUT;
		}

int
oms_buffer_bgutil_packet_find (OmsT_Buffer_Bgutil_Handle buffer_bgutil_handle,
		Packet * pkptr)
		{
		OmsT_Buffer_Bgutil_Element *	element_ptr;
		int								i;

		/** Search for the matching packet in the buffer, returning the		**/
		/** position in the buffer
			**/
		/** Returns OmsC_Buffer_Element_Not_Found if the element is not found	**/
		FIN (oms_buffer_bgutil_packet_find (buffer_bgutil_handle, pkptr));

		if (buffer_bgutil_handle == OPC_NIL)
			FRET (OmsC_Buffer_Element_Not_Found);

		if (pkptr == OPC_NIL)
			FRET (OmsC_Buffer_Element_Not_Found);

		for (i = 0; i < buffer_bgutil_handle->buffer.num_elements; i++)
			{
			element_ptr = (OmsT_Buffer_Bgutil_Element *)
				oms_buffer_access (&buffer_bgutil_handle->buffer, i, OPC_NIL);
			if (element_ptr->pkptr == pkptr)
				{
				FRET (i);
				}
			}

		FRET (OmsC_Buffer_Element_Not_Found);
		}

Boolean
oms_buffer_bgutil_is_empty (OmsT_Buffer_Bgutil_Handle buffer_bgutil_handle)
		{
		/** Checks if buffer is empty (explicit and bgutil data) **/
		FIN (oms_buffer_bgutil_is_empty (buffer_bgutil_handle));

		if (buffer_bgutil_handle == OPC_NIL)
			FRET (OPC_TRUE);

		oms_buffer_bgutil_update (buffer_bgutil_handle);

		/* If there is at least one packet, it is not empty */
		if (buffer_bgutil_handle->buffer.num_elements != 0)
			{
			FRET (OPC_FALSE);
			}
		else
		 {
			/* No explicit packets doesn't mean no bgutil */
			FRET (buffer_bgutil_handle->trailing_bgutil_bytes == 0 ? OPC_TRUE : OPC_FALSE);
```

```
                    }
            }

double
oms_buffer_bgutil_num_packets_get (OmsT_Buffer_Bgutil_Handle buffer_bgutil_handle,
        double * bgutil_packets_ptr)
        {
        double          num;

        /** Return current number of explicit and bgutil packets in buffer                  **/
        /** bgutil_packets_ptr, if not OPC_NIL, is set to just the bgutil count **/
        FIN (oms_buffer_bgutil_num_packets_get (buffer_bgutil_handle, bgutil_packets_ptr));

        if (buffer_bgutil_handle == OPC_NIL)
                {
                if (bgutil_packets_ptr != OPC_NIL)
                        *bgutil_packets_ptr = 0.0;
                FRET (0.0);
                }

        oms_buffer_bgutil_update (buffer_bgutil_handle);

        num = oms_buffer_num_elements_get (&buffer_bgutil_handle->buffer);
        if (bgutil_packets_ptr != OPC_NIL)
                *bgutil_packets_ptr = buffer_bgutil_handle->total_bgutil_packets;
        FRET (num);
        }

double
oms_buffer_bgutil_free_packets_get (OmsT_Buffer_Bgutil_Handle buffer_bgutil_handle)
        {
        /** Obtain available number of packet slots in buffer          **/
        /** Returns OmsC_Buffer_Infinite or positive number                   **/
        FIN (oms_buffer_bgutil_free_packets_get (buffer_bgutil_handle));

        if (buffer_bgutil_handle == OPC_NIL)
                FRET (0.0);

        oms_buffer_bgutil_update (buffer_bgutil_handle);

        FRET (oms_buffer_free_elements_get (&buffer_bgutil_handle->buffer));
        }

double
oms_buffer_bgutil_num_bytes_get (OmsT_Buffer_Bgutil_Handle buffer_bgutil_handle,
        double * bgutil_bytes_ptr)
        {
        double          num;

        /** Return current number of explicit and bgutil bytes in buffer     **/
        /** bgutil_bytes_ptr, if not OPC_NIL, is set to just the bgutil count  **/
        FIN (oms_buffer_bgutil_num_bytes_get (buffer_bgutil_handle, bgutil_bytes_ptr));

        if (buffer_bgutil_handle == OPC_NIL)
                {
                if (bgutil_bytes_ptr != OPC_NIL)
                        *bgutil_bytes_ptr = 0.0;
                FRET (0.0);
                }

        oms_buffer_bgutil_update (buffer_bgutil_handle);

        num = oms_buffer_num_units_get (&buffer_bgutil_handle->buffer);
        if (bgutil_bytes_ptr != OPC_NIL)
                *bgutil_bytes_ptr = buffer_bgutil_handle->total_bgutil_bytes;
        FRET (num);
        }

double
oms_buffer_bgutil_free_bytes_get (OmsT_Buffer_Bgutil_Handle buffer_bgutil_handle)
        {
        /** Obtain available number of bytes in buffer                 **/
        /** Returns OmsC_Buffer_Infinite or positive number            **/
        FIN (oms_buffer_bgutil_free_bytes_get (buffer_bgutil_handle));

        if (buffer_bgutil_handle == OPC_NIL)
                FRET (0.0);

        oms_buffer_bgutil_update (buffer_bgutil_handle);

        FRET (oms_buffer_free_units_get (&buffer_bgutil_handle->buffer));
        }

/* Generate ODB detailed report on the specified buffer */
void
```

```
oms_buffer_bgutil_report (OmsT_Buffer_Bgutil_Handle          buffer_bgutil_handle,
        const char * header_str)
        {
        char            line1 [OMSC_BUFFER_TRACE_STR_LEN];
        char            line2 [OMSC_BUFFER_TRACE_STR_LEN];
        char            line3 [OMSC_BUFFER_TRACE_STR_LEN];
        char            str [100];
        OmsT_Buffer *           buffer_ptr;

        /** Generate detailed report on the buffer for ODB output          **/
        FIN (oms_buffer_bgutil_report (buffer_handle, header_str));

        if (buffer_bgutil_handle == OPC_NIL)
                FOUT;

        oms_buffer_bgutil_update (buffer_bgutil_handle);

        buffer_ptr = &buffer_bgutil_handle->buffer;

        sprintf (line1, "Oms Buffer (%s background utilization) %s",
                (buffer_bgutil_handle->do_bgutil ? "with" : "without"),
                oms_buffer_trace_name (buffer_ptr));
        if (buffer_ptr->parent_pool->name == OPC_NIL)
                {
                strcat (line1, "(unamed parent pool)");
                }
        else
                {
                strcat (line1, "(parent pool");
                strcat (line1, buffer_ptr->parent_pool->name);
                strcat (line1, ")");
                }
        strcpy (line2, "Capacity: ");
        strcat (line2, oms_buffer_trace_elements (buffer_ptr,
                buffer_ptr->max_elements));
        strcat (line2, ", ");
        strcat (line2, oms_buffer_trace_units (buffer_ptr,
                buffer_ptr->max_units));
        strcpy (line3, "Content: ");
        strcat (line3, oms_buffer_trace_elements (buffer_ptr,
                buffer_ptr->num_elements));
        if (buffer_bgutil_handle->do_bgutil)
                {
                if (buffer_bgutil_handle->total_bgutil_packets < OMSC_BUFFER_TRACE_MAX_ULONG)
                        sprintf (str, "(%f background)", buffer_bgutil_handle->total_bgutil_packets);
                else
                        sprintf (str, "(%.g background)", buffer_bgutil_handle->total_bgutil_packets);
                strcat (line3, str);
                }

        strcat (line3, ", ");
        strcat (line3, oms_buffer_trace_units (buffer_ptr,
                buffer_ptr->num_units));
        if (buffer_bgutil_handle->bgutil_routed_state_ptr != OPC_NIL)
                {
                if (buffer_bgutil_handle->total_bgutil_bytes < OMSC_BUFFER_TRACE_MAX_ULONG)
                        sprintf (str, "(%f background)", buffer_bgutil_handle->total_bgutil_bytes);
                else
                        sprintf (str, "(%.g background)", buffer_bgutil_handle->total_bgutil_bytes);
                strcat (line2, str);
                }

        if (header_str != OPC_NIL)
                op_prg_odb_print_major ((char *)header_str, line1, line2, line3, OPC_NIL);
        else
                op_prg_odb_print_major (line1, line2, line3, OPC_NIL);

        FOUT;
        }

/* ---- (SECTION_ICP) Internally Callable Procedures ---- */

static void
oms_buffer_bgutil_pool_update_core (OmsT_Buffer_Bgutil_Pool * pool_ptr,
        double cur_time, OmsT_Buffer_Bgutil_Pool * invoker_pool,
        OmsT_Buffer_Bgutil * invoker_buffer,
        double invoker_delta_bytes, double invoker_delta_packets,
        double * delta_bytes_ptr, double * delta_packets_ptr)
        {
        OmsT_Buffer_Bgutil *                    buffer_ptr;
        OmsT_Buffer_Bgutil_Pool *               subpool_ptr;
        OmsT_Buffer_Bgutil_Pool *               parent_pool;
        double                                                          delta_bytes, delta_packets;
        double                                                          sum_bytes, sum_packets;
        PrgT_List_Cell *                                cell_ptr;
```

```
/** Update bgutil information in the pool as well as its              **/
/** subpools, parent pool, or buffers.                                                                **/
/** To break recursion, does not invoke itself for a pool            **/
/** (parent or child) matching the invoker_pool or a buffer        **/
/** matching the invoker_buffer.                                                                                  **/
/** delta_bytes and delta_packets represent changes from          **/
/** the invoker (a buffer or a subpool) for the subtree or           **/
/** leaf that will not be invoked.                                                                                **/
/** delta_bytes_ptr and delta_packets_ptr are to be filled          **/
/** with the changes in the pool's subtree for use by the            **/
/** invoker.
**/
FIN (oms_buffer_bgutil_pool_update_core (pool_ptr, invoker_pool, invoker_buffer));

sum_bytes = 0;
sum_packets = 0;

/* Iterate through buffers, skipping invoker_buffer */
/* Sum all changes, including what was passed in if                  */
/* we match one of the buffers as the invoker.                       */
for (cell_ptr = prg_list_head_cell_get (pool_ptr->pool.buffer_list);
          cell_ptr;
          cell_ptr = prg_list_cell_next_get (cell_ptr))
         {
         buffer_ptr = (OmsT_Buffer_Bgutil *) prg_list_cell_data_get (cell_ptr);
         if (buffer_ptr != invoker_buffer)
                  {
                  oms_buffer_bgutil_update_core (buffer_ptr, cur_time, OPC_NIL, 0.0,
                           OPC_FALSE, /* no recurse back to here */
                           &delta_bytes, &delta_packets);
                  sum_bytes += delta_bytes;
                  sum_packets += delta_packets;
                  }
         else
                  {
                  /* This is the invoker.  Add its changes to the sum */
                  sum_bytes += invoker_delta_bytes;
                  sum_packets += invoker_delta_packets;
                  }
         }

/* Iterate through subpools, skipping invoker_pool */
/* The subpool update should not recurse back to us                  */
for (cell_ptr = prg_list_head_cell_get (pool_ptr->pool.subpool_list);
          cell_ptr;
          cell_ptr = prg_list_cell_next_get (cell_ptr))
         {
         subpool_ptr = (OmsT_Buffer_Bgutil_Pool *) prg_list_cell_data_get (cell_ptr);
         if (subpool_ptr != invoker_pool)
                  {
                  oms_buffer_bgutil_pool_update_core (subpool_ptr,
                           cur_time, pool_ptr, OPC_NIL, /* invoked from pool, not buffer */
                           0.0, 0.0, /* no effect on subpool's totals */
                           &delta_bytes, &delta_packets);
                  sum_bytes += delta_bytes;
                  sum_packets += delta_packets;
                  }
         else
                  {
                  /* This is the invoker.  Add its changes to the sum */
                  sum_bytes += invoker_delta_bytes;
                  sum_packets += invoker_delta_packets;
                  }
         }

/* Update pool count based on what is passed to us                   */
/* and what has been obtained from subelements                       */
pool_ptr->bgutil_bytes += sum_bytes;
pool_ptr->bgutil_packets += sum_packets;

/* If the parent is not the invoker, recurse up to it    */
/* such that it will not invoke us again.                                                */
parent_pool = (OmsT_Buffer_Bgutil_Pool *)(pool_ptr->pool.parent_pool);
if ((pool_ptr->pool.parent_pool != OPC_NIL) &&
         (parent_pool != invoker_pool))
         {
         oms_buffer_bgutil_pool_update_core (parent_pool,
                  cur_time, pool_ptr, OPC_NIL, /* invoked from pool, not buffer */
                  sum_bytes, sum_packets, /* how much this child changed */
                  OPC_NIL, OPC_NIL); /* don't care about parent's changes */
         }

/* Report changes to invoker if requested */
if (delta_bytes_ptr != OPC_NIL)
```

```
			*delta_bytes_ptr = sum_bytes;
		if (delta_packets_ptr != OPC_NIL)
			*delta_packets_ptr = sum_packets;

		FOUT;
		}

static void
oms_buffer_bgutil_update_core (OmsT_Buffer_Bgutil * buffer_ptr,
		double cur_time, Packet * pkptr, double processed_bgutil,
		Boolean update_pool, double * delta_bytes_ptr, double * delta_packets_ptr)
		{
		double		free_space;
		double		bgutil_seconds, bgutil_bytes, bgutil_packets;
		double		delta_bytes = 0.0, delta_packets = 0.0;
		OmsT_Bgutil_Router_Work_Proc_State * pk_work_proc_state_ptr = OPC_NIL;


		/** Update bgutil information in the buffer.					**/
   /** Optionally provide an amount of bgutil that is				**/
		/** considered processed externally and thus needs to be		**/
		/** removed from the pending amount. If update_pool is true,**/
   /** triggers an update in its parent pool.						**/
		/** delta_bytes_ptr and delta_packets_ptr are to be filled		**/
		/** with the changes in the pool's subtree for use by the		**/
		/** invoker.
		**/
		FIN (oms_buffer_bgutil_update_core (buffer_ptr, processed_bgutil, update_pool));

		if (buffer_ptr->do_bgutil)
			{
			/* Only do it once per time value if no processed_bgutil */
			if ((cur_time > buffer_ptr->bgutil_last_update_time) ||
				(processed_bgutil > 0))
				{
				free_space = oms_buffer_free_units_get (&buffer_ptr->buffer);

				/** The underlying bgutil keeps track of the current		**/
				/** background flows in term of packets and bytes.		**/
				/** If the buffer is currently empty, we want to use		**/
				/** the "free-flowing" mode to let the underlying KPs		**/
				/** figure out how much bgutil was produced and		**/
				/** consumed since the last explicit packet was removed	**/
				/** On the other hand, if the buffer is not empty, we		**/
				/** want to control how bgutil is consumed.					**/
				/** No explicit work is added but some bgutil work			**/
				/** might have been processed.
		**/

				bgutil_seconds = oms_bgutil_router_delay_calculate_core (pkptr,
					&buffer_ptr->bgutil_routed_state_ptr,
					cur_time, free_space,
					buffer_ptr->bgutil_average_rate, buffer_ptr->bgutil_rate_pps,
					0,
					(OpT_Boolean) (buffer_ptr->buffer.num_elements == 0 ? buffer_ptr->allow_outgoing : OPC_FALSE),
					processed_bgutil,
					&bgutil_bytes, &bgutil_packets);

				if (buffer_ptr->bgutil_routed_state_ptr == OPC_NIL)
					{
					bgutil_bytes = 0.0;
					bgutil_packets = 0.0;
					}
				else
					{
					/* Use a local varible pointing to our pk_work_proc_state. */
					pk_work_proc_state_ptr = (OmsT_Bgutil_Router_Work_Proc_State *)
						buffer_ptr->bgutil_routed_state_ptr->pk_work_proc_state_ptr;

					if (buffer_ptr->bgutil_rate_pps)
						{
						/* Forwarding rate is in terms of packets_per_sec */
						if (bgutil_seconds < 1.0/buffer_ptr->bgutil_average_rate)
							{
							/* Remaining work is from a fraction of a bgutil packet */
							bgutil_packets = ((bgutil_seconds == 0.0) ? 0.0 : 1.0);

							bgutil_bytes = bgutil_packets * pk_work_proc_state_ptr-
>avg_bytes_per_packet;
							}
						else
							{
							/* Remaining work is from more than one packet */
							bgutil_packets = bgutil_seconds * buffer_ptr->bgutil_average_rate;
```

```
                                                bgutil_bytes   = pk_work_proc_state_ptr->avg_bytes_per_packet *
bgutil_packets;
                                                }
                                        }
                                else
                                        {
                                        /* Forwarding rate is in terms of bits_per_sec */
                                        if (bgutil_seconds <
                                                (pk_work_proc_state_ptr->avg_bytes_per_packet * 8.0 / buffer_ptr-
>bgutil_average_rate))
                                                {
                                                /* Remaining bgutil work is from a fraction of a bgutil packet */
                                                bgutil_packets = ((bgutil_seconds == 0.0) ? 0.0 : 1.0);
                                                bgutil_bytes = bgutil_packets * pk_work_proc_state_ptr-
>avg_bytes_per_packet;
                                                }
                                        else
                                                {
                                                /* Remaining bgutil work is from more than one packet */
                                                bgutil_bytes = (bgutil_seconds * buffer_ptr->bgutil_average_rate) / 8.0;
                                                bgutil_packets = bgutil_bytes / pk_work_proc_state_ptr-
>avg_bytes_per_packet;
                                                }
                                        }
                                }

                        delta_bytes = bgutil_bytes - buffer_ptr->total_bgutil_bytes;
                        delta_packets = bgutil_packets - buffer_ptr->total_bgutil_packets;

                        /* Any new bgutil goes in the trailing area */
                        buffer_ptr->trailing_bgutil_bytes += delta_bytes;
                        buffer_ptr->trailing_bgutil_packets += delta_packets;
                        buffer_ptr->trailing_bgutil_seconds +=
                                (bgutil_seconds - buffer_ptr->total_bgutil_seconds);
                        /* deal with possible rounding errors */
                        if (buffer_ptr->trailing_bgutil_seconds < 0.0)
                                buffer_ptr->trailing_bgutil_seconds = 0.0;

                        /* The new totals can be updated */
                        buffer_ptr->total_bgutil_bytes = bgutil_bytes;
                        buffer_ptr->total_bgutil_packets = bgutil_packets;
                        buffer_ptr->total_bgutil_seconds = bgutil_seconds;

                        buffer_ptr->bgutil_last_update_time = cur_time;
                        }
                }

        if (update_pool)
                oms_buffer_bgutil_pool_update_core (
                        (OmsT_Buffer_Bgutil_Pool *)buffer_ptr->buffer.parent_pool,
                        cur_time,
                        OPC_NIL, buffer_ptr,                /* invoked from buffer              */
                        delta_bytes, delta_packets,  /* provided to pool for change */
                        OPC_NIL, OPC_NIL);                          /* don't care about pool changes */

        /* Report changes to invoker if requested */
        if (delta_bytes_ptr != OPC_NIL)
                *delta_bytes_ptr = delta_bytes;
        if (delta_packets_ptr != OPC_NIL)
                *delta_packets_ptr = delta_packets;

        FOUT;
        }

static double
oms_buffer_bgutil_element_byte_size_get (void * elemptr_v)
        {
        OmsT_Buffer_Bgutil_Element * elemptr = (OmsT_Buffer_Bgutil_Element *)elemptr_v;

        /** Return the packet total size in bytes             **/
        FIN (oms_buffer_bgutil_element_byte_size_get (pkptr));

        /* The size is stored in the element since tracer packets          */
        /* are considered to be of size 0.                                                        */

        FRET (elemptr->size);
        }

static void
oms_buffer_bgutil_element_destroy (void * elemptr_v)
        {
        OmsT_Buffer_Bgutil_Element * elemptr = (OmsT_Buffer_Bgutil_Element *)elemptr_v;

        /** Destroy packet stored in oms_buffer **/
        FIN (oms_buffer_bgutil_element_destroy (elemptr));
```

```
	op_pk_destroy (elemptr->pkptr);
	prg_mem_free (elemptr);

	FOUT;
	}

static const char *
oms_buffer_bgutil_element_trace (void * elemptr_v)
	{
	OmsT_Buffer_Bgutil_Element * elemptr = (OmsT_Buffer_Bgutil_Element *)elemptr_v;
	static char	trace_buffer [40];

	/** Return the packet id for ODB tracing in oms_buffer **/
	FIN (oms_buffer_bgutil_element_trace (pkptr));

	sprintf (trace_buffer, SIMC_PK_ID_FMT, op_pk_id (elemptr->pkptr));

	FRET (trace_buffer);
	}
```

Appendix G :Implementation of Video Streaming

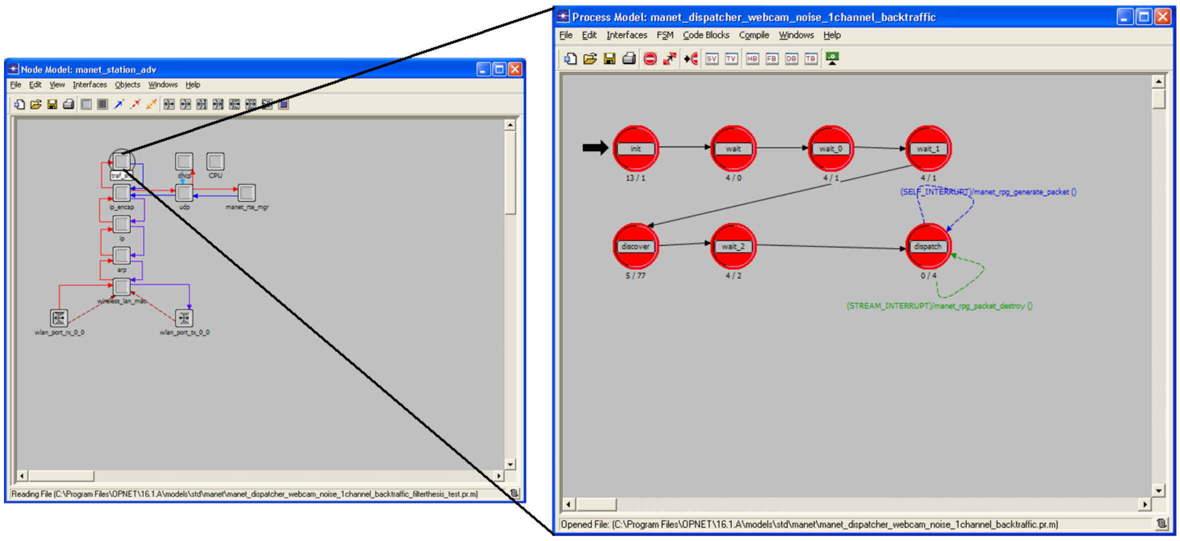

Node Model: manet_station_adv
File Edit View Interfaces Objects Windows Help
traf_s
dhcp
CPU
ip_encap
udp
manet_rte_mgr
ip
arp
wireless_lan_mac
wlan_port_rx_0_0
wlan_port_tx_0_0
Reading File (C:\Program Files\OPNET\16.1.A\models\std\manet\manet_dispatcher_webcam_noise_1channel_backtraffic_filterthesis_test.pr.m)
Process Model: manet_dispatcher_webcam_noise_1channel_backtraffic
File Edit Interfaces FSM Code Blocks Compile Windows Help
init
13 / 1
wait
4 / 0
wait_0
4 / 1
wait_1
4 / 1
(SELF_INTERRUPT)/manet_rpg_generate_packet ()
discover
5 / 77
wait_2
4 / 2
dispatch
0 / 4
(STREAM_INTERRUPT)/manet_rpg_packet_destroy ()
Opened File: (C:\Program Files\OPNET\16.1.A\models\std\manet\manet_dispatcher_webcam_noise_1channel_backtraffic.pr.m)

Figure G-8: Implementation of Video Streaming

```
static void  manet_rpg_sv_init (void)
                {
                /** Initializes all state variables used in this process model.        **/
                FIN (manet_rpg_sv_init ());

                /* Obtain the object identifiers for the surrounding module,        */
                /* node and subnet.
                                */
                my_objid        = op_id_self ();
                my_node_objid    = op_topo_parent (my_objid);
                my_subnet_objid  = op_topo_parent (my_node_objid);

                /* Create the ici that will be associated with packets being        */
                /* sent to IP.
                                */
                ip_encap_req_ici_ptr = op_ici_create ("inet_encap_req");

                /* Register the local and global statictics.                                                    */

                local_packets_sent_hndl                    = op_stat_reg ("MANET.Traffic Sent (packets/sec)",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
                local_bits_sent_hndl                       = op_stat_reg ("MANET.Traffic Sent (bits/sec)",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
                local_packets_received_hndl        = op_stat_reg ("MANET.Traffic Received (packets/sec)",                  OPC_STAT_INDEX_NONE,
OPC_STAT_LOCAL);
                local_bits_received_hndl       = op_stat_reg ("MANET.Traffic Received (bits/sec)",                  OPC_STAT_INDEX_NONE,
OPC_STAT_LOCAL);
                local_delay_hndl              = op_stat_reg ("MANET.Delay (secs)",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
                global_delay_hndl              = op_stat_reg ("MANET.Delay (secs)",
OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
                global_packets_sent_hndl                   = op_stat_reg ("MANET.Traffic Sent (packets/sec)",
OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
                global_bits_sent_hndl                      = op_stat_reg ("MANET.Traffic Sent (bits/sec)",
OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
                global_packets_received_hndl     = op_stat_reg ("MANET.Traffic Received (packets/sec)",                  OPC_STAT_INDEX_NONE,
OPC_STAT_GLOBAL);
                global_bits_received_hndl                  = op_stat_reg ("MANET.Traffic Received (bits/sec)",                  OPC_STAT_INDEX_NONE,
OPC_STAT_GLOBAL);
```

```
		local_packets_sent_hndl			= op_stat_reg ("MANET.Traffic Sent (packets/sec)",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
		local_bits_sent_hndl			= op_stat_reg ("MANET.Traffic Sent (bits/sec)",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
		local_packets_received_hndl	= op_stat_reg ("MANET.Traffic Received (packets/sec)",			OPC_STAT_INDEX_NONE,
OPC_STAT_LOCAL);
		local_bits_received_hndl	= op_stat_reg ("MANET.Traffic Received (bits/sec)",			OPC_STAT_INDEX_NONE,
OPC_STAT_LOCAL);


		local_delay_hndl		= op_stat_reg ("MANET.Delay (secs)",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);

   local_packets_received_real_hndl	= op_stat_reg ("MANET.Traffic Received video (packets/sec)",			OPC_STAT_INDEX_NONE,
OPC_STAT_LOCAL);
   local_bits_received_real_hndl		= op_stat_reg ("MANET.Traffic Received video (bits/sec)",		OPC_STAT_INDEX_NONE,
OPC_STAT_LOCAL);

   local_delay_real_hndl		= op_stat_reg ("MANET.Delay video (secs)",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);

   local_packets_received_back_hndl	= op_stat_reg ("MANET.Traffic Received Background (packets/sec)",		OPC_STAT_INDEX_NONE,
OPC_STAT_LOCAL);
   local_bits_received_back_hndl		= op_stat_reg ("MANET.Traffic Received Background (bits/sec)",		OPC_STAT_INDEX_NONE,
OPC_STAT_LOCAL);

		local_delay_back_hndl		= op_stat_reg ("MANET.Delay Background (secs)",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);


		FOUT;
		}


static void  manet_rpg_register_self (void)
		{
		char						proc_model_name [128];
		OmsT_Pr_Handle				own_process_record_handle;
		Prohandle					own_prohandle;

		/** Get a higher layer protocol ID from IP and register this	**/
		/** process in the model-wide process registry to be discovered **/
		/** by the lower layer.
		**/
		FIN (rpg_dispatcher_register_self ());

		/* Register RPG as a higher layer protocol over IP layer			*/
		/* and retrieve an auto-assigned protocol id.							*/
		higher_layer_proto_id = IpC_Protocol_Unspec;
		Inet_Higher_Layer_Protocol_Register ("rpg", &higher_layer_proto_id);

		/* Obtain the process model name and process handle.					*/
		op_ima_obj_attr_get (my_objid, "process model", proc_model_name);
		own_prohandle = op_pro_self ();

		/* Register this process in the model-wide process registry			*/
		own_process_record_handle = (OmsT_Pr_Handle) oms_pr_process_register
			(my_node_objid, my_objid, own_prohandle, proc_model_name);

		/* Set the protocol attribute to the same string we used in		*/
		/* Ip_Higher_Layer_Protocol_Register. Necessary for ip_encap*/
		/* to discover this process. Set the module object id also.		*/
		oms_pr_attr_set (own_process_record_handle,
									"protocol",			OMSC_PR_STRING, "rpg",
									"module objid",		OMSC_PR_OBJID,		my_objid,
									OPC_NIL);
		FOUT;
		}


static void  manet_rpg_sent_stats_update (double size)
		{
		/** This function updates the local and global statictics			**/
		/** related with packet sending.												**/
		FIN (manet_rpg_sent_stats_update (<args>));


		/* Update the local and global sent statistics.							*/
		op_stat_write (local_bits_sent_hndl,     size);
		op_stat_write (local_bits_sent_hndl,     0.0);
		op_stat_write (local_packets_sent_hndl,  1.0);
		op_stat_write (local_packets_sent_hndl,  0.0);
		op_stat_write (global_bits_sent_hndl,    size);
```

```
          op_stat_write (global_bits_sent_hndl,    0.0);
          op_stat_write (global_packets_sent_hndl, 1.0);
          op_stat_write (global_packets_sent_hndl, 0.0);

          FOUT;
          }

static void  manet_rpg_received_stats_update (double size)
          {
          /** This function updates the local and global statictics                    **/
          /** related with packet sending.                                                                                **/
          FIN (manet_rpg_received_stats_update (<args>));


          /* Update the local and global sent statistics.                                         */
          op_stat_write (local_bits_received_hndl,     size);
          op_stat_write (local_bits_received_hndl,     0.0);

          op_stat_write (local_packets_received_hndl,  1.0);
          op_stat_write (local_packets_received_hndl,  0.0);

          op_stat_write (global_bits_received_hndl,    size);
          op_stat_write (global_bits_received_hndl,    0.0);
          op_stat_write (global_packets_received_hndl, 1.0);
          op_stat_write (global_packets_received_hndl, 0.0);



          FOUT;
          }


static void   manet_rpg_received_stats_real_update (double size)
          {
          /** This function updates the local and global statistics for Video stream             **/
          /** related with packet sending.                                                                                **/
          FIN (manet_rpg_received_stats_real_update (<args>));

          /* Update the local and global sent statistics.                                         */

          op_stat_write (local_bits_received_real_hndl,     size);
          op_stat_write (local_bits_received_real_hndl,     0.0);
          op_stat_write (local_packets_received_real_hndl ,  1.0);
          op_stat_write (local_packets_received_real_hndl ,  0.0);


          FOUT;
          }

static void  manet_rpg_received_stats_back_update (double size)
          {
          /** This function updates the local and global statistics for Background Traffic              **/
          /** related with packet sending.                                                                                **/
          FIN (manet_rpg_received_stats_back_update (<args>));

          /* Update the local and global sent statistics.                                         */

          op_stat_write (local_bits_received_back_hndl,     size);
          op_stat_write (local_bits_received_back_hndl,     0.0);
          op_stat_write (local_packets_received_back_hndl,  1.0);
          op_stat_write (local_packets_received_back_hndl,  0.0);



          FOUT;
          }


static void  manet_rpg_packet_flow_info_read (void)
          {
          int                                                     i, row_count;
          char                                          temp_str [256];
          char                                          interarrival_str [256];
          char                                          pkt_size_str [256];
          char                                          log_message_str [256];
          Objid                                         trafgen_params_comp_objid;
          Objid                                         ith_flow_attr_objid;
          double                                        start_time;
          InetT_Address*                      dest_address_ptr;
          InetT_Address                       dest_address;
          static OmsT_Log_Handle    manet_traffic_generation_log_handle;
          static Boolean                      manet_traffic_generation_log_handle_not_created = OPC_TRUE;
```

```
/** Read in the attributes for each flow                **/
FIN (manet_rpg_packet_flow_info_read (void));

/* Get a handle to the Traffic Generation Parameters compound attribute.*/
op_ima_obj_attr_get (my_objid, "Traffic Generation Parameters", &trafgen_params_comp_objid);

/* Obtain the row count
        */
row_count = op_topo_child_count (trafgen_params_comp_objid, OPC_OBJTYPE_GENERIC);

/* If there are no flows specified, exit from the function                              */
if (row_count == 0)
        {
        FOUT;
        }

/* Allocate enough memory to hold all the information.                                  */
manet_flow_info_array = (ManetT_Flow_Info*) op_prg_mem_alloc (sizeof (ManetT_Flow_Info) * row_count);

/* Loop through each row and read in the information specified.                         */
for (i = 0; i < row_count; i++)
        {
        /* Get the object ID of the associated row for the ith child.          */
        ith_flow_attr_objid = op_topo_child (trafgen_params_comp_objid, OPC_OBJTYPE_GENERIC, i);

        /* Read in the start time and stop times */
        op_ima_obj_attr_get (ith_flow_attr_objid, "Start Time", &start_time);
        op_ima_obj_attr_get (ith_flow_attr_objid, "Stop Time", &manet_flow_info_array [i].stop_time);

        /* Read in the packet inter-arrival time and packet size                        */
        op_ima_obj_attr_get (ith_flow_attr_objid, "Packet Inter-Arrival Time", interarrival_str);
        op_ima_obj_attr_get (ith_flow_attr_objid, "Packet Size", pkt_size_str);

        /* Set the distribution handles            */
        manet_flow_info_array [i].pkt_interarrival_dist_ptr = oms_dist_load_from_string (interarrival_str);
        manet_flow_info_array [i].pkt_size_dist_ptr = oms_dist_load_from_string (pkt_size_str);

        /* Read in the destination IP address.                                                          */
        op_ima_obj_attr_get (ith_flow_attr_objid, "Destination IP Address", temp_str);

        if (strcmp (temp_str, "auto assigned"))
                {
                /* Explicit destination address has been assigned    */
                dest_address = inet_address_create (temp_str, InetC_Addr_Family_Unknown);
                manet_flow_info_array [i].dest_address_ptr = inet_address_create_dynamic (dest_address);
                inet_address_destroy (dest_address);
                }
        else
                {
                /* The destination address is set to auto-assigned    */
                /* Choose a random destination address                                  */
                dest_address_ptr = manet_rte_dest_ip_address_obtain (iface_info_ptr);
                manet_flow_info_array [i].dest_address_ptr = dest_address_ptr;
                if (dest_address_ptr == OPC_NIL)
                        {
                        /* Write a simulation log                */
                        char                                            my_node_name [64];

                        op_ima_obj_attr_get_str (my_node_objid, "name", 64, my_node_name);

                        if (manet_traffic_generation_log_handle_not_created)
                                {
                                manet_traffic_generation_log_handle = oms_log_handle_create
(OpC_Log_Category_Configuration, "MANET", "Traffic Setup", 32, 0,
                                        "WARNING:\n"
                                        "The following list of MANET traffic sources do not have valid
destinations:\n\n"
                                        "Node name
        Traffic row index\n"
                                        "-----------------                                      ------------------
--\n",
                                        "SUGGESTION(S):\n"
                                        "Make sure that 1 or more prefixes are available for the MANET source to
send its traffic to.\n\n"
                                        "RESULT(S):\n"
                                        "No Traffic will be generated at the MANET source.");

                                manet_traffic_generation_log_handle_not_created = OPC_FALSE;
                                }

                        sprintf (log_message_str, "%s\t\t\t\t\t\t%d\n", my_node_name, i);
                        oms_log_message_append (manet_traffic_generation_log_handle, log_message_str);
                        }
                }
```

```
                    /* Schedule an interrupt for the start time of this flow.                          */
                    /* Use the row index as the interrupt code, so that we can handle                 */
                    /* the interrupt correctly.
                    */
                    op_intrpt_schedule_self (start_time, i);
                    }

          FOUT;
          }

static void
manet_rpg_generate_packet (void)
          {
          int                                       row_num;
          double                          schedule_time;
          double                          pksize;
          double                          next_pkt_interarrival;
  Packet*                       pkt_ptr;
          InetT_Address         src_address;
          InetT_Address*        src_addr_ptr;
          InetT_Address*        copy_address_ptr;
          double   tf_scaling_factor = 1;

/****************************Alex Egaji****************************/

         int           pksize1;
         IplImage        *image4;



          /** A packet needs to be generated for a particular flow            **/
          /** Generate the packet of an appropriate size and send it          **/
          /** to IP. Also schedule an event for the next packet                        **/
          /** generation time for this flow.                                                         **/
          FIN (manet_rpg_generate_packet (void));

          /* Identify the right packet flow using the interrupt code          */
          row_num = op_intrpt_code ();

          /* If no destination was found, exit                                                  */
          if (manet_flow_info_array [row_num].dest_address_ptr == OPC_NIL)
                    FOUT;

          /* Schedule a self interrupt for the next packet generation         */
          /* time. The netx packet generation time will be the current*/
          /* time + the packet inter-arrival time. The interrupt code         */
          /* will be the row number.                                                                                                     */


          tf_scaling_factor = Oms_Sim_Attr_Traffic_Scaling_Get ();
          next_pkt_interarrival = (oms_dist_nonnegative_outcome (manet_flow_info_array [row_num].pkt_interarrival_dist_ptr)) / (tf_scaling_factor);
          schedule_time = op_sim_time () + next_pkt_interarrival;

          /* Schedule the next inter-arrival if it is less than the           */
          /* stop time for the flow                                                                                        */
          if ((manet_flow_info_array [row_num].stop_time == -1.0) ||
                    (schedule_time < manet_flow_info_array [row_num].stop_time))
                    {
                    op_intrpt_schedule_self (op_sim_time () + next_pkt_interarrival, row_num);
            }



          /* Create an unformatted packet        */
          pksize = (double) ceil (oms_dist_outcome (manet_flow_info_array [row_num].pkt_size_dist_ptr));

          /* Size of the packet must be a multiple of 8. The extra bits will not be modeled                    */
          pksize = pksize - fmod (pksize, 8.0);

          //pkt_ptr = op_pk_create (pksize);
          pkt_ptr = op_pk_create (pksize);


          /*********************************Alex Egaji*****************************************/


                    if (row_num==0 ){



      /* create a formatted packet */
          pkt_ptr = op_pk_create_fmt ("webcam_pk");

           cap = cvCreateCameraCapture(0);   // Capture the Video Stream
```

```
                image3= cvCreateImage(cvSize(image1->width/4,image1->height/4),8,3); /*Resize frame size from 640 x 480 to 320 x 240*/

                image1  = cvQueryFrame(cap);

                cvResize(image1,image3,0);

                cvShowImage("Node 1 (Source)",image3);

                   image_gray=cvCreateImage(cvGetSize(image3),IPL_DEPTH_8U,1);
                  cvCvtColor(image3,image_gray,CV_RGB2GRAY);

                   c = cvWaitKey(1);

                /* create a formatted packet */

pkt_ptr = op_pk_create_fmt ("webcam_pk");

/* allocate a new data structure */
ds_ptr = (Custom_pk *) op_prg_mem_alloc (sizeof (Custom_pk));

/* fill in certain data structure values */

ds_ptr->image2=image3;

op_pk_nfd_set (pkt_ptr,"TOS", 0);
op_pk_nfd_set (pkt_ptr, "Data", ds_ptr, op_prg_mem_copy_create,op_prg_mem_free, sizeof (Custom_pk)); // Encapsulate image in OPNET custom packet

                 }
else{

    pksize1= op_pk_total_size_get (pkt_ptr);

   pkt_ptr = op_pk_create_fmt ("backgroundtraffic_pk");
   op_pk_nfd_set (pkt_ptr,"TOS",1);
                op_pk_total_size_set (pkt_ptr, pksize1);

                }

/*******************************************************************************************/


                /* Update the packet sent statistics       */
                manet_rpg_sent_stats_update (pksize);

                /* Make a copy of the destination address               */
                /* and set it in the ICI for ip_encap                   */
                copy_address_ptr = manet_flow_info_array [row_num].dest_address_ptr;

                /* Set the destination address in the ICI */
                op_ici_attr_set (ip_encap_req_ici_ptr, "dest_addr", copy_address_ptr);

                /* Set the source address of this node based on the  */
                /* IP address family of the destination                                      */
                if (inet_address_family_get (copy_address_ptr) == InetC_Addr_Family_v6)
                        {
                        /* The destination is a IPv6 address     */
                        /* Set the IPv6 source address                                       */
                        src_address = inet_rte_intf_addr_get (iface_info_ptr, InetC_Addr_Family_v6);
                        }
                else
                        {
                        /* The destination is a IPv4 address     */
                        /* Set the IPv4 source address                                       */
                        src_address = inet_rte_intf_addr_get (iface_info_ptr, InetC_Addr_Family_v4);
                        }

                /* Create the source address               */
                src_addr_ptr = inet_address_create_dynamic (src_address);

                /* Set the source address in the ICI      */
                op_ici_attr_set (ip_encap_req_ici_ptr, "src_addr", src_addr_ptr);
                        /* Install the ICI */
op_ici_install (ip_encap_req_ici_ptr);

                /* Send the packet to ip_encap            */

                /* Since we are reusing the ici we should use                                       */
                /* op_pk_send_forced. Otherwise if two flows generate a          */
                /* packet at the same time, the second packet genration will*/
                /* overwrite the ici before the first packet is processed by*/
                /* ip_encap.
                */
                op_pk_send_forced (pkt_ptr, outstrm_to_ip_encap);

                //op_pk_send_forced (pkptr, outstrm_to_ip_encap);


                /* Destroy the temorpary source address                */
                inet_address_destroy_dynamic (src_addr_ptr);

                 cvReleaseCapture(&cap);

                /* Uninstall the ICI */
                op_ici_install (OPC_NIL);


                FOUT;
                }

static void
manet_rpg_packet_destroy (void)
                {
                Packet*                                 pkptr;
                double                                  delay;
                double                                  pk_size,pk_size1;
   Custom_pk *lex;
                                        int             cl;
   IplImage  *image4;


                /** Get a packet from IP and destroy it. Destroy     **/
                /** the accompanying ici also.                                                      **/
                FIN (manet_rpg_packet_destroy (void));



                /* Remove the packet from stream       */
                pkptr = op_pk_get (instrm_from_ip_encap);

                /* Update the "Traffic Received" statistics             */
                pk_size = (double) op_pk_total_size_get (pkptr);

/**********************Alex Egaji****************************************/

                 op_pk_nfd_access (pkptr, "TOS", &cl);


                if (cl == 0){

                op_pk_nfd_access (pkptr, "Data", &lex);

               image4=lex->image2;

              cvNamedWindow("Node 4 (Destination)",CV_WINDOW_AUTOSIZE);
              cvShowImage("Node 4 (Destination)",image4);

                /* Compute the delay for video traffic */
                delay = op_sim_time () - op_pk_creation_time_get (pkptr);

                /* Update the "Delay" statistic            */
                op_stat_write (local_delay_real_hndl, delay);

                pk_size = (double) op_pk_total_size_get (pkptr);

                manet_rpg_received_stats_real_update (pk_size);
                }


                else if (cl==1) {

                /* Compute the delay for  background traffic          */
                delay = op_sim_time () - op_pk_creation_time_get (pkptr);

                /* Update the "Delay" statistic            */
                op_stat_write (local_delay_back_hndl, delay);

                pk_size = (double) op_pk_total_size_get (pkptr);

                manet_rpg_received_stats_back_update (pk_size);

                }
/*****************************************************************************/

/* Update the "Traffic Received" statistics             */
                pk_size = (double) op_pk_total_size_get (pkptr);
```

```
/* Update the statistics for the packet received         */
manet_rpg_received_stats_update (pk_size);

/* Compute the delay       */
delay = op_sim_time () - op_pk_creation_time_get (pkptr);

/* Update the "Delay" statistic             */
op_stat_write (local_delay_hndl, delay);
op_stat_write (global_delay_hndl, delay);

op_ici_destroy (op_intrpt_ici ());
op_pk_destroy (pkptr);

FOUT;
}
```

Appendix H :Publications

This work has lead to a number of publications, namely:

O. A. Egaji, A. Griffiths, M. S. Hasan, and H. Yu, 'Optimisation of Delay for Multimedia Applications in a Wireless Network Control System ', (Accepted) IEEE sponsored Science and Information Conference (SAI), London, 2014

O. A. Egaji, A. Griffiths, M. S. Hasan, and H. Yu, 'Development of a realistic simulation model for IEEE 802.11n based on Experimental Results', in The 7 th International Conference on Software, Knowledge, Information Management and Applications, Chiang Mai, Thailand, 2013.

O. A. Egaji, A. Griffiths, M. S. Hasan, and H. Yu, 'Fuzzy logic based packet scheduling algorithm for Mobile Ad-Hoc Network with a realistic propagation model', Autom. Comput. ICAC 2013 19th Int. Conf. On, pp. 1-6, Sep. 2013.

O. A. Egaji, A. Griffiths, M. S. Hasan, and H. Yu "A Comparison of Mamdani and Sugeno Fuzzy Based Packet Scheduler for MANET with a Realistic Wireless Propagation Model". International Journal of Automation and Computing (IJAC). Vol. 12, Issue 1. pp. 1-12, 2015.