Implementation of IPv6 Flow Label in UMJaNetSim

A thesis submitted to

the Faculty of Computer Science & Information Technology,

University of Malaya

in Partial Fulfillment of the Requirements for

the Degree of Bachelor of Computer Science

by

AU YEE BOON

(WEK010019)

Feb, 2004

Supervisor: Mr. Phang Keat Keong

ABSTRACT

The IPv6 Flow Label [Deering&Hinden, 1998a] is defined as a 20-bit field in the IPv6 header which may be used by a source to label sequences of packets for which it requests special handling by the IPv6 routers, such as non-default quality of service or "real-time" service.

This project discusses the implementation of IPv6 flow label to UMJaNetSim. It involves the proposed specification of IPv6 flow label in order to provide a more efficient quality of service of network.

Currently, the IPv6 flow label was still experimental, and subject to change, as the requirements for flow supports in the Internet were evolving.

This project provides an analysis of the IPv6 definition of the flow label, the rules governing its use, and their implications. It subsequently makes a proposal for additions/modifications to these rules, which improve the usability of the IPv6 flow label, in term of providing a better quality of service of network.

The main aim of this project is to enhance the UMJaNetSim with IPv6 Flow Label, and study the uses of IPv6 flow label to provide a more efficient quality of service of network.

ACKNOWLEDGEMENT

I am in deep debt to my supervisor Mr Phang Keat Keong for providing me valuable guidance and assistance throughout the whole my research. His understanding and encouragement are indispensable for the completion of this thesis. I'm also grateful to Mr Ang Tan Fong, my moderator and Mr Ling Teck Chaw for their valuable suggestions and feedback.

I am thankful to Mr Koh Chee Hong for giving me the explanation of the API of UMJaNetSim.

It is impossible for me to name each and every person who contributed to this work, but I would like to thank my fellow researchers namely Andrew Chiam Ming Jer, Chan Chin We, Chee Wai Hong, Chia Kai Yan, Lim Lee Wen, Malini and Tang Geck Hiang.

CONTENTS

List Of Figures

List of Table

ABBREVIATIONS

ARP	Address Resolution Protocol
AS	Autonomous Subnet
ATM	Asynchronous Transfer Mode
ATM LSR	Asynchronous Transfer Mode Label Switching Router
BGP	Border Gateway Protocol
B-TE	Broadband Terminal Equipment
CIDR	Classless Inter-Domain Routing
DNS	Domain Name System
DHCP	Dynamic Host Configuration Protocol
EUI-64	64-bit Extended Unique Identifier
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPsec	Internet Protocol Security
IPv6	Internet Protocol Version 6
IPv4	Internet Protocol Version 4
IPX	Internetwork Packet Exchange
ISP	Internet Service Provider
ICMPv4	Internet Control Message Protocol Version 4
ICMPv6	Internet Control Message Protocol Version 6
LAN	Local Area Network
MAC	Media Access Control

MLD	Multicast Listener Discovery
NAT	Network Address Translator
ND	Neighbor Discovery
OSPF	Open Shortest Path First
PDU	Protocol Data Unit
QoS	Quality of Service
RIPng	Routing Information Protocol next generation
RSVP	Resource ReSerVation Protocol
SLA	Site-level Aggregators
TCP	Transmission Control Protocol
TOS	Type of Service
UDP	User Datagram Protocol
VBR	Variable Bit Rate
WAN	Wide Area Network

Chapter 1 Introduction

1.1 Introduction to Flow Label in IPv6

[Deering&Hinden, 1998a] A flow is a sequence of packets sent from a particular source, and a particular application running on the source host, using a particular host-to-host protocol for the transmission of data over the Internet, to a particular (unicast or multicast) destination, and particular application running on the destination host, or hosts, within a certain set of traffic, and QoS requirements. The IPv6 Flow Label is defined as a 20-bit field in the IPv6 header, as shown in figure 1.1, which may be used by a source to label sequences of packets for which it requests special handling by the IPv6 routers, such as non-default quality of service or "real-time" service.

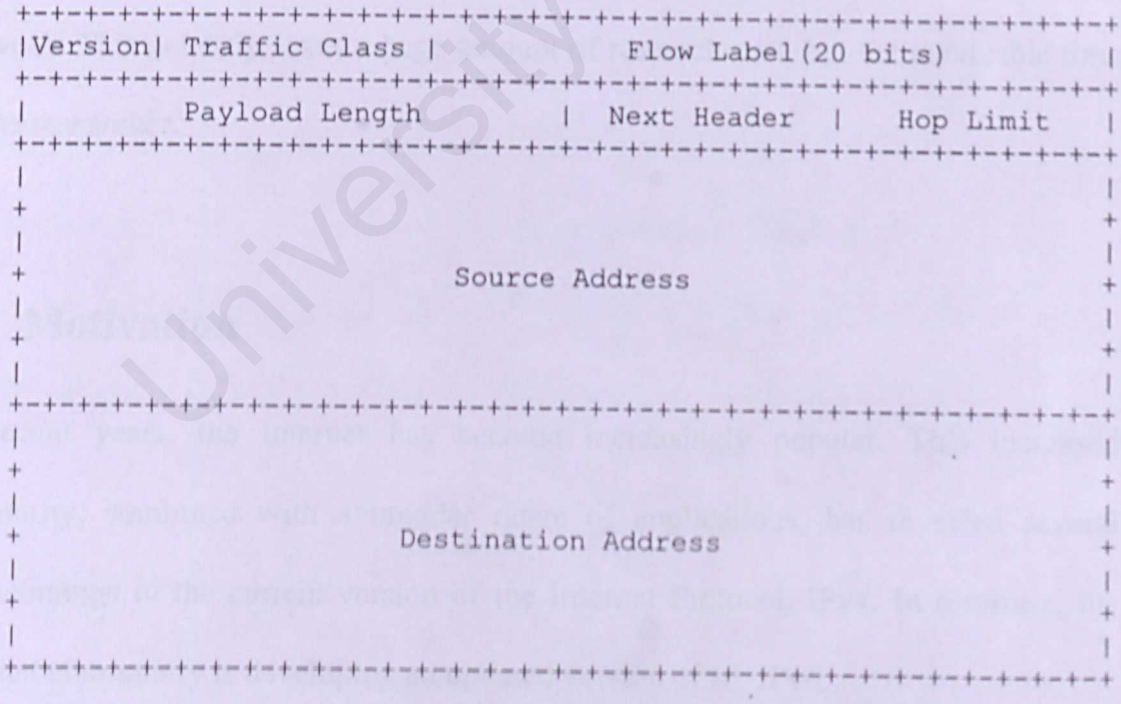


Figure 1.1 IPv6 Header Format

1.2 Introduction to Network Simulator

The Internet's rapid growth has spurred development of new protocols and algorithms to meet changing operational requirements—such as security, multicast transport, mobile networking, policy management, and quality-of-service support. Network simulator plays a valuable role in these network researches. It allow user to make correct decision on designing a network without the need to invest into the technology.

Network researchers can use the network simulator to test Internet protocols under varied conditions to determine whether they are robust and reliable. By using network simulator, network researchers can evaluate their network design under varying network conditions. They can analyze and predict the performance of their network design based on the generated result of network simulator. Hence, network researchers do not have to build a real network in order to develop and analyze a network. This not only saves a huge amount of research cost, but the invaluable time of the researcher.

1.3 Motivation

In recent years, the Internet has become increasingly popular. This increased popularity, combined with a broader range of applications, has revealed several shortcomings in the current version of the Internet Protocol, IPv4. In response, the Internet community is developing an updated version of IP, IPv6.

As the Internet gradually migrates from the current IPv4 to the next generation, IPv6, and a dramatic increase in the number of packet based applications that require end-

to-end Quality of Service (QoS) guarantees, it is expected that there will be a need to the IPv6 based QoS approach.

1.4 Project Objectives

The primary objective of this project is to study and understand the flow label field in IPv6 and uses it to provide an efficient quality of service of network. This involves the research in the flow label specification, IPv6, QoS of network and etc.

The second objective of this project is to enhance the UMJanetSim with the support of flow label field in IPv6. By implement flow label of IPv6 into UMJanetSim, I hope it will improve the QoS of the simulator.

1.5 Project Scope

Enhance the UMJanetSim with the support of IPv6's flow label.

Allow the user to simulate the flow label QoS approach on UMJanetSim

Show the simulation result after the implementation of the flow label of IPv6.

Evaluate the QoS's efficiency gain after implement the flow label of IPv6.

1.6 Project Schedule

The timeline of the project will be show on Figure 1.2. In overall, the project will be start from June 2003 until the end of January 2004.

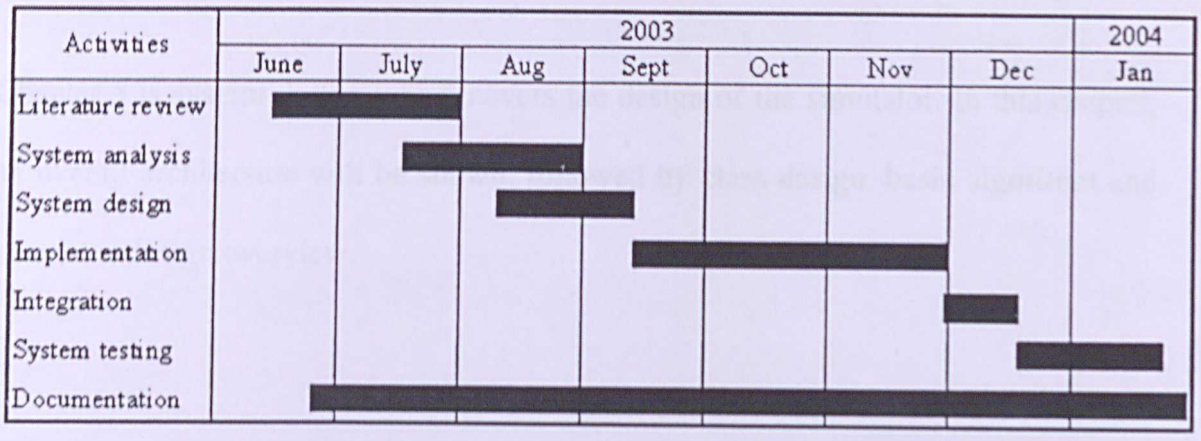


Figure 1.2 Project Schedule

1.7 Project Organization

This report has a total of 5 chapters. It is organized as follows:

Chapter 1 covers an introduction to IPv6 Flow Label, network simulator, motivation, project objectives, project scope and project schedule.

Chapter 2 contains survey on different data communication technologies, especially those related to Quality of Service of network. This includes the introduction to TCP/IP, Introduction to Network QoS, IPv4 TOS, Integrated Service, Differentiated Services, Multi-protocol Label Switching (MPLS), IPv6 and the introduction to the Computer Simulator.

Chapter 3 is concentrate on the IPv6 Flow label. It covers the existing flow label specification and the proposed flow label specification.

Chapter 4 discusses the overview of the system analysis. It talks about the programming language and development tools chosen to create the network simulator components.

Chapter 5 is system design which covers the design of the simulator. In this chapter, an overall architecture will be shown, followed by class design, basic algorithm and simulator design overview.

Chapter 2 Literature Review

2.1 Introduction to TCP/IP

TCP/IP is made up of two acronyms, TCP, for Transmission Control Protocol, and IP, for Internet Protocol. TCP handles packet flow between systems and IP handles the routing of packets. However, that is a simplistic answer that we will expound on further.

All modern networks are now designed using a layered approach. Each layer presents a predefined interface to the layer above it. By doing so, a modular design can be developed so as to minimize problems in the development of new applications or in adding new interfaces.

The ISO/OSI protocol with seven layers is the usual reference model. Since TCP/IP was designed before the ISO model was developed it has four layers; however the differences between the two are mostly minor. Below, is a comparison of the TCP/IP and OSI protocol's Stacks:

Table 2.1 OSI Protocol Stack

7. Application --	End user services such as email.
6. Presentation --	Data problems and data compression
5. Session --	Authentication and authorization
4. Transport --	Guarantee end-to-end delivery of packets
3. Network --	Packet routing
2. Data Link --	Transmit and receive packets
1. Physical --	The cable or physical connection itself.

Table 2.2 TCP/IP Protocol Stack

5. Application	--	Authentication, compression, and end user services.
4. Transport	--	Handles the flow of data between systems and provides access to the network for applications.
3. Network	--	Packet routing
2. Link	--	Kernel OS/device driver interface to the network interface on the computer.
1. Physical	--	The cable or physical connection itself.

Below are the major difference between the OSI and TCP/IP:

i. The application layer in TCP/IP handles the responsibilities of layers 5, 6, and 7 in the OSI model.

ii. The transport layer in TCP/IP does not always guarantee reliable delivery of packets as the transport layer in the OSI model does. TCP/IP offers an option called UDP that does not guarantee reliable packet delivery.

2.1.1 Software Components of TCP/IP

Application Layer

Some of the applications we will cover are SMTP (mail), Telnet, FTP, Rlogin, NFS, NIS, and LPD.

Transport Layer

The transport uses two protocols, UDP and TCP. UDP which stands for User Datagram Protocol does not guarantee packet delivery and applications which use this must provide their own means of verifying delivery. TCP does guarantee delivery of packets to the applications which use it.

Network Layer

The network layer is concerned with packet routing and used low level protocols such as ICMP, IP, and IGMP. In addition, routing protocols such as RIP, OSPF, and EGP will be discussed.

Link Layer

The link layer is concerned with the actual transmittal of packets as well as IP to Ethernet address translation. This layer is concerned with ARP, the device driver, and RARP.

2.2 Introduction to Network Quality of Service (QoS)

The Internet Protocol (IP) and the architecture of the Internet itself are based on the simple concept that every packet is routed through a network based on the destination address contained within the packet. Each router has the routing table that identifies the appropriate next hop for all known IP destination addresses. When a packet arrives, the router simply looks up the routing table and forwards to the output port that goes to the next router. All packets, regardless of which application or service they come from, are treated equally. Routers drop packets indiscriminately when

congestion occurs. Therefore IP can only provide one type of service called Best Effort (BE), and give no guarantees about when data will arrive, or how much it can deliver.

Quality of Service (QoS) refers to the successful delivery of an agreed upon level, or class of service. A class of service is characterized by a set of performance parameters including:

- **Latency** (Delay)

  Refers to the time interval it takes a packet to be forwarded between two reference points;

- **Jitter** (delay variation)

  Refers to the variation in transit time for all packets in a stream taking the same route;

- **Throughput**

  Refers to the rate at which packets go through or transit a network or network device, expressed as an average or peak rate;

- **Packet loss**

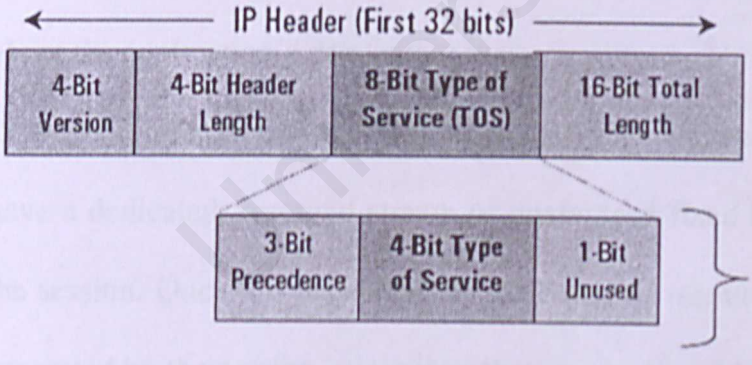
  Refers to the maximum rate at which packets are discarded during transfer through a network.

QoS addresses the issue on how to provide the capability in network elements so that traffic and service requirements can be guaranteed. QoS itself does not generate

bandwidth. The total bandwidth demanded by all the services and applications must be available. QoS can manage to allocate resources to individual data streams with a guaranteed service level. The QoS has to ensure that BE traffic or other low priority services are not starved after reservations are made for high priority applications.

2.3 IPv4 Type of Service (T0S)

TOS field in the IP header was designed to deliver QoS by tagging IP packets with different service characterizations. These service characterizations describe the how network nodes reading the IP header should treat the packet. The most recent TOS specification, RFC 1349, defines the TOS field as a set of bits to be considered collectively. The TOS values, shown in Figure 2.1, denote how the network should treat the packet with respect to tradeoffs between throughput, delay, reliability, and cost.



Binary Value	Service Characterization
1000	Minimize Delay
0100	Maximize Throughput
0010	Maximize Reliability
0001	Minimize Monetary Cost
0000	Normal Service

Figure 2.1 IPv4 TOS

The first 3 precedence bits of the TOS octet are intended to denote the importance or priority of the packet. A network router can use the TOS field when choosing a path over which to forward the packet, and when making queuing decisions. Although the TOS field has been a part of the IP specification since its implementation, it has been

little used in the past. This lack of use can be attributed to poorly defined service characterizations. The ambiguous nature of these service characterizations provided no quantifiable or even relative service parameters, thus making it extremely difficult to differentiate traffic with a consistent level of service quality. The lack of TOS implementation, coupled with its inability to allow an application to quantify the level of service that it desires, makes TOS an inappropriate mechanism for delivering service guarantees.

2.4 Integrated Services

An architecture called the Integrated Services model (IntServ) has been devised by IETF, attempts to layer QoS on top of the best-effort model. Applications specify their bandwidth requirements to the network, which then tries to provide the requested resources. The IETF IntServ working group has established a protocol, Resource reSerVation Protocol (RSVP) [Braden et al, 1997] which, during session set-up, reserves resources (such as bandwidth on an interface) in every intermediate router along the application's data path.

If every hop on the journey agrees to reserve sufficient resources, then the user will have a dedicated, reserved stream of guaranteed fixed bandwidth for the duration of the session. Once the session ends, the reserved resources are dropped. If the traffic generated by the session is less than that required establishing the reservation, then the approach proves futile. Hence, bandwidth reservation is unsuitable for traffic such as e-mail. It is aimed at applications where large amounts of real-time traffic are generated, such as video streaming

2.4.1 Architecture

Internet community has recognized the need for support of new distributed applications mostly arising in the field of multimedia. The current architecture is incapable of providing QoS, which is the main requirement of the emerging applications. The IETF has been for some time examining how the architecture can be enhanced to provide such services. IntServ is an attempt to merge the advantages of two different paradigms:

i. Datagram,

ii. Circuit switched networks.

Like the Internet, datagram networks maximize network utilization by multiplexing multiple data streams. They also provide for multi-point communication and robustness by adapting to network dynamics. Up to now, however, datagram networks only provided a best-effort delivery service.

Conversely, current circuit-switched and ISDN networks inefficiently utilize network resources when sending bursty data traffic, but can provide service guarantees.

The aim set by the IETF is to provide for robust yet flexible services that allow for QoS support and multi-point communication while making efficient use of network resources.

2.4.1.1 Basic Assumptions

Multiparty communication is regarded as one of the main requirements of the new applications, thus multicasting is considered vital for the new Internet infrastructure. In addition to the provision of QoS for real-time applications, the sharing of bandwidth between different traffic classes is considered a desirable characteristic of

the future infrastructure. Therefore, the IntServ model shall include, besides best-effort and real-time services, a so-called controlled-link service. The approach taken to introduce these partially new services is not to design a new architecture from scratch, but to enhance the existing architecture with some new components. The base concept of IntServ is that resources must be managed; otherwise guarantees cannot be given, whether they are statistical or deterministic, strict or approximate.

To perform the resource reservation in the network, the IntServ approach assumes the existence of flow-related state, where each node maintains local information on each flow. This means a partial shift away from one of the basic principles in the Internet; the so-called end-to-end principle. This claims that flow-related state should only be maintained in the end-systems. However, in order to relax this paradigm shift away from connectionless to connection-oriented services, the IntServ model operates on soft state in the network, thereby trying to maintain the robustness characteristics of the Internet. By using soft state, a hybrid form between connectionless and connection-oriented services is used.

Since resource reservation leads to privileges, the IntServ model recognizes the need for policy and administrative control over whose packets receive the contracted QoS, which in turn leads to authentication requirements.

Another assumption of the approach is the integration of real-time and non-real-time communication into a single Internet infrastructure, and thereby enabling statistical sharing between these traffic classes. Hence, a unified protocol stack for both real-time and non-real-time traffic on the Internet layer is envisaged in the implementation of the IntServ model.

2.4.1.2 The Basic Architectural Components

The Integrated Services working group in the IETF has developed an enhanced Internet service model that includes real-time service and best-effort service. Together with RSVP [Wroclawski, 1997a], this architecture is a comprehensive approach to provide applications with the type of service they need, with the quality they choose. To support this capability, two things are required:

i. A method of communicating QoS requirements

ii. Management information between applications and network elements, and between network elements themselves (hosts and routers).

Individual network elements along the data path capable of supporting mechanisms to control the QoS allocated to packets traveling via that network element.

RSVP communicates a request for QoS along the data path in a hop-by-hop fashion, resulting in a reservation if the request was successful. RSVP alone will not give a better QoS. RSVP is only a control protocol that sets up a reservation, but enforcement of the reservation has to be performed by some other component of the architecture.

Routers on the path need to distinguish between best-effort flows and reserved flows, in order to honor reservations. A flow is a stream of related packets from one source to a unicast or multicast destination. Routers use sophisticated scheduling techniques to provide service according to the reservations and to other policies. Reservation set-up, management and enforcement consume bandwidth and processing power in the router. The number of RSVP messages and the amount of state information needed in the routers are proportional to the number of reservations. If a router handles tens of thousands of flows, keeping state information for all these can be a problem. Enforcement of large numbers of reservations at the same time is also very difficult.

On large interfaces, the bottleneck is usually the processing speed of the router, not the bandwidth of the outgoing link.

2.4.1.3 Overview of the Traffic Control Components

The traffic control module consists of three components as shown in Figure 2.3.1; Packet Classifier; Packet Scheduler; and Admission Control.

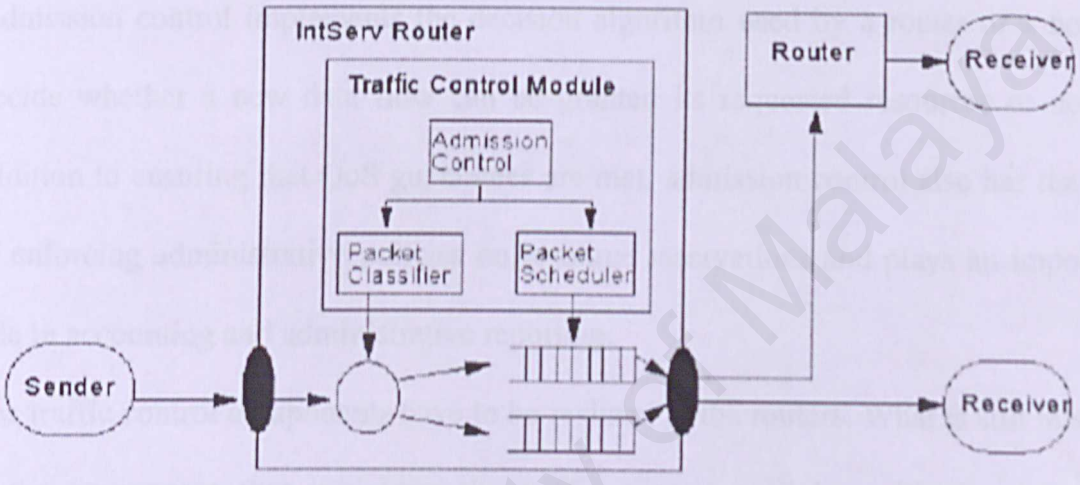


Figure 2.2 Traffic Control Components

Packet Classifier

The packet classifier's task is to perform a mapping of incoming packets into classes, characterized by the fact that all packets of the same class get the same treatment from the packet scheduler. Possible classifications are, for example, all video flows or all flows attributable to an organization. However, a single data flow can also represent a whole class. This will be observed at the edge routers of the network. The Internet core routers should preferably be using aggregation mechanisms. Currently packet classification for routers is complicated by the fact that the destination address is the only source of information for routing purposes in the Internet, is not sufficient for classification of service. (The flow label field of IPv6 should alleviate this problem).

Packet Scheduler

The packet scheduler is responsible the forwarding of different packet streams using a set of queues and possibly other mechanisms like timers. In order to achieve the desired QoS provision, the basic function of the scheduler is to reorder these queues, according to a priority scheme.

Admission Control

Admission control implements the decision algorithm used by a router or a host to decide whether a new data flow can be granted its requested resources or not. In addition to ensuring that QoS guarantees are met, admission control also has the task of enforcing administrative policies on resource reservations and plays an important role in accounting and administrative reporting.

The traffic control components have to be realized in the routers. What is still missing is the component that initializes the traffic control modules with the necessary parameters in a distributed fashion, by conveying the application needs from the end-users to the routers. This is achieved in the IntServ model by the use of RSVP.

Packet Dropping

An augmentation of the service model is the integration of a packet dropping mechanism. This is led by the observation that in many audio and video streams, some packets are less important than others, and should therefore, in a case of congestion, be dropped first. It is therefore a means of delivering application-specific information to the network, in order to assist the network to make more reasonable decisions.

The IntServ model goes one step further by introducing expendable packets, which in contrast to pre-emptable packets do not have to pass admission control any more. The expectation for these packets is that many of them are to be dropped, while for pre-emptable packets most should still be delivered, and therefore they are considered a part of the flow subject to admission control.

2.4.2 Service Models

Two service model has been introduce by the IntServ architecture,

i. Controlled Load [Wroclawski, 1997b].

ii. Guaranteed Service [Shenker et al, 1997].

These specifications determine how traffic is handled within individual network elements.

Guaranteed Service provides firm (mathematically provable) bounds on end-to-end datagram queuing delays. This makes it possible to provide a service that guarantees both bandwidth and delay.

Controlled Load provides the client data flow with a QoS closely approximating the QoS that same flow would receive from an unloaded network element, but uses capacity (admission) control to assure that this service is received even when the network element is overloaded. Hence, the application may assume that a very high percentage of its transmitted packets will successfully reach the destination, and that the transit delay experienced by a very high percentage of the successfully delivered packets will not be significantly bigger than the minimum transit delay of any packet.

The main features of Controlled Load and Guaranteed Service are summarized in Table 2.3:

Table 2.3 IntServ Service Models

Template components	*Controlled Load*	*Guaranteed Service*
End-to-End Behaviour	Approximates best-effort over unloaded network	Guaranteed maximum delay
Motivation	Applications sensitive to network congestion	Real-time applications

2.4.3 Resource Reservation Protocol (RSVP)

Each node capable of resource reservation has several modules that work together for reservation set-up and enforcement (see Figure 2.3). The IntServ architecture of routers and hosts is the same, but the implementation is usually different. An RSVP process on every network element (both hosts and routers) handles all protocol messages needed to set up and tear down reservations.

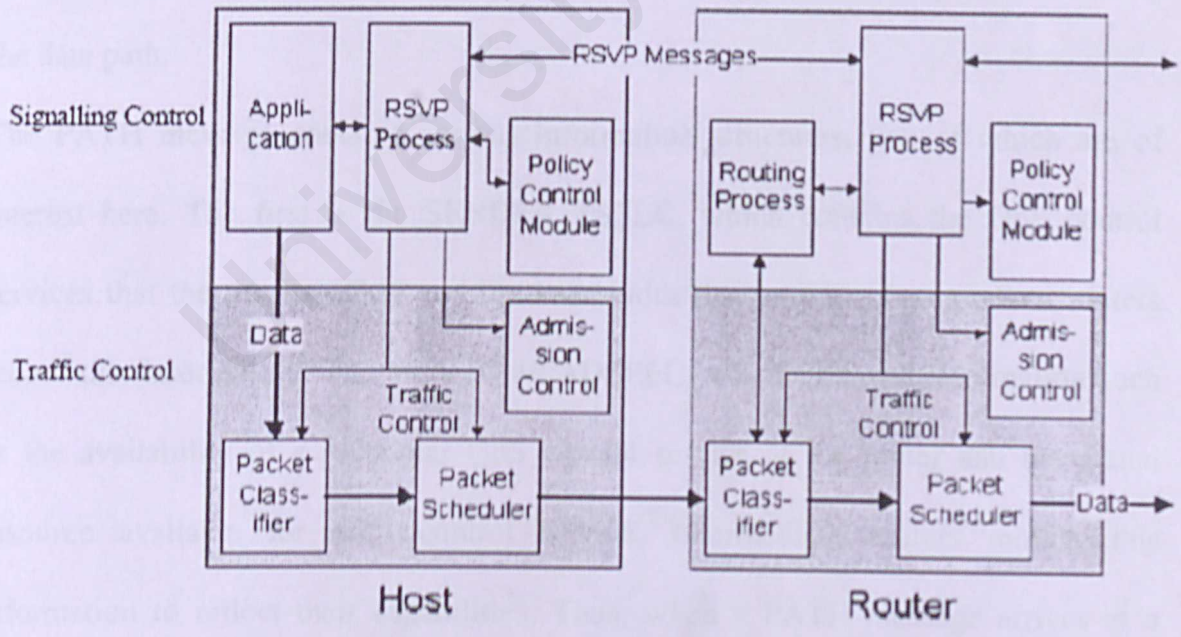


Figure 2.3 RSVP Implementation Overview

An application requests a certain QoS from the RSVP process running on the host. Note that the receiver initiates reservations; if a sender had to maintain a reservation

for each receiver, the protocol would not scale for large multicast groups. A sender does not need to know the number and specifics of reservations or the location of the receivers. In this respect, RSVP closely follows the multicast model. Also, a receiver knows best the level of QoS needed. Different receivers might request and receive different QoS.

The disadvantage of receiver-initiated reservations is that a receiver does not know which path the data packets are taking. The data path from the sender to the receiver might be different from the path from the receiver to the sender. To solve this problem, the sender sends a "PATH" message to the unicast or multicast address of the data receiver. A PATH message contains the unicast address of the next RSVP-capable host upstream (towards the sender). PATH messages also contain information about the expected size of the flow. Every node that receives a PATH message saves this information. It then passes the PATH message on with its own unicast address as the next upstream RSVP hop. Therefore, PATH messages build a trail of path states along the data path.

The PATH message contains various information structures, two of which are of interest here. The first is the SENDER_TSPEC, which contains the QoS control services that the sender offers and the bandwidth they require. Intermediate routers record this information. The other is the ADSPEC, which contains information such as the availability of a particular QoS control service at the router and the actual resource available for each control service. Intermediate routers modify this information to reflect their capabilities. Thus, when a PATH message arrives at a receiver, the ADSPEC information structure contains a summary of the data path's available QoS. Receivers can then use this information to make a QoS reservation that the data path can sustain.

A receiver sends a reservation request ("RESV") message towards the source to make a reservation. Reservation requests specify the requested service and the size of the expected data flow (FLOWSPEC). It is also specified which packets can use the reservation (known as the FILTERSPEC). There are three reservation styles: Fixed Filter, (FF) Shared Explicit, (SE) and Wildcard Filter, (WF)

In fixed filter style, a reservation is made for packets sent by exactly one sender that is specified in the FILTERSPEC. In shared-explicit style, packets from several sources listed explicitly in the FILTERSPEC can use the reservation. In wildcard style, all sources sending to the multicast group address share the reservation.

Each time a "RESV" message is received at a node, the RSVP process checks with the Policy Control module to see if the user has administrative permission to make a reservation. Future accounting for reservations will also be carried out by Policy Control. The RSVP process also checks with the Admission Control module to find out whether the node has sufficient resources to supply the requested QoS.

If either check fails, the RSVP process sends an error notification back to the host. If both checks succeed, parameters are set in other modules (such as the Packet Classifier & Packet Scheduler modules) to enforce the reservation. The RSVP process then sends the reservation request to the next hop on the data path. As soon as every node on the data path accepts the reservation, the flow should receive the requested QoS.

The Packet Classifier and Packet Scheduler modules on every node are responsible for the QoS given to a flow for which a reservation has been made. The Packet Classifier looks at every data packet to determine whether the appropriate flow has a reservation and what QoS the flow should receive. Flows are identified in one of several formats. The simplest format contains the sender IP address and sender port

together with the destination address and port. In this case, the Packet Classifier has to examine the IP and UDP/TCP headers. In the future, IPv6 and/or MPLS flow labels could be used to specify RSVP flows, making the classification of packets more efficient.

The Packet Scheduler then makes the forwarding decision according to the QoS class. For example, the Packet Scheduler decides which queue the packet is to be placed in. The Packet Scheduler also polices the data flow to ensure that the reservations are not being violated. Packets which are outside the specified flow parameters could be marked as 'Out-of-Profile', could be treated as best effort, or could simply be discarded. The Packet Scheduler is a key component of the architecture because it actually gives different services to different flows. To ensure that flows receive their requested QoS, the Packet Schedulers on all nodes and routers must support the distinction between different services.

On the Internet, routes are subject to dynamic change, and receivers can disappear. RSVP dynamically copes with such changes by maintaining the reservations as a soft state. This means that the senders and receivers periodically send refresh PATH and RESV messages. Reservations automatically time out if such refresh messages are not received within a specified time period. Senders and receivers can also explicitly terminate reservations via standard RSVP messages.

2.4.4 Scalability Issues

Managing the reservation state for a large number of sessions is the primary scaling problem with RSVP. The number of RSVP control messages processed by each router is proportional to the number of QoS flows going through the router. RSVP deals with application-level flows, such as one multicast audio session or a video transmission

from a single source. Reservation state is kept on a per-flow basis. Hence, managing state and processing control messages scales linearly with the number of flows. However, managing reservation state puts a heavy strain on routers with large interfaces. Information about thousands of reservations needs to be stored, accessed and changed. The primary function of routers is packet forwarding. Managing a large amount of state information and performing additional lookups will necessarily degrade router performance. The management capabilities of RSVP routers must scale in proportion to their forwarding path bandwidth to fully utilize the capacities. Unicast routing tables store information per destination, aggregated by hierarchical routing. Multicast routing tables store information per multicast session and possibly per sender, depending on the multicast routing protocol. Aggregation of multicast routing state across groups is impossible with the current addressing scheme. In addition to unicast and multicast routing state, RSVP needs per-flow state, further straining the router's management capabilities. The amount of routing state in a router depends only on the network topology and is insensitive to the size of the router's links. In contrast, the state required for RSVP grows with the bandwidth of the links. The larger the links, the more flows can be served, but the more state information needs to be managed. Compared to unicast routing state, reservation state is relatively short-lived and thus frequently changed.

As links become even larger and support even more reservations, however, it is unlikely that the management capabilities can keep up. It is not expected that large routers on inter-domain backbones keep per-flow state. Some form of aggregation will be necessary. Aggregation means treating several RSVP flows as one. Of course, only flows with similar QoS requirements can be aggregated. Flows that are

aggregated into a super-flow share the delay. Isolation is not possible for aggregated flows.

2.4.4.1 Class-Based Aggregation

On entry into the aggregating region, each flow, for which a reservation was made, is assigned to one of the service classes. Flows with similar service requirements are grouped together into a service class. Each packet is marked with a tag that identifies which service the flow should receive. For IP, this tag could consist of the Type of Service (TOS) bits in the packet header or the packet could be encapsulated. Inside the aggregating regions, packets are scheduled according to their assigned service class. Because the number of classes is fixed, packet scheduling is far less expensive. However, there is a risk of congestion within any individual service class. Instead of simply combining all flows blindly into one service class, the overall bandwidth available for each service class can be specified. RSVP admission control is used to admit new flows if there is sufficient bandwidth within the service class. In this fashion, the advantages of Admission Control still apply, but the packets within each service class can be processed and routed far more efficiently.

2.5 Differentiated Services (DiffServ)

2.5.1 Overview

Differentiated Services (DiffServ) is an IETF specified QoS mechanism that handles traffic flows in one or more networks based on the needs of the traffic type.

DiffServ pushes the work to edges of network, while forwarding can be done very quickly in the core of the network. In the DiffServ framework, packets carry their own state in a few bits of the IP header (the DS Code Point), which also leads to scalability of this QoS mechanism, making it appropriate for end-to-end QoS. Policy decisions and implementations are left to local trust domains. The behavior of traffic classes is currently being defined in the IETF DiffServ working group .

DiffServ, proposed in [Nichols et al, 1998] and [Blake et al, 1998], defines a scalable service discrimination policy without maintaining the state of each flow and signaling at every hop.

The primary goal of differentiated services is to allow different classes of service to be provided for traffic streams on a common network infrastructure. Differentiated Service aggregates multitude of QoS-enabled flows into a small number of aggregates. These aggregated flows are given differentiated treatment within the network. The DiffServ approach attempts to push per-flow complexity away from the network core and towards the edge of the network where both the forwarding speeds and the fan-in of flows are smaller.

Each DiffServ flow is policed and marked according to the service profile at the edge of the network, and service only a small number of traffic aggregates in the core.

DiffServ will provide a controlled and coarsely predictable IP Class of Service (CoS). To support different classes of IP service over the Internet, the IP differentiated services architecture defines three main building blocks:

i. Packet classifiers,

ii. Forwarding/per-hop behavior,

iii. Traffic conditioning policies.

The differentiated services model utilizes static configuration of classification and forwarding policies in each node along a network path.

2.5.2 How does the DiffServ work?

Each packet receives a particular forwarding treatment based on the marking in its IP Type of Service (TOS) octet (now called as DS Code Point). The packet may be marked anywhere in the network, but probably mostly at domain boundaries. The packet is treated the same way as others with the same mark. There is no per-flow state required inside the network; core devices know only markings, not flows. Per-flow state is kept at the network edge, that is, flows are aggregated based on desired behavior. A note – this scheme requires
overall network engineering so that aggregates get the appropriate or desired services.

Services are built by applying rules: Rules for how packets are marked initially; rules for how marked packets are treated at boundaries. At boundaries of domains, the only requirement is to have bilateral agreement between the parties on each side of the boundary (i.e. no multilateral agreements required). Three elements work together to deliver a DiffServ service:

i. Per-Hop Behaviors (PHBs) deliver special treatment to packets at forwarding time,

ii. Traffic Conditioners alter packet aggregates to enforce rules of service,

iii. Bandwidth Brokers (also known as Policy Managers) apply and communicate policy.

2.5.3 DiffServ Code Point (DSCP)

DSCP, earlier called as TOS field, is an 8-bit field in IP packet's header. Of the 8 bits in the field, 6 bits define the per-hop behavior (PHB) the packet will receive with respect to policies established at a network boundary, and the rest of the 2 bits are currently unused. figure 2.4 illustrates the location of DSCP in IPv4 header and figure 2.54 the contents of the DSCP.

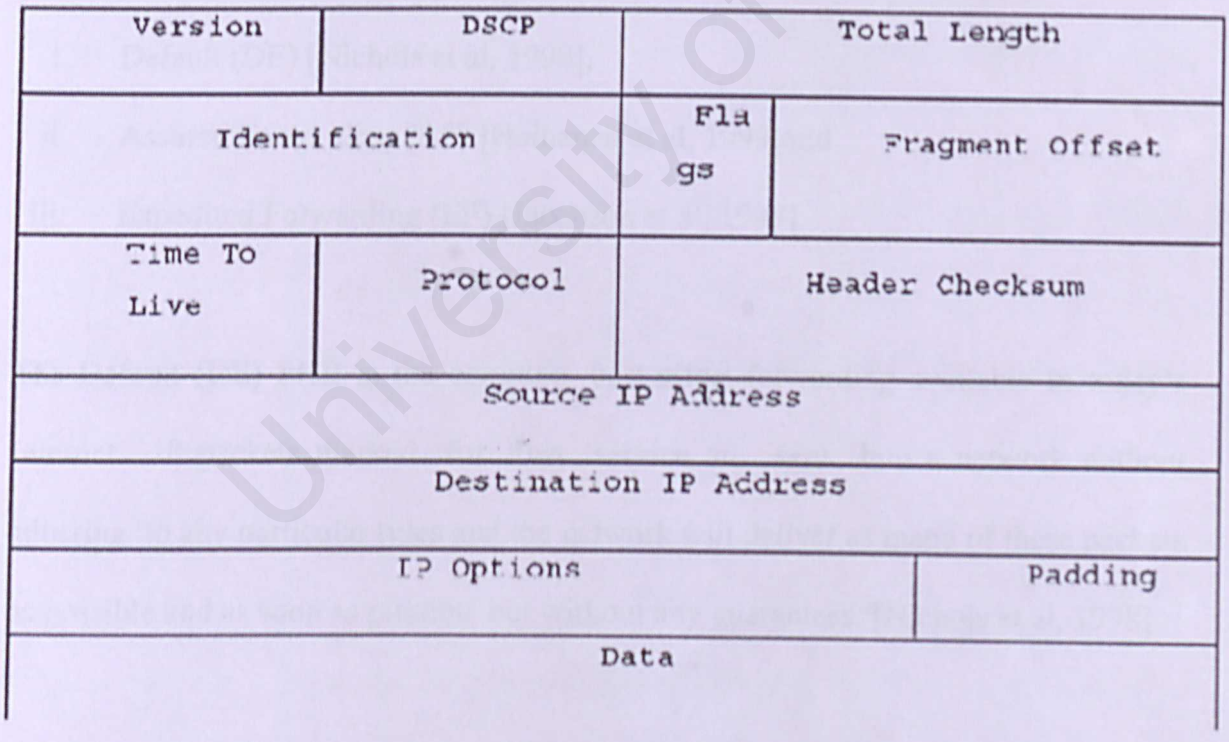


Figure 2.4 IPv4 Datagram with DiffServ Code Point.

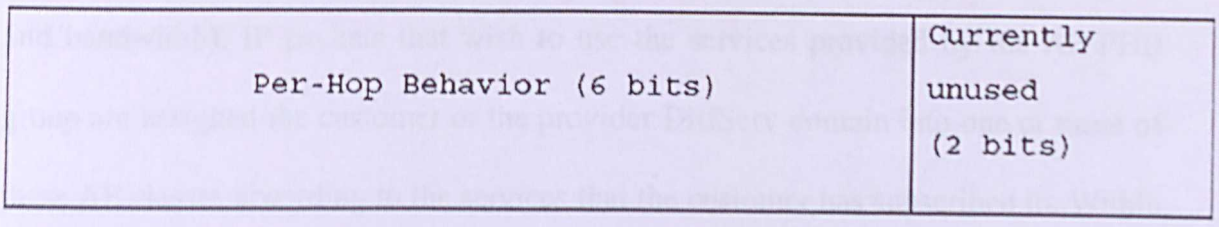


Figure 2.5 DiffServ Code Point.

2.5.4 Per-Hop Behavior (PHB)

PHBs are the packet forwarding treatments that delivers the "differentiated service" to packets at the network node output: shaping, policing and possible remarking of DS Code Point, en-queuing treatment (e.g. drop preference) and scheduling.

Three PHBs currently have been defined:

i. Default (DE) [Nichols et al, 1998],

ii. Assured Forwarding (AF) [Heinanen et al, 1999]and

iii. Expedited Forwarding (EF) [Jacobson et al, 1999].

The Default (DE) PHB is the common, best-effort forwarding available in today's Internet. IP packets marked for this service are sent into a network without adhering to any particular rules and the network will deliver as many of these packets as possible and as soon as possible but without any guarantees. [Nichols et al, 1998]

Assured Forwarding (AF) PHB group is a means for a provider DiffServ domain to offer different levels of forwarding assurances for IP packets received from a customer DiffServ domain. Four AF classes have been defined, where each AF class is in each DS node allocated a certain amount of forwarding resources (buffer space

and bandwidth). IP packets that wish to use the services provided by the AF PHB group are assigned the customer or the provider DiffServ domain into one or more of these AF classes according to the services that the customer has subscribed to. Within each AF class, IP packets are marked with one of the three possible drop precedence values. In case of congestion, the drop precedence of a packet determines the relative importance of the packet within the AF class. A congested DiffServ node tries to protect packets with a lower drop precedence value from being lost by preferably discarding packets with a higher drop precedence value. [Heinanen et al, 1999]

The Expedited Forwarding (EF) PHB is defined as a forwarding treatment for a particular DiffServ's aggregate, where the departure rate of the aggregate's packets from any DiffServ node must equal or exceed a configurable rate. Expedited Forwarding can be used to build a low latency, low loss, low jitter, assured bandwidth, end-to-end service through DS domains.

Such a service appears to the endpoints like a point-to-point connection or a 'virtual leased line'. Latency, loss and jitter are all due to the queues traffic experiences while transiting the network. Therefore providing low latency, loss and jitter for some traffic aggregate means ensuring that the aggregate sees no (or very small) queues. [Jacobson et al, 1999]

The classifier determines the value of the DS field for each incoming flow at the edge of the network or administrative boundary. These incoming flows are aggregated over an outgoing flow based on the DS field. The router implementations in the intermediate nodes use this 6-bit PHB field to index into a table for selecting a particular packet-handling mechanism.

This forwarding policy determines how routers will handle the packets in terms of providing a class of service by combining traffic management functions, such as packet queuing, scheduling, and buffer reservations at each node.

DiffServ expects advance provisioning and reservations made in each of the intermediate nodes along the network path. If a network path crosses multiple DS domains or multiple ISPs, the ISPs must support the same PHBs to provide a consistent end-to-end service. The work is underway to support per-domain behaviors (PDBs). A PDB specifies a forwarding path treatment for traffic aggregate and, due to the role that particular choices of edge and PHB configuration play in its resulting attributes; it is where the forwarding path and the control plane interact. The measurable parameters of a PDB should be suitable for use in Service Level Specifications (SLSs) at the network edge.

2.5.5 Traffic Classification

Packets selects by the classifier based on the combination of one or more predefined set of header fields. The mapping of network traffic to the specific behaviors that result in different class of service is indicated by the DSCP which uniquely identifies the per-hop behavior or the treatment given to the traffic at each hop along the network path. The DiffServ architecture supports a maximum of 64 classes of service and each router sorts the packets into queues based on the DSCP. The queues might get different treatment based on their priority, share of bandwidth, and discard policies.

2.5.6 Traffic Conditioning

The DiffServ architecture offers a framework within which service providers can offer each customer a range of network services that are differentiated on the basis of performance in addition to pricing of tiers used in the past. These services are monitored for fairness and in meeting the service agreements. In order to deliver service agreements, each DiffServ enabled edge router implements Traffic Conditioning function which performs metering, shaping, policing and marking of packets to ensure that the traffic entering a DiffServ network conforms to the Traffic Conditioning Agreement (TCA). The functions of Traffic Conditioning are following:

□

Metering – Monitors the traffic pattern of each flow against the traffic profile. For out-of-profile traffic the metering function interacts with other components to either remark, or drops the traffic for that flow.

Shaping – The routers control the forwarding rate of packets so that flow does not exceed the traffic rate specified by its profile. The shapers ensure fairness between flows that map to the same class of service, and control the traffic flow to avoid congestion.

Policing – At the ingress edge routers, the incoming traffic is classified into aggregates. These aggregates are policed according to the traffic conditioning agreement.

The out-of-profile traffic is either dropped at the edge or is remarked with a different PHB.

Marking – Customers request a specific performance level on a packet by packet basis, by marking the DS field of each packet with a specific value. This

value specifies the PHB to be allocated to the packet within the provider's network. The edge routers classify the packets to identify the PHB and a DSCP for that packet.

The location of Traffic Conditioning components is shown in Figure 2.6.

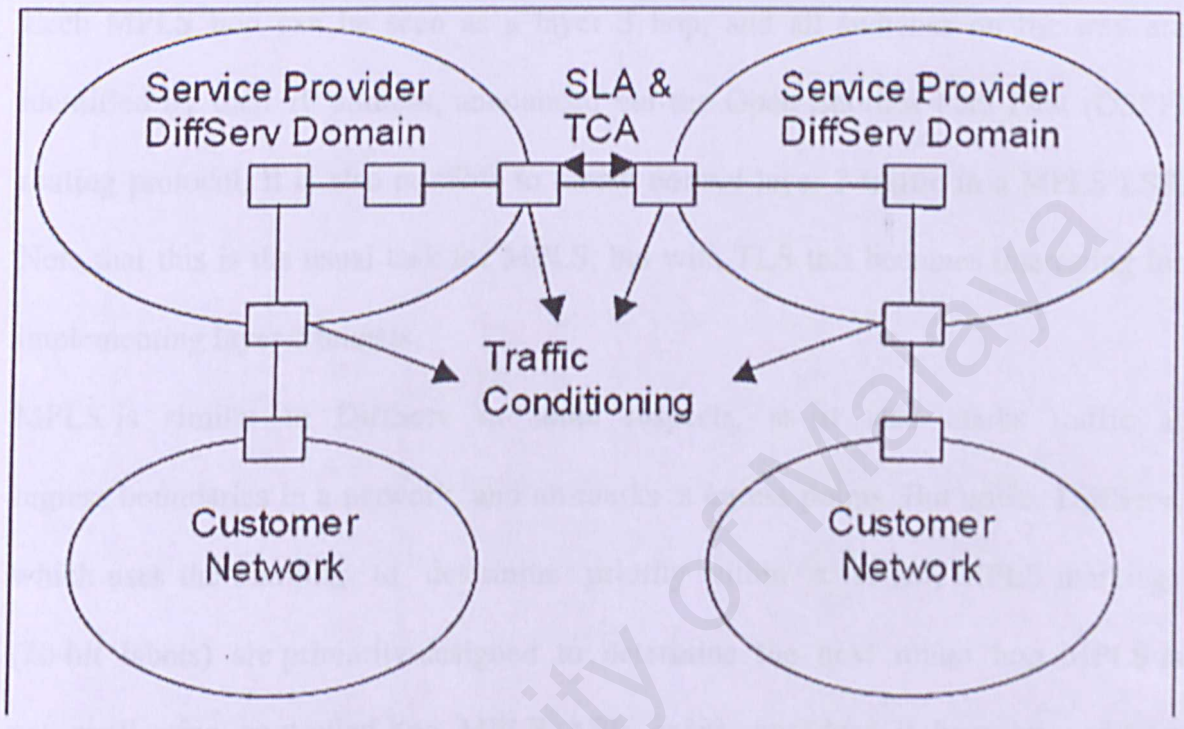


Figure 2.6 Traffic Conditioning Locations in DiffServ Architecture

2.6 Multi-protocol Label Switch (MPLS)

2.6.1 Overview

Multi-protocol Label Switching, MPLS [Rosen et al, 2001], as the name suggests is a switching method that forwards IP traffic using a label. MPLS can be seen as a shim layer between layer 2 and layer 3 in the OSI model. It is similar to ATM virtual

circuits and Frame Relay, because only a label is used in the routing through a core network. Some people call it layer 2.5. Despite the comparison with older and more expensive solutions, MPLS seems to be a good compromise between multiple existing standards. It provides a lot of good features while retaining flexibility and ease of use.

The idea is to use Label Switching in the core network, with MPLS enabled switches. Each MPLS hop can be seen as a layer 3 hop, and all switches on the way are identified by their IP address, announced via the Open Shortest Path First (OSPF) routing protocol. It is also possible to tunnel normal layer 3 traffic in a MPLS LSP. Note that this is the usual task for MPLS, but with TLS this becomes interesting for implementing layer 2 tunnels.

MPLS is similar to DiffServ in some respects, as it also marks traffic at ingress boundaries in a network, and un-marks at egress points. But unlike DiffServ, which uses the marking to determine priority within a router, MPLS markings (20-bit labels) are primarily designed to determine the next router hop. MPLS is not application controlled (no MPLS APIs exist), nor does it have an end-host protocol component, it resides only on routers. And MPLS is protocol-independent (i.e. multi-protocol), so it can be used with network protocols other than IP (like IPX, ATM, PPP or Frame-Relay) or directly over data-link layer as well.

2.6.2 Advantages

Today, MPLS has become a well-known standard. It is mature and it provides everything that is necessary for a fast switching backbone to provide VPN capabilities. If two adjacent routers are both BGP peers and MPLS label switching routers, all the label switching information is easily transferred via BGP-4. The internal version of BGP, iBGP, is used for MPLS. Extreme Networks are using dedicated MPLS add-on

cards to provide secure transit to a MPLS enabled core. Other layer 2 and layer 3 functionalities are intact and unaffected by the addition of MPLS.

2.6.3 Disadvantages

A transition to MPLS can be very expensive. The access switches/routers need extra hardware and a special version of the software or MPLS capable devices must replace them. Unfortunately this software often lags behind the normal software releases, and therefore a MPLS-enabled node cannot frequently make use of all the latest features that other nodes may use.

Today the hardware from Extreme Networks consists of expensive add-on cards specially made for MPLS, without the fast implementation in ASICs that's possible once the standards are complete. Since MPLS as a working standard is still evolving, the software for it is being upgraded all the time.

Another disadvantage of MPLS is the long time required to rebuild all the LSPs when a link goes down. In a complex network a lot of CPU power is required to calculate new LSPs. These tables also take considerable memory, especially when recalculating all paths. It seems that this will not have much impact in a modern core switch like the Extreme Networks' Black Diamond, because of the huge amounts of memory and processing power available both in the switch and in the MPLS add-on card.

2.6.4 Components and Mechanisms

2.6.4.1 Label Switch Router (LSR)

LSR is a device that is capable of forwarding packets at layer 3 and forwarding frames that encapsulate the packet at layer 2. The label swapping mechanism is implemented at layer 2. LSR use Label Distribution Protocol, LDP to determine where and in what manner packets are forwarded.

2.6.4.2 Label Edge Router (LER)

LER is both a switch and router that is capable of forwarding MPLS frames to and from an MPLS domain. It performs the IP to MPLS FEC binding including the aggregation of incoming flows. It also communicates with interior MPLS LSRs to exchange label bindings. LER often referred to as an ingress or egress LSR, this is because it is situated at the edge of a MPLS domain.

2.6.4.3 Label Switch Path (LSP)

LSP is an ingress-to-egress switched path built by MPLS nodes to forward the MPLS encapsulated packets of a particular FEC using the label swapping forwarding mechanism. It is similar to the concept of Virtual Channels within an ATM context.

2.6.4.4 Forwarding Equivalency Class (FEC)

FEC is a group of packets that will be forwarded through the MPLS network along the same LSP and with the same forwarding treatment. The ingress edge MPLS LSR usually assigns the FEC for particular packets, according to the packets destination

(IP Packet header) and incoming interface value. These FEC's are then assigned to a particular LSP, before being forwarded through the MPLS network.

2.6.4.5 Label

It is a short, fixed length, locally significant identifier that is used to identify a FEC. The packet may be assigned to a FEC based on its network layer destination address; However, the label does not directly encode any information from the network layer header. A packet which a label has been encoded is called a labeled packet. In an ATM network, the label is placed in the VPI/VCI fields of each ATM cell header, while in a LAN environment, the header is a "shim" located between the Layer 2 and Layer 3 headers as shown in figure 2.7.

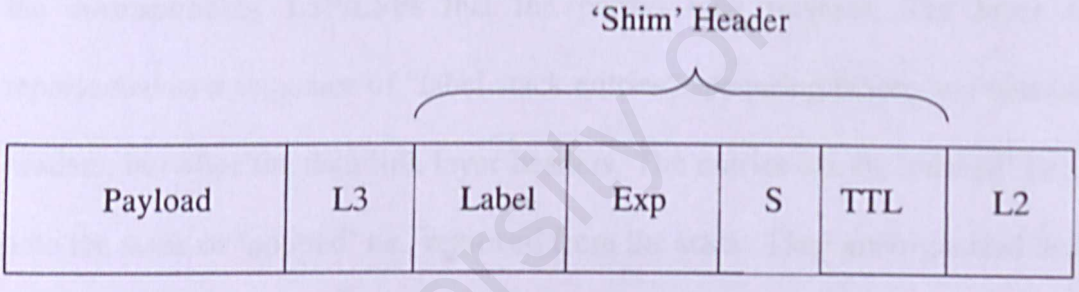


Table 2.7 MPLS "Shim" header

The 32-bit MPLS header contains the following fields:

I. Label Value (20 bits) - This field carries the actual value of the label.

II. Experimental bits (Exp) (3 bits) - Set for experimental use, not yet defined.

III. Bottom of Stack (S) (1 bit) - This bit is set to 1 if the current label is the only one present or is the last label in the stack and set to 0 for all other label stack entries.

IV. Time to Live (TTL) (8 bits) - *-Time-*To-Live, an inherent part of IP functionality.

2.6.4.6 Label Stack

Label Stack is an ordered set of labels appended to a packet that enables it to explicitly carry information about more than one FEC that the packet belongs to and the corresponding LSP/LSPs that the packet may traverse. The label stack is represented as a sequence of "label stack entries" appearing before any network layer headers, but after the data link layer headers. The entries can be 'pushed' i.e., placed onto the stack or 'popped' i.e., removed from the stack. They are organized in a last in first out manner.

2.6.5 Label Encapsulations

MPLS is intended to run over multiple data link layers for example in ATM the label is contained in the VPI/VCI field of the ATM header where in Frame Relay the label is contained in the DLCI field in the FR header. In PPP/LAN the 'shim' header is inserted between the layer two and three headers.

2.6.6 How MPLS works

Multi Protocol Label Switching tries to make use of a fast switching technique based on a short 32 bit header, placed between Data Link and Network headers. A Label Switched Path (LSP), see Figure 2.8, can switch the traffic on the way in between two LERs. The first LER adds a label depending on which VLAN the packet belongs to and another label based on an IP longest prefix match scheme, i.e. a normal routing decision, to the next MPLS-hop. It then sends the packet to next LSR or LER, depending on its own MPLS forwarding table. The LSRs in between only look at the outermost 32 bit long label in the label stack and forwards the packet to the next hop, according to its own forwarding table, after the label is changed to its label for the next hop. The last LER strips off the label and forwards the packet to the right interface/VLAN according to the setup made for this specific LSP.

A typical MPLS network is shown in Figure 2.8. Here there are several paths through the network to a LER, but the routers have build up their forwarding databases via the Label Distribution Protocol (LDP), and created a Label Switching Path. Each LSP can contain both layer 2 traffic and IP-traffic.

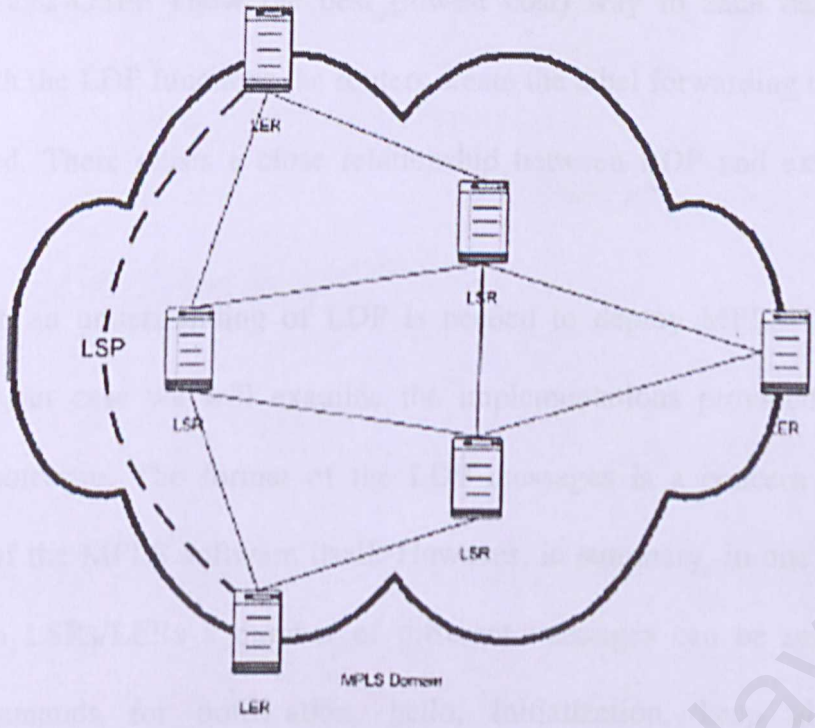


Figure 2.8 Typical MPLS network

2.6.7 Label Distribution Protocol (LDP)

There are two different ways that labels are transported and assigned by all LERs and LSRs: either (1) downstream or (2) downstream on demand. The simplest way is to enable OSPF in the backbone and with the help from LDP just let it distribute possible LSPs to all neighbors. Each LER/LSR opens a TCP-session on port 646 to its neighbors and exchanges LDP messages, and keeps them alive a specified time. When OSPF announces a route change, all LERs and LSRs rearrange their local forwarding tables accordingly. Each LER then knows on which LSP to send traffic, for a certain next-hop router.

Another way is to enable BGP-4, and let BGP take care of all route distribution. The MPLS labels can then be piggybacked on the BGP UPDATE message. If two adjacent MPLS routers are using BGP, then they can use that for the label distribution as well.

Both BGP and OSPF know the best (lowest cost) way to each destination, and together with the LDP functions the routers create the label forwarding tables, and the LSPs needed. There exists a close relationship between LDP and existing routing protocols.

Even though an understanding of LDP is needed to deploy MPLS in an existing network, in our case we will examine the implementations provided by Extreme Networks' software. The format of the LDP messages is a concern only for the developers of the MPLS software itself. However, in summary, in one LDP session between two LSRs/LERs a number of different messages can be sent. They can contain commands for notification, hello, initialization, keep alive, address assignment, address withdrawal, label mapping, label request, label abort request, label withdrawal, and label release.

2.6.8 Resource Reservation Protocol with Traffic engineering extensions (RSVP-TE)

Resource Reservation Protocol with Traffic Engineering extensions (RSVP-TE) is another way to set up all LSPs. Several different objects are proposed to extend RSVP to make use of intelligent signaling, so that LSPs are automatically routed away from traffic congestions, network failures, and similar. Basically it is a protocol that sends out Traffic Specification (Tspec) objects in the same data path as the actual data, with requests for different QOS parameters. The signaling done makes sure that all routers provide requested services to all nodes along a path. It is not another routing protocol, but it makes use of existing routing table.

2.7 Internet Protocol version 6 (IPv6)

IP version 6 (IPv6), is an evolutionary enhancement of IPv4. IPv6 is designed to redress IPv4's shortfalls, retain IPv4's strong points, and accommodate the expected future growth and diversity of the global Internet.

2.7.1 IPv6 Motivation

IPv6 has been designed to enable high-performance, scalable inter-networks to remain viable well into the next century. A large part of this design process involved correcting the inadequacies of IPv4. The enhanced features of IPv6 are larger address space, streamlined packet design, well-structured and efficient routing hierarchy, ease of administration, better support of security and QoS (Quality of Service). The motivations for the development of a new Internet Protocol are listed below:

2.7.1.1 Address Space Depletion

The world is running out of IP addresses for networked devices results of the rapid growth of the Internet. Communications technologies need permanent connection to the Internet [Microsoft, 2000].

Because of insufficient address space, some organizations are forced to use temporary technologies such as Network Address Translator (NAT) to map multiple private addresses to a single public IP address. However, problems arise when connecting two organizations that use the private address space because they do not support standards-based network layer security or the correct mapping of all higher layer protocols.

2.7.1.2 Hierarchical Addressing System

Without an address hierarchy system, backbone routers would be forced to store routing table information on the reach-ability of every network in the world [BAY, 1997]. With an address hierarchy system, backbone routers can use IP address prefixes to determine how traffic should be routed through the backbone.

Currently, IPv4 uses Classless Inter-Domain Routing (CIDR) [Fuller, 1993] to allow flexible use of variable-length network prefixes. CIDR permits considerable "route aggregation" at various levels of the Internet hierarchy so that backbone routers can store a single routing table entry that provides reach-ability to many lower-level networks. However, CIDR does not guarantee an efficient and scalable hierarchy. Legacy IPv4 address assignment originated before CIDR does not facilitate summarization. The lack of uniformity of the current hierarchical system coupled with the rationing of IPv4 addresses complicate the situation.

The hierarchical approach to IPv6 is believed to make automatic router configuration a much more viable proposition than it is at the moment.

2.7.1.3 System Management

The current IPv4 implementation must be either configured manually or use a stateful address configuration protocol such as Dynamic Host Configuration Protocol (DHCP) [Drom, 1997]. With more computers and devices use IP, there's a need for a simpler and more automatic configuration of addresses and other configuration settings to reduce address administration workloads.

2.7.1.4 Security

Encryption, authentication, and data integrity safeguards are increasingly a standard aspect of enterprise internetworking. Vendors in the IPv4 arena are not very successful in adding robust security features to Network Layer components largely due to the lack of interoperability caused by proprietary security extensions. In IPv4, Internet Protocol security (IPsec) is not mandatory.

The proponents of IPv6 claim that to achieve the same level of security with IPv4 as is available with IPv6 would need more work, and would thus cost more money, than upgrading to the improved protocol.

2.7.1.5 Quality of Service (QoS)

Real-time traffic support relies on the IPv4 Type of Service (TOS) field and the identification of the payload, typically using a User Datagram Protocol (UDP) or Transmission Control Protocol (TCP) port to deliver Quality of Service. QoS is getting more important as one of the significant shifts in the future Internet traffic is a huge growth in the use of the Internet as a broadcast medium carrying video and audio.

However, the IPv4 TOS field has limited functionality and there were various local interpretations. Furthermore, payload identification using a TCP and UDP port is not possible when the IPv4 packet payload is encrypted.

2.7.2 IPv6 Features

The important features of the IPv6 protocol are as follows:

2.7.2.1 New Header Format

The new IPv6 header format is designed to keep header overhead to a minimum. Both non-essential fields and option fields in IPv4 are moved to extension headers that are placed after the IPv6 header [Deering&Hinden, 1998a].

Most of these optional headers are not examined or processed by intermediate nodes on the packet's path. In this way, IPv6 exhibits more efficient forwarding than IPv4. It's also easier now to add additional options. Processing of the IPv6 packet header is simpler as the IPv6 packet header is fixed-length whereas the IPv4 header is variable-length. Furthermore, only the source but not the IPv6 routers can perform packet fragmentation that is always time consuming.

IPv4 headers and IPv6 headers are not interoperable. A host or router must implement both IPv4 and IPv6 in order to recognize and process both headers formats. The new IPv6 header is only twice as large as the IPv4 header although IPv6 addresses are four times larger than IPv4 addresses.

2.7.2.2 Large Address Space

IPv6 has 128-bit (16-byte) source and destination IP addresses that can express over 3.4 * 1038 possible combinations [Deering&Hinden, 1998b]. The large address space of IPv6 allows multiple levels of sub-netting and address allocation from the Internet backbone to the individual subnets within an organization. Address-conservation techniques such as NAT are no longer necessary.

2.7.2.3 Efficient and hierarchical addressing and routing infrastructure

IPv6 implements address hierarchy where backbone routers use IP address prefixes to determine how traffic should be routed through the backbone, thus improve the routing efficiency. The IPv6 routing is simplify as the IPv6 packet header is fixed-length whereas the IPv4 header is variable-length and packet fragmentation is not permitted by IPv6 routers as it's only be performed by the source [Deering&Hinden, 1998a].

2.7.2.4 Stateless and stateful address configuration

IPv6 supports both stateful address configurations, such as using Dynamic Host Configuration Protocol (DHCP) server and stateless configuration (in the absence of DHCP server). With stateless configuration [Thompson&Narten, 1998], hosts on link automatically configure themselves with IPv6 addresses for the link (called link-local addresses) and with addresses derived from prefixes advertised by local routers. This feature will be able to reduce the address administration workload significantly.

2.7.2.5 Built-in security

IPSec is mandatory in IPv6, whereas it's optional in IPv4. IPv6 provides native data security capabilities based on its flexible header extensions. The native authentication of IPv6 gives the industry a standard-based method to determine the authentication of packets received at the Network Layer.

2.7.2.6 Better support for QoS

The IPv6 packet format contains a new 24-bit traffic-flow identification field called Flow Label, typically designed for QoS in IPv6. IPv6 Flow Label can be used to identify to the network a stream of packets that needs special handling above and beyond the default, best-effort forwarding. Because the traffic is identified in the IPv6 header, support for QoS can be achieved even when the packet payload is encrypted through IPSec.

2.7.2.7 New protocol for neighboring node interaction

The Neighbor Discovery (ND) [Narten et al, 1998], protocol for IPv6 is a series of Internet Control Message Protocol version 6 (ICMPv6) messages that manages the interaction of neighboring nodes (nodes on the same link). Neighbor Discovery replaces the broadcast-based Address Resolution Protocol (ARP), Internet Control Message Protocol version 4 (ICMPv4) Router Discovery, and ICMPv4 Redirect messages with efficient multicast and unicast Neighbor Discovery messages.

2.7.2.8 Extensibility

IPv6 can easily be extended for new features by adding extension headers after the IPv6 header. The size of IPv6 extension headers is only limited by the size of the IPv6 packet, compare to IPv4 which can only support 40 bytes of options in IPv4 header [Deering&Hinden, 1998a].

2.7.2.9 Multicast and Anycast

IPv6 extends IP multicasting capabilities in IPv4 by defining a very large multicast address space and a scope identifier that is used to limit the degree to which multicast routing information is propagated throughout an enterprise. Multicasting is an important feature in IPv6 as it eventually replace IPv4 broadcast feature.

Conceptually anycast is a cross between unicast and multicast. Two or more interfaces on an arbitrary number of nodes are designated as an anycast group. A packet addressed to the group's anycast address is delivered to at least one of the interfaces in the group, typically "nearest" interface in the group, according to the routing protocols' measure of distance.

2.7.3 Differences between IPv4 and IPv6

The following table demonstrates the few main differences between IPv4 and IPv6.

Table 2.4 Differences between IPv4 and IPv6

IPv4	IPv6
Source and destination addresses are 32 bits (4 bytes) in length.	Source and destination addresses are 128 bits (16 bytes) in length.
Fragmentation is supported at both routers and the sending host.	Fragmentation is not supported at routers. It's only supported at the sending host.
Header includes a checksum.	Header does not include a checksum.
Header includes options.	All optional data is moved to IPv6 extension headers.

IPSec support is optional.	IPSec support is mandatory.
No identification of payload for QoS handling by routers is present with the IPv4 header.	Payload identification for QoS handling by routers is included in the Flow Label inside the IPv6 header.
Address Resolution Protocol (ARP) broadcast ARP Request frames to resolve an IPv4 address to a link layer address.	ARP Request frames are replaced with multicast Neighbor Solicitation messages.
Uses Internet Group Management Protocol (IGMP) to manage local subnet group membership.	Replace IGMP with Multicast Listener Discovery (MLD).
Must be configured either manually or through DHCP.	Do not need manual configuration or DHCP.
Uses host address (A) resource records in the Domain Name System (DNS) to map host name to IPv4 addresses.	Uses host address (AAAA) resource records in DNS to map host names to IPv6 addresses.
Uses pointer (PTR) resource records in the IN-ADDR.ARPA DNS domain to map IPv4 addresses to host names.	Uses PTR resource records in the IP6.INT DNS domain to map IPv6 addresses to host names.

2.8 Computer Simulation

Simulation is the process of designing a model of a real or imagined system and conducting experiments with that model [Smith, 2000]. Modern simulations are usually carried out using digital computers, hence the term computer simulation or

digital simulation. The primary motivation for simulation is to allow the experiments and analyses of a system without the need to construct the actual system. To build an actual system for every experiment is usually too expensive and impractical, and sometimes simply impossible. In the case of communication networks, simulation is an effective approach in the design, evaluation and experimentation of new networking protocols, theories or algorithms.

2.8.1 Simulation Model

Simulation is the process of designing a model of a real or imagined system and conducting experiments with that model [Smith, 2000]. A model is an abstraction of a system intended to replicate some properties of that system [Overstreet, 1982]. The collection of properties the model is intended to replicate must include the modeling objective of the simulation. In other words, since a model is an abstraction, it cannot fully represent the system in question, but it must capture the system characteristics that are relevant to the simulation objective.

The simulation model can be a collection of objects that interact with each other, where each object is associated with a set of attributes. The simulation itself then involves the execution of the model (let the objects interact), followed by the analysis of the simulation outcome.

2.8.2 Simulation Approach

According to [Nance, 1993], computer simulation may be divided into three categories based on the simulation approach:

1. Monte Carlo simulation, a method by which an inherently non-probabilistic problem is solved by a stochastic process, where explicit representation of time is not required.
2. Continuous, in which the variables within the simulation are continuous functions (normally involve solving differential equations)
3. Discrete event, where the change of the values of program variables happens at finite number of time points in simulation time (not necessarily evenly spaced)

Usually, the actual simulation involves the use of a combination of techniques. For example, a "combined" simulation refers generally to a simulation that has both discrete event and continuous components, whereas a "hybrid" simulation refers to the use of an analytical sub-model within a discrete event framework. For network simulations, especially packet-level simulations, the discrete event approach is the most significant. The JaNetSim, as well as all network simulators discussed in this chapter, are based on the discrete event approach.

It is often possible to use a simulation model in conjunction with a less realistic but "cheaper-to-use" analytical model [Bratley, 1987]. An analytical model describes the system in question with mathematical formulas obtained through analyses (e.g. probability theory, queuing theory, etc.). Once the formulas are derived, the evaluation of the system can be quickly done. Analytical model often involves too many simplifying assumptions and may not represent the actual system correctly. On the other hand, the simulation model more closely resembles the actual system and is generally more accurate, but with a high computation cost. One approach is to first

evaluate a system using an analytical model, then use simulation to validate the analytical model.

2.8.3 Existing Network Simulators

There is a wide variety of network simulation tools available, with a wide variety of focuses. A network simulator can be either a general-purpose simulator that enables a wide range of possible simulations, or a special-purpose simulator targeting a particular area of research. In terms of simulation approach, most network simulators are based on the discrete event technique, and are sometimes augmented with some analytic models for better performance or accuracy [Breslau et al, 2000]. This section reviews a number of major network simulators, describing their features, advantages, and weaknesses.

2.8.3.1 OPNET

The OPNET Modeler is originally developed at MIT, and introduced in 1987 as the first commercial network simulator. It uses the hybrid simulation approach where the discrete event approach is assisted by the analytical model. It is object-oriented based on the C/C++ programming languages. The simulator has full GUI support and consists of three hierarchically related editors: the network editor, the node editor, and the process editor.

Advantages

The primary advantage of the OPNET Modeler is the inclusion of a comprehensive library of detailed networking protocol and application models. The library also

includes models of both generic and vendor specific network devices. This library provides a wide range of options in performing network simulations.

Disadvantages

On the other hand, the OPNET Modeler is not fully platform independent since it supports only the Solaris, Windows NT/2000, and the HP-UX operating systems. As a commercial product, the use of the OPNET Modeler is costly from a financial point of view.

2.8.3.2 INSANE

The INSANE (Internet Simulated ATM Networking Environment) network simulator is an object-oriented, discrete-event simulator designed to simulate a medium-to-large IP-over-ATM internetworking environment. It simulates the operation of a set of typical Internet applications (ftp, telnet, WWW browser, audio, video) using empirical traffic models. The TCP, UDP and IP stack is simulated. In terms of the data link layers, both ATM and shared-media LAN are available.

Advantages

The Tk-based graphical simulation monitor enable user to check the progress of multiple running simulation process. Besides that, it is able to support the simulation on a large network, which the result is processed off-line.

Disadvantages

The GUI of INSANE facilitates monitoring of multiple simulation runs but does not provide other features concerning the creation of the simulation environment. Another disadvantage of INSANE is that it only runs on Unix-based platforms.

2.8.3.3 MAISIE and PARSEC

PARSEC (Parallel Simulation Environment for Complex systems) is a C-based discrete-event simulation language [UCLA, 1999]. It is derived from Maisie [UCLA, 1995] with several improvements, both in the syntax and the simulation execution environment.

Advantages

PARSEC able to execute a discrete-event model using several different asynchronous parallel simulation protocols on a variety of parallel architectures.

Disadvantages

Since the entire simulation process involves the use of the language itself without a GUI, it might be less efficient in creating various simulation environments. Besides, the use of a new simulation language may result in less portability among different platforms.

2.8.3.4 REAL and Ns

Ns [VINT 2003] is a discrete event network simulator, derived from the REAL network simulator, and now supported by DARPA through the VINT project [Breslau et al, 2000]. It provides substantial support for simulation of TCP, unicast and

multicast routing over both wired and wireless networks. The simulator is written in C++ (for its core), and simulation scenarios are designed using the Tcl scripting language (or OTcl for Ns version 2).

Advantages

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 [Breslau et al, 2000], may prove useful for diagnostics of protocol implementation errors. Another advantages of is Ns allow simulation with multiple levels of abstraction, where higher abstraction levels (with the use of analytical models) trade off accuracy for performance.

Disadvantages

Ns does not have a GUI for general simulation manipulation and scenario setup. However, it does provide a network animation tool that provides network visualization features.

2.8.3.5 DRCL JavaSim

JavaSim is a component-based, compositional simulation environment developed at the Distributed Real-time Computing Laboratory of the Ohio State University. It is built upon the notion of the autonomous component programming model [DRCL, 2001]. Similar to the Ns, JavaSim uses a dual-language approach, where the core language is the Java programming language, while the Tcl is used for simulation setups.

Advantages

JavaSim supports the simulation of various network architectures based on a generalized packet switched network model, including the DiffServ architecture, the mobile wireless network and the WDM-based optical network architecture.

Disadvantages

The requirement for a scripting language and the lack of a GUI are some of its disadvantages.

2.8.3.6 NIST ATM/HFC Network Simulator

The NIST Asynchronous Transfer Mode (ATM) / Hybrid Fiber Coax (HFC) Network Simulator [Golmie et al, 1998] is a simulator that provides a flexible test bed for studying and evaluating the performance of ATM and HFC networks. It is based on the discrete event approach and uses the C programming language.

Advantages

The simulator has a GUI and uses the X Window System running on Unix-based platforms. The simulator has a well-defined message passing mechanism based on the sending of events among simulation components, handled by an event manager.

Disadvantages

The use of the procedural approach makes the component development process difficult. Since the simulator relies on the X Window System for its GUI and UNIX in general, it lacks portability between different platforms.

2.8.3.7 UMJaNetSim

UMJaNetSim simulator is a flexible test bed for studying and evaluating the performance of MPLS network without the expense of building a real network. It is written in JAVA Language whereby it is developed in object-Oriented programming approach. Typically, the simulator is a tool that gives the 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 save and load the network configuration

Advantages

UMJaNetSim has a very good graphical user interface that provides a very user-friendly environment and viewable output performance text based and graphical representation on the screen while the simulation is running. It also has high portability among various platforms since it is written in Java. Furthermore, user can add in new components without affecting the whole simulator because it is written in object-oriented programming approach.

Disadvantages

UMJaNetSim is still consider a new network simulator, thus some of it features is still not mature enough.

2.8.4 Summary of Existing Simulators

To summarize, the simulators that have been discussed all have their advantages and weaknesses in terms of simulation techniques, the programming approach, availability of a graphical user interface, platform dependence or independence and focus of network research areas.

All simulators that have been studied are discrete event simulators. However, not even one simulator is web-enabled. Table 3.1 gives a comparison among the studied simulators in terms of object-oriented, graphical user interface (GUI), multithread and platform independence

Table 2.5 *Comparisons of Simulators*

Simulator	ObjectOriented	GUI	Multithreaded	Platform Independence
INSANE	Yes	Poor	Yes	No
OMNET++	Yes	Good	No	No
OPNET	Yes	Normal	No	No
PARSEC	No	Poor	Yes	No
REAL NS	No	Poor	No	No
NIST ATM/HFC	No	Normal	No	No
NS-2	Yes	Normal	Yes	No
UMJaNetSim	Yes	Good	Yes	Yes

2.9 Summary

This chapter has covered the primary research background of this project and relevant knowledge needed to develop the network simulator. It also includes the researches and comparison of the existing network simulators. A more detailed explanation of IPv6 Flow Label will be covered in the following chapter.

Chapter 3 IPv6 Flow Label

As stated by [Deering&Hinden, 1998a], at the time when the IPv6 specifications were written, the IPv6 flow label was still experimental, and subject to change, as the requirements for flow support in the Internet were evolving.

In this chapter, I will propose the specification of IPv6 flow label to provide a better Quality of Service of network.

The IPv6 flow label is a function that, as it was designed, can be used towards a more efficient processing of packets in next hop lookup, quality of service, or packet filtering engines in IPv6 forwarding devices. These devices would normally be IPv6 routers or switches.

An IPv6 flow label classifier is basically a 3 element tuple - source and destination IPv6 addresses and the IPv6 flow label. It is an alternative to the 5 element tuple (addresses, ports, and protocol). It will help the IPv6 flow label to achieve, as it is supposed, a more efficient processing of packets in quality of service engines in IPv6 forwarding devices.

3.1 IPv6 Flows

A flow is a sequence of packets sent from a particular source, and a particular application running on the source host, using a particular host-to-host protocol for the transmission of data over the Internet, to a particular (unicast or multicast)

destination, and particular application running on the destination host, or hosts, with a certain set of traffic, and quality of service requirements.

3.2 IPv6 Flow Label

The IPv6 Flow Label is defined [Deering&Hinden, 1998a] as a 20 bit field in the IPv6 header which may be used by a source to label sequences of packets for which it requests special handling by the IPv6 routers, such as non-default quality of service or "real-time" service. According to [Deering&Hinden, 1998a], the nature of that special handling might be conveyed to the routers by a control protocol, such as a resource reservation protocol, or by information within the flow's packets themselves, e.g., in a hop-by-hop option.

The characteristics of IPv6 flows and flow labels, or the rules that govern the flow label functions are further defined in [Deering&Hinden, 1998a]. The text from one paragraph in [Deering&Hinden, 1998a] was rearranged as an item list, as follows:

I. A flow is uniquely identified by the combination of a source address and a non-zero flow label.

II. Packets that do not belong to a flow carry a flow label of zero.

III. A flow label is assigned to a flow by the flow's source node.

IV. New flow labels must be chosen (pseudo-)randomly and uniformly from the range 1 to FFFFF hex. The purpose of the random allocation is to make any set of bits within the Flow Label field suitable for use as a hash key by routers, for looking up the state associated with the flow.

V. All packets belonging to the same flow must be sent with the same source address, destination address, and flow label.

VI. If packets of a flow include a Hop-by-Hop Options header, then they all must be originated with the same Hop-by-Hop Options header contents (excluding the Next Header field of the Hop-by-Hop Options header).

VII. If packets of a flow include a Routing header, then they all must be originated with the same contents in all extension headers up to and including the Routing header (excluding the Next Header field in the Routing header).

VIII. The routers or destinations are permitted, but not required, to verify that these conditions are satisfied. If a violation is detected, it should be reported to the source by an ICMP Parameter Problem message, Code 0, pointing to the high-order octet of the Flow Label field (i.e., offset 1 within the IPv6 packet).

IX. The maximum lifetime of any flow-handling state established along a flow's path must be specified as part of the description of the state-establishment mechanism, e.g., the resource reservation protocol or the flow-setup hop-by-hop option.

X. A source must not reuse a flow label for a new flow within the maximum lifetime of any flow-handling state that might have been established for the prior use of that flow label. When a node stops and restarts (e.g., as a result of a "crash"), it must be careful not to use a flow label that it might have used for an earlier flow whose lifetime may not have expired yet.

3.3 Proposed IPv6 Flow Label specifications

In order to provide a better quality of service of inter-network, some of the specification specify in [Deering&Hinden, 1998a] should be modify.

3.3.1 Random Generation of Flow Label Value

The flow label classifier fields have to known a priori, before traffic is being generated by a source of packets in order to provide different quality of service to different packet, This is contradicted by a random generation of the flow label value. In order to resolve this contradiction, rule marked (IV) in Section 3.2, extracted from [Deering&Hinden, 1998a], Appendix A, which states that the flow label should be pseudo-random, must be relaxed or removed.

3.3.2 Mutable/Non-mutable IPv6 Flow Label

Another topic of controversial discussion is whether the flow label should be mutable or non-mutable, that is it should be read-only for routers or not.

Statements that advocate a non-mutable characteristic are certainly based on the advantage of the simplicity implied by such a characteristic.

Opposite statements, that the flow label should be mutable, are based on the flexibility that this provides, in particular if the label has a hop-by-hop significance. However, using mutable flow labels would not work without a certain agreement, or negotiation between neighboring nodes (routers), or certain configuration of those routers. This would require the use of a negotiation mechanism between neighboring routers, or a certain setup through router management or configuration, to make sure that the values or the changes made to the flow label are known to all routers on the portion of the path of the packet, in which the flow label changes.

As the hop-by-hop significance of the flow label can be enhanced by a mutable characteristic, the specification or definition of the flow label should not preclude this.

A mutable flow label though requires the relaxation or elimination of the rules marked (I), (III), (IV), and (X) in Section 3.2. These rules were extracted from [Deering&Hinden, 1998a], Appendix A.

3.3.3 Using Random Numbers in setting the IPv6 Flow Label

The rule marked (IV) in Section 3.2, extracted from [Deering&Hinden, 1998a], Appendix A, specifies the requirement of pseudo-randomness in setting the value of a flow label. The reason given is the use of a hashing function, and hashing table for flow lookup by routers. Randomness certainly helps if the flow label is the only criterion used in the flow lookup.

The use of a hashing mechanism is one possible choice for the flow lookup in routers, or hosts.

Another possible choice is to use the label as an index in an array, which is a direct and faster lookup, or retrieval of the flow state, and so a contiguous set of values, starting from 1, would be more helpful, in particular if the flow label is not the only criterion used.

However, the authors of this document believe that the specification of the flow label should not mandate any implementation choices, whether they are random values, with hashing functions, or just contiguous values, with array indexing.

Furthermore, a random value in the header is introducing the unpredictability of the field. Although this may be an argument of philosophical nature, predictability is a necessary condition for deterministic behavior. Deterministic behavior is a MUST in a network. Network operators may require that packets of a flow have always the same IPv6 content. Random values in the IPv6 flow label certainly break such a requirement.

To resolve these issues would certainly require the relaxation or elimination of rule marked (IV), in Section 3.2, extracted from Appendix A of [Deering&Hinden, 1998a].

3.3.4 Characteristic of Proposed IPv6 Flow Label

The characteristics of IPv6 flows and flow labels are further defined as:

i. A flow is uniquely identified by the combination of source address, destination address and a non-zero flow label.

ii. A flow label of zero means that the flow label has no significance, the field is unused, and therefore has no effect on, or for the packet processing by forwarding, QOS, or filtering engines.

iii. A flow label is assigned to a flow by the flow's source node. It can be changed en-route, with the condition that its original significance be maintained, or restored, when necessary. For instance if the source of the flow intended that the flow label has a certain significance to the destination end-node, than the nodes en-route, that process and eventually change the value of the flow label, should make sure, in conjunction with the destination end-node, that even when the value or significance has changed en-route, the original information and significance is restored when or before the packet arrives to its destination.

If the action to be performed on a particular flow label is not known, a router must not change the value of that flow label.

iv. The flow label must have a value between preferred for choosing the value. However, the value must satisfy the following requirements:

a. It can be communicated to all routers on the path of the flow to the final destination, as well as the destination node, by ways of a resource reservation protocol, a flow label distribution protocol, a signaling mechanism, or by any other means.

b. It can be configured, uploaded, or transmitted to a router or a group of routers in any other possible way, as long as it can be stored in the classification rules tables of the forwarding engines of routers along the path of the flow to the final destination. The values of the flow labels are preset or agreed upon, and specified in a Service Level Agreement (SLA), Service Level Specification (SLS), Traffic Conditioning Agreement (TCA), or Traffic Conditioning Specification (TCS).

v. In general, all packets belonging to the same flow are sent with the same source address, destination address, and flow label. However, flows can be trunked, or aggregated in macro-flows. The flows, members of a macro-flow, may have different source or destination addresses. The trunking, or aggregation of flows is achieved by simply wildcarding some bits or all bits in some of the fields of the multi-field classification rules, which contain source address, destination address, and flow label. In other words range addresses and/or flow labels can be used.

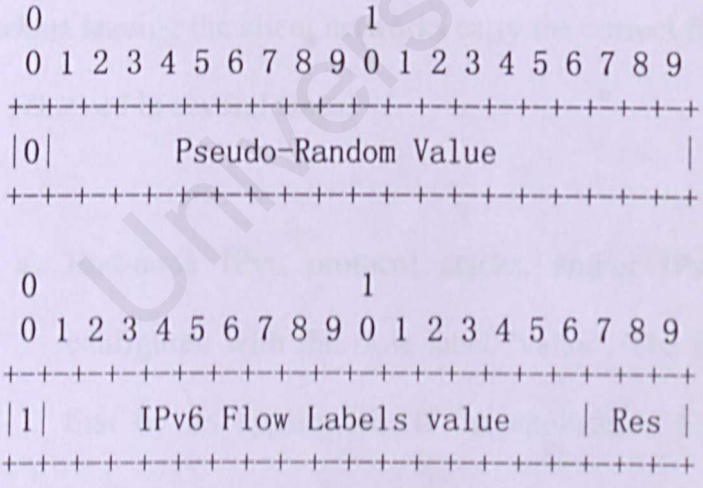
vi. The routers or destinations are permitted, but not required, to verify that these conditions are satisfied. If a violation is detected, it should be reported to the

source by an ICMP Parameter Problem message, Code 0, pointing to the high-order octet of the Flow Label field (i.e., offset 1 within the IPv6 packet).

vii. There is no time to live rule to flow label. However, changes to the value of a flow label of a flow, and/or the correspondent flow label classifier values MUST be synchronized. When the flow label value of a flow is changed, the change must be reflected in the change of the value of the flow label in the multi-Field flow label classifier.

3.4 IPv6 Flow Label Format

In order to preserve compatibility with the random number method of selecting a flow label value defined in [Deering&Hinden, 1998a], the following new format of the flow label could be used:



The "Res" bits are reserved for future use.

When the IPv6 router or switches receive a packet, it will first check the first bit of the flow label field. If the value of the bit is '0', it will simply forward that packet with

end-nodes will force the correct flow label in the IPv6 headers of outgoing packets.

If a. is not TRUE, then

b. The first hop routers would have to force the correct flow label on packets leaving the network. To accomplish this role, these routers would be configured with MF classifiers. These routers would classify the traffic that is forwarded downstream from, and away from the originating end-nodes. The action subsequent to the classification would be to set the correct flow label in each packet. Classification on such a router's input line card, or interface would result, for the matching packets, in a correct flow label being forced in the IPv6 headers of packets when they are transmitted on the output interface or line card.

2. Packets coming into the provider network can be policed based on flow label. The provider, based on the SLAs, SLSs, TCAs, TCSs agreed with the client, configures MF classifiers that look like:

C = (SA, SAPrefix, DA, DAPrefix, Flow-Label)

Or

C' = (SA, SAPrefix, DA, DAPrefix, Flow-label-Min: Flow-label-Max)

Where

C = MF Classifier

SA = Source Address

SAPrefix = Source Address Prefix

DA = Destination Address

DAPrefix = Destination Address Prefix

Flow-Label = Flow Label's value

Another representation of the classifier for example is:

Flow-label-classifier:

Type : IPv6-3-tuple

IPv6DestAddrValue : 1:2:3:4:5:6:7:8::1

IPv6DestPrefixLength : 128

IPv6SrcAddrValue : 8:7:6:5:4:3:2:1::2

IPv6SrcPrefixLength : 128

IPv6FlowLabel : 57

Or

Flow-label-classifier:

Type : IPv6-3-tuple

IPv6DestAddrValue : 1:2:3:4:5:6:7:8::1

IPv6DestPrefixLength : 128

IPv6SrcAddrValue : 8:7:6:5:4:3:2:1::2

IPv6SrcPrefixLength : 128

IPv6FlowLabelMin : 1

IPv6FlowLabelMax : 57

The classifiers are configured in the network provider's edge routers, etc...

The classification engines in those routers would match packet header information to classification rules as follows:

Incoming packet header (SA, DA, Flow Label)

Match

Classification rules table entry (C or C')

Where

SA = Source Address

DA = Destination Address

Flow Label = Flow Label's value

C = Classifier

3.5 Summary

This chapter has explained about what the IPv6 flow and IPv6 flow label is. The characteristics and specification of IPv6 Flow Label have been discussed through out the chapter. The proposed IPv6 Flow Label specification has been discussed. The following chapter will discuss the analysis for the system.

Chapter 4 System Analysis

Systems analysis and design seeks to systematically analyze data input or data flow, processing or transforming data, data storage, and information output within the context of a particular business. Systems analysis and design is used to analyze, design, and implement improvements in the functioning of businesses. Installation of a system without proper planning will leads to great dissatisfaction and frequently causes the system to fall into disuse. Systems analysis and design lends structure to the analysis and design of information systems, a costly endeavor that might otherwise have been done in a haphazard way. It can be thought of as a series of processes systematically undertaken to improve a business through the use of computerized information systems.

4.1 Software and Hardware Selection

4.1.1 Language Selection

The step of considering the advantages and disadvantages of programming language is very necessary in building a simulator or any application programs. The selection of a proper programming language play an important role on the functionally of the program. Below are the discussions about both procedural programming and object-oriented programming.

4.1.1.1 Procedural Programming

The procedural approach is the earliest method to use in programming of computing.

There are procedures or function which is mean by a place that the procedural language placed code into blocks. Procedural program is written as a list of instructions, telling the computer, step-by step, what to do like open a file, read a word, counting number, divide by 8, display something and other else. Most traditional computer languages like Pascal, C and FORTRAN are procedural. Procedural programming is fine for small projects. It is very simply and is the most natural way to tell a computer what to do. Because of the computer processor's own language, machine code is procedural, so the translation of the procedural high-level language into machine code is very straightforward and efficient. Besides, procedural programming can simplified those complicated steps of some process. Procedural programming has a building way of splitting big lists of instructions into smaller lists which is named functions.

Each of these blocks was to act like a black box, which completed one task or another. This type of programming believed that one could always write these functions without modifying external data. There are also difficult problems with this language method is to write all functions in such a way that they actually do not modify data outside their boundary. So, when functions began changing data outside their boundary like C is done by passing a pointer, a problem called coupling began to surface. Here now developing a new programming approach call object-oriented programming language.

In a procedural-based programming language, a programmer writes out instructions that are followed by a computer from start to finish. For example, a procedural oriented program might work like this:

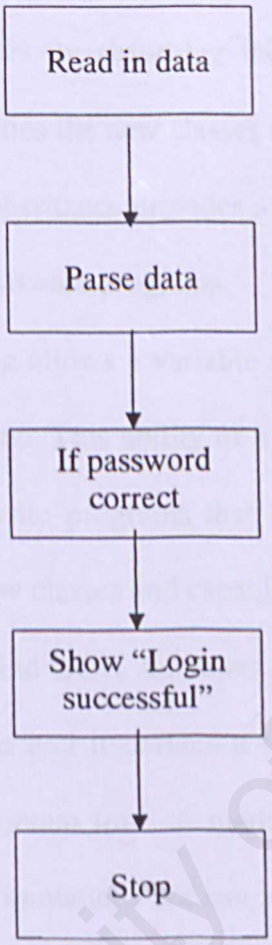


Figure 4.1 Simple procedural program

4.1.1.2 Object-oriented Programming

Object-oriented programming is the result of many years of theoretical development, and many consider it the current extension of the theory behind modular programming, in which code is combined into reusable modules. It is a programming model that views a program as a set of self-contained objects. These objects interact with other objects by passing messages. Object-oriented programming also lets you create procedures that work with objects whose exact type may not be known until the program actually runs.

There are two key components in Object-oriented programming (OOP) – inheritance and polymorphism. Inheritance is a form of software reusability in which new classes are created from existing classes by absorbing their attributes and behaviors and embellishing these with capabilities the new classes require. A class inherits state and behavior from its super-class. Inheritance provides a powerful and natural mechanism for organizing and structuring software programs.

The concept of dynamic binding allows a variable to take different types dependent on the content at a particular time. This ability of a variable is called polymorphism. Polymorphism enables us to write programs that handle a wide variety of related classes and facilitates adding new classes and capabilities to a system.

Objects are the central idea behind OOP. An object is an instance of a class. It can be uniquely identified by its name and it defines a state which is represented by the values of its attributes at a particular time. A method is similar to a procedure. The basic idea behind an object is simulation. We can ask an object to perform a method without knowing what its class is. Instead of calling a function to perform some operation on an object, we send that object a message asking it to perform that operation on itself. Depending on the class of the object, different code will be executed by the computer. This is useful since a given message may be meaningful to different classes of objects (e.g. "size" is meaningful both to a stack and a queue even though each may calculate its size in a very different way).

Encapsulation is the concept in which objects contain both data and methods. 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. Similarly, one does not need to know how a class is implemented, but just to know which methods to invoke.

The concepts and rules used in object-oriented programming provide these important benefits:

- The concept of a data class makes it possible to define subclasses of data objects that share some or all of the main class characteristics. Called inheritance, this property of OOP forces a more thorough data analysis, reduces development time, and ensures more accurate coding.

- Since a class defines only the data it needs to be concerned with, when an instance of that class (an object) is run, the code will not be able to accidentally access other program data. This characteristic of data hiding provides greater system security and avoids unintended data corruption.

- The definition of a class is reusable not only by the program for which it is initially created but also by other object-oriented programs (and, for this reason, can be more easily distributed for use in networks).

- The concept of data classes allows a programmer to create any new data type that is not already defined in the language itself.

4.1.1.3 JAVA Programming Language

Java is a programming language and development environment created by Sun Microsystems, designed to create distributed executable applications for use with a Web browser containing a Java runtime environment. Java technology has been licensed by literally hundreds of companies, including IBM, Microsoft, Oracle, Hewlett-Packard, and others interested in developing Web-based and platform-

independent applications. It is a powerful programming language built to be secure, cross platform and international.

The Java programming language is a portable, object-oriented language, loosely modeled after C++, with some of the more troubling C++ constructs such as pointers removed. This similarity to C and C++ is no accident; it means that the huge population of professional programmers can quickly apply their previous C experience to writing code in Java.

Java is designed to support networking and networking operations right from the start and begins with the assumption that it can trust no one, implementing several important security mechanisms.

Java is a small, simple, safe, object-oriented, interpreted or dynamically optimized, byte-coded, architecture-neutral, garbage-collected, multithreaded programming language with a strongly typed exception-handling mechanism for writing distributed, dynamically extensible programs. Java is architecturally neutral, does not care about the underlying operating system, and is portable because it makes no assumptions about the size of data types and explicitly defines arithmetic behavior. Security and safety are main features of Java programming language. Its execution semantics guarantees that every run-time error is detected and reflected in a throw exception. Java eliminated the use of pointers of

C++ and replaced it with references, which prevents program from accessing illegal areas of the system's memory. Besides, Java also supports dynamically run time method identification. Libraries of Java is supported through the use of packages and allow complicate programs to be build but the overhead of keeping track of all libraries is reduced.

Java is also multithreaded to support different threads of execution and can adapt to a changing environment by loading classes as they are needed, even across a network. Rather than writing code targeted at a specific hardware and operating-system platform, Java developers compile their source code into an intermediate form of byte-code that can be processed by any computer system with a Java runtime environment. The Java class loader transfers the byte-code to the Java Virtual Machine (JVM), which interprets the byte-code for that specific platform.

Java class libraries, those files that make up the standard application programming interface (API), are also loaded dynamically.

The runtime environment then executes the application, which can run within a Web browser or as a stand-alone application.

4.1.2 Integrated Development Environment (IDE) Selection

Jbuilder will be used to implement the simulator in this project. Jbuilder and other Integrated development environment (IDE) for Java is discussed in the following section:-

4.1.2.1 Visual Age

The VisualAge for Java product is IBM's integrated development environment (IDE) for Java developers. VisualAge for Java allows transforming existing applications for the Web.

Below are the benefits of using Visual Age:

- Multiple developers can work on multiple projects, with automatic versioning control that facilitates the rapid creation and deployment of applications to the WebSphere platform,
- Don't start from scratch - reuse the existing applications and extend them to e-business with VisualAge for Java. rather than coding to low-level interfaces,
- Helps reduce overall effort and cost to build, deploy, and maintain Java applications:
- Helps reduce effort of change and maintenance.
- Scalable data solutions - Leverage the complete range of WebSphere servers and supporting platforms.
- Easy to use - With a persistence framework and unit test environment for WebSphere, VisualAge for Java provides a fast way to develop, test, and deploy end-to-end e-business applications.
- Improved interoperability with other tools - Investments in tools and skills are maintained through integration with VisualAge for Java.

Who should be the user of VisualAge for Java?

- Java technology developers or teams of Java developers building e-business applications.
- Java technology developers, web integrators, and systems integrators creating dynamic web applications targeted to the WebSphere Application Servers.

4.1.2.2 KAWA

Kawa is a full Scheme implementation. It implements almost all of R5RS, plus some extensions. By default, symbols are case sensitive. It is completely written in Java. Scheme functions and files are automatically compiled into Java byte-codes. Kawa does some optimizations, and the compiled code runs at reasonable speed. Kawa uses Unicode internally, and uses the Java facilities to convert files using other character encodings. Kawa provides the usual read-eval-print loop, as well as batch modes. Besides, Kawa also provides a framework for implementing other progressing languages, and comes with incomplete support for CommonLisp, Emacs Lisp, and EcmaScript, and the draft XML Query language.

Kawa is written in an object-oriented style and has builtin pretty-printer support, and fancy formatting. Kawa supports class-definition facilities, and separately-compiled modules.

Kawa implements the full numeric tower, including infinite-precision rational numbers and complex numbers. It also supports "quantities" with units, such as `3cm`. User can optionally declare the types of variables and also can conveniently access Java objects, methods, fields, and classes.

Kawa implements most of the features of the expression language of DSSSL, the Scheme-derived ISO-standard Document Style Semantics and Specification Language for SGML. Of the core expression language, the only features missing are character properties, `external-procedure`, the time-relationed procedures, and character name escapes in string literals. Also, Kawa is not generally tail-recursive. From the full expression language, Kawa additionally is missing `format-number`, `format-number-`

list, and language objects. Quantities, keyword values, and the expanded lambda form (with optional and keyword parameters) are supported.

4.1.2.3 Jbuilder

Increase productivity with visual development tools providing maximum flexibility for creating Pure Java applications on the development platform such as Windows®, Linux® and Solaris™ as well as the latest Java® technologies, including applets, JSP/Servlets, JavaBeans®, Enterprise JavaBeans®, and distributed CORBA® applications. JBuilder features the unmatched AppBrowser™ environment with XML-based project manager, StructureInsight™, HTML and XML viewers, advanced graphical smart debugging, CodeInsight™ coding wizards, extensible code editor, Two-Way-Tools™, visual JFC/Swing designers, BeansExpress™, DataExpress™, and lightning-fast compiler.

Some features of Jbuilder are as follows:

- Enhanced editor with advanced syntax highlighting
- Visual Studio and Brief keymap bindings
- Visual keymap editor
- HTML 4, XML and CSS cascading style-sheet layout viewer
- Documentation Type Definition (DTD) support
- Expanded search and save options
- CodeInsight package context view
- StructureInsight
- Hosted on JDK 1.3 with built-in Hotspot™ Client Virtual Machine for ultimate performance

- Fast load time
- Integrated package migration tool to easily import existing Java source packages
- Message view toggle to turn message view on or off
- Multiple instances of AppBrowser
- Context sensitive help (F1) in source code and designers
- Package view for displaying source packages in project view
- Enhanced Open Tools API
- Additional Open Tools API documentation
- Referential computer based training CD
- Enhanced Servlet wizard supports XHTML, WML and XML Servlets
- Many new EJB wizards including EJB Group, EJB Group from Descriptors,
- Enterprise JavaBean, Entity Bean Modeler, EJB Test Client and EJB Interface
- Archive Builder to build deployable archives quickly
- Archive viewer for JAR, ZIP, WAR, EAR
- Extended Project wizard for adding existing Java projects
- Wizards
- Library configuration wizard for defining your project and user defined libraries
- JDK configuration wizard

4.1.3 Hardware Selection

4.1.3.1 Workstation

Dell Optiplex with 256 Mbytes of RAM is used to develop the network simulator. Remote-login can be used for the ease of development. The development can be done using any Dell workstations available.

4.2 Phases of Implementation

Several phases are required in the building of the simulation environment for IPv6 Flow Label:

1. Creates IPv6 packet according to IPv6 Protocol Specification [Deering&Hinden, 1998a] including the IPv6 header and IPv6 extension headers.
2. Ready the basic IPv6 environment, such as implement Neighbor Discovery [Narten et al, 1998], Stateless Address Autoconfiguration [Thompson&Narten, 1998] and Internet Control Message Protocol Version 6 (ICMPv6) [Conta&Deering, 1998].
3. Building the Unicast routing protocol, the Routing Information Protocol next generation (RIPng) [Malkin&Minnear, 1997].
4. Ready the basic IPv6 Flow Label environment, such as implementing IPv6 Flow Label field in IPv6 packet, Flow Label constant.
5. Implement the Weighted Round Robin Queue scheme.

4.3 Analysis of UMJaNetSim

4.3.1 Overview of Simulation Environment

Simulation plays a valuable role in the design, evaluation and experimentation of new networking protocols, theories or algorithms. Simulation allows the experiments and analysis of a system without the need to construct the actual system, as it's usually too expensive and impractical to build an actual system for every experiment.

JaNetSim Network Simulator, a discrete event, Java-based object-oriented network simulator developed by Lim, S.H. et al. (2001) is used as the base engine for the simulation. The UMJaNetSim architecture mainly consists of two important parts, namely the simulation engine and the simulation topology. The basic objects of the simulator are shown in the following figure.

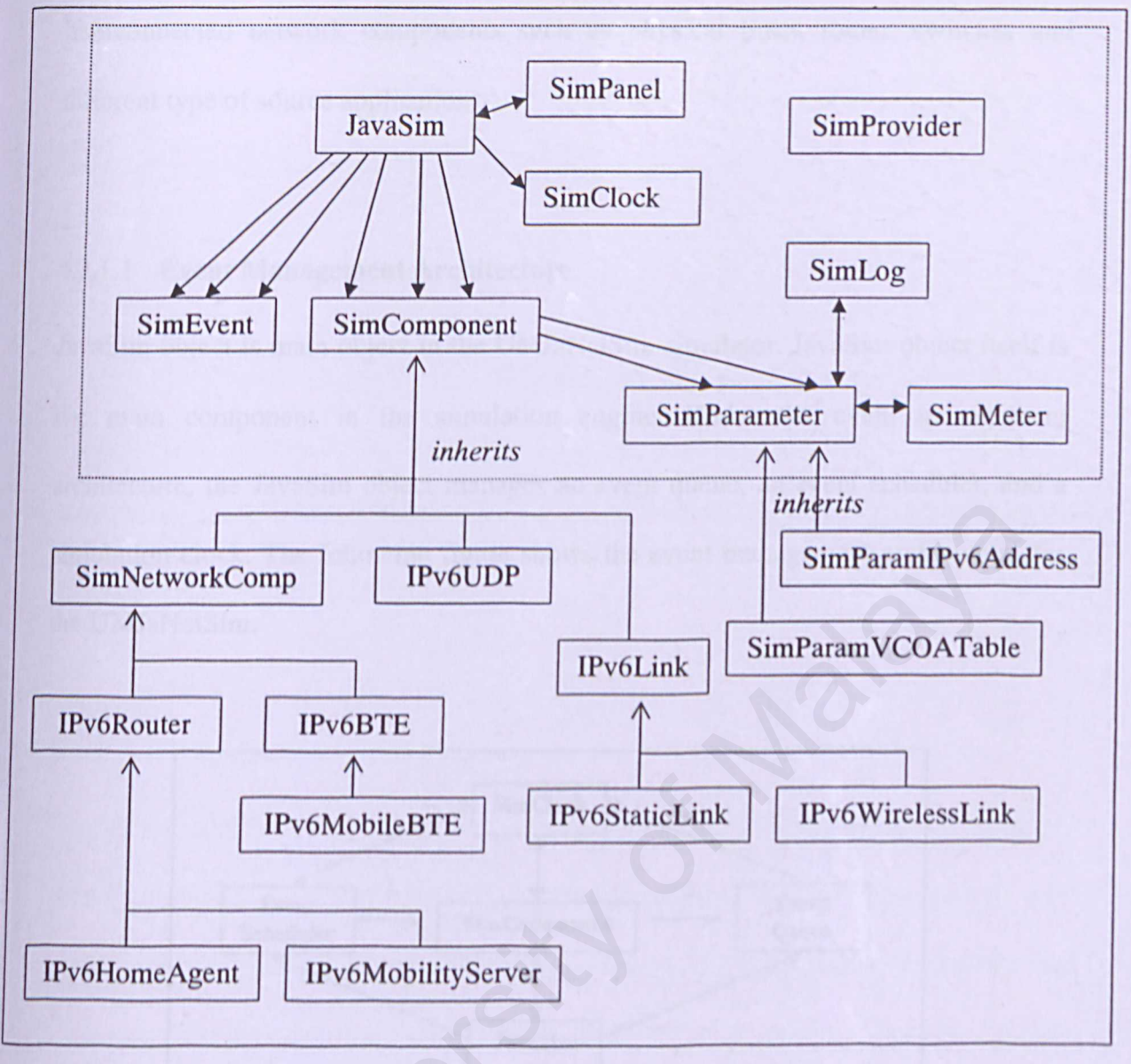


Figure 4.2 UMJaNetSim Network Simulator Objects

The simulation engine is the main controller of the entire simulation. It performs the event management task, GUI management as well as input and output processing such as topology's saving and result logging.

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 consist of a group of

interconnected network components such as physical links, router, switches and different type of source applications.

4.3.1.1 Event Management Architecture

JavaSim object is main object in the UMJaNetSim simulator. JavaSim object itself is the main component in the simulation engine. Under the event management architecture, the JavaSim object manages an event queue, an event scheduler, and a simulation clock. The following figure shows the event management architecture for the UMJaNetSim.

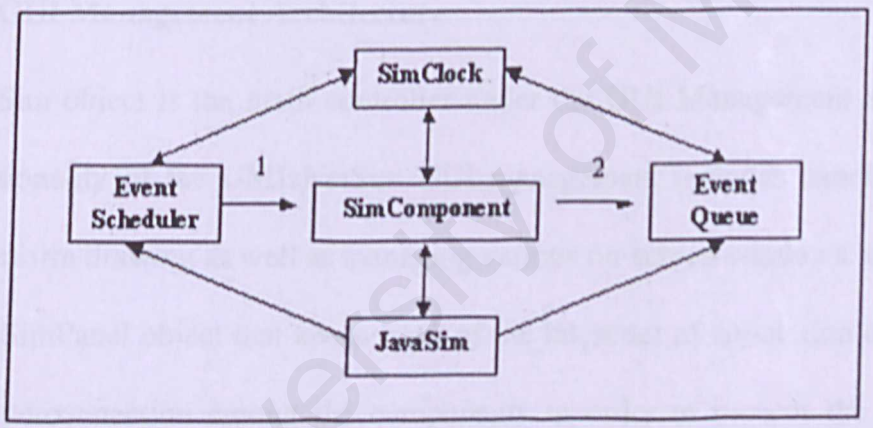


Figure 4.3 Event Management Architecture

Under the event management architecture, there are basically two main operations that are in process. First, a simulation component will schedule an event for a target component to be happening at a specific time using the en-queue operation. This target component can be others component or the component itself. Then, the events are stored inside a queue and are sorted by the event-firing time.

When a specific time is reached, the simulation engine will invoke the event handler of the target component. In this time, the event scheduler will fetch and remove the first event in the event queue. The target component will react to the event according to its behavior.

UMJaNetSim uses an asynchronous approach of the discrete event model, where any event can happen at any time. The SimClock object is the global time reference used by every component in the simulation and managed by the simulation engine. The simulation time in the UMJaNetSim is measured by tick and this tick can be converted to real time or vice versa.

4.3.1.2 GUI Management Architecture

The JavaSim object is the main controller under the GUI Management architecture. The functionality of the UMJaNetSim GUI management includes handling of user inputs; perform drawing as well as managing various on-screen windows. Besides that, there is a SimPanel object that 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. Thus, the new topology can be designed easily. The following figure shows the GUI management architecture for the UMJaNetSim.

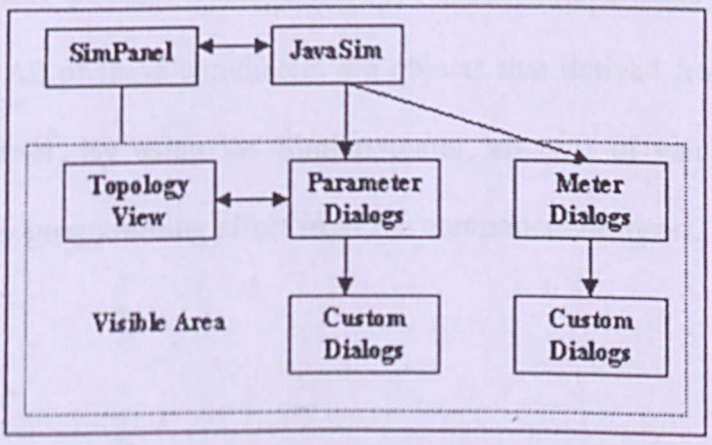


Figure 4.4 GUI Management Architecture

Under the GUI management architecture, there are two common types of dialog, namely the parameter dialogs and the meter dialogs. Each simulation component is associated with a parameter dialog that displays the list of parameter for the component. Meanwhile, a meter dialog is normally used for graphical display of a particular output value of a component such as utilization as well as cell loss. Besides that, the two types of dialogs are also associated with one or many custom dialogs that show extra information about the information. These custom dialogs are normally the OSPF routing table, VPN detail and others information.

4.3.1.3 Simulation Components

The SimComponent object is the main simulation object in the UMJaNetSim. It is a well-defined base object with all the necessary interfaces that enable the interaction between the simulation engine and the component. There will be an event handler in every simulation component, which is invoked by the event scheduler in order to fire an event. All interactions between simulation components are achieved through the sending of messages in the form of a SimEvent.

Each of the SimComponent is associated with its properties that can be configured by using the parameter dialog. There are many types of parameters for each of the SimComponent. All of these parameters are objects that derived from a base object called SimParameter. By using the SimParameter, all type of values can be setup easily without any programming effort from the component designer.

4.3.2 UMJaNetSim API

After the study of the UMJaNetSim architecture, this section focuses on defining well-known interfaces for simulation objects. The UMJaNetSim API provides a consistent way of creating simulation components. Table 4.1 shows the major objects of UMJaNetSim and their functions.

Table 4.1 Major objects of UMJaNetSim and functions.

Object	Major Functions
JavaSim	The JavaSim object is the main object in the UMJaNetSim simulator. JavaSim object manages an event queue, an event scheduler and a simulation clock during the event management and provides all GUI functions together with SimPanel under the GUI management. The following are some of the important method of the JavaSim object. java.util.List getSimComponents() – This method returns a list of all existing SimComponent. long now() – This method returns current simulation time (in tick). boolean isCompNameDuplicate(String name) – This method ensures no duplicated name for SimComponent. void notifyPropertiesChange(SimComponent comp) – This method executes whenever there are

	structural changes to the parameters. void enqueue(SimEvent e) – This method will enqueue an event that has been invoked. void dequeue(SimEvent e) – This method will dequeue an event when a specific time is reached for that event to be executed.
SimEvent	Every SimComponent communicates with each other by enqueuing SimEvent for the target component. For example, when component A wants to send a cell 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. There are two types of events, namely the public (well-known) events and private events. Public events are defined in the SimProvider object. This event can be enqueued for itself and or for another SimComponent. All private events are defined within the particular SimComponent source itself and can only be enqueued for itself. The following are some of the important method of the SimEvent object. SimComponent getSource() – This method retrieves the source SimComponent.

	SimComponent getDest() – This method retrieves the destination SimComponent. int getType() – This method retrieves the event type. long getTick() – This method retrieves the event-firing time. Object [] getParams() – This method retrieves the event parameters.
SimClock	The SimClock object is the global time reference used by every component in the simulation and managed by the simulation engine. The following are some of the important method of the SimClock object. static double Tick2Sec(long tick) – This method converts ticks to seconds. static double Tick2MSec(long tick) – This method converts ticks to milliseconds. static double Tick2USec(long tick) – This method converts ticks to microseconds. static long Sec2Tick(double sec) – This method converts seconds to ticks. static long MSec2Tick(double msec) – This method converts miliseconds to ticks. static long USec2Tick(double usec) – This method

	converts microseconds to ticks.
SimComponent	The SimComponent object is the main simulation object in the UMJaNetSim from the topology view. Every components such as link, broadband terminal equipment, application, switch and other network components must inherit this base class in order to obtain the capability to interact with the simulation engine. Every SimComponent has a reference to the main object of the simulation engine, the JavaSim object, in order to access services provided by the simulation engine. Every SimComponent also maintains a list of all its external parameters and a list of all its neighbors. The neighbors of a component are all components that are directly connected to the component. The following are some of the important properties and method of the SimComponent object. protected transient JavaSim theSim – This property is a reference to the main JavaSim object. protected java.util.List neighbors – This is a list of all (directly connected) neighbors of the SimComponent protected java.util.List params – This is a list of all external parameters of the SimComponent

	Object [] compInfo(int infoid,SimComponent source, Object [] paramlist) - This method provides a way for inter-component information exchange without sending run time events. boolean isConnectable(SimComponent comp) – This method is called by the simulation engine when a new component is about to be connected to this component. This method will verify whether the new component can be connected to this component. void addNeighbor(SimComponent comp) – This method is called by the simulation engine when a new component is connected to this component. void removeNeighbor(SimComponent comp) – This method is called by the simulation engine when a new component is disconnected to this component. void copy(SimComponent comp) - This method is used to copy parameter values from another SimComponent of the same type. void reset() – This method performs a reset operation in order to bring the status of the component back to the same status as if it is just newly created. void start() – This method performs any operations

	needed when the simulation starts. void resume() – This method perform any operations needed when the simulation need to be resume. One possible use is to capture any special changes /that have been done by the user during the pause period. void action(SimEvent e) - This is the event handler of this component, and will be called by the simulator engine whenever a SimEvent with this component as the destination fires.
SimParameter	Every SimComponent has internal parameters or external parameters, which can be shown or accessible by users. All external parameters must inherit SimParameter. By extending SimParameter, the components obtain parameter logging and meter display features automatically. The parameter can simply holds one single value, such as an integer. It also can represent a complex piece of information, such as the entire routing table of a network router. In some cases, the parameter itself can create and manage additional custom dialogs. A complex parameter type may just use a JButton that opens up new custom dialogs when invoked. The choice of components to use is dependent on

	the type of interaction needed by the component designer. The following are some of the important method of the SimParameter object. String getString() – This method returns a String representation of the parameter value. This is used for logging purpose. void globalSetValue(String value) – This method supports setting of the same parameter values for multiple components in one command. int getValue() – This method will read a value. void setValue(int val) – This method will write a value. JComponent getJComponent() – This method will return a Java swing component and its name.

4.3.3 Network Simulation Component

Several components are created during the process of preparing the simulation environment for Mobile IPv6.

Table 4.2 Major Network Simulation Component

SimNetworkComp	SimNetworkComp represents a network component that is connectable with each other in the network via links. This class inherits SimComponent and will be further inherited by IPv6Router and IPv6BTE. It performs the following functions:

<table>
<tr><td></td><td>

Implement Neighbor Discovery Protocol on all physical links of the component
Implement output queue when physical link is busy
Perform packing/unpacking function for data link packet

The following table lists the design of the major attributes and methods of the SimNetworkComp class.
<table>
<tr><th>Major Attribute</th><th>Major Method</th></tr>
<tr><td>A list of physical links</td><td>Add links, remove links</td></tr>
<tr><td>A list of instantiated Neighbor Discovery Protocol Processor</td><td>Kickstart the processor, resolve IP address into MAC address</td></tr>
</table>
</td></tr>
<tr><td>IPv6BTE</td><td>IPv6BTE inherits the SimNetworkComp, act as the Broadband Terminal Equipment (B-TE) in the IPv6 environment. It represents the aggregate traffic from the customer site and support route optimization as defined in Mobile IPv6. It performs the following functions:

Perform pack/unpacking function for IPv6 packet

</td></tr>
<tr><td>IPv6Router</td><td>IPv6 Router inherits the SimNetworkComp, act as a generic router in the IPv6 environment. It supports RIPNG, Internet Control Message Protocol Version 6 (ICMPv6). It performs the following functions:

Enqueue the IPv6Packet into the correct flow label queue.
Route packets among subnets by weighed round robin scheme.

</td></tr>
</table>

	➢ Perform RIPv6 protocol to maintain routing table

Major Attribute	Major Method
Routing table	Add, update and remove next hop for all subnets

IPv6TCP	IPv6TCP inherits the SimComponent, which acts as the source of the IPv6 traffic. It performs the following functions: ➢ Generate data as traffic source ➢ Maintain a segment token queue to keep track of acknowledgment from peer ➢ Mark the IPv6 Flow Label value

Major Attribute	Major Method
Token queue	Take a token for each outgoing segment, put back a token for each acknowledgment from peer

IPv6Link	IPv6Link inherits the SimComponent, which acts as a generic link for IPv6 network. It supports packet switching. It performs the following functions: ➢ Switch packet among different network components

Major Attribute	Major Method
A list of packets	A list of packet now being transferred by the link. Take the packet, identify the

<table>
<tr><td></td><td></td><td>destination MAC address then switch it to the correct device</td></tr>
<tr><td>IPv6StaticLink</td><td colspan="2">IPv6StaticLink inherits the IPv6Link, which acts as a static link for IPv6 network. It performs the following functions:
➢ Switch packet among different network components

<table><tr><th>Major Attribute</th><th>Major Method</th></tr><tr><td>A list of packets</td><td>A list of packet now being transferred by the link. Take the packet, identify the destination MAC address then switch it to the correct device</td></tr></table></td></tr>
</table>

4.3.4 Objects and Classes

Besides the new components added, several helper java files are included as well in the UMJaNetSim network simulator. The java files added are listed as follows.

4.3.4.1 ICMPv6Header.java

It defines structure of an ICMPv6 header in IPv6 packet

4.3.4.2 ICMPv6Processor.java

It defines structure for ICMPv6 messages and handles the ICMP messages in IPv6 packet. It performs the following functions:

- Create Ping message and respond to Ping message

4.3.4.3 IPv6Address.java

It stores IPv6Address and provides helper function that manages the address. It performs the following functions:

- Create IPv6 address from string
- Check validity of an IPv6 address
- Compare two IPv6 address for similarity

Table 4.3 Major Attributes and Methods for IPv6Address

Major Attribute	Major Method
IPv6 Address	Initialize the address from string, check the validity of the address, compare a given IPv6 address for similarity, compare a given subnet for matching

Major Attribute	Major Method
list of IPv6 headers	Initialize the address from string, check the validity of the address, compare a given IPv6 address for similarity, compare a given subnet for matching

4.3.4.4 IPv6Header.java

It defines structure for IPv6 header and provides helper functions to manage the headers in an IPv6 packet. It performs the following functions:

- Add new header into an IPv6 packet
- Search IPv6 packet for a particular type of header

Table 4.4 Major Attributes and Methods for IPv6Header

Major Attribute	Major Method
A list of IPv6 headers	Initialize the address from string, check the validity of the address, compare a given IPv6 address for similarity, compare a given subnet for matching

4.3.4.5 IPv6Packet.java

It defines structure for an IPv6 packet

4.3.4.6 IPv6RouteEntry.java

It defines the structure of a route entry in RIPv6 routing table

4.3.4.7 NeighborDiscoveryProcessor.java

It defines structure of Router Advertisement and Router Solicitation messages, other Neighbor Discovery Protocol options and handles the messages in IPv6 packet. It performs the following functions:

- Create Router Solicitation, Router Advertisement
- Handle Router Solicitation and Router Advertisement
- Recognize a given IPv6 address whether the address is on its subnet
- Resolve a given IPv6 address into MAC address
- Handle outgoing packet from client, put the packet into a queue until the outgoing link is free

Table 4.5 Major Attributes and Methods for NeighborDiscoveryProcessor

Major Attribute	Major Method
Packet Queue	Add outgoing packet into the queue when link is busy, remove packet from the queue when link is free

4.3.4.8 RIPv6Processor.java

It defines structure of RIPv6 messages and handles the RIPv6 messages in IPv6 packet. It performs the following functions:

- Create RIPv6 messages to neighboring routers

- Handle and update routing table based on the RIPv6 messages received from neighboring routers

Table 4.6 Major Attributes and Methods for RIPv6Processor

Major Attribute	Major Method
Route Table	Add, update and remove route from the table when receive RIPv6 messages from neighboring routers

4.3.4.9 SimParamAddressExpiryTimeTable.java

It defines structure of a table that contains entries of IPv6 address and corresponding expiry time. It provides helper functions to manage the table.

4.3.4.10 SimParamFileChooser.java

It allows user to browse for a file in the local file system. It provides helper functions to read and parse input from the file.

4.3.4.11 SimParamHomeAddressTable.java

It defines structure of a table that contains entries of mobile home address, mobile current care-of-address and corresponding expiry time. It provides helper functions to manage the table.

4.3.4.12 SimParamIPv6Address.java

It stores and displays an IPv6 address in component property page.

4.3.4.13 SimParamIPv6RTable.java

It defines structure for a RIP routing table and stores the table. It provides helper functions that manage the table and calculate least-cost route to a destination subnet.

4.3.4.14 SimParamSubnetTable.java

It defines structure of a table that contains entries of subnet. It provides helper functions to manage the table and search for matching subnet for a given IPv6 address.

4.4 Summary

Software and hardware selections have been discussed in the beginning of the chapter. Some introduction regarding the existing programming languages and integrated development environment had been done. This chapter also reveals the phases of implementation for the IPv6 Flow Label into the existing UMJaNetsim.

A thorough analysis of UMJaNetSim has been done. All of the important objects and classes had been explained.

Chapter 5 System Design

This chapter describes the overall system architecture design, class design, algorithm design and simulator design overview.

5.1 System Architecture Design

There are two main component of the IPv6 Flow Label module in UMJaNetSim:

1. IPv6 router: IPv6 router marks packets with a flow label's value according to the policy specified. It also examines packets' flow label's value marking and forwarding them accordingly.
2. IPv6 TCP: IPv6 TCP acts as the traffic source of the simulation. It will specify which flow it belongs to.

When traffic source generate the packet, it will specify that it belong to which flow. When a packet is passed to the IPv6 router, the IPv6 router will first check to the IPv6 packet whether its flow label field has been mark. If the field has been mark, then the router will put the packet into the correspondent flow queue, waiting to send to next hop. The queuing algorithm being using is Weighted Round-Robin queuing algorithm. Every flow gets its turn to send a number of packets that is determined by the flow's weight.

If the IPv6 router found that the flow label field in the packet has not been mark, it will simply forward the packet without any special quality of service.

When the packet leave the next edge router, reach the destination, the edge router will the remove the value in the flow label.

5.2 Classes Design

This section gives a description on classes design in this project. Among the classes highlighted are IPv6 Router and IPv6 TCP.

5.2.1 IPv6 Router

It has all the attributes and behaviors of generic IPv6 router plus the flow label support. It supports RIPNG, Internet Control Message Protocol Version 6 (ICMPv6). It performs the following functions:

1 Route packets among subnets

2 Perform RIPv6 protocol to maintain routing table

3 Put the receiving packet into the corresponding flow queue.

4 Output the packet according to the weight specify in the weighted round-robin weight attribute.

This class provides the functionality of en-queue the packet with the Weighted Round-Robin algorithm. When the IPv6 router reads the flow label's value in the packet, it will put the packet into the corresponding queue base on the policy.

Each queue in the weighted round-robin will be representing by a buffer.

5.2.2 IPv6 TCP

Class IPv6 TCP acts as the generic traffic source for the simulator under the flow label environment. This class will specify which flow type the packet belongs to before it had been send. It can specify how many packets to be send. It will also show how many packets had been received.

5.3 Basic Algorithm

Executing a series of actions in specific order can solve computing problem. An algorithm is a procedure for solving a problem in term of:

1. the action to be executed,
2. the order in which these actions are to be executed

The following is the basic algorithm of Edge Router and Core Router.

5.3.1 IPv6 Router Algorithm

When a packet from the traffic source passed to the IPv6 router, the router will run its process:

If the packet flow label field has been mark {

Check the flow label value

Compare the value with the policy

Put the packet into the corresponding flow queue

If the total packets in the queue exceed the queue size{

Drop the packet

}

}

Else {

put the packet into the default quality queue

}

Schedule the output packet according to the WRR weight

5.4 Simulator Design Overview

Simulation model would firstly introduce the simulator architecture design that provides information for the user to create network topologies using simulated IPv6 router, IPv6 Packet, and physical links.

Features that available in UMJaNetSim include a variety of applications, the behavior of which will determine the kind of traffic generated for transmission through the network. Users have the ability to control the parameters associated with these components, define the routes, and specify many details concerning the logging and display of performance data.

User interface to the simulator is through a Java application display screen. The screen simultaneously displays the network configuration, a control panel for running the simulation, and parameter information. The display contains a text window for user prompts which also provides a place for parameter data entry. Output parameter values may be displayed in numerical form in "information windows" or as graphical

"meter". Output parameter values may also be tagged for logging to a file; the date logging frequency is user dependent.

5.4.1 Graphical User Interface Design

Interface can improve the efficiency and effectiveness of the user when using the simulator. Thus, the interface design for the UMJaNetSim must be easy to understand and easy to use. The users no need to remember any DOS commands and what they need to do is just some mouse click. We have to create the interface design as friendly as possible. The aim of this design is able to prevent failures and improper procedures.

5.4.2 Design of Screen

The design of the graphical user interface display for the simulator is divided into 3 major parts:

1. A network window to display IPv6 network configuration. This window is used both while creating the configurations and to show network activity while the simulation is running.
2. A text window for messages that will prompt the user, and to provide a place for the user to input text or parameter values.
3. A control panel that consists of a clock and several control buttons, such as START, RESET, RESUME and etc.

5.4.3 The Network Window

The default setting for network windows is a blank area where user can start built their own network topology in this area. To use the function of the system, the user just have to click on the selected tasks that are available such as button to create a IPv6 Edge Router, IPv6 Core Router, IPv6 Packet and physical links. Add and drop feature are also support by the simulator, making it as friendly as possible.

The physical links are also considered components and are identified by name, but they are represented on the figure by straight lines. The connection between a Edge Router and an IPv6 application is represented by a line but is not considered a component, i.e., it is not a physical entity and has no associated parameters.

5.4.4 The Control Panel

The control panel appears on the bottom part of the screen. It contains a digital clock, and an array of control buttons. This button includes START, RESET, CONNECT MODE and PAUSE. The digital clock indicates the passage of simulator time in a graphic style.

5.5 Expected Design Output

The design of output serves the purpose of providing the information that the user needs, based on the criteria selected by the users:

- User can create any kind of network topology on the working space.

- The simulator can correctly route the packet from one source to destination with the label of flow label field in the packet.
- The simulator can log any information required by user to show the detail of the process during the simulation.
- Based on the simulation result, a more efficient quality of service of network should obtain.

5.6 Summary

This chapter covers the major design issues for the implementation of IPv6 Flow Label in UMJaNetSim. This includes an overview of the system architecture which focuses on the two main components, which are IPv6 router and IPv6 TCP. The class design gives an illustration of what those classes do.

Chapter 6 System Implementation

This chapter details the implementation of the IPv6 flow label environment and the implementation of IPv6 Flow Label. It includes the implementation of the IPv6 flow label component classes. All the important attributes and methods are discussed.

6.1 Implementation

The following section discusses the implementation for each simulator components.

6.1.1 SimNetworkComp Class

The SimNetworkComp class is a generic network component that can perform communication on the network. It is inherited by IPv6Router and IPv6BTE.

Attributes

```
class SimNetworkComp extends SimComponent implements
java.io.Serializable
{
  private final int DEFAULT_PROCESSING_DELAY = 10; // unit: us
  private final int DEFAULT_QUEUE_SIZE = 1024; // unit: bytes

  // to be overriden by BTE/Router
  protected boolean isRouter = false;
  protected boolean isMobile = false;

  protected boolean init = true;
  protected java.util.List NDProcessor = null;
  protected java.util.List outLink = null;
  protected SimParamInt myQueueSize = null;
  protected SimParamInt myProcessingDelay = null;
}
```

Link Management

```
private void addLink(IPv6Link comp)      //to add a link to link table
                                          //of this component
private void removeLink(IPv6Link comp)    //to remove a link from link
                                          //table of this component
```

Neighbor Discovery Processor Management

```
private void addNDProcessor(NeighborDiscoveryProcessor comp)   //to
//add a NDProcessor for a link to this component
private void removeNDProcessor(NeighborDiscoveryProcessor comp)
//to remove a NDProcessor for a link from this component
NeighborDiscoveryProcessor[] getNDProcessors()   //get a list of all
NDProcessor of this component
void startAllNDProcessor()   //kick start all NDProcessor of this
component
NeighborDiscoveryProcessor findNDProcessor(IPv6Address IP)   //find
NDProcessor with matching subnet of the given IP
NeighborDiscoveryProcessor findNDwithInterfaceID(int ID)   //find
NDProcessor with given interface ID
NeighborDiscoveryProcessor findNDwithLink(IPv6Link link)   //find
NDProcessor with given link
```

6.1.2 IPv6BTE Class

IPv6BTE inherits the SimNetworkComp, act as the Broadband Terminal Equipment (B-TE) in the IPv6 environment.

Attributes

```
class IPv6BTE extends SimNetworkComp implements Serializable {
```

```
  private final int BTE_PACKET_COLOR = 0x008080;

  private SimParamInterfaceTable myInterfaceID = null;
  private SimParamIPv6Address myPingIP = null;
  private SimParamBool myBindingCacheSupport = null;
  private SimParamHomeAddressTable myHAtable=null;

  protected SimComponent outAPP = null;
  protected ICMPv6Processor myICMPProcessor = null;
}
```

Configuration

```
IPv6Address getAssignedIP()    //get component IP
IPv6Address getTerminalIP()    //get component IP, same as
getAssignedIP()
IPv6Address getCurrentSubnetMask()    //get subnet mask where this
                                      component is reside
```

6.1.3 IPv6Router Class

IPv6 Router inherits the SimNetworkComp, act as a generic router in the IPv6 environment. It is responsible to implement the weighted round-robin queue to the IPv6 flow.

Attributes

```
class IPv6Router extends SimNetworkComp implements Serializable {
  private SimParamInterfaceTable myITable=null;
```

```
protected ICMPv6Processor myICMPProcessor = null;
protected RIPv6Processor myRIPv6Processor = null;
protected SimParamIPv6RTable myRTable=null;
protected SimParamInt myPacketHandled  = null;
protected SimParamInt myMemoryUsage  = null;
```

Packet Processing

```
protected void process(int nextLevelProtocol, IPv6Packet
receivedPacket, IPv6Link receiverLink)   //process the payload when
the packet is sent to the router
protected int processHeader(int headerType, Object header, IPv6Packet
receivedPacket, IPv6Link receiverLink)   //process the headers when
the packet is sent to the router
SimParamIPv6RTable getRoutingTable()   //get the current routing
                                       //table
```

Weighted Round-Robin Queue Scheme

```
private void sw_demux_spq(Port voport)       //enqueue the IPv6 flow to the
                                             //correspondence flow queue
private void sw_schedule_output(Port voport) //output the packet in the flow queue
                                             //according to the flow's weight
```

6.1.4 IPv6TCP Class

IPv6TDP inherits the SimComponent, which acts as the source of the IPv6 traffic. The flow label type can be set here.

Attributes

```
class IPv6UDP extends SimComponent implements Serializable {
```

```
class token implements Serializable
{
  int seqNumber = 0;
  boolean ack = false;
}

private final int maxTokenQueueSize = 20;
private int seqNumberNow = -1;
private java.util.List tokenQueue = null;
private SimComponent outBTE;
private SimParamIPv6Address peerIP=null;
private SimParamInt myTCPStartTime  = null;
private SimParamInt myTCPSendInterval  = null;
private SimParamInt myPacketMissed  = null;
private SimParamDouble myPacketLatency  = null;
 private SimParamInt totalToSend=null;
 int packetToSend =500;
 private SimParamInt packetSent = null;
 private SimParamInt packetReceived = null;
 private SimParamIntTag cn_ds_class;    //Flow Label type
 int ds = 0;                //int that represent flow label type
 int totalSent = 0;
 int totalReceived = 0;
}
```

Token Queue Management

```
private void updateTokenQueue(int seqNumber)   //add a new token into
the queue
private void ackTokenQueue(int seqNumber)   //acknowledge an existing
token in the queue
```

6.1.5 IPv6Link Class

IPv6Link inherits the SimComponent, which acts as a generic link for IPv6 network.

Attributes

```
class IPv6Link extends SimComponent implements Serializable {
  private final int MAXPACKET = 10;
  private final int DEFAULT_SPEED = 1000; // unit: Kbps
  private final int DEFAULT_DISTANCE = 1000; // unit: meter
  private final int DEFAULT_SWITCHING_DELAY = 10; // unit: us
  private final int DEFAULT_NOTIFYING_DELAY = 3; // unit: us
  private final int DEFAULT_CHANNEL = 1;
  private final int NUM_OF_STEPS = 5;

  private int busy = 0; // indicate whether the link is busy
  private int packetHandled = 0;
  private SimParamBool myAnimate = null;
  private SimParamDouble myMessageOverhead = null;
  private SimParamDouble myChannelUtilization = null;

  protected java.util.List packets = null;
  protected SimParamInt mySpeed = null;
  protected SimParamInt myDistance = null;
  protected SimParamInt mySwitchingDelay = null;
  protected SimParamInt mySupportedChannel = null;
}
```

Configuration

```
int getLinkID()    //get interface ID of this link
```

```
int findInterfaceID(int ID)   //find which neighbor is having this
interface ID
boolean isBusy()   //determine if all channel on this link is busy
void setBusy(boolean status)   //utilize a new channel and increase
the busy count by 1
```

6.1.6 IPv6StaticLink Class

IPv6StaticLink inherits the IPv6Link, which acts as a static link for IPv6 network.

Attributes

```
class IPv6StaticLink extends IPv6Link implements Serializable {
  private final int DEFAULT_SPEED = 10000; // unit: Kbps
  private final int DEFAULT_DISTANCE = 100; // unit: meter
  private final int DEFAULT_SWITCHING_DELAY = 10; // unit: us
  private final int DEFAULT_WIRED_CHANNEL = 1;
}
```

6.2 Summary

The Implementation of the important objects and classes has been discussed in this chapter. The attributes and configurations of all those objects and classes also covered throughout the chapter.

After the implementation of the system, it is time to enter to another important phase, that is system testing. The following chapter will cover all the system testing for the implemented objects and classes.

Chapter 7 System Testing

This chapter discusses the testing phases that need to be done for the simulator. Testing of the components and protocols implemented is very important before running a simulation. Failure to do this would mean that the simulations might have to be rerun if problems unveiled later.

Simulator testing is done in two parts. First, the IPv6 flow label environment will be tested. Finally, it will focus on the testing for IPv6 flow label.

7.1 Neighbor Discovery and Stateless Autoconfiguration

Overview

Neighbor Discovery and Stateless Autoconfiguration are implemented to provide a basic simulation environment for IPv6. The routers will advertise the prefixes to each attached links. The BTEs will capture the receive prefix and configure their interface addresses.

Topology Used

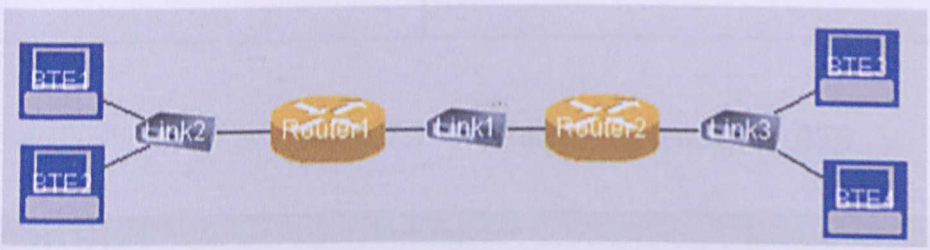


Figure 7.1 Topology for Neighbor Discovery Protocol Testing

7.1.1 Parameter Settings and Configurations

Table 7.1 Parameter setting for Router1

Parameter	Value
EUI-64 identifiers for interface to Link1	0000:0000:4875:95D8
Prefix for interface to Link1	00F1:0000:0000:0000:0000:0000:0000:0000/64
EUI-64 identifiers for interface to Link2	0000:0000:01BA:8040
Prefix for interface to Link2	00F2:0000:0000:0000:0000:0000:0000:0000/64

Table 7.2 Parameter setting for Router2

Parameter	Value
EUI-64 identifiers for interface to Link1	0000:0000:4CD5:F52F
Prefix for interface to Link1	00F1:0000:0000:0000:0000:0000:0000:0000/64
EUI-64 identifiers for interface to Link3	0000:0000:478A:36FF
Prefix for interface to Link2	00F3:0000:0000:0000:0000:0000:0000:0000/64

Table 7.3 Parameter setting for BTE

BTE	Parameter	Value
BTE1	EUI-64 identifiers for interface to Link2	0000:0000:3FA1:9DB0
BTE2	EUI-64 identifiers for interface to Link2	0000:0000:3B41:3E58
BTE3	EUI-64 identifiers for interface to Link3	0000:0000:36C1:9AA0

BTE4	EUI-64 identifiers for interface to Link3	0000:0000:3261:3B47

7.1.2 Simulation Results

Table 7.4 IPv6 addresses for BTEs after simulation

BTE	Parameter	Value
BTE1	IPv6 Address for Link2	00F2:0000:0000:0000:0000:0000:3FA1:9DB0
BTE2	IPv6 Address for Link2	00F2:0000:0000:0000:0000:0000:3B41:3E58
BTE3	IPv6 Address for Link3	00F3:0000:0000:0000:0000:0000:36C1:9AA0
BTE4	IPv6 Address for Link3	00F3:0000:0000:0000:0000:0000:3261:3B47

The BTEs obtain the prefixes information from their directly attached router, performing autoconfiguration.

7.2 RIPng

Overview

Routing Information Protocol next generation (RIPng) is implemented in the thesis as the unicast routing protocol for IPv6. The routers will advertise RIPng messages periodically and some triggered updates to build the route table.

Topology Used

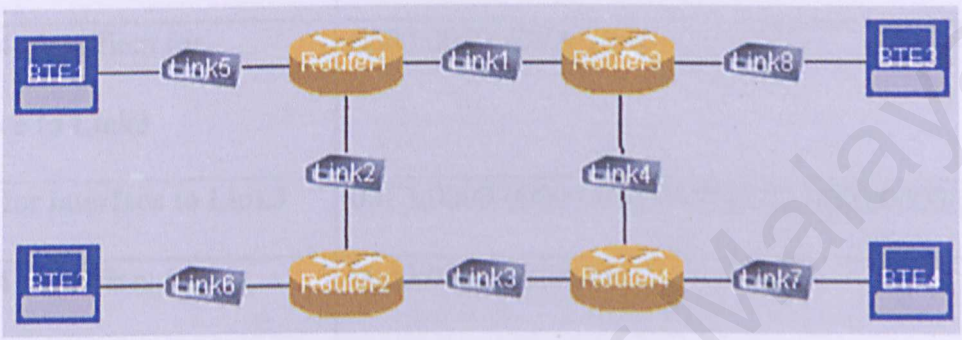


Figure 7.2 Topology for RIPng Testing

7.2.1 Parameter Settings and Configurations

Table 7.5 Parameter setting for Router1

Parameter	Value
EUI-64 identifiers for interface to Link1	0000:0000:4875:95D8
Prefix for interface to Link1	00F1:0000:0000:0000:0000:0000:0000:0000/64
EUI-64 identifiers for interface to Link2	0000:0000:01BA:8040
Prefix for interface to Link2	00F2:0000:0000:0000:0000:0000:0000:0000/64
EUI-64 identifiers for interface to Link5	0000:0000:1FBA:3D78

Prefix for interface to Link5	00F5:0000:0000:0000:0000:0000:0000:0000/64

Table 7.6 Parameter setting for Router2

Parameter	Value
EUI-64 identifiers for interface to Link2	0000:0000:02A5:DF18
Prefix for interface to Link2	00F2:0000:0000:0000:0000:0000:0000:0000/64
EUI-64 identifiers for interface to Link3	0000:0000:478A:36FF
Prefix for interface to Link3	00F3:0000:0000:0000:0000:0000:0000:0000/64
EUI-64 identifiers for interface to Link6	0000:0000:261A:7944
Prefix for interface to Link6	00F6:0000:0000:0000:0000:0000:0000:0000/64

Table 7.7 Parameter setting for Router3

Parameter	Value
EUI-64 identifiers for interface to Link1	0000:0000:5136:5486
Prefix for interface to Link1	00F1:0000:0000:0000:0000:0000:0000:0000/64
EUI-64 identifiers for interface to Link4	0000:0000:72A6:1244
Prefix for interface to Link4	00F4:0000:0000:0000:0000:0000:0000:0000/64
EUI-64 identifiers for interface to Link8	0000:0000:49E5:B9EC

Prefix for interface to Link8	00F8:0000:0000:0000:0000:0000:0000:0000/64

Table 7.8 Parameter setting for Router4

Parameter	Value
EUI-64 identifiers for interface to Link3	0000:0000:3EC9:784D
Prefix for interface to Link3	00F3:0000:0000:0000:0000:0000:0000:0000/64
EUI-64 identifiers for interface to Link4	0000:0000:7706:719E
Prefix for interface to Link4	00F4:0000:0000:0000:0000:0000:0000:0000/64
EUI-64 identifiers for interface to Link7	0000:0000:6789:D0A1
Prefix for interface to Link7	00F7:0000:0000:0000:0000:0000:0000:0000/64

Table 7.9 Parameter setting for BTE

BTE	Parameter	Value
BTE1	EUI-64 identifiers for interface to Link5	0000:0000:3F43:D096
BTE2	EUI-64 identifiers for interface to Link6	0000:0000:3AC4:2CDC
BTE3	EUI-64 identifiers for interface to Link8	0000:0000:3625:44C0
BTE4	EUI-64 identifiers for interface to Link7	0000:0000:31E4:29C3

7.2.2 Simulation Results

Table 7.10 Route Table for Router1

Destination Network	Link Cost	Next Hop Address
00F1:0000:0000:0000:0000:0000:0000:0000	1	Direct link
00F2:0000:0000:0000:0000:0000:0000:0000	1	Direct link
00F5:0000:0000:0000:0000:0000:0000:0000	1	Direct link
00F4:0000:0000:0000:0000:0000:0000:0000	2	00F1:0000:0000:0000:0000:0000:5136:5486
00F7:0000:0000:0000:0000:0000:0000:0000	3	00F2:0000:0000:0000:0000:0000:02A5:DF18
00F3:0000:0000:0000:0000:0000:0000:0000	2	00F2:0000:0000:0000:0000:0000:02A5:DF18
00F8:0000:0000:0000:0000:0000:0000:0000	2	00F1:0000:0000:0000:0000:0000:5136:5486
00F6:0000:0000:0000:0000:0000:0000:0000	2	00F2:0000:0000:0000:0000:0000:02A5:DF18

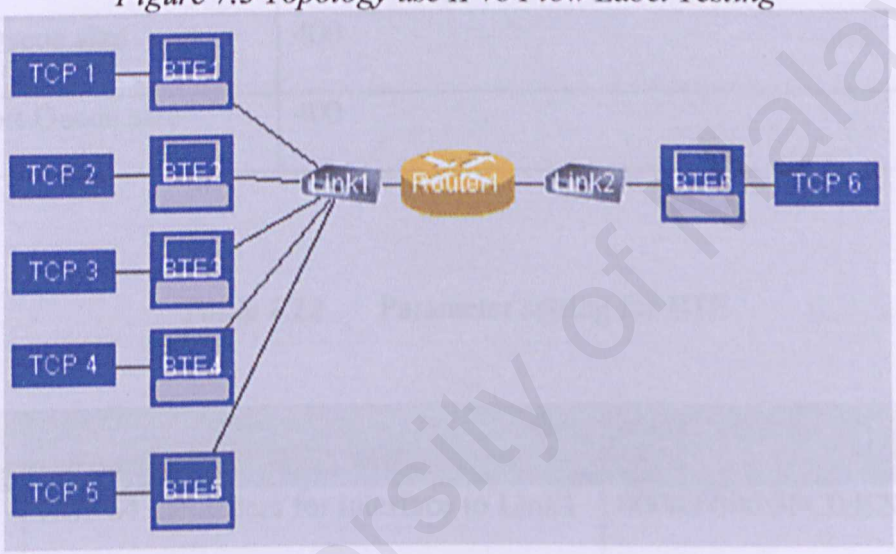
The table above only shows part of the information in the route table. The correctness of the route table information in the Router1 justified that the RIPng is successfully implemented. Route tables for the other routers are not shown here to simplify the discussions.

7.3 IPv6 Flow Label

IPv6 Flow Label is implemented in the simulator to provide a better QOS for network. In the implemented environment, each traffic source will belong to one flow type. The router will put the receive packets into the correspondence flow queue and output those packets according to their weight. The weight of each flow can be set in the router.

Topology Used

Figure 7.3 Topology use IPv6 Flow Label Testing



7.3.1 Parameter Settings and Configurations

Table 7.11 Parameter setting for Router1

Parameter	Value
EUI-64 identifiers for interface to Link1	0000:0000:4875:95D8
Prefix for interface to Link1	00F1:0000:0000:0000:0000:0000:0000:0000/64
EUI-64 identifiers for interface to Link2	0000:0000:01BA:8040

Prefix for interface to Link2	00F2:0000:0000:0000:0000:0000:0000:0000/64
WRR weight for Flow. 1	100
WRR weight for Flow. 2	70
WRR weight for Flow. 3	30
WRR weight for Flow. 4	10
Flow.1 Queue size	400
Flow.2 Queue size	400
Flow.3 Queue size	400
Flow.4 Queue size	400
Best Effort Queue size	400

Table 7.12 Parameter setting for BTE

BTE	Parameter	Value
BTE1	EUI-64 identifiers for interface to Link1	0000:0000:3FC0:E20E
BTE2	EUI-64 identifiers for interface to Link1	0000:0000:3B60:82B7
BTE3	EUI-64 identifiers for interface to Link1	0000:0000:3700:2360
BTE4	EUI-64 identifiers for interface to Link1	0000:0000:329F:C409
BTE5	EUI-64 identifiers for interface to Link1	0000:0000:2E3F:64B2
BTE6	EUI-64 identifiers for interface to Link2	0000:0000:29BF:C0F8

Table 7.13 Parameter setting for TCP

TCP	Peer IP	Flow Label Class	Packets To Send
TCP1	F2:0:0:0:0:0:29BF:C0F8	Flow. 1	500

TCP2	F2:0:0:0:0:0:29BF:C0F8	Flow. 2	500
TCP3	F2:0:0:0:0:0:29BF:C0F8	Flow. 3	500
TCP4	F2:0:0:0:0:0:29BF:C0F8	Flow. 4	500
TCP5	F2:0:0:0:0:0:29BF:C0F8	Best Effort	500
TCP6	0:0:0:0:0:0:0:0	--	--

In order to show the packets in the correspondent flow queue, the output link's speed needs to be very slow.

Table 7.14 Parameter setting for Link

Link	Speed (kbps)
Link1	100
Link2	5

7.3.2 Simulation Results

Table 7.15 Packets and Dropped Packets in each Flow Queue

Time(s)	*0*	*15*	*30*	*45*	*60*	*75*	*90*	*105*	*120*	*135*	*150*	*165*	*180*	*195*
Packets in Flow.1Queue	0	70	191	297	299	283	199	99	0	0	0	0	0	0
Packets Dropped in Flow.1 Queue	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Packets in Flow.2Queue	0	140	226	370	371	330	260	203	169	0	0	0	0	0
Packets Dropped in Flow.2Queue	0	0	0	0	29	29	29	29	29	29	29	29	29	29
Packets in Flow.3Queue	0	140	291	400	400	370	359	340	310	238	46	0	0	0
Packets Dropped in Flow.3Queue	0	0	0	10	69	69	69	69	69	69	69	69	69	69
Packets in Flow.4Queue	0	131	282	400	400	390	390	380	370	350	290	20	0	0
Packets Dropped in Flow.4Queue	0	0	0	21	80	80	80	80	80	80	80	80	80	80
Packets in BE Queue	0	141	292	400	400	400	400	400	400	400	400	400	186	0
Packets Dropped in BE Queue	0	0	0	41	100	100	100	100	100	100	100	100	100	100

The above table shows the packet queue for each flow. In the beginning, all the queues are empty. After the simulator run, the packets begin to enqueue in the correspondence flow's queue. Because flow 1 has the largest weight, follow by flow 2, flow 3 and flow 4, flow 1 always has the highest priority to send out the packet. Thus, it always has fewest packets in its queue follow by the subsequent flow queues.

In the other hand, Best Effort queue will only send out its packets when all the other queues are empty. It has the lowest priority to send out the packet.

The packets in the queue will drop if it exceeds the queue's size. Because the Best Effort flow has the lowest priority to send out, it has the highest dropped packets. These follow by flow 4, flow 3, flow 2 and flow 1.

7.4 Summary

In this chapter, all the testing of the components and protocols implemented had been explained. According to the testing results, all the implemented components and protocol function properly. The third testing which is testing to the implementation of IPv6 Flow Label show that a differentiate service had been provide by flow label, and this can provide a better quality of service for computer network.

Chapter 8 Conclusion and Future Works

In summary, the thesis presented the usage of IPv6 Flow Label to provide a better quality of service to the network.

In the beginning of the thesis, some researches regarding the technologies of the quality of service of computer network have been studied. The thesis continues with the studies of the IPv6 flow label. It reveals some proposed IPv6 flow label specification. Then, the analysis and design of the system have been discussed. Finally, the thesis had included the implementation and testing of the system.

8.1 Summary of Contribution

IPv6 architecture, IPv6 addressing and fundamental IPv6 protocols i.e. Neighbor Discovery Protocol, RIPng and IPv6 flow label has been explored and studied.

A simulation environment for IPv6 flow label has been built and tested. IPv6 Flow label simulation components i.e. IPv6 Link, IPv6 BTE and IPv6 TCP have been incorporated into existing UMJaNetSim simulator.

The IPv6 router use weighted Round-Robin scheduling mechanism to schedule the output of the packets in the flow queues. A differentiate quality of service can be provide to different flow according to their flow label value. Thus, the conclusion is that IPv6 flow label has success to provide a better quality of service to the computer network.

8.2 Suggestion for Future Research

There are still many unavoidable issues related to this work that require further research. The graphs generated by the simulator for the dropped packets in the flow queue are not precise enough and hard to do comparison among flow queue. Future works can be involve by implementing a more precise graph for the dropped packets and a combine graph for all the dropped packets in all the flow queues, so that a comparison among flow queues can be conduct more easier.

Future research on the IPv6 flow label may use the Open Shortest-Path First (OSPF) for IPv6 as the underlying unicast routing protocol. The implementation of OSPF for IPv6 will allows simulations of a bigger network topology. The simulations running for the thesis are limited to the scope defined by the RIPng.

REFERENCES

[BAY, 1997] Bay Networks (1997). *White Paper: IPv6.* Bay Networks, July 1997. [Online] Available from: <http://www.cs-ipv6.lancs.ac.uk/ipv6/documents/papers/BayNetworks/index.html> [Accessed 23 August, 2003]

[Blake et al, 1998] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss(1998). *An Architecture for Differentiated Service.* RFC 2475. December 1998.

[Braden et al, 1997] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, S. Jamin. *Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification.* RFC 2205. September 1997.

[Bratley, 1987] P. Bratley, B. L. Fox, L. E. Schrage, A Guide to Simulation, 2nd Ed., Springer-Verlag, 1987.

[Breslau et al, 2000] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, H. Yu, "Advances in network simulation", IEEE Computer Magazine, Vol. 33, pp. 59-67, May 2000.

[Conta&Deering, 1998] Conta, A. and Deering, S. (1995). Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. RFC 2463, December 1998.

[Deering&Hinden, 1998a] Deering, S. and Hinden, R. (1998). *Internet Protocol, Version 6 (IPv6) Specification.* RFC 2460, December 1998.

[Deering&Hinden, 1998b] Hinden, R. and Deering, S. (1998). *IP version 6 Addressing Architecture.* RFC 2373, July 1998.

[DRCL, 2001] DRCL, "The Autonomous Component Architecture", The Ohio State University, Jan. 2001.

[Drom, 1997] Droms, R. (1997). *Dynamic Host Configuration Protocol.* RFC 2131, March 1997.

[Fuller, 1993] Fuller, V. et al. (1993). *Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy.* RFC 1519, September 1993.

[Golmie et al, 1998] N. Golmie, F. Mouveaux, L. Hester, Y. Saintillan, A. Koenig, D. Su, The NIST ATM/HFC Network Simulator: Operation and Programming Guide, High-Speed Networks Technologies Group, NIST, US Dept. of Commerce, Dec. 1998.

[Heinanen et al, 1999] J. Heinanen, F. Baker, W. Weiss, J. Wroclawski(1999). *Assured Forwarding PHB Group.* RFC 2597, June 1999.

[Jacobson et al, 1999] V. Jacobson, K. Nichols, K. Poduri(1999). *An Expedited Forwarding PHB.* RFC 2598. June 1999.

[Malkin&Minnear, 1997] Malkin, G. and Minnear R. (1997). RIPng for IPv6. RFC 2080, January 1997.

[Microsoft, 2000] Microsoft Corporation (2000). *Introduction to IPv6.* Microsoft Corporation, June 2000.

[Narten et al, 1998] Narten, T. et al. (1998). *Neighbor Discovery for IP version 6 (IPv6).* RFC 2461, December 1998.

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

[Nance, 1993] R. E. Nance, "A History of Discrete Event Simulation Programming Languages", Proc. of the 2nd ACM SIGPLAN History of Programming Lang. Conf., Reprinted in ACM SIGPLAN Notices, 28(3), pp. 149-175, Apr. 1993.

[Overstreet, 1982] C. M. Overstreet, "Model Specification and Analysis for Discrete Event Simulation", PhD Dissertation, Dept. of Comp. Sc., Virginia Tech, Dec. 1982.

[Rosen et al, 2001] E. Rosen, A. Viswanathan, R. Callon(2001). *Multiprotocol Label Switching Architecture.* RFC 3031. January 2001.

[Shenker et al, 1997] S. Shenker, C. Partridge, R. Guerin(1997). Specification of Guaranteed Quality of Service. RFC 2212. September 1997.

[Smith, 2000] R. D. Smith, Simulation Article, *Encyclopedia of Computer Science*, 4th Ed., Grove's Dictionaries Inc., July 2000.

[Thompson&Narten, 1998] Thompson, S. and Narten, T. (1998). *IPv6 Stateless Address Autoconfiguration.* RFC 2462, December 1998.

[UCLA, 1999] PARSEC User Manual Release 1.1, UCLA Parallel Computing Lab, Sep. 1999.

[UCLA, 1995] Maisie User Manual Release 2.2, UCLA Parallel Computing Lab, Dec. 1995.

[VINT 2003] The ns Manual, The VINT Project, July 2003.

[Wroclawski, 1997a] J. Wroclawski(1997). *The Use of RSVP with IETF Integrated Services.* RFC 2210. September 1997.

[Wroclawski, 1997b] J. Wroclawski(1997). *Specification of the Controlled-Load Network Element Service.* RFC 2211. September 1997.