FACULTY OF
COMPUTER SCIENCE
AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA

Implementation Of Differentiated Services In UMJaNetSim

TANG GECK HIANG (WEK010281)

Under the Supervision of
Mr. Phang Keat Keong

Moderator
Mr. Ang Tan Fong

SESSION 2003/2004

This project is submitted to the Faculty of Science and Information Technology,
University of Malaya,
in partial fulfillment of the requirement of the Bachelor of Computer Science

ABSTRACT

Deployment of a large number of Internet applications built around the Internet Protocol (IP), require a wide range of Quality of Service (QoS) in terms of throughput, delay and reliability (guarantee to reach the destination without any packet loss). Consequently, there is growing demand for replacing the current best-effort service paradigm with a model in which network traffic will be classified into different traffic classes based on their service needs and the classes will be treated differently.

In this thesis, an existing object oriented and discrete event network simulator, UMJaNetSim is developed to enable the creation of a simulation environment for DiffServ framework. The goal of DiffServ is to define configurable types of packet forwarding that can provide service differentiation for large aggregates of network traffic. The DiffServ being implemented classifies packets into different PHBs and proposes that packets with higher priority will receive relatively better service in any load condition in the network. This simulator intends to show that the proposed scheme improves the performance of existing schemes in terms of packet loss ratio, link utilization, end-to-end delay and throughput.

ACKNOWLEDGEMENT

First of all, I would like to express my utmost gratitude to my supervisor Mr. Phang Keat Keong for his invaluable guidance and support throughout my thesis work. His encouragement to do innovative work inspired me. His able supervision prevented me from digressing from the main area of focus for my thesis and thus helping me in timely completion of the thesis. I also appreciate his patience, support and understanding when things went wrong.

I will always cherish the time I spend with my project members in the Network Research Laboratory especially Chia Kai Yan, Lim Lee Wen, Andrew Chiam, Au Yee Boon, Chee Wai Hong, Chan Chin We and Malini. They are willing to share their knowledge throughout the duration of project. My discussion with them on various topics of networking helped me to enhance my knowledge and understanding in this area.

The members of the academic staffs in FSKTM have been very influential in laying a strong foundation in my knowledge in Computer Science. I am especially very grateful to Mr. Ling Teck Chaw. He is the source of inspiration to me. He also has provided much guidance in this research.

Last but not the least, I also would like to express my deepest appreciation to all my dear family members and friends for their moral and mental support during the period of my thesis work. Life would not have been easy without them.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

Today the Internet hosts a wide range of applications and users with differing requirements. Therefore, this project is developed using existing network simulator to offer proper quality of service (QoS) for all needs.

1.1 Project Overview

IP networks are destined to become the ubiquitous global communication infrastructure. An increasing number of different applications are continually conveyed by them causing a fragmentation of performance and service requirements. Continuous efforts have been made to develop a number of new technologies for enhancing Quality of Service capabilities.

In early 90's, the Integrated Service Model (IntServ) was proposed which provides an integrated infrastructure to handle conventional Internet applications and those QoS-sensitive applications together. IntServ uses resource ReSerVation Protocol (RSVP) as its signaling protocol. Although IntServ/RSVP can provide QoS guarantees to applications, it has a scalability problem since each router in the model has to keep track of individual flows. To address the scalability issue, a new core stateless model, called Differentiated Service Model (DiffServ) was proposed and has become a popular research topic as a low-cost method to bring QoS to today's Internet. The DiffServ architecture is the traffic management scheme defined by IETF to provide scalable services differentiation on the Internet. That's why DiffServ is chosen as main topic of development for this thesis.

1.2 Project Objectives

The main objective of this project is to develop UM Java Network Simulator (UMJaNetSim) with the implementation of DiffServ. By doing so, it is possible to provide differentiated classes of service to the traffic of network simulator and produce a small, well defined set of building blocks from which a variety of services may be constructed. The mechanism is that a small bit-pattern in each packet, in the IPv4 TOS octet or the IPv6 Traffic Class octet, is used to mark a packet to receive a particular forwarding treatment, or per-hop behavior, at each network node.

From other point of view, reduces the burden on network devices and easily scales as the network grows are other objectives to develop this project. It is aim to alleviate bottlenecks through efficient management of network resources

1.3 Project Scope

The thesis undertakes a detailed study of IP QoS with an emphasis in creating a simulation environment for the testing and evaluation of various IP QoS architectures. Using this simulation environment, a traffic engineering enhancement to the differentiation of services is proposed and evaluated. The objectives of this research are summarized as the following:

- Develop Differentiated Services (DiffServ) network simulator.
- Allow the user to classify the incoming packets to different PHBs and drop precedence.
- Scheduling packets using Weighted Round Robin (WRR) mechanism.

- Show the simulation result and the traffic parameters.
- A GUI interface for user to perform action.

1.4 Project Schedule

The project schedule is the operating timetable of the project. It serves as the fundamental basic of monitoring and controlling project activity. By using Gantt chart, a schedule of earliest possible start and finish times for the activities is given that will meet the earliest possible project completion date.

Below is a Gantt chart on the development phase scheduled along the intended time frame for each phase of the system.

Project Schedule

ID	Task Name	Start	Finish	Duration
1	Project Definition	16/06/2003	03/07/2003	16d
2	Literature Review	04/07/2003	24/07/2003	18d
3	Diffserv Research	25/07/2003	07/08/2003	12d
4	System Analysis	08/08/2003	23/08/2003	14d
5	System Design	25/08/2003	10/09/2003	15d
6	Implementation	11/09/2003	19/12/2003	86d
7	System Testing	22/12/2003	04/02/2004	39d
8	Documentation	24/06/2003	13/02/2004	202d

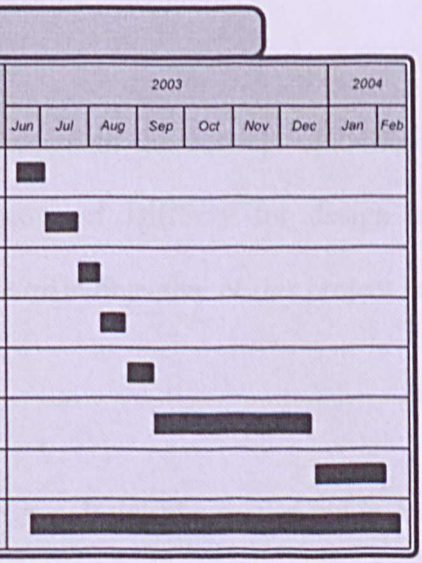


Figure 1.1: WXES 3181 and WXES 3182 project schedule.

1.5 Report Layout

This project proposal report consists of five chapters. The purpose of this layout is to give overview of the major phases involved during development of the project. Below is the report layout:

Chapter 1 presents an introduction to the objectives, scope, schedule and organization of report which associate with this project. It is quite important as what the system is going to do and the boundary of project will be clearly stated. Besides that, it is helpful to monitor the progress of project in terms of deliverable in time.

To start doing this project, the step of review the current technologies can't be missed out. That's why Chapter 2 comes out with literature review. In chapter intend to review current technologies which related to this project as well as the existing simulators.

To implement this project, there is a must to fully understand the concept of DiffServ. Therefore, chapter 3 is arranged to describe the theory of DiffServ for design and implementation phases. This is an essential part since the main objective of this project is to simulate the real world using DiffServ.

Chapter 4 is used to depict the analysis done over this project. It must be carried out in order to produce a successful project. The analyzed topics include software and hardware selection, simulation approach, architecture of UMJaNetSim, system requirement and so on.

After analyze relevant topic, the design phase can be started which is defined in Chapter 5. Appropriate and carefully design is needed to ensure the system operate in the desired manner. Each important function that designed for the system will be characterized here also.

Now it is time to implement the DiffServ into UMJaNetSim. Chapter 6 intends to discuss about how the project is going to be implemented. All related class files will also discussed in this chapter.

In order to make sure the simulator is running properly, component testing and system testing are done in Chapter 7. The way and topology used to test the simulation is explained through out this chapter. Even the simulation results are explained and presented here.

The last chapter for this project document, which is Chapter 8, is conclusion. The system strengths, system limitations and future enhancements have been noted down in this chapter.

CHAPTER 2 LITERATURE REVIEW

Internet traffic has increased at an exponential rate recently and shows no signs of slowing down. In the mean while, some applications raise requirements for underlying network infrastructure to provide Quality of Service (QoS) guarantees. It is big challenges to current Internet, since the current Internet provides only one simple service class to all uses with respect to QoS, which is best-effort datagram delivery. Best-effort datagram delivery cannot provide any service quality guarantees. The gap between QoS provisioning and demanding is even enlarged.

2.1 TCP/IP

Transmission Control Protocol (TCP) and Internet Protocol (IP) were developed by a Department of Defense (DOD) research project to connect a number of different networks designed by different vendors into a network of networks (the "Internet"). Transmission Control Protocol/Internet Protocol (TCP/IP) is a protocol suite that defines how all transmissions are exchanged across the Internet. It has been active use for many years and has demonstrated its effectiveness on a worldwide scale. It is the basic communication language or protocol of the Internet. It can also be used as a communications protocol in a private network (for examples an intranet or an extranet).

As with all other communications protocol, TCP/IP is composed of layers:

- IP is responsible for moving packet of data from node to node. IP forwards each packet based on a four byte destination address (the IP number). The Internet authorities assign ranges of numbers to different organizations. The

organizations assign groups of their numbers to departments. IP operates on gateway machines that move data from department to organization to region and then around the world.

- TCP is responsible for verifying the correct delivery of data from client to server. Data can be lost in the intermediate network. TCP adds support to detect errors or lost data and to trigger retransmission until the data is correctly and completely received.
- Socket is a name given to the package of subroutines that provide access to TCP/IP on most systems.

2.1.1 TCP/IP and OSI

TCP was developed before the Open Systems Interconnection (OSI) model. Therefore, the layers in the TCP/IP protocol do not match exactly with those in the OSI model. The TCP/IP protocol is made of five layers: physical, data link, network, transport, and application. The application layer in TCP/IP can be equated with the combination of session, presentation, and application layers of the OSI model. Figure 2.1 shows TCP/IP in relation to the OSI model.

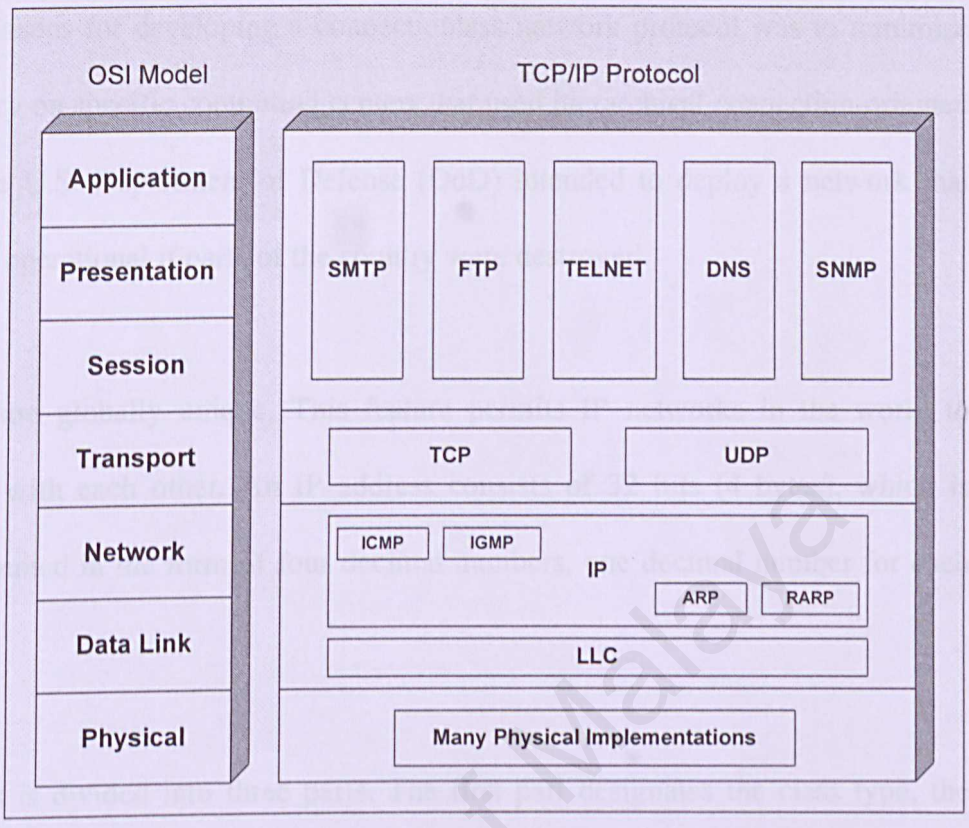


Figure 2.1: Relationship of the TCP/IP and OSI model.

2.1.2 Network Layer

At the network layer, TCP/IP supports the IP. IP in turn, contains four supporting protocols, which are ARP, RARP, ICMP and IGMP.

IP is the transmission mechanism used by TCP/IP protocols. It is an unreliable, best-effort and connectionless packet delivery protocol. Here best-effort means that the packets sent by IP may be lost, out of order, or even duplicated, but IP will not handle these situations. It is up to the higher-layer protocols to deal with these situations.

One of the reasons for developing a connectionless network protocol was to minimize the dependency on specific computing centers that used hierarchical connection-oriented networks. The U.S. Department of Defense (DoD) intended to deploy a network that would still be operational if parts of the country were destroyed.

IP addresses are globally unique. This feature permits IP networks in the world to communicate with each other. An IP address consists of 32 bits (4 bytes), which is usually represented in the form of four decimal numbers, one decimal number for each byte.

An IP address is divided into three parts. The first part designates the class type, the second part designates the network address (or netid), and the third part designates the host address (or hostid). An IP address belongs to one of five classes depending on the value of its first four bits (A fifth class, class E, is not commonly used.). The different classes are designed to meet the needs of different types of organizations. Figure 2.2 gives a clear picture of the structure of each IP addresses class.

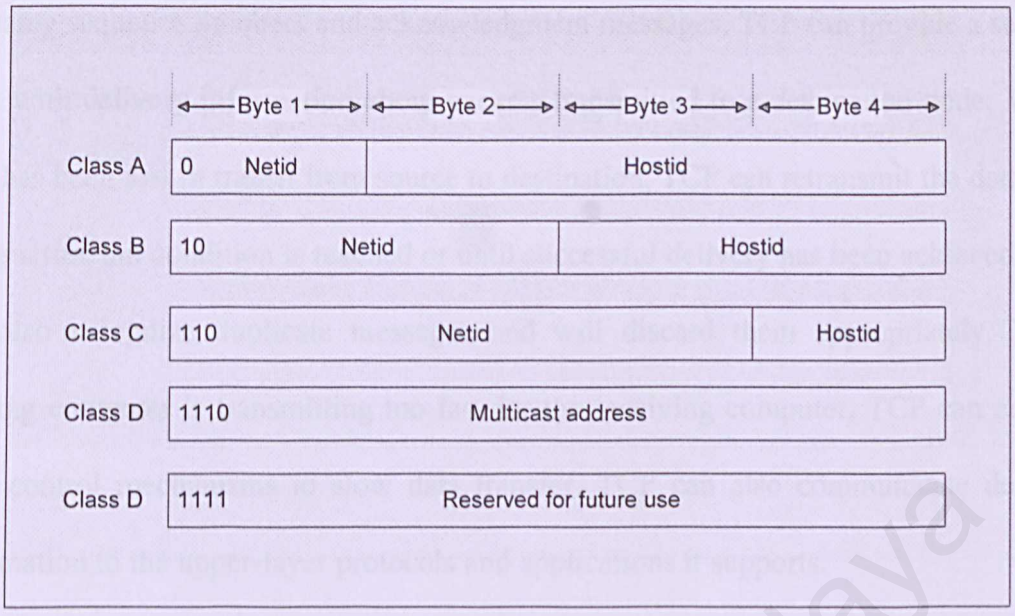


Figure 2.2: The structure of each IP addresses class.

2.1.3 Transport Layer

The transport layer is represented in TCP/IP by two protocols: TCP and UDP.

TCP

The TCP provides full transport layer services to applications. TCP is a reliable connection-oriented transport port-to-port protocol that sends data as an unstructured stream of bytes. As a connection-oriented service, TCP is responsible for the reliable delivery of the entire stream of bits contained in the message originally generated by the sending application. Provided error detection and retransmission of damaged frames ensures reliability (all segments must be received and acknowledged before the transmission is considered complete).

By using sequence numbers and acknowledgment messages, TCP can provide a sending node with delivery information about packets transmitted to a destination node. Where data has been lost in transit from source to destination, TCP can retransmit the data until either a timeout condition is reached or until successful delivery has been achieved. TCP can also recognize duplicate messages and will discard them appropriately. If the sending computer is transmitting too fast for the receiving computer, TCP can employ flow control mechanisms to slow data transfer. TCP can also communicate delivery information to the upper-layer protocols and applications it supports.

UDP

User Datagram Protocol (UDP) is the simpler protocol among the two standard transport protocol of TCP/IP. It is a connectionless datagram delivery service that does not guarantee delivery. UDP provides only the basic functions needed for end-to-end delivery of a transmission. It contains only a checksum and does not provide any sequencing or reordering functions. Therefore, it cannot specify the damaged packet when reporting an error.

If the application developer chooses UDP instead of TCP, then the application is talking almost directly with IP. UDP takes messages from application process, attaches source and destination port number fields for the multiplexing/demultiplexing service, adds two other fields of minor importance, and passes the resulting "segment" to the network layer. The network layer encapsulates the segment into an IP datagram and then makes a best-effort attempt to deliver the segment to the receiving host. If the segment arrives at the

receiving host, UDP uses the port numbers and the IP source and destination addresses to deliver the data in the segment to the correct application process.

UDP is no handshaking between sending and receiving transport-layer entities before sending a segment. Therefore, UDP is said to be connectionless and unreliable. DNS is an example of an application layer protocol that uses UDP.

Even though TCP provides a reliable data transfer service and UDP does not, an application developer would ever choose to build an application over UDP rather than over TCP as many applications are better suited for UDP for the following reasons:

- No connection establishment. TCP uses a three-way handshake before it starts to transfer data but UDP just blasts away without any formal preliminaries. Thus UDP does not introduce any delay to establish a connection. This is probably the principle reason why DNS runs over UDP rather than TCP -- DNS would be much slower if it ran over TCP. HTTP uses TCP rather than UDP, since reliability is critical for Web pages with text.
- No connection state. Connection state includes receive and send buffers, congestion control parameters, and sequence and acknowledgment number parameters. This state information is needed to implement TCP's reliable data transfer service and to provide congestion control. UDP does not maintain connection state and does not track any of these parameters. For this reason, a server devoted to a particular application can typically support many more active clients when the application runs over UDP rather than TCP.

- Small segment header overhead. The TCP segment has 20 bytes of header overhead in every segment, whereas UDP only has 8 bytes of overhead.

- Unregulated send rate. TCP has a congestion control mechanism that throttles the sender when one or more links between sender and receiver becomes excessively congested. This throttling can have a severe impact on real-time applications, which can tolerate some packet loss but require a minimum send rate. On the other hand, the speed at which UDP sends data is only constrained by the rate at which the application generates data, the capabilities of the source (CPU, clock rate, etc.) and the access bandwidth to the Internet. We should keep in mind, however, that the receiving host does not necessarily receive all the data - when the network is congested, a significant fraction of the UDP transmitted data could be lost due to router buffer overflow. Thus, the receive rate is limited by network congestion even if the sending rate is not constrained.

2.1.4 Application Layer

The TCP/IP application layer is equivalent to the combined session, presentation and application layer of the OSI model. This means that all of the functionalities associated with those three layers are handled in one single layer, the application layer.

The most widely known and implemented TCP/IP application layer protocols are listed below:

- File Transfer Protocol (FTP). Performs basic interactive file transfers between hosts.
- Telnet. Enables users to execute terminal sessions with remote hosts.
- Simple Mail Transfer Protocol (SMTP). Supports basic message delivery services.
- HyperText Transfer Protocol (HTTP). Supports the low-overhead transport of files consisting of a mixture of text and graphics. It uses a stateless, connection- and object-oriented protocol with simple commands that support selection and transport of objects between the client and the server.
- Domain Name Service (DNS). Also called name service; this application maps IP addresses to the names assigned to network devices.
- Routing Information Protocol (RIP). Routing is central to the way TCP/IP works. RIP is used to exchange routing information by network devices.
- Simple Network Management Protocol (SNMP). A protocol that is used to collect management information from network devices.

2.2 *QoS*

Quality of Services (QoS) refers to the capability of a network to provide better service to selected network traffic over various technologies, including Frame Relay, Asynchronous Transfer Mode (ATM), Ethernet and 802.1 networks, SONET, and IP-routed networks that may use any or all of these underlying technologies. The primary goal of QoS is to provide priority including dedicated bandwidth, controlled jitter and latency (required by some real-time and interactive traffic), and improved loss

characteristics. Also important is making sure that providing priority for one or more flows does not make other flows fail (Cisco Systems, 2003).

Many different approaches for providing QOS have been proposed and implemented. Unfortunately it is often difficult or even impossible for IP to utilize the QoS capabilities of the underlying technology unless a QoS framework is built into IP itself. This is why IP QoS is needed.

2.3 *Best Effort Service Model*

The basic QoS model of the Internet is called Best Effort because the network tries to transmit as many packets as possible and as rapidly as possible. The network doesn't care who wrote the packets or what information the packets contain and that how urgent the data in packets are. Best effort does not give any guarantees because it offers uniform treatments to every packet. That means that the packets are subject to data loss, data duplication or out-of-order delivery. The TCP protocol solves these problems by assigning a sequence number to all data transmitted in the network and requiring a positive acknowledgment from the receiver.

Traditionally, networks use FIFO (first in first out) queuing to forward traffic, which means that an incoming order on a Web commerce site might be left waiting behind an employee's download of the latest game on the net. The network makes a best effort attempt to retain all traffic in order, but will drop whatever it needs to, including million

dollar transactions as it can't differentiate which of the data is of urgent nature, when it becomes overloaded.

To fix the best effort service model, the technique used is to add more bandwidth the network. Adding more bandwidth is a good solution if it can be economically justified. Unfortunately adding bandwidth is costly and you can never really add enough. The limitation of best effort services model raised the need for a new QoS model.

2.4 *IntServ and RSVP*

Although the TOS field in the IP header has been defined for quite a long time, it is practically ignored in most of the router implementations (Tanenbaum, 1996). The IntServ model (Braden et. al., 1994.) was the first step in altering the best-effort service model in IP. The IntServ model is based on a fundamental philosophy that routers must be able to reserve resources in order to provide special QoS for specific user packet streams, or flow. This in turn requires flow-specific state in the routers. In order to provide different QoS for each flow, the IntServ framework requires that a router should implement three traffic control components:

- Packet scheduler to forward packets in different flows differently.
- Classifier to identify the different flows.
- Admission control to determine whether the requested QoS by a new flow can be granted.

The desired QoS is provided by resource reservation along the path, therefore signaling and state maintaining at each hop is needed. The RSVP (Braden et al, 1997) is protocol used for signaling. Its major features include the use of "soft state" in the routers, receiver-controlled reservation requests and the use of IP multicast for data distribution.

The signaling and reservation of the desired QoS are needed for each flow in the network. A flow is defined as an individual, unidirectional data stream between two applications, and is uniquely identified by the 5-tuple (Source IP address, Source Port, Destination IP Address, Destination Port and the Transport Protocol).

Currently, there are two types of services (other than the default best-effort service) have been implemented:

i) The guaranteed service (Shenker et al, 1997) is intended for applications that require real-time service delivery, with a fixed delay bound.

ii) The controlled-load service (Wroclawski, 1997) is intended for applications that can tolerate some delay but are sensitive to traffic overload conditions.

The drawbacks of this model are:

i) The reservations in each device along the path are "soft", which means that they need to be refreshed periodically; if refresh packets are lost there is a risk of reservation time out.

ii) The need for signaling and maintaining the state of each flow in each router is a strong limitation to scalability

These problems make the IS model less practical in the global Internet, but may be suitable in edge networks.

2.5 *IP Over ATM*

The success of Asynchronous Transfer Mode (ATM) lies largely in its ability to transport legacy data traffic, mostly IP, over its network infrastructure. The complexity of interoperating IP with ATM originates from the following two major differences:

i) Connection oriented versus connectionless

ATM is connection oriented, which means a connection is needed to establish between two parties before they can send data to each other. Once the connection is set up, all data between them is sent along the connection path. On the other hand, IP is connectionless which means that no connection is needed and each IP packet is forwarded on a hop-by-hop basis by routers independently. When IP traffic is needed to transport over an ATM network, it either establishes a new connection on demand between two parties or forwards the data through preconfigured connection or connections. With the first approach, when the amount of data to be transferred is small, the expensive cost of setting up and tearing down a connection is not justified. On the other hand, with the second approach the preconfigured path(s) may not be an optimal path and may become overwhelmed by the amount of data being transferred.

ii) QoS aware versus Best Effort

Quality of Service is an important concept in ATM networks. It includes the parameters like the bandwidth and delay requirements of a connection. Such requirements are included in the signaling messages used to establish a connection. Current IP (IPv4) has no such concepts and each packet is forwarded on a best effort basis by the routers. To take advantage of the QoS guarantees of the ATM networks, the IP protocol need to be modified to include that information.

To run IP on top of ATM networks, we first need to figure out how to relate ATM protocol layers to TCP/IP protocol layers. Two models are proposed which are peer model and the overlay model. Peer model considers the ATM layer a peer networking layer as IP and propose the use of the same addressing scheme as IP for ATM-attached end systems. ATM signaling requests will contain IP addresses and the intermediate switches will route the requests using existing routing protocols like Open Shortest Path First (OSPF). This scheme was rejected because although it simplifies the addressing scheme for end systems, it complicates the design of ATM switches by requiring them to have all the functions of an IP router. Moreover, if the ATM network will also support other networking layer protocols like IPX or Appletalk, the switch has to understand all their routing protocols.

The overlay model, which is finally, adopted views ATM as a data link layer protocol on top of which IP runs. In overlay model, ATM networks will have its own addressing scheme and routing protocols. The ATM address space is not logically coupled with the

IP addressing space and there will be no arithmetic mapping between them. Each end system will typically have an ATM address and an unrelated IP address as well. Since there is no nature mapping between the two addresses, the only way to figure out one from the other is through some addressing resolution protocol.

With overlay model, there are essentially two ways to run IP over ATM. One treats ATM as a LAN and partitions an ATM network into several logical subnets consisting of end systems with the same IP prefix. This is known as Classical IP over ATM. In Classical IP over ATM, end systems in the same logical subnet communicate with each other through end-to-end ATM connections, and like in LAN, Address Resolution Protocol (ARP) servers are used in logical subnets to resolve the IP addresses into ATM addresses. However, traffic between end systems in different logical subnets has to go through a router even though they are attached to the same ATM network. This is not desirable since routers introduce a high latency and become the bandwidth bottleneck.

2.6 MPLS

There is a recent protocol deployment that offers some QoS functionality, based around the idea of label switching. Multiprotocol Label Switching (MPLS) (Rosen et al, 2001) was mainly the result of efforts to effectively match IP over ATM networks. It tries to integrate layer 2 switching and layer 3 datagram forwarding. Within an MPLS network, a label and a Forwarding Equivalence Class (FEC) are assigned to each packet when entering the network, and then all forwarding decisions are based on these values. Packet forwarding is performed on a hop-by-hop basis and Label Switched Routers (LSRs)

simply perform label swapping and take local decisions about the next hop that the packet should be addressed to.

The motivation for MPLS is that it provides for simplified forwarding based on the match of a short label and for efficient explicit routing carried only at the time a label switched path is set up rather than within each packet. MPLS provides traffic engineering by selecting paths chosen by data traffic in order to balance the traffic load within the network. It also provides QoS routing where a route for a particular stream is chosen in response to the QoS required for that stream (Callon, 1999).

MPLS can support QoS on a per-user basis by assigning per-user labels to packets, or on a per-flow basis by detecting and assigning appropriate labels to individual flows. Labels can make use of a Class of Service (CoS) field, which offers the flexibility of choosing between coarse or fine-grained QoS support.

On the other hand, MPLS raises some scalability concerns when it is to support label assignment for short flows and its normal operation can be assured only for well-managed environments due to its complex mechanisms. MPLS is favored by telecommunication operators who were traditionally basing their services on top of ATM but it is doubtful whether it can provide end-to-end QoS solutions across large networks and consequently in the Internet.

2.7 *Existing Network Simulator*

Simulation Modeling is becoming an increasingly popular method for network performance analysis. Software simulator is a valuable tool especially for today's network with complex architectures and topologies. It can be either a general-purpose simulator that enables a wide range of possible simulations or a special purpose simulator that targeting a particular area of research. Designers can test their new ideas and carry out performance related studies, therefore freed from the burden of the "trial and error" hardware implementations. This section intends to review a number of major network simulators by describing their features. The following are some examples of the current network simulator:

- OPNET
- INSANE
- NS
- REAL
- NIST ATM/HFC
- UMJaNetSim

2.7.1 OPNET Network Simulator

Optimized Network Engineering Tool (OPNET) is a commercial network simulator marketed by OPNET, Inc. OPNET was originally developed at MIT and introduced as a commercial network simulator in 1987. OPNET provides a comprehensive development environment for the specification, simulation and performance analysis of communication networks. It can support a large range of communication systems from a

single LAN to global satellite networks. Discrete event simulations are used as the means of analyzing system performance and their behavior. OPNET has full GUI support and consists of three hierarchically related editors, which is network editor, node editor and process editor. The key features of OPNET are summarized here as:

- Modeling and Simulation Cycle

  OPNET provides powerful tools to assist user to go through three out of the five phases in a design circle.

- Hierarchical Modeling

  OPNET employs a hierarchical structure to modeling. Each level of the hierarchy describes different aspects of the complete model being simulated.

- Specialized in communication networks

  Detailed library models provide support for existing protocols and allow researchers and developers to either modify these existing models or develop new models of their own.

- Automatic simulation generation

  OPNET models can be compiled into executable code. An executable discrete-event simulation can be debugged or simply executed, resulting in output data.

However, OPNET is not a fully platform independent simulator as it only supports the Solaris, Window NT and 2000, and the HP-Ux operating systems. It is also costly to use from a financial point of view.

2.7.2 INSANE Network Simulator

Internet Simulated ATM Networking Environment (INSANE) is a network simulator designed to test various IP-over-ATM algorithms with realistic traffic loads derived from empirical traffic measurements. Its ATM protocol stack provides real-time guarantees to ATM virtual circuits by using Rate Controlled Static Priority (RCSP) queuing. The ATM signaling is implemented by using a protocol similar to the Real-Time Channel Administration Protocol (RCAP).

A Tk-based graphical simulation monitor can provide an easy way to check the progress of multiple running simulation processes. Besides that, it is able to support the simulation on a large network, which the result is processed off-line. It is written in C++ object oriented programming approach. Internet protocols supported include large subsets of IP, TCP, and UDP. It works quite well on distributed computing clusters as a large number of sequential processes can easily be run in parallel.

However, this simulator can only works on a few platforms and hardware and this restricted the portability of the simulator. Furthermore, there are a few software requirements to run the simulator and this will be troublesome for the user to use the software.

2.7.3 NS Network Simulator

NS has been developed at the Lawrence Berkeley National Laboratory (LBNL) of the University of California, Berkeley (UCB). It has an extensible background engine implemented in C++ that uses OTcl (an object oriented version of Tcl) as the command and configuration interface. Thus, the entire software hierarchy is written in C++, with OTcl used as a front end. The extensibility of NS makes the tool very dynamic. NS is an event-driven network simulator.

NS is a free network simulation program that can be downloaded from the web and is compatible with a number of operating systems. The tool has substantial functionality for simulating different network topologies and traffic models. NS also has an open architecture that allows users to add new functionality.

NS allows simulation with levels of abstraction, where higher abstraction level (with the use of analytical models) trade off accuracy for performance. Moreover, NS includes a network emulation interface that permits network traffic to pass between real world network nodes and the simulator. This feature, while still under development, may prove useful for diagnostics of protocol implementation errors. Although NS does provide a network animation tool that provides network visualization features but NS does not consists of a GUI for general simulation manipulation and scenario setup.

2.7.4 REAL Network Simulator

REAL is a network simulator originally intended for studying the dynamic behavior of flow and congestion control schemes in packet switched data networks. It is design for testing flow and congestion control mechanisms. It provides users with a way of specifying such networks and to simulate their behavior. It emulates the actions of several well-known flow control protocols (such as TCP) and scheduling disciplines (such as Fair Queuing and Hierarchical Round Robin). Only little effort needed to add new modules to the system due to the modular design of the system.

The simulator takes as input a *scenario*, which is a description of network topology, protocols, workload, and control parameters. It produces as output statistics such as the number of packets sent by each source of data, the queuing delay at each queuing point, and the number of dropped and retransmitted packets.

REAL is written in C language and will run on Digital Unix/ SunOS/ Solaris/ IRIX/ BSD4.3/Ultrix /UMIPS systems on VAX, SUN, SPARC, MIPS, Alpha, SGI or DECstation hardware.

2.7.5 NIST ATM/HFC Network Simulator

The NIST Asynchronous Transfer Mode (ATM) / Hybrid Fiber Coax (HFC) Network Simulator is a simulator that provides a flexible test bed for studying and evaluating the performance of ATM and HFC networks without the expense of building a real network.

The simulator is based on the discrete event approach and uses the C programming language.

NIST ATM/HFC has a well-defined message passing mechanism based on the sending of events among simulation components, handled by an event manager. Although this basic architecture enables a wide range of simulation possibilities, the use of the procedural approach makes the component development process difficult.

The simulator gives user an interactive modeling environment with a graphical user interface which provides the user with a means to display the topology of the network, define the parameters and connectivity of the network, log data from simulation run and to save and load the network configuration. Its GUI uses the X window System running on UNIX based platforms. Since the simulator relies on the X window System for its GUI and UNIX in general, it lacks portability between different platforms.

2.7.6 UMJaNetSim Network Simulator

UM Java Network Simulator is a flexible test bed for studying and evaluating the performance of ATM network without the expenses of building a real network. This simulator is written in Java programming language, which applies object oriented programming approach. It is a tool that give user an interactive modeling environment with a graphical user interface which provides the user with a means to display the topology of the network, define the parameters and connectivity of the network, log data

from simulation run, and to save and load the network configuration. Moreover, it is a cross platform simulator.

2.7.7 Comparison of Existing Network Simulators

After studying a few network simulators, comparison done based on a few features such as discrete-event simulator, object-oriented, GUI, multithreaded, web enabled and platform independent are show in Table 2.1.

Table 2.1: Comparison among several network simulators.

Simulator	Discrete Event Simulation	Object Oriented	GUI	Multithread	Web Enable	Platform Independent
OPNET	√	√	Normal	X	X	X
INSANE	√	√	Poor	X	X	X
NS	√	√	Poor	X	X	X
REAL	√	X	Poor	X	X	X
NIST ATM/HFC	√	X	Normal	X	X	X
UMJaNetSim	√	√	Good	√	X	√

CHAPTER 3 THE DIFFSERV MODEL

Service differentiation is desired to accommodate heterogeneous application requirements and user expectations, and to permit differentiated pricing of Internet service. A "Service" defines some significant characteristics of packet transmission in one direction across a set of one or more paths within a network. These characteristics may be specified in quantitative or statistical terms of throughput, delay, jitter, and/or loss, or may otherwise be specified in terms of some relative priority of access to network resources.

The main purpose of the DiffServ model is to provision end-to-end QoS guarantees by using the service differentiations in the Internet. Unlike the IntServ model, it does not keep soft states for individual flows; instead, it achieves QoS guarantees by a low-cost method, which is aggregating individual flows into several service classes. Therefore, the DiffServ model has a good scalability. In order to achieve scalability, there are two basic characteristics of the DiffServ model:

- It does not rely on per-microflow states in the network. Instead, it utilizes aggregated classification states.
- Complex processing (traffic classification and conditioning) is moved from the core of the network to the edge of the network.

3.1 DiffServ Architectural Model

The architecture model of DiffServ is based on the concept, where traffic entering the network is allocated a classification and possibly conditioned. These actions take place

in the network boundaries and the outcome of this is that the packets entering the network are collected into same behavior aggregates (a collection of packets with the same DS code point crossing a link in a particular direction) that are to be treated in similar manner.

Inside the network the packets are forwarded to their destination on per-hop behavior, which is indicated by DS code point. Basically, each of the packets gets treated only per-hop basis during the forwarding path within the network region.

3.1.1 DS Domain

A DS (DiffServ) domain is a contiguous set of DS nodes, which operate with a common service provisioning policy and set of PHB groups implemented on each node (Black, 1998). The DS domain is the entity that provides a coherent set of PHBs in the network domain. Usually the nodes belonging to a DS domain are under same network administration.

A DS domain has a defined boundary that consists of boundary nodes and interior nodes. DS boundary nodes interconnect the DS domain to other DS or non-DS-capable domains, whilst DS interior nodes only connect to other DS interior or boundary nodes within the same DS domain (Black, 1998). Both of them classify incoming packets and possibly apply configured conditions to their forwarding. Nodes within the DS domain select the forwarding behavior for packets based on their DS code point.

However, the boundary nodes and interior nodes are not totally same. The DS boundary nodes may be required to perform traffic conditioning functions as defined by a Traffic Conditioning Agreement (TCA) between their DS domain and the peering domain, which they connect to but the interior nodes may only perform limited traffic conditioning functions such as DS code point re-marking.

The boundary nodes are further divided into ingress and egress nodes. The ingress nodes are being responsible of packet stream's incoming traffic while the egress nodes are being responsible of outgoing traffic. They are differing from the directions of traffic.

From the administrative point of view, a DiffServ network could consist of multiple DS domains. A set of one or more contiguous DS domains forms a DS region. DS regions support differentiated services along paths, which span the domains within the region. The DS domains may support different DS code point and PHB mappings. To achieve end-to-end QoS guarantees, the negotiation and agreement between these DS domains are needed. In this case, the peering DS domains need to establish a peering Service Level Agreement (SLA) that specifies how the transits between domains are mapped. Figure 3.1 and Figure 3.2 illustrate the hierarchical and graphical view of DS region and DS domain respectively.

The SLA is a service contract between a customer and a service provider that specifies the forwarding services a customer should receive. It may specify the traffic conditioning rules that constitute a Traffic Conditioning Agreement (TCA). The TCA is

an agreement that specifies classifier rules and any corresponding traffic profiles and rules, which will be applied to the traffic streams selected by the classifier.

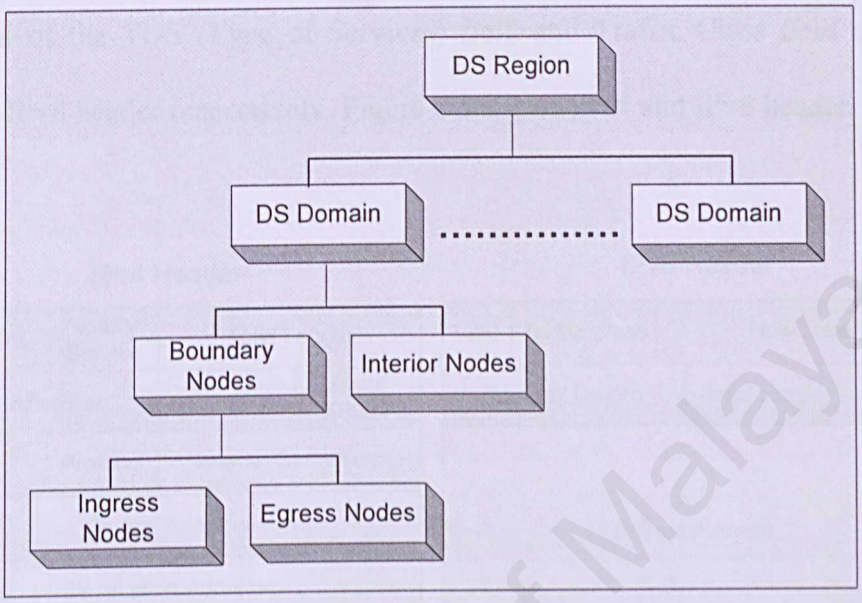


Figure 3.1: Overview of DS region and DS domain.

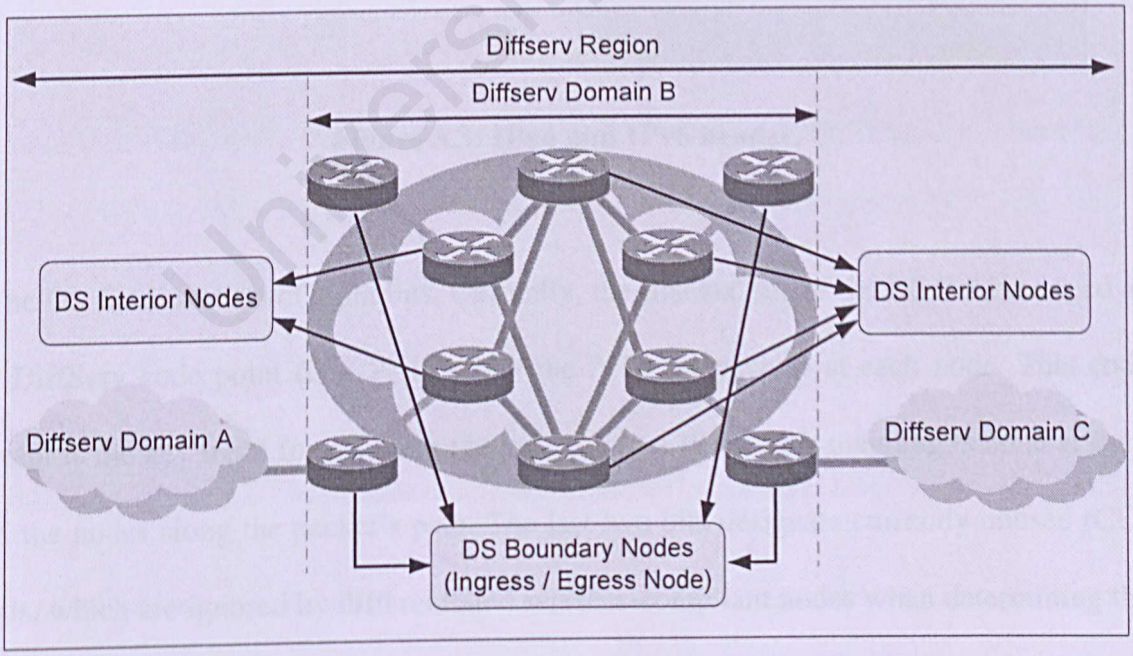


Figure 3.2: Graphical view of DS region and DS domain.

3.1.2 DS Field

The architecture of DiffServ relies on the DS code point to identify the appropriate behavior aggregates. RFC2474 (Nichols et al, 1998a) defines the DS field as the replacement of the TOS (Type of Services) field and Traffic Class field of the IPv4 header and IPv6 header respectively. Figure 3.3 shows IPv4 and IPv6 header.

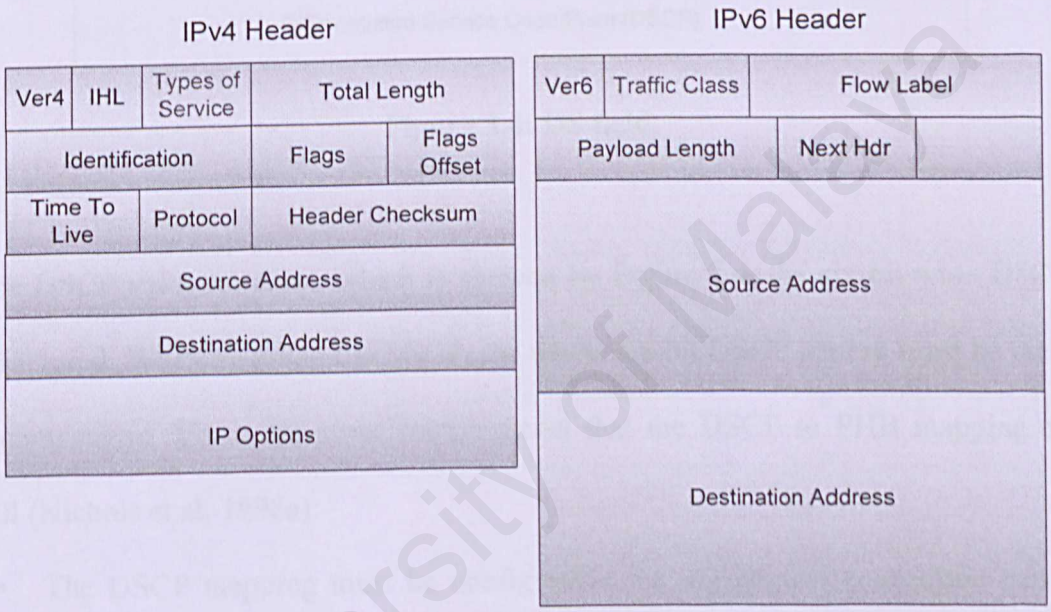


Figure 3.3: IPv4 and IPv6 header.

The DS field consists of eight bits. Currently, the first six bits of the DS field are used as a DiffServ code point (DSCP) to select the PHB for packets at each node. This code point is the key input for mapping the packet into a PHB. This mapping is done at each of the nodes along the packet's path. The last two bits designate currently unused (CU) bits, which are ignored by differentiated services-compliant nodes when determining the per-hop behavior to apply to a received packet. CU bits are reserved for future use.

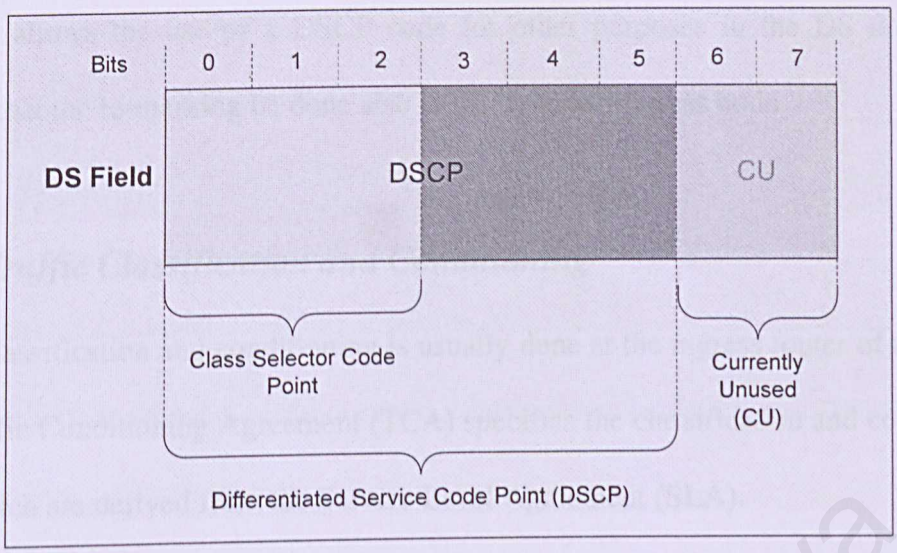


Figure 3.4: DS field.

In the DSCP value notation, which is showed by Figure 3.4, the six-bit wide DSCP is unstructured. When mapping the DSCP, the whole six-bit DSCP pattern must be used in the comparison. There are some requirements that the DSCP to PHB mapping must fulfill (Nichols et al, 1998a):

- The DSCP mapping must be configurable, i.e. the logical connection must be such that it can be configured later on to adapt other mappings.
- There must be a configurable mapping table or similar to support multiple DSCP to PHB mappings.
- Recommended and unique DSCP to PHB mappings should be supported.
- When there is an unrecognized DSCP, it must be treated according to a default PHB.

If DSCP to PHB mappings are not as recommended, there must be a re-marking procedure at the DS domain's boundary ingress nodes. This re-marking procedure

basically allows the use of a DSCP code for other purposes in the DS domain, but requires that the re-marking be done also in the boundary egress node.

3.2 *Traffic Classification and Conditioning*

Traffic classification and conditioning is usually done at the ingress router of a network. The Traffic Conditioning Agreement (TCA) specifies the classification and conditioning rules, which are derived from the Service Level Agreement (SLA).

The packet classification policy identifies the subset of traffic, which may receive a differentiated service by being conditioned and/or mapped to one or more behavior aggregates within the DS domain. In the meanwhile, Traffic conditioning performs metering, shaping, policing and/or re-marking to ensure that the traffic entering the DS domain conforms to the rules specified in the TCA, in accordance with the domain's service provisioning policy. The extent of traffic conditioning required is dependent on the specifics of the service offering, and may range from simple code point re-marking to complex policing and shaping operations.

3.2.1 Traffic Classifiers

Packet classifiers select packets in a traffic stream based on the content of some portion of the packet header. Classifiers are used to steer packets that matching some specified rule to an element of a traffic conditioner for further processing. They must be configured by some management procedure in accordance with the appropriate TCA and authenticate the information which it uses to classify the packet. Every packet is

classified to belong to one of the several classes. Generally there are two types of classifiers (Black, 1998):

i) Behavior Aggregate (BA) Classifier

This is the simplest DiffServ classifier. It uses only the DiffServ Code Point (DSCP) in a packet's IP header to determine the logical output stream to which the packet should be directed.

ii) Multi-Field (MF) Classifier

It classifies packets based on one or more fields in the packet. A common type of MF classifier classifies based on six fields from the IP and TCP or UDP headers. The six fields designate destination address, source address, IP protocol, source port, destination port and DSCP. However, it can also classify based on other fields such as MAC address or other higher layer protocol fields.

3.2.2 Traffic Conditioners

Traffic conditioners perform various functions on the incoming packets based on the associated traffic profiles. A traffic profile specifies the temporal properties (for example transmission rate, burst size etc) of a traffic stream by using the notion of token bucket or other mechanism. A traffic conditioner block which showed by Figure 3.5 may contain the following four elements (Black, 1998):

i) Meters

Traffic meters measure the temporal properties of the stream of packets selected by a classifier against a traffic profile specified in a TCA. A meter passes state information to marker and shaper/dropper to trigger a particular action for each packet which is either in- or out-of-profile (to some extent).

ii) Markers

Packet markers set the DS field of a packet to a particular code point, adding the marked packet to a particular DS behavior aggregate. A packet is said to have "re-marked" when the marker changes its code point.

iii) Shapers

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

iv) Droppers

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

Note that each traffic flow that go through the boundary node, only either policing or shaping is performed as they are excluding each other (Nichols et al, 1998b).

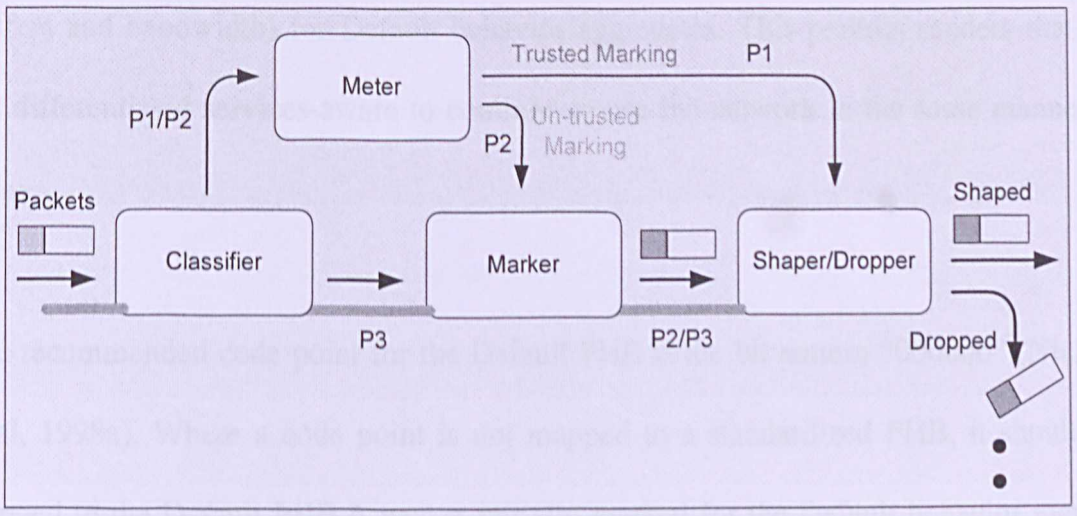


Figure 3.5: DiffServ traffic conditioner block.

3.3 *Per-Hop Behaviors*

A per-hop behavior (PHB) is a description of the externally observable forwarding behavior of a DS node applied to a particular DS behavior aggregate (Black, 1998). In more concrete terms, a PHB refers to the packet scheduling, queuing, policing, or shaping behavior of a node on any given packet belonging to a BA, and as configured by an SLA or policy. There are four standard PHBs are available to construct a DiffServ-enabled network.

3.3.1 Default PHB

A "default" PHB must be available in a DS-compliant node. This is the common, best-effort forwarding behavior available in existing routers. When no other agreements are in place, it is assumed that packets belong to this aggregate (Nichols et al, 1998a). A reasonable implementation of this PHB would be a queuing discipline that sends packets of this aggregate whenever the output link is not required to satisfy another PHB. A mechanism in each node could be enforced to reserve some minimal resources (e.g,

buffers and bandwidth) for Default behavior aggregates. This permits senders that are not differentiated services-aware to continue to use the network in the same manner as today.

The recommended code point for the Default PHB is the bit pattern "000000" (Nichols et al, 1998a). Where a code point is not mapped to a standardized PHB, it should be mapped to the Default PHB.A packet initially marked for the Default behavior may be re-marked with another code point as it passes a boundary into a DS domain so that it will be forwarded using a different PHB within that domain.

3.3.2 Class-Selector PHB

Class-Selector (CS) PHB's objective is to preserve backward compatibility with the IP-Precedence scheme (Cisco System, 2001). It is up to 8 class selectors as the DSCP values of the form 'xxx000', where x is either 0 or 1 are defined (Nichols et al, 1998a). These code points are called Class-Selector Code Points. Use of class selector should yield at to least 2 independently forwarding classes. For example, if x>y, the router should give a higher probability of timely forwarding to CS_x packets than CS_y packets. In this case, packets with CS_6 and CS_7 should receive a better treatment than best-effort traffic.

3.3.3 Expedited Forwarding PHB

The objectives of Expedited Forwarding (EF) PHB (Jacobson et al, 1999) are to build a low loss, low latency, low jitter, assured bandwidth, end-to-end service through DS

domains. Since loss, latency and jitter are all due to the queues traffic experiences while transiting the network, therefore providing low loss, latency and jitter for some traffic aggregate means ensuring that the aggregate sees no or very small queues. The EF PHB provides the first part of the service. The network boundary traffic conditioners described in section before provide the second part.

The EF PHB is defined as a forwarding treatment for a particular DiffServ aggregate where the departure rate of the aggregate's packets from any DiffServ node must equal or exceed a configurable rate (Jacobson et al, 1999). The EF traffic should serve at departure rate independent of the intensity of any other traffic through the node. The network administrator should be able to configure the minimum rate. The recommended code point for EF PHB is 101110 (Jacobson et al, 1999).

A few scheduling and queuing mechanisms can be employed to deliver and implement the EF PHB forwarding behavior. A simple priority queue will give the appropriate behavior as long as there is no higher priority queue that could preempt the EF for more than a packet time at the configured rate. It's also possible to use a single queue in a group of queues serviced by a weighted round robin scheduler where the share of the output bandwidth assigned to the EF queue is equal to the configured rate.

3.3.4 Assured Forwarding PHB

The objective of Assured Forwarding (AF) PHB (Heinanen et al, 1999) is to offer different levels of forwarding assurances for IP packets received from a customer DS

domain. There are 4 defined AF classes where each AF class is in each DS node allocated a certain amount of forwarding resources (buffer space and bandwidth).

Each of the AF class IP packets will be marked with one of three possible drop precedence values. An IP packet that belongs to an AF class x and has drop precedence y is marked with the AF code point AFxy, where 1 <= x <= 4 and 1 <= y <= 3. The format of AF code point is "xxxyy0" (Heinanen et al, 1999). The recommended code points for AF PHBs are depicted in Table 3.1.

Figure 3.6 gives a clearer picture about each AF class and their different drop precedence. When congestion occurred, the drop precedence of a packet will be used to determine the relative importance of the packet within the AF class. Always, a congested DS node will try to protect packets with a lower drop precedence value from being lost by preferably discarding packets with a higher drop precedence value.

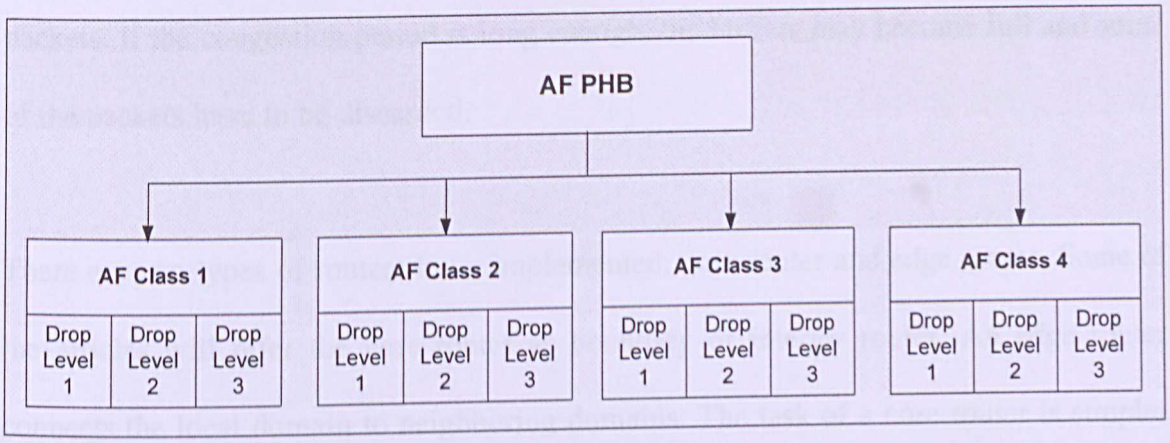


Figure 3.6: The break down of AF PHBs with different drop precedence.

Table 3.1: AF recommended code point with different drop precedence.

Drop Precedence	Class 1	Class 2	Class 3	Class 4
Low Drop Precedence	001010 (AF11)	010010 (AF21)	011010 (AF31)	100010 (AF41)
Medium Drop Precedence	001100 (AF12)	010100 (AF22)	011100 (AF32)	100100 (AF42)
High Drop Precedence	001110 (AF13)	010110 (AF23)	011110 (AF33)	100110 (AF43)

3.4 DiffServ Router

There are several ways for a router to implement differing service, but the most important mechanisms are scheduling and queue management. Both become significant, when the routers are subject to transient congestion, as they often are in the current Internet. During congestion, the router has more packets destined to a particular output port than it is able to forward. This means that the router has to buffer some of the

packets. If the congestion period is long enough, the buffers may become full and some of the packets have to be discarded.

There are two types of routers to be implemented: core router and edge router. Some of the articles will refer the core router as boundary or interior router. An edge router connects the local domain to neighboring domains. The task of a core router is simpler than an edge router. It is edge routers' task to check and enforce that traffic crossing domain boundaries conforms to the existing SLAs. The notions of core and edge router are illustrated in figure below.

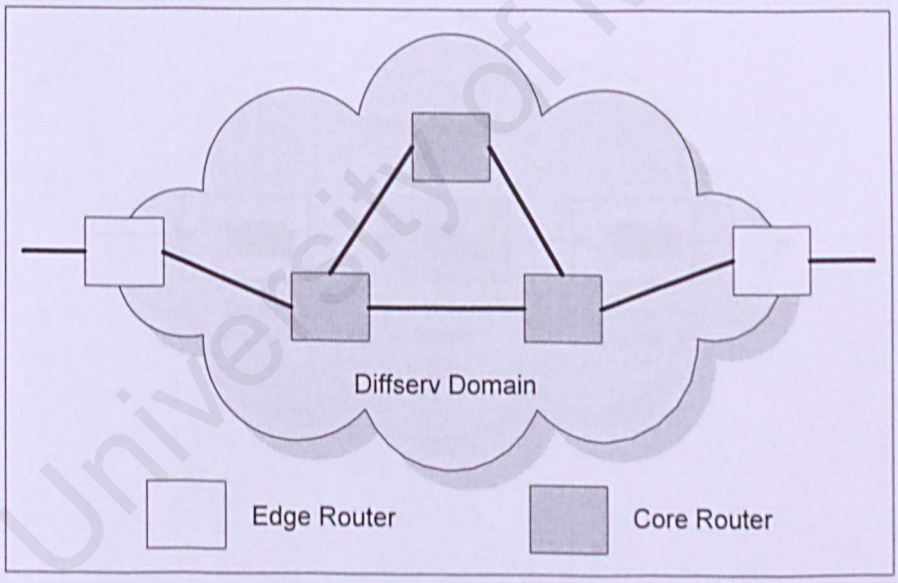


Figure 3.7: Routers in a DiffServ domain.

Core router is responsible to (Pieda et al, 2000):

- Examine incoming packets for the code point marking done on the packet by the edge routers.

- Forward incoming packets according to their markings (core routers provide a reaction to the marking done by edge routers.)

Edge router responsible to (Pieda et al, 2000):

- Examine incoming packets and classify them according to policy specified by the network administrator.
- Mark packets with a code point that reflects the desired level of service.
- Ensure that user traffic adheres to its policy specifications, by shaping and policing traffic.

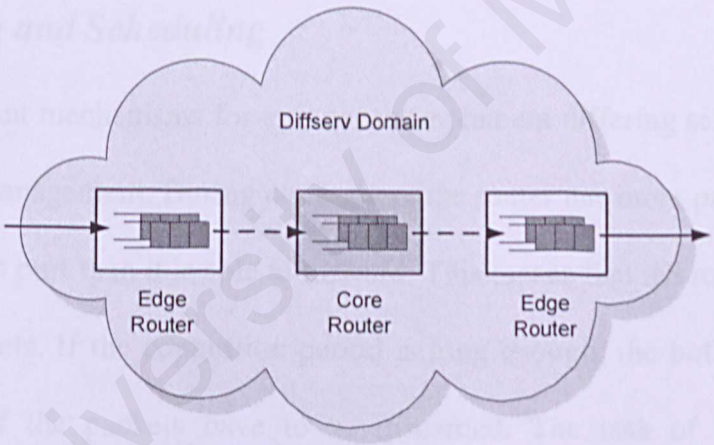


Figure 3.8: Packet forwarding path inside a DS domain.

Figure 3.8 illustrates the process that packets go through inside a router. Packets arrive from an incoming interface are processed and eventually forwarded through an outgoing interface. In this architecture, policing is implemented in the incoming interface while shaping is implemented in the outgoing interface. Once packets arrive at the kernel, they

go through a classifier that categorizes them according to the value of the DS field in the IP header. For example, if the packet is an EF packet, it is passed to the policing module. Otherwise, it is immediately given to the forwarding module. The specification of the EF behavior dictates that EF packets should be given priority at the outgoing interface over best-effort (BE) packets. Moreover, the amount of EF traffic must be shaped according to the existing SLA at the domain boundaries. The difference between edge routers and core routers is that for edge routers policing and shaping parameters are configured by the domain's Bandwidth Broker, while for core routers these parameters are statically configured.

3.5 *Queuing and Scheduling*

The most important mechanisms for a router to implement differing service are queuing and scheduling management. During congestion, the router has more packets destined to a particular output port than it is able to forward. This means that the router has to buffer some of the packets. If the congestion period is long enough, the buffers may become full and some of the packets have to be discarded. The task of the queuing and scheduling is to determine which packet will be transmitted to the next output link and decide which packet/packets will be dropped when the buffer overflows.

The routers implement scheduling in a FIFO basis. This means the packets are always transmitted in the same order they are received. Queue management is very simple. When a packet arrives, it is placed into the tail of the queue based on the PHB values if

there is enough space in the buffer. Otherwise, the packet is dropped. The algorithm is called "drop-tail".

Figure 3.9 illustrates a simplified model of the output buffer of a DiffServ capable router. Each output has a number of logical queues, and the router maps (classifies) each incoming packet into one of the queues based on the PHB value. The queues are then served according to a particular scheduling algorithm. The main idea is that some of the queues get better service than others, thus packets with a PHB corresponding to a high-priority queue usually experience less delay than the other packets.

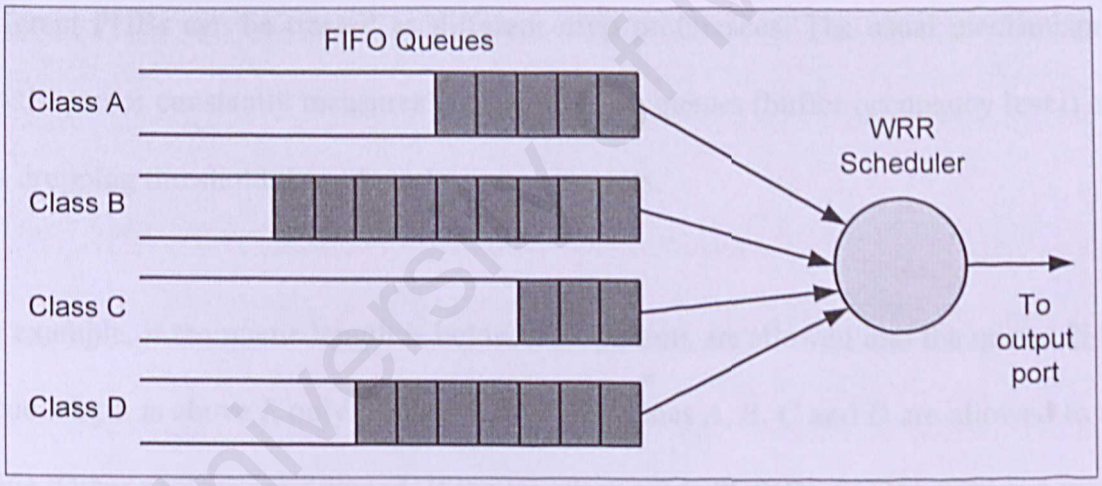


Figure 3.9: Packet scheduler with four logical queues.

Weighted Round Robin (WRR) is used as the scheduling algorithms. WRR is a method used to guarantee a certain amount of bandwidth for each queue. WRR serves each queue in a round-robin fashion, and for each turn, a number of bits corresponding to the queue's weight are "pulled out" from the queue. The queues with higher weights are served more frequently by the scheduler. Thus the link capacity is divided according to

the weights as in WFQ. In a worst-case situation, a packet arrives to a queue just after the queues turn. In that case, the maximum queuing delay will be the sum of the weights of all other queues. It is simple to implement (one loop in software). This may become a deciding factor, if the link speeds increase faster than the pure processing power does.

When the buffer space in a router runs out, some packets necessarily have to be dropped. It does not matter, whether the router is input or output buffered, or whether the ports have shared or individual buffer spaces.

In DiffServ the idea is that the dropping decisions take the PHB values into account. Different PHBs can be treated as different drop preferences. The usual mechanism is that the router constantly measures the length of its queues (buffer occupancy level) and sets dropping thresholds based on the measurements.

For example, if the queue length is below l, all packets are allowed into the queue. If the queue length is above l, only packets with PHB values A, B, C and D are allowed to the queue. Other packets are dropped. If the length exceeds $2l$, PHB A is the only one which can get in. If the router offers several logical queues on the interface (as explained in the previous section), the queue length means the total length of all queues on that interface. This means that delay and drop preferences are independent from each other (orthogonal): high delay priority PHB can have either low or high drop preference.

CHAPTER 4 SYSTEM ANALYSIS

System analysis is carried out to determine what the best for the system to be developed is. To ensure the system is built in the most efficient way, a set of steps is followed to help to understand the system and its specifications before develop the system.

4.1 Simulation Approach

In the context of a network simulator, there are two approaches to modeling. There two approaches are analytical modeling and discrete event modeling.

4.1.1 Analytical Modeling

Analytical modeling is a powerful tool that can offer accurate performance analysis at a fraction of the cost of a benchmark. Analytical models are mathematical representations of a particular computer system. Queuing theory is used to define the relationships between various resources and their queues. These algorithms are populated (parameterized) using measurements taken from a running system. Once the model is built, parameters can be changed to represent possible changes to the running system. The model can accurately project the impact of these changes. The main disadvantage of analytical models is over simplistic view of the network and their inability to simulate the dynamic nature of a computer network (Lim, 2001).

4.1.2 Discrete Event Modeling

A discrete event system is a process characterized by sequence of events. In particular, a change in a system state of a process is precipitated by the occurrence of an action or

event, not merely by the passage of time. This approach is more accurate but it requires more modeling time in developing the system (Lim, 2001). Besides that, it need more time in processing the real world objects.

4.1.3 Simulation Approach Choice

For simulation approach, discrete event modeling will be choosing as it replicates the real world objects. Moreover, the analytical modeling does not able to simulate the dynamic nature of a computer network.

4.2 Programming Approach

There are several widely used programming approaches to develop a network simulator. These programming approaches include procedural approach, structured approach and object oriented approach which will be discussed in this section.

4.2.1 Procedural Programming Approach

In procedural approach, the program codes are placed into blocks that are referred as procedures or functions. A function or procedure is a relatively simple program that is called by other programs and returns a value to the program that called it. With the use of procedural approach, the task was broken down into separate blocks, in which separate blocks would perform separate tasks. Computer languages like Pascal, C and FORTRAN are examples of procedural programming languages.

4.2.2 Structured Programming Approach

Structured programming approach adopted the idea of divide and conquers. A computer program can be thought of as a set of tasks. Any task that is too complex to be described simply would be broken down into a set of smaller component tasks, until the tasks were sufficiently small and self-contained enough that they were easily understood.

Structured programming is a disciplined approach to writing programs that are clearer than unstructured programs, easier to test, debug and modify. However, by the late 1980s, some of the deficiencies of structured programming had become all too clear.

4.2.3 Object Oriented Programming Approach

The programming challenge was seen as how to write the logic, not how to define the data. Objects are essentially reusable software components that model items n the real world. Software developers are discovering that using a modular, object-oriented design and implementation approach can make software developments group much more productive than is possible with previous popular programming techniques such as structured programming. Object oriented programs are easier to understand.

Object Oriented Programming (OPP) approach groups everything as object. It gives more natural and intuitive way to view the programming process by modeling real world objects, their attributes and behaviors. OPP models communications between objects via messages.

One of the importance features of OPP is data encapsulation. OPP performs encapsulation data and method into packages called objects. This could hide unimportant implementation details from other objects, which provides modularity as the source code for an object can be written and maintained independently of the source code for those objects.

Another important feature of OPP is the concept of inheritance where newly created classes of objects inherit the characteristic of the existing classes, yet contain unique characteristics of their own.

Besides that, OPP's polymorphism enable programmer to write programs in a general fashion to handle a wide variety of existing yet-to-be-specified related classes. It makes developers easy to add new capability to a system. There are a few benefits of OPP which are listed in Table 4.1.

Table 4.1: Benefits of using OOP approach.

Benefits	Description
Extensibility	New features can be added to the system where changes on new objects can be done by modification of existing objects.
Maintainability	Maintenance and modification of objects can be done individually
Reusability	Objects that are used in a system can also be used in another newly built system with little or no changes.
Simplicity	It is simple and less complex using the OOP approach while building programs, which attempts to model the objects interactions of the real world. Any changes are easy to modify with no much affect within the entire system.
Modularity	Objects within the program are individual separate entities, the internal workings of which are isolated and de-coupled from other objects in the system. This solves the problem of coupling in procedural programming approach.

4.2.4 Programming Approach Choice

After making the analysis on several approaches, the object oriented programming approach will be used for this project due to its benefits mentioned before.

4.3 Programming Language

It is very important to use an appropriate type of programming language in building any application programs and simulators. Thus, it is a need to consider the advantages and disadvantages of several programming languages here.

4.3.1 C++

C++ evolved from C, which evolved from two precious programming languages, BCPL and B. BCPL, was developed in 1967 by Martin Richards as a language for writing operating system software and compilers.

C++ provides a collection of predefined classes along with the possibility of user defined classes. The classes of C++ are data types, which can be instantiated any number of times. Such an instantiation in C++ are merely object or data declarations. Classes can name one or more parent classes, providing inheritance and multiple inheritances, respectively. Classes inherit the data members and member functions of the parent class that are specified to be inherited.

Dynamic binding in C++ is provided by virtual class functions. A pointer to an object of class A can also point to an overloaded virtual function, the function of the current type is chosen dynamically. Both function and classes can be template which means that they can be parameterized.

One of the factors make C++ became a popular language is the availability of good and inexpensive compilers. Another factor in favor of the popularity of C++ is that it is almost completely downward compatible with C and in most implementations it is possible to link C++ code with C code. Besides that, programming is now intensely interested in object oriented programming approach.

However, C++ is a large and complex language. It latterly suffer drawbacks similar some other complex language. It inherited most of the insecurities which make it less safe than other languages such as Java. To write large program in C++ was difficult and the results of trying are described as "spaghetti code". Many of the object oriented features of C++ have been introduced to address this problem.

4.3.2 Java

Java was developed by Sun Microsystems. Sun formally announced Java at a major conference in May 1995. Now Java is used to create Web Pages with dynamic and interactive content, to develop large scale enterprise applications, to enhance the functionality of World Wide Web servers, to provide applications for consumer devices and for many other purposes. The important Java features that make it an attractive programming language are listed in Table 4.2.

Table 4.2: Features of Java programming language.

Features	Description
Object Oriented	Even though Java has the look and feel of C++, it is a wholly independent language which has been designed to be object-oriented from the ground up. In object-oriented programming (OOP), data is treated as objects to which methods are applied. Java's basic execution unit is the *class*. Advantages of OOP include: reusability of code, extensibility and dynamic applications.
Familiarity And	Java was developed by taking the best points from other

Simple	programming languages, primarily C and C++. Therefore, it utilizes algorithms and methodologies that are already proven. Error prone tasks such as pointers and memory management have either been eliminated or are handled by the Java environment automatically rather than by the programmer. Since Java is primarily a derivative of C++ which most programmers are conversant with, it implies that Java has a familiar feel rendering it easy to use.
Secure	The Java language has built-in capabilities to ensure that violations of security do not occur. Because Java does not use pointers to directly reference memory locations, as is prevalent in C and C++, Java has a great deal of control over the code that exists within the Java environment. Sun Microsystems will soon be adding another dimension to the security of Java. They are currently working on a public-key encryption system to allow Java applications to be stored and transmitted over the Internet in a secure encrypted form.
Interpreted	When Java code is compiled, the compiler outputs the Java Byte code which is an executable for the Java Virtual Machine. The Java Virtual Machine does not exist physically but is the specification for a hypothetical processor that can run Java code. The byte code is then run through a Java interpreter on any given platform that has the interpreter ported to it. The interpreter converts the code to the target hardware and executes it.
Robust	The Java objects can contain no references to data external to themselves or other known objects. This ensures that an instruction cannot contain the address of data storage in another application or in the operating system itself, either of

	which would cause the program and perhaps the operating system itself to terminate or "crash". The Java virtual machine makes a number of checks on each object to ensure integrity.
Distributed	Commonly used Internet protocols such as HTTP and FTP as well as calls for network access are built into Java. Internet programmers can call on the functions through the supplied libraries and be able to access files on the Internet as easily as writing to a local file system.
Architecturally Neutral	The Java compiler compiles source code to a stage which is intermediate between source and native machine code. This intermediate stage is known as the byte code, which is neutral. The byte code conforms to the specification of a hypothetical machine called the Java Virtual Machine and can be efficiently converted into native code for a particular processor.
Portable and platform-independent	By porting an interpreter for the Java Virtual Machine to any computer hardware/operating system, one is assured that all code compiled for it will run on that system. This forms the basis for Java's portability. Another feature which Java employs in order to guarantee portability is by creating a single standard for data sizes irrespective of processor or operating system platforms.
Multithreading	Multithreading is the ability of an application to execute more than one task (thread) at the same time. Java is able to use the idle time to perform the necessary garbage cleanup and general system maintenance that renders traditional interpreters slow in executing applications.
Dynamic	During the execution of a program, Java can dynamically load classes that it requires either from the local hard drive, from

	another computer on the local area network or from a computer somewhere on the Internet.

4.3.3 Programming Language Choice

The programming language used to develop this simulator is Java. The main reason to choose Java for the development of simulator is that it support multithreading. This feature enables the simulator to perform a few tasks at the same time. It does not contain any additional features added like Visual J++ that only can be supported by Microsoft software.

4.4 Software Selection

Using the right software tools will help in developing a perfect system. The following sections consider the features of using JBuilder and JCreator in order to choose a right tool.

4.4.1 JBuilder

JBuilder is a group of highly productive tools for creating high performance and platform independent application for Java. It is designed for all levels of development of project, ranging from applets and applications that require networked database connectivity to client/server and enterprise wide, distributed, multi-tier computing solution.

The JBuilder IDE supports a variety of technologies including 100% Pure Java, JavaBeans, Java2, Java SDK 1.2.2 and JFC/Swing. The additional technologies

supported by JBuilder Professional edition are Servlets, Remote Method Invocation (RMI), Java Database Connectivity (JDBC), Open Database Connectivity (ODBC) and all major corporate database servers. The additional technologies supported by JBuilder Enterprise are Enterprise JavaBeans, JavaServer Pages (JSP) and Common Object Request Broker Architecture (CORBA).

JBuilder also provides developers with a flexible, open architecture that makes it easy to incorporate new SDKs, third party tools, add-ins, and JavaBeans components.

4.4.2 JCreator

JCreator is a powerful IDE for Java. JCreator provides the user with a wide range of functionality such as project management, project templates, code-completion, debugger interface, editor with syntax highlighting, wizards and a fully customizable user interface. The features of JCreator are listed at the following:

- Manage projects with ease in the interface that is much like Microsoft® Visual Studio®.
- Define your own color schemes for unlimited ways to organize your code.
- Unlike most IDEs, JCreator wraps around your existing projects and allows you to use different JDK profiles.
- Get down to writing code quickly with our project templates.
- Our class browser makes viewing your project a breeze.
- Debug with an easy, intuitive interface. No need for silly DOS prompts!

- Walk through our wizards and cut to the chase of writing your project, quickly and easily.
- You don't have to spend valuable time on Class path configuration—JCreator does it all for you.
- Customize our user interface the way that you like it.
- Set up your own run-time environments to run your application as an applet, in a JUnit environment or in a DOS window.
- JCreator has lower system requirements, yet faster speed, than all those other IDEs.

Unlike most IDEs, JCreator has two types of tools that can be configured. The first type is the Java Development Kit (JDK) tools. JDK tools can be used to compile, debug, and run the project. Users can attach these tools to their project using the Project Properties dialog box. If no project is available, JCreator runs the default projects. Users can easily create their own tools for calling the JDK applications, such as the following:

- Compiler
- Interpreter
- Applet viewer
- JDK Help files

The second type of tool is more general and allows users to extend the capabilities of JCreator to fit their needs—by allowing users to call external functions and utilities. Users can assign these general tools to the Wrench buttons located on the Tools toolbar

in the workspace. These buttons display tool tips, such as User Tool 1, User Tool 2, and so forth. These tools can have many uses, such as the following:

- XML validator
- RMI compiler
- JAVA indent formatters
- Batch files

JCreator is written entirely in C++, which makes it fast and efficient compared to the Java based editors/IDEs. Professionally designed to meet windows interface guidelines, JCreator users can expect a familiar and intuitive user interface.

4.4.3 Software Choice

JCreator will be chosen as the software tool for developing this project. It is because it provides a wide range of functionalities. Moreover, the features compared to JBuilder are also taking into consideration.

4.5 Hardware Consideration

As the networks technology grow the aspects of openness and transparency of the software and hardware layer has reached a point that generally agreed upon. Thus, even though the system evolves both hardware and software, the architecture of the hardware is not that critical. Table 4.3 lists the minimum requirements of hardware.

Table 4.3: Hardware requirements.

Main Machine/PC	Requirement
Hard Disk Space	500 MB hard disk space recommended minimum (includes space required during installation)
Memory	256 MB RAM recommended minimum
Processor	Intel Pentium II/233 MHz or higher (or compatible)
Operating System	Microsoft Windows 2000 (SP2), XP, or NT 4.0 (SP6a)

Note: hardware above must be well installed and working properly in order to run the system.

4.6 *Architecture of UMJaNetSim*

UMJaNetSim (Lim, 2001) is a flexible test bed for studying and evaluating the performance of DiffServ without the expenses of building a real network. It uses Java (OOP) programming approach and is a discrete event model simulator. It consists of a central simulation engine with a centralized event manager. The simulation scenario consists of a finite number of interconnected components (simulation objects), each with a set of parameters (component properties). Simulation execution involves components sending messages among each other. A message is sent by scheduling an event (to happen some time later) for the target component. With these basic features, the simulator can simulate virtually "anything" that can be modeled by a network of components that send messages to one another. These concepts are adopted from the NIST ATM/HFC Network Simulator.

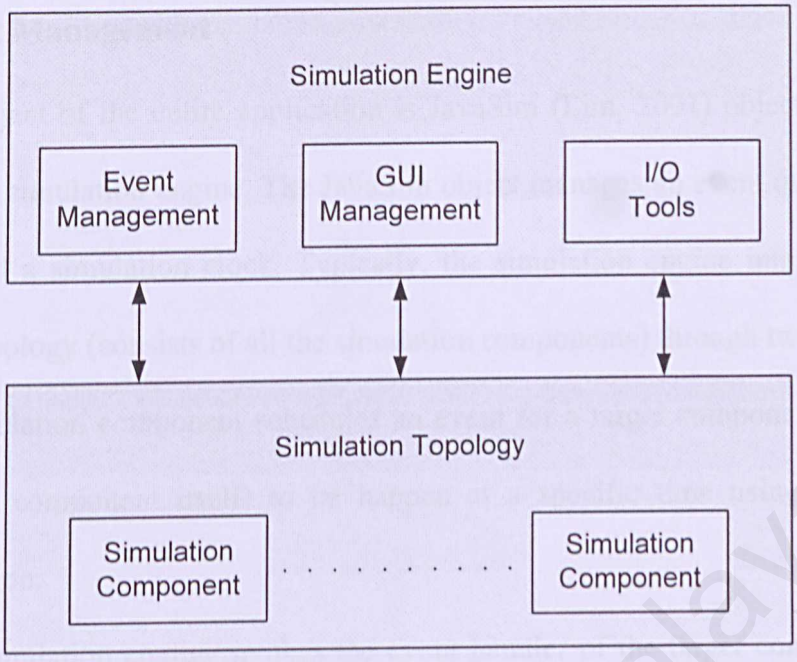


Figure 4.1: Overall architecture of UMJaNetSim.

As showed in Figure 4.1, simulation engine and simulation topology are the mainly 2 part of UMJaNetSim architecture. The simulation engine is the main controller of the entire simulation that will handle event management task and GUI management task. Besides that, it also handles the input/output process and provides many tools that help the simulation process. The simulation topology consists of all the simulation objects which are also referred to as simulation components. These simulation components are the main subject of a simulation scenario and these simulation components typically consists of a group of interconnected network components such as router, switch, physical link and different types of source applications.

4.6.1 Event Management

The major object of the entire application is JavaSim (Lim, 2001) object, which itself represents the simulation engine. The JavaSim object manages an event queue, an event scheduler, and a simulation clock. Typically, the simulation engine interacts with the simulation topology (consists of all the simulation components) through two operations:

- A simulation component schedules an event for a target component (can be the source component itself) to be happen at a specific time using the enqueue operation.
- The simulation engine invokes the event handler of the target component when that specific time is reached. The target component will react to the event according to its behavior.

Figure 4.2 shows the event management architecture of UMJaNetSim. The event queue is actual a java.util.List object consists of all the scheduled events in the form of SimEvent objects. The events are sorted by the event-firing time. The event scheduler always fetches and removes the first event in the event queue, and fires the event by invoking the event handler of the target component.

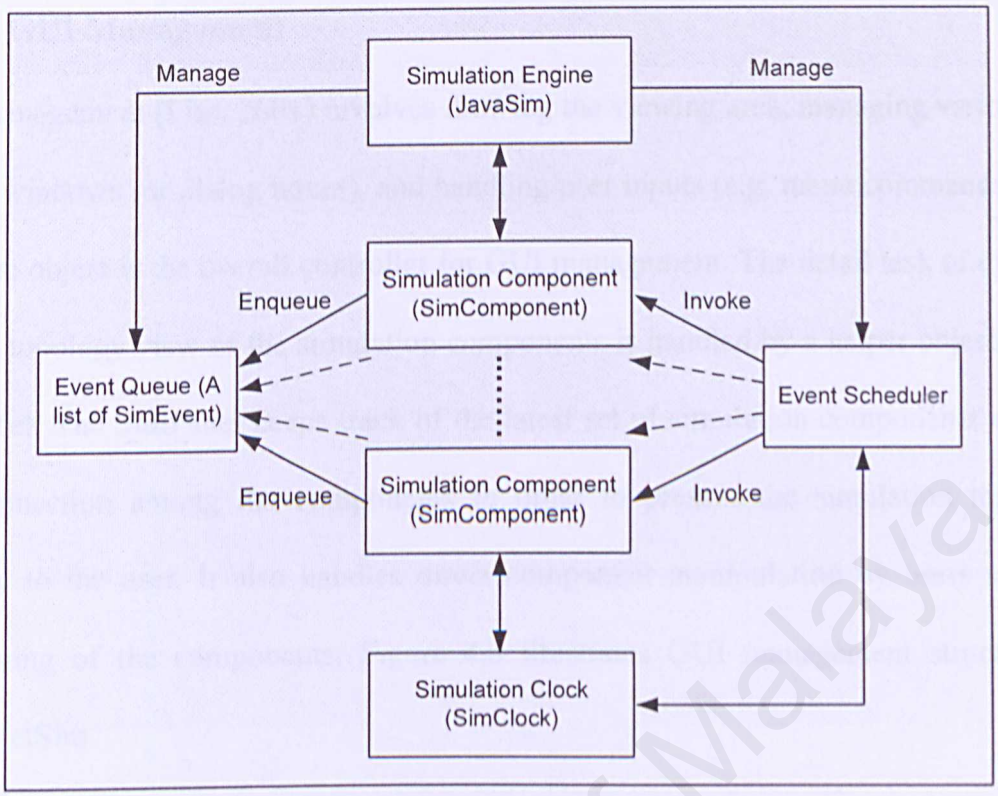


Figure 4.2: Event management architecture.

In a discrete event type of simulation, the simulation time is an important concept. The UMJaNetSim uses an asynchronous approach of the discrete event model, where any event can happen at any time, up to the precision allowed by the granularity of the simulation clock. The simulation time in the UMJaNetSim is based on "ticks". The duration of a tick is configurable in the simulator. By default, a tick is equivalent to 10 nanoseconds. The SimClock object is the global time reference used by every component in the simulation and managed by the simulation engine. The SimClock object also provides helper methods for the conversion between real time and the tick.

4.6.2 GUI Management

GUI management (Lim, 2001) involves drawing the viewing area, managing various on-screen windows (or dialog boxes), and handling user inputs (e.g. menu commands). The JavaSim object is the overall controller for GUI management. The detail task of drawing out the topology view of the simulation components is handled by a helper object called SimPanel. The SimPanel keeps track of the latest set of simulation components and the interconnection among the components in order to present the simulation topology visually to the user. It also handles direct component manipulation by users such as positioning of the components. Figure 4.3 illustrates GUI management structure of UMJaNetSim.

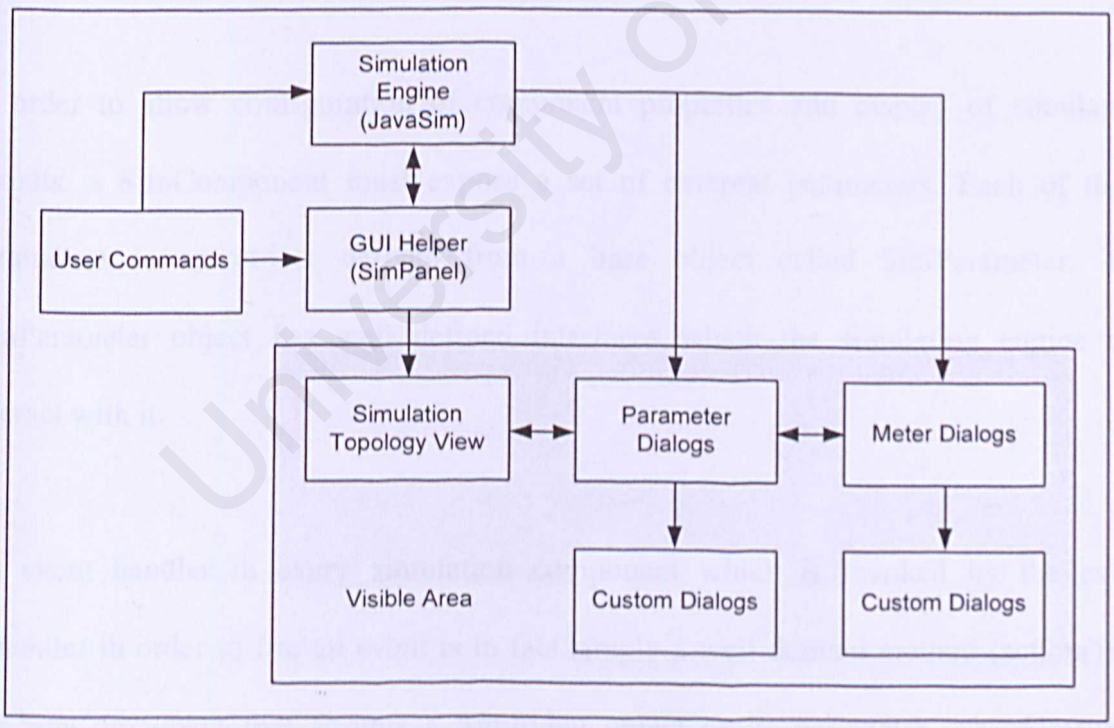


Figure 4.3: GUI management structure.

4.6.3 Simulation Components

The primary simulation objects in the UMJaNetSim are called simulation components (Lim, 2001), where each is represented by the object SimComponent. The SimComponent is a well defined base object with all the necessary interfaces that enable the interaction between the simulation engine and the component. Actual simulation components all inherent the properties and methods of this base object. The default interaction with the simulation engine (e.g. the component graphical image) can be easily modified by overriding the proper methods in the SimComponent object. With this, the component designer needs not concerned with the issues of "talking" to the simulation engine, instead, the focus is on the design of the proper behaviors of the components to archive the simulation objectives (Lim, 2001).

In order to allow configuration of component properties and display of simulation outputs, a SimComponent must expose a set of external parameters. Each of these parameters is an object derived from a base object called SimParameter. The SimParameter object has well defined interfaces which the simulation engine can interact with it.

An event handler in every simulation component which is invoked by the event scheduler in order to fire an event is in fact simply a well defined method (action()) in the SimComponent that accepts a SimEvent object as its parameter. The SimEvent object has complete description of an event including the event ID, the source component, and the optional parameters that come with the event. All components

should override the action() method in order to react to events. All interactions between simulation components are achieved through the sending of messages in the form of a SimEvent.

4.6.4 UMJaNetSim API

The JavaSim object is the main object of the simulator. It keeps a list of all the network components, which are the descendents of SimComponent, and a list (a queue) of all events that in the form of SimEvent. Every component contains a set of parameters, which inherit SimParameter. All other classes are mostly helpers that provide certain services such as time service, logging and meter display.

4.6.4.1 JavaSim

JavaSim object is the main object of the simulator. It keeps a list of all the network components and a list of all events. Each component contains a set of parameters.

4.6.4.2 SimClock

Components send each other events in order to communicate and send cells through the network. The software contains an event manager, which provides a general facility to schedule and send, or fire an event. An event queue is maintained in which events are kept sorted by time. To fire an event, the first event in the queue is removed, the global time is set to the time of that event and action scheduled to take place is undertaken.

Events can be scheduled at the current time or at any time in the future. Scheduling events for the past is considered illogical. Events scheduled at the same time are not guaranteed to fire in any particular order. Simulator time is maintained by the event manager in units of ticks. The time is maintained as an unsigned 32 bits value. The simulator time represented by one tick can be changed by software modification, but nor by the simulator user. It provides a set of time translation functions for normal translation between tick and actual time.

4.6.4.3 SimEvent

Each SimComponent communicates with each other by enqueuing SimEvent for the target component. For example, when component A wants tp send a packet to component B, component A creates a SimEvent that specifies B as its destination, and enqueue the event. The SimEvent object also contains a time so that this event is fired at exactly the specified time. Component B will then be able to react to the event accordingly.

4.6.4.4 SimComponent

This is a very important class to understand in order to develop new components in the simulator. Each network component in the simulator must inherit SimComponent. The SimComponent class itself should not be instantiated because it only provides the skeleton for an actual component. A new component should extends SimComponent and override its various methods in order to provide meaningful operations for the component.

4.6.4.5 SimParameter

Every SimComponent can have internal parameters or external parameters. All external parameters must inherit SimParameter. By extending SumParameter, one obtains parameter logging and meter display features automatically. Obviously, SimParamInt, SimParamDouble, SimParamBool, and SimParamString objects provide support for integer, double, boolean and string parameters. Other types of parameters can be created by extending SimParameter accordingly.

4.7 Requirement Analysis

Requirement analysis is an important method to enable the system engineer to specify software elements and establishes design constraints that software must meet. Requirement analysis can be divided into functional requirements and non-functional requirements. The following will discuss in detail about the functional and non-functional requirement.

4.7.1 Functional Requirements

Functional Requirements describe functions and features that the system should provide for the users. The system is considered incomplete if any of the necessary functions are not included. The functional requirements for this project are listed below.

- Graphical User Interface (GUI)

  It is a device for user, operating system and network simulator to communicate to each other. To make this network simulator more attractive, the GUI must be more users friendly.

- Input System

  Before the simulation runs, the system must be able to let user enter or select values for certain fields. This is important in order to make sure the simulation run under the desired configuration.

- DiffServ Router

  The router must be able to classify packets to in different PHBs which are being implemented. After classifying, the packet should be scheduled according to the WRR mechanism. The WRR should forward packets according to the weight of each queue and the drop precedence of PHB.

- Output System

  The system will generate the output traffics by using the scheduling mechanism. A GUI will be used to show how the simulation process is going on and the performance of the particular network.

4.7.2 Non-functional Requirements

In order to ensure the quality of the system produced, certain quality factors must be conformed. Non-functional requirements are those constraints on the service or functions offered by the system. The following non-functional requirements have been considered for this project.

- Reliability

  Systems will not produce any dangerous when it is used in a reasonable manner, which mean in a manner that a typical user expects is normal. In other words, reliability is referred to the expectation of a system to perform its intended function accurately. Whenever a button is clicked, the system should be able to execute that particular function or generate some messages to inform the user about what is happening.

- User friendly

  This network simulator is designed based on the concept of user friendly which means that user has the ability to use the program at the lowest possible of getting confuse with the interface of this network simulator. The interface should be design to suit need and not to the developer point of view.

- Efficiency

  Efficiency is understood as the ability of a process procedure to be called or accessed unlimitedly to produce similar performance outcomes at an acceptable

or credible speed (Sommerwille, 2001). Even so, the efficiencies are referred to the consumption of the local and remote machine memory and the bandwidth of network usage during the run time. The lesser it uses, the higher effiencicy.

- Maintainability

  System maintenance would require more effort if the system is not designed according to good programming practices. Maintainability is the ease with which a program can be corrected if an error is encountered, adapted if its environment changes, or enhanced if the customer desires a change in requirements. As the to-be-developed network simulator will be built by using object oriented concept, it is strongly believed that bugs or system faults can be detected and fixed in the shortest time. This is because object oriented design makes sure that each class or object will only strictly handle one particular task or functionality.

CHAPTER 5 SYSTEM DESIGN

The information collected during research phase and analysis phase is used to design the system of network simulator. The simulation of the system is designed to allow various network topologies being simulated including DiffServ network. The user interface is designed to allow user to create the network topology while the output is designed to allow the integration of DiffServ into the existing network simulator.

5.1 Router Architecture Design

The queue architecture designated the queuing model used in this simulation and WRR algorithm used for scheduling cell at output port.

5.1.1 Queuing Model

Router is the component that routes cells over several virtual channel links. A local routing table is provided for each router. This table contains a route number (that is read from incoming cell structure and equivalent to the cell's virtual channel identifier), a next link entry, a next BTE entry, and so on.

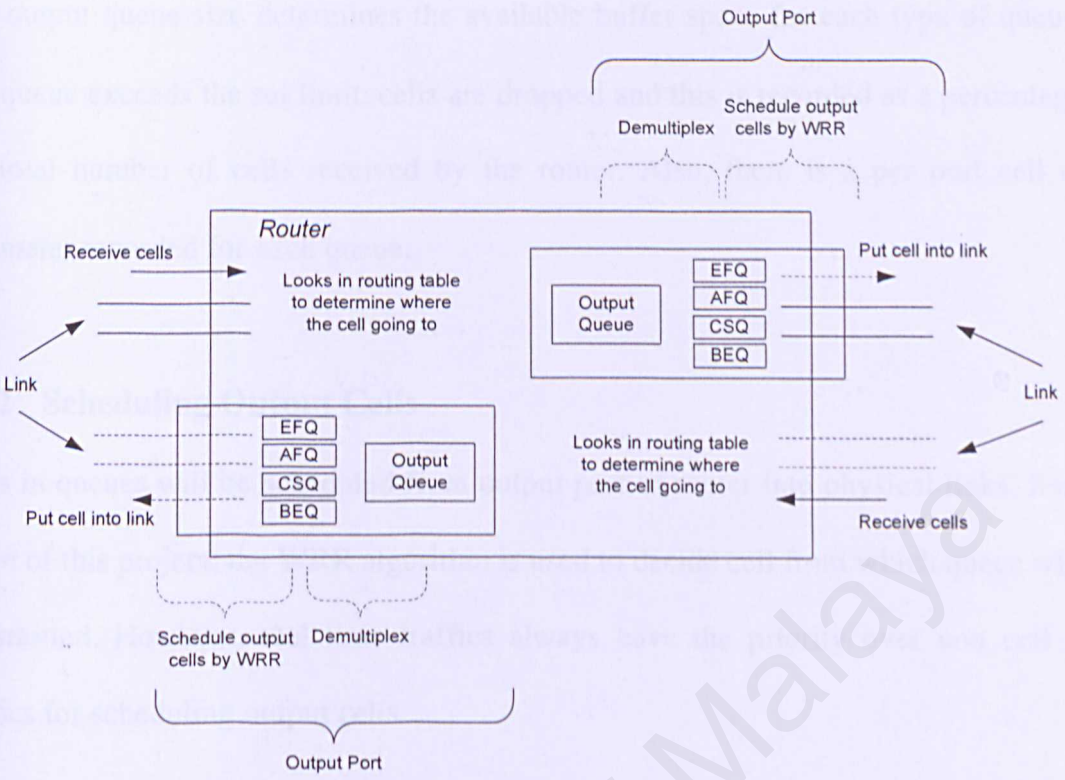


Figure 5.1: Output port queuing model.

From Figure 5.1, the output port is one of the components of the router. Since router is model as a thread, the processes of an output ports are controlled under a router. At first, a cell arrives at the router from a physical link. After that, the router will look in its local routing table to determine which outgoing link it should redirect the cell to. If the link has an empty slot available, the router puts the cell on the link. Otherwise, the cell will be demultiplex (for awaits transmission) into one of the different forwarding priority queues such as EF queue or AF queue, depending on the type of service configured by users. Cells in EF queue have priority over AF queue. That means it is only when the EF queue is empty, then the cells in AF queue are sent to the link at schedule output port.

The output queue size determines the available buffer space for each type of queue. If any queue exceeds the set limit, cells are dropped and this is recorded as a percentage of the total number of cells received by the router. Also, there is a per port cell drop parameter recorded for each queue.

5.1.2 Scheduling Output Cells

Cells in queues will be scheduled from output port of router into physical links. For the scope of this project, the WRR algorithm is used to decide cell from which queue will be transmitted. However, real time traffics always have the priority over non real time traffics for scheduling output cells.

For the implementation of DiffServ, the cells are classified into different PHBs which in turn are demultiplex into different queues. In order to guarantee the services, each queue is assigned different weight according to their priority of transmission. Scheduling is worked in a round robin fashion whilst the cells with higher weight are transmitted first.

5.2 System Functionality Design

The system is designed to meet a set of functions to simulate the real network. These functionalities are described in the following section.

5.2.1 Design of Demultiplex

Demultiplex is designed to accept a stream of cells destined to specific ports. It first process the cells by identifies their cell types, then it forwards them to the respective

signal queue according to their cell type, and lastly follow from this point of processing, the Tail Drop Buffer Management function is executed.

5.2.2 Design of Queue

All of the queues in the DiffServ router will be simple FIFO discipline.

5.2.3 Design of Tail Drop Buffer Management

The Tail Drop function is implanted with the demultiplex code. This scheme is designed so that every queue at every port will be serviced by its own buffer management. Its process begins as soon as demultiplex has identified the cell type. After that, the buffer management scheme will take over. It determines whether this cell will be either dropped or enqueued by comparing the current queue length with the queue capacity. If the comparison is lesser, the cell is inserted to the location of queue at the pointer and the queue pointer is incremented. Statistic of router such as the time of the cell's entry, and the total numbers of cells enqueued is recorded. However, if the comparison is to be equal, the cell will be dropped and the statistic of router will be updated.

5.2.4 Design of Scheduler

The content if every queue is retrieved by a scheduler which is work based on WRR algorithm. The scheduler will extract cells from the output queue to the available link.

5.2.5 Design of Control Function

A control bar on the UMJaNetSim simulator consists of several buttons that is useful to the processing of simulator. The buttons includes Start, Pause, Resume and Reset. The digital Global Clock is also included in the control bar area. The function of each control element is described in Table 5.1.

Table 5.1: Function of each control element in the control bar.

Button	Function
Start	Start the new simulation on the topology.
Pause	Halt the simulation.
Resume	Resume the simulation after the simulation process is halted.

5.2.6 Design of Log File

The log file is to be developed to record the values of a parameter while the simulation is running. All statements outputting variable data must be preceded with appropriate literals. When a parameter is to be logged during the simulation, every new value of the parameter with a corresponding time stamp will be saved in a log file.

5.3 Process Design

In order to run the simulator successfully, appropriate design of process flow is needed. Therefore, relevant structure chart and flow chart will be presented in the following section.

5.3.1 Component Creation

To create a network topology for simulation, the simulator must design to have a set of network components. Figure 5.2 shows the designed components of UMJaNetSim.

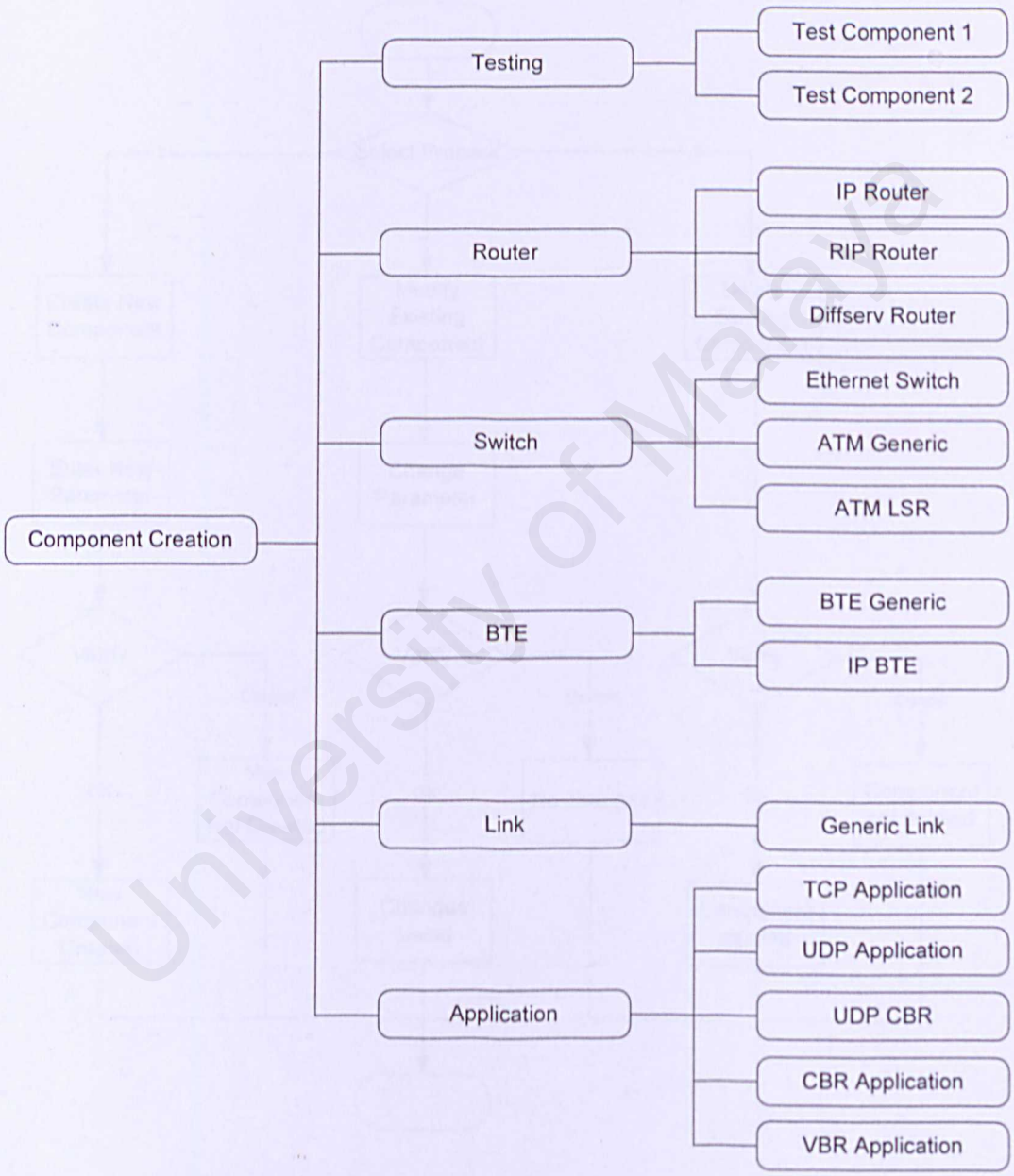


Figure 5.2: A set of network components in simulator.

After getting the overview of network components, the process associated to those components such as create new component, modify existing component and delete existing component will be described using a flow chart as in Figure 5.3.

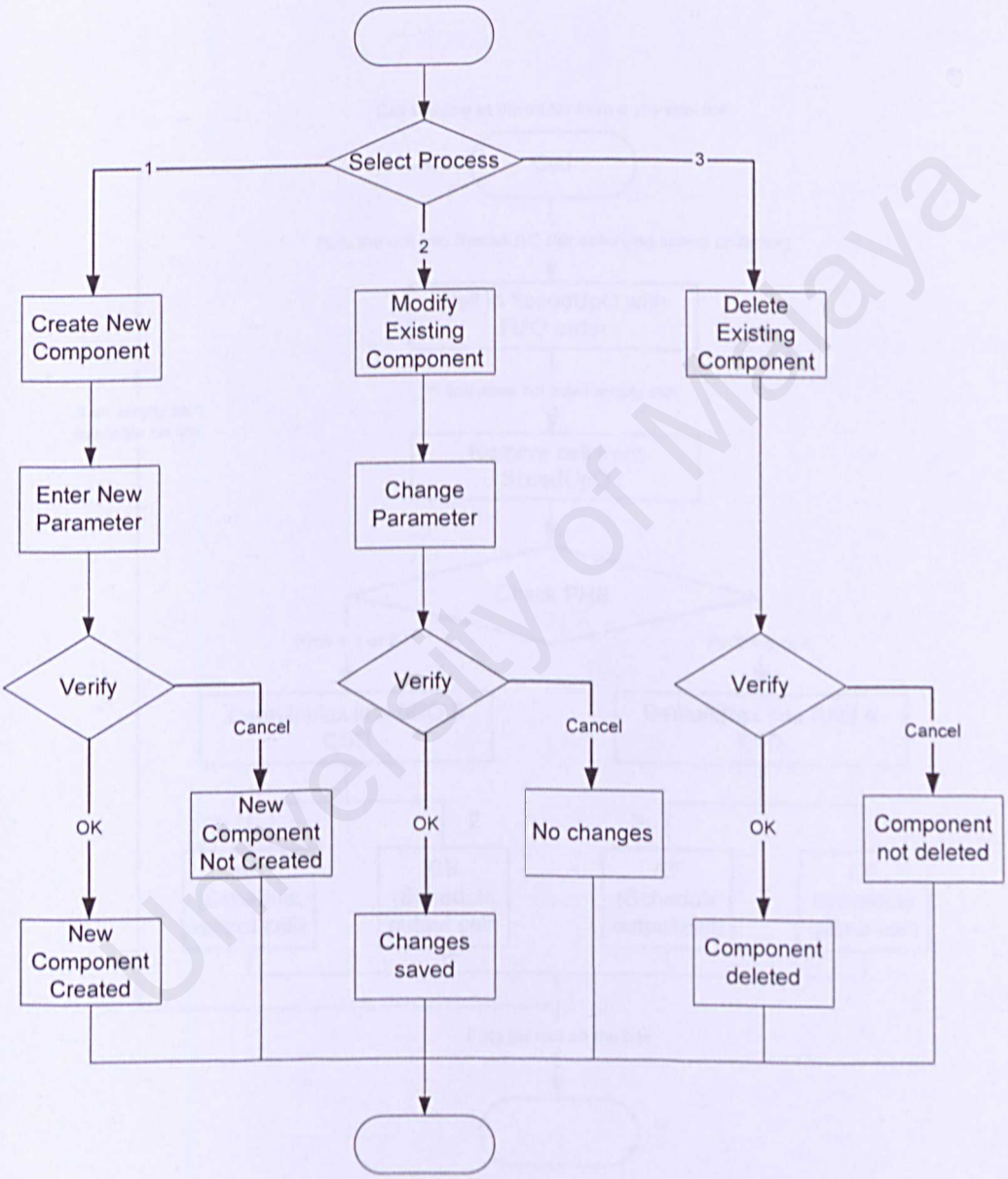


Figure 5.3: Flow chart for network component.

5.3.2 Flow of Cells

The flow of cells in router from the beginning (arrive at the router from a physical link) until the end (cells are put on the link) is described using a flow chart as showed in Figure 5.4.

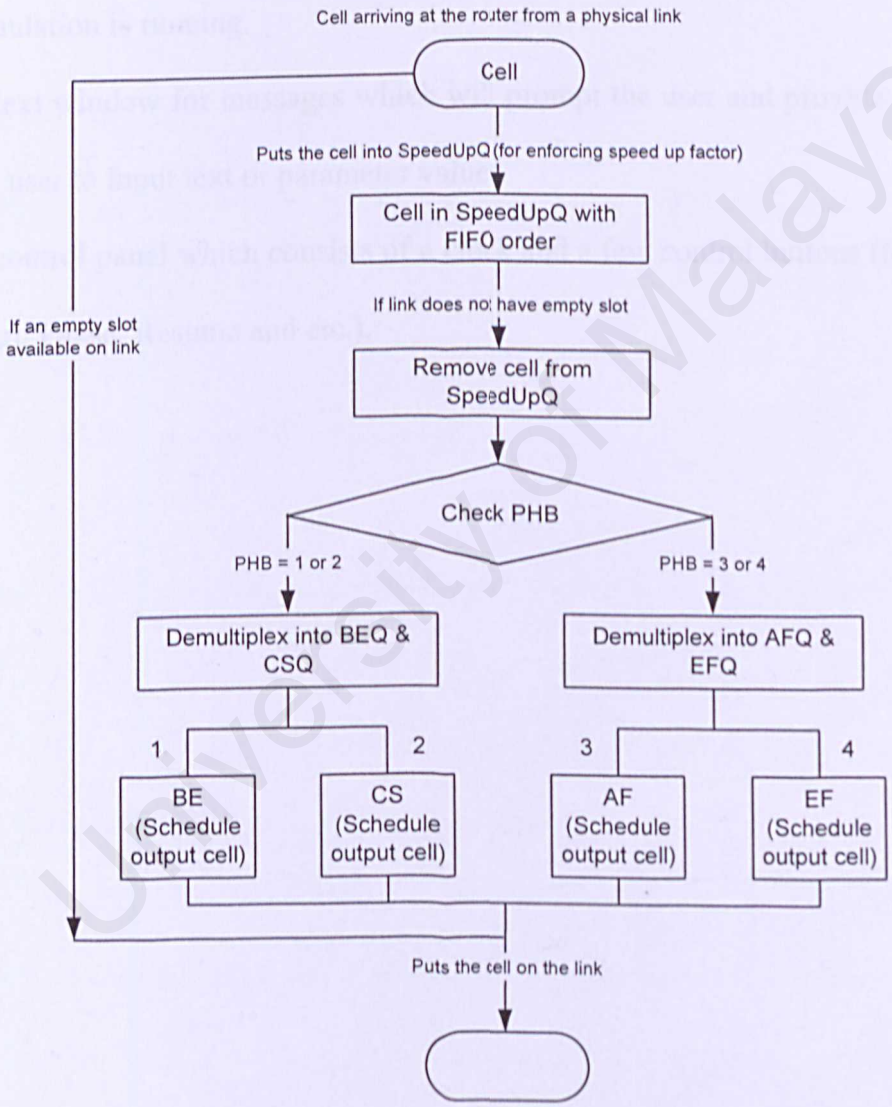


Figure 5.4: Flow chart for cells in router.

5.4 *Interface Design*

The user interfaces are designed based on the concept of easy for user to understand and used to create the network topology. The design of GUI is divided into 3 major parts:

i.) A network window used to display network topology. This window is used to create the components, set parameters as well as show network activity while the simulation is running.

ii.) A text window for messages which will prompt the user and provide a place for the user to input text or parameter values.

iii.) A control panel which consists of a clock and a few control buttons (for example Start, Pause, Resume and etc.).

CHAPTER 6 IMPLEMENTATION

This chapter will cover the implementation aspect that need to be done for the simulator. It will provide a look into how the component is designed and implemented. During the implementation phase, all the classes with important attributes will be shown together with the explanation of these attributes as well as methods contained within the classes.

6.1 System Implementation

The routing process is start when a packet arriving at the router from a physical link. At the next processing slot time, after some delay, the router looks in its routing table to determine which outgoing link it should redirect the packet to and add the packet into spq (a queue or buffer for incoming packets).

At the same time, a function will remove each packet from the spq in a FIFO basis, check that particular packet for which PHB the packet belongs to, and add it to the particular PHB's queue. Then, the packet will go through a schedule output process depend on the user selects priority scheduling algorithm.

EF packets have priority over other PHBs; AF packets have priority over CS and BE packets. Thus, before the simulation start, user needs to choose the priority scheduling algorithm and initial suitable threshold values or the simulator will treat it as BE packets (default setting).

6.2 Class Implementation

The implementation of the UMJaNetSim is the phase that transforms the theoretical into the practical. This section will look in turn at the implementation of each of the object classes that make up the DiffServ's four PHBs, as well as some of the other object classes that DiffServ's PHBs makes use of.

6.2.1 IPPacket.java

IPPacket can be considering as one of the fundamental objects in the network simulator. The *TOS constants* are used as DSCP information for each packet. It is very important for routing process as the router needs to get the correct DSCP information for each packet to schedule out it based on the different priorities of different PHBs. The default value of *TOS* is set to Default PHB. This value will be changed if the user classified the packet as a different PHB. The *TOS* coding is as follow:

```
//Default TOS value for packet
public static int TOS=0;

//TOS constants for packet
public static final int TOS_EF  = 0xB8;   //Expedited Forwarding
public static final int TOS_AF1 = 0x28;   //Assured Forwarding
public static final int TOS_AF2 = 0x48;   //Assured Forwarding
public static final int TOS_AF3 = 0x68;   //Assured Forwarding
public static final int TOS_AF4 = 0x88;   //Assured Forwarding
public static final int TOS_CS  = 0x38;   //Class Selector
public static final int TOS_BE  = 0;      //Default (Best-Effort)
```

6.2.2 UDP_CBR.java

UDP_CBR has been chosen to be the application that will implement DiffServ in the network simulator. *UDP_CBR* class generates UDP packets at a constant bit rate for the duration of the simulation. The header of the file is as below:

```
public class UDP_CBR extends SimComponent implements
java.io.Serializable {
  //User initial value
  private SimParamDouble cn_bit_rate;
  private SimParamInt cn_start_time;
  private SimParamInt cn_packet_size;
  private SimParamDouble cn_trans_size;
  private SimParamInt cn_repeat;
  private SimParamInt cn_delay;
  private SimParamBool cn_random_size;
  private SimParamBool cn_random_delay;
  private SimParamBool cn_start_delay;
  private SimParamBool cn_random_target;
  private SimParamBool cn_name_seed;
  private SimParamIP cn_destip;
  private SimParamInt cn_destport;

  //Display purposes
  private SimParamInt cn_thisport = null;
  private SimParamInt cn_conattempt;

  //DiffServ PHB which can be selected by user
  private SimParamIntTag cn_ds_class;

  private int cn_status;
  private long cn_cur_trans_size;
  private int cn_con_done;
  private long cn_num_sent;
  private int cn_this_ip = 0;

  /*Display total packets sent out from this particular UDP_CBR
    application */
  private SimParamLong cn_total_sent;
```

```
private java.util.Random randgen;

//connection status constants
private static final int CON_NULL = 0;
private static final int CON_ACTIVE = 2;

//Declare the protocol used
private static final int MY_PROTOCOL = IPPacket.PRO_UDP;

//private events
private static final int MY_SENDCELL = SimProvider.EV_PRIVATE + 1;
private static final int MY_START = SimProvider.EV_PRIVATE + 2;
```

Some attributes used in the *UDP_CBR* application are user input parameters. These attributes are used to specify the type of traffic that will be generated for the simulation. There are also some attributes used for display purpose. These attributes can not be modified or initial value by users at all.

When *UDP_CBR* is connected, the method performed in this class will setup connections when the connection status is null. Then, the method will assign appropriate TOS value to the packet based on the type of DiffServ PHB selected by user. After each packet has been assigned to correct value, which indicates the PHB assigned by user, the method in this class will send the packet out with its information including the TOS value. To send all the information to the destination, an *Object* must be declared to carry all the information. Then, the appropriate method will enqueue that particular Object when the packet is being sent out. The total packet sends out will be updated each time the packet being send out. The way to assign the TOS value to the particular TOS constants declared in *IPPacket.java* is as follow:

```
switch( cn_ds_class.getValue() ) {        //Get the type of PHB
      //If the packet has been assigned to Best Effort
      case 0: packet.TOS=IPPacket.TOS_BE;
              break;

      //If the packet has been assigned to Class Selector
      case 1: packet.TOS=IPPacket.TOS_CS;
              break;

      //If the packet has been assigned to Assured Forwarding 1
      case 2: packet.TOS=IPPacket.TOS_AF1;
              break;

      //If the packet has been assigned to Assured Forwarding 2
      case 3: packet.TOS=IPPacket.TOS_AF2;
              break;

      //If the packet has been assigned to Assured Forwarding 3
      case 4: packet.TOS=IPPacket.TOS_AF3;
              break;

      //If the packet has been assigned to Assured Forwarding 4
      case 5: packet.TOS=IPPacket.TOS_AF4;
              break;
      //If the packet has been assigned to Expected Forwarding
      case 6: packet.TOS=IPPacket.TOS_EF;
              break;
}
```

The way to send out the packet information is as follow:

```
//Declare a new Object with 2 elements
Object [] paramlist = new Object[2];
//assigned value to the first element
paramlist[0] = packet;
//assigned value to the second element
paramlist[1] = new Integer(packet.TOS);

//sending the packet out with packet identification and TOS value
theSim.enqueue( new SimEvent ( SimProvider.EV_RECEIVE,this,
                               neighbor(0),theSim.now(),paramlist ) );
```

6.2.3 IPRouter.java

In order to implement DiffServ to the network simulator, *IPRouter* needed to be added some attributes and methods. Some of the existing methods in the *IPRouter* also have been modified to meet the need of DiffServ. Some attributes added to the *IPRouter* are user input parameters while some are used for display purpose. The attributes that have been added to *IPRouter*'s header file are as follow:

```
//User initial buffer size for each PHB queue
private SimParamInt sw_efqsize;
private SimParamInt [] sw_afqsize;
private SimParamInt sw_csqsize;
private SimParamInt sw_beqsize;

//Display total frames sent out from source router for each PHB
protected SimParamInt countEF;
protected SimParamInt countAF1;
protected SimParamInt countAF2;
protected SimParamInt countAF3;
protected SimParamInt countAF4;
protected SimParamInt countCS;
protected SimParamInt countBE;

//Display total frames received at destination router for each PHB
protected SimParamInt received_countEF;
protected SimParamInt received_countAF1;
protected SimParamInt received_countAF2;
protected SimParamInt received_countAF3;
protected SimParamInt received_countAF4;
protected SimParamInt received_countCS;
protected SimParamInt received_countBE;
```

In order to send and receive packets, router must maintain buffer (queue) to keep every packet that will be sent out or received. The number of buffer will depend on how the router is going to be implemented. For this network simulator, the router will create one

in-queue and seven out-queues and maintains the queues during simulation. All the buffer queues are created as follow:

```
protected class Port implements Serializable {
//Creates in-queue now
    java.util.List spq = null;      //Create a queue for all packets
    int spq_size;
//Creates out-queue now
    java.util.List efq = null;      //Create a queue for EF packets
    int efq_size;
    AFQ [] afq = null;              //Create a queue for AF packets
    int afqindex = 0;
    java.util.List csq = null;      //Create a queue for CS packets
    int csq_size;
    java.util.List beq = null;      //Create a queue for BE packets
    int beq_size;
    :
}
```

There are two major parts in this class. The first part is to handle the overall send functionality and the second part is to handle the receive functionality. Besides that, router needs to keep track of source IP address for the packet forwarding usage.

To handle the send functionality, means send the packets received from application to the link, the router need to perform a few methods in this class. Firstly, the *action* method brings it to the *sw_receive* method when the router starts to receive packets. Secondly, the packet goes to *sw_my_receive* method when the router checked out that it was come from application. At this point, the method will fill in necessary information for this packet. The *sw_my_receive* method is as follow:

```
protected void sw_my_receive(SimEvent e) {
    :
    if(src.getCompClass().equals("Link")) {
      :          //Perform necessary function here
    }
    else {       //IP Packet from applications
      //Get the necessary parameters for packet
      Object [] params = ( Object [] )e.getParams();
      IPPacket packet = ( IPPacket )params[0];
      packet.TOS = ( (Integer)params[1] ).intValue();
      packet.sourceIP=getMyIP();

      //Then send for processing as usual
      sw_receive_IP(packet,null);
    }
```

Thirdly, the packet goes to *sw_receive_IP* method. Fourthly, the *sw_receive_IP* method will forward the packet to *send_etherframe* method. The function of *send_etherframe* method is to convert the packet to Ethernet frame. That means a new frame is being created here and the new frame will be filled with necessary information including TOS value of the packet. It is important for the frame to keep the TOS value along the way because the value will be used for scheduling algorithm and the destination router will need to use TOS value to check out which PHB's frame it received.

Fifthly, *sw_send_spq* method adds the Ethernet frame to the in-queue, *spq*. This method schedules a processing slot if *spq* is not empty. Also, it is important that the frame is sent out with its TOS value. When the *sw_proc_slot_time* method is called, that particular frame will be removed from *spq*. Here, the demultiplexing operations and buffer management will be performed. The important functions of this method are as follow:

```
protected void sw_proc_slot_time(SimEvent e) {
    :
    //In order to implement DiffServ, user must disable RED
    if( sw_red.getValue() == true ) {     //whether using RED
      :
    }
    else {          //RED is disabled
      //Perform demultiplex operations here
      if( frame.ds_type == IPPacket.TOS_EF ) {
         :  //If the EF queue is not full, add the frame else drop it
      }
      else if( frame.ds_type == IPPacket.TOS_AF1 ) {
         :  //If the AF1 queue is not full, add the frame else drop it
      }
      else if( frame.ds_type == IPPacket.TOS_AF2 ) {
         :  //If the AF2 queue is not full, add the frame else drop it
      }
      else if( frame.ds_type == IPPacket.TOS_AF3 ) {
         :  //If the AF3 queue is not full, add the frame else drop it
      }
      else if( frame.ds_type == IPPacket.TOS_AF4 ) {
         :  //If the AF4 queue is not full, add the frame else drop it
      }
      else if( frame.ds_type == IPPacket.TOS_CS ) {
         :  //If the CS queue is not full, add the frame else drop it
      }
      else {        //if( frame.ds_type == IPPacket.TOS_BE )
         :  //If the BE queue is not full, add the frame else drop it
      }
    }

    //Schedule next processing slot if needed
    :
    //Output frame to link if possible
    :
}
```

Lastly, the frame will be scheduled out to the link according the priority of different PHBs using WRR mechanism. The method is scheduling frame as follow:

```
protected void sw_schedule_output(Port voport) {
      :
      if( !voport.efq.isEmpty() ) {
            :     //Remove frame from this queue
            :     //Updated total frames sent out
      }
      else if( !voport.afq[0].ptr.isEmpty()
            || !voport.afq[1].ptr.isEmpty()
            || !voport.afq[2].ptr.isEmpty()
            || !voport.afq[3].ptr.isEmpty() ) {
            :     //Remove frame from this queue
            :     //Updated total frames sent out
      }
      else if( !voport.csq.isEmpty() ) {
            :     //Remove frame from this queue
            :     //Updated total frames sent out
      }
      else {      //Schedule BE frame
            :     //Remove frame from this queue
            :     //Updated total frames sent out
      }

      //Enqueue the frame to link with frame ID and TOS value
}
```

In order to handle the receive functionality, means receive the frames from the link, the *action* method brings it to the *sw_receive* method when the router start to receive frames from link. The method does the function as follow:

```
protected void sw_receive(SimEvent e) {
      if(src.getCompClass().equals("Link")) {
            //Updated total frames received (all PHBs' frames)
            //Check out the type of PHB for the frame received
            //Then updated the total frames received for that PHB
      }
      else {
            //Process the frame received from application
      }
}
```

6.2.4 EtherFrame.java

EtherFrame can also be considering as one of the fundamental objects in the network simulator. The *TOS constants* are declared as DSCP information for each frame. It is used to provide information about the type of PHB that particular frame belongs to when a packet is being converted to a frame. The *TOS* coding is as follow:

```
//Default TOS value for frame
public static int ds_type=0;

//TOS constants for frame
public static final int EF  = 0xB8;      //Expedited Forwarding
public static final int AF1 = 0x28;      //Assured Forwarding
public static final int AF2 = 0x48;      //Assured Forwarding
public static final int AF3 = 0x68;      //Assured Forwarding
public static final int AF4 = 0x88;      //Assured Forwarding
public static final int CS  = 0x38;      //Class Selector
public static final int BE  = 0;         //Default (Best-Effort)
```

6.2.5 GenericLink.java

GenericLink plays an important role in the simulator as it forwards frames from source router to the destination router. From another point of view, this class is going to send the bits along its way. As mentioned earlier, the TOS value needed by the destination router to count the total frames received for each PHB. That means this class also declares *Object* to get the necessary information when it received frames from source router. After that, it will send the bits down to the destination router with TOS value so that the destination router can updated the total frames received for each PHB. The functions performed by this class's receive method is as follow:

```
private void ln_receive(SimEvent e) {
      //Get all the Object parameters
            :
      //For each GenericLink
            :
            //Create a new Object with 2 elements
            //Assigned the first element as bits ID
            //Assigned the second element as TOS value
            //Send it to the destination router with these information
}
```

CHAPTER 7 TESTING

Testing is done step by step to compare the simulation result by running the simulator with different priority scheduling algorithm. The simulator could be tested by various conditions in two parts which are component testing and system testing.

7.1 Component Testing

The purpose of the component testing is to ensure that parameters, attributes and methods perform in every class is running without error during simulation. Most of the testing is done by observing the output parameters generated during simulation.

7.1.1 UDP_CBR Testing

The test is performed to check whether this component is able to get the correct value of the DiffServ class entered by user. Besides, it also check the correct TOS value assigned when the packet is being sent to router. The *System.out.println* statement is used for testing purpose as stated follow:

```
System.out.println( "DiffServ Class: " + cn_ds_class.getValue() );
System.out.println( "TOS value: " + param[1]);
```

7.1.2 UDP_CBR Testing Results

The tests are executed using all different DiffServ class. The correct values are expected for the output from the additional debugging codes. Table 7.1 below shows the consistent results of the testing.

Table 7.1: Testing results for UDP_CBR.

Test Condition		Value Entered By User	Expected Output	Output
1	DiffServ Class	Best Effort	0	0
	TOS value	Best Effort	0	0
2	DiffServ Class	Class Selector	1	1
	TOS value	Class Selector	56	56
3	DiffServ Class	Assured Forwarding 1	2	2
	TOS value	Assured Forwarding 1	136	136
4	DiffServ Class	Assured Forwarding 2	3	3
	TOS value	Assured Forwarding 2	104	104
5	DiffServ Class	Assured Forwarding 3	4	4
	TOS value	Assured Forwarding 3	72	72
6	DiffServ Class	Assured Forwarding 4	5	5
	TOS value	Assured Forwarding 4	40	40
7	DiffServ Class	Expected Forwarding	6	6
	TOS value	Expected Forwarding	184	184

7.1.3 IPRouter Testing

The test is performed to check whether this component is able to get the correct TOS value entered by user when the received function is called. The total counts for frames sent out and frames received also have been tested to ensure that the appropriate counter is updated based on the TOS value. The *System.out.println* statement is used for testing purpose as stated follow:

```
/*Check with this statement when received frames from links        */
        System.out.println( "TOS value: " + frame.ds_type );

/*Check with this statement for all methods that may receive frames from application or
methods that may receive frames which is still in the process of scheduling out the router
*/
        System.out.println( "TOS value: " + IPPacket.TOS );

/*For the method sw_schedule_output, for each PHB, if the frame is scheduled out, the
statement will be printed and the total frames sent out should be updated once      */
        //For scheduling EF frame
        System.out.println( "One EF frame has been scheduled out" );
        System.out.println( "Total EF frames sent out: " + countEF.getValue() );

        //For scheduling AF1 frame
        System.out.println( "One AF1 frame has been scheduled out" );
        System.out.println( "Total AF1 frames sent out: " + countAF1.getValue() );
```

```
//For scheduling AF2 frame
System.out.println( "One AF2 frame has been scheduled out" );
System.out.println( "Total AF2 frames sent out: " + countAF2.getValue() );

//For scheduling AF3 frame
System.out.println( "One AF3 frame has been scheduled out" );
System.out.println( "Total AF3 frames sent out: " + countAF3.getValue() );

//For scheduling AF4 frame
System.out.println( "One AF4 frame has been scheduled out" );
System.out.println( "Total AF4 frames sent out: " + countAF4.getValue() );

//For scheduling CS frame
System.out.println( "One CS frame has been scheduled out" );
System.out.println( "Total CS frames sent out: " + countCS.getValue() );

//For scheduling BE frame
System.out.println( "One BE frame has been scheduled out" );
System.out.println( "Total BE frames sent out: " + countBE.getValue() );
```

/*Check the total frames received from link and total frames received for each PHB by comparing the total frames scheduled out. The respective total should be the same value */

```
System.out.println("Total frames received: "+sw_frames_received.getValue());
System.out.println("Total EF frames received: "+received_countEF.getValue());
System.out.println("Total AF1 frames received: "+received_countAF1.getValue());
System.out.println("Total AF2 frames received: "+received_countAF2.getValue());
System.out.println("Total AF3 frames received: "+received_countAF3.getValue());
System.out.println("Total AF4 frames received: "+received_countAF4.getValue());
System.out.println("Total CS frames received: "+received_countCS.getValue());
System.out.println("Total BE frames received: "+received_countBE.getValue());
```

7.1.4 IPRouter Testing Results

The tests are executed for a number of times. The correct values are expected for the output from the additional debugging codes. Table 7.2 below shows the consistent results of the testing.

Table 7.2: Testing results for IPRouter.

	Test Condition	Value Entered By User	Expected Output	Output
1	TOS value (frame received from link)	Best Effort	0	0
	TOS value for the relevant methods (frame received from application)	Best Effort	0	0
	Total frames sent out	Best Effort	The statement printed. Total BE frames sent out is updated.	The statement printed. Total BE frames sent out is updated.
	Compare total frames sent out with total frames received	Best Effort	The comparison result is same.	The comparison result is same.
2	TOS value (frame received from link)	Assured Forwarding 1	0	0
	TOS value for the relevant methods (frame received from application)	Assured Forwarding 1	0	0
	Total frames sent out	Assured Forwarding 1	The statement printed. Total AF1 frames sent out is updated.	The statement printed. Total AF1 frames sent out is updated.
	Compare total frames sent out with total frames received	Assured Forwarding 1	The comparison result is same.	The comparison result is same.

3	TOS value (frame received from link)	Assured Forwarding 3	0	0
	TOS value for the relevant methods (frame received from application)	Assured Forwarding 3	0	0
	Total frames sent out	Assured Forwarding 3	The statement printed. Total AF3 frames sent out is updated.	The statement printed. Total AF3 frames sent out is updated.
	Compare total frames sent out with total frames received	Assured Forwarding 3	The comparison result is same.	The comparison result is same.
4	TOS value (frame received from link)	Expected Forwarding	0	0
	TOS value for the relevant methods (frame received from application)	Expected Forwarding	0	0
	Total frames sent out	Expected Forwarding	The statement printed. Total EF frames sent out is updated.	The statement printed. Total EF frames sent out is updated.
	Compare total frames sent out with total frames received	Expected Forwarding	The comparison result is same.	The comparison result is same.

7.2 System Testing

System testing is done by building a new testing topology to test the whole simulator to ensure that it runs on the actual DiffServ environment. The topology showed in Figure 7.1 has been used for this testing session. In this topology, there are 2 IP routers, one generic link and eight UDP CBR applications.

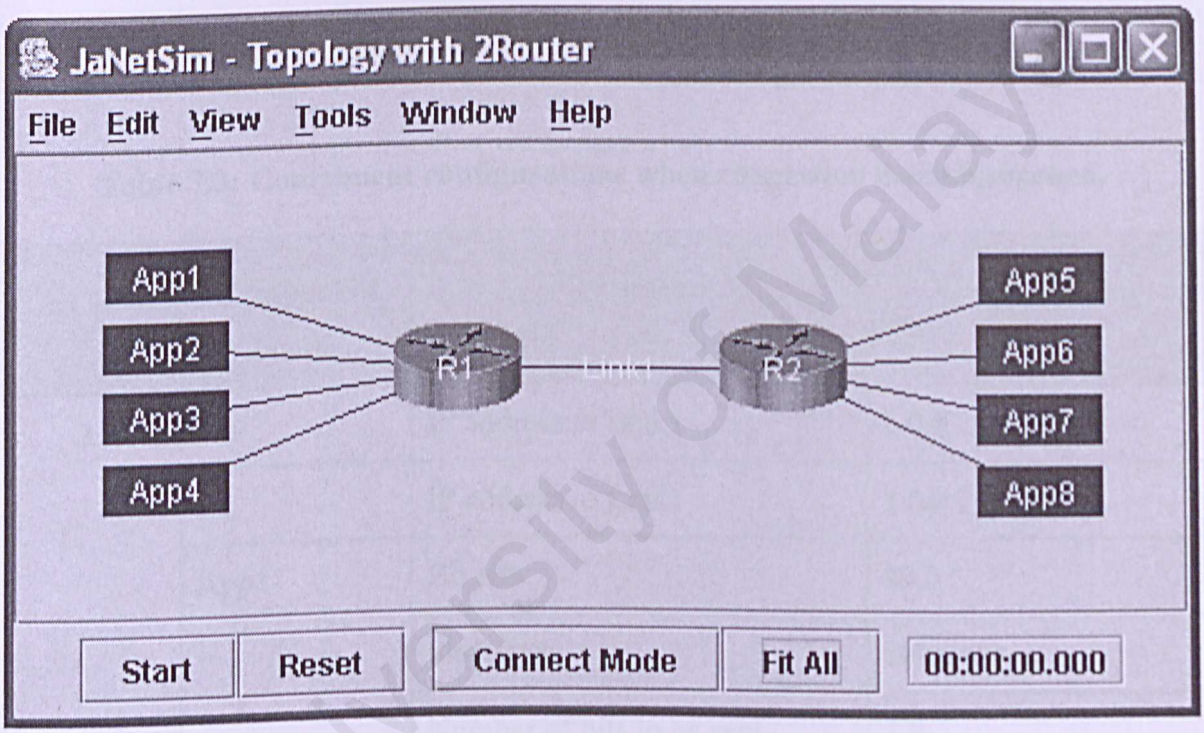


Figure 7.1: Testing Topology.

Two different types of system testing are done, they are:

- Compare the priority scheduling for different PHBs when the congestion is not happened.
- Compare the priority scheduling for different PHBs when the congestion is happened.

7.2.1 Component Configurations

In order to thoroughly test the simulation regarding the priority scheduling for different PHBs when the congestion is not happened during simulation, the necessary configuration of components can be done as stated in Table 7.3. Those parameters that are not showed in Table 7.3 have been set to get the default value. For the clearer understanding, please refer to Appendix for the GUI (property dialogs) for each type of components in the simulation topology.

Table 7.3: Component configurations when congestion is not happened.

Test Case	Component Name	Parameter	Configuration
1	R1	IP address to Link1	1.0.0.1
	R2	IP address to Link1	1.0.0.2
	App1	Bit rate	10.0
		Start time	1000000
		Number of bits to be sent	1.0
		Repeat count	1
		DiffServ Class	Best Effort
		Random destination	Disabled
		Destination IP	1.0.0.2
	App2	Bit rate	10.0
		Start time	1000000
		Number of bits to be sent	1.0

		Repeat count	1
		DiffServ Class	Class Selector
		Random destination	Disabled
		Destination IP	1.0.0.2
	App3	Bit rate	10.0
		Start time	1000000
		Number of bits to be sent	1.0
		Repeat count	1
		DiffServ Class	Assured Forwarding 1
		Random destination	Disabled
		Destination IP	1.0.0.2
	App4	Bit rate	10.0
		Start time	1000000
		Number of bits to be sent	1.0
		Repeat count	1
		DiffServ Class	Expected Forwarding
		Random destination	Disabled
		Destination IP	1.0.0.2
	App5	Bit rate	10.0
		Start time	1000000
		Number of bits to be sent	1.0
		Repeat count	1
		DiffServ Class	Best Effort

		Random destination	Disabled
		Destination IP	1.0.0.1
	App6	Bit rate	10.0
		Start time	1000000
		Number of bits to be sent	1.0
		Repeat count	1
		DiffServ Class	Class Selector
		Random destination	Disabled
		Destination IP	1.0.0.1
	App7	Bit rate	10.0
		Start time	1000000
		Number of bits to be sent	1.0
		Repeat count	1
		DiffServ Class	Assured Forwarding 1
		Random destination	Disabled
		Destination IP	1.0.0.1
	App8	Bit rate	10.0
		Start time	1000000
		Number of bits to be sent	1.0
		Repeat count	1
		DiffServ Class	Expected Forwarding
		Random destination	Disabled
		Destination IP	1.0.0.1

In order to thoroughly test the priority scheduling for different PHBs when the congestion is happened during simulation, the necessary configuration of components can be done as stated in Table 7.4. Those parameters that are not showed in Table 7.4 have been set to get the default value.

Table 7.4: Component configurations when congestion is happened.

Test Case	Component Name	Parameter	Configuration
1	R1	Switching Speed	100
		IP address to Link1	1.0.0.1
	R2	Switching Speed	100
		IP address to Link1	1.0.0.2
	Link1	Link Speed	50.0
	App1	Bit rate	100.0
		Start time	1000000
		Number of bits to be sent	1.0
		Repeat count	1
		DiffServ Class	Best Effort
		Random destination	Disabled
		Destination IP	1.0.0.2
	App2	Bit rate	100.0
		Start time	1000000
		Number of bits to be sent	1.0

		Repeat count	1
		DiffServ Class	Class Selector
		Random destination	Disabled
		Destination IP	1.0.0.2
	App3	Bit rate	100.0
		Start time	1000000
		Number of bits to be sent	1.0
		Repeat count	1
		DiffServ Class	Assured Forwarding 1
		Random destination	Disabled
		Destination IP	1.0.0.2
	App4	Bit rate	100.0
		Start time	1000000
		Number of bits to be sent	1.0
		Repeat count	1
		DiffServ Class	Expected Forwarding
		Random destination	Disabled
		Destination IP	1.0.0.2
	App5	Bit rate	100.0
		Start time	1000000
		Number of bits to be sent	1.0
		Repeat count	1
		DiffServ Class	Best Effort

		Random destination	Disabled
		Destination IP	1.0.0.1
	App6	Bit rate	100.0
		Start time	1000000
		Number of bits to be sent	1.0
		Repeat count	1
		DiffServ Class	Class Selector
		Random destination	Disabled
		Destination IP	1.0.0.1
	App7	Bit rate	100.0
		Start time	1000000
		Number of bits to be sent	1.0
		Repeat count	1
		DiffServ Class	Assured Forwarding 1
		Random destination	Disabled
		Destination IP	1.0.0.1
	App8	Bit rate	100.0
		Start time	1000000
		Number of bits to be sent	1.0
		Repeat count	1
		DiffServ Class	Expected Forwarding
		Random destination	Disabled
		Destination IP	1.0.0.1

2	R1	Switching Speed	100
		EF Q size, AF1 Q size, AF2 Q size, AF3 Q size, AF4 Q size, CS Q size, BE Q size	100
		IP address to Link1	1.0.0.1
	R2	Switching Speed	100
		EF Q size, AF1 Q size, AF2 Q size, AF3 Q size, AF4 Q size, CS Q size, BE Q size	100
		IP address to Link1	1.0.0.2
	Link1	Link Speed	50.0
	App1	Bit rate	100.0
		Start time	1000000
		Number of bits to be sent	1.0
		Repeat count	1
		DiffServ Class	Best Effort
		Random destination	Disabled
		Destination IP	1.0.0.2
	App2	Bit rate	100.0
		Start time	1000000
		Number of bits to be sent	1.0
		Repeat count	1
		DiffServ Class	Class Selector
		Random destination	Disabled

		Destination IP	1.0.0.2
	App3	Bit rate	100.0
		Start time	1000000
		Number of bits to be sent	1.0
		Repeat count	1
		DiffServ Class	Assured Forwarding 1
		Random destination	Disabled
		Destination IP	1.0.0.2
	App4	Bit rate	100.0
		Start time	1000000
		Number of bits to be sent	1.0
		Repeat count	1
		DiffServ Class	Expected Forwarding
		Random destination	Disabled
		Destination IP	1.0.0.2
	App5	Bit rate	100.0
		Start time	1000000
		Number of bits to be sent	1.0
		Repeat count	1
		DiffServ Class	Best Effort
		Random destination	Disabled
		Destination IP	1.0.0.1
	App6	Bit rate	100.0

		Start time	1000000
		Number of bits to be sent	1.0
		Repeat count	1
		DiffServ Class	Class Selector
		Random destination	Disabled
		Destination IP	1.0.0.1
	App7	Bit rate	100.0
		Start time	1000000
		Number of bits to be sent	1.0
		Repeat count	1
		DiffServ Class	Assured Forwarding 1
		Random destination	Disabled
		Destination IP	1.0.0.1
	App8	Bit rate	100.0
		Start time	1000000
		Number of bits to be sent	1.0
		Repeat count	1
		DiffServ Class	Expected Forwarding
		Random destination	Disabled
		Destination IP	1.0.0.1

7.2.2 Simulation Results

The simulation results are divided into two major phases. Phase 1 is simulating the network without any congestion while phase 2 is simulating the network with congestion. The relation between the bit rates, switching speed and link speed plays very important role. It may decide whether the network is congested or not. In order to ensure the simulator runs properly, the application's bit rate should not be greater than the switching speed.

To set the network traffic to be very smooth for the simulation, users just need to let the switching speed to be very large as compare to link speed. In more concrete words, the switching speed should be at least 2 times of link speed.

When the congestion is not happened, all the PHBs' frames will seem to be scheduled out and received at the same times. It is hard for us to notice there is a priority scheduling algorithm. This is because when the link is able to afford the traffic load, the buffer queues, which are used to keep frames, always empty. When the method in the class wishes to schedule a low priority PHB's frame, it will allow the frame to be scheduled out as it found that all higher priority PHB's queues are empty.

To set the simulation to be congested, configure the applications' bit rates equals to switching speed and the link speed should be less than switching speed at least 50%. That means when switching speed is set to 100Mbits/s, the link speed should be 50Mbits/s or lower.

When the congestion is happened, the priority to schedule the frames becomes very clear. It has proved that DiffServ can work well in the simulator. When the traffic is too heavy while the buffer size is set until it is not able to keep incoming frames anymore, the frames is dropped and the total frames dropped are updated.

The following discussed about the simulation results which have been done during system testing:

i. Congestion not happened: Test case 1

- EF, AF1, CS and BE frames seems to be scheduled out with the same priority.
- EF, AF1, CS and BE frames seems to be received at the destination router with the same priority.
- The total frames scheduled out and total frames received for each PHB can be consider same nearly all the time.

ii. Congestion happened: Test case 1

- Firstly, EF frames scheduled out. After finish scheduled out EF frames, the AF1 frames take turns. Then the CS frame and lastly BE frames take turns to schedule out.
- Firstly, EF frames received. After finish receiving EF frames, the AF1 frames take turns. Then the CS frame and lastly BE frames take turns to be received by destination router.
- The priority scheduling is clear at all and no frames are being dropped.

iii. Congestion happened: Test case 2

- Firstly, EF frames scheduled out. After finish scheduled out EF frames, the AF1 frames take turns. Then the CS frame and lastly BE frames take turns to schedule out.
- Firstly, EF frames received. After finish receiving EF frames, the AF1 frames take turns. Then the CS frame and lastly BE frames take turns to be received by destination router.
- The priority scheduling is clear at all.
- Frames which can not be scheduled out in time make the buffer queue full. Incoming frames with the same PHB are being dropped until this particular PHB frames are being scheduled out.

CHAPTER 8 CONCLUSION

A lot of knowledge and experiences were gained throughout the development of the simulator. The most valuable experience is to study and understand more detail into DiffServ and WRR scheduling algorithm. All the problems encountered and experiences gained during the development of this simulator should be very useful in my future endeavors.

This thesis managed to achieve the overall project objectives and goals, i.e. development of object oriented and multithreading network simulator. This thesis is also able to classify packets using 4 different PHBs and scheduling the packets with different priority using WRR mechanism. Lastly, the following highlights system strengths, system limitations, as well as the proposed future enhancements.

8.1 System Strengths

The system strengths are described as follow:

- The design of network simulator is user friendly and easy to use. The user can easily add a component to the topology and simulate the network.
- The simulator is fully object-oriented whereby all the functions and modules are built in class.
- The simulator is able to treat the traffic with different priority scheduling algorithm. It is able to recognize different classification of packets.

8.2 System Limitation

The system limitations are described as follow:

- Functions of this simulator are not as many as the existing NS2 network simulator.
- This simulator is not able to send packets across more than one router as the dynamic routing protocol is not completed.
- Help File is very important in any applications. It is definitely a good strategy if the help file is implementing in this simulator. Due to time constraints, this simulator does not include a complete help file.
- The GUI may not as attractive as the other programs.

8.3 Future Enhancements

Due to the limitation of this simulator, there are a few suggestions that may be useful for future enhancement of this simulator listed as following:

- It is hoped that this simulator can be added in more functions to simulate more complex network traffics.
- It is hoped that this simulator is extended to support dynamic routing protocol.
- It is hoped that a complete and useful help file can be included in this simulator to provide necessary information for those who may need it.
- It is hope that the GUI can be enhanced so that it is more attractive. For example, the GUI for graph could be presented to user as an interesting interface.

REFERENCES

[1] Black, D., Blake, S., Carlson, M., Davies, E., Wang, Z. and W. Weiss. (December 1998). Architecture for Differentiated Services, RFC 2475.

[2] Braden R., Clark D., Shenker S., (June 1994). Integrated Services in the Internet Architecture: An Overview, RFC1633.

[3] Braden R., Zhang L., Berson S., Herzog S. and Jamin S., (September 1997). Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification, RFC2205.

[4] Callon, R., Doolan, P., Feldman, N., Fredette, G., Swallow, G. and Viswanathan, A., (September 1999). A Framework for Multiprotocol Label Switching, Internet Draft.

[5] Cisco System, Inc., (2001). DiffServ – The Scalable End-to-End QoS Model. White Paper.

[6] Cisco Systems, Inc., (2003). Quality of Services Networking. Available from: http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/qos.pdf [Accessed 10 July 2003]

[7] Davie, B., Charny A., Bennett J.C.R., Benson K., Boudec J.Y. Le, Courtney W., Davari S., Firoiu V., Stiliadis D., (March 2002). An Expedited Forwarding PHB (Per-Hop Behavior), RFC3246.

[8] Differentiated Services Working Group Charter, (3 February 2003). Differentiated Services (DiffServ). Available from: http://www.ietf.org/html.charters/OLD/DiffServ-charter.html [Accessed 3 July 2003].

[9] Grossman, D., (April 2002). New Terminology and Clarifications for DiffServ, RFC3260.

[10] Heinanen, J., Baker, F., Weiss, W., Wroclawski, J., (June 1999). Assured Forwarding PHB Group, RFC2597.

[11] IP over ATM: Classical IP, NHRP, LANE, MPOA, PAR and I-PNNI. Available from: http://www.cis.ohio-state.edu/~jain/cis788-97/ftp/ip_over_atm/index.htm [Accessed 8 July 2003]

[12] Jacobson, V., Nichols, K., Poduri, K., (June 1999). An Expedited Forwarding PHB, RFC2598.

[13] Lim, S.H., (2001). Traffic Engineering Enhancement to OSPF for IP QoS with DiffServ and MPLS. Thesis (Degree). University of Malaya.

[14] Nicolas, C., (October 2000). Current Directions in the DiffServ World. Department of Computer Science, University of Virginia.

[15] Nichols, K., Blake, S., Baker, F. and D. Black. (December 1998). Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, RFC 2474.

[16] Nichols, K., Blake, S., (February, 1998). Differentiated Services Operational Model and Definitions. Internet Draft. Available from: http://ds.internic.net/internet-drafts/draft-nichols-dsopdf-00.txt [Accessed 2 July 2003].

[17] Nichols, K., Jacobson, V., Zhang, L., (November 1997). A 2-Bit Differentiated Services Architecture for the Internet. Internet Draft. Available from: http://ds.internic.net/internet-drafts/draft-nichols-diff-svc-arch-00.txt [Accessed 2 July 2003].

[18] Pieda, P., Ethridge, J., Baines, M., and F. Shallwani., (July 2000). A Network Simulator Differentiated Services Implementation. Nortel Networks.

[19] Rosen E., Viswanathan, A. and Callon, R., (January 2001). Multiprotocol Label Switching Architecture, RFC3031.

[20] Shenker S., Partridge C., Guerin R., (September 1997). Specification of Guaranteed Quality of Service, RFC2212.

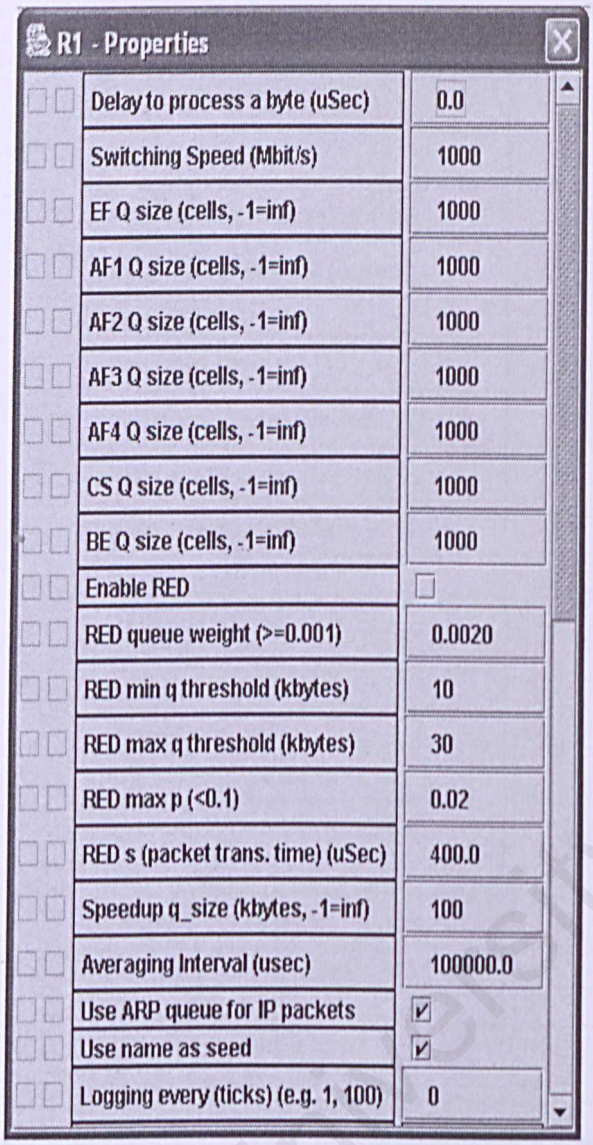
[21] Tanenbaum, A.S., (1996). Computer Networks, 3rd Edition. Prentice-Hall International Inc.

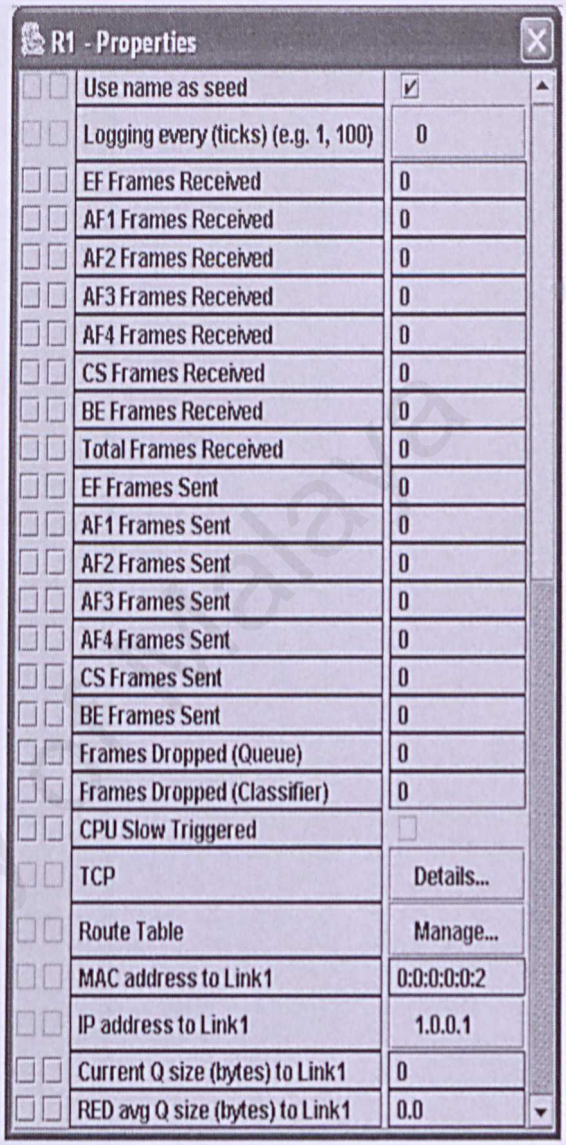
[22] The Network Simulator - ns-2. Available from: http://www.isi.edu/nsnam/ns/ [Accessed 3 July 2003]

[23] Wroclawski, J., (September 1997). The Use of RSVP with Integrated Services, RFC2210.

[24] Wroclawski J., (September 1997). Specification of the Controlled-Load Network Element Service, RFC2211.

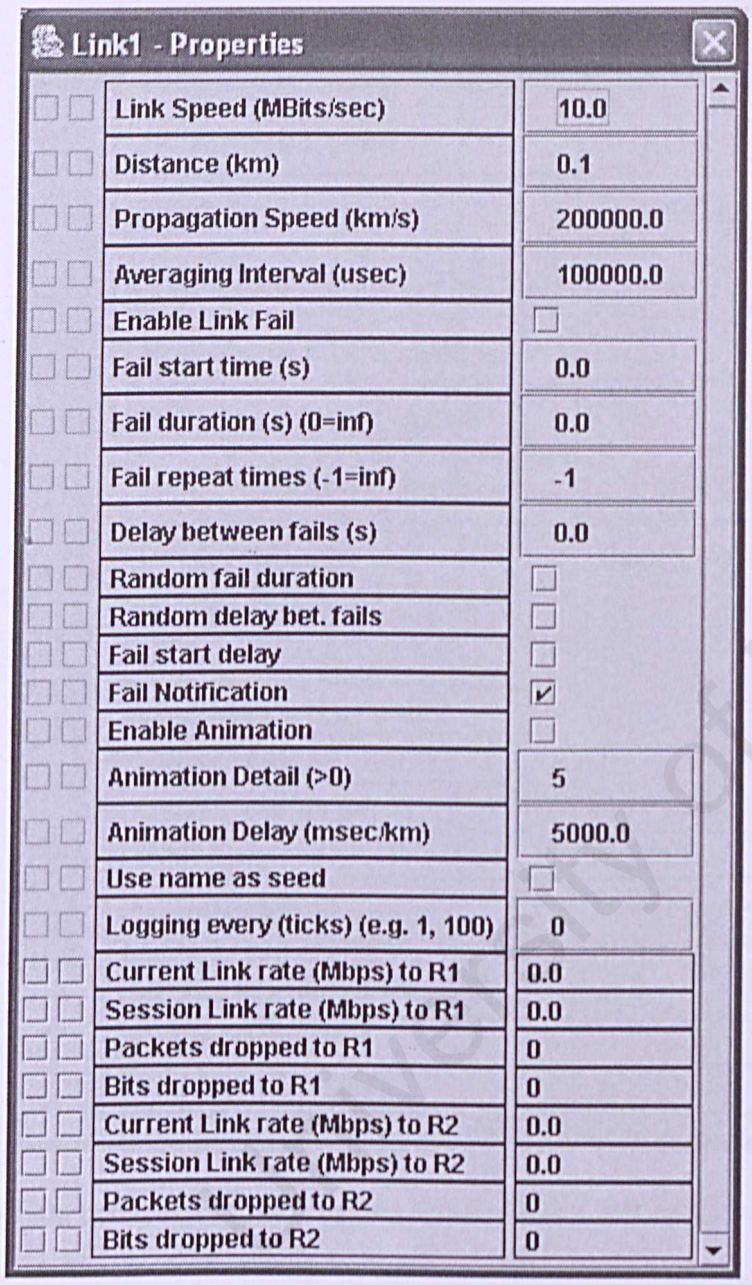
APPENDIX A





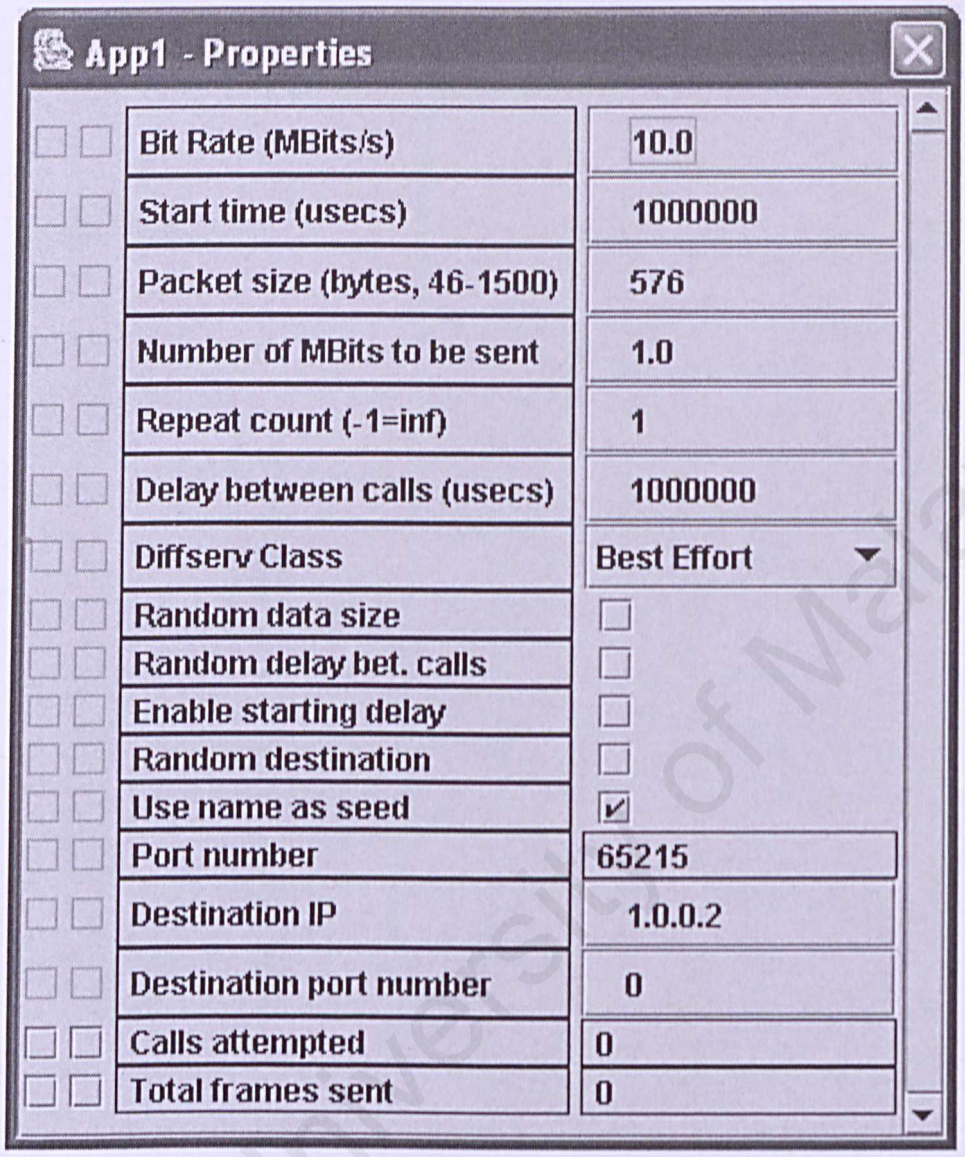
These are screen shots from the same router property dialog. The first screen shot indicates the upper half of the property dialog while the second screen shot indicates the lower half of the property dialog.

APPENDIX B



This is the screen shot for the whole generic link property dialog.

APPENDIX C



This is the screen shot for the whole UDP CBR application property dialog.