Scalable QoS and QoS Management Models for IP Networks

by

El Bahlul Fgee

Submitted in partial fulfillment of the
requirement for the degree of

DOCTOR OF PHILOSPHY

Major Subject: Engineering Mathematics and Internetworking

at

DALHUSIE UNIVERSITY

Halifax, Nova Scotia November, 2005

Your file Votre référence
ISBN: 0-494-13048-2
Our file Notre référence
ISBN: 0-494-13048-2

DALHOUSIE UNIVERSITY

To comply with the Canadian Privacy Act the National Library of Canada has requested that the following pages be removed from this copy of the thesis:

Preliminary Pages
- Examiners Signature Page
- Dalhousie Library Copyright Agreement

Appendices
- Copyright Releases (if applicable)

I dedicate this great

piece of work

To Suhaib, Hidaya and Alaa

In memory of my beloved parents!

Contents

List of Tables x

List of Figures xii

List of Acronyms xv

Acknowledgements xviii

Abstract xix

1 Introduction 1

1.1 Motivation 1

1.2 Problem Definition 3

1.2.1 Definition of QoS 4

1.2.2 QoS Parameters 5

1.2.3 QoS Control Mechanisms 6

1.3 State of the Art 8

1.3.1 QoS Approaches 8

1.3.2 Internet Protocol IP 10

1.4 The proposal 11

1.4.1 Proposed end-to-end QoS Model 11

1.4.2 The Proposed QoS Manager 13

1.5 Document Organization 14

2 Related Work 16
2.1 An Overview of QoS Control . . . 16
2.1.1 Scheduler and Buffer Management . . . 17
2.1.2 QoS Models . . . 17
2.1.3 Research Efforts on QoS Model . . . 22
2.2 Network Resource Management . . . 25
2.2.1 Bandwidth Broker Architecture . . . 25
2.2.2 Dynamic Reservation . . . 29
2.2.3 Pricing-based Approaches . . . 32

3 Mathematical Modeling 34
3.1 Introduction . . . 34
3.2 Definition of Basic Network Calculus Elements Used . . . 35
3.2.1 Arrival and Service Curves . . . 35
3.2.2 The Fundamental Bounds . . . 37
3.2.3 Packetization Effects . . . 40
3.2.4 Calculated Delay Bounds for Some Schedulers . . . 40
3.3 QoS Mathematical Service Element Model . . . 43
3.3.1 Model Layout . . . 43
3.3.2 The IntServ Model . . . 44
3.3.3 The DiffServ Model . . . 45
3.3.4 The Proposed Model . . . 46
3.4 Comparison of Models . . . 47
3.4.1 Comparison . . . 47
3.4.2 Numerical Example . . . 49

4 An end-to-end QoS Management system proposal 54
4.1 Introduction . . . 54
4.2 Proposed QoS Manager . . . 56
4.3 IP Edge Router . . . 61

4.3.1 Architecture for the IP Edge Router Structure:- 61
4.4 Reservation Scenarios 63
4.4.1 Local Reservation 64
4.4.2 Domains Reservation 67

5 Implementation and Experiments **70**
5.1 Choice of queue simulations 71
5.1.1 Simulated Results Analysis 72
5.2 The Implementation of the IPv6 QoS management scheme: 73
5.2.1 The Model Layout 73
5.2.2 NS-2 Code Implementation 76
5.2.3 Simulation Setup and Results 78
5.3 Simulation of Other QoS Models 86
5.3.1 RSVP Simulation Results 86
5.3.2 MPLS Simulation Results:- 90
5.3.3 DiffServ Simulation Results: 94
5.3.4 Simulation Results Comparison between the Proposed IPv6 QoS Manager and other QoS Techniques 100
5.3.5 Average end-to-end delay 100
5.3.6 Average Packet Loss Rate 102
5.4 Simulation of two and three QoS domains using IPv6 QoS management models 106
5.4.1 Two Domains Simulation 106
5.4.2 Three Domains Simulation 109

6 Implementation of Internet billing on the IPv6 QoS manager **112**
6.1 Introduction and Motivation:- 112
6.2 Review of the Internet Pricing Schemes 113
6.2.1 Flat Pricing [58] 113
6.2.2 Priority Pricing [58] 114

6.2.3 Smart-Market Pricing [82] 115
6.2.4 Edge Pricing [83] 115
6.2.5 Per-Packet pricing scheme [59] 116
6.2.6 Dynamic Pricing model[60] [61] 116
6.3 The Integration of the Proposed IPv6 Manager with the Dynamic Pricing Model 119
6.3.1 Introduction: 119
6.3.2 Implementation and Deployment 120
6.3.3 Simulation Results 121

7 Conclusions and Future Work **126**
7.1 Thesis Summary 126
7.1.1 Mathematical Modeling 127
7.1.2 Model Architecture 127
7.1.3 Model Implementation and Simulation 128
7.1.4 Integrating Dynamic Pricing with the QoS Model 129
7.2 Future Directions 129

Bibliography **131**

A QoS Regulators and Schedulers **141**
A.1 Traffic Regulators 141
A.2 Traffic Schedulers 142
A.2.1 Weighted Fair Queuing : 143
A.2.2 Class-Based Queuing : 146

B QoS Approaches used in Literature **148**
B.1 Integrated Services (IntServ) and RSVP 148
B.1.1 QoS Classes in IntServ 149
B.1.2 IntServ Traffic Control Model 149
B.1.3 RSVP: a signaling protocol for IntServ 151

B.2 Differentiated Service Architecture 154
B.2.1 DiffServ Routers 156
B.2.2 Resource allocation in DiffServ Domain using BB: 158

C Next Generation Internet Protocol IPv6 **161**
C.1 Introduction 161
C.2 Some Aspects of IPv6 162
C.2.1 Addressing 162
C.2.2 Performance 162
C.2.3 Network service 163
C.2.4 Addressing Flexibility 163
C.2.5 Security Capabilities 164
C.3 QoS in IPv6 164
C.3.1 Flow Label 164
C.3.2 Priority Field (Traffic Class) 167

D ns-2 added code **169**
D.1 the C++ Code implementation 169
D.1.1 QoS Manager Code 169
D.1.2 IP Edge Router Code 195
D.2 TCL Code Scripets 206
D.2.1 QoS Manager Simulation Code 206
D.2.2 Pricing Integration Simulation Code 213

List of Tables

1.1 Heterogeneous traffic behavior and QoS requirements of Internet applications 3

3.1 End-to-end delayed calculation using the three QoS models 53

5.1 Average End-to-end delay during normal rates comparison 72
5.2 Average End-to-end delay during source 2 rate increase comparison . 72
5.3 Packet Loss Rate comparison 73
5.4 End-to-end delay, packet loss rate and degradation rate comparison results for three traffic sources during violation and non-violation scenarios 81
5.5 End-to-end delay, packet loss rate results for six traffic sources simulation scenario 86
5.6 RSVP simulation results for End-to-end delay and packet loss rate . . 89
5.7 MPLS simulation results for End-to-end delay and packet loss rate . . 90
5.8 MPLS constraint routing simulation results for End-to-end delay and packet loss rate 93
5.9 DiffServ simulation results using Token Bucket policer for End-to-end delay and packet loss rate 95
5.10 Jobs simulation results testing QoS parameters, End-to-end delay and packet loss rate 99
5.11 Simulation results comparison between the IPv6 model and other QoS methods 101

5.12 Two domains simulation results testing QoS parameters, End-to-end delay and packet loss rate 109
5.13 Three domains simulation results testing QoS parameters, End-to-end delay and packet loss rate 110

B.1 IntServ and DiffServ architecture comparison 160

C.1 The Priority Values 167

List of Figures

1.1 QoS Management cycle 8
1.2 Thesis Organization 14

2.1 Differentiated Service Domain architecture 20
2.2 MPLS Packet Format 21
2.3 MPLS architecture 22
2.4 BB basic components 27

3.1 Leaky bucket and IETF IntServ Arrival Curves 36
3.2 Service Curve 37
3.3 An example of traffic flow bounded by $\alpha(t)$ and $\beta(t)$ 38
3.4 Delay and Backlog bounds for token bucket model 39
3.5 Delay and Backlog bounds for IETF IntServ model 40
3.6 FIFO and WFQ max delay and backlog 42
3.7 Block Diagram for the Network model 44
3.8 DiffServ maximum delay calculation 46

4.1 Proposed QoS management scheme structure 57
4.2 Proposed IPv6 Edge router structure 62
4.3 Procedures for requesting QoS for a node connected to an edge router 64
4.4 Procedures for requesting QoS for a node connected to a leaf router . 67
4.5 Procedures for requesting QoS for a receiving node located at different domain 68

5.1 NS-2 network layout used for testing IPv6 QoS 71
5.2 Proposed IPv6 QoS management scheme implementation on ns-2 . . 74
5.3 Testing the proposed scheme by simulating a simple network 79
5.4 Sample output run showing Acceptance messages and some statistical results showing flow ID, total packets, dropped packets and degraded packets 80
5.5 Delay for the tested three traffic flows during Source 2 non-conformant traffic flow packets 82
5.6 Delay for the tested three traffic flows when all the three flows are conformant 83
5.7 Six sources network topology 84
5.8 End-to-end delay for Six sources simulation 85
5.9 Network topology for RSVP simulation 87
5.10 PATH and RES messages printed out during RSVP simulation 88
5.11 End-to-end delay for three traffic flows during flow ID 200 non-conformant when RSVP QoS protocol is simulated 89
5.12 End-to-end delay for the tested three traffic flows during flow ID 200 rate increase and the involvement of MPLS 91
5.13 Printed out messages showing the explicit routes for each flow during MPLS-constraint routing simulation 92
5.14 Delay for the tested three traffic flows when MPLS-constraint routing is simulated 94
5.15 Delay for the tested three traffic flows for DiffServ simulation using Token Bucket policer 95
5.16 Delay for the tested three traffic flows for QoSbox simulation 97
5.17 Simulation messages during jobs simulation 98
5.18 Simulated two domain network topology 107
5.19 Delay for the tested three traffic flows for two domain simulation . . . 108
5.20 Simulated three domain network topology 111

5.21 Average delay for the three traffic flows 111

6.1 General Pricing strategy 117
6.2 Flow chart for the IPv6 pricing model 121
6.3 Simulated pricing model network 123
6.4 Traffic flow 15 Prices during network congestion and low traffic load . 124
6.5 Traffic flow 12 Prices during high traffic and low traffic load 124
6.6 Prices for the three traffic flows FID(15) & FID(12) and FID(8) . . . 125

A.1 Leaky Bucket Model 142
A.2 Basic structure of a QoS capable router 143
A.3 WFQ 145
A.4 CBQ 147

B.1 IntServ Control Model 150
B.2 RSVP setup Procedures 153
B.3 Conceptual Model of a DiffServ Router 156
B.4 DiffServ core and edge routers structures 157
B.5 A generic model for Differentiated Service Domain 159

List of Acronyms

ACA	IPv6 QoS manager Admission Control Agent
AF	Assured Forwarding
ARMBB	Active Resource Management BB
ATM	Asynchronous Transfer Mode
BA	Behavior Aggregate
BB	Bandwidth Broker
CBQ	Class Based Queuing
CBR	Constant Bit Rate
CIR	Committed Information Rate
DiffServ	Differentiated Service
DSCP	Differentiated Service Code Point
EDP	Early Packet Discard
EF	Expedited Forwarding
FEC	Forward Equivalence Class
FCFS	First Come First Serve Queue
GDI	IPv6 Global Domain Identifier
IntServ	Integrated Service
IETF	Internet Engineering Task Force
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
LSP	Label Switching Path

LSR	Label Switching Router
MRA	IPv6 QoS manager Management Reservation Agent
$MPLS$	Multi-Protocol Label Switching
NS	Network Simulator
PHB	DiffServ Per Hop Behavior
PPB	IPv6 Edge router Packet Processing Block
QoS	Quality of Service
$QoSBox$	QoS Box
RB	IPv6 Edge router Routing Block
RFA	IPv6 Edge router Request Forwarding Agent
RPA	IPv6 QoS manager Request Processing Agent
$RSVP$	Resource Reservation Protocol
SLA	Service Level Agreement
TC	IPv6 Traffic Class field
$TCDB$	IPv6 QoS manager Traffic Control Data Box
ToS	IPv4 Type of Service field
VLL	Virtual Leased Line
WFQ	Weighted Fair Queuing
$WFQ-CA$	IPv6 QoS manager WFQ Weight Calculation Agent
$5tuple$	TCP/IP 5 tuple. IP source address, IP Destination address, Source port number, Destination port number and Protocol version
$\alpha(t)$	Arrival curve
$\beta(t)$	Service Curve
T_{spec}	IntServ traffic specification
$R(t)$	Accumulated input traffic flow
$R^*(t)$	Output traffic flow
T_{lat}	Service curve latency delay
σ	Token bucket arrival curve burst tolerance

ρ	Token bucket arrival curve average rate.
M	IntServ arrival curve maximum packet size.
p	IntServ arrival curve peak rate
b	IntServ arrival curve burst tolerance.
r	IntServ arrival curve sustinable rate.
d_{max}	scheduler (node queue) maximum delay.
R	scheduler (node queue) rateor latency service curve rate.
L_{max}	Maximum packet size across a node.
L	Traffic flow's maximum packet size.
g	WFQ or CBQ guarantted rate.
δ_d	Router lookup time (packet processing time).
$D_{end2end}$	Flow end-to-end delay.
P^i_{base}	Class Service base price.
P_{max}	Class service maximum price.
f_i	Class service fill factor $(\frac{\text{Target capcity of a traffic class}}{\text{Flow maximum capacity}})$
$P_i(t)$	Dynamic price of a service class.

Acknowledgements

Pursuing a PhD is an amazing journey. Throughout its duration one meets a lot of interesting people that help finish this journey.

First and foremost I would like to thank my advisors, Dr. William Phillips and Dr. William Robertson, for giving me the opportunity to set off this amazing journey. They guided and supported me throughout the period of this journey. They always provide me with excellent feedback that made this thesis complete and interesting.

I thank Dr. Shyamala Sivakummar for her valuable ideas and suggestions that were used in this research.

I thank the thesis committee members: Dr. Otman Basir and Dr. Srinivas Sampalli for their time and effort in reviewing this thesis draft, and their valuable suggestions.

Special thanks to my friend Jason Kenney for his endless time, inspiration, incredible support and help.

Thanks to the Department of Engineering Mathematics and Internetworking Program for their academic assistance.

Thanks to all my friends who stood by me during this challenging journey.

I thank the administration of the Gharian High Polytechnic Institute, Libya for their support.

Above all I would like to express my deepest gratitude to people I love and to whom I dedicate this thesis: My parents, my wife and children, my brothers and sisters. I deeply believe that this thesis is a much theirs as it is mine.

Abstract

Significant challenge has been introduced for IP networks as new emerging applications, such as VOIP and E-commerce, require delivery quality guarantees and IP networks supports only best effort delivery. Three main Quality of Service (QoS) standards, Integrated Service (IntServ), Differentiated Service (DiffServ) and Multiple Label Switching Protocol (MPLS), have been introduced to handle QoS issues. These schemes use a management scheme to handle QoS requests, control bandwidth and monitor traffic flows.

A new QoS mathematical model is proposed in this thesis that guarantees requested QoS in which end-to-end delay and packet loss bounds are determined. This model is implemented at each domain then managed by a domain QoS manager. The proposed QoS manager establishes bandwidth reservation on intra and inters domain links for individual flows. In addition to resource reservation, two other essential resource control tasks within the model, admission control and traffic policing, are served. The proposed model and QoS manager retain the best features of IntServ and DiffServ however it has low complexity and provides end-to-end QoS (or per-flow QoS). The proposed new model has the following advantages:

- **Scalable:** complexity is left at the edge router and core routers forward packets only.
- **Manageable**: only one management system per back bone network is implemented to process QoS requests.
- The combination is defined in this thesis as the **Global Domain Identifier (GDI)** and used for forwarding and reservation decisions.
- No mapping either to DiffServ classes or MPLS formats.

As the next generation protocol, IPv6, is becoming important due to the growth of the Internet, the new model was simulated based on IPv6. The simulation demonstrates the model's performance with respect to the performance of DiffServ and IntServ, the predominate QoS methods in use today. IPv6 is intended to support QoS with a new field the Flow ID. The simulations implement the proposed new model using the Flow ID and traffic class fields to manage QoS in IPv6 domains.

The simulation results and the numerical experiments of the proposed QoS model show that a minimum end-to-end delay and minimum packet loss rate are achieved for higher priority traffic flows. In conclusion, a dynamic pricing system is integrated with the proposed QoS management system to test how prices can change when domain networks are congested. This pricing model uses the flow identifier, flow ID and the application source IP address, for calculating prices during network congestion. It also uses this identifier for charging customers. Excellent pricing and revenues results were achieved.

Chapter 1

Introduction

1.1 Motivation

Network multimedia applications, which form a large part of today's Internet traffic, present a significant challenge, as they require quality guarantees from the network. Internet Protocol (IP) however, is best effort and does not provide any guarantee for delivery as it has no network Quality of Service (QoS) [1]. That is, there are no mechanisms in IP for policing or controlling unresponsive and high bandwidth flows that can cause congestion in the network. IP best effort allows complexity to stay at the end hosts (sender and receiver applications control delivery) which scales well and allows the network to expand. As a result, all QoS management is left to the application [2]. However, service degrades gracefully and some applications can not tolerate such a degradation of service caused, for example, by delay and jitter. Multimedia applications have very limited feedback control to stop them from causing congestion in the network. Consequently, QoS management for network multimedia applications over IP is a significant and immediate challenge. To provide adequate service, some level of quantitative or qualitative determinism, the IP protocol's service must be supplemented. This requires adding some "smarts" to the network to distinguish traffic with strict timing requirements from those that can tolerate delay, jitter and loss. This is what Quality of Service (QoS) protocols are designed to accomplish.

QoS does not create bandwidth, but manages it so it is used more effectively to meet the wide range of application requirements. Therefore, the goal of QoS is to provide some level of predictability and control beyond the current IP "best effort" service [3].

Internet Scaling Challenges

As mentioned, more intelligence is needed in the Internet infrastructure to make it capable of providing different levels of performance assurance, e.g., guaranteed bandwidth and packet delivery within a delay bound. Nevertheless, any research or implementation efforts are faced with the following challenges:-

- **Scale in Number**

  The tremendous Internet growth over the past few years makes its scalability a crucial problem for network designers. The number of hosts in the Domain Name Service (DNS) has doubled compared to the 1990's. In the same time, the number of users online worldwide has also doubled. Therefore, the feasibility of a QoS mechanism depends on whether it can scale well under unpredictable growth.

- **Growth in Heterogeneity**

  The scalability concerns are multi-dimensional, not just an issue of "number". The Internet carries traffic generated from a variety of applications with different traffic characteristics and performance requirements. Users are connected via heterogeneous access networks, using devices that differ in capabilities. Table 1.1 compares the traffic behavior and QoS requirements between traditional data applications and emerging multimedia applications.

- **Distributed Internet Administration**

  The decentralized control of the Internet poses another technical challenge in providing end-to-end QoS. Privataization of the Internet introduces separate and independent administration over operational domains. These private organizations that operate the different networks and provide end users with access

Table 1.1: Heterogeneous traffic behavior and QoS requirements of Internet applications

Applications	Traffic Behavior	QoS Requirements
Electronic mail(SMTP) file transfer (FTP) remote terminal (Telnet)	Small batch file transfers	very tolerant of delay Bandwidth requirements:low Best effort
HTML web browsing	A series of small, bursty file transfer	Tolerant of moderate delay. Bandwidth requirements:Varies Best effort
Client-Server	Many small two-way transactions	Sensitive to loss & delay Bandwidth requirements:Moderate Must be reliable
IP-based voice (VoIP)	Constant or variable bit rate	Very sensitive to delay & jitter. Bandwidth requirements:Low Required predictable delay & loss
Streaming Video	variable bit rate	Very sensitive to delay & jitter. Bandwidth requirements:High Required predictable delay & loss

to the Internet are referred to as Internet Service Providers (ISPs). These ISPs were not originally designed to operate over separate domains that result in interoperability problems. End-to-end QoS can truly be achieved only if there is a reliable and efficient way to manage inter-domain resource allocation and QoS negotiations.

1.2 Problem Definition

This thesis answers the following question: *Is it possible to provide end-to-end Quality of Service (QoS) in IP networks without compromising the scalability, flexibility and bandwidth efficiency of the current Internet infrastructure?*

The solution is to design a QoS method to meet the stated requirements, and to

implement a QoS manager that administrates, controls and manages all the resources in the network domain. The remainder of the thesis will provide the details both a suitable QoS method (Chapter 3) and a QoS manager (Chapter 4).

1.2.1 Definition of QoS

QoS in general is defined as the ability of a network element (e.g. an application, host, router) to have some degree of assurance that its traffic and service requirements can be defined [4]. This can be achieved by measuring and improving characteristics such as transmission rate and error rate, that result in advanced guaranteed transmission [5]. In other words, it is the ability of a network provider to support a user's application requirements with regard to service categories through QoS parameters such as bandwidth, delay, jitter and traffic loss [defined in 1.2.2]. These parameters are used to measure traffic flows at the end point to ensure that the users requirements are accomplished and to make QoS model complete. In short, QoS can be defined either at the application level or the network level:

- ⋆ Application-level QoS characterizes how well the user application level is satisfied, and are usually subjective , e,g., clear voice, jitter free video, etc.
- ⋆ Network-level QoS refers to tangible measurements such as control latency, available bandwidth, packet loss rate, etc.

Network support of QoS has some features that provide better and more predictable network services by [6]:-

- · Supporting dedicated bandwidth.
- · Improving characteristics.
- · Avoiding and managing network congestion.
- · Shaping network traffic.
- · Setting traffic priorities across the network.

1.2.2 QoS Parameters

QoS parameters provide a means of specifying user requirements which may be supported by underlying networks. These requirements have to be agreed upon between the network administrator and the user before real data starts flowing through the network. For example, to support video communication high throughput is required and high bandwidth guarantees will have to be made. End-to-end delay and delay variations are other factors which must be taken into account for time-critical traffic. A set of QoS parameters suitable for characterizing the quality of service of individual connections or data flows is as follows [7] [8].

Delay

End-to-end delay is the elapsed time for a packet to be passed from a sender to a receiver through the connecting network. This parameter is important for real time applications that are sensitive to delay such as VOIP and video conferencing applications.

Delay variations (jitter)

The variation in end-to-end transient delay is called *jitter*, also often referred to as delay variation. In packet switched networks jitter defines the distribution of the inter-packet arrival times compared to the inter-packet times of the packet transmission.

Bandwidth

The maximum data transfer rate that can be sustained between two end points of the same network is defined as the bandwidth of the network link. It is not limited only to the physical infrastructure of the traffic path but also to the number of flows sharing common resources on this end-to-end path. The term bandwidth is used as the upper bound of the data transfer rate where throughput is used as an instantaneous measurement of the actual exchange data rate between two entities.

Traffic loss (Packet Loss rate)

Packet loss rate is defined as the ratio of dropped packets to the total number of sent packets. The user has to specify the peak and average transmission rates so that

resources can be reserved as required for the user.
These parameters should be negotiated during end-to-end connection time or at data flow establishment time.

1.2.3 QoS Control Mechanisms

Computer network traffic using the Internet protocol (IP) is based on a "best effort" service model. This model states that each node in the network domain will make an attempt to deliver each packet of data to its destination within a reasonable time, but it makes no guarantees at all. Packets may be delivered late, reordered or dropped [9]. To introduce guarantees to the applications, QoS in today's Internet infrastructure, several changes have to be made to allow nodes to differentiate traffic flows according to their QoS requirements. New QoS capable software and hardware would be required in most cases. Therefore, each component in IP networks must be equipped with new logical QoS supporting facilities and functionalities. According to [10][7] [11][12] the following ordered elements are the most crucial components in an architecture that supports QoS:

1- **A QoS Specification**
   A QoS Specification mechanism is used by users' applications to specify their requirements for their traffic flows.

2- **Resource Management and Admission Control**
   Certain control mechanisms are needed to allocate adequate resources to different service classes and limit the volume within each class, e.g., reservation protocols. The admission control function determines if new applications can be admitted into the system without affecting the QoS level of others already admitted.

3- **Service Verification and Traffic Policing**
   The policy control authorizes specific users to receive a particular service at a particular time. Most of the policing/shaping mechanisms used today are

based on the leaky bucket mechanism Appendix A. At the same time, admitted flows must comply to their allocated bandwidth share and should be penalized otherwise. Several techniques have been proposed [11], however it is still not clear which one will dominate. In this thesis the leaky bucket technique is used for service verification and traffic policing, as explained in the mathematical model presented in Chapter 3.

There are some parameters used to compare QoS models resource management techniques such as flexibility and scalability. **Flexibility** is defined as the ability to implement network technologies and policies aimed at smoothing users' demands and increase model's domains resources. **Scalability** is used to measure how QoS models' forwarding engines, core routers, are efficient and fast since thousands of traffic flows are processed by these engines. Therefore, as the size of theses forwarding engines overhead increases as their speed degrades which result in higher processing delay. This results in scalability degradation such as in the case of IntServ QoS model.

4- **Packet Forwarding and scheduling mechanisms**

Traffic scheduling is achieved by implementing the appropriate queue with the right parameters and buffer sizes. Different queuing algorithms will be presented in Appendix A.

5- **QoS Routing**

The basic function of the QoS routing is to find a network path which satisfies the given QoS constrains, e.g., with sufficient bandwidth or minimal delay.

Figure 1.1 shows an implementation of the QoS management requirements mentioned above. Once a QoS request has been generated and sent to the network, a provision and configuration process which includes admission, reservation, scheduling and shaping procedures takes place. This is the QoS capable network cycle responsibility. The traffic is then monitored and certain policies are applied to avoid traffic violations that affects other flows. In case of a link failure or congestion, network elements

report any of these problems that can be solved by choosing an alternative QoS path. A signaling protocol such as resource reservation protocol (RSVP) and MPLS may be used, or the user may be instructed to stop sending data until the local routing algorithms find another capable route [13].

This thesis addresses the second and third components, resource management and traffic policing, to allow proper provisioning of IP network resources so that latency sensitive applications can deliver streams of data with predictable QoS. The proposed architecture scales well with respect to the rate of Internet growth, large networks and many users, since it takes the advantage of the next generation protocol, IPv6.

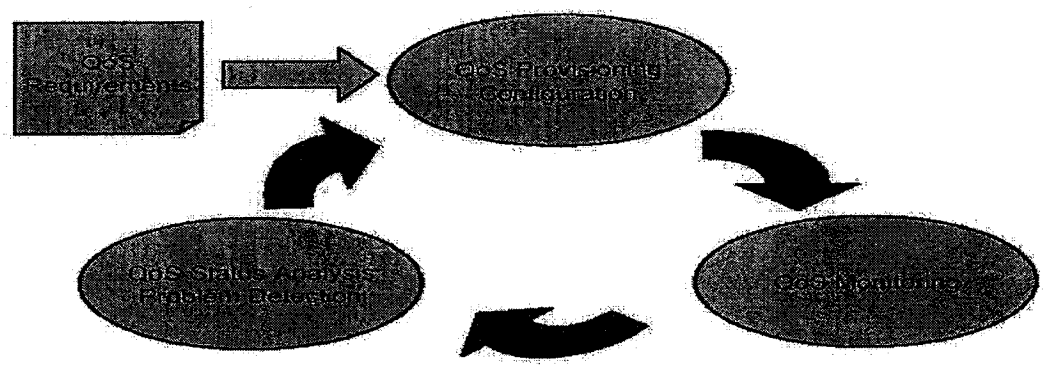

Figure 1. QoS Management Cycle

FIGURE 1.1: QoS Management cycle

1.3 State of the Art

1.3.1 QoS Approaches

In recent years, there has been considerable research focused on extending the Internet architecture to provide better QoS. Three major classes of approaches that have been proposed, for both IPv6 and IPv4, by the IEFT are: Integrated Service (IntServ) [14]),

Differentiated Services (DiffServ) [15] and Multi-Protocol Label Switching (MPLS) [16]. The mathematical models of the first two algorithms are shown in Chapter 3.

IntServ with RSVP [17] [18] introduces end-to-end per flow reservation, such that each flow is guaranteed a certain amount of bandwidth at each router along its path from the source to the destination. However, this approach requires maintenance of individual flow states in the routers, and its signaling complexity (processing and reservation setup delays) grows with the number of users. As a result this architecture does not scale well since each router has to store the reservation information for each flow. In addition to the storage problem reservation messages have to be periodically repeated for each flow which results in a large number of messages to be processed at each node. The proposed QoS model overcomes both problems by pushing the complexity to the edge nodes and inserting a central management that handles all the reservations and model policies. Therefore no refershing messages are needed and there is no need for other QoS and signaling methods involvement.

DiffServ, on the other hand, relies on packet marking and policing at the access or edge routers and different Per-Hop Behaviors (PHB) at core routers to provide service differentiation to aggregate traffic. Edge routers are boundary points at which flow enters or leaves a DiffServ domain, while core routers are internal routers within the domain. A network that supports DiffServ provides different levels of service to meet a client's request for guarantees. Network resources are partitioned between these levels of service. Packets belonging to client flows are marked with a specific code point (DSCP) that map these packets to a particular level of service depending on the Service Level Agreement (SLA) between the client and the network service provider. Recent proposals [19] [20] [21] use agents known as Bandwidth Brokers (BB) to allocate preferred services to users as requested, and to configure the network routers with the correct forwarding behavior for the defined services. BBs act as resource managers that provision resources at domain boundaries and negotiate requested traffic parameters. They track available network resources and classify flows using service polices based on client requirements and service classes offered by the

DiffServ network domain. This scheme is a scalable [10][13], however the mapping of traffic flows, to either Expedited Forwarding (EF) or Assured Forwarding (AF) (defined in Appendix B), is time consuming. The proposed QoS model avoids packets mapping and achieves per-flow QoS guarantees.

MPLS [16] has been introduced by researchers as another solution for providing QoS. MPLS [23] [24] is similar to DiffServ in some respect as it also marks traffic flows at the MPLS domain ingress boundaries in a network and un-marks them at the egress points of the same domain. 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. It is used to establish fixed bandwidth pipes analogous to Asynchronous Transfer Mode (ATM) or frame relay virtual circuits. It also uses a control driven model to initiate the assignments and distribution of label bindings for the establishment of label switching paths (LSPs) allowing different routes to be used by different packets depending on their traffic type. Using fixed labels simplifies the routing process by decreasing overhead to achieve high performance. It is scalable, however it raises the complexity of each router in the network.
The three methods mentioned achieve some level of QoS by using different approaches. However, each one of them has some drawbacks as mentioned above. Chapter 2 will discuss in detail these methods, their approaches to achieve high QoS, and their drawbacks.

1.3.2 Internet Protocol IP

The Internet protocol is designed for use in interconnected systems of packet-switched data communication networks. Its function or purpose is to move datagrams through an interconnected set of networks. This is done by passing the datagrams from one Internet model (network element) to an other until the destination is reached. The selection of the transmission path and the subsequent forwarding of datagrams along this path is called *routing*. The datagrams are routed from one Internet domain to another based on the interpretation of the Internet address in this datagram.

Two IP protocols, IPv4 and IPv6, are used in the Internet, however IPv4 is more commonly used. IPv6 has been deployed in the Far East and will, in time, become the protocol of choice, hence it was chosen for the simulations used in this thesis. A detailed description of IPv6 is given in Appendix C.

1.4 The proposal

1.4.1 Proposed end-to-end QoS Model

The QoS model proposed in this thesis is presented by a mathematical model using network calculus [26], and was studied experimently using network simulator (ns-2 [27]). The mathematical modeling is presented in Chapter 3 and the simulation results are presented in Chapter 5. Using these two techniques a direct comparison between this model and the IETF standard QoS models, IntServ and DiffServ, was made to prove that the proposed model works better than the other two schemes. The mathematical model presented in Chapter 3 works for IP networks in general. However, the simulation was based on IPv6 to utilize its new way for tracing, reserving and recording reservations using only the flow label and the source IP address. The proposed QoS model uses a per-flow QoS approach, as does IntServ, in which each traffic flow's packets are processed independetely. On the other hand, core nodes are kept simple , just forward and schedule packets, as in DiffServ which results in avoiding traffic flow mapping. QoS admissions and reservations are implemented by a central QoS manager. The model policies are observed and deployed by the edge router which monitors all incoming traffic flows.
The proposed model has the following advantages:

- Scalable and flexible, since classification and scheduling are kept at the edge points and QoS guaranteed can be assured in more than one domain. Core nodes just forward traffic flows to the next node using the GDI. Then schedule them in the proper queues depending on their priority level, traffic class TC field. Also, no refresh reservation messages, as in the case of IntServ, are communicated

between domain nodes. In addition, there is no need to decrypt packets at the core routers since the GDI is only used for forwarding decisions.

- The proposed scheme obviates the need for expensive Longest-match lookup procedure (IP routing as used in IntServ and DiffServ) for each packet at each router along the path by using the GDI for forwarding packets. This results in performance enhancement and less end-to-end delay.

- The hierarchy of routing when this algorithm is invoked reduces the size of the routing table for internal routers within a domain when compared to IP routing as used in IntServ and DiffServ.

- Negotiations are done between the edge points and customers and the manager communicates only with the edge routers which simplifies the negotiation procedures and reduces the setup time when compared to DiffServ.

- The GDI is the only field used for tracking reservations and QoS requests. This simplifies the implementation of QoS management purposes when compared to a BB in DiffServ domains.

- Dropping or remarking packets, violating the QoS agreement, is done at the edge nodes. Core nodes process packets only. This reduces the chances of overflowing the core buffers and avoids congestion.

- Mapping of incoming flows' traffic to pre-defined classes is avoided and there is no limitations on number of classes. All priority levels are considered and treated differently. Also end-to-end delay is minimized as mapping each packet to a certain class is not required.

- Simplicity since most routers support IPv6 and there is no need to support other protocols such as RSVP and MPLS. Local routing protocols are used to configure the path to a destination. This avoids running different routing protocols to configure paths as in the case of MPLS.

- Traffic flows are guaranteed end-to-end delivery as in the IntServ QoS model with less complexity and domain core routers scale well.

- Avoids the layer violation problem that is caused by port numbers lookup procedure during packets routing. This causes encrypted packets to be decrypted at each router to find the source and destination port numbers from the TCP header. The proposed model uses the GDI for packet routing, so there is no need to decrypt packets to look for the port numbers. The layer violation problem is explained in Appendix C.

1.4.2 The Proposed QoS Manager

The IPv6 traffic class field [1] is used for calculating the schedulers weights. The usage of the flow label [2] achieves faster forwarding and processing, similar to MPLS, since the IP address of the source node and the flow label are used for forwarding decisions. The traffic class field will be used to differentiate traffic flows to implement the requiring QoS at the core routers thus making routing less complex than in MPLS. The fast forwarding is achieved by using the combination of the source IP address and the traffic assigned flow label to find the next hop. This results in a reduction of processing delay during the matchup procedure at each router in the network domain, resulting in less end-to-end delay. The QoS is achieved by using the class traffic field to identify the traffic priority to assign the proper queue. Also this manager knows the topology of the network and keeps track of all the resources reserved, similar to a BB and makes decisions if QoS requests have been received. The complexity is kept at the edge points where traffic will be scheduled and classified if QoS requests have been accepted by the manager. The proposed QoS manager receives requests from the edge nodes, not from the source node as in DiffServ, resulting in faster processing of requests since both end points communicate with the manager and are pre-configured. Also there are no other protocols invoked during request processing as in MPLS. All

[1] Traffic class is used here as a generic term. In IPv4 the ToS may be used

[2] Flow label is used here as a generic term. In IPv4 a shim header may be used

domain core nodes have to send QoS requests to the domain edge routers, unlike to a BB, to be forwarded to the QoS domain manager. Delay is minimized since requests are forwarded to the manager by the edge nodes. Flows' traffic specifications are extracted from the QoS requests to be used for the classification and scheduling process at the edge routers. There is no need to map traffic to a specific set of pre-defined classes then stamp each packet with the code of the class to which it belongs as in DiffServ.

1.5 Document Organization

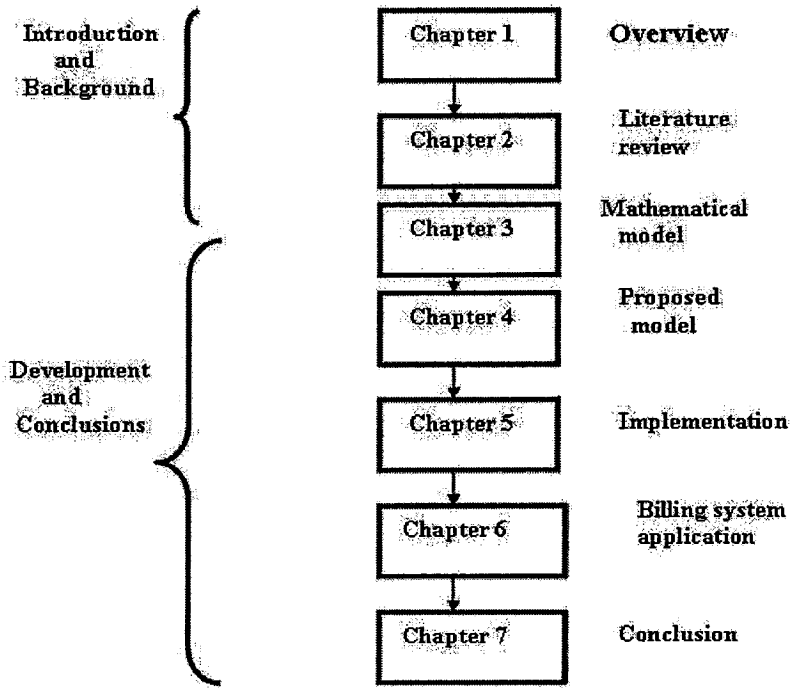

FIGURE 1.2: Thesis Organization

Figure 1.2 gives a visual presentation of how the document is organized.
In this chapter, the concept of QoS was introduced by defining it then explaining its

importance and requirements. Also the QoS model proposed in this thesis was briefly introduced specifying its advantages over other techniques.

Chapter 2, presents the state of the art in QoS. The standard QoS models used in the literature are presented with their advantages and disadvantages.

In Chapter 3, a network calculus mathematical modeling using the arrival and service curves bounds techniques to model QoS algorithms is introduced. This technique is used to develop a mathematical model for the IP QoS management proposed in this thesis. The model is used to find the worest case end-to-end delay bounds assuming no packet loss. DiffServ and IntServ mathematical models are also described in this chapter to allow direct comparison with the proposed model. In Chapter 4, the proposed QoS manager is described in detail followed by different reservation scenarios. The implementation of this model and also the simulation results is presented in Chapter 5. In Chapter 6, an important Internet application, Internet billing, is introduced and implemented with a QoS manager. This document concludes with the conclusion and future work presentation in Chapter 7.

Chapter 2

Related Work

Providing service guarantees to multimedia applications has evoked much research interest and many different approaches have been proposed. This chapter presents a survey of related work that serves as background to this thesis. In particular the survey will focus on recent development in QoS control in Section 2.1, resource provisioning techniques and bandwidth brokering architecture in Section 2.2 and traffic monitoring and policing in Section 2.3. The influence of these efforts to the this proposed architecture is discussed.

2.1 An Overview of QoS Control

The emergence of IP telephony, video conferencing and other applications with very different throughput, loss, and delay requirements are calling for substantial changes in the Internet infrastructure that was originally designed to offer a single, best-effort level of service. Providing different levels of service in the network requires new QoS control and management capabilities, which can be classified along two major axes: **data path** and **control path**. Data path mechanisms are responsible for classifying and mapping user packets to their intended service class and enforcing treatment received by each service class. Control path mechanisms allow the users and the network to agree on service definitions. They are also needed to determine to which

user to grant service, and appropriately allocate resources to each service class. The QoS mechanisms discussed in this thesis have largely been applied to IP layer (Layer 3).

2.1.1 Scheduler and Buffer Management

Data path mechanisms are the basic building blocks in a QoS-aware infrastructure. They control how packets access network resources, such as buffers and bandwidth, to provide service differentiation. The two most important mechanisms are scheduling algorithms and buffer management schemes. Scheduling mechanisms control which packets are selected for transmission on the link, while buffer management schemes decide which packets can be stored or dropped as they wait for transmission. Their importance is evident in the models of Chapter 3.

Apostolopoloulos et al. [25] review the different scheduling and buffer management schemes in and discuss their associated trade-offs in terms of fairness, isolation, efficiency, performance and complexity. For example, Weighted Fair Queuing (WFQ) [28] and its many variants provide rate and delay guarantees to individual flows, while class based scheduling mechanisms, e.g. CBQ [29] provide aggregate service guarantees to the set of flows mapped into same class. The buffer management method is an input control mechanism that determines whether a packet of a flow is to be admitted based on current buffer occupancy. The basic buffer management method allocates a certain amount of buffer space to each flow. When packets of a flow occupy all the space allocated, new packet arrivals of this flow will be discarded. Examples of buffer management schemes include: First Come First Serve (FCFS), Early Packet Discard (EPD) [7] and Random Early Detection Drop [7]. In this thesis WFQ will be used for scheduling in simulated nodes.

2.1.2 QoS Models

IntServ, DiffServ and MPLS, the three major QoS approaches still undergoing development, and their usage to implement QoS are discussed in this section.

Integrated Service (IntServ)

The philosophy behind IntServ is that routers must be able to reserve resources for individual flows to provide QoS guarantees to end users. IntServ QoS control framework supports two additional classes of service besides "best effort" : (a) Guaranteed service [30] and (b) Controlled load service [31]. Guaranteed service provides quantitative and hard (deterministic) guarantees, e.g., lossless transmission and upper bound on end-to-end delay. Controlled load service is intended to support a broad class of applications that are highly sensitive to overload conditions. Both services must ensure that adequate bandwidth and packet processing resources are available to satisfy the level service requested. This must be accomplished through active admission control. Many research contributions have been made to define IntServ components, needed to provide end-to-end QoS, functionality and study their implementation issues.

IntServ major components are:

- A signaling protocol to set up and tear down reservations, e.g., resource Reservation Protocol (RSVP) [17].
- An application level interface (API) for applications to communicate their QoS needs.
- Per-flow scheduling in the network (e.g., WFQ or CBQ).

Unfortunately, IntServ faces the following major challenges that make intermediate deployment in core routers infeasible.

i) The increase in per flow state maintenance at routers is proportional to the number of flows. This incurs huge storage and processing overhead at routers, and therefore, does not scale well in the Internet core backbone,

ii) The RSVP/IntServ model needs to work over different data links such as Ethernet, and ATM. Therefore, mechanisms to map integrated services onto specific shared media are needed.

The QoS model proposed in this thesis overcomes the shortcoming while retaining the per-flow QoS.

Differentiated Service (DiffServ)

DiffServ, on the other hand, aggregates multiple flows with similar traffic characteristics and performance into a few classes. This approach requires either end user applications, first hop routers or Ingress routers (interface where packets enter an administrative domain) to mark the individual packets to indicate different service classes as discussed in Section 1.2.2. Currently this QoS information is carried in band within the packet in the Type of Service (ToS) field in the IPv4 header or Class of Service (CoS) field in the IPv6 header [32]. The backbone routers provide per-hop differential treatments to different service classes as defined by the Per Hop Behaviors (PHBs) [33]. Two service models have been proposed: assured service [34] and premium service [35]. Assured service is intended for customers that need reliable services from service providers. Premium service provides low delay and low jitter service, and is suitable for Internet telephony, video conferencing and E-commerce applications. Figure (2.1) shows the DiffServ architectural model [36]. Therefore, individual micro flows are classified at the edge routers into one of the classes defined by the approach.

The DiffServ approach has several advantages over IntServ:-

- DiffServ is simpler than IntServ and does not require end-to-end signaling.
- DiffServ is more efficient for core routers since classification and PHBs are based on a few bits rather than per-flow information. Only a limited number of service classes indicated by ToS or CoS fields, the amount of state information is proportional to the number of classes rather than number of flows. Therefore, the DiffServ approach is more scalable than IntServ.
- DiffServ requires minimum changes to the network infrastructure. End hosts (Ingress routers) mark packets while intermediate routers (core routers) can

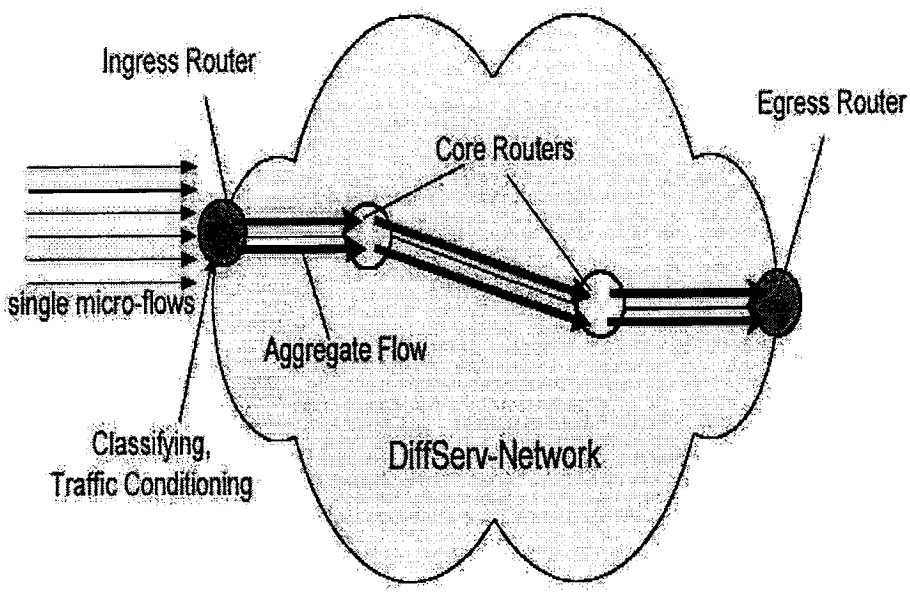

FIGURE 2.1: Differentiated Service Domain architecture

employ active queue management to provide service differentiation based on the packet headers.

Although flow aggregation improves scalability in DiffServ, there is no level of satisfaction guarantees provided for individual flows by DiffServ, unlike IntServ. The model proposed in this thesis has the advantages of DiffServ and overcomes the disadvantages of DiffServ in which traffic flows are guaranteed resources and without mapping traffic flows to predefined classes.

Multiprotocol Label Switching (MPLS)

In recent years, Multi-Protocol Label Switching, or MPLS, has been proposed as the solution to overcome many of the performance and scaling problems that service providers are experiencing in their IP (Internet Protocol) networks. MPLS networks contain network nodes, called Label Switching Routers (LSRs), and network links

connecting nodes [37]. MPLS organizes the network in domains. Edge LSRs define the boundaries of the domain and are the traffic demand ingress/egress nodes. Other nodes, named core LSRs, can exist on the network to provide communications between edge LSRs. The forwarding of IP packets from ingress to egress LSRs is done by means of routing paths, called Label Switched Paths (LSPs). In the ingress LSR, incoming IP packets are labeled based on their destination and required quality of service (QoS) and, depending on this classification, are forwarded through the appropriate LSP towards an egress LSR. MPLS enables source based routing, i.e. the forwarding path of a LSP from an ingress router to an egress router is not constrained by the paths of other LSPs, which is the basis for more efficient traffic engineering methods.

Each MPLS packet has a header containing a 20-bit label, a 3-bit *experimental* field, a 1-bit label stack indicator and an 8-bit Time to Live (TTL) field. The MPLS packet format is shown in Figure (2.2) [7]. At the ingress LSRs of an MPLS-capable

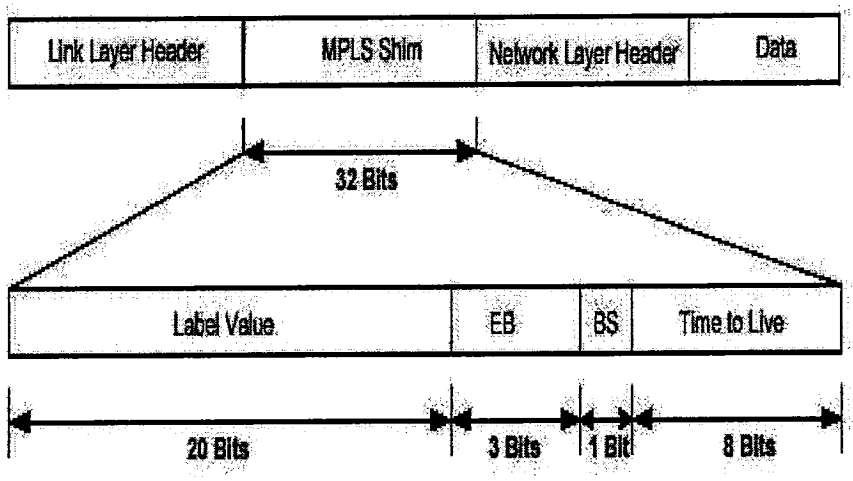

FIGURE 2.2: MPLS Packet Format

domain IP packets are classified into Forwarding Equivalent Classes (EFC) based on the combination of the information carried in the IP header of the packets and the

local routing information maintained by the LSRs. An MPLS header is then inserted into each packet. Within an MPLS-domain, each LSR uses the label as the index to look up the forwarding table of the LSR. The incoming label is replaced by outgoing label and the packet is switched to the next LSR as seen in Figure (2.3) [41]. The model proposed in this thesis can be implemented in IPv4 using a shim. However, in IPv6 the protocol provides the flow label which can be used to support the proposed model.

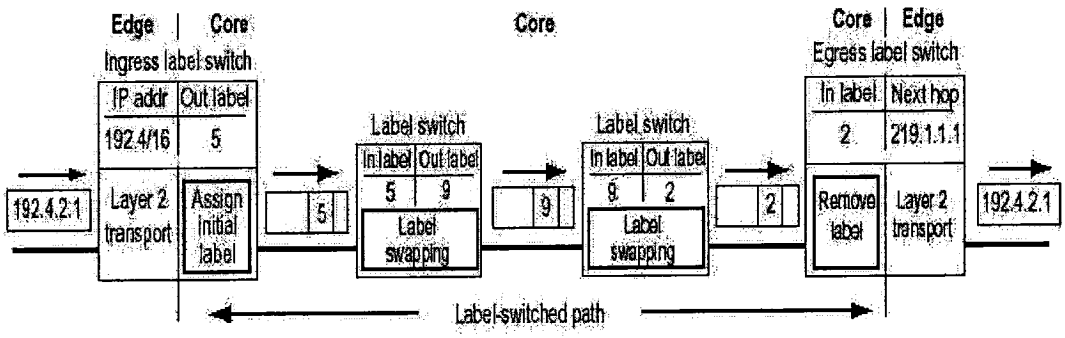

FIGURE 2.3: MPLS architecture

2.1.3 Research Efforts on QoS Model

Having introduced the main QoS Control schemes, a brief discussion of recent QoS control research is presented.

Assured Service

This model, originally called Expected-Capacity framework, was introduced by David Clark [42] [34]. The idea is to mark traffic as IN traffic if its rate is less than the requested rate or OUT traffic if it is greater. This marking is used in case of congestion in order to distinguish the treatment of IN and OUT traffic. This results in guaranteeing of no-dropping treatment for the IN traffic so that each user will get a

contracted bandwidth profile. Out of profile packets are dropped.
The model proposed in this thesis monitors the incoming traffic at the edge points and marks it. However, out of profile higher priority traffic flows are degraded to lower priorities rather than being dropped. This avoids packet loss.

IntServ Operation over DiffServ [43] [7] [44]

In the framework presented in RFC 2998 [44], end-to-end quantitative QoS is provided by applying the IntServ model across a network containing one or more DiffServ regions. From the perspective of IntServ, DiffServ regions of the network are treated as virtual links connecting IntServ capable routers or hosts. Within the DiffServ regions of the network, routers implement specific PHBs (aggregate traffic model). The total amount of traffic that is admitted into the DiffServ region that will receive a certain PHB may be limited by policing at the network's edge.
Requests for IntServ services must be mapped onto the underlying capabilities of the DiffServ network region that include appropriate PHB selection, performing appropriate admission control and policing. Boundary routers residing at the edge of the DiffServ region will typically police traffic submitted into the DiffServ region in order to protect resource within the domain. DiffServ routers classify and schedule incoming traffic in aggregate based on DSCP not on the per-flow classification criteria used by standard RSVP/IntServ routers. The IntServ edge routers process PATH and RESV messages sent from the IntServ domain to the DiffServ domain, and PATH statistics are installed in these routers. The DiffServ domain routers ignore both PATH and RESV messages. At the Ingress IntServ domain, that connects the host that initiated the reservation request, the RESV message triggers admission control processing where the resources requested in the RSVP/IntServ request are compared to the resources available in the DiffServ region. If there are enough resources, the RESV message is admitted and allowed to continue towards the sender.
The QoS model proposed in this thesis does not require RSVP to run over its domain.

The domain QoS manager decides whenever a request is received and resources are allocated if the request is accepted. This will reduce the delay during traffic mapping and allows all per-flow QoS instead of aggregate flows QoS.

MPLS over DiffServ [45] [46]

As explained before, DiffServ provides a scalable and operational simple QoS treatment, as it does not require per-flow signaling and state, to traffic aggregates. However, DiffServ can not guarantee QoS because it has no influence on traffic flows during congestion, no control on domain resources i.e. bandwidth, which result in dropping high priority packets.
MPLS on the other hand, can force packets into specific paths, LSPs, that guarantee QoS. But in its basic form MPLS does not specify class-based differentiated treatment of flows.
Therefore, combining the DiffServ classification and PHBs with MPLS leads to true QoS in the packet backbone. There are two defined types for LSPs: E-LSP and L-LSP.

1. **E-LSP**
   If a network supports up to 8 PHBs, then the EXP bits in the MPLS header are sufficient for that network. A LSR keeps a mapping of EXP values to PHBs and maintains a mapping of DSCPs to PHBs. In this case the label tells an LSR where to forward a packet, and the EXP bits determine the PHB that should be used to treat the packet. A LSR that is set up under these conditions is referred to as E-LSP.

2. **L-LSP**
   If a network has more than 8 PHBs, then 3 EXP bits not be able to convey all the PHBs to LSPs. One way is to use the label itself to convey PHBs, therefore, an LSR uses the label to determine the PHB. The EXP field is used only for the indication of the drop priority.

Although the functionality of MPLS over DiffServ is the same as that of the proposed models that latter have advantages that includes:

1- Simplicity, since no need to deploy MPLS and DiffServ on all the routers. The proposed QoS model provides flows and keeps complexity at the edge points.

2- The proposed QoS model provides per-flow guaranteed delivery and no limitions on classes.

3- Violated Higher priority traffic flows are degraded not dropped as in DiffServ.

2.2 Network Resource Management

The issues of resource allocation and management are not unique to the DiffServ architecture. In late 1997, the concept of "**Bandwidth Broker (BB)**" was introduced by K. Nicholas [47] as an entity in charge of resource management in administrative domain. The Internet2 QoS working group [48] has made an attempt to harmonize the different ideas and proposals to define a BB model to be deployed in an inter-domain DiffServ [49]. This section discusses the prior work on DiffServ BB and other resource management techniques and architecture relevant to this thesis, including: dynamic allocation and pricing based approach. Next a brief explanation will be given on BB, components and usage, and two models that are used in this thesis for comparison with the proposed QoS management model.

2.2.1 Bandwidth Broker Architecture

A Bandwidth Broker (BB) is an agent responsible for allocating preferred service to users as requested, and for configuring the network routers with the correct forwarding behavior for the defined service. A BB is associated with a particular trust region, one per domain. A BB has a policy database that keeps the information on domain reservations which includes Tspec and DSCP for all reservations. Only a BB can configure capable leaf routers that can deliver a particular service to traffic

flows. When an allocation is desired for a particular flow, a request is sent to the BB. Requests include a service type, a target rate, a maximum burst rate, and the time period when the service is required. The request can be made by a user or it might come from another region's BB. A BB first authenticates the credentials of the requester, then verifies if there is sufficient unallocated bandwidth to meet the request. If a request passes these tests, the available bandwidth is reduced by the requested amount and the flow specification is recorded. An acceptance message is sent to the neighbouring BBs and then a service level agreement (SLA) is reached between the customer and the network service provider. The BB configures the appropriate leaf router with the information about the packet flow to be given a service at the time that the service is to commence. This configuration is a "soft state" that the BB will refresh periodically. The idea of a BB was introduced as part of the Differentiated Service Architecture. The BB plays several roles in administering a differentiated service resource management [19] [20].

Need for a Bandwidth Broker [50]:

A BB :

1. is necessary to provide reliable QoS within and across the network,
2. manages resources for a single domain,
3. allocates preferred services with policies and service agreements,
4. configures network routes with correct packet forwarding behaviors,
5. performs internal and external admission control.

Bandwidth Broker Components

The bandwidth broker consists of the basic components shown in Figure (2.4) [22] [21] [51]:

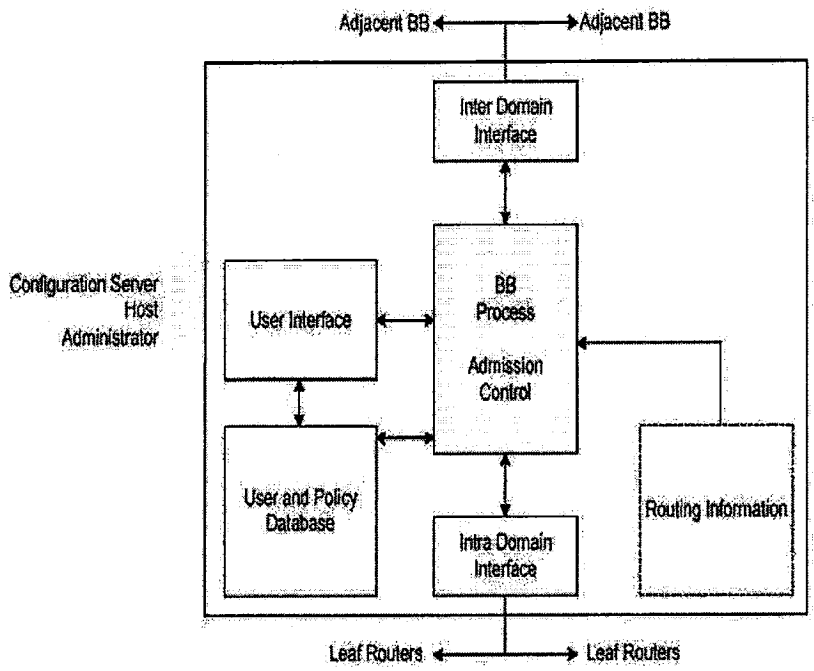

FIGURE 2.4: BB basic components

1- **User Interface:** The user/application interface provides a means for the user to make resource requests directly, or to the network operator who enters the users' requests. The interface also receives messages from setup protocols (for example RSVP messages).

2- **Inter-domain Interactions:** The interactions provide a method of allowing peer BBs to make requests for resources and take admission control decisions to enable flow of traffic.

3- **Intra-domain Interactions:** The interactions provide a method for the BB to configure the edge routers within the domain to provide the required quality of service.

4- **Routing Table (configuration client):** A routing table is maintained at the BB to access inter-domain routing information so that a BB can determine [52]

the following:

- The domain Ingress/Egress routers for its domain.
- The next domains in the path of a flow towards the destination in the case where a destination is located in an other domain.
- The first leaf router in the case where a request is generated by any host in its own domain.

Further, additional routing paths may be maintained in the routing table for different flows within the domain.

5- **Database:** A database is used to store information about all the BB parameters. The information that is stored within the repository includes [53]:-

- **SLA Data**: Parameters of each user's SLA such as maximum and minimum data rate, and the type of data (tolerant or non-tolerant).
- **Resource Allocation**: The data base maps all resource allocations, user ID, max data rate, type of data, to the user's SLA.
- **Domain Topology Data**: Data for routers, switches, and data for adjacent domains' BBs.

6- **Simple Policy Service (BB server):** It handles all communications between BB components, performs authentication, and device configuration. It is responsible for granting /denying resources, tracking available and allocated bandwidth resources within its domain, and negotiating QoS requests with adjacent BBs. It also updates the data base when new requests arrive and are granted.

Virtual Leased Line (VLL) Service [42]

Van Jacobson proposed the model of the VLL service, or premium service. It was the first model which was designed in DiffServ. VLL service focuses on guaranteeing peak-bandwidth services with marginal queuing delays and losses. Therefore, this service is

similar to a leased line in circuit switched networks. At the ingress router the network controls the peak rate of the VLL traffic that is contracted between service provider and customers using traffic shaping at the edge routers. VLL traffic should not exceed the packet rate at the network ingress. The peak rate is provisioned, reserved and classified with the highest priority in network cores along its path. Even though the VLL traffic is provisioned at networks edges and core nodes, traffic burst violation could exist somewhere in networks. The reason for this is the multiplexing of VLL traffic from different input interfaces in core routers which makes traffic become very bursty. This results in an increase in queuing delays and packet loss. The researchers later solved this by inserting a BB as a centralized control point for monitoring and control bandwidth utilization and reservations of links within a network.
Unlike VLL, the QoS management model proposed here degrades traffic. In the proposed model presented in this thesis, traffic burst are avoided since there is a QoS management that controls the resource and nodes schedulers. Traffic flows with higher priorities will not be queued for long and their packets will not be dropped.

2.2.2 Dynamic Reservation

In this subsection, two examples will be given on dynamic reserveration, Active Resource Management (ARM) [21] [51] and QoSBox [54] [55], which uses different approaches to allocate resources and control traffic flows. These models will be used in Chapter 5 as a basis of comparison with the QoS management model developed in this thesis.

Active Resource Management (ARM) BB

Bandwidth broker agents are used in the DS domain to enable more intelligent allocations for network resources. BB agents maintain a database of parameters pertaining to the various traffic flows. These parameters include service level agreement (SLA), reservation/allocation, edge routers configurations, service mappings, reservations and violations policy information and management information. Based on

these parameters, the broker agent makes a reservation for the client and assigns a DSCP for that service. Each client is provided with a service level specification (SLS) specifying the amount of bandwidth, duration of the connection and a few other parameters. These parameters are mapped together to a particular DSCP that defines the assigned level of service. Incoming client packets are then marked using this DSCP to inform the routers to forward packets with appropriate priority level. Usually the bandwidth agreed upon will be reserved for that particular traffic flow however, there will be waste of resources during low traffic rates. In this resource management scheme, traffic flows are monitored and in case of violation traffic packets are either dropped or DSCP is lowered down to suite the current rate. The remaining unused bandwidth is now sent to a pool of bandwidth that is maintained for best effort services. In case of client increases its rate to the peak rate and more bandwidth is needed, this required bandwidth to be retrieved by dipping into the pool of bandwidth belonging to best effort services [21] [51].

ARM bandwidth broker is the latest BB implemented in ns-2 simulator for DiffServ simulations. It was chosen for the comparison with the proposed QoS manager since it also dynamically re-allocates resources, in which more unused resources are freed. ARM uses five tuples for reserving, admitting and tracing QoS requests, and requests are communicating with the customer. The proposed QoS manager uses the GDI for the same functions, which is expected to result in less time during request processing, and requests are forwarded from the edge router which extracts accepted QoS request parameters for scheduling and monitoring purposes, which is expected to reduce negotiation time.

QoSbox

The QoSbox is a configurable IP router that provides per-hop service differentiation to classes of traffic flows with similar QoS requirements. All service guarantees are provided over a finite length time interval whose beginning is defined as the last time the output queue was not backlogged. Similarly, the loss rate and throughput are

computed over the current interval. The key difference between the QoSbox and other QoS architectures is that the QoSbox does not rely on any external mechanism, as does BB in DiffServ, or any component to enforce the desired service guarantees. For instance, there is no need for BB or traffic shapers to regulate the traffic arrivals at any given router. Instead, the QoSbox adapts packet scheduling and dropping decisions based on a function of the instantaneous traffic arrivals.

When a packet passes to the interface governing the output link, a classifier looks up to which class the packet belongs and places it in the appropriate per-class buffer. The classifier does not need to identify the flow to which the incoming packets belong. After, the incoming packets have been placed in a per-class buffer, the rate allocation and packet dropping algorithms adjust the service rates allocated to each class of traffic and possibly drop packets in order to enforce the desired service agreements. The key difference between the mechanisms used in QoSbox and other mechanisms used in other QoS architectures is that QoSbox uses a single algorithm for adapting traffic demands such as service and loss rates. The two main parameters for this algorithm are delay and service rate allocated. Others use different strategies to handle this issue [54] [55].

The following are the disadvantages of using this box:

1. Manages the in traffic without QoS management consultation, therefore sending hosts send traffic without knowing whether or not if there are enough resources.

2. Limitation of number of pre-defined classes, only four classes are recognizable by the router. This causes different priority traffic flows packets treated the same way.

3. No communication with other networks neighbors since the box works independently. This causes lake of QoS assurance, no resources guarantee outside the backbone network.

In this thesis these disadvantages are overcome by 2) no limitation on classes, and 1) and 3) communication between the the QoS manager and the edge routers.

2.2.3 Pricing-based Approaches

Currently the Internet based on IP networks supports a single best-effort service. In this scheme, all packets are queued and forwarded with the same priority. No guarantees are made that a given packet will actually reach its destination; much less arrive in a time [56]. However, many Electronic Commerce applications make use of the Internet as a transport infrastructure because of its reachability, popularity and cost efficiency. Typically, these applications are delay and loss sensitive and the packet may be encrypted for security reasons. Challenges faced by ISPs supporting e-commerce traffic include enhancing their traffic flow handling capabilities, speeding the processing of these packets at core routers, and incorporating Quality of Service (QoS) methods to differentiate between traffic flows of different classes [57]. These schemes add to the infrastructure costs of network providers which can be recovered by introducing extra charges for traffic requiring special handling. Many pricing schemes have been proposed for QoS-enabled networks.

The following are some of these pricing approaches used in the literature:

1. **Flat Pricing** [58], where users are charged a fixed amount per unit time.

2. **Priority Pricing** [58], packets with the highest priority are charged the most since resources are guaranteed for them at all network situations.

3. **Per-Packet Pricing** [59], customers are charged on the basis of how many packets they sent and how many hops their traffic packets passed through. Each hop in the path to the flow's destination marked each individual packet and then at the destination the cost is determined by adding the charges for all the flow's packets.

4. **Dynamic Pricing** [60] [61], packet with different priority are charged according to their traffic contents. This charge is called the base price, which covers equipment and maintenance/administrative costs. Packets exceed their initial rates are charged extra and resources are guaranteed for them even during network congestion.

These four schemes will be studied in detail in Sections 6.2 and 6.3.

However, integrating pricing and admission control has not been studied in detail. In this thesis a dynamic pricing model [60] is integrated with the proposed QoS manager [62] to study the effects of increasing traffic flows rates on the increased cost of delivering high priority traffic flows. The pricing agent that is part of the QoS manager assigns the prices for each traffic flow accepted by the domain manager. These prices are dynamically calculated according to the network status. Combining the pricing strategy with the QoS manager allows only higher priority traffic packets that are willing to pay more to be processed during congestion. This approach is flexible and scalable as end-to-end pricing is decoupled from the network core and core nodes are not involved in QoS decisions and reservations. The implementation of one of the latest pricing models with the proposed QoS manger will be described and simulated in Chapter 6.

Chapter 3

Mathematical Modeling

3.1 Introduction

Network calculus is a collection of results based on Min-Plus algebra, which can be applied to deterministic queuing systems found in communication networks. It is a set of recent developments which provide a deep insight into flow problems encountered in networking [63]. The Network calculus approach is deterministic and does not depend on probabilistic descriptions of traffic. It is used with envelope bounded traffic models to provide a worst-case analysis on network performance. Network calculus is based on the idea that given a regulated flow of traffic into the network, one can quantify the characteristics of the flow as it moves from element to element through the network [65]. This means that traffic flows are bounded at the ingress of networks' domains by regulators that have arrival curves then these constrained flows are bounded again by the nodes' schedulers that have latency service curves. The latency service curves introduce a processing delay or rate latency which is different from scheduler to another. The end-to-end delay will include this latency delay.
The deterministic network calculus has became a fundamental theory for QoS networks, and has provided powerful tools for reasoning about delay and backlog in a network with service guarantees to individual traffic flows. Two QoS methods, IntServ and DiffServ, have been studied recently and delay bounds have been mathematically

derived for each one of them.

Section 3.2 of this chapter defines the arrival and service curves and the three main bounds that affect traffic flows. Then, delay bounds for three well know schedulers to be used in modeling QoS methods are derived. Section 3.3 starts by defining a general model for QoS then the implementation of this model on IntServ, DiffServ and the proposed IP QoS model is shown. The end-to-end delay for each model is derived then compared to prove that the proposed IP QoS model introduces the lowest end-to-end delay. In the last section some numerical experiments have been illustrated to enforce the analytical results.

3.2 Definition of Basic Network Calculus Elements Used

This section is divided into two subsections, first subsection defines the major Network Calculus elements used in the modeling. The second subsection discusses the major bounds that will be later determined for each QoS mechanism.

3.2.1 Arrival and Service Curves

Deterministic bounds on quantities such as loss and delay can be expressed if there are constraints on traffic flows and service guarantees. Therefore, traffic flows sent by sources are regulated or smoothed at the network ingress point by arrival curve modeling. Networks nodes are modeled by service curves. Data flows are described by means of a cumulative function $R(t)$ that defines the number of bits seen on the flow in time interval $[0, t]$. The arrival and service curves are defined next.

Arrival Curve

An arrival curve is a function that defines an upper bound on the arrival rate of a flow to a particular network node.

Given a wide-sense increasing function [26] α defined for $t \geq 0$, a flow R is constrained by α if and only if for all $s \leq t$

$$R(t) - R(s) \leq \alpha(t - s) \tag{3.1}$$

Two standard arrival curves, token bucket and IETF IntServ models, are used in this chapter. The token bucket is defined as:

$$\alpha(t) = \rho t + \sigma \tag{3.2}$$

where ρ is the average rate and σ is the burst tolerance. The IETF IntServ model is defined as:

$$\alpha(t) = \min\{M + pt, rt + b\} \tag{3.3}$$

where M is the maximum packet size, p is the peak rate, b is the burst tolerance and r is the sustainable rate. These 4 tuples are defined as the traffic specification (T_{spec}) parameters. The two curves are shown in Figure (3.1) [65]

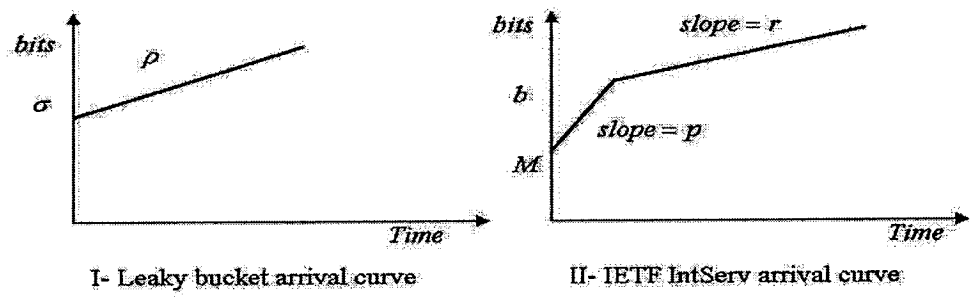

FIGURE 3.1: Leaky bucket and IETF IntServ Arrival Curves

Service Curve

A service curve is a function that defines a lower bound on the departure rate from a network.

If a system S has an input flow $R(t)$ and output flow $R^*(t)$, then S offers to the flow a service curve $\beta(t)$ if and only if for all $t \geq 0$ [26].

$$R^*(t) \geq \inf_{s \leq t}(R(s) + \beta(t-s)) \tag{3.4}$$

A simple curve is a GPS (Generalized Processor Sharing) node [66]. A rate-latency service curve is of the form

$$\beta(t) = \left\{ \begin{array}{ll} R(t - T_{lat}) & t \geq T_{lat} \\ 0 & t < T_{lat} \end{array} \right\} \tag{3.5}$$

The traffic flow is guaranteed a service at rate R bps during a busy period. However, there is a latency delay, T_{lat}, added to the traffic end-to-end delay as shown in Figure (3.2) . Each router scheduler allocates a service curve based on both the traffic

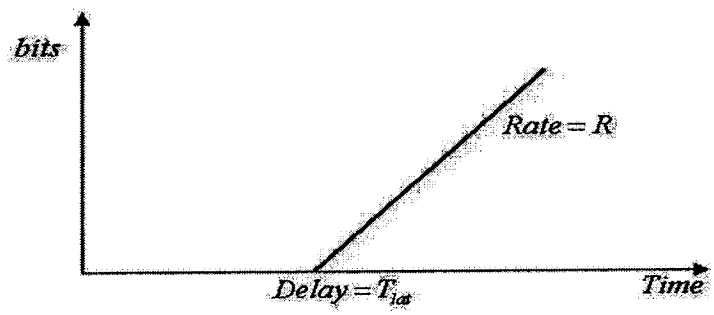

FIGURE 3.2: Service Curve

specification and a local delay bound, T_{lat}, that includes the scheduler processing time T_{proc} and the router's lookup time δd.

Figure(3.3) shows a cumulative traffic $R(t)$ constrained by arrive curve $\alpha(t)$ which is IntServ arrival curve. Then this flow is lower bounded and delayed by latency service curve ($\beta(t)$). The resulted flow is $R^*(t)$. .

3.2.2 The Fundamental Bounds

Using the arrival and service curves, bounds are derived that determine the input-output relationship for regulated traffic as it passes through basic network elements.

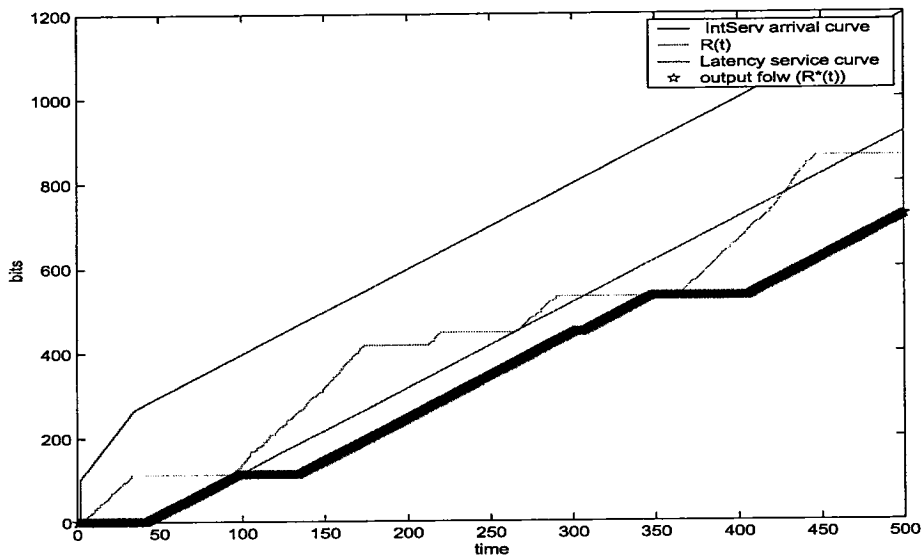

FIGURE 3.3: An example of traffic flow bounded by $\alpha(t)$ and $\beta(t)$

There are three fundamental bounds that are used in the theory of network calculus for lossless system with service guarantees. [26] [67]

•Backlog

The backlog is defined as the amount of bits that are held inside the system. In Network Calculus the backlog is the vertical deviation between the input flow $R(t)$ and the output flow R^*. The following bound a backlog is derived in [26].

Assume a flow $R(t)$ constrained by arrival curve α, traverses a system that offers a service curve β. The backlog $R(t) - R^*(t)$ for all t satisfies

$$R(t) - R^*(t) \leq \sup_{s \geq 0}\{\alpha(s) - \beta(s)\} \tag{3.6}$$

• Maximum Delay

The second bound is defined as the delay experience by a bit arriving at time t. In [26] it is shown that the delay is bounded by the maximum horizontal deviation between the arrival and service curves.

$$d_{max} \leq h(\alpha, \beta) = \sup_{t>0}\{\inf\{\tau \geq 0 : \alpha(t) \leq \beta(t+\tau)\}\} \tag{3.7}$$

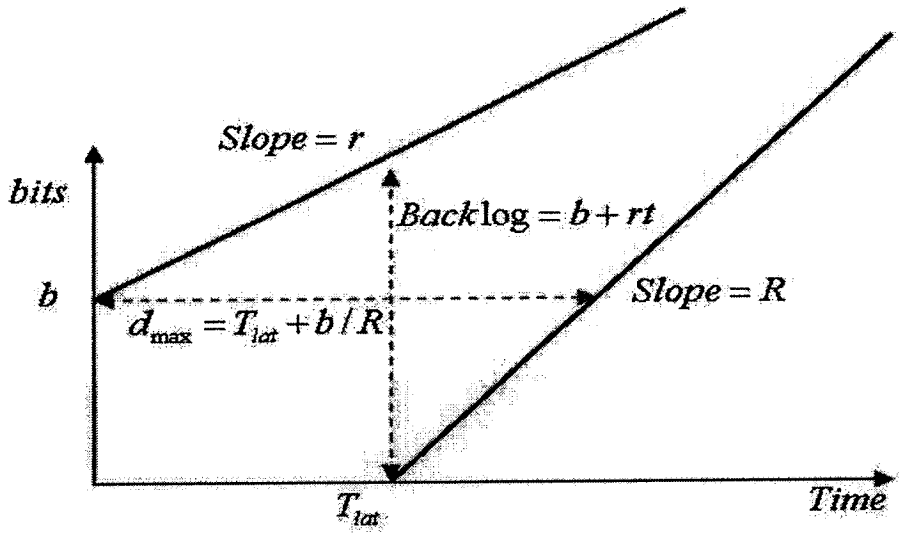

FIGURE 3.4: Delay and Backlog bounds for token bucket model

Figures (3.5) (3.4) shows these two bounds for the IETF IntServ and token buckets models.

From Figure (3.4), the maximum delay and the maximum backlog are:

$$\begin{aligned} d_{max} &= T_{lat} + b/R \\ Backlog &= b + rt \end{aligned} \tag{3.8}$$

The formulas for IETF IntServ are derived in [26]:

$$\begin{aligned} \theta &= \frac{b-M}{p-r} \\ Backlog &= b + rt(\theta - T_{lat})^{+}\left((p-R)^{+} - p - r\right) \\ Delay &= \frac{M + \theta(p-R)^{+}}{R} + T_{lat} \end{aligned} \tag{3.9}$$

where $(\theta - T_{lat})^{+}$ equals zero if $T_{lat} < \theta$ and $(p-R)^{+}$ equals zero if $R < p$.

- **Output Flow Bound**

Assume a flow constrained by arrival curve α traverses a system that offers a service curve β. The output flow is constrained by the arrival curve [26].

$$\alpha^{*}(t) = \sup_{u \geq 0}\{\alpha(t+u) - \beta(u)\} \tag{3.10}$$

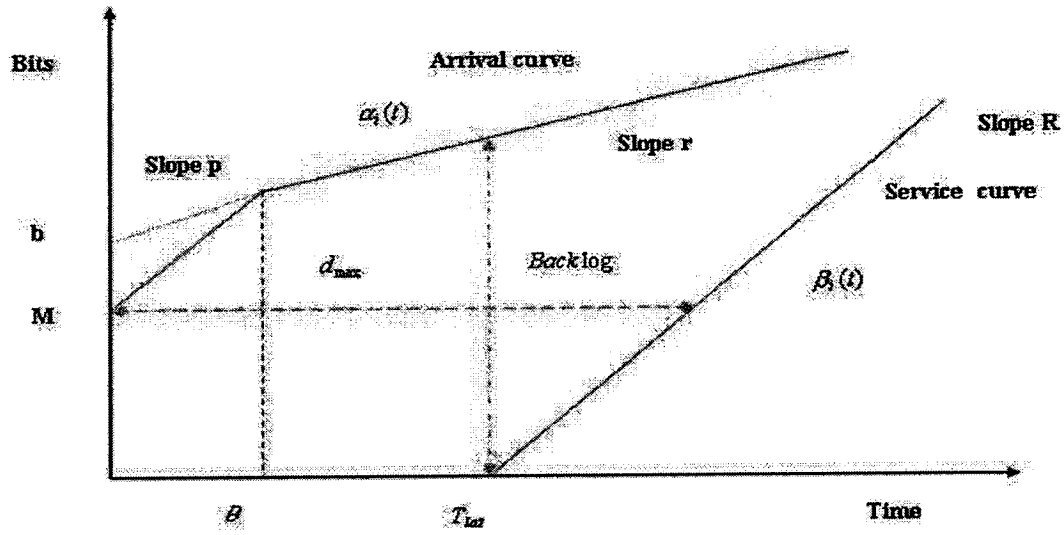

FIGURE 3.5: Delay and Backlog bounds for IETF IntServ model

3.2.3 Packetization Effects

A packetizer can be thought of as a service that collects bits until entire packets can be delivered. Therefore, packetization introduces additional latency to the service curve, $\frac{L_{max}}{R}$, where L_{max} is the maximum packet size and R is the scheduler rate. Therefore, the end-to-end delay is increased due to the added packetization factor. Thus Packetization factor will be added to all the traffic flow path nodes, resulting in end-to-end delay increase [68] [69].

3.2.4 Calculated Delay Bounds for Some Schedulers

Using the arrival curves, service curves and the fundamental bounds the maximum end-to-end delay and other performance measures can be obtained for networks of arbitrary topologies. In this section three schedulers, FIFO, WFQ and CBQ, are used to illustrate how delay bounds can be calculated for the case of traffic regulated by burstiness σ and rate ρ. The worest case end-to-end delay experienced by a traffic flow in a network of schedulers can be calculated by the sum of the latencies of the

traffic flow path schedulers and traffic parameters of the flow that guarantees the packet.

$$D_{end2end} \leq \frac{\sigma}{\rho} + \sum_{m=1}^{M} T_{lat_k}^m \tag{3.11}$$

where $T_{lat_k}^m$ is the latency experienced by connection k at server (m).
FIFO, WFQ and CBQ are used to illustrate how the delay bound can be calculated using the arrival and service curves parameters such as σ and ρ.

Guaranteed Rate Nodes:

A guaranteed service network offers delay and throughput to flows, provided that the flows satisfy some arrival curve constraints. This requires that network nodes implement some form of packet scheduling. Packet scheduling is defined as the function that decides, at every buffer inside a network node, the service order for different packets. Simple forms of packet scheduling are FIFO, CBQ and WFQ in which packets are scheduled by different scheduling techniques as will be explained in Appendix B. The scheduling delay for each one of them is derived next.

•**FIFO** [65]
In this case, the arrival curve $\alpha(t) = \sigma + \rho t$, and the service curve $\beta(t) = Rt$ are shown in Figure (3.6). The maximum delay calculated is

$$d_{max} = \frac{\sigma}{R} \tag{3.12}$$

where R is the link rate.

•**WFQ** [65] [71]
The token bucket arrival curve is used in this calculation. The service latency is given by the following equation:

- **Token Bucket:**

$$T_{lat} = \frac{L}{g} + \frac{L_{max}}{R} \tag{3.13}$$

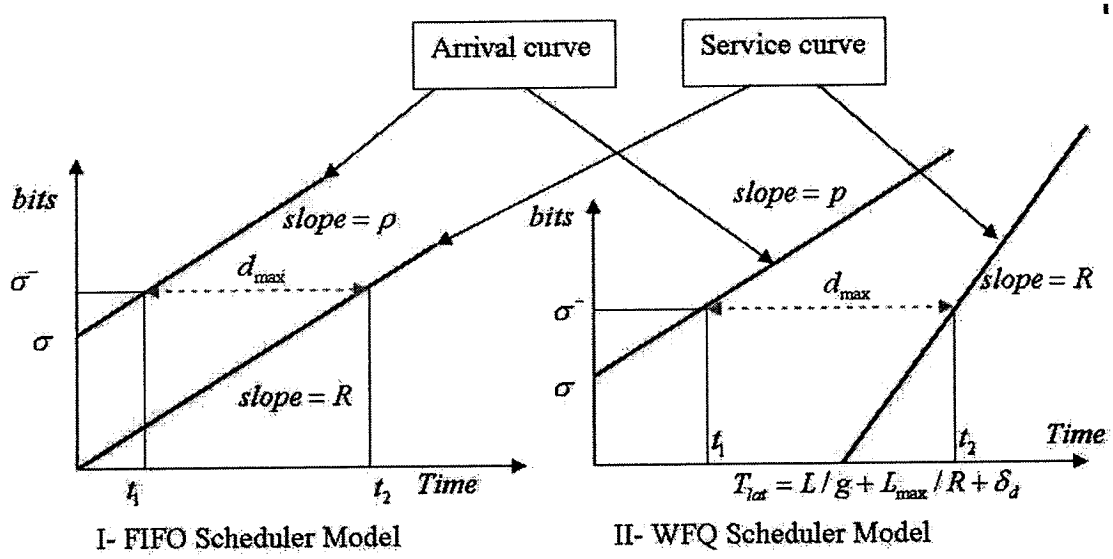

FIGURE 3.6: FIFO and WFQ max delay and backlog

where L is the maximum packet size of the flow, L_{max} (MTU) is the maximum packet size of all flows, g is guaranteed rate for each individual flow. This rate is calculated using the flow's assigned weights [70] [72].

$$g = \frac{\phi_i}{\sum_{j=1}^{M} \phi_j} * R \tag{3.14}$$

where ϕ_j are the weights of the individual flows.

- **IEFT IntServ:** The maximum calculated delay is

$$d_{max} = \frac{\sigma + L}{g} + \frac{L_{max}}{R} \tag{3.15}$$

Figure (3.6) shows the WFQ model.

•**CBQ** [65]

The same arrival and service curves used in WFQ are used in determining the max delay in CBQ model.

$$d_{max} = \frac{\sigma_p + L_p}{g_p} + \frac{L_{max}}{R} \tag{3.16}$$

where d_{max} is the maximum delay for traffic class p.

3.3 QoS Mathematical Service Element Model

In this section, a general mathematical model will be laid out, then three QoS techniques, IntServ, DiffServ and the proposed model, are discussed and compared. End-to-end delay is calculated for each QoS technique.

3.3.1 Model Layout

The mathematical model consists of three parts: regulator, delay elements and the router's schedulers. The regulator is located at the ingress of the network domain and is used to regulate all the coming traffic flows. It is represented by an arrival curve that depends on the incoming traffic parameters. Two regulators will be used in this section, the token bucket and the IETF IntServ regulator. The delay element causes the input traffic to be delayed by T_{prop} which is the propagation delay. The last element is the router's scheduler which use latency service curves to constrain the incoming traffic. The last two elements will be combined and presented by a latency service curve that has a latency T_{lat} equals to the sum of the link propagation T_{prop} and node processing delay T_{proc}, and the router's lookup time, δd. The model is shown in Figure (3.7) [69] [67]. In this chapter, an end-to-end delay bound equation will be derived for each QoS model then compared to find out which one results in smaller bounds.

For an α-smooth flow served by a scheduler, guaranteed node, with rate r and latency T_{lat}, the delay bound for a packet is [26] [75]

$$d_{bound} = \sup_{t>0}[\frac{\alpha(t)}{r} - t] + T_{lat} \tag{3.17}$$

where T_{lat} is the node latency that includes the scheduler latency plus the node lookup or processing time δd.

The previous equation will be used for the end-to-end delay bounds.

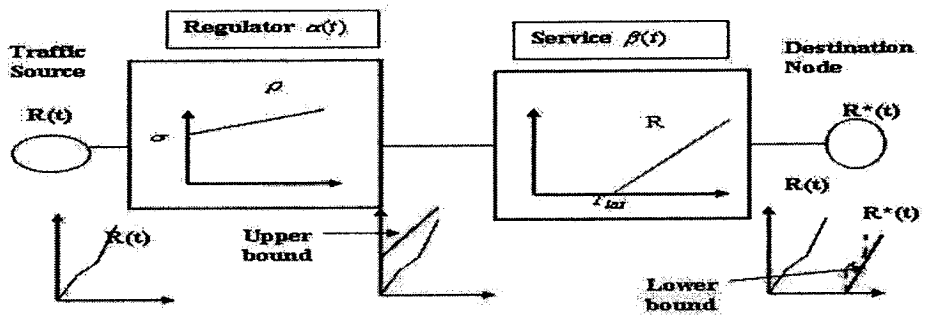

FIGURE 3.7: Block Diagram for the Network model

3.3.2 The IntServ Model

As shown in the previous section, the IntServ model has an arrival curve $\alpha(t) = \min\{M + pt, rt + b\}$ The delay bound for each IntServ node as derived as follows. Figures (3.5) shows how the delay is calculated.

$$D_{max} = T_{lat} - \frac{(b - M)}{(p - r)} + \frac{p(\frac{b-M}{p-r}) + M}{R} \tag{3.18}$$

The IntServ model for a router is that the service curve offered to a flow is always a rate latency function with parameters related by the following relation.

$$T_{lat} = \frac{C}{R} + D + \delta d \tag{3.19}$$

where C is the maximum packet size (L) of the flow and $D = \frac{L_{max}}{R_{link}}$, where L_{max} is the maximum packet size in the router across all flows, and R_{link} is the rate of the scheduler link. The following equation is used to find the end-to-end delay bound assuming all the nodes in the path are IntServ nodes [64] [65] [26].

$$\text{end-to-end delay} = \frac{b - M}{R}\left(\frac{p - R}{p - r}\right)^{+} + \frac{M + C_{total}}{R} + D_{total} + \delta d_{total} \tag{3.20}$$

C_{total} and D_{total} are the sum of the parameters C and D of all the routers in the path. δd_{total} is the sum of the lookup time for all the routers in the path.

If R (service rate) is greater than or equal to the traffic peak rate (p), Equation (3.20)

is rewritten as:

$$\text{end-to-end delay} = \frac{M + C_{total}}{R} + D_{total} + \delta d_{total} \tag{3.21}$$

The term $\frac{M+C_{total}}{R} + D_{total}$ represents the WFQ queuing delay and the term $\frac{b-M}{R}(\frac{p-R}{p-r})$ is the delay caused by the reservation and arrival curve. Therefore the end-to-end delay can be rewritten as the sum of scheduling/regulating delay, Queuing and link propagation delay and the processing delay which is the lookup time needed for each router. The IntServ model lookup delay δd is high since each router checks whether the incoming packet belongs to one of the reserved flows or not, then check the best match up to forward it to the next hop.

3.3.3 The DiffServ Model

DiffServ micro flows are constrained by the token bucket arrival curve $(\rho_i t + \sigma_i)$ at the network access. Inside the network, EF micro flows are not shaped. The DiffServ flows traverse a maximum of 10 hops inside each DiffServ domain. Therefore, the delay range for each flow is $[0, (h-1)D]$, thus the arrival curve for the EF aggregate at this node is $vr_m(t + (h-1)D)r_m\tau$, where h is the total number of hops used by a flow. v_m is defined as the utilization factor $v_m = \frac{\rho_i}{r_m}$, m is on the path of the micro flow i, D is a bound for the queuing delay for an EF node and τ is the scale burstiness factor and defined as $\tau_m = \frac{\sigma_i}{r_m}$. The data arriving at node m has undergone a variable delay $[0, (h-1)D]$ thus an arrival curve for the EF aggregate at node m is

$$\alpha(t) = vr_m(t + (h-1)D) + r_m\tau \tag{3.22}$$

By applying Equation (3.17), the maximum delay at the EF node is:-

$$\begin{aligned} D &= \frac{\alpha(t)}{r} - t + T_{lat} \\ D &= \frac{vr_m(t + (h-1)D) + r_m\tau}{r_m} - t + T_{lat} \\ D &= v(t - (h-1)D) + \tau - t + T_{lat} \\ D - (h-1)D &= \tau + T_{lat} \end{aligned}$$

$$D = \frac{\tau + T_{lat}}{1 - (h-1)v} \tag{3.23}$$

where τ is the upper bound on all τ_m and v is the upper bound on all v_m.
Also from Figure (3.8), the horizontal distance between the arrival curve $\alpha_i(t)$ and the service curve $\beta_i(t)$ is bounded as derived in Equation (3.23) [26] [69].
The latency parameter T_{lat} is the scheduler latency, in this thesis WFQ is used and

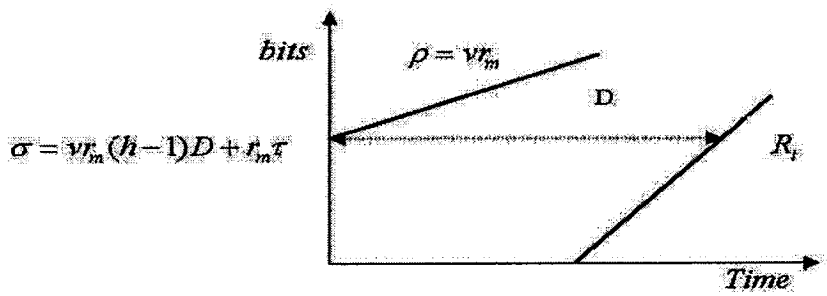

FIGURE 3.8: DiffServ maximum delay calculation

T_{lat} is given in Equation (3.13).
The end-to-end delay for the DiffServ model is the sum of all the EF delay nodes.

$$\begin{aligned} \text{DiffServ end-to-end delay} &= h * D + h * \delta d \\ &= h\left(\frac{\tau + T_{lat}}{1 - (h-1)v}\right) + h * \delta d \end{aligned} \tag{3.24}$$

where $h * \delta d$ is the total lookup time for all the DiffServ nodes.

3.3.4 The Proposed Model

In this model WFQ schedulers are implemented at each router and a token bucket regulator is used at the ingress to constrain the incoming traffic. Therefore the end-to-end delay will be the sum of the schedulers' latency and the the sum of the routers' lookup time δd or time required by the router to find the next hop.

Using Equation (3.13), the model end-to-end delay can be written as:

$$D_{end2end} = \frac{\sigma_k}{min_m(g_k^m)} + \sum_{m=1}^{M}(\frac{L_k}{g_k^m} + \frac{L_{max}}{R^m}) + \sum_{m=1}^{M} \delta d^m$$

$$D_{end2end} = \frac{\sigma_k}{g_k} + M \times \left(\frac{L_k}{g_k} + \frac{L_{max}}{R}\right) + M \times \delta d \tag{3.25}$$

where L_k is the maximum packet size for the k^{th} flow, L_{max} is the maximum packet size of all flows that are passing through these nodes, R is the m^{th} latency server rate and M is the total number of hops that a traffic flow uses to reach destination.
Equation (3.25) can be expressed as that the end-to-end delay is the sum of the time caused by a regulator or smoother plus the total schedulers queuing delay (in this case WFQ schedulers delay).
δd is the router lookup time which is the time needed to find the GDI match for the packet since in this model only the GDI is needed to process each packet. This will result in less processing time and reduce the router entry tables instead of five fields to look at only two will be used. Therefore, the end-to-end delay is reduced.

3.4 Comparison of Models

In this section, the delay bounds for the three QoS models mentioned in the previous section are compared to show that the proposed model results in the lowest end-to-end delay. Then, a numerical example is presented to illustrate the comparison.

3.4.1 Comparison

In this subsection the three models' equations are compared to find the model that results in the lowest end-to-end delay. The comparison is based on the assumption that the same traffic flow parameters are used by the three models and h hops will be needed to reach the destination. Therefore Equations (3.20, 3.24, 3.25) become:

- IntServ Model

$$\text{end-to-end delay} = \frac{b-M}{R}\left(\frac{p-R}{p-r}\right)^{+} + \frac{M}{R} + h \times \left(\frac{L_k}{g_k} + \frac{L_{max}}{R}\right) + h \times \delta d_{IntServ}$$

- DiffServ Model

$$\text{end-to-end delay} = h \times \left(\frac{\tau + \left(\frac{L_k}{g_k} + \frac{L_{max}}{R}\right)}{1-(h-1)v}\right) + h \times \delta d_{DiffServ}$$

- Thesis proposal Model

$$\text{end-to-end delay} = \frac{\sigma_k}{g_k} + h \times \left(\frac{L_k}{g_k} + \frac{L_{max}}{R}\right) + h \times \delta d$$

Relating these three equations to Equation (3.13), WFQ scheduler queuing delay, results in:

$$\begin{aligned}
\text{IntServ delay} &= \frac{b-M}{R}\left(\frac{p-R}{p-r}\right)^{+} + \frac{M}{R} + \text{WFQ schedulers delay} + h \times \delta d_{IntServ} \\
\text{DiffServ delay} &= h \times \left(\frac{\frac{\sigma}{R}}{1-(h-1)\frac{\rho}{R}}\right) + \frac{\text{WFQ schedulers delay}}{1-(h-1)\frac{\rho}{R}} + h \times \delta d_{DiffServ} \\
\text{Proposed model delay} &= \frac{\sigma}{g} + \text{WFQ schedulers delay} + h \times \delta d
\end{aligned} \tag{3.26}$$

Assuming that the three models use WFQ which results in the same queuing delay. δd is smaller than $\delta d_{DiffServ}$ and $\delta d_{IntServ}$ since two fields are needed to find the next hop comparing with the lookup for the 5 tuples to find the best match up. In addition IntServ has to look for each packet to find if it belongs to reserved traffic flow.

In IntServ model $M = L$ which is σ, therefore, IntServ has the term $\frac{b-M}{R}\left(\frac{p-R}{p-r}\right)$ extra than the proposed model plus the difference between the lookup delay caused by both models.

In DiffServ, the term $\frac{\sigma}{R}$ is multiplied by the number of hops which results in more delay than the proposed model in addition to the lookup delay difference between both models.

In conclusion, the proposed model produces lower end-to-end delay bounds than DiffServ and IntServ.

3.4.2 Numerical Example

In this subsection, three traffic flows with different priority levels are tested under each one of the QoS model. The parameters used in these calculations will be used later for simulation purposes. For each traffic flow, average rate, packet size, queue buffer size and the priority level are chosen. The link rate is also chosen. The packet size is chosen to be the same as the maximum packet size since IPv6 nodes do not fragment packets. These calculations do not include the nodes lookup time δd .
The following are the traffic parameters:

1. **Traffic 1** has an average rate of 500 Kbps, packet size of 500 bytes and priority is 15.

2. **Traffic 2** has an average rate of 250 Kbps, packet size of 500 bytes and priority is 12.

3. **Traffic 3** has an average rate of 250 Kbps, packet size of 500 bytes and priority is 8 "Best effort".

The link rate is chosen to be 1 Mpbs and the propagation delay is 1 msec.

Latency Calculations

The WFQ scheduler guaranteed rates for the three flows are calculated using Equations (3.14).

$$\begin{aligned} g_1 &= \frac{15}{15+12+8} \times 1 \times 10^6 = 0.4286 Mbps \\ g_2 &= \frac{12}{15+12+8} \times 1 \times 10^6 = 0.3429 Mbps \\ g_3 &= \frac{8}{15+12+8} \times 1 \times 10^6 = 0.2286 Mbps \end{aligned} \tag{3.27}$$

The latencies for the three flows are calculated using Equation (3.13), however, $L_{max} = L$ since IPv6 does not support fragmentation and all the three flows packets have the same size.

1. Flow 1 latency is:

$$T_{lat1} = \frac{L \times (g_1 + R)}{g_1 \times R} = \frac{0.5 \times 8 \times 10^3 (1.4286 \times 10^6)}{0.4286 \times 10^6 \times 1 \times 10^6} = 13.3msec \quad (3.28)$$

2. Flow 2 latency is:

$$T_{lat2} = \frac{L \times (g_2 + R)}{g_2 \times R} = \frac{0.5 \times 8 \times 10^3 (1.3429 \times 10^6)}{0.3429 \times 10^6 \times 1 \times 10^6} = 15.7msec \quad (3.29)$$

3. Flow 3 latency is:

$$T_{lat3} = \frac{L \times (g_3 + R)}{g_3 \times R} = \frac{0.5 \times 8 \times 10^3 (1.2286 \times 10^6)}{0.2286 \times 10^6 \times 1 \times 10^6} = 21.5msec \quad (3.30)$$

End-to-end Delay Calculations

In this subsection, the equations derived in Section 3.3 will be used to find the maximum end-to-end delay for the three traffic flows when the three QoS models are involved. This allows direct comparison between the schemes.

IntServ Model

Equation (3.20) is used to calculate the end-to-end delay for the three flows. The parameters used for the service and rate latency curves are:

- Flow 1 IETF arrival curve parameters:
  M=σ=0.5 KBytes.
  r=0.5Mbps
  b=0.9 KBytes
  p=1.5 Mbps

- Service curve parameters:
  R (Flow Guaranteed rate)= 0.5 Mbps.
  R_{link}=1 Mbps
  L=L_{max} = 0.5 KBytes
  Total T_{lat}=$\frac{M+h\times L}{R} + \frac{h\times L_{max}}{R_{link}} = \frac{8\times(0.5+10\times0.5)\times10^3}{0.5\times10^6} + \frac{10\times0.5\times8\times10^3}{1\times10^6}$=128 msec

This is the latency per node. The end-to-end delay will be the same of these latencies plus $\frac{0.9-0.5}{0.5\times10^3}(\frac{1.5-1}{1.5-0.5}) = 0.4msec$ which results in total end-to-end delay equals to $\sum_{m=1}^{m=10} T_{lat_m} + 0.4 = 128.4msec$

If $p < R$, then the end-to-end delay is the sum of the nodes latencies = 128 msec.

- Flow 2 IETF arrival curve parameters:

  M=σ=0.5 KBytes.

  r=0.25Mbps

  b=0.9 KBytes

  p=1.5 Mbps

  R (Flow Guaranteed rate)= 0.25 Mbps.

- Traffic 2 latency calculations :

  T_{lat}=$\frac{M+h\times L}{R} + \frac{h\times L_{max}}{R_{link}} = \frac{8\times(0.5+10\times0.5)\times10^3}{0.25\times10^6} + \frac{10\times0.5\times8\times10^3}{1\times10^6}$=216 msec

Flow 2 end-to-end delay is:

$\frac{0.9-0.5}{0.25\times10^3}(\frac{1.5-1}{1.5-0.25}) + \sum_{m=1}^{m=10} T_{lat_m} = 0.64 + 216 = 216.64msec$

DiffServ Model

Equation (3.23) is used to find the maximum delay at the aggregation point, Ingress point, for this model. The parameters used for the arrival and service curves, and end-to-end delay calculations are:

- The leaky bucket arrival curve parameters:

  ρ_1=0.5 Mbps, ρ_2=0.25 Mbps and ρ_3=0.25 Mbps.

  σ_1= σ_2= σ_3=0.5 KBytes.

- Service curve parameters:

  R= 1 Mbps.

  T_{lat_1}=13.3 msec, T_{lat_2}=15.7 msec, T_{lat_3}=21.5 msec.

- Flows Guaranteed rates:
  g_1=0.4286 Mbps, g_2=0.3429 Mbps, g_3=0.2286 Mbps.

- The number of hops is 10

- The EF flows parameters are
  $v_1=\frac{\rho_1}{R}=\frac{0.5\times10^3}{1\times10^6} = 0.5 \times 10^{-3}$
  $v_2=\frac{\rho_2}{R}=\frac{0.25\times10^3}{1\times10^6} = 0.25 \times 10^{-3}$
  $v_3=\frac{\rho_3}{R}=\frac{0.25\times10^3}{1\times10^6} = 0.25 \times 10^{-3}$
  $\tau_1=\frac{\sigma_1}{R}=\frac{0.5\times8\times10^3}{1\times10^6} = 5.0 \times 10^{-3}$
  $\tau_2=\frac{\sigma_2}{R}=\frac{0.5\times8\times10^3}{1\times10^6} = 5.0 \times 10^{-3}$
  $\tau_3=\frac{\sigma_3}{R}=\frac{0.5\times8\times10^3}{1\times10^6} = 5.0 \times 10^{-3}$

- Flows delays calculations at the aggregate node:
  $D_1 = \frac{5+13.3}{1-(10-1)\times0.0005} = 18.378msec$
  $D_2 = \frac{5+15.7}{1-(10-1)\times0.00025} = 20.744msec$
  $D_3 = \frac{5+21.5}{1-(10-1)\times0.00025} = 26.5064msec$

- Flows end-to-end delays calculations is based on Equation (3.24):
  DiffServ end-to-end delay = $10 * D$
  Flow1 end-to-end delay = $10 \times 18.378 = 183.78msec$
  Flow2 end-to-end delay = $20.744 \times 10 = 207.44msec$
  Flow1 end-to-end delay = $26.506 \times 10 = 265.06msec$

Proposed IP QoS Model

The end-to-end delay calculation for the proposed model is based on Equation (3.25) and the following parameters are needed:

- The leaky bucket arrival curve parameters:
  ρ_1=0.5 Mbps, ρ_2=0.25 Mbps and ρ_3=0.25 Mbps.
  σ_1= σ_2= σ_3=0.5 KBytes.

Table 3.1: End-to-end delayed calculation using the three QoS models

Flow No.	IntServ	DiffServ	Proposed model
Flow 1	128.4 msec	183.78 msec	134.166 msec
Flow 2	216.64 msec	207.44 msec	158.458 msec
Flow 3	non	265.06 msec	217.1872 msec

- Service curve parameters:

  R= 1 Mbps.

  T_{lat_1}=13.3 msec, T_{lat_2}=15.7 msec, T_{lat_3}=21.5 msec.

- Flows Guaranteed rates:

  g_1=0.4286 Mbps, g_2=0.3429 Mbps, g_3=0.2286 Mbps.

- The number of hops is 10

- Flows end-to-end delays calculations

  $D_{flow_1} = \frac{\sigma_1}{g_1} + \sum_{m=1}^{10} T_{lat_{1_m}}$

  $D_{flow_1} = \frac{0.5 \times 8 \times 10^3}{0.4286 \times 10^6} + 10 \times T_{lat_1}$

  $D_{flow_1} = 1.166 + 10 * 13.3 = 134.166 msec$

  $D_{flow_2} = 1.458 + 10 * 15.7 = 158.458 msec$

  $D_{flow_3} = 2.1872 + 10 * 21.5 = 217.1872 msec$

Results Conclusion

Table (3.1) shows the end-to-end calculation for the three models. The IP proposed model achieves the lowest end to end delay bounds for the three flows compared to DiffServ and IntServ even without including the lookup delay bounds. For flow 1, the guaranteed rate was half of the link rate which results in lower delay when IntServ was used. This will allow other flows to have only half of the links bandwidth to share which causes traffic congestion. However the second flow results in higher delay bounds than the proposed model and DiffServ.

Chapter 4

An end-to-end QoS Management system proposal

4.1 Introduction

The proposed QoS management scheme incorporates the following elements that are commonly used to handle QoS requests. These elements will be involved in processing, monitoring, and controlling the traffic flows.
The QoS elements are [7]:

- **Admission Control:** Responsible for determining access to available network resources and keeping track of all reservations. This function is handled by the domain QoS manager.
- **Policing:** Performed by a QoS manager when a flow's actual data traffic exceeds the requested values given in the traffic specifications. In such cases the packets are dropped or downgraded to a best effort service class or marked as nonconforming.
- **Packet classification:** Identifies packets belonging to a specific flow and designates a QoS class for this flow. It is implemented at the edge points. A Packet

scheduler ensures that the flows identified by the packet classifier receive the requested QoS. It is also implemented at edge points.

- **Traffic control implementation:** Implements queuing methods, such as priority queuing and weighted fair queuing to control traffic at the domains' routers interfaces.

WFQ is used to separate the flows in which separate queues are assigned for each traffic flow. The proposed QoS management system uses the GDI and the Type of Service (Traffic Class (TC)) field for reserving and tracking traffic flows. This scheme is unique in that the IP network can be managed without invoking any other QoS protocols such as RSVP or MPLS. The QoS manager uses the GDI, that uniquely defines domain traffic flows, to trace and reserve resources. As the backbone routers use the GDI for forwarding decisions (MPLS label forwarding technique), the end-to-end delay is less when compared to the longest match procedure used by current forwarding schemes. This technique is scalable as the edge routers handle QoS requests and communicate with other QoS managers. The generating nodes do not negotiate requests with the QoS manager as in the case of the BB in the DiffServ domain. Traffic flows are classified based on the TC field so that each priority level is treated differently. The TC classification avoids the limitation of classes seen in DiffServ where traffic packets are mapped to pre-defined classes which results in limited QoS handling capability. Whenever a sender wants to send real time traffic, it sends a QoS request to the network edge router. Upon receiving requests from the sender, the edge router communicates with its domain manager to approve or reject these requests. The edge router forwards the manager's responses, either positive or negative, to the sender. When accepted, the source starts sending data packets to the edge router where traffic flow packets are classified, scheduled and monitored. Packets are queued depending on their TC field and the policies set by the QoS manager. The leaky bucket algorithm is implemented to police incoming traffic. The algorithm parameters for the accepted traffic are set up according to their traffic specifications. When a flow violates its requested specification, its priority level is degraded or its

packets are dropped. In the following sections, the QoS manager and the edge router are described.
This proposed QoS management scheme has introduced two new ideas:-

1. A QoS manager that uses the GDI in its data base to trace and reserve resources inside the domain network. Also, it is a new QoS technique that is implemented in IP networks without invoking other QoS methods.

2. Source hosts send QoS requests directly to the edge routers instead of the BB (as in DiffServ), and therefore the QoS managers communicate only with edge routers.

The proposed management scheme consists of two units, a QoS manager and an IP edge router. In the next section the proposed QoS manager is presented. This manager is based on the ARM bandwidth broker that uses dynamic allocation for reserving resources. A brief explanation on the ARM BB was given in Section 2.2.2. The data base of the ARM BB, which is a part of the manager's main units, will be modified to use the GDI for tracing the resources through the network it is managing. In Section 4.3, the proposed IP edge router that classifies, monitors and schedules incoming traffic flows packets is presented. This model is based on the QoSbox which was presented in Section 2.2.2. Finally, in Section 4.4 different reservation scenarios are explained in details.

4.2 Proposed QoS Manager

The manger proposed in this thesis manages IP networks and uses the GDI for controlling network domain resources. The following are the main model's blocks and their functions. Figure (4.1) shows these blocks.

1- **Request Processing Agent (RPA)**: Receives QoS requests for edge nodes and forwards them to the decision blocks. The requested parameters used for reservation decisions are, GDI, destination address, class of service and traffic

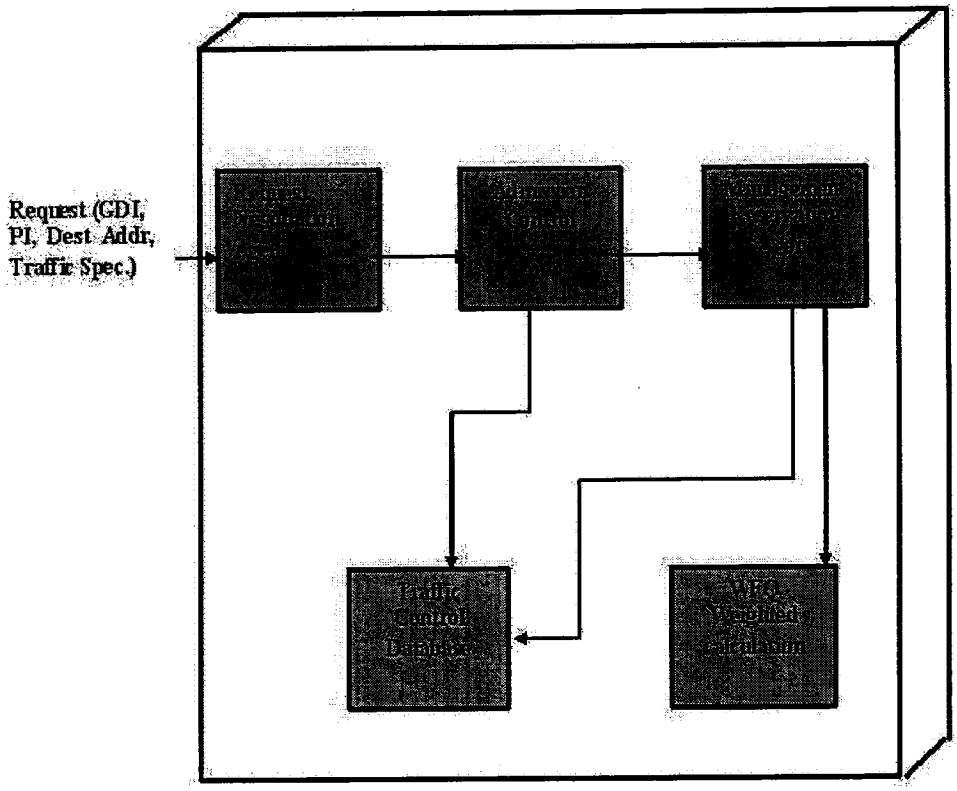

FIGURE 4.1: Proposed QoS management scheme structure

specifications (peak rate, average rate and burst rate). These parameters are then forwarded to the next block, admission control, for decision making.

2- **Admission Control Agent (ACA)**: Makes decisions on the requests received from the RPA. It starts first by checking the intended destination address to see if it is located in its domain or not. If the destination node is located in a different domain, it forwards the request to a neighbouring QoS manager. Otherwise, its resource availability has to be checked to accept or deny the request. The manger decision should not affect previous reservations.

3- **Management Reservation Agent (MRA)**: Manages all the reservations inside the domain by recording all the accepted requests using GDI and its

corresponding traffic specifications. It also sends the GDI and its corresponding traffic class for the accepted flows to the WFQ Weight calculation agent (WFQ-CA). Finally, this agent initiates policies to handle traffic violations that are implemented at the domain edge routers where traffic flows are monitored.

4- **Traffic Control Data Box (TCDB)**: Records all the QoS manager information, domain reservations, by using the GDI for tracing resources. It also handles the domain topology information which will be used to determine if the destination node is located in this domain or not.

5- **WFQ Weight Calculation Agent** : Calculates the weights for all the accepted traffic flows and sends them to the node queues that use them for classifying packets.

The following is the pseudo code for the QoS Management model:-

Request Negotiation Agent

{

Receives QoS requests (GDI, Prio, D_{IP}, CIR) from Edge router.

Forward QoS requests to Admission Control Agent

Forward Admission message (Answer) to the Edge router

Admission Control

{

/*Checking Priority level */

if(piro $\geq$ 8 && pir $\leq$ 15)

continue;

else

printf("Wrong Priority level setup");

/* Checking Destination IP address */

/* Destination IP address is sent to the Traffic Control Database */

if($Table_{entry} \rightarrow$ Destination == D_{IP})

```
{
  Flag1 =1;    /* indication of finding the IP address*/
  Forward GDI, CIR, Prio to the Management Agent
}
else
{
  Flag1 =0;
   printf("Destination node is not found in the domain");
   Send QoS Request to neighbouring QoS manager;
}
 /* Send the Traffic specification to the Management Agent */
 /* Response after receiving a message from the Management Agent */
if(message =='Yes')
{
   printf("QoS Request has been guaranteed resources");
   Answer ='YES'    /* Forward the answer to the Negotiation Agent*/
   Setup the policies to monitor traffic flows
   Send these policies to the Edge router where traffic flows are monitored.
}
else
{
   Answer ='NO'    /* Forward the answer to the Negotiation Agent*/
}
```

Reservation Management

```
{
/*Actions after receiving traffic specification from the Admission Agent */
  if(Flag1 ==1)
  {
```

if(Max_{BW}-Res_{BW} - CIR $\geq$ 0)

{

$Flag_2$ =1; /* indication of enough resource available*/

$database - Table_{entry} \rightarrow$ BW = CIR

$database - Table_{entry} \rightarrow$ FID = GID

$database - Table_{entry} \rightarrow$ ToS = $Prio$

Send Prio and GDI to WFQ Agent;

message ='Yes' ;

Go to the Admission Control Agent

}

else

{

$Flag_2$ =0; printf("No resources available");

message ='No' ;

Go to the Admission Control Agent

}

}

}

WFQ Weighted Calculation

{

/*Actions after receiving priority and GDI from Reservation Agent */

if(message=='Yes')

{

$T_{weight}=\frac{Prio}{\sum allpriorities}$ /* T_{weight} is the traffic weight that is used to schedule and queue it */

Broadcast all the new GDIs and T_{weight}s of all the traffic flows to all routers located in the domain

}

}

4.3 IP Edge Router

This section shows how the IP edge router is implemented since it is the second most important unit in this proposal. The IP edge router handles traffic monitoring, scheduling and classifying. In Section 4.3.1 there is a detailed explanation on the model's main units and their functions.

4.3.1 Architecture for the IP Edge Router Structure:-

The proposed edge router is designed to receive QoS requests from either connected hosts or other leaf routers. These requests are then forwarded to the domain QoS manager for decision making. The edge router also performs routing functionality by using the GDI to find the next hop or the intended destination node. The following are the main blocks of the IP edge router model. Figure (4.2) shows these blocks.

1. **Routing Blocks (RB):** Performs routing using the GDI for matching. The time taken to find the next hop using the GDI (flow lookup) is less than the normal header lookup. This results in a smaller processing time.

2. **Requests Forwarding Agent (RFA):** Forwards and receives QoS requests from/to the domain manager and all domain hosts. It also forwards new priority levels for degraded traffic flows to the manager to adjust the WFQ weights. In the case of transmission rate reduction in one of the accepted flows, the RFA sends the new rate to its domain manager to free more resources, resulting in dynamic adjustment of resources.

3. **Packet Processing Block (PPB):** Performs packet classification, scheduling and traffic monitoring. Accepted traffic flows are classified and scheduled based on the traffic specification of the accepted flows. The broadcast WFQ weights

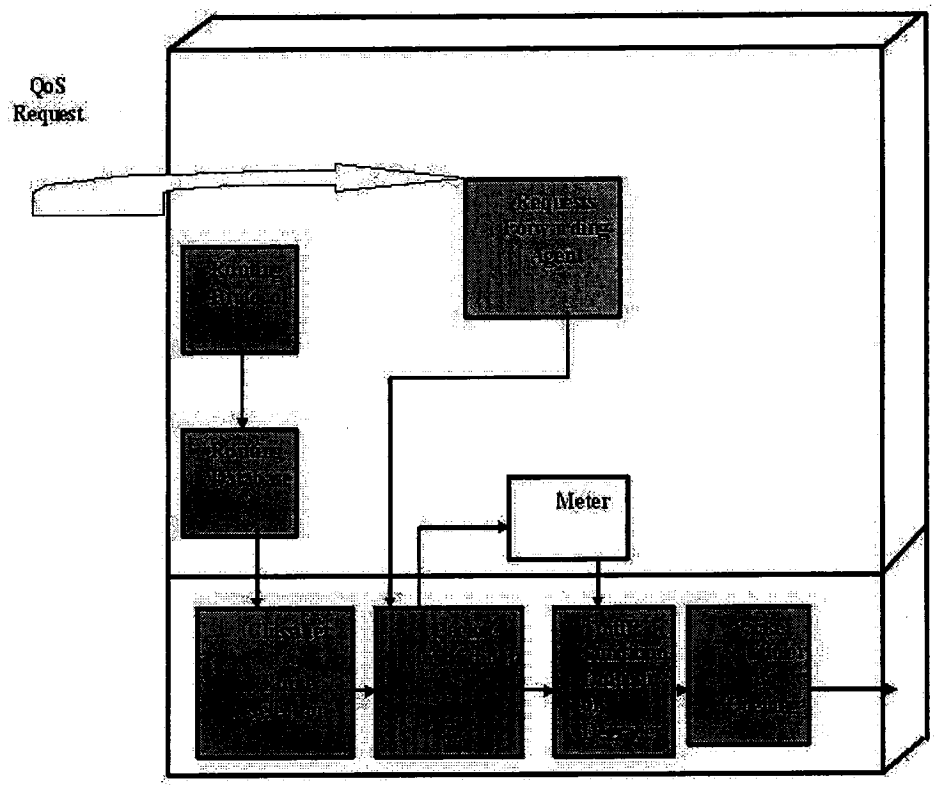

FIGURE 4.2: Proposed IPv6 Edge router structure

are used by each domain router along the path to buffer and queue the incoming packets according to the assured priority level. Traffic monitoring and conditioning is implemented in this router using standard algorithms such as leaky bucket. Packets that violate their specification are either dropped or degraded depending on the policies set by the QoS manager.

The following is the pseudo code for the proposed model Edge router:-

Request Forwarding Agent

{

Receives QoS requests (GDI, Prio, D_{IP}, CIR, T_{burst}) from either source nodes or leaf routers.

Forward QoS requests to the QoS management model

Forward Admission message (Answer) to the nodes that generate the QoS requests
if(Answer=='Yes') /* Positive response from the QoS manager */
{
Forward GDI, CIR, Prio, T_{burst}, D_{IP} to the packet classification.
Forward GDI, Prio, CIR, T_{burst} to the Traffic Condition.
{
else
Forward the response to the nodes that generate the QoS requests
}

Traffic Condition

{
if(T_{rate} > CIR)
{
if(Pior > 10)
Prio=Prio-1; /* Degrade the priority to a lower level */
else
Drop packets; /* Drop packets belonging to these priority levels*/
}
else
Continue monitoring traffic flows.
}

4.4 Reservation Scenarios

There are two cases for QoS reservations. The first is if a request has been issued from a local host to a destination node located in the same network domain. The second is if the destination node is located in an other domain.

4.4.1 Local Reservation

There are two possibilities in this case: 1) if the sending host is connected to the edge router, or 2) if it is connected to a leaf router. The procedures for these two cases are shown below.

I- Source Node Connected to the Edge Router:

Figure (4.3) shows the procedures followed to generate a reservation for a QoS request generated by a node connected directly to the edge router. The following are the

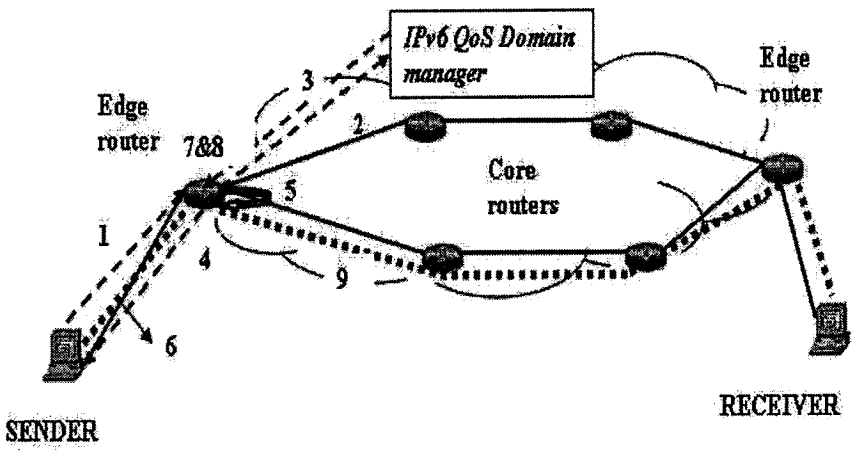

FIGURE 4.3: Procedures for requesting QoS for a node connected to an edge router

procedures involved during granting a reservation:-

1- A sending host generates a QoS request to the edge router. The request includes the data specification parameter such as average rate, committed information rate (CIR), burst rate. This request should also have the GDI and TC fields marked, and the IP address for the destination node. These will be used to locate the user, trace the reservation, and set the edge router scheduling parameters in case of request acceptance.

2- The edge router forwards this request to the network QoS manager for processing, and a decision is made depending on the availability of resources.

3- After taking a decision, the manager forwards this decision to the edge router that sent the request.

4- The edge router that receives the decision forwards it to the sending host that generated the request.

5- In the case of acceptance, the edge router extracts traffic flow parameters such as, traffic rate, priority level (TC field) to be used for classification purpose, and uses the GDI to schedule and to process packets that belong to this flow.

6- After receiving the acceptance message, the sending host starts sending traffic datagrams to the edge router.

7- The edge router monitors all incoming traffic flows and either drops or lowers the priority level of the traffic packets that violate the service parameters. The new priority levels are sent to the manager to re-adjust the WFQ weights.

8- The QoS manager has to set up a policy for reallocating the resources in the case of a drop in incoming traffic flow rate. The drop percentage and the watching period have to be set by the manager. The edge router monitors the incoming traffic flow and reports drops to the manager by sending the new rate, the GDI and the destination IP address.

9- Core routers forward the packets to the next hop using the GDI. This avoids more processing delay and layer violation in case of encrypted packets Appendix C. The GDI is only used to find the hop instead of the 5-tuples which include the port numbers. Packets are decrypted by the core routers to find the port numbers which are needed for the longest matchup procedure.

As a result of this proposal, QoS for non tolerant traffic flows can be achieved for the following reasons:-

1. Resources are reserved at the ingress point by the QoS manager.

2. Classification and scheduling are done based on the priority level (TC field) that has been marked by the source (application generates the traffic).

3. End-to-end delay is minimized since lookup procedure or longest matchup (forwarding packets to next hop) is done using a unique GDI.

4. DS mapping is avoided resulting in less processing time at the ingress node. Packets are processed faster than in the DiffServ domain where packets are processed and aggregated to pre-defined classes, and the DSCP are marked.

5. QoS request negotiations are done between the sending host and the ingress node. Negotiation is done between the sending host and the domain manager in the DiffServ involving a higher time cost.

II- Source Node Connected to a Leaf Router:

The procedures of reserving a QoS request generated by a host connected to a leaf router is shown in Figure (4.4). The procedures are similar to the previous ones except for the following steps:-

1. The source host sends its request to the leaf router that connects it with the network.

2. The leaf router forwards this request to the edge router which forwards it to the domain QoS manager.

3. When the edge router receives the reply for the request from the QoS manager, the edge router forwards this message to the leaf router. If the reply is positive, the request is accepted, and the edge router extracts the traffic flow parameters to be used for classification and monitoring. The leaf router forwards the reply message to the host that generated the request.

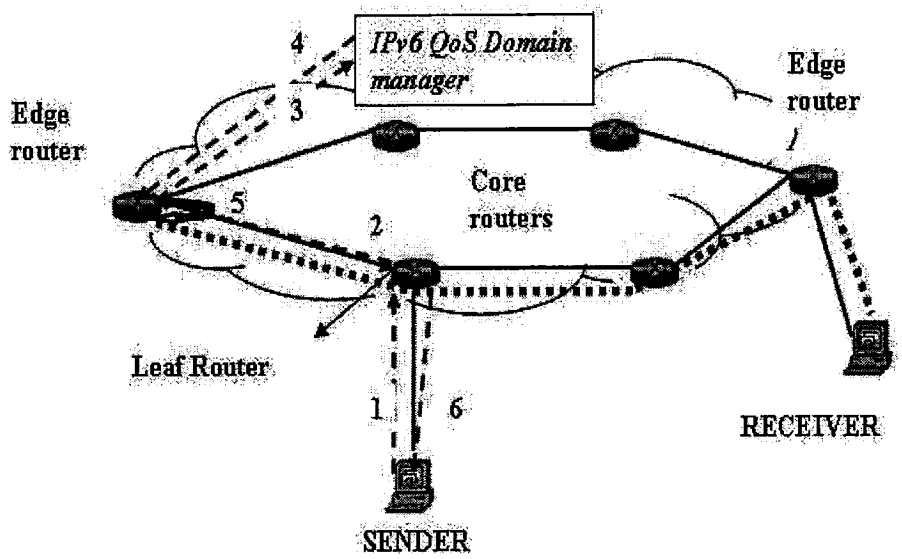

FIGURE 4.4: Procedures for requesting QoS for a node connected to a leaf router

4. Traffic data flows are sent to the leaf router first, then to the edge router for classification and scheduling. Scheduling and classification are not done at intermediate core routers.

4.4.2 Domains Reservation

This type of reservation is done when the destination node is located at a different network domain.

The following are the procedures followed when a sending host requests a QoS reservation for a receiving host located at a different domain. The procedures are shown in Figure (4.5).

1. A source host sends a QoS request to the network edge router. This message includes traffic specifications, IP address for the source and the destination nodes, the flow label and the traffic class fields.

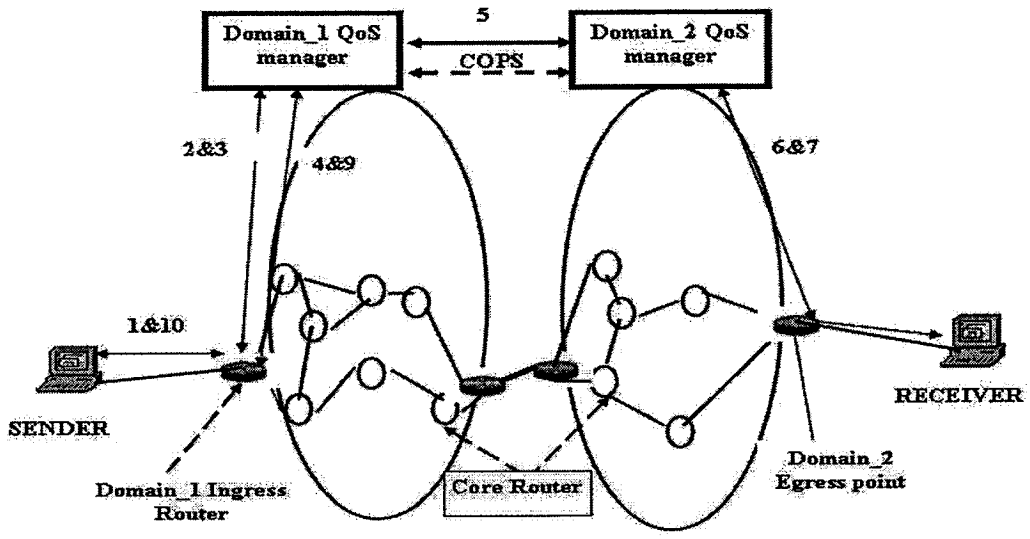

FIGURE 4.5: Procedures for requesting QoS for a receiving node located at different domain

2. The edge router forwards this request to the network QoS manager and waits for a decision.

3. The manager checks for resource availability and determines if the destination is located in its domain. This results in the following possibilities:-

   a- If there are resources and the destination is located at the same domain. A positive response is sent back to the edge node that forwarded the request.

   b- If there are no resources a negative response is sent back to the edge node.

   c- If there are resources and the destination is not located at the same network domain then the manager forwards the request to its neighbouring QoS manager.

4. The neighbouring QoS manager checks if the destination IP address is located in its domain and if there are resources available. It then decides either to accept, to reject, or to forward the request to the next QoS manager.

5. The neighbouring QoS manager, that administrates the destination host, forwards its decision, acceptance or rejection, to the QoS manager that initially sent the request. The first domain manager forwards the decision back to the edge router that initially sent it. In case of acceptance, the manager has to update its data base.

6. The edge router extracts the traffic flow specifications and the GDI to be used for classifying and tracing traffic flow packets.

7. The edge router forwards the decision to the node that generated the QoS request. If a positive decision is received, the host starts sending data traffic packets. Otherwise, the request has to be adjusted and resent again, or the traffic flow is sent and treated as best effort.

These are the scenarios considered in this research. Other QoS schemes are implemented to allow comparison with the proposed algorithm. Results are presented for comparison.

Chapter 5

Implementation and Experiments

In this chapter, the results obtained by testing QoS management in IPv6 networks are presented. Different scenarios are designed for testing different aspects of IPv6 QoS using the network simulator (ns-2). NS-2 is an event driven simulator that simulates a variety of IP networks. It is written in C++ and OTCL languages. It is used in these experiments since it is a well known simulator and used by most researchers. The simulation starts by simulating an IPv6 network that uses flow labels and traffic class (TC) fields to handle non tolerant traffic flows (delay and packet loss sensitive traffic flows). Two queuing mechanisms, WFQ and CBQ, are generally used to schedule and process traffic flows packets. In Section 5.1 the choice of WFQ for this research is justified by experiments. In Section 5.2, the ns-2 code has been modified to implement the proposed IP QoS management scheme. Different simulation scenarios have been run to see how this scheme reacts for different traffic flow types under different networking conditions. Then the results obtained are compared with IntServ, DiffServ and MPLS techniques applied for the same network topology with the same traffic flows specifications. End-to-end delay and packet loss were used as the main parameters for the comparison. At the end of this Chapter, multiple domains simulation scenarios is presented to test the flexibility and scalability of the proposed QoS management model. The topology used for the two and three domains simulation is similar to the topology used in the single domain simulation.

The traffic flows, tested in these simulations, are real traffic flows where packets are dropped and delayed. In contrast, traffic flows used for numerical calculation in Chapter 3 are assumed to be lossless, no packet loss.

5.1 Choice of queue simulations

In this section, a network topology that has a sending node, $n0$, two receiving nodes, $n7$ and $n8$, two edge nodes, $n1$ and $n6$ and four core nodes, $n2$, $n3$, $n4$ and $n5$ is built using ns-2. The network layout is shown in Figure (5.2) [76]. The sending

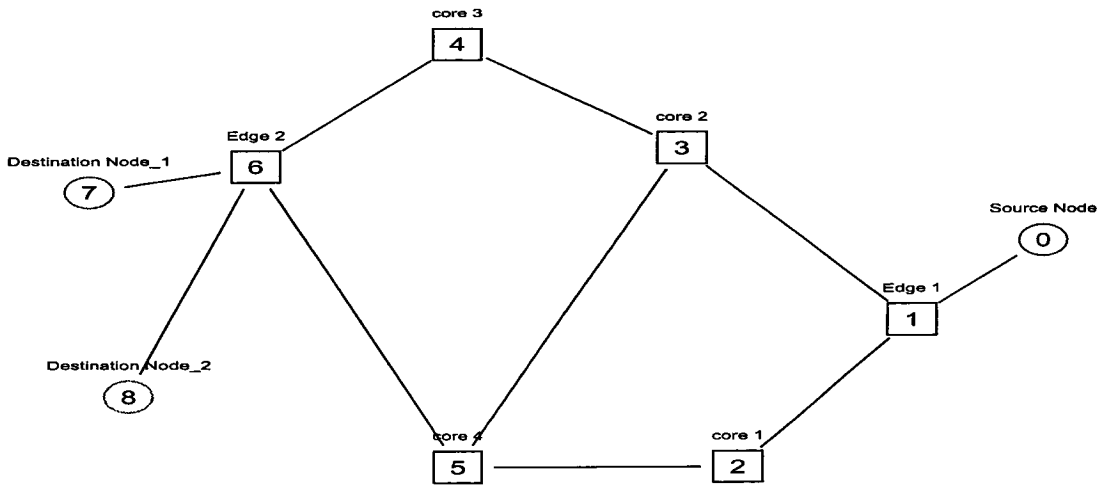

FIGURE 5.1: NS-2 network layout used for testing IPv6 QoS

node generates four traffic flows, two for each receiving node. The generation rate set for these traffic flows is 500 kbps and the packet size is set to 500 bytes. Two of these flows, Flow 2 and Flow 3, are non tolerant traffic type with higher priority marked fields. The other two are data traffic flows with lower priority level. WFQ and CBQ are used, in different simulations, as the nodes' outgoing queues since they are most effective [72] [73] when different traffic data flows are processed. The queues' weights are set to 40% for the higher priority and 10% for the lower priority. The links bandwidth and delay parameters are set to 1 Mbps and 10 msec. The exception is the link that connects $n0$ and $n1$ which is set to 1.5 Mbps. Two sets of experiments

Table 5.1: Average End-to-end delay during normal rates comparison

Flow No.	IPv6 using WFQ		IPv6 using CBQ	
	Average	STD	Average	STD
Flow 1	71.1143 msec	0.1945	70.9381 msec	0.13858
Flow 2	70.71815 msec	0.1378	71.02122 msec	.214684
Flow 3	70.7795 msec	0.196913	70.7197 msec	0.1812
Flow 4	71.02084 msec	0.286167	71.04162 msec	0.167125

Table 5.2: Average End-to-end delay during source 2 rate increase comparison

Flow No.	IPv6 using WFQ		IPv6 using CBQ	
	Average	STD	Average	STD
Flow 1	93.7742 msec	12.07764	71.60756 msec	0.307918
Flow 2	71.2224 msec	0.27702	93.4432 msec	11.1128
Flow 3	71.3431 msec	0.25670	71.095 msec	0.18278
Flow 4	87.4412 msec	13.9326	71.6547 msec	0.29097

have been simulated, one when all the flows are summing up to 1 Mbps and the other one when Source flow rate increases to 0.5 Mbps.

5.1.1 Simulated Results Analysis

Table (5.1), Table (5.2) and Table (5.3) show the average simulation results. The following conclusions can be made:-

1- IPv6 simulation using CBQ and WFQ schemes achieved lower end-to-end delay for all the flows during normal rates, all the rates are 0.25 Mbps.

2- WFQ simulation achieved low end-to-end delay for Flow 2 and Flow 3, non tolerant traffic sources, when source 2 rate increase, network congestion. However, Flows 3 and 4 have been affected and their packets have been delayed more.

3- CBQ simulation, during source 2 rate increase, resulted in high end-to-end delay

Table 5.3: Packet Loss Rate comparison

Flow No.	IPv6 using WFQ		IPv6 using CBQ	
	Average	STD	Average	STD
Flow 1	0%	0	0%	0
Flow 2	0%	0	0%	0
Flow 3	0%	0	0%	0
Flow 4	0%	0	0%	0

for Flow 2 during its rate increase. The other flows were not affected and they achieved low delay as the case of normal rates.

4- No packets have been dropped from all the sources during the two scenarios.

Therefore, non-conformant non-tolerant traffic flows, during network congestion, have been delayed during CBQ simulation. These flows, non-tolerant flows, were not affected during WFQ simulation by the non-conformant scenario and the other flows, best effort were delayed more.

Based on the results, WFQ was chosen to be implemented for the proposed IPv6 QoS management scheme since it achieves lower delay for the high priority traffic flows during network congestion.

5.2 The Implementation of the IPv6 QoS management scheme:

5.2.1 The Model Layout

The IPv6 QoS management scheme that was presented in Section 4.2 is implemented using the ns-2 simulator. The implementation of this model is shown in Figure (5.2) [62]. The following is a brief explanation of the procedures shown in Figure (5.2):

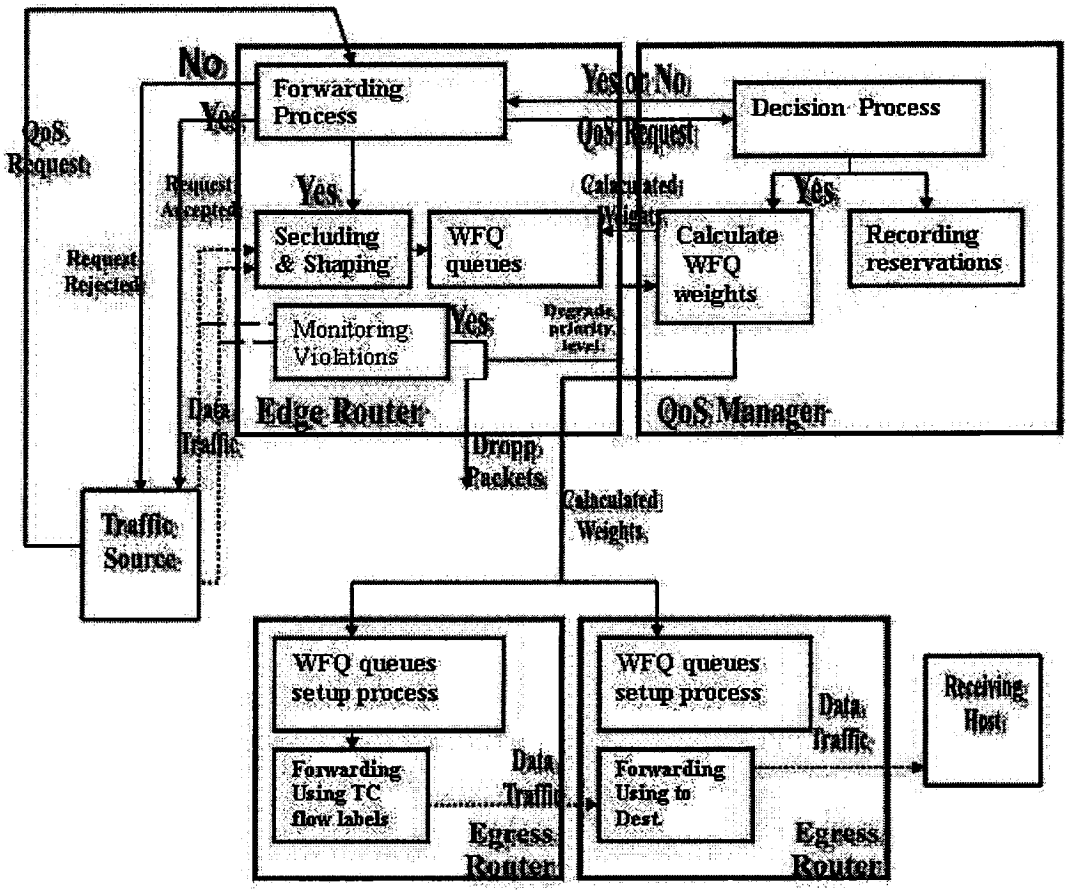

FIGURE 5.2: Proposed IPv6 QoS management scheme implementation on ns-2

[1] The traffic source generates a QoS request and sends it to the Requests Forwarding Agent (RFA) located at the Ingress router.

[2] The Ingress router's RFA forwards this request to the Request Processing Agent (RPA) at the QoS manager where a decision is to be taken after checking resources.

[3] A Yes or No answer is sent to the edge router. In the case of a Yes answer, traffic specifications, bandwidth requested and priority level, are recorded in

the data base by using the GDI. Also, the TC field is extracted to be used for the WFQ weights calculations. These weights will be broadcasted to all WFQ queues in the domain.

[4] The Ingress router's RFA forwards the QoS manager answer to the traffic source. In the case of a Yes answer, the traffic parameters sent with the request are extracted to be used for scheduling traffic packets entering the domain. Also, the router queue (WFQ) is set up since its weights are calculated and broadcasted.

[5] Upon receiving the answer to the request, the source node has two options:

I- If the answer is Yes, the source node starts sending data to the Ingress node at the rate and specification agreed upon.

II- In the case of a No answer, the source node application either changes the traffic specification and tries again, or sends the traffic as Best Effort traffic.

[6] The edge router monitors the incoming traffic by adding a token bucket used to implement the QoS manager policies. Packets that violate their traffic specifications will be punished depending of the policies set by the manger.

[7] Two actions are to be used in this model for any non-conformant traffic flow:

⋆ Drop low priority packets.

⋆ Reduce their priority level if they are high priority type. The new priority levels will be sent to the manger to recalculate the WFQ weights if there are no other flows with this priority level. Otherwise, the packets that have this priority will be delayed more, or even dropped in the case of link congestion.

5.2.2 NS-2 Code Implementation

NS-2 Implementation for IPv6 QoS Manager

The ARM bandwidth broker is the latest BB implemented in the ns-2 simulator for DiffServ simulations. It was modified to act as an IPv6 QoS manager since it dynamically re-allocates unused resources. The ns-2 source is shown in Appendix D. The following are the modifications proposed to the available ARM bandwidth broker:

I Receiving requests come from the edge routers, not from sending hosts. Requests processing will be handled by the QoS manager and the edge router. Therefore, requests are sent from the sending nodes to the edge routers. Traffic parameters for the accepted requests will be extracted at the edge routers for scheduling and shaping purposes instead of receiving them from the domain BB as in the DiffServ scenario. The GDI is used for processing requests which results in lower delay than in the DiffServ.

II The QoS manager Database uses the GDI to record and trace current reservations. The BB database uses IP address, port numbers, and protocol number to identify traffic flow packets (Section 2.2.1).

III Changes in traffic request specifications such as average rate and maximum burst, have to be reported to the manager by the edge routers since traffic flows are monitored at the edge point.

IV QoS policies will use the TC (priority level) field as the main parameter for action in case of traffic violations. For example, higher priority traffic flows will be dropped to lower priority level and low priority traffic flows will be discarded. This is unlike DiffServ Assured service where packets are dropped if they violate their traffic specification, even if they contain non tolerant real time traffic with high priority.

NS-2 Implementation for IPv6 Edge Router

The ns-2 simulator's QoSbox was modified to act as IPv6 edge router. In the next section a simulation comparison between the QoSbox originally, designed for DiffServ, operate on traffic processing, and the proposed IPv6 management scheme will be shown. NS-2 source code is shown in Appendix D.
The following are the major proposed modifications to QoSbox:-

1. Change the classification and shaping to have rather than four traffic classes to allow more differentiation between traffic flows. This will achieve fairer treatment for packets with different priority levels. Also, only packets with lower priority are dropped during congestion. The number of class buffers will depend on the priority level (TC field) and the demanded bandwidth. Therefore, incoming packets are not mapped resulting in fair and fast treatment at the Ingress domain entries.

2. The modified QoSbox router receives QoS requests from hosts connected to it, or from network leaf routers, and sends the requests to the domain QoS manager for reservation decisions. In the case of available resources, a positive response is received from the manager. A confirmation is sent back to either the generating node or the leaf router. The traffic specifications for the accepted requests are extracted from them to be used for scheduling and classification procedures. If no resources are available, a rejection response is sent back to the source node.

3. The router has to update the QoS manager with traffic flow rates. In the case of a drop in bandwidth demand for one of the traffic flows spare bandwidth is reallocated. The percentage of drop is a policy set by the QoS manager.

4. WFQ and Class-Based Queuing (CBQ) will be implemented and tested at the edge router. A mapping policy is introduced to assign queue weights using the priority level and the requested bandwidth.

5.2.3 Simulation Setup and Results

Three Traffic Flows Simulation

Simulation setup

This model was tested on a network topology that has different traffic sources, edge routers to handle QoS requests with the involvement of the QoS manager. The test network is shown in Figure (5.3). The network consists of 11 nodes. Nodes 6, 7, 8 are the sending nodes that generate traffic flows with different priority levels. Node 6, the Highest Priority node, generates CBR (Constant Bit Rate) traffic flow with priority level set to 15 (highest priority level) and the rate (Committed Information Rate (CIR)) is set to 0.5 Mbps. Node 7, Source 2, generates CBR traffic with priority level set to 12 and its CIR is set to 0.25 Mbps. Node 8, Source 3, generates best effort traffic with the lowest priority level, 8, and its CIR is set to 0.25 Mbps. Node 0 is the edge router that connects the three traffic sources. Each traffic flow assigned a flow ID to identify it and to be used for reservation and routing purposes. The following are the assignments for the flow IDs for the traffic flows involved in the simulation:-

* **Flow ID 15** for the Highest Priority (node 6) traffic flow packets.
* **Flow ID 12** for the Source 2 (node 7) traffic flow packets.
* **Flow ID 8** for the Source 3 (node 8) traffic flow packets.

Node 0, edge 1, receives the request for bandwidth (CIR rate) and priority levels from traffic sources, Highest Priority and Source 2. Then these requests are forwarded to the network manager. Traffic scheduling and shaping for the incoming traffic flows are also done at this node. The links are set up for 1 Mbps throughput and 1 msec propagation delay. The packet size for all traffic flows is set to 500 bytes. Two simulation scenarios are presented in this section. In the first scenario, traffic flow generated from Source 2 is set to have 30% non-conformant factor, a rate of 0.325 Mbps instead of 0.25 Mbps. This traffic flow exceeds its profile which results in enforcement of QoS manager policies by degrading packets belonging to this flow to

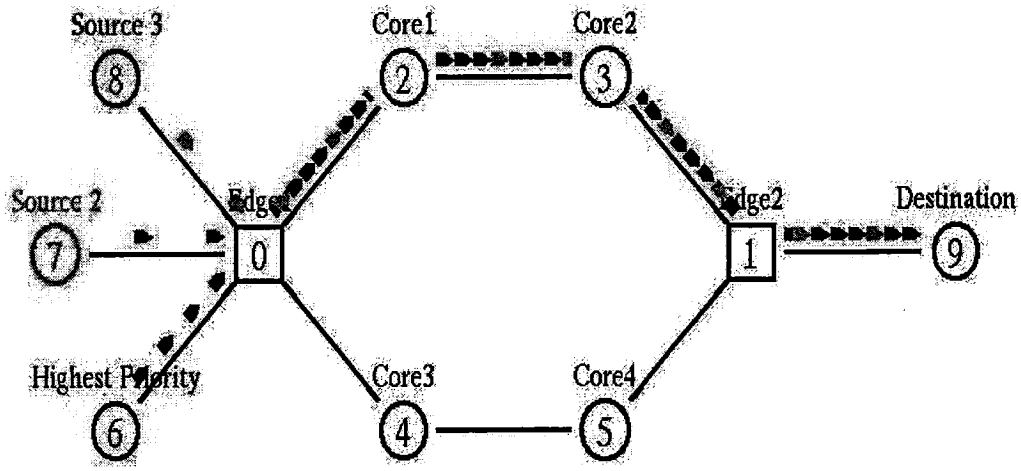

FIGURE 5.3: Testing the proposed scheme by simulating a simple network

best effort. This results in link congestion since total traffic flow rates exceed the link capacity causing best effort packets to be dropped at the ingress router. Some of the degraded packets belonging to Source 2 are also dropped since they are now best effort type.

In the simulation, the token bucket algorithm is implemented to monitor all the traffic flows. The following policies are set for this simulation model:-

- Degrade of Highest Priority traffic source to level 12 from level 15.
- Degrade Source 2 traffic to best effort class type from level 12.
- Drop the best effort traffic in case of violation or congestion.

Figure (5.4) shows how the QoS manager degrading the traffic priority down to the level set by the manager and enforced by the token bucket module. It also shows how messages are created during acceptance of requests from Source 2 and Highest Priority nodes.

The QoS manager accepting requests by comparing the remaining bandwidth and the requested bandwidth sent initially for each QoS request independetely. It sends acceptance messages to node 0 (edge node) if the requested bandwidth is equal or less

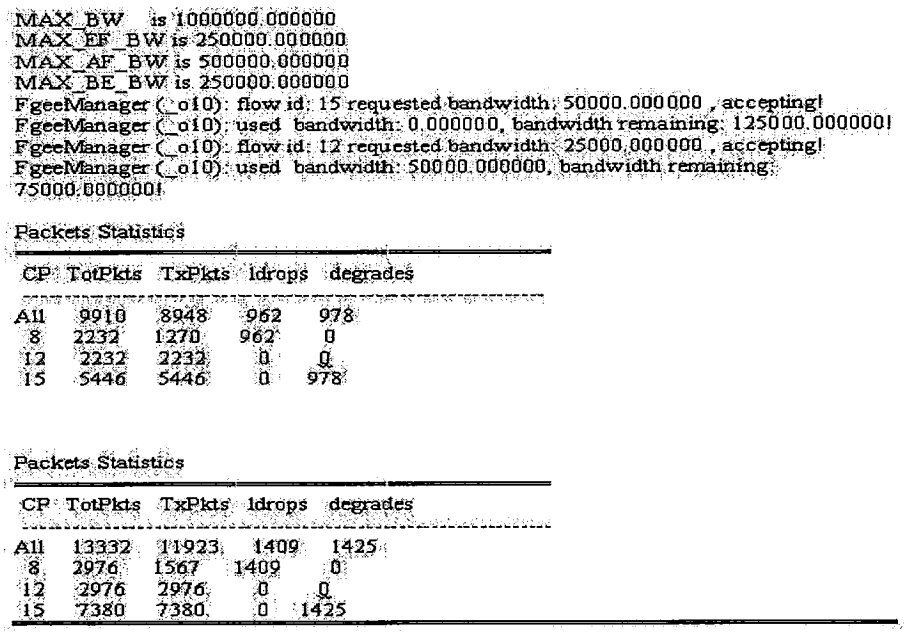

FIGURE 5.4: Sample output run showing Acceptance messages and some statistical results showing flow ID, total packets, dropped packets and degraded packets

than the remaining bandwidth. Then these messages are forwarded to the Highest Priority source and Source 2 nodes in this simulation scenario. The manger also calculates the WFQ weights to differentiate degraded packets during processing at the nodes' queues. Priority levels are used to find the weights, that represent the percentage of time the queue processor spent processing packets. These weights are broadcast to all backbone nodes' queues.

The second scenario, measures QoS performance for the same topology when the Highest Priority traffic flow increases by 30% of its rate resulting in a new rate of 0.650 Mbps. The same policies were used which means this flow's packets were degraded to level 12. This affected the other two flows since the number of packets that belong to priority 12 increases and links become congested resulting in packets to be dropped.

Table 5.4: End-to-end delay, packet loss rate and degradation rate comparison results for three traffic sources during violation and non-violation scenarios

Traffic Source No.	Flow ID	Ave delay		P. Degraded rate		loss rate	
		conformant	non	conformant	non	conformant	non
H. Priority	15	13.714ms	14.601ms	0%	19.31%	0%	0%
Source 2	12	13.59 ms	53.709ms	0%	10.75%	0%	8.62%
Source 3	8	18.346 ms	102.448ms	0%	0%	0%	8.094%

Simulated Results Analysis

Table (5.4) shows the results obtained from simulating the proposed algorithm and testing conformant and non-conformant scenarios for each traffic flow. Four simulation scenarios are performed to test how the proposed IPv6 QoS manager acts during conformant and non-conformant traffic flows. In addition, to test all the three flows when they are conformant to their rates, the Highest Priority source and the Source 2 traffic flows are tested during non-conformant case. The QoS parameters, end-to-end delay and packet loss rate, are measured for all the scenarios mentioned and the effects of rates increase to the other flows are also observed. These two QoS parameters will be used to measure the performance of all the QoS techniques used in this thesis allowing direct performance comparison with the proposed IPv6 QoS manager. The percentage of degraded packets was also measured. The following conclusions are made from simulating the proposed IPv6 QoS manager:

A- Conformant Traffic Flows

I All the traffic flows have lower end-to-end delay, since traffic rates did not exceed their initial rates. However, flow ID 15 and 12 achieved lower end-to-end delay than flow ID 8 since their priorities are higher.

II No packets neither dropped nor degraded for all the three traffic flows for the same reason mentioned early.

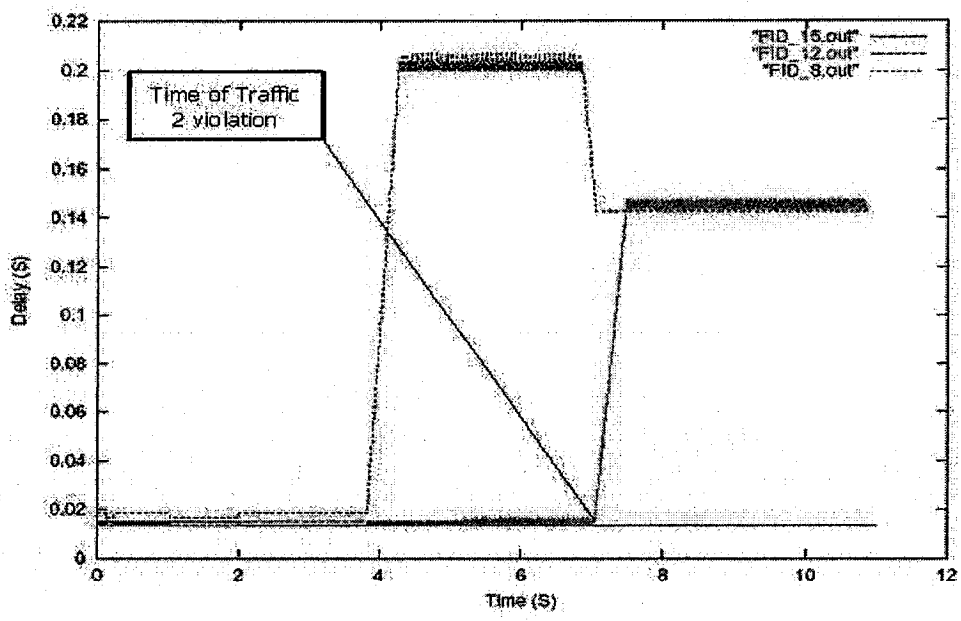

FIGURE 5.5: Delay for the tested three traffic flows during Source 2 non-conformant traffic flow packets

B- Non-Conformant Traffic Flows

Each traffic flow is tested when its flow rate increases by 30% of the initial rate.

I Traffic flow ID 15 achieved 14.601 msec average end-to-end delay and no packets have been dropped. However, 19.31% of the flow's packets have been degraded to priority level 12.

II- Traffic flow ID 12 achieved 53.709 msec average end-to-end delay and 8.62% of the flow's packet have been lost. In addition, 10.75% of the total packets were degraded to best effort.

III- Best effort traffic has been effected by both flows during non-conformant scenarios. In case of the flow ID 15 non conformant simulation, the best effort flow loss rate was 93.217% and the end-to-end delay was 222.448 msec. In the case of flow ID 12 non-conformant simulation, best effort loss rate was 8.094% and end-to-end delay was 102.448 msec.

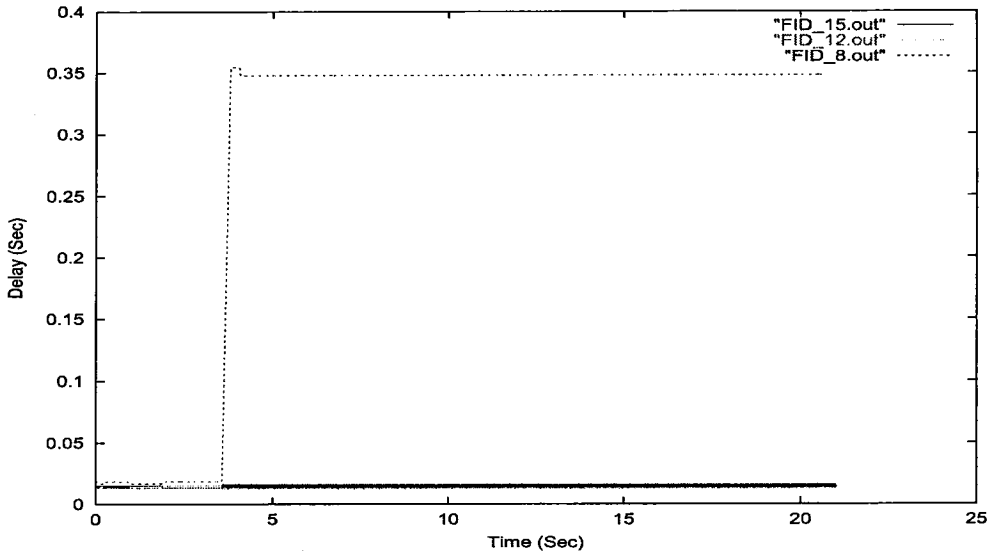

FIGURE 5.6: Delay for the tested three traffic flows when all the three flows are conformant

Also, during the Highest Priority source non-conformant traffic simulation, 8.62% of Source 2 traffic packets are dropped, however, lower end-to-end delay achieved for Source 2's packets.

Figure (5.6) shows the delay graph for all the traffic flows during the conformant simulation scenario. Higher priority traffic flows, flow IDs 15 and 12 experience lower average end-to-end delay than the flow ID 8, best effort, which experience the highest delay.

Figure (5.5) gives a clear picture of packets end-to-end delay for all the three traffic flows during flow ID 12 traffic rate increase. Flow ID 8 packets delayed the most. Traffic flow ID 12 packets started with a lower delay untill it reaches non-conformant stage, traffic rate increases, at time t=7 sec. At this time packets are dropped to best effort resulting in a higher end-to-end delay. The QoS manager policy is set to drop traffic flow ID 12 to best effort if the original rate is exceeded. Highest Priority traffic flow packets, flow ID 15, are delayed the lowest since they received the highest WFQ weight compared to the other two flows.

In summary, the proposed IPv6 QoS management model performed very well and excellent results have been achieved for non-tolerant traffic flows with high priority set up. In addition, if traffic flows exceeds their initial traffic rates, non-conformant, their priorities are dropped to a lower level which resulted in a higher delay and a higher loss rate.

Six Traffic Flows Simulation

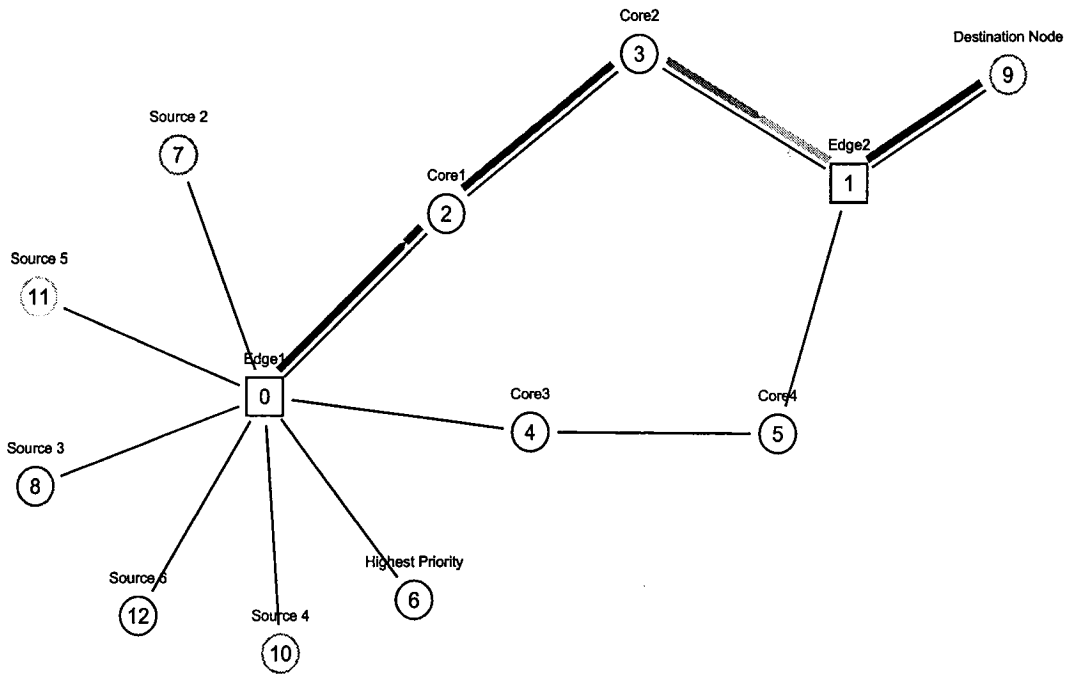

FIGURE 5.7: Six sources network topology

The topology used in this simulation has six traffic sources instead of the three traffic sources used previously to test how the proposed QoS management model reacts when there are more traffic sources. Three priority levels are used, high, medium and low.

- ⋆ **High Priority:** Priority for the Highest Priority packets are set to 15 and Source 2 packets are set to 14.

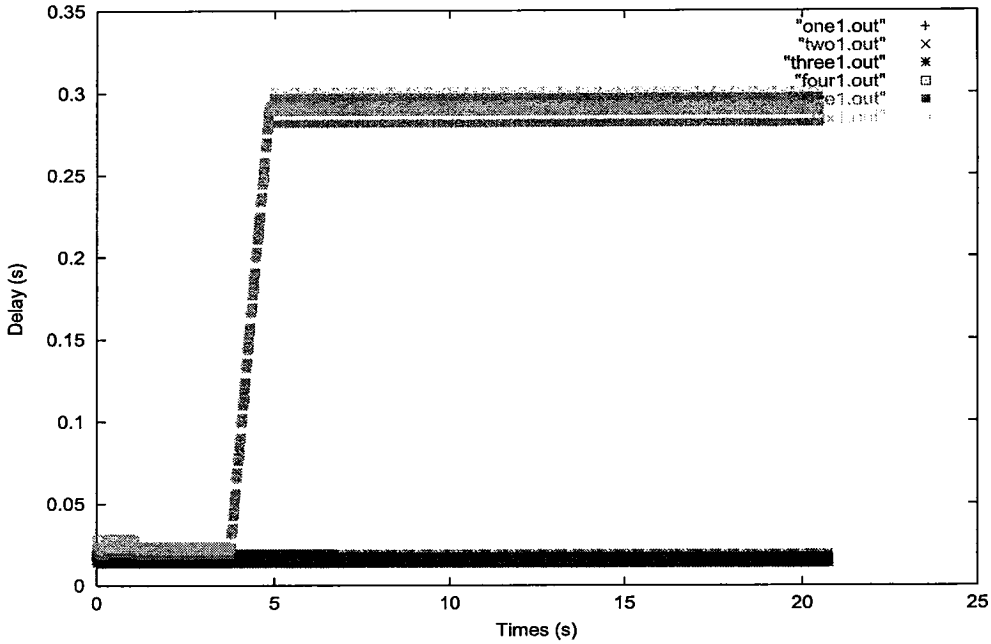

FIGURE 5.8: End-to-end delay for Six sources simulation

- ⋆ **Medium Priority:** Source 3 and Source 4 traffic flows with priority levels set to 12 and 11 respectively.
- ⋆ **Low Priority:** Sources 5 and Source 5 traffic flows with priority levels set to 9 and 8 respectively.

The network topology is shown in Figure (5.7).

Table (5.5) shows the simulation results of the six sources scenario. The high and medium priority traffic flows achieved low end-to-end delay. The low priority traffic flows achieved higher delay than the other two levels as shown in Figure (5.8). No packets have been dropped from all the six flows. During non-conformant tests, packet are dropped and delayed more from the low priority traffic flows. Packets belong to the other two classes, high and medium are degraded to lower classes only. The simulation results proves that the proposed QoS model scales well even if the number of traffic flows doubled. In addition, high and medium priority traffic flows achieve lower delay and 0% loss rate during congestion, traffic non-conformant.

Table 5.5: End-to-end delay, packet loss rate results for six traffic sources simulation scenario

Traffic Source No.	Flow ID	Ave delay	Packet loss rate
H. Priority	15	13.58ms	0%
Source 2	14	15.26 ms	0%
Source 3	12	15.15 ms	0%
Source 4	11	17.798 ms	0%
Source 5	9	21.08 ms	0%
Source 6	8	24.846 ms	0%

5.3 Simulation of Other QoS Models

In this section the QoS performance of the network topology used in Section 5.2 was tested under the IntServ, the DiffServ and the MPLS QoS models. In each simulation scenario the network components have to support different QoS algorithms depending on the scheme used.

5.3.1 RSVP Simulation Results

Topology Setup

An RSVP contributed patch was used to implement RSVP [78] in ns-2. All the nodes used in Section 5.2 simulation including the destination and source nodes become RSVP agents and support RSVP. The links between nodes are set to RSVP duplex links with Param, RSVP admission control model and NULL scheduler defined by the simulator. The link parameters are set to 99% of the link's utilization is set for reserved traffic, 200 bytes for RSVP signaling messages and 5000 bytes best effort queue size. The flow IDs assigned for the traffic flows are, 100 for Highest Priority, 200 for Source 2 and 300 for Source 3 traffic flows packets. Two sessions, A and B are set for the two traffic flows, flow ID 100 and 200. Session A is reserved for the Highest Priority traffic packets, flow ID 100, and is intended for *Destination*

1. Session B is reserved for Source2 traffic packets, flow ID 200, and intended to *Destination* 2. Changing path messages for these two sessions are set to start at 1.0 sec for the Highest Priority source and 1.5 sec for Source 2. The first RES message is set to be sent at 2.0 sec from *Destination* 1 to the Highest Priority source that iniate the PATH message. The other RES message is set to be sent at 3.0 sec from *Destination* 2 to Source 2 in response to its PATH message. The Highest Priority source starts by sending data packets at 2.5 sec and Source 2 at 3.5 sec after receiving the reservation messages. Change of PATH & RES messages for these two sessions is shown in Figure (5.13). Figure (5.9) shows the simulated network topology and how traffic packets travel from the source nodes to the destination nodes.

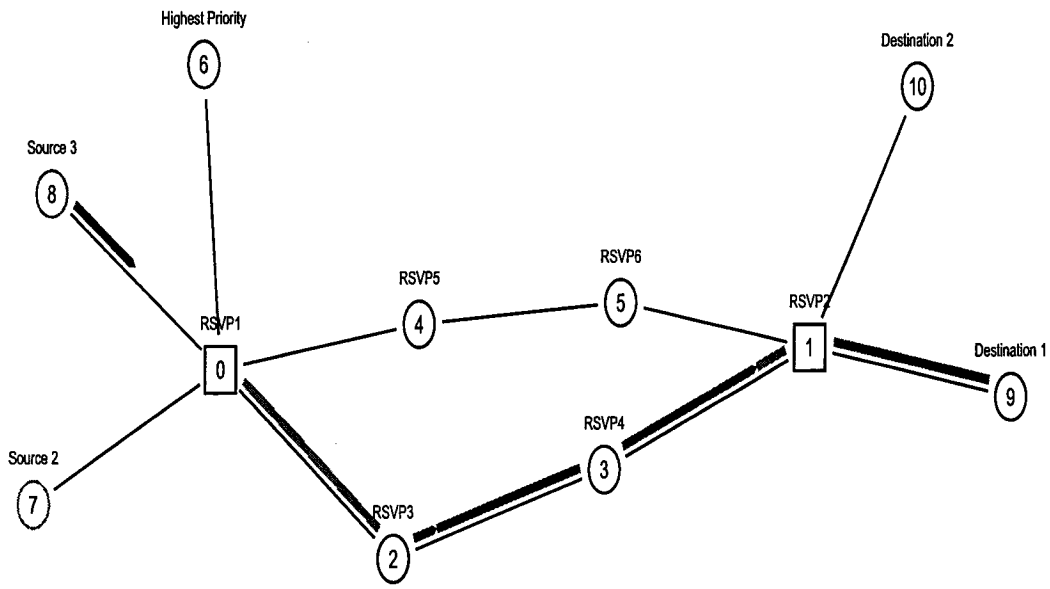

FIGURE 5.9: Network topology for RSVP simulation

Simulation Results

The simulation results are summarized in Table (5.6). Two simulation scenarios are performed to test QoS performance during conformant and non-conformant traffic flows.

```
0  PATH EVENT at 1.002 : SID: 0 RATE: 500000 BUCKET: 500000 SENDER: 6
3  PATH EVENT at 1.006 : SID: 0 RATE: 500000 BUCKET: 500000 SENDER: 6
1  PATH EVENT at 1.008 : SID: 0 RATE: 500000 BUCKET: 500000 SENDER: 6
9  PATH EVENT at 1.009 : SID: 0 RATE: 500000 BUCKET: 500000 SENDER: 6
0  PATH EVENT at 1.502 : SID: 1 RATE: 300000 BUCKET: 300000 SENDER: 7
3  PATH EVENT at 1.506 : SID: 1 RATE: 300000 BUCKET: 300000 SENDER: 7
1  PATH EVENT at 1.508 : SID: 1 RATE: 300000 BUCKET: 300000 SENDER: 7
10  PATH EVENT at 1.509 : SID: 0 RATE: 300000 BUCKET: 300000 SENDER: 7

2.00    1  RESV EVENT at 2.002 : SID: 0 RATE: 500000 BUCKET: 500000 SENDER: 6
3  RESV EVENT at 2.004 : SID: 0 RATE: 500000 BUCKET: 500000 SENDER: 6
0  RESV EVENT at 2.008 : SID: 0 RATE: 500000 BUCKET: 500000 SENDER: 6
6  RESV EVENT at 2.010 : SID: 0 RATE: 500000 BUCKET: 500000 SENDER: 6
1  RESV EVENT at 3.002 : SID: 1 RATE: 250000 BUCKET: 250000 SENDER: 7
3  RESV EVENT at 3.004 : SID: 1 RATE: 250000 BUCKET: 250000 SENDER: 7
0  RESV EVENT at 3.008 : SID: 1 RATE: 250000 BUCKET: 250000 SENDER: 7
7  RESV EVENT at 3.010 : SID: 0 RATE: 250000 BUCKET: 250000 SENDER: 7


Delays when source 2 violates:

Prio  15 had 5502 packets total
0 packets were degraded (0%)
Average delay: 15.846703017085 4ms
Max delay: 16.120000000008ms
Min delay: 13.399999999999ms

Prio  12 has 2604 element in it
0 packets were degraded (0%)
Average delay: 1692.34001305684ms
Max delay: 3694.92ms
Min delay: 13.782ms

Prio  10 has 2676 element in it
0 packets were degraded (0%)
Average delay: 17.872466367711 9ms
Max delay: 18.440000000018ms
Min delay: 13.399999999999 9ms
```

FIGURE 5.10: PATH and RES messages printed out during RSVP simulation

A. Conformant Traffic Flows

- ⋆ Traffic flows ID 100 and 200 achieved 0% loss rate and lower end-to-end delay than flow ID 300.
- ⋆ Traffic flow ID 300 achieves also 0% loss rate.

B. Non-Conformant Traffic Flows

- ⋆ Traffic flows ID 100 and 200 achieved 0% loss rate and a very high end-to-end delay when their rates increase.
- ⋆ Traffic flow ID 300 achieves 24.17% loss rate, however, the measured end-to-end delay was lower than flows ID 100 and 200 because the extra packets are dropped not queued.

Path setup delay, that includes changing of PATH and RES messages during RSVP path setup, has to be added to the average end-to-end delay shown in Table (5.6).

Table 5.6: RSVP simulation results for End-to-end delay and packet loss rate

Traffic Source No.	Priority Level	Flow ID	Ave delay		loss rate	
			conformant	non	conformant	non
Highest Priority	15	100	15.846msec	160.499 msec	0%	0%
Source 2	12	200	14.04 msec	169.234msec	0%	0%
Source 3	Best Effort	300	17.872msec	136.97 msec	0%	24.168%

This delay was 1.5sec and is added to each recorded average delay.

Figure (5.11) shows the delay diagram for the traffic flows when Source 2 exceeds its Tspec. As shown packets belonging to this flow are delayed the most during non-conformant, transmission rate increase. Source 3 traffic packets recorded higher delay before the increase and as soon as the Source 2's traffic rate exceeds its rate, Source's 3 packets measured lower end-to-end delay than Source 2. The reason is that Source 3's packets are dropped if there are no resources, however the Highest Priority and Source 2 flows are delayed more and not dropped because they reserved resources and their priorities are high.

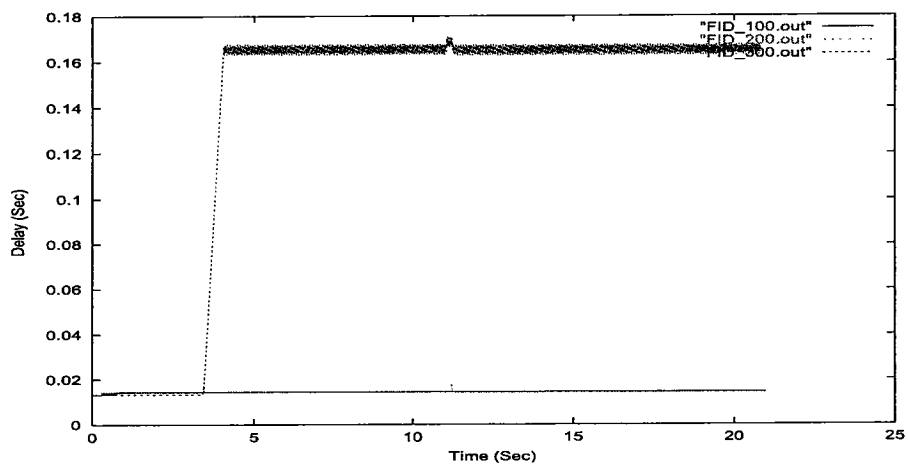

FIGURE 5.11: End-to-end delay for three traffic flows during flow ID 200 non-conformant when RSVP QoS protocol is simulated

Table 5.7: MPLS simulation results for End-to-end delay and packet loss rate

Traffic Source No.	Priority Level	Flow ID	Ave delay		loss rate	
			conformant	non	conformant	non
Highest Priority	15	100	13.87 msec	40.93 msec	0%	29.89%
Source 2	12	200	14.47 msec	42.65msec	0%	7.66%
Source 3	Best Effort	300	16.20msec	43.40 msec	0%	25.91%

5.3.2 MPLS Simulation Results:-

The MPLS network simulator (MNS) [79] patch was added to ns-2. Two simulation scenarios are tested in this section, the MPLS secnario and the constrained based routing using MPLS secnario. In each case the QoS performances are tested to compare them with the proposed algorithm. The same network topology setup used for the previous simulations is used here. However all the core nodes involved in the simulation become MPLS agents, different routing and labeling algorithms are supported. The links between nodes are set to duplex links and the MPLS domain queues are set to CBQ queue. At edge 1, packets are encapsulated and new MPLS labels are added to the IP packets. Three crlsp, constant based routing with lsp, paths have been setup for the three flows to secure resources before they start sending packets.

MPLS Simulation

Two simulation scenarios are presented in Table (5.7), conformant and non-conformant traffic flows.

A- Conformant Traffic Flows

⋆ Traffic flows ID 100 and 200 achieved 0% loss rate and lower end-to-end delay.

⋆ Traffic flow ID 300 achieves also 0% loss rate, however, the measured end-to-end delay was higher than flows ID 100 and 200 since flow ID 300 is best effort traffic type.

B Non-Conformant Traffic Flows

⋆ All traffic flows achieve high end-to-end delay when each one of the three flow is tested as non-conformant.

⋆ Packets belonging to the three flows are dropped during non-conformant scenario. Flow ID 100 has the highest loss rate since its rate is equal to the sum of flows' ID 200 and 300 rates. Flow ID 300 has a higher loss rate than flow ID 200 because it is best effort and all the packets exceed the rate are dropped.

Figure (5.12) presents the delay for the three traffic flows during flow ID 200 non-conformant simulation.

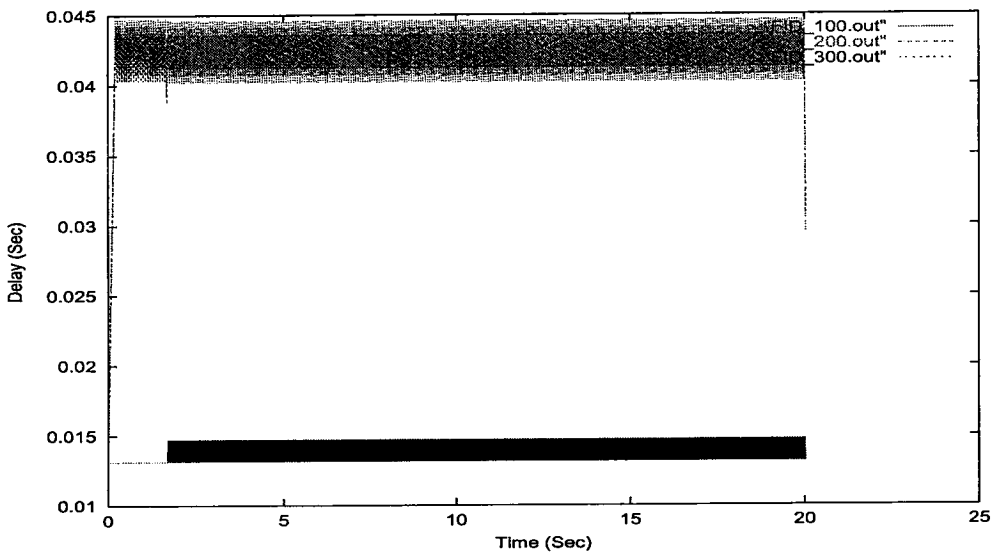

FIGURE 5.12: End-to-end delay for the tested three traffic flows during flow ID 200 rate increase and the involvement of MPLS

The other simulation secnario, is using MPLS constraint routing where routing paths are set depending on the QoS requirements. This is different from the previous

simulation case in which crlsps are set manually. The MPLS routing algorithms are invoked and the paths are set up accordingly. The same set up used in MPLS is used in this scenario and the outgoing nodes' queues are CBQ queues. Paths in this secnario are setup automatically according to the source rates and they change from setup to an other setup as shown in Figure (5.13).

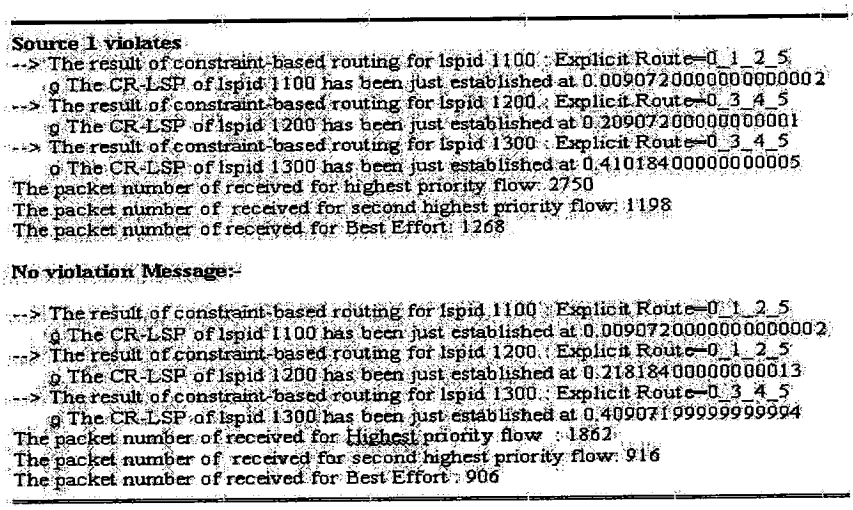

FIGURE 5.13: Printed out messages showing the explicit routes for each flow during MPLS-constraint routing simulation

The same simulation procedures are followed where traffic flows are conformant at the first time then each one of the flows are set to be non-conformant separately to observe how the QoS parameters are affected by non-conformant. Table (5.8) summarizes the results for the MPLS simulation that includes both conformant and non-conformant simulation scenarios.

A. Conformant Traffic Flows

- ⋆ All the traffic flows achieve 0% loss rate.
- ⋆ Traffic flow ID 200 achieves the highest end-to-end delay since each traffic flow uses different path resulting in a different end-to-end delay.

Table 5.8: MPLS constraint routing simulation results for End-to-end delay and packet loss rate

Traffic Source No.	Priority Level	Flow ID	Ave delay		loss rate	
			conformant	non	conformant	non
Highest Priority	15	100	31.75 msec	209.67 msec	0%	22.93%
Source 2	12	200	37.69 msec	205.64msec	0%	9.24%
Source 3	Best Effort	300	29.39msec	208.46 msec	0%	18.66%

B. Non-Conformant Traffic Flows

- ⋆ All traffic flows achieved a high end-to-end delay when each one of the three flow is tested as non-conformant.
- ⋆ Packets belonging to the three flows are dropped during non-conformant scenario. Flow ID 100 has the highest since its rate is equal to the double of flow ID 200 and 300. Flow ID 300 has a higher loss rate than flow ID 200 because it is best effort.

The average end-to-end delay for the three traffic flows for the conformant test is shown in Figure (5.14)

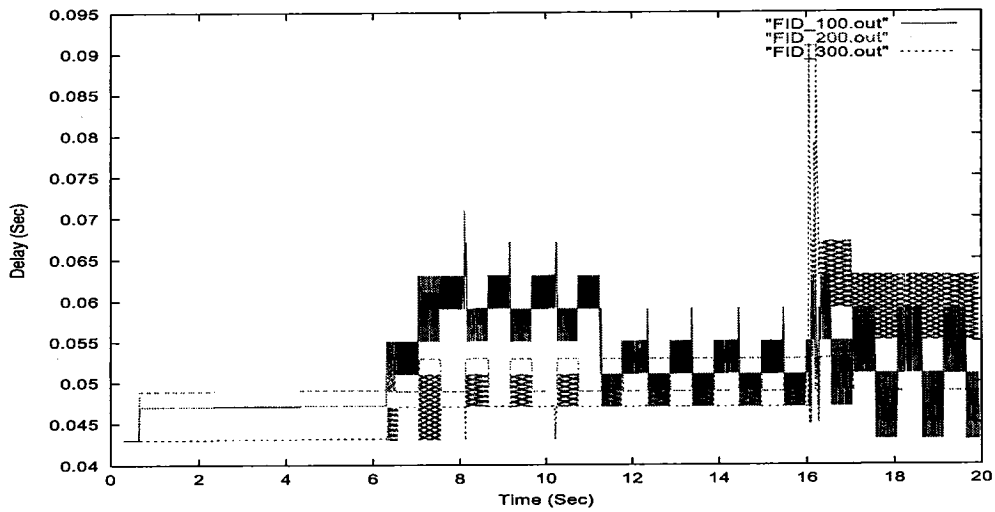

FIGURE 5.14: Delay for the tested three traffic flows when MPLS-constraint routing is simulated

5.3.3 DiffServ Simulation Results:

In this section different simulation scenarios are designed to test the QoS performance of a DiffServ domain using a BB and implementing a Token Bucket policy model. The other simulation secnario is testing the implementation of the QoSbox, that has been modified in the thesis to act as the IPv6 edge router. The same network topology used in Section 5.2 will be used in the DiffServ simulations. This box acts as a QoS manager, scheduler, and a policer. Traffic flows are processed at this box before entering the network domain. However, there is no cooperation with the other elements located in its domain or other neighbouring domains. The lack of cooperation results in no adjustment in processing of traffic flows during congestion or node failures.

DiffServ Simulations using Token Bucket

In this section the ARM BB is used to control the DiffServ resources and to have a direct comparison with the proposed IPv6 QoS manager [74]. A token bucket policer is chosen for monitoring and policing purposes since it was used to monitor traffic

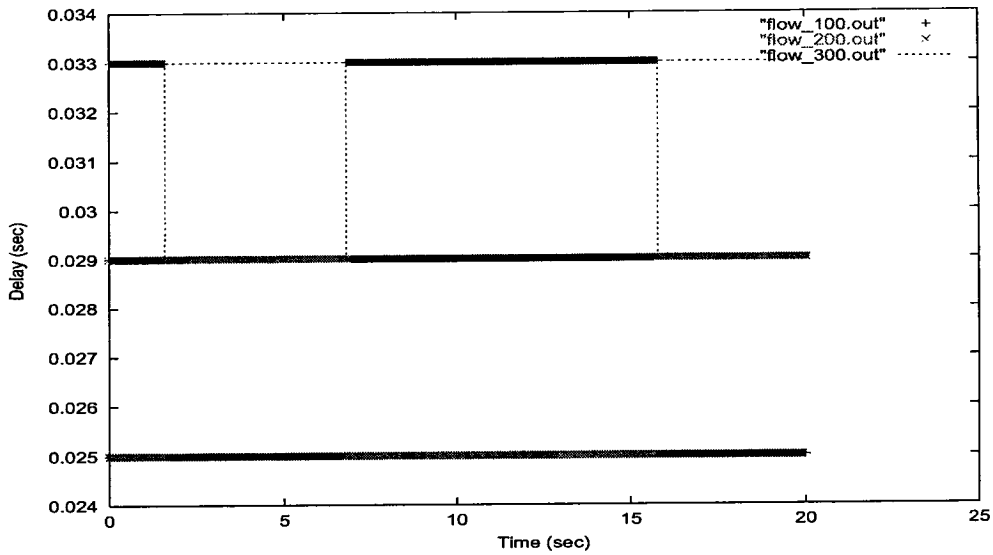

FIGURE 5.15: Delay for the tested three traffic flows for DiffServ simulation using Token Bucket policer

flows during the proposed IPv6 QoS management simulations. The simulation results are summarized in Table (5.9) which includes both conformant and non-conformant scenarios for each one of the traffic flows.

A. Conformant Traffic Flows

⋆ There are no packets dropped from all the three traffic flows.

Table 5.9: DiffServ simulation results using Token Bucket policer for End-to-end delay and packet loss rate

Traffic Source No.	Priority Level	Flow ID	Ave delay		loss rate	
			conformant	non	conformant	non
Highest Priority	15	100	29.20 msec	211.94 msec	0%	14.78%
Source 2	12	200	26.88 msec	200.85msec	0%	7.40%
Source 3	Best Effort	300	30.88msec	199.59msec	0%	5.914%

⋆ Traffic flow ID 300 end-to-end delay is the highest since it is a best effort traffic type.

B. Non-Conformant Traffic Flows

⋆ All traffic flows achieve high end-to-end delay when each one of the three flow is tested as non-conformant.

⋆ Packets belonging to the three flows are dropped during non-conformant scenario. Flow ID 100 has the highest loss rate since its sending rate equals to the sum of the rates of the other two flows. In addition non conformant packets are dropped at the ingress routers.

Figure (5.15) shows the delay digram for the DiffServ secnario secnario for the three traffic flows during flow ID 200 non-conformant test.

QoSbox Simulations

In this section the QoSbox that has been discussed in Section 2.2.2 is simulated to compare its QoS performance with the QoS manager proposed in this thesis. This box uses delay and bandwidth for reserving resources and classifying traffic flows. The traffic classification is done by mapping the flows to pre-defined classes according to the flows specifications. In this scenario, the same topology is used and the edge routers are QoSboxs. The three flows have assigned flow IDs for recording purposes and they are, 100 for the Highest Priority source, 200 for Source 2 and 300 for Source 3. The pre-defined classes are set as shown in Figure (5.17) in which five jobs queues are shown.

- **RDC (relative delay constraint)** has been set for the four pre-defined classes as follows:-

  a- Not concerned for class 1.

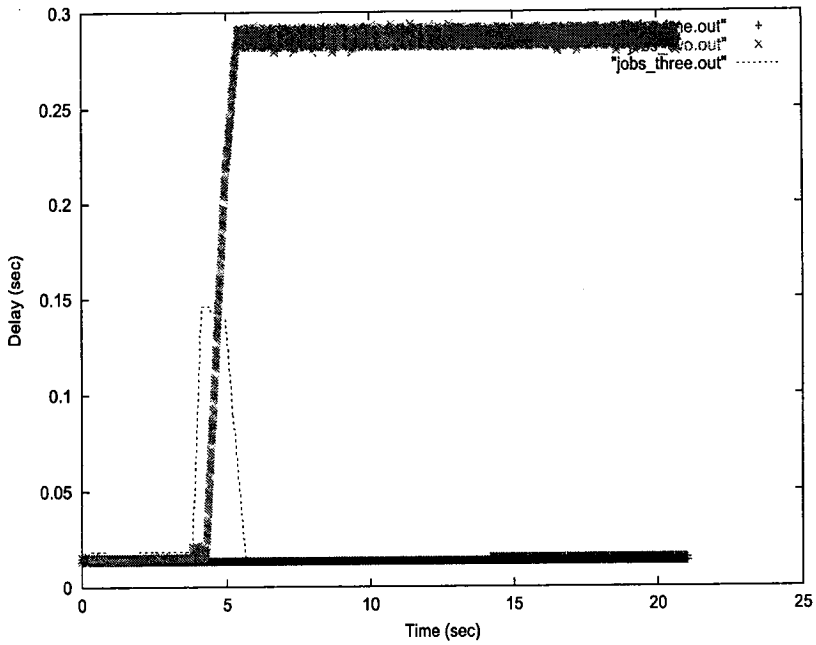

FIGURE 5.16: Delay for the tested three traffic flows for QoSbox simulation

b- Classes from two to four has a factor of 4 means that class 3 delay is 4 times class 2 and class 4 delay is 4 time class 3.

- **rlcs (relative loss constraint)** where class 1 is set to -1 "not concerned", class 3 loss rate is set to be double class 2 loss rate and also for class 4 twice class 3 loss rate.

- **alcs (absolute loss constraint)** in which class1 assured loss rate that does not exceed 10%. and the other class are set to be not concerned.

- **adcs (absolute delay constraint)** as the previous queue only class 1 guaranteed delay does not exceed 0.001 msec.

- **arcs (absolute rate constraint)** and all class classes are set to not concerned.

The same simulation procedures followed in Section 5.2.2 were followed in this section. Each traffic flow tested when its rate is conformant and non-conformant. The effect

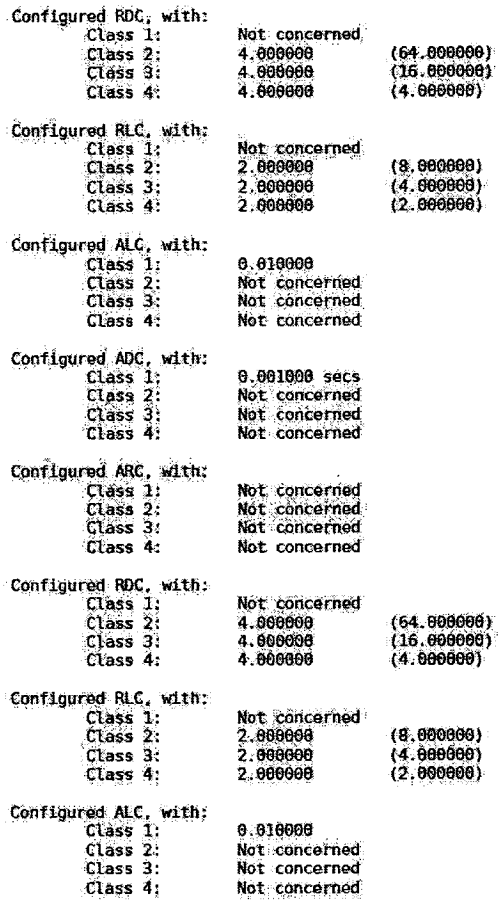

FIGURE 5.17: Simulation messages during jobs simulation

on the conformant flows when one of the other flows is non-conformant was also tested. The results are summarized in Table (5.10). As seen from this table, during conformant traffic flows, only the Highest Priority source drops 0.954% and the other two sources suffer no packet loss. The average end-to-end delay for all the flows were small. Figure (5.16) shows the end-to-end delay for the three flows.

The second secnario was to test each flow independently when its rate increases, non-conformant. The QoS parameters for non-conformant flows were observed and also the effect on others, conformant flows was observed.

The following is the summary of the results during non-conformant traffic flows:

Table 5.10: Jobs simulation results testing QoS parameters, End-to-end delay and packet loss rate

Traffic Source No.	Priority Level	Flow ID	Ave delay		loss rate	
			conformant	non	conformant	non
Highest Priority	15	100	14.52 msec	14.144 msec	0.954%	1.81%
Source 2	12	200	13.97 msec	227.10msec	0%	7.275%
Source 3	Best Effort	300	17.30msec	239.28 msec	0%	14.18%

1. Non-conformant Highest Priority source traffic flow, the average end-to-end delay is not affected however, more packets are dropped. Flows ID 200 and 300 QoS parameters suffer QoS degradation and loss rates recorded for both flows were 17.94% and 42.779% respectively. The average delay recorded for the flows IDs' 200 and 300 were 319.94msec and 15.38msec respectively. Therefore, flow ID 100 non-conformant situation has affected flow IDs 200 and 300 by both higher end-to-end delay and higher loss rate.

2. Non-conformant Source 2 traffic flow, the average delay for this traffic flow is increased to 227.10 msec and loss rate is 7.27%. The Highest Priority source traffic flow delay and loss rate are not affected by this rate increase. Traffic generated by source 3, recorded 24.34 msec delay and 14.70% loss rate during flow ID 200 non-conformant simulation. Therefore, flow ID 200 non-conformant test has resulted in a high packet loss and a high end-to-end delay for flow ID 300 only.

3. Non-conformant Source 3 traffic flow, the average delay is 239.29 msec and loss rate is 14.18%. Source 2 traffic packets recorded 25.24 msec and 6.26% loss rate which is indication that flow ID 200 has been effected by flow ID 300 rate increase.

The Highest priority traffic packets were not affected during testing the other two flows non-conformant rates. However the other two sources flows packets experience more delay and more loss rate when their flows are non-conformant and also when other flows are non-conformant.

5.3.4 Simulation Results Comparison between the Proposed IPv6 QoS Manager and other QoS Techniques

In Sections 5.2 and 5.3, the simulation results for the proposed IPv6 QoS management model and the IntServ [74], DiffServ [74] and MPLS models were presented. In this section, a summary comparison of the achievements of all these schemes will be discussed. Table (5.11) shows the comparison between the IPv6 proposed scheme and the other QoS techniques. The flowing are the comparison comments that are concluded:

5.3.5 Average end-to-end delay

- ⋆ The proposed IPv6 model achieved lower delay during conformant and non-conformant scenarios for the Highest Priority traffic source. The rate increase did not causes the delay to increase. Source 2 and Source 3 achieves low delay when their flows are conformant and high delay during non-conformant traffic flows.

- ⋆ The IntServ model achieves low delay for all the three flows during conformant traffic flows simulation, however, during each traffic flow rate increase the delay increase to 160 msec for the Highest priority source and Source 2 flows that reserve resources. Source 3 traffic flow achieved high delay when its flow flow is non conformant.

- ⋆ During MPLS simulation, all the three traffic flows achieve low delay. On the other hand, the delay increases for each traffic flow when its rate increase and

Table 5.11: Simulation results comparison between the IPv6 model and other QoS methods

QoS Method	Traffic Source	Ave delay		loss rate	
		conformant	non-confor.	conformant	non-confor.
IPv6 model	H. Priority	13.714 msec	14.60 msec	no drops	19.31% Degrad
	Source 2	13.59 msec	53.71 msec	no drops	8.62% drop
	Source 3	18.25 msec	102.45 msec	no drops	8.094% drop
IntServ model	H. Priority	15.85 msec	160.499 msec	no drops	no drops
	Source 2	14.04 msec	169.234 msec	no drops	no drops
	Source 3	17.87 msec	136.97 msec	no drops	24.168% drop
MPLS model	H. Priority	13.87 msec	40.93 msec	no drops	29.89% drop
	Source 2	14.47 msec	42.65 msec	no drops	7.66% drop
	Source 3	16.20 msec	43.40 msec	no drops	25.91% drop
MPLS model	H. Priority	31.75 msec	209.67 msec	no drops	22.93% drop
Constraint	Source 2	37.69 msec	205.64 msec	no drops	18.66% drop
Routing	Source 3	29.39 msec	208.46 msec	no drops	9.24% drop
DiffServ model	H. Priority	29.20 msec	211.94 msec	no drops	14.78% drop
	Source 2	26.88 msec	200.85 msec	no drops	7.40% drop
	Source 3	30.88 msec	199.59 msec	no drops	5.914% drop
QoSbox model	H. Priority	14.52 msec	14.14 msec	0.954% drop	1.81% drop
	Source 2	13.97 msec	227.10 msec	no drops	7.275% drop
	Source 3	17.30 msec	239.28 msec	no drops	14.18% drop

be as a non conformant traffic flow.

- ⋆ The constraint MPLS routing simulation results in a high delay for the three flows when their rates are conformant, since paths have to be set for each individual traffic flow. The rate increase for each individual traffic flow results in a very high end-to-end delay for packets belonging to that traffic flow.

- ⋆ The DiffServ simulation using ARM BB results in a high delay for all the three flows when their rates are conformant. This is caused by the time needed to map each flow to a DiffServ class. The delay will be higher for each one of them when its rate is non conformant.

- ⋆ The QoSbox simulation results in a low delay for all the three flows when their rates are conformant. During rates increase, the Highest Priority traffic flow achieves lower delay than the other two traffic flows.

5.3.6 Average Packet Loss Rate

- ⋆ No packets have been dropped during the proposed IPv6 QoS management model simulation when all the three traffic flows are conformant. Packets belonging to the Highest Priority source were degraded to a lower priority during the flow's rate increase. The other two flows achieve less than 10% drop rate when their rates are non-conformant.

- ⋆ No packets are dropped during the IntServ model simulation for both the Highest Priority and Source 2 flow for both conformant and non-conformant scenarios. Only Source 3 traffic flows packets are dropped when its rate is non conformant.

- ⋆ During the MPLS simulation, all the three traffic flows achieve 0% drop rate during conformant test. On the other hand, during non-conformant test, 30% of the Highest priority source packets are dropped, 8% of the Source 2 packets are dropped and 26% of the Source 3 packets are dropped.

- ⋆ The constraint MPLS routing simulation results in a zero drop rate for the three flows when their rates are conformant. The rate increase for each individual flow results in packets dropped, 23% from the Highest Priority, 19% from Source 2 and 9% from Source 3 traffic packets.
- ⋆ The DiffServ simulation using ARM BB results in a 0% packet drop rate for all the three flows when their rates are conformant. All the three flows packets suffer loss rate when their flows are non conformant.
- ⋆ The Highest Priority traffic source packet are dropped during conformant and non-conformant scenarios, 0.954% for conformant test and 1.81% for non-conformant. The other two flows have 0% loss rate during conformant scenario and 7% of Source 2 packets and 14% of Source 3 packets during non-conformant scenario.

The following are performance comparison during the two tests for all the models for the following traffic types :

1. Highest Priority traffic type and conformant test.
   - IPv6 and MPLS models achieved the lowest average end-to-end delay.
   - QoSbox and IntServ achieved also lower average end-to-end delay.
   - DiffServ and MPLS with constrain routing achieved the highest average end-to-end delay.
   - No packets have been dropped during all the tests except the QoSbox which dropped 0.954% of total flow packets.
2. Highest Priority traffic type and non-conformant test.
   - IPv6 and QoSbox achieved the lowest average end-to-end delay.
   - MPLS achieved a high average end-to-end delay.
   - DiffServ and MPLS with constrain routing achieved a very high average end-to-end delay.

- IntServ achieved the highest average end-to-end delay.
- No packets have been dropped during IPv6 model non-conformant test but they were degraded to a lower priority.
- No packet have been dropped during IntServ model non-conformant test.
- QoSbox model achieved the lowest drop rate and MPLS model achieved the highest drop rate.

3. Medium Priority traffic type and conformant test.

   - IPv6, QoSbox, MPLS and IntServ models achieved the lowest average end-to-end delay.
   - DiffServ and MPLS with constrain routing achieved the highest average end-to-end delay.
   - No packets have been dropped during all the tests.

4. Medium Priority traffic type and non-conformant test.

   - MPLS model achieved the lowest average end-to-end delay, IPv6 achieved a higher average end-to-end delay than the MPLS model.
   - QoSbox, DiffServ and MPLS with constrain routing achieved a very high average end-to-end delay, however, IntServ achieved the highest average end-to-end delay.
   - No packets have been dropped during IntServ test.
   - IPv6, QoSbox, DiffServ and MPLS achieved a low drop rate, however MPLS with constrain routing achieved the highest drop rate during this test scenario.

5. Best effort traffic type and conformant test.

   - MPLS, IntServ, QoSbox, and IPv6 models achieved the low average end-to-end delay.

- DiffServ and MPLS with constrain routing achieved the highest average end-to-end delay.
- No packets have been dropped during all the tests.

6. Best effort traffic type and non-conformant test.
   - MPLS model achieved the lowest average end-to-end delay, IPv6 achieved a higher average end-to-end delay than the MPLS model.
   - IntServ, DiffServ and MPLS with constrain routing achieved a very high average end-to-end delay, however, QoSbox achieved the highest average end-to-end delay.
   - DiffServ model achieved the lowest drop rate.
   - IPv6, QoSbox and MPLS with constrain routing achieved a high drop rate than DiffServ, however IntServ and MPLS achieved the highest drop rate.

Therefore IPv6 simulation of the proposed model outperformed the others for the highest priority traffic types either by achieving lower delay or 0% packets drop rate. It also achieved good results with the two traffic types.

5.4 Simulation of two and three QoS domains using IPv6 QoS management models

5.4.1 Two Domains Simulation

Simulation setup

As mentioned in Chapter 4, the last simulation scenario will be to test if a destination node is connected with an other IPv6 domain that supports QoS. Source node 12 , Highest Priority, sends a request for a CBR traffic flow with a rate 0.5 Mbps and has priority level set to 15 to Destination Node(1). Source node 13, Source 2, sends a request for CBR traffic flow with 0.25 Mbps CIR and priority set to 12 to Destination Node(2). Source node 14, Source 3, send a Best Effort traffic at 0.25 average rate to Destination Node(3). All the generating sources are connected to domain 1 through node 0, Edge1. Therefore, all QoS requests have to be sent to this node to be forwarded to the domain1 manager and all responses are also forwarded back through this node to the nodes generated these requests. Node 2, Edge3, is the entrance node for domain 2, which means all packets passing through this domain has to pass this node first. The network topology is shown in Figure (5.18)

The Highest Priority source requests are processed locally inside domain 1 since the destination node is located at the same domain. Domain 1 manager checks for resources availability only since it has already confirmed that Destination Node(1) is located at the same domain. Source 2 requests are forwarded to domain 2 QoS manager since destination Node (2) is not located at the first domain. Therefore resource are reserved in both domains allowing Source 2 data packets to receive the QoS requested. Token bucket is implemented at edge 1 to monitor all the traffic flows packets and applying domain 1 QoS manager policies concerning non-conformant cases. The Highest Priority packets will be degraded to level 12, packet with priority 12 degraded to best effort and finally best effort packet are dropped during congestion.

The simulation setup procedures will be the same as explained in Section 5.2.2. First

all the follows will not exceed their initial rates, conformant, then each traffic flow exceeds its rate by 30% , non-conformant. Average end-to-end delay and packet loss rate are measured in each scenario.

Figure (5.19) shows the average end-to-end delay for the three flows during non congestion (non violation).

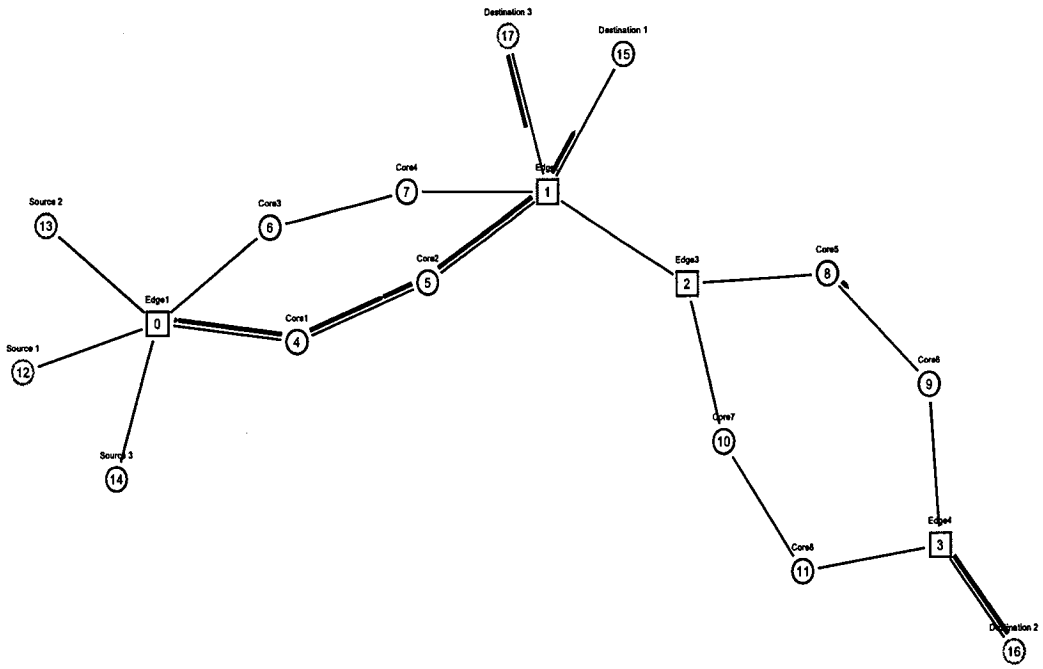

FIGURE 5.18: Simulated two domain network topology

Results Analysis:

Table (5.12) summarizes the results obtained from simulating two IPv6 domains. Traffic flows are tested for two scenarios, during conformant and non conformant traffic flows that caused by one of the traffic sources rate increases of 30%. Each time a traffic flow is non-conformant, its rate increases, the delay and the loss rate are recorded at the destinations. The following is the summary of simulation results comments:

I- Each time Highest Priority and Source 2 traffic flows were non-conformant, best

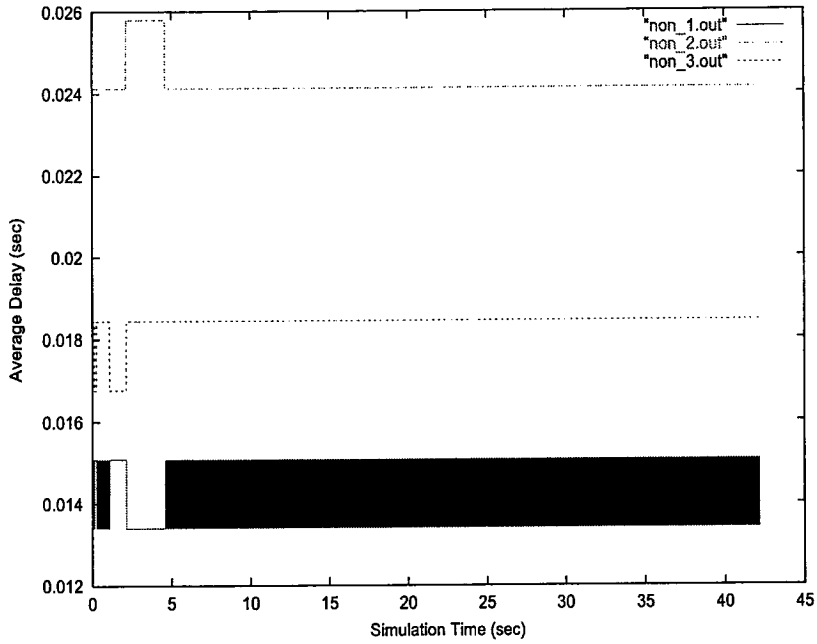

FIGURE 5.19: Delay for the tested three traffic flows for two domain simulation

effort traffic flow delayed the most and also some of its packets are dropped. For example when the highest priority traffic rate increases, average delay for Source 2 packets is 25.800msec, however; Source 3 measured delay was 275.086msec.

ii- Source 2 average delay was higher than highest priority and best effort since it was passing two domains.

III- In the case of Source 2 traffic flow was non-conformant, its flow packets were degraded to best effort and then dropped in the case of congestion. There is no QoS guarantee for Source 2 traffic flow where its rate increases.

Table 5.12: Two domains simulation results testing QoS parameters, End-to-end delay and packet loss rate

Traffic Source No.	Priority Level	Flow ID	Ave delay		loss rate	
			conformant	non	conformant	non
Highest Priority	15	100	14.214 msec	14.94 msec	0.0%	0.0%
Source 2	12	200	24.218 msec	25.661msec	0%	0%
Source 3	Best Effort	300	18.392msec	131.374 msec	0%	20.866%

5.4.2 Three Domains Simulation

In this simulation scenario, three IP QoS domains are tested and three destination nodes are attached to each one of the domains as shown in Figure (5.20). The purpose of this simulation scenario is to test the flexibility of the proposed QoS model. The following is the setup of the traffic flows used in this simulation:-

- Source 1 generates a traffic flow at average rate of 0.5 Mbps, priority level is set to 15, highest priority, and intended to Destination 3, located in the third domain.
- Source 2 generates a traffic flow at average rate of 0.25 Mbps, priority level is set to 12, medium priority, and intended to Destination 2, located in the second domain.
- Source 3 generates a best effort traffic flow at average rate of 0.25 Mbps and intended to Destination 1, located in the same domain.

Therefore, Source 1 traffic packets has to traverse all the three domains to reach the destination and QoS reservation has to be done in all the three domains. Source 2 packets has to traverses only two domains and reservation has to be done on Domain 1 and 2.

Table 5.13: Three domains simulation results testing QoS parameters, End-to-end delay and packet loss rate

Traffic Source No.	Priority Level	Flow ID	Ave delay	Packet loss rate
Highest Priority	15	100	35.14 msec	0.0%
Source 2	12	200	25.661msec	0%
Source 3	Best Effort	300	18.35msec	0%

Results Analysis:

Table (5.13) shows the results obtained from this simulation. The three traffic flows are tested when they are conformant and only Source 1 traffic flow tested during non-conformant to see the effect on delay and loss rate on the other traffic flows. The following are the summary of the comments drawn from Table (5.13) and Figure (5.21):-

- ⋆ Flow ID 100, high priority, packets have achieved low end-to-end delay during conformant and non-conformant tests compared with the other two flows since they traverse three domains. There are no packets dropped during both conferment or non-conferment.

- ⋆ Flow ID 200, medium priority, packets have achieved also low end-to-end delay compared with the best effort even though it traverses two domains. No packets have been lost. During Flow 100 non-conferment test, delay increased by 11 msec however no packets have been lost.

- ⋆ Flow ID 300, best effort, packets have achieved the highest delay and packets have been lost when Source 1 increases its sending , flow ID 100 non-conferment test.

In conclusion, the proposed QoS management model proved to be flexible and scalable since excellent results have been achieved even with two and three domains.

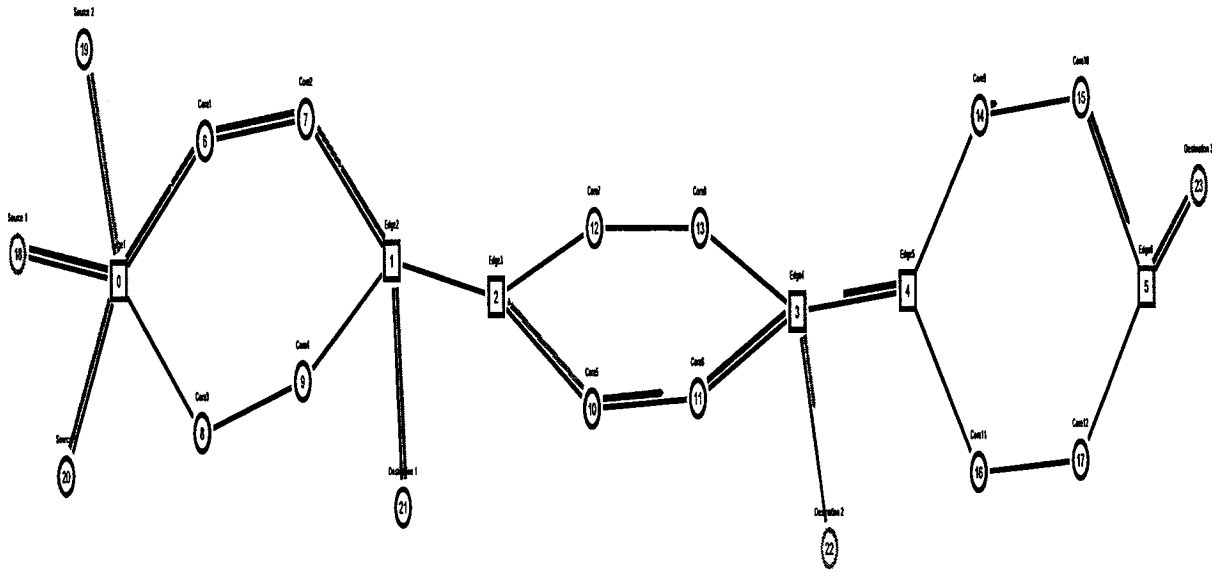

FIGURE 5.20: Simulated three domain network topology

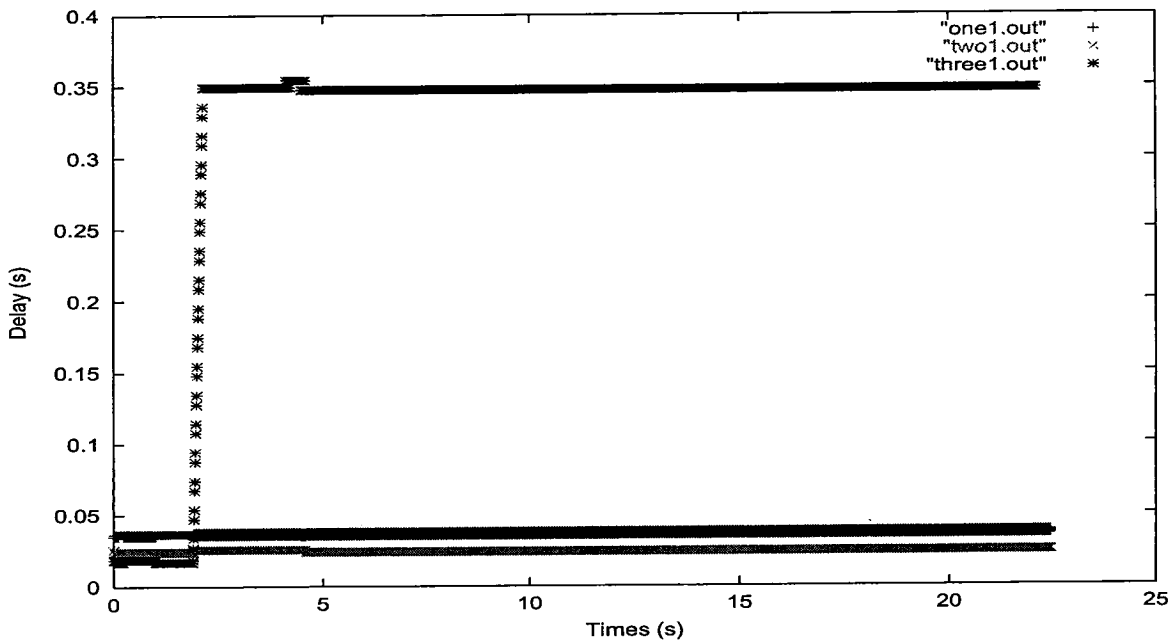

FIGURE 5.21: Average delay for the three traffic flows

Chapter 6

Implementation of Internet billing on the IPv6 QoS manager

6.1 Introduction and Motivation:-

It is expected that in the very near future integrated QoS capable networks will emerge which provides a variety of transmission services, such as telephony, video, interactive games, teleconferencing, file transfer and all the other traditional Internet services. Upon accepting a user connection, this system will be capable of negotiating QoS parameters and guaranteeing the agreed quality. This differentiation of traffic flows causes the Internet elements to process them differently according to their data contents. Pricing, therefore, becomes an important but not critical issue in today's Internet, however this is changing [80]. Traditionally, the individual users have not been charged for their use of networks resources, and have not generally been aware of the impact of their use on network performance. As a result, the traditional Internet pricing schemes, institutional funding or flat rate for unlimited usage, are unfair and have no differentiation between different traffic flows that have different QoS requirements. The achievement of QoS-enabled networks makes pricing an important issue. Most of the proposed pricing schemes address the following categories [60]:

I- Economic Efficiency and Optimality which means achieving the optimal overall net value of network usage.

II- Simplicity and Scalability in which designing a simple and a scalable architecture/scheme.

III- Quality of Service (QoS) by allowing the service providers to provide better QoS guarantee to the customers.

IV- Low cost in implementation and usage by keeping the implementation and usage of a pricing scheme low for both network providers and Internet users.

Most of the previously used proposed pricing schemes [58] [81] [82] [83] [59] [60] [61] tried to address one or more of these issues and so far there has not been a well-accepted solution.
This Chapter uses a new dynamic pricing scheme that considers the status of the network and the traffic type when prices are calculated and addresses the four points shown above. This scheme when combined with the IPv6 QoS management scheme, takes the advantage of GDI for recording and tracing customer charges. The combination is referred to as the integration pricing model in the following sections.

6.2 Review of the Internet Pricing Schemes

Recently, pricing schemes for the Internet has been an active research area. In this section some of the known pricing approaches are reviewed.

6.2.1 Flat Pricing [58]

Under a flat pricing scheme the user is charged a fixed amount per unit time, irrespective of usage. This scheme has desirable advantages. First, it is simple and convenient since no assumptions are made about the underlying deployed network,

and no measurements are required for billing and accounting. Secondly, the scheme assumes that there is relatively stable demand for resources, and makes no attempt to influence the individual traffic flows. For this reason the scheme is unsuitable for congestion control or traffic management. All users are charged the same even if some of them suffer packet loss while others consume all the resources.

6.2.2 Priority Pricing [58]

Priority pricing requires users to indicate their traffic value by setting the priority field in every IP packet header. In this scheme, measurements are required for billing and accounting to keep track of the priority level of each user transmitted packet. During periods of congestion, traffic is transmitted by priority level, and low priority traffic is either delayed or dropped. Priority pricing scheme, P_{b_1,b_2} is defined by two flags (two bits), b_1 priority flag and b_2 non-drop flag. This results in four service classes:-

1. $P_{0,0}$ Base class (best-effort).
2. $P_{1,0}$ non-drop and low priority class .
3. $P_{0,1}$ drop and high priority class.
4. $P_{1,1}$ non-drop and high priority (real time) class.

The cost relation between these classes are $P_{1,1}cost = 3 * P_{0,0}cost$ and $P_{1,0}cost$ and $P_{0,1}cost = 2 * P_{0,0}cost$ [81]. Priority pricing raises the economy efficiency of the network since only low value packets are dropped. Under this scheme, the user selects one of the four classes to maximize the overall of satisfaction (QoS insurance). The network processes all the incoming packets by priority, maintaining different queues for each class. The packets are queued on an FIFO basis and transmitted by priority level.

6.2.3 Smart-Market Pricing [82]

Smart-market pricing focuses on the issues of capacity expansion and costs imposed on customers. The latter includes such as connection cost and per packet cost that covers the incremental cost of sending a packet. Mason and Varian [82] introduced a usage charge, during network congestion, which is determined through an auction. The user inserts a bid price in each packet's, carried in the packet's header, communicating the user's willingness to pay for transmitting the packet. The network collects and sorts all the bids, and sets a threshold value. All packets whose bid exceeds the threshold value are transmitted. The threshold valued is determined by the network's capacity and represents the marginal cost of congestion. Each transmitted packet is then charged this marginal cost. This scheme performs like a priority scheme in which traffic-flows with low QoS demand are not guaranteed resources. Traffic packets are transmitted according to their relative priority and bid prices during congestion.

6.2.4 Edge Pricing [83]

Edge pricing combines the approximation of congestion conditions such as time of day and expected path, where charges depends only on the source(s) and destination(s). Therefore, the resulting prices can be determined and charges are assessed locally at the access point. Prices are computed at the domain providers' edges where the users' packets enter the domain network rather than computing them in a distributed fashion along the entire path. Therefore, the focus is shifted to locally computed charges based on a simple expected values of congestion and route. Such a pricing scheme is much simpler than the previous one, and facilitates receiver payments. Traffic management can be supported when this scheme is associated with ATM/RSVP. However, traffic measurements for billing and accounting may still be required.

6.2.5 Per-Packet pricing scheme [59]

In this scheme each packet carries electronic money, which is used to pay routers in return for services. Each packet header includes the following fields:-

1. **User identity header** which is used to identify the user by the billing system.
2. **Accumulated charge field** that includes the total amount of electronic money that the user is to be charged. It is updated by routers based on pricing scheme.
3. **Money field unit** is used by users to set an upper limit to the total amount that will be charged by the network for servicing transmitted packets.

Each router in the per-packet billing domain has a field that is used to accumulate the payments for its services. This field is incremented for each packet serviced when the packet arrives at the last hop. Information about the user is sent to a central billing data base. Users are charged according to the accumulated collected information for the packets they sent.

6.2.6 Dynamic Pricing model[60] [61]

This scheme is introduced as the DiffServ end-to-end pricing scheme. In this scheme pricing is focused on the core networks and end-to-end pricing is pushed to the access network. It is up to the end-user access network that decide whether a flow should enter its domain or not. Two pricing tables, domain and global tables, are defined by this scheme. A price entry in the domain price table represents the price of a service class from one edge node to another edge node within the same domain. A centralized pricing station for each domain communicates with all domain elements and collects all the price information. As a result this station maintains a price table for each ingress-egress pair within the domain. However, a price entry in the global price table represents the price for a service class from one domain to another.
This model is based on market model where the values for the **base price** P^i_{base} and the **fill factor** f_i is set by the network provider for each traffic type. The fill factor

f_i is defined as the ratio between the target capacity T_i and the maximum capacity C^i_{max} for a class service i. This scheme starts by assuming the base price P^i_{base} which reflects the equipment cost, maintenance/administrative costs and business revenue considerations for a network element. P^i_{base} is computed offline and prices are classified according to the traffic class QoS demand, higher prices are applied to higher QoS traffic flows. Service providers sets different f_i for each traffic class to enable service differentiation between classes. To compute the dynamic price of a service class, the following equation is used [84]

$$P_i(t) = P_i(t-1) + \alpha_i * (D_i - T_i)/T_i \tag{6.1}$$

Where $P_i(t)$ denotes the price for a class i at time t and D_i is the demand or current load for class i and α_i is the convergence rate factor.

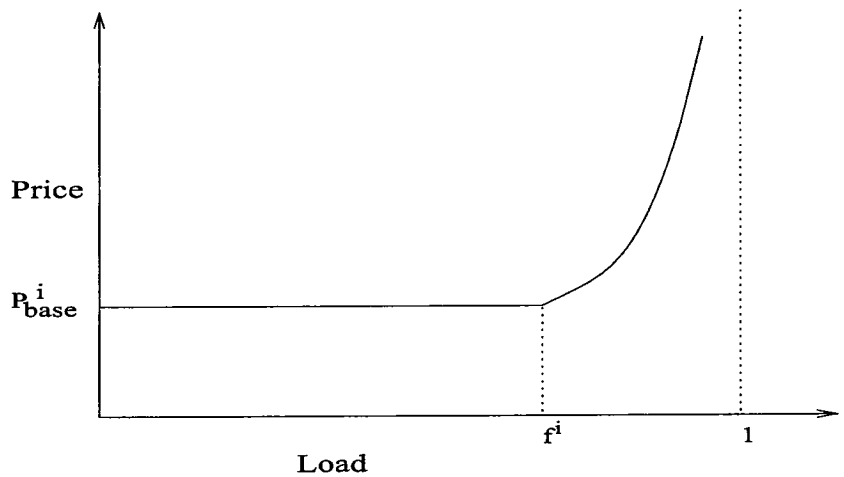

FIGURE 6.1: General Pricing strategy

Figure (6.1) illustrates the general pricing strategy. When the load for a particular service class is lower than its targeted capacity, the price is the base price P^i_{base} for that particular service class. As the load exceeds its target capacity, the price will be increased rapidly and even dramatically when the load is close to the maximum capacity. The following equation is used to adopt the exponential growth of the

pricing strategy during the demand increase.

$$P_i(t) = \left\{ \begin{array}{ll} P^i_{base} & if D_1 \leq T_i \\ P^i_{base} e^{\alpha_i[\frac{D_i}{T_i}-1]} & otherwise \end{array} \right\} \tag{6.2}$$

A price limit, P^i_{max}, can be set for each class service and this indicates the price when the demand reaches the maximum capacity. It is calculated using the following equation

$$P^i_{max} = P^i_{base} e^{\alpha_i[\frac{D_i}{T_i}-1]} \tag{6.3}$$

Therefore knowing P^i_{max}, P^i_{base} and fill factor f_i for a class service i gives the solution for the convergence factor α_i.

$$\alpha_i = \log(\frac{P^i_{max}}{P^i_{base}}) * (\frac{f^i}{1-f^i}) \tag{6.4}$$

The total revenue will be the sum of all classes prices. This model is used in DiffServ domain.

The dynamic pricing scheme was chosen to be integrated with the proposed IPv6 QoS management model for the following reasons:

1. Prices are changing according to the network status which results in high prices for the QoS guaranteed traffic flows resulting in a good profit.

2. Revenues are accumulated at the edge routers which works perfect for the proposed model since all the domain's traffic flows are monitored at this point.

3. The proposed QoS manager reserves resource dynamically and this pricing method is charges the customers dynamically.

6.3 The Integration of the Proposed IPv6 Manager with the Dynamic Pricing Model

6.3.1 Introduction:

It is more desirable to tie the pricing scheme with admission control in any QoS environment. This will solve the concern of users and network providers since resources are controlled by the domain control and prices for each traffic flow are managed by the pricing scheme. Therefore, integrating management control with the billing system will handle all customers and providers concerns. The design of the proposed architecture is based on the following:

1. A QoS management system that differentiates between traffic flow and secures their requested resources in the case where resources are available.

2. The price will reflect the availability of the domain resources on which prices are different for each traffic flow. Also, as the filling factor f_i for a flow i increases, the price for that flow increases. Therefore, prices during network congestion increase, and only traffic flows that request QoS are capable of paying higher prices are allowed.

3. Scalability in which core routers do not perform any accounting procedure. Billing and accounting agents are kept at the edge of the network. The edge router receives all QoS requests and forwards them to the QoS manager where a decision is taken, and traffic specification is sent to the pricing agent in case of acceptance. The pricing agent calculates the price of the accepted traffic flow by checking f_i and then using formula (6.2) to find the right price for the accepted traffic flow. The price is then sent to the edge router to be forwarded to the customer that iniated the request.

4. Simplicity and fairness. The model charges customers according to their traffic type and status of the network. Each traffic flow is assigned a base price different

than the others based on its contents. Real time traffic is charged more than data traffic. The base price increases as the network reaches congestion, and only traffic packets for customers who accepted the new prices are delivered

6.3.2 Implementation and Deployment

Implementation

Figure (6.2) shows the flow chart of the integrated pricing model. In this figure, each unit function involved in the pricing and accounting process are shown. It starts at the source which initiates the QoS request and then waits for the responses of the acceptance messages and their associated prices. The edge router does the forwarding for requests, and in addition monitors all traffic packets entering the domain. The IPv6 QoS manager processes the QoS requests and then sends the network status for each accepted traffic flow to the pricing agent. The pricing agent calculates the price for each accepted traffic flow by first finding f_i using the information received from the domain QoS which includes the expected traffic rate (T_i) and the max allowed rate (C^i_{max}). The pricing agent, attached to the edge router, will iniate the price according to the network status and the defined base price (P^i_{base}) for each traffic class initially accepted by the manager. All the prices are then sent to the customers who iniated the requests together with the resources acceptance messages. Finally each customer decides either to start sending packets or reject the price.

Pricing Strategy

Two parameters can affect any dynamic pricing model, response time and pricing interval. The value of these two parameters affect the stability of the pricing system. Response time is the time taken to convey the price signals back to customers plus the time for customers to react to these signals. The pricing interval is how often a price will be updated or recalculated. If the response time is too large, then users may react to a signal that is no longer true and visa versa.

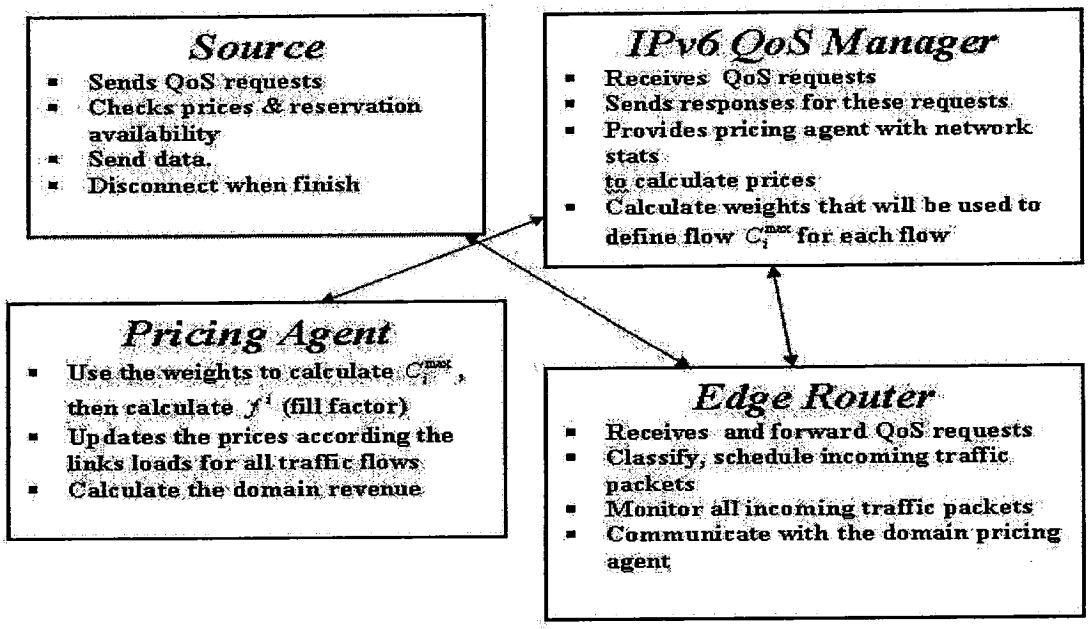

FIGURE 6.2: Flow chart for the IPv6 pricing model

The integrated pricing model consider these two parameters. The response time is expected to be short since the GDI is used for tracing reservations and communicating requests and responses between the domain QoS manager and edge routers. Also, the QoS manager reservation procedure is fast when the GDI is involved since the GDI combination is unique in the domain and the QoS manager uses it to trace and reserve requests. Therefore, there is no need to find the best match by examining the IP headers 5-tuples and using them for tracing and reservation purposes. This results in less processing delay in both the router processing and the QoS manager, which in fact reduces the response time that affects the dynamic pricing model.

6.3.3 Simulation Results

In order to study the robustness and behavior of the proposed pricing model, an IPv6 QoS capable network environment using ns-2 has been setup. The ns-2 code

that was used to simulate the IPv6 QoS manager was modified to in-cooperate the pricing model. The code and TCL script are shown in Appendix D. The goal of the simulation is to evaluate the pricing model setting strategy and how it cooperates with the QoS manager. Figure (6.3) [85] illustrates the network topology used in this simulation which consists of 4 core routers and 2 edge routers. The Ingress router, Edge1, acts as the pricing agent and handles QoS requests generated by source 1 and source 2 nodes. The total capacity of each link is 1 Mpbs and the propagation delay is 1 msec. The specification for the generated traffic flows are:

1. Source 1 traffic rate is 500 Kbps, priority is 15 and Flow ID is set to 15.
2. Source 2 traffic generation rate is 250 Kbps, priority is set to 12 and flow ID is 12.
3. Source 3 traffic generation rate is 250 Kbps, best effort type of traffic and flow ID is 8.

The base prices P^i_{base} for each class are set to \$0.16, \$0.09 and \$0.04 per unit time respectively starting from the highest priority. The prices start to increase for each load according to the following percentages:

1. When the link capacity reaches 50% for FID(15)
2. 70% of the total link load for FID(12)
3. 100% of the total load for Best Effort FID(8).

Two simulation scenarios have been tested for each traffic flow, one when the total link load is less than the percentage assigned for each flow and the other one when the traffic exceeds these percentages. Figure (6.4) shows the change of the prices for traffic flow FID(15). $Flow15_{base}$ is the price which represent the load traffic (total load less than 50%) and $flow15$ is the price during congestion. As shown in Figure (6.4) the prices change rapidly as the load increases which results in more revenue, corresponding to QoS assurance for this flow packets. Figure (6.5) shows the change

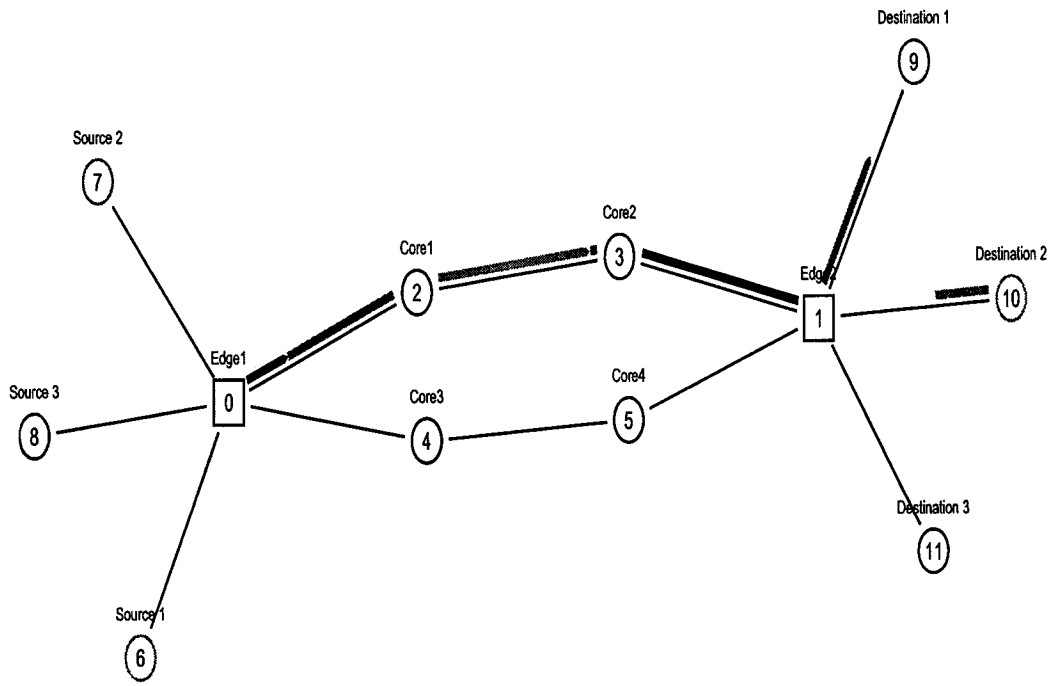

FIGURE 6.3: Simulated pricing model network

of the prices for traffic flow FID(12). $Flow12_{base}$ is the price which represent the load traffic (total load less than 70%) and $flow12$ is the price during congestion. As shown in Figure (6.5) $FID(12)$ prices are not changing much since the percentage is set to 70% and the packets of this flow are not as critical as the Highest Priority flow. The increase in the revenue is small compared to Highest Priority. However, during congestion packets belonging to this flow are degraded to best effort and dropped as the policies set by the QoS manager indicates.

The prices for source 1 traffic flow, FID(15), have the most significant changes since it represents the highest priority among the three, and its price starts increasing exponentially, according to Equation 5.2, after reaching 50%. The Best effort has no change from the base price since its its expected load is set to 100%.

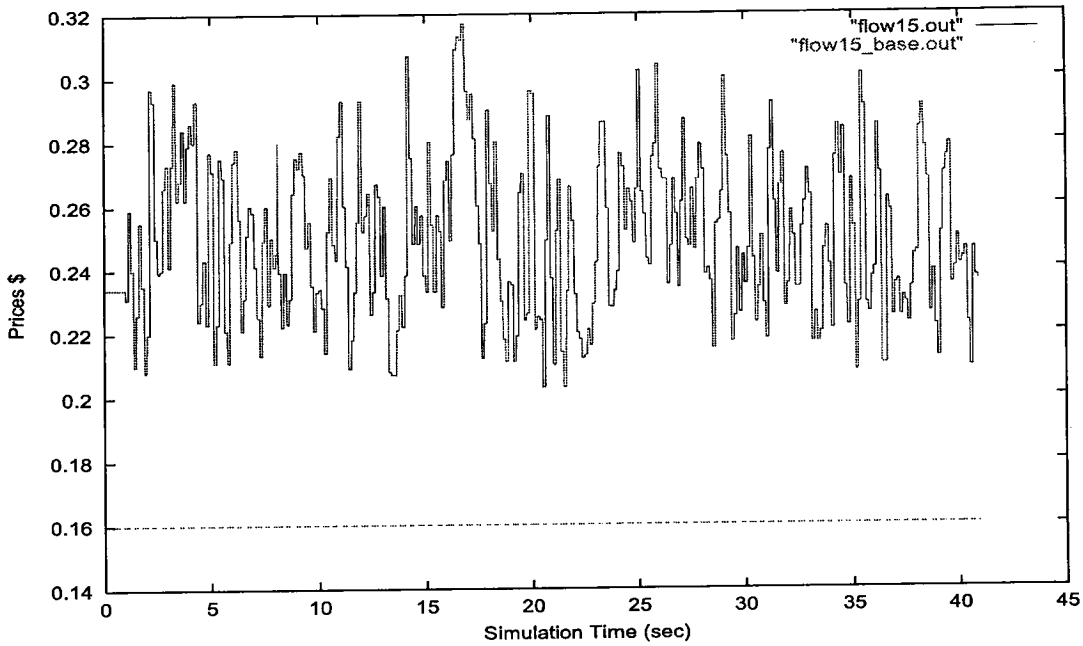

FIGURE 6.4: Traffic flow 15 Prices during network congestion and low traffic load

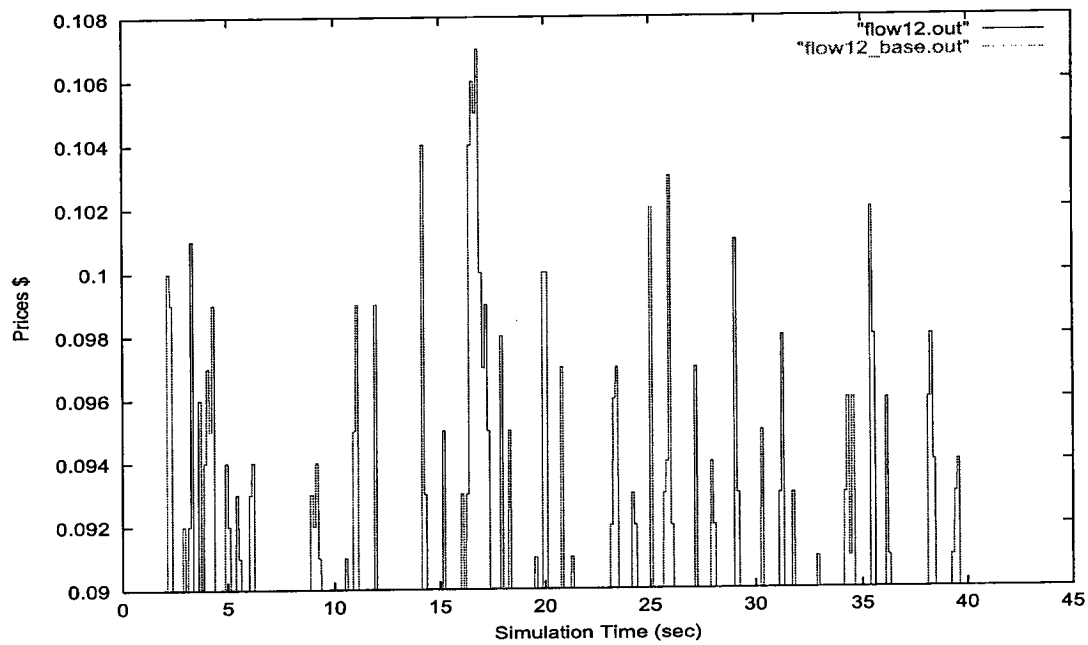

FIGURE 6.5: Traffic flow 12 Prices during high traffic and low traffic load

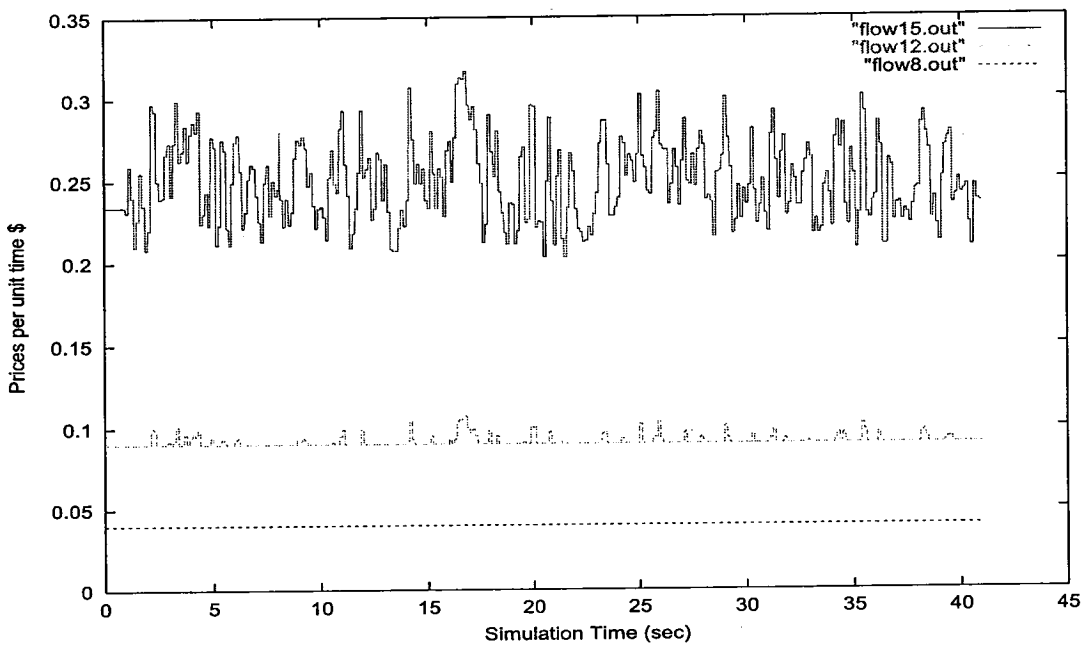

FIGURE 6.6: Prices for the three traffic flows FID(15) & FID(12) and FID(8)

Chapter 7

Conclusions and Future Work

This chapter concludes the thesis by summarizing the major contributions of the thesis and suggesting some key directions for future work.

7.1 Thesis Summary

In Chapter 3, a new QoS model was introduces. This model has a lower end-to-end delay bound than both the currently used models, DiffServ and IntServ. A scalable IP QoS management model that guarantees network resources and achieves low end-to-end delay was introduced in Chapter 4. The new proposed QoS model combines some advantages from the three QoS approaches used in the literature, IntServ, DiffServ and MPLS. It is intended to work for per-flow reservations as in IntServ, however reservations decisions are made by a central QoS manager not by each node independetely as in IntServ. Core nodes are kept simple, they forward and schedule traffic flows only as in DiffServ, and the complexity is pushed to the edge nodes. This model does not map traffic flows to pre-defined classes such as in DiffServ and does not use the DSCP to schedule packets inside its domain. Each traffic flow's packets are admitted and treated independently, not in aggregates as in DiffServ. In addition, flow labels are used inside the domain to forward and schedule packets as in MPLS where fast forwarding can be achieved when labels are used.

The major contributions and results of this thesis have four core components:

- Mathematical modeling,
- Model Architecture,
- Model implementation and simulation,
- Integrating dynamic pricing with the QoS model

each of which is discussed in the following sub-sections.

7.1.1 Mathematical Modeling

Network calculus principles were used to develop the model proposed in this thesis, and to find the end-to-end delay bounds of the model. Different schedulers, queues, and regulators were discussed then a WFQ scheduler and a leaky bucket regulator were chosen to be implemented with the QoS management model. The leaky bucket regulator will smooth all the incoming traffic flows. It is implemented at the ingress node. The leaky bucket is represented by an arrival curve with two main parameters, ρ "average rate" and σ "burst tolerance". The scheduler, WFQ, was implemented at each node along the path from the source node to the destination. WFQ is represented by a latency service curve with latency T_{lat} that approximates the queuing and processing delay needed by each packet. The flow's packet priority will be a factor in finding T_{lat}.

Numerical results showed that the proposed model achieved less end-to-end delay bounds than IntServ and DiffServ. The modeling can be to find the worst case end-to-end delay for any network topology. These calculations can be used later by the QoS manager to evaluate QoS requests with specific end-to-end delay.

7.1.2 Model Architecture

An IP QoS management model has been designed that facilitates per-flow resource reservation across IP network domains. The key ideas that contribute to scalability

and simplicity are:

1. QoS requests are sent to ingress routers which communicates with its QoS domain manager.

2. Core routers forward and schedule traffic flows only, no reservation decisions are to be made as in the IntServ case.

The second key idea that contributes to faster forwarding and reservation is the usage of GDI to request, reserve and trace resources at the QoS manager. The third key idea that contributes to support sensitive traffic flow's applications is the dropping of the high priority traffic flows to a lower priority during network congestion or traffic bursts instead of marking them as non-conformant. This results in higher delay but no high priority packets will be lost.

7.1.3 Model Implementation and Simulation

All the design ideas mentioned in section (7.1.2) were implemented using ns-2, a commonly used simulator. Additional software modules were added to the simulator to support the model implementation. Different simulation test scenarios were implemented in Chapter (5) for the IPv6 QoS model and IntServ, DiffServ and MPLS. The purpose of these tests was to evaluate how the IPv6 QoS management model behaves under different network conditions, normal or congestion (traffic violation). The same conditions were used for testing other QoS techniques for comparison purposes.
The results showed that the proposed model performed better that all other schemes simulated under both conditions. No packets have been dropped from higher priority traffic flows under traffic violation tests for the proposed model. These results support the numerical evaluations [Chapter 3] and prove that the IPv6 QoS management model is a better solution than IntServ and DiffServ. At the end of Chapter (5), a two and a three domains simulation scenarios was implemented and magnitude of the end-to-end delay for traffic applications that request QoS in more than one domain

was proportional to delay over one domain. This indicates that the proposed model is flexible and works over more than one domain as does DiffServ.

7.1.4 Integrating Dynamic Pricing with the QoS Model

Internet pricing will be a critical issue in the future since the demand for more network resources to send and receive real time data is increasing. Therefore, IP networks have to implement QoS tools that guarantee fast delivery for delay and loss sensitive traffic flows. The implementation will add costs to the networks which makes billing and pricing an important issue. In this thesis, a dynamic pricing model is integrated with the IPv6 QoS manager to control resources and charge customers according to their usage.

The most recent pricing strategy, dynamic pricing policy, was integrated with the IPv6 QoS management scheme to control resources during various network loads. Only customers willing to pay more during network congestion were allowed to send their traffic packets. Therefore, other traffic flows were either dropped or queued. This implemention results in more profit for the service provider since only the few flows with the highest service rates were admitted. In addition to profit increase, customer satisfaction is achieved as no QoS degradation occurred during network congestion. Without such a management scheme packets from all flows would be dropped or suffer increased delay resulting in QoS degradation for real time traffic flows. Simulation results show higher profits and no higher priority packet loss thus proving that the proposed QoS manager can be used to the advantage of the service provider and customers who are prepared to pay higher costs.

7.2 Future Directions

There are a few interesting future research directions that are either extension of this work or are motivated by using the proposed model to improve network service.

The following are suggestions that could extend the work done in this thesis:

- The focus in this thesis was modeling and simulation of the proposed method. A useful extension would be implemention on a real IPv6 network. This will ensure measuring the lookup time when only GDI is involved and allow a comparison when the 5-tuples are involved during routing packets.

- Modify the proposed model to use a specified path between the generating node and the destination node to guarantee both lower delay and less packet loss. This path will satisfy two parameters, end-to-end delay and bandwidth. The manager will keep track of all the paths and in the case of path failure another path would be chosen.

- Modify the proposed QoS manager to use the mathematical modeling, end-to-end delay bounds calculation, if a delay is requested. The decision on the QoS request should include the delay parameter.

- Test how the proposed model end-to-end delay calculation changes if the IETF IntServ arrival curve is used to regulate the incoming traffic instead of the leaky bucket.

Bibliography

[1] X. Xiao, L.M.Ni, "Internet QoS: The Big Picture," *IEEE Network Magazine* March/April, pp. 8-18, 1999.

[2] N. Shaha, A. Dessai, and M. Parashar," Multimedia Content Adaptation for QoS Management over Heterogeneous Networks," *Proceedings of the International Conference on Internet Computing (IC 2001),* Nevada, USA, pp 642 - 648, Computer Science Research, Education, and Applications (CSREA) Press, June 2001.

[3] Stardust.com. White Paper - QoS Protocols & architectures, July 1999 `http://www.qosforum.com`.

[4] Fayaz A. Sekib, Stan McClellan, Manpreet Singh and Sannedhik Chakravarthy, *"End-to-End Testing of IP QoS Mechanisms"*, IEEE Magazine 2002.

[5] G. Armitag, *Quality of Service in IP Networks, Foundation for Multimedia Service Internet,* Macmillan Technical Publishing, April 2000.

[6] Zhansong Ma, *Performance and cost Analysis of QoS routing in an Internet,* Master's Thesis October 2000.

[7] Sanjay Jha and Mahbub Hassen, *Engineering Internet QoS* , Artec House Boston, London 2002.

[8] Stephan Schemid, *QoS based Real time Audio Streaming in the Internet,* Master's Thesis 1998.

[9] Karl Ahlin, *Quality of Service in IP Networks,* Master's Thesis, LiTH-ISY-EX-3323-2003, *Linköping* 2003.

[10] Chen-Nee Chuah, *A Scalable Framework for IP-Networks Resource Provisioning Through Aggregation and Hierarchical Control,* PhD Dissertation, Fall 2001, University of California at Berkeley.

[11] Roland Stader, *QoS Provisioning for IP Telephony Networks by Advanced bandwidth management*, Master's of Science Thesis Feb. 2001.

[12] Garcia Macias, *Mobile Communication Architecture with Quality of Service*, PhD Thesis , Jose Antonio 08 January 2002.

[13] Jae-Young Kim, *Edge-to-Edge Throughput Monitoring Methods to manage bandwidth usage in IP Networks supporting Differentiated Services*, PhD Thesis 2002.

[14] J. Wroclawski, *" The use of RSVP with IETF Integrated Services"*, IETF RFC 2210, Sep 1997.

[15] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang and W. Weiss, "An Architecture for Differentiated Services," *RFC2475*, 1998.

[16] Bernet Y. et al, *" A FRame work for Differentiated Services"*, Internet Draft -ietf-diffserv-framework-02.txt. Feb 1999.

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

[18] S. Shenker and J. Wroclawski, *" General characterization Parameters for Integrated Service Network Elements"* RFC 2215 (proposed standard) IETF Sep 1997.

[19] Ibrahim Khalil and Torsten Braun, *"Implementation of a Bandwidth Broker for Dynamic End-to-End Capacity Reservation over Multiple Di.serv Domains 6"*, Springer-Verlag Berlin Heidelberg 2001.

[20] Rob Neilson, BCIT, Jeff Wheeler, Francis Reichmeyer, and Susan Hares, Merit, *"A Discussion of Bandwidth Broker Requirements for Internet2 Qbone Deployment"*, Internet2 Qbone BB Advisory Council, Aug 1999.

[21] Manish Mahajan and Manish Parashar, *"Content Bandwidth Broker for the Differentiated Services Environment"*, `http://www.caip.rutgers.edu/manishm/cabb/cabb.htm`. May, 2003.

[22] QBone Bandwidth Broker Architecture, `http://qbone.internet2.edu/bb/bboutline2.html` [2000, May].

[23] E. Rosen, A. Viswanathan, and R. Callon, *"Multiprotocol Label Switching Architecture"*, April 1999 < draft.ieft-mpls-arch-02.txt >, `http://www.ietf.org/ids.by.wg/mpls.html`, [2003, 20 September].

[24] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell and J. Manus, *"Requirments for Trafffic Engineering over MPLS"*, RFC 2702 IETF, [1999, September].

[25] G. Apostolopoulos, R. Guerins, S. Kamat, A. Orda and S.K. Tripathi, "Intra-Domain QoS routing in IP Networks, A Feasibility and cost/Benefit Analysis", *IEEE Special Issue on Integrated and Differentiated Services for the Internet,* Sep. 1999.

[26] J. Boudec and P. Thiran, *Network Calculus, A Theory of Deterministic Queuing Systems for the Internet*, Online version of the Book Springer Verlag-LNCS 2050, May 10, 2004.

[27] The Network Simulator ns-2 `http://www.isi.edu/nsnam/ns/`, [2003, 13 December].

[28] A. Demers, S. Keshav, and Shenker, "Analysis and Simulation of Fair Queuing Algorithms," *Journal of Internetworking: research and Experience,* vol. 1, pp. 3-26, Jan. 1990.

[29] S. Floyd and V. Jacobson, " Link-Sharing and Resource Management, Model for Packet Networks," *IEEE/ACM Truns. Networking,* vol. 3, no. 4, pp. 365-413, Aug. 1993.

[30] S. Shenker, C. Partridge and R. Guerin, "Specification of Guaranteed Quality of Service." *RFC 2212,* Internet Engineering Task Force, Sep. 1997.

[31] J. Wroclawski, "Specification of the Controlled-load Network Element Service," *rfc 2211*, Internet Engineering Task Force, Sep. 1997.

[32] K. Nicholas, S. Blake, F. Baker and D. Black, "Defination of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," *RFC 2474*, Internet Engineering Task Force, Dec. 1998.

[33] S. Brim, B. Carpenter, and F. LeFaucheur, "Per Hop Behavior Identification Codes," *RFC 2836*, Internet Engineering Task Force, May 2000.

[34] J. Heinanen, F. Baker, W. Weiss, and W. Weiss, "Assured Forwarding PHB Group", *RFC 2597*, Internet Engineering Task Force, June 1999.

[35] V. Jacobson, K. Nichols and K. Poduri, "An Expedited Forwardiung PHB," *RFC 2598*, Internet Engineering Task Force, June 1999.

[36] Geoff Huston, Telstra, *"Quality of ServiceFact or Fiction?"*, The Internet Protocol Journal, Volume 3 No. 1, March 2000.

[37] Multiprotocol Label Switching Architecture (MPLS) *RFC3031*, Jan 2001.

[38] William Stallings, *"MPLS"*, The Internet Protocol Journal, Voulme 4, Number 3, September 2001.

[39] Jnniper. Networks.Inc White Paper-*"Multiprotocol Label Switching Enhanced Routing in the New Public Networks"*. Sep. 1999. `http://www.uniper.net`.

[40] Nortel Networks, *"MPLS-An Introduction to multiprotocol Label Switching"*, `http://wwwnortelnetworks.com/corporate/technology/mpls/collateral.`, April 2003.

[41] Callon, R. et al, *" A frame work for Multiprotocol Label Switching"*. Internet Draft, draft-ieft-mpls-framework-05.txt, Sep. 1999.

[42] Ngo Quynh, Thu, *The Influence of Proportional Jitter Scheduling Algorithms on Differentiated Services Networks*, PhD Thesis 2003, Technical University of Berlin.

[43] E. Dutkiewicz and P. Boustead, "Analysis of per-flow and aggregate QoS in scalable QoS Networks," *ICON 99 Proceedings, IEEE International Conference on*, pp. 289-294, Oct. 1999.

[44] Y. Bernet, P. Ford, R. Yavatkar, F. Baker, L. Zhang, M. Speer, R. Braden, B. Davie, J. Wroclawski and E. Felstaine, "A Framework for Integrated Services operation over DiffServ Networks,", *RFC 2998*, Internet Engineering Task Force, Nov. 2000.

[45] IP Infusion, "Quality of Service and MPLS Methodologies",`http://www.ipinfusion.com/pdf/IP_InfusionQoS_MPLS2.pdf`, [2004, April].

[46] P. Trimintzios, I. Andrikopowo, G. Pavlou and P. Flegkas, "A management and control architecture for providing IP Differentiated services in MPLS-based network", *IEEE Communication Magazine 2001*, pp. 80-88, May 2001.

[47] K. Nicholas, V. Jacobson, and L. Zhang, "A Two-bit Differentiated Architecture for Internet", *Internet Draft*, Internet Engineering Task Force, November 1997.

[48] Internet2 QoS Working Group, `http://www.internet2.edu/qos.` [2005, May].

[49] Ben Teitelbaum (editor), "QBone Architecture (v1.0), " *Internet2 QoS Working Group Draft*, August 1999.

[50] L. Freedman, S. Ni, J. Pinkett and L. Welsh, ECPE 6504, *"Bandwidth Broker"*, March 27, 2001.

[51] A. Ramanathan, M. Parashar, *"Active Resource Management for Differentiated Services Environment"*, IEEE 2002 p.p 78-86.

[52] S. Sohail and S. Jha*" The Survey of Bandwidth Broker"*. UNSW-CSE-TR-02 May 2002.

[53] Eusebi Calle Ortega, *"MPLS dynamic multilevel protection"*, Research Project Girona University 2001.

[54] Nicolas Christin and Jörg Liebeherr, "The QoSbox: A Pc-Router for Quantitative Service Differentiation in IP Networks", Technical Report, University of Virginia, 2001.

[55] Nicolas Christin and Jörg Liebeherr, *"A QoS Architecture for Quantitative Service Differentiation"*, IEEE 2003.

[56] M. Borella, V. Upadhyay and I. Sidhu, Pricing Framework for a differential services Internet , *Communication Networks* 2004.

[57] J. Walrand and P. Variya, *High-Performance Communication Networks*, Morgan Kaufmann Publishers, San Francisco, California 2000.

[58] Matthias Falkner, Michael Devetsikiotis and Ioannis Lambadaris, " An Overview of Pricing Concepts for Broadband IP Networks", *IEEE Communication Surveys 2000*, pp. 2-13, Second Quarter 2000.

[59] Y. Elovici, Y. Ben-Shimol and A. Shabtai, "Per-Packet Pricing Scheme for IP Networks", 2003 IEEE, pp. 1494-1500.

[60] Tianshu Li, Yousef Iraqi and Raouf Boutaba, " Pricing and admission control for QoS-enabled Internet", 2004 Published by Elsevier. `http://www.sciencedirect.com`, [2004, December].

[61] Tianshu Li, Yousef Iraqi and Raouf Boutaba, "Traffic-Based Pricing and Admission Control for DiffServ Networks", pp. 73-86.

[62] El-Bahlul Fgee, Jason D. Kenney, William J. Phillips, William Robertson and S. Sivakumar, "Implementing an IPv6 QoS management scheme using flow label & class of service fields", *Electrical and Computer Engineering, 2004. IEEE CCECE 2004. Canadian Conference,*

[63] J. Boudec and P. Thiran, A short Tutorial on Network Calculus I: Fundamental Bounds in Communication Networks, *IEEE Symposium on Circuits and Sysytems*, May 28-31, 2000, Geneva Switzerland, pp. IV-93-IV-96.

[64] S. Terrasa, S. Sáez and J. Vila, Comparing the utilization bounds of IntServ and DiffServ, `www.comp.brad.ac.uk/het-net/ HET-NETs04/CameraPapers/P10.pdf`, [2005, April].

[65] T. Nyirenda-Jerre and V. Frost, *Impact of Traffic Aggregation on Network Capacity and Quality of Service*, Techincal Report ITTC-FY2002-22730-01, Unversity of Kansas, Nov. 2001.

[66] A. Parekh, R. Gallager, A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case, *IEEE/ACM Transactions on Networking*, Vol. 1, No. 3 June 1993, pp. 344-357.

[67] R. Agrawal, R. Cruz, C. Okino and R. Rajan, Performance Bounds for Flow Control Protocols, *IEEE/ACM Transaction on Networking*, Vol. 7 No. 3 June 1999.

[68] V. Firoiu, J. Le Boudec , D. Towsley and Z. Zhang, "Advances in Internet Quality of Service", `http://www.citeseer.ist.psu.edu/467645.html`, [2004, December].

[69] V. Firoiu, J. Boudec , D. Towsley and Z. Zhang, Theories of Models for Internet Quality of Service, *Proceeding of the IEEE*, Vol. 90, NO. 9, Sep. 2002 pp. 1565-1591.

[70] P. Barta, F. Németh, R. Szabó and J. Bíró, End-to-end delay Calculation in Generalized Processor Sharing Networks, *IEEE 2001* pp. 282-287.

[71] J.-Y. Le Boudec and A. Charny, Packet scale rate guarantee for non-FIFO nodes, Tech. Rep. DSC200138, EPFL-DSC, `http://dscwww.epfl.ch./EN/publications/documents/tr01 038.pdf`, July 2001.

[72] R. Guérin and V. Peris, Quality-of-Service in Packet Networks Basic Mechanisms and Directions, *Special issue on Internet telephony*, Volume 31 , Issue 3, 1999, pp. 169 - 189.

[73] A. Millet and Z. Mammeri, Delay bound guarantees with WFQ-based CBQ discipline, *Quality of Service, 2004, IWQOS 2004 Twelfth IEEE International Workshop,* 7-9 June 2004 pp. 106 - 113.

[74] Fgee, E.-B.; Kenney, J.D.; Phillips, W.J.; Robertson, W.; Sivakumar, S., "Comparison of QoS Performance between IPv6 QoS Management Model and IntServ and DiffServ QoS Models", *Communication Networks and Services Research Conference, 2005*, Proceedings of the 3rd Annual 18 May 2005 Page(s):287 - 292

[75] G. Rizzo and Le Boudec, "'Pay bursts only once' does not hold for non-FIFO Guranteed rate nodes", Technical report Jan. 13, 2005.

[76] Fgee, E.-B., Phillipsi, W.I., Robertson, W., Sivakumar, S.C., " Implementing QoS capabilities in IPv6 networks and comparison with MPLS and RSVP," *Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference,* vol 2 , May 4-7, 2003, pp. 851 - 854.

[77] Kishor Shridharbhai and Trivedi *P*robability and statistics with reliability, queuing, and computer science applications , Englewood Cliffs, NJ : Prentice-Hall, 1982.

[78] Marc Geris, "RSVP patch for ns-2.26", `http://www.ncc.up.pt/~prior/nsindex-en.html`, [2004, September].

[79] Gaeil Ahn and Woojik Chun, "Design and Implementation of MPLS Network Simulator (MNS)" `http://flower.ce.cnu.ac.kr/~fog1/mns/`, [2004, October].

[80] Safiullah Faizuullah, "A Pricing Framework for QoS Capable Intener", in PDCS 2000, Las Vegas Nevada, Nov. 6-9 2000, pp. 569-577.

[81] Ron Cocchi, Deborah Estrin, Scott Shenker, and Lixia Zhang, "A study of priority pricing in multiple service class networks", *ACM SIGCOMM Computer Communication*, Volume 21, Issue 4, September 1992, pp. 123-130.

[82] J. K. Mackie-Masson and H. R. Varian, "Pricing the Internet", *Int'l Conf. Telecommunication Systems Modelling*, Nashville, TN, USA, March 1994, available from URL `http://www.spp.umich.edu/papers/listing.html`, pp. 378-93.

[83] S. Shenker et al., "Pricing in Computer Networks: Reshaping the Research Agenda", *ACM Computer Communication Review*, 1996, pp. 19-43.

[84] X. Waog and H. Schulzrine, " Pricing Network Resource for Adaptive Applications in a Differentiated Service Network", In Proceding of INFOCOM 2001, Anchorage, Alaska, April 2001.

[85] E. Fgee, S. Sivakumar, W. Phillips, W. Robertson and J. Kenney, "Implementing a Dynamic Pricing Scheme for QoS enabled IPv6 Networks", *Proceedings of the seventh International Conference on Enterprise Information Systems*, May 25-28, 2005, Miami Florida, Vol. IV, pp. 289-292.

[86] White Paper, *Bringing Comperhensive Quality of Service Capabilities to Next-Generation Networks,* `http://e-www.motorola.com/files/netcomm/doc/white_paper/QOSM-WP.pdf`, [2003, November].

[87] S. Bhatti & J. Crowcroft, *"QoS-Sensitive flows: Issues in IP packet handling"* IEEE/Computer Internet Computing, July/Aug. 2000 pp. 48-57.

[88] Leonardo Balliache, *" Network QoS using Cisco HOWTO"*, `http://www.opalsoft.net/QoS/whyQoS.html`. [2004, April].

[89] `http://ipsit.bu.edu/Sc 546/spring 2003/wfq/wfq.html`. [2004, January].

[90] `http://www.nwfusion.com/links/Encyclopedia/c/656.html`. [2004, January].

[91] P.P. White, *"RSVP and Integrated Services in the Internet: A Tutorial"*. IEEE Communications Magazine, Pages 100-106 May 1997.

[92] R. Branden, D. Clark, S. Shenker "Integrated Services in the Internet Architecture: an Overview," *RFC1633*, 1994.

[93] Pekk Pessi, *"RSVP and the Internet Integrated Services"* 1997. `http://www.tonl.hut.fi/Opinnot/Tik-110.551/1997/rsvp.html`, [2004, December].

[94] Ray Hunt, *"IP Quality of Service Architectures"*, pp. 338-343 IEEE 2001.

[95] V. Jacobson and K. Nichols Cisco Systems, K. Poduri Bay Networks, "An Expedited Forwarding PHB," *RFC2598*, Jnue 1999.

[96] Y. Bernet, D. Durham, and F. Reichmeyer, *Requirements of Diff-serv Boundary Routers,* IETF Internet-Draft, draft-bernet-diffedge-01.txt, November 1998.

[97] Y. Bernet, A. Smith, S. Blake, and D. Grossman, *"A Conceptual Model for Diff-serv Routers,* IETF Internet-Draft, draft-ietf-diffserv-model-03.txt, May 2000.

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

[99] C. Dovrolis and P. Ramanathan, *"A case for relative Differentiated Services and the Propertational Differentiation Model"*, IEEE Network, October 1999.

[100] M. Brunner and J. Quittek, *"MPLS Management using Policies"*, IEEE 2001, pp 515-528.

[101] The International Engineering Consortium-Tutorial, *"Multiprotocol Label Switching (MPLS)"*, `http://www.iec.org/online/tutorial/mpls/`, [2003, March].

[102] Jerry Ryan, The Technology Guide Service, *"Multiprotocol Label Switching"*. `http://www.techguide.com`. Applied Technology Group 1998.

[103] David Lee, Daniel Lough, Scott Midkiff, Nathaniel Generation and Phiilip Benchoff, *"The Next Generation of the Internet: Aspects of the Internet Protocol version 6"*, pp. 28-33 IEEE Network Jan/Feb 1998.

[104] William Stallings, *"IPv6: The New Internet Protocol"*, pp. 96-108 IEEE Communication Magazine, July 1996.

[105] Latif Ladid, *"IPv6 on Everyting: The New Internet IPv6 helps Network Architects address the IP address shortage, security, QoS, multicast and management"*, pp. 317-322 3G Mobile Communication Technologies 26-28 March 2001.

[106] Hi Huang and Jian Ma, *"IPv6-Future Approval Networking"*, pp. 1734-1739 IEEE 2002.

[107] Jianbo Xue, *"Adaptive QoS-Supporting Architecture for Real-time Application in wireless IP Network"*, Master's Thesis January 2002.

[108] Loukola, M.V. and Skyttä, J.o, *"New possibilities offered by IPv6"*, `http://www.hut.fi/~mloukola/pub7/p1.pdf.` , November 2003.

[109] Douglas E. Comer, *Internetworking With TCP/IP Vol.1: Principles Protocols And Architecture: Principles, Protocols, and Architecture, Volume 1,* Published by Pearson Education Canada 2000.

[110] ipinfusion. White paper, *"IPv6 Network Processing"*, 2001.

[111] Mark A. Miller, *Implementing IPv6 migrating to the Next Generation Protocol,* MT&T Books 1998.

[112] Gray K. Kessler, *"IPv6: The Next Genertaion Internet Protocol"*, Feb. 1997. `http://www.graykessler.net/library/ipv6-exp.html.`, April 2003.

[113] S. Deering and H. Rinden, *"Internet Protocol Version 6 (IPv6)"*, Dec. 1995.

[114] J. Rajahalme, A. Couta, A. Carpenter and S. Deering, *"IPv6 flow lable specification"*, < draft-ietf-ipv6-flow-label-0.1.txt >, March 2003.

[115] William Stallings, *High-Speed Networks TCP/IP and ATM design principles*, Prentice Hall 1999.

Appendix A

QoS Regulators and Schedulers

A.1 Traffic Regulators

Traffic regulators are used to police and to regulate traffic at the network entering nodes. The most common regulator is the leaky bucket. To understand the leaky bucket mechanism, imagine a bucket with a small hole at the bottom. It does not matter at what rate the water enters the bucket, it can only leak out at the rate determined by the hole at the bottom of the bucket. Thus a leaky bucket reshapes the flow of water to the rate determined by the hole. For policy shaping the leaky bucket could be implemented using a counter. The counter holds tokens where each token may represent a cell/packet or a certain number of bytes. Tokens are added to the counter at fixed intervals of time and decremented as the data flows through. If there are no tokens available, or the tokens that are available do not cover the entire length of the data, the cell/packet would be buffered and not allowed to enter the network until sufficient tokens have accumulated. The number of tokens that can accumulate in the counter is generally referred to as the leaky bucket depth. If the flow is idle, then tokens may accumulate to the extent determined by the QoS policy and the bucket depth. The amount of tokens accumulated represents the burst size that may be admitted into the network. By controlling the depth of the bucket, the network could regulate the permissible burst size. For example, if a flow is to

be shaped at a particular Time Division Multiplexing (TDM) type peak rate, then the rate at which the tokens are added to the counter would specify the peak rate. By keeping the bucket depth such that it only allows a single cell/packet worth of a token. The network can thus shape a flow to a particular profile by adjusting the rate at which tokens are added to the counter as well as the number of tokens that can be accumulated. Often times it is desirable to shape to the peak as well as the average rate. This would require the use of two buckets. One bucket regulates the peak rate and the other one regulates the average rate. A cell/packet would enter the network if there are enough tokens in both the buckets and would be buffered otherwise. This arrangement is commonly referred to as the dual leaky bucket configuration as shown in Figure ((A.1) [86].

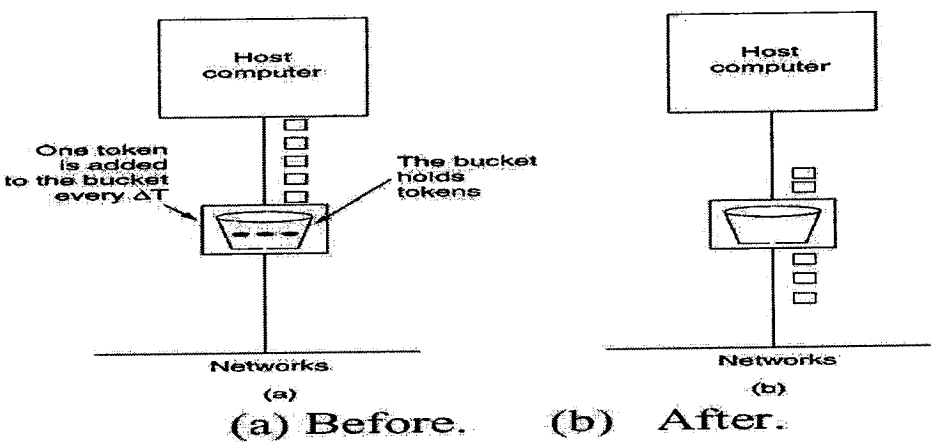

FIGURE A.1: Leaky Bucket Model

A.2 Traffic Schedulers

Traffic packets are processed by a routing engine (switch fabric) before they are shaped and then forwarded to the next hop. Therefore, packets are stored for each interface in output queues. Figure A.2 shows a very simple router diagram [87]. The main role

of the packet scheduler is to decide in what order the incoming packets are put into the output queues and lines. In other words, packet schedulers manage the output queues allowing resources to be allocated according to the established policies.
Several scheduling and queuing techniques have been developed to provide specific bandwidth, delay and packet loss to particular flows in each node.
A brief review will be given for two queuing algorithms, class based queuing (CBQ) and weighted fair queuing (WFQ), that are used in the thesis.

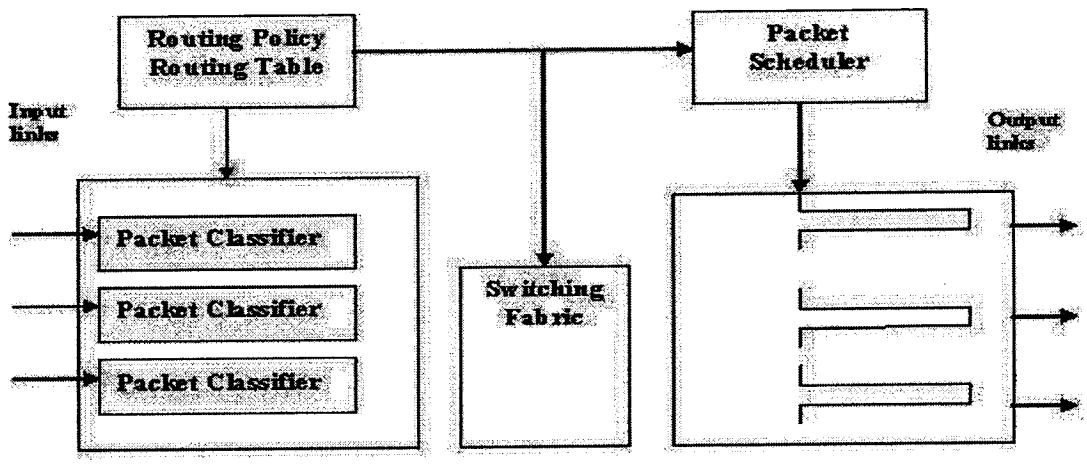

FIGURE A.2: Basic structure of a QoS capable router

A.2.1 Weighted Fair Queuing :

Weighted Fair Queuing WFQ [88] [89] [7] is an approximation of GPS (Generalized Processor Sharing), which does not make the assumption of infinitesimal packet size. GPS is the ideal method for best-effort and interactive connections but it is impossible to implement due to its requirement of infinitesimal sized packets. WFQ is a complex scheduler used for various size packets. It provides traffic priority management that automatically sorts among individual traffic streams without requiring an access list. Its basis is on fluid-flow fair queuing. WFQ uses a servicing algorithm that attempts to provide a predictable response time and negate inconsistent packet transmission

timing. It does this by sorting and interleaving individual packets by flow, and queuing each flow based on the volume of traffic in this flow. Using this approach each queue is serviced fairly in terms of byte count creating bit-wise fairness. WFQ is used under conditions that require consistent response time to heavy and light network users alike without adding excessive bandwidth (BW). Two categories of WFQ sessions are interactive traffic, low bandwidth, and best effort traffic, high bandwidth. Low BW traffic has priority over high BW traffic. WFQ also ensures that queues do not starve for BW, thus providing predictable service. Any bandwidth not used by a flow will be proportionally divided up among remaining flows. In WFQ, packets are classified by flows. Packets usually are classified based on their source IP address, their destination Internet protocol, IP address, source Transmission Control Protocol, TCP, or User datagram Protocol, UDP port, or the destination TCP or UDP port numbers. The virtual finish time is defined as the time that the GPS would have finished sending a packet. The virtual finish time of a packet that is arriving in a WFQ, is equal to all of the finishing times of packets ahead of it in its queue, plus all higher priority packets with higher weights plus the arriving packet's size (in bits). If an arriving packet of size 10 bytes reaches a queue which is occupied by a packet with a virtual finishing time of 20 secs and these higher priority packets with larger weights, it will have a virtual finishing time of 30 secs. WFQ helps solve problems with round-trip delay variability. Therefore using WFQ avoids the spread of transfer rates and inter-arrival periods when multiple high-volume flows are active. Also when bursty traffic is added to the network, response time is still predictable. Using WFQ increases predictability in throughput and response time and provides predictable inter-arrival periods. Figure A.3 shows an example of interactive traffic delay. WFQ works with both the IP precedence and Resource Reservation Protocol (RSVP) to help provide QoS. RSVP can use WFQ to allocate buffer space, schedule packets, and guarantee BW for a reserved flow. WFQ is also IP precedence aware, and detects higher-priority packets marked with precedence by the IP header and schedules them with bigger weights. The IP precedence values range from 0 to 7.

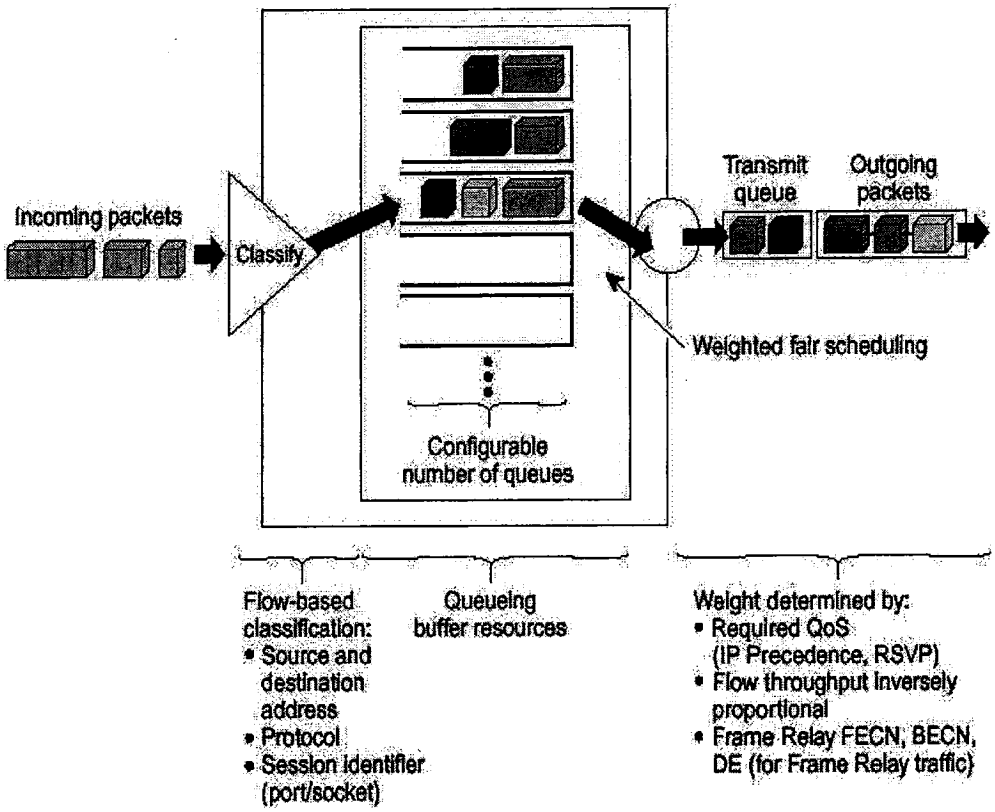

FIGURE A.3: WFQ

More BW is allotted to a flow as its precedence value increases. This provides faster service for the flow when congestion occurs. The following describes the effect of IP precedence settings:

If there are eight active flows, precedence levels of 0,1,2,3,4,5,6,7, then they will get $1/36(1+2+3+4+5+6+7+8=36), 2/36, 3/36$, and so on of the total bandwidth.

However, if there are 18 precedence 1 flows and 1 of each of the others, then the flows will get $1/70(1+2(18)+3+4+5+6+7+8=70), 2/70, 2/70, ., 3/70$, and so on.

A.2.2 Class-Based Queuing :

Class-Based Queuing CBQ [7] [90] is a traffic management algorithm developed by the Network Research Group at Lawrence Berkeley National Laboratory as an alternative to traditional router-based technology. Now in the public domain as an open technology, CBQ is deployed by companies at the boundary of their WANs. Network managers can use CBQ to easily classify traffic to meet business priorities and to ensure each traffic class has the appropriate quality of service. CBQ integrates easily with a company's existing network to protect its investment and to provide IT (Information Technology) managers with more control over the network, thus reducing bandwidth costs. The concept behind CBQ is simple, it divides user traffic into a hierarchy of classes based on any combination of IP addresses, protocols and application types. A company's accounting department, for example may not need the same Internet access privileges as the engineering department. Because every company is organized differently and has different policies and business requirements, it is vital for traffic management technology to provide flexibility and granularity in classifying traffic flows. CBQ lets network managers classify traffic in a multilevel hierarchy. For instance, some companies may first identify the overall needs of each department or business group, and then define the requirements of each application or group of applications within each department. For performance and architectural reasons, traditional router-based queuing schemes are limited to a small number of classes and only one-dimensional classification is allowed. By providing network managers with better control over user traffic, CBQ lets companies meet the needs of response-time-sensitive applications, supports service-level agreements and keeps inappropriate traffic off the network. CBQ operates at the IP network layer which allows it to provide the same benefits across any Layer 2 technology and to be equally effective with any IP protocol, such as TCP and UDP. It also operates with any client or server TCP/IP stack variation, since it takes advantage of standard TCP/IP flow control mechanisms to control end-to-end traffic. Bandwidth is the largest cost in wide-area networking. CBQ lets network managers define bandwidth allowances in absolute

terms, unlike router-based schemes, which provide a rough percentage to best-effort traffic classes. Figure (A.4) shows how packets are classified according to their priorities.

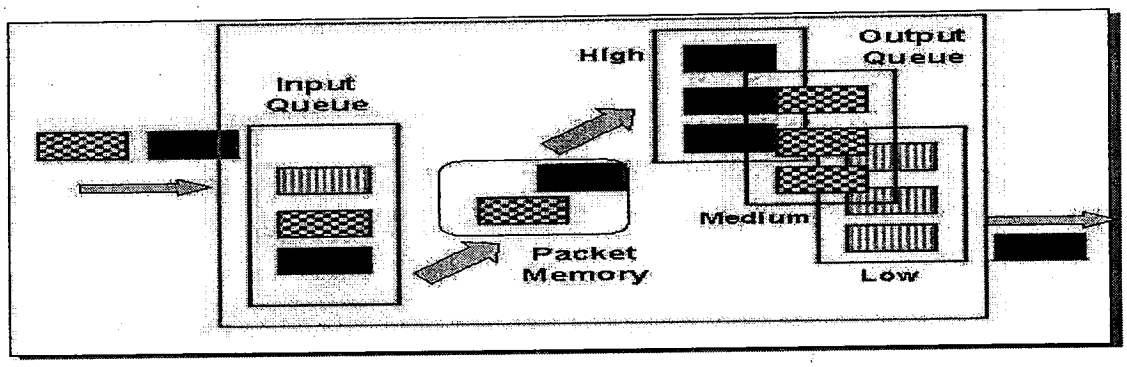

FIGURE A.4: CBQ

Appendix B

QoS Approaches used in Literature

Currently, the IETF (Internet Engineering Task Force) is working on three different approaches, Integrated Service model (IntServ), Differentiated Service model (DiffServ) and Multi-protocol Label Switching Protocol (MPLS), to provide QoS for the Internet. Two of these methods, IntServ and DiffServ will be described next in details since they were compared with the thesis proposed model.

B.1 Integrated Services (IntServ) and RSVP

The Integrated service, IntServ, framework is aimed at providing per-flow QoS guarantees to individual application sessions. It is intended to provide the closest thing to circuit emulation on IP Networks. Also, it represents a significant departure from the best effort service of the Internet by attempting to provide a high level of QoS in terms of service guarantees, granularity of resource allocation, and details of feedback [91] [92] [18]. It defines several classes of service along with the existing best effort service. The main idea behind this framework is that applications should be able to choose a particular class based on their QoS requirements. IntServ has classified applications into three categories [7] as follows:

1. **Elastic Applications:-**

   These applications are flexible in terms of their QoS requirements.

2. **Tolerant real-time applications:-**
   Timeliness is very important for this category of applications.

3. **Intolerant real-time Applications:-**
   Intolerant real-time applications demand more stringent QoS from the network. These applications have precise bandwidth, delay and jitter constraints.

B.1.1 QoS Classes in IntServ

The Integrated service model defines two service classes on the top of best effort service, namely controlled load and guaranteed services, to meet the requirement of the applications mentioned above.

1) *control load service class*:- this is designed to handle tolerant real-time applications that require a sufficient amount of bandwidth and can tolerate occasional delays and losses.

2) *guaranteed service class*:- The guaranteed service provides a framework for delivering traffic for applications with a bandwidth guarantee and a delay bound. Service specification, a source's traffic characteristics are provided by specifying traffic parameters such as data rate and burst rate.

B.1.2 IntServ Traffic Control Model

Each router or a node in the IntServ model as seen in Fig (B.1) has the following components in order to reserve resources [77] [7] [6]:-

1- **Admission Control**
   The main purpose of admission control in a network domain is to determine whether access to resources available at the domain elements (routers) can be guaranteed to a new flow without affecting others. It is invoked at each router along the path of the new flow.

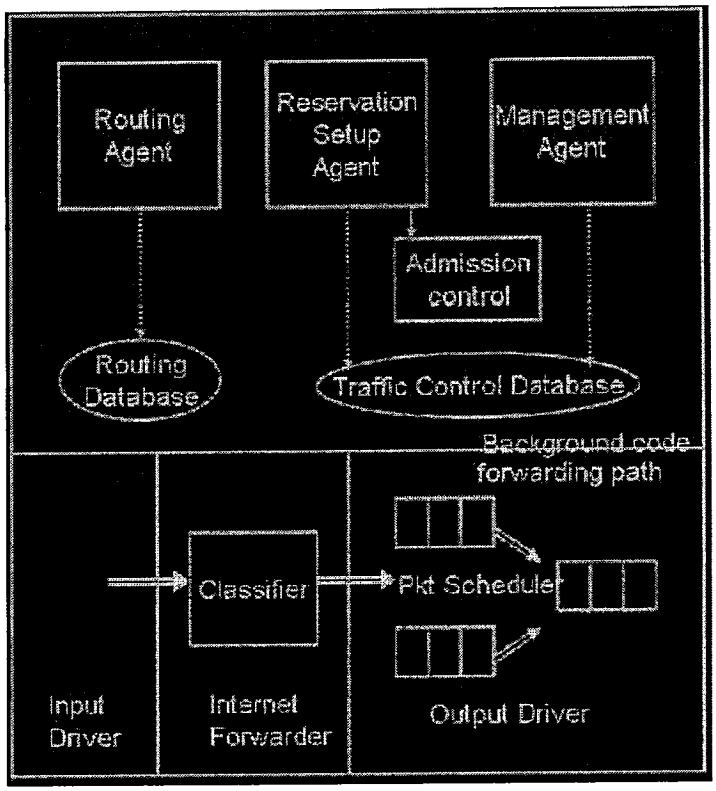

FIGURE B.1: IntServ Control Model

2- **Policing and shaping**

Policing is a set of actions performed by a network element when a flows' actual data traffic characteristics exceed the negotiated values given in the flows' traffic specification (Tspec). Network elements mark the violated packets as non-conformant to be dropped or delayed more.

3- **Packet Classifier**

A packet classifier on a network element is responsible for identifying packets corresponding to a particular flow in order to provide special treatment to the packets. Packet classification can be performed by looking at the source and destination hosts addresses, protocol number, and port field.

4- **Packet Scheduler**

A packet scheduler is responsible for ensuring that the flows identified by a packet classifier receive the negotiated QoS guarantees.

5- **Reservation setup Agent**

The reservation setup protocol is necessary to create and maintain the flow specific state at the end point hosts and routers along the path of the flow.

B.1.3 RSVP: a signaling protocol for IntServ

RSVP is used by a host, on behalf of an application data stream, to request a specific QoS from the network for particular data streams or flows. RSVP protocol is also used by routers for:

1. Delivering QoS control requests to all nodes along the path(s) of the flow.
2. Establishing and maintaining a reservation state [92] at all nodes that accept the QoS request.
3. Permitting reservation of additional resources within a current connection.
4. Freeing resources that are not required any more.

RSVP is called a "**soft state**" protocol which is defined as a state in routers and end nodes that can be updated by certain RSVP messages. Soft states are created and periodically refreshed by path and reservation messages. A soft state is deleted if no matching refresh messages arrive before the expiration of a cleanup timeout. Also, a soft state can be deleted as the result of an explicit teardown message [3]. Each node has to individually save, update and maintain all states of its flow. This is achieved by sending **PATH** and **RESV** messages between senders and receivers.

RSVP Data Structures and Messages [11] [3]

RSVP defines the following data structures used for objects within RSVP messages:-

* **Tspec:** Describes data traffic attributes transmitted by a sender. It includes parameters such as data transfer rate, peak rate and maximum burst size.

* **Filterspec:** Contains the IP address and port numbers of all permitted senders within a session.

* **Sender Template:** Contains sender's IP address and possibly some additional information to uniquely identify this sender.

* **Flow Sec:** Describes the desired QoS parameters in a RSVP message.

RSVP Messages [11] [91] [7] [93]
RSVP has two main messages : **PATH** and **RESV**. The source and receiver nodes transmits PATH and RESV messages every 30 seconds. RSVP messages travel hop-by-hop and the next hop is determined by the routing table. Routers remember where the message came from and maintain this state (route pinning). A PATH message contains the following objects and structures:

⋆ Sender Tspec: Traffic parameters of the flow along the network path.

⋆ Sender Template: Identifies the sender.

⋆ Phop - Address of the previous hop. Contains the IP address of the next upstream RSVP node. Each node places its own address in this field.

The path messages store a *path state* structure in each node along the path; the state includes all relevant data of the sender.
Reservation (RESV) messages are generated by the receiving nodes upon receiving a PATH message. RESV messages contain a request for resources to be reserved. The resource reservation request is expressed by filter specification (Filterspec) and flow specification (Flow Spec). Filter specification defines the packets in the flow that will receive a specific class of service which helps later in the packet classification processes. Flow specification, defined by the QoS parameters requested by the sender, is used by a packet scheduler to insert packets in the proper queues. Other RSVP messages are as follows :

1. Path Teardown is initiated by senders and nodes release states upon receiving this message.

2. Resv Teardown are initiated by receivers or any node in which time out is occurred. Nodes delete the matching reservation state when they receive this message.

3. Path Error is generated by a node when an error occurred during processing of a path message and was reported to the sender.

4. Resv Error is generated by a node when an error occurred during processing of a Resv message and was reported to the receiver.

5. Confirmation messages are sent as acknowledgments for successful reservation requests.

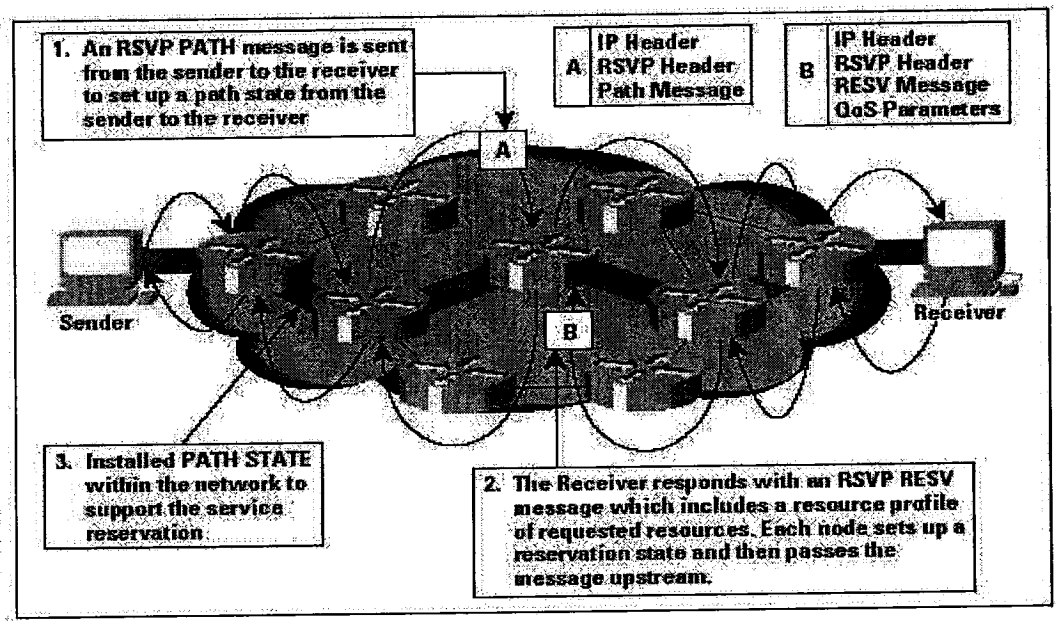

FIGURE B.2: RSVP setup Procedures

An application in a sending host initiates the RSVP signaling flow and the important objects such as Sender Template and TSPEC information is placed into the PATH message. The path is determined by the dynamic local routing algorithms, such as OSPF, that run on each router in the network domain. Each router receives the PATH message, installs a path state and records the IP address of the upstream router. A receiver generates RESV message whenever it receives the PATH message. The RESV message contains two parts, FILTERSPEC that contains the address and port number of the sending application and Flowspec that contains the traffic and QoS information that the receiver application is requesting. The RESV message travels using the upstream route (reverse path of incoming PATH message) to the sender. The network element (router), upon receiving a RESV message, makes a check to find a valid installed PATH state, and performs the admission control. The result of admission control is either to accept (install state and allocate resources) or reject the request and generate error messages. After receiving the RESV message, the application can perform its data transmission. The procedures are shown in Figure B.2 [36].

B.2 Differentiated Service Architecture

The Differentiated Service , DiffServ, is a set of technologies that are used to provide quality of service in a world of best effort service provision [51]. DiffServ is a bridge between IntServs' guaranteed QoS requirements and the best effort service offered by the Internet today [15]. In this scheme, the complexity is pushed out to the edge routers and the core routers are maintained as simple as possible. DiffServ architecture [15] [16] is based on a simple model where the traffic entering a network is classified and possibly conditioned at the boundaries of the network and then assigned to different behavior aggregates. Therefore, individual micro flows are classified at the edge routers into one of the following [11] [7] [94] classes defined by the approach

that have been defined by the IETF DiffServ group.

I **Expedited Forward (EF)** [95] : This class is referred to as the Premium Service class. It supports low loss, low delay, low jitter and assured bandwidth connections. Ingress routers have to ensure that EF traffic is burst free and conforms to the specified rate.

II **Assured Forwarding (AF)**: AF is a group of classes designed for customers who require an improved QoS over best effort. AF is actually aims to offer different levels of packet forwarding assurances during network congestion. It also tries to offer a service that does not guarantee bandwidth.

The classification is done at the ingress router based on one or more bits in the packet. Then the packet is marked, using code points, as belonging to one of the classes and injected into the network. The core routers that forward the packet examine this marking and use it to decide how the packet should be treated. Therefore, most of the work is done at the edge routers, especially packet classification. Packets are classified at the edge routers using a multifield classifier and a traffic meter. The traffic meter is used to measure the traffic to ensure that the traffic conforms to the traffic profile previously agreed upon by the network provider and the customer. The classifier first extracts the IP header fields that contain the source and destination addresses and port numbers to do the multifield classification. Then the ToS field is extracted to do behavior aggregate classification. Packets are then marked with the DiffServ code point (DSCP) which uses six bits of the IPv4 or IPv6 header to convey the DSCP. Then one of the DiffServ pre-defined classes is selected based on the QoS requirements for the per hop behavior (PHB) classification. Therefore, PHB is defined as the certain behavior a packet may receive at each hop. All packets with the same code point are grouped together and are known as a behavior aggregate (BA).

B.2.1 DiffServ Routers

A DiffServ router is a fundamental DiffServ-enabled network node. The conceptual model and requirements of the DiffServ routers are discussed in [96] [97].A DiffServ router is considered to have a routing module, a set of Traffic Control Blocks (TCBs), a queuing module, and a configuration and monitoring module as shown in Figure (B.3).

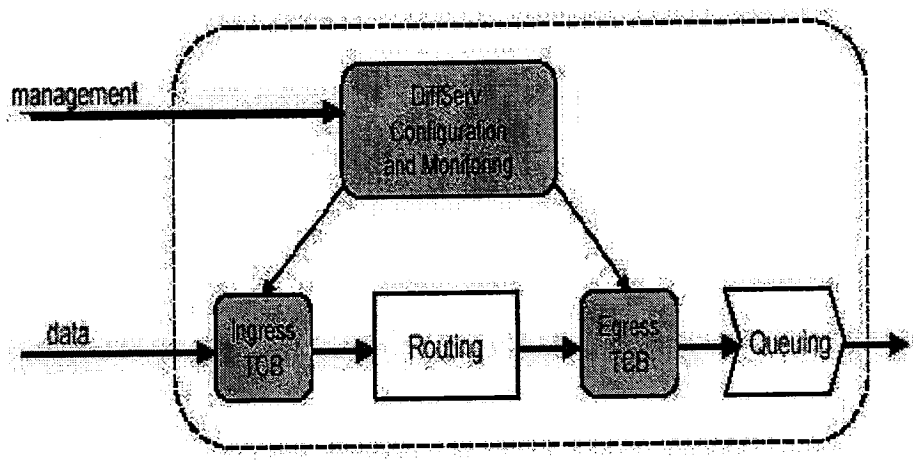

FIGURE B.3: Conceptual Model of a DiffServ Router

A DiffServ-enabled network node has a cascaded set of traffic conditioning blocks (TCB) for handling DSCP-marked network packets. A traffic conditioning block is a minimum logical element that controls DiffServ packets passing through the DiffServ network node. It receives packets from the network and classifies them into a predefined set of traffic aggregates by looking up the DSCP value in the packet headers. Each traffic aggregate is metered, marked, shaped, or dropped separately. Figure (B.4) illustrates four components of a traffic conditioning block [98] [7]. Functions of these components are:-

- *Classifier*: The classifier selects packets based on the values of one or more

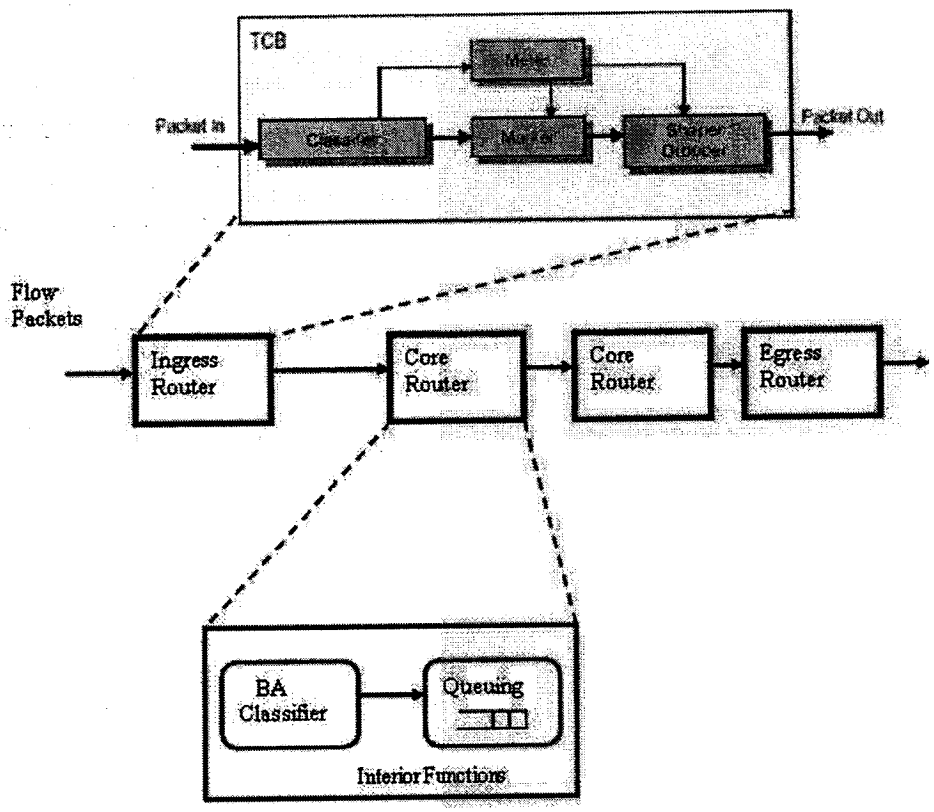

FIGURE B.4: DiffServ core and edge routers structures

packet header fields. Two types of classification are supported by DiffServ:-

- *Multifield* **(MF)** *classification*: Supports classification based on multiple fields. It is required at the edge of IntServ router connecting DiffServ domain. The MF flows need to be marked by the appropriate DSCP.
- *Behavior aggregate* **(BA)** *classification*: Sorts packets based on ToS (type of service) or CoS (class of service) field that contains the DSCP.

- *Marker*: The job of the marker is to insert the appropriate DSCP value in the DS bytes that specifies the appropriate aggregate flow and determine which forward treatment the packet should receive allowing packets to receive the appropriate service (PHB) needed in order to perform only BA classification.

- *Meter*: A meter is used to compare the incoming flow with the negotiated traffic profile. It passes the violating packets to the shaper and dropper, or remarks them with a lower grade of service using different DSCPs.
- *Shaper*: This module is used to introduce some delay in order to bring the flow into compliance with the profile. It has a limited size buffer used to buffer a burst of traffic flows before resending them at an acceptable rate to the next hop.
- *Dropper*: A dropper performs a policing function by simply dropping the packets that are out of profile. Traffic metering, shaping, and dropping are collectively refereed to as policing.

The combination of these components is called a **traffic conditioner**. It is used to facilitate building a scalable DiffServ network. The core routers does not need to maintain per-flow state information as the classification is performed based on BA. Therefore, packets are scheduled and classified according to the marking of the DSCP field.
Figure (B.5) shows a DiffServ model that contains egress and ingress nodes, core nodes and a bandwidth broker [19].

B.2.2 Resource allocation in DiffServ Domain using BB:

1. A host application sends a request for a QoS traffic flow with its specification, burst rate, average rate, max rate, and delay, to the bandwidth broker of its domain.
2. The domain BB first checks if the destination node is located in its domain, since it has a global knowledge of the domain topology. The user data unit then verifies if there are enough resources to accept a new QoS request. This shows that the BB control the resources and the topology of its domain, admission control functionality.

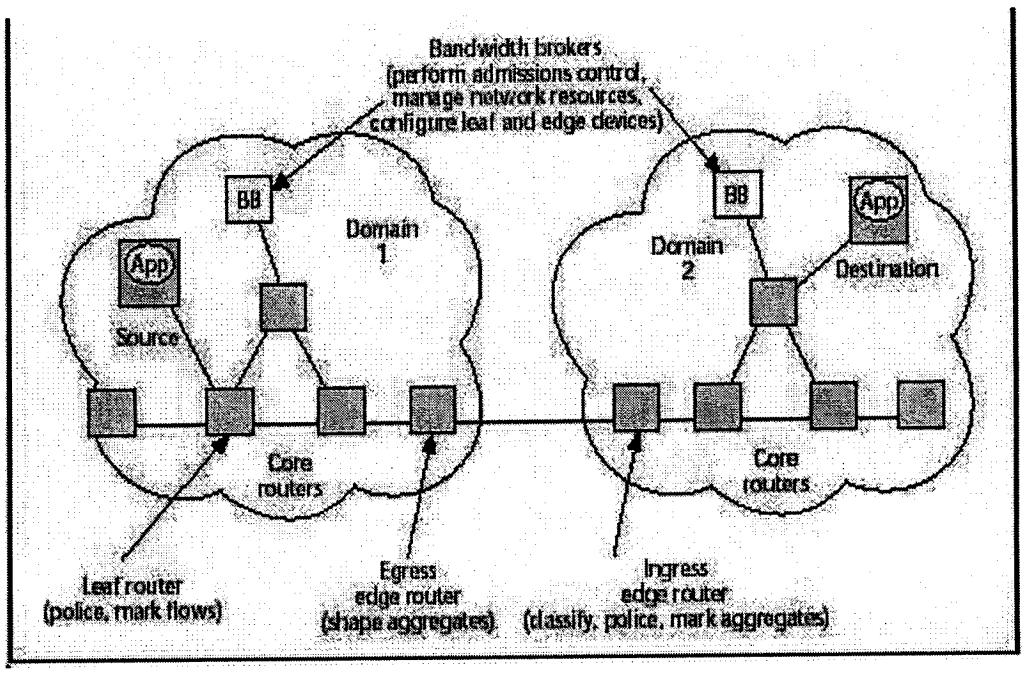

FIGURE B.5: A generic model for Differentiated Service Domain

3. The BB configures the right edge router or leaf router to send the guaranteed traffic flows specifications for classification and scheduling purposes.

4. The ingress router uses these parameters to classify and mark the traffic packets by inserting the DSCP. It also monitors the traffic flow for any violations for which action has to be taken. The remarked or violated packets are either dropped or their priority degraded into lower class depending on the policy set by the domain BB.

5. The DSCP and the traffic specification are stored in the BB database for the purpose of monitoring the resources of the domain.

6. If the destination address is not located in the BB domain, the request is forwarded to the neighboring BB which checks if the address is located in its domain. It also checks resource availability before responding to the request.

Table B.1: IntServ and DiffServ architecture comparison

	IntServ	**DiffServ**
Granularity of service differentiation	Individual flow	Aggregates flows
State maintenance in routers (eg. buffers, scheduling)	Per-flow	Per-aggregate
Packet classification	Several header fields (the 5-tuple)	DS byte of IP header
Type of service differentiation	Deterministic or statistical guarantees	Absolute or relative assurances
Admission control	Required	Required only for absolute differentiation
Signaling protocol	Required (RSVP)	Not required for relative schemes. Absolute schemes need semi-static reservations or broker agents
Coordination	End-to-End	Local (per-hop)
Scalability	Limited by number of flows	limited by number of classes of service
Network accounting	Based on flow characteristics and QoS requirements	Based on class usage
Network management	similar to circuit-switched network	Similar to existing IP networks
inter-domain deployment	Multilateral agreements	Bilateral agreements

Comparison between DiffServ and IntServ Table (B.1) summarizes the differences between DiffServ and IntServ [99].

Appendix C

Next Generation Internet Protocol IPv6

C.1 Introduction

The rapid growth of the current Internet, which operates using the Internet Protocol version 4(IPv4), has created a number of problems for the administration and operation of the global network. These problems include the decreasing number of available IPv4 addresses for network nodes, and the rapid growth of memory and performance requirements for network routers. While changes to IPv4 have extended the life of the current Internet, these changes tend to create new problems and require a significant amount of overhead for network administration. The Internet Protocol version 6(IPv6) has been designed to support these extensions without creating additional problems [103] [104]. There are two significant important components of the IPv6 protocol that may be in fact provide a method to help deliver QoS. The first component is the 8-bit priority field in the IPv6 header, which is functionally equivalent to the IP precedence bits in the IPv4 protocol specification, with somewhat of an expanded scope. This field can be used to identify and discriminate traffic types based on the contents of this field. The second component is the flow label which has been added to enable the labeling of packets that belong to particular traffic flows for

which the sender's request may need special handling [105].

C.2 Some Aspects of IPv6

In the following subsections, some of the useful aspects of the IPv6 that gives advantages in using this protocol are presented in this section.

C.2.1 Addressing

With a 32-bit address field, it is in principle possible to assign 2^{32} different addresses, which is over 4 billion possible addresses. This number of addresses is not adequate for the following reasons [104]:-

- The two level structure of the IP address (network number and host number) is convenient but wasteful of address space. Once a network number is assigned to a network, all of the host-number addresses for that network number are assigned to that network.
- Growth of TCP/IP usage in new areas will result in a rapid growth in the demand for unique IP addresses.
- Typically, a single IP address is assigned to each host. A more flexible arrangement is to allow multiple IP addresses per host; this increases the demand for more IP addresses.

To meet these needs, IPv6 uses 128-bit addresses instead of 32-bit addresses of IPv4. This is an increase of address space by a factor of 2^{96}.

C.2.2 Performance

Both LANs and WANs have progressed to ever greater data rates pushing for gigabits traffic rates. Also, as more services, especially graphics based services, become available over the Internet. This makes having routers that process and forward IP

packets fast enough to keep up with the traffic flow an important and urgent matter in Inter networking.

Three aspects of IPv6 design contribute to meeting performance requirements [104] [8]:-

I- The number of fields in the IPv6 packet header are reduced from IPv4. A number of IPv6 options are placed in separate optional headers located between the IPv6 header and the transport layer header. This simplifies and speeds up the routing of IPv6 header packets compared to IPv4 datagrams since these optional headers are not examined or processed.

II- The IPv6 packet header is fixed length whereas the IPv4 header is variable length (again simplifying the processing).

III- Packet fragmentation is not permitted by IPv6 routers , although it is in IPv4. Fragmentation in IPv6 is performed by the source. A discovery packet is sent to the destination node and the maximum transmission unit (MTU) is returned back to the source node where packets are fragmented at that size.

C.2.3 Network service

In IPv6 it is possible to associate packets with particular service classes, perform routing functions on the basis of those classes and allow the network along the route to make use of this class information. This means IPv6 is able to support real-time services and to specify priority levels to determine discard strategy in the event of congestion. IPv4 provides minimal support in this area. Also, IPv6 enables labeling of packets belonging to a particular traffic flow when a sender requests a special handling [104] [106].

C.2.4 Addressing Flexibility

IPv4 is best employed for unicast addressing: a single address bit pattern corresponds to a single host. Other forms of addressing are poorly supported, partly because the

address size is limited to 32 bits and no provision is made for certain addressing modes. IPv6 includes the concept of an *anycast address* for which a packet is delivered to just a one node in a set of nodes. The scalability of the multicast routing is improved by adding a scope field to multicast address [108].

C.2.5 Security Capabilities

IPv4 provides no security capabilities other than an optional security label field. Although end-to-end security can be provided at the application level, there is support for a standardized IP-level security service which any application can use without providing security features in that application. IPv6 provides a range of features that support authentication and privacy [108] [106].

C.3 QoS in IPv6

The QoS issue was introduced with the aim of supporting real-time services (e.g., IP voice, video, etc) and providing a network client with a range of service-offering QoS such as Integrated Services and Differentiated Service mechanisms require that different types of traffic flows to be treated differently by intervening routes in Internet. To achieve this goal, IPv6 has an elusive definition of a "flow" as described in the previous section. With this "flow label", the sender requests special handling, such as non default QoS or real-time services. In addition to the flow label, IPv6 has an 8-bit traffic class field. This field is used to identify and distinguish different classes of IPv6 packets with different priorities, providing various forms of differentiated services for IP packets [106]. Next the advantages of using both fields to support QoS will be discussed.

C.3.1 Flow Label

According to the IPv6 specification, the flow label might be used by the source to label packets that require special handling by intervening IPv6 routers, such as non

default QoS or real-time service. In order to classify packets belonging to the same flow, they are labeled with the same pre-defined flow label value. As a result, network elements are now capable of classifying packets based on IP semantics alone. This allows efficient mapping of packets to their flows and hence to their flow specification policy (for example, QoS requirements or class of service). Flow labels are assigned to flows by the sources, or sending nodes, in an unique manner. A source can never have more than one flow with the same flow label at a given time. New flow labels must be chosen randomly [107].

The Benefits of the Flow Label [108] [8] [114]:-

The flow label properties are ideal for proper and efficient packet classification. Three significant characteristics are introduced. First, the flow label **identifies** packets requiring special treatment. A flow label of zero indicates that a particular packet does not belong to this flow. This allows routers to immediately identify (a simple check within the IPv6 header is sufficient) whether a packet needs a special handling or not. Second, the flow label in conjunction with the source IP address serves as **unique** identifier for flows since each source node must ensure unique local flow labels. The great benefit for packet classification is that all information needed to uniquely classify packets is available within the IPv6 header. Third, the flow label is chosen **randomly**. The advantage to this attribute is that any set of bits within the flow label is suitable for use as lookup-key in routers.

Another benefit that can be achieved from using the flow label is the avoidance of extracting information from upper layers to forward or process packets. IPv4 intervening routers rely on the transport protocol or application level information for forwarding flows' packets. This means, instead of using the network layer only to process packets, routers require information from either transport or application protocol (ie socket port) to map packets to their reserved resources. This introduces what is know as the **Layer Violation Problem** [8]. This problem has some serious drawbacks with respect to the performance of packet classification such as:-

1. Accessing higher layer protocol information to distinguish different flows of the

same host pair is an expensive operation (especially in IPv6 networks).

2. Intermediate nodes decrypt security packets to get the port numbers to process packets even though, they should be hidden and only the receiver can decrypt these packets. This makes forwarding and processing complicated and more time is added when packets are encrypted and decrypted.

To overcome this layer violation problem, flow labels and source addresses are used as a hash key for routers during processing and forwarding packets. Therefore, there is no need to get the port number since all packets going to the same destination have to have the same flow label and the same source address.
Finally, utilizing the flow label for packet classification has the following advantage:-

I Use of the flow labels decreases the average processing load of the network routers. This results in reducing the end-to-end delay of real time traffic flows for the following two reasons:-

   a) when the flow label is consistently used to indicate real time flows, routers need to perform packet classification only for packets with non-zero flow labels.

   b) less processing time for IPv6 packets even if extension headers are used since all packets from the same flow must have identical extension headers. As a result, routers along the path ,source node to destination node, have to process the headers only on a per flow basis rather than a per packet basis.

II Flow label usage facilitates end-to-end IP level security mechanisms within resource reservation since packet classification does not rely on higher level information (ports). Therefore, IP Security Protocol (IPSEC) mechanisms specifically encryption can not obscure important information at the domains' networks during decryption.

Table C.1: The Priority Values

Congestion-controlled Traffic	Non-Congestion-controlled Traffic
0 un characterized traffic	8 real time traffic with lowest priority
1 filler traffic (e.g., email)	.
2 unattended data transfer(e.g., email)	.
3 (reserved)	.
4 attended bulk transfer (e.g., FTP, HTTP)	.
5 (reserved)	.
6 interactive traffic (e.g., telnet, X)	.
7 Internet control traffic	Least willing to discard
(e.g.,routing protocols, SNMP)	(e.g., low-fidelity audio)

III The flow label has the potential to facilitate implementation of QoS based flow routing mechanisms in which flow labels (in conjunction with the IP source address) are used for a lookup procedure which reduces packet processing time (end-to-end delay is reduced as a result). Also a QoS management unit can use the flow label to process QoS requests and to keep track of reservations.

C.3.2 Priority Field (Traffic Class)

The Priority field in the IPv6 header enables a source to identify the desired priority of its packets, relative to other packets from the same source. Values 0-7 are used to specify the priority of traffic for which the source is providing congestion control. Transmission control procedure for congestion-controlled traffic includes congestion control loops identical or similar to Van Jacobsons slow start . Values 8-15 are used to specify the priority of traffic that does not back off in response to congestion, e.g., real-time traffic. Priority classes can be used to complement queuing polities such as fair queuing or class-based queuing [109] [108] [115]. Table 1. Shows the recommended values for congestion-controlled traffic and for real time traffic. **Congestion-controlled traffic** refers to traffic for which the source "backs off" in response to

congestion. An example is the TCP congestion control mechanism. The nature of this traffic is that it is acceptable for there to be a variable amount of delay in the delivery of packets. IPv6 defines the following categories of congestion-controlled traffic in order of decreasing priority:-

. **Internet control traffic** This is the most important traffic to deliver, especially in times of high congestion.

. **Interactive traffic** The second most important traffic type after Internet control traffic. An On-line-to-host connection is one of the interactive traffic examples where the user efficiency depends on the rapid response time during interactive sessions so end-to-end delay is minimized.

. **Attended bulk transfer** Some applications require transfer of large amount of data. Therefore, delay is not sensitive as in the case of interactive traffic, however, the completion of transfer is very important. Examples of this type are FTP and HTTP.

. **Unattended data transfer** This type of traffic is initiated by the user and not expected to be delivered instantly. The user will do other tasks before the transfer is completed. Example of this type is electronic mail.

. **Uncharacterized traffic** This is the lowest priority traffic (best effort).

Non-congestion-controlled traffic is traffic for which a constant data rate and a constant delivery delay (smooth data rate and delivery delay). Examples are real time video and audio. Eight levels of priority are allocated for this type of traffic, from lowest priority (most willing to discard) to the highest priority 15 (least willing to be discard).

Appendix D

ns-2 added code

In this appendix, the C++ , tcl codes which has been created to do the IPv6 simulation is shown.

D.1 the C++ Code implementation

D.1.1 QoS Manager Code

Listing D.1: fgeeManager.h

```
/*
 * FgeeManager IP QoS module.
 *
 * Authors: El-Bahul Fgee <fgeeee@dal.ca>,
 *          Jason Kenney <jdkenney@dal.ca>, 2003
 */
#ifndef fgeeManager_h
#define fgeeManager_h
#include "dsPolicy.h"
#include "dsred.h"
#include <string.h>
#include "edge.h"
#include "core.h"

class FgeeManager;
```

```
#include "../queue/fgee-marker.h"

#include <vector>
#include <map>
#include <algorithm>

#define MAX_REALLOC 40

#define SUCCESS 100
#define NO_ENDPOINT 101
#define NOT_ENOUGH_BAND 102

/*-----------------------------------------------------------------------
structure of reallocation table entry
-----------------------------------------------------------------------*/

struct reallocTableEntry
{
  int codePt;
  double initialCir;
  double firstChange;
  double secondChange;
};

struct nodepair
{
        int n1;
        int n2;
};

class FgeeManager : public TclObject
{

 private:
  edgeQueue* edgePolicyObj[10];
  coreQueue* corePolicyObj[10];

  int numEdgePolicyObj;
  int numCorePolicyObj;
  int codepointTableSize;
  int reallocTableSize;
  double TOTAL_BW_USED;
  double MAX_BW;
  double MAX_EF_BW, EF_BW_USED;
  double MAX_AF_BW, AF_BW_USED;
```

```
double MAX_BE_BW, BE_BW_USED;
int NumQ, cp, AF1, AF2;
double cir, cbs, ebs, pir, pbs;

Tcl_Interp *interp;
Policy* polObj;
int allocate;
double avgRate;
policyTableEntry* tableEntry;
reallocTableEntry reallocTable[MAX_REALLOC];

std::vector<int> nodeids;

std::map<int, double> flows;        // keep track of the flows!

std::map<int, int> ftop;                          // map flowid to priority
std::map<int, double> flowprice;                        // map flowid to priority

std::vector<int> classes;
std::vector<float> weights;
std::vector<int> beginnings;     // keep track of beginning priority
std::vector<int> ends;           // keep track of end priorities
std::vector<FgeeMarker*> markers; // keep track of the edge routers
std::vector<FgeeManager*> managers; // keep track of the edge routers
std::map<int, struct nodepair> npairs; // fid to nodepairs
std::map<int, double> priotocriticalpoint;
std::map<int, double> priotobaseprice;

Policy policy;

public:
FgeeManager();
int command(int argc, const char*const* argv);
inline void addNewPolicy(nsaddr_t SNode, nsaddr_t DNode, policerType policer, int cp,
                         double cir, double cbs, double ebs, double pir, double pbs);
inline void dynamicAllocation(nsaddr_t source, nsaddr_t dest);
// performing dynamic allocation on the policies that have been defined
inline void setMaxBw(int bw);
void addEdgePolicyObj(edgeQueue* edgePol);
void addCorePolicyObj(coreQueue* corePol);
void setNumQueues(int numQ);
void policyObjEntries(nsaddr_t SNode, nsaddr_t DNode, policerType policer,
                      int cp, double cir, double cbs, double ebs, double pir, double pbs);
void addNewPHB(policerType policer);
void addReallocTableEntry(int cp, double intialCir, double first, double second);
```

```
  void downgradeOne(policyTableEntry * tableEntry);
  void downgradeTwo(policyTableEntry * tableEntry);
  void upgradeOne(policyTableEntry * tableEntry);
  void upgradeTwo(policyTableEntry * tableEntry);
  void endFlow(nsaddr_t SNode, nsaddr_t DNode);

  int get_class(int prio);
  void classify_flow(int fid, int prio);
  int mark(Packet *p) { return policy.mark(p); }
  void downgrade_flow(int fid, int after);
  int add_range(int i, int j, double k);
  int request(int fid, double band, int s, int d);
  int release(int fid);
  int push_edge(const char *s, int n1, int n2);
  int push_mang(const char *s);
  int besteffort(int fid);
  int add_nodepair(int fid, int n1, int n2);
  int check_bw(int fid, double band, int d);
  int set_point(int prio, double fcrit);
  int set_price(int prio, double baseprice);
  double get_price(double total, double used, int p);
  double check_price(int fid);
};

#endif /* fgeeManager_h    */
```

Listing D.2: fgeeManager.cc

```
/*-----------------------------------------------------------------------
 * FgeeManager QoS management module
 *
 * Authors: El-Bahul Fgee <fgeeee@dal.ca>,
 *          Jason Kenney <jdkenney@dal.ca>, 2003
 *
 *
 */
-----------------------------------------------------------------------
#include <stdio.h>
#include "fgeeManager.h"
#include <string.h>


/*-----------------------------------------------------------------------
class FgeeManagerClass : public TclClass
Instantiates a FgeeManager in a new simulation.
```

```
static class FgeeManagerClass : public TclClass {
public:
        FgeeManagerClass() : TclClass("FgeeManager") {}
        TclObject* create(int, const char*const*) {
                return (new FgeeManager);
        }
} class_fgeemanager;


/*-----------------------------------------------------------------
Manager() constructor
-----------------------------------------------------------------*/

FgeeManager::FgeeManager()
{
  numEdgePolicyObj = 0;          // numPolicyObj defined to enter client requests through the BB
  numCorePolicyObj = 0;
  reallocTableSize = 0;
  codepointTableSize = 0;
  TOTAL_BW_USED = 0;             // calculated using CIR, so bw_used is defined in bytes per second
  AF_BW_USED = 0;                // Bandwidth reserved for assured forwarding.
  EF_BW_USED = 0;                // Bandwidth reserved for expedited forwarding.
  BE_BW_USED = 0;                // Bandwidth reserved for best effort.
  NumQ = 0;
  AF1 = 0;
  AF2 = 0;

  classes.push_back(0);
  weights.push_back(0.99);
  beginnings.push_back(0);
  ends.push_back(15);
}

/*-----------------------------------------------------------------
int FgeeManager::command(int argc, const char*const* argv)
Takes care of interface with Tcl
-----------------------------------------------------------------*/

int FgeeManager::command(int argc, const char*const* argv)
{
  policerType policer;
  nsaddr_t SNode, DNode;
```

```
if(strcmp(argv[1], "setMaxBw") == 0)
  {
    setMaxBw(atoi(argv[2]));
    return TCL_OK;
  }
else if(strcmp(argv[1], "setNumQueues") == 0)
  {
    setNumQueues(atoi(argv[2]));
    return TCL_OK;
  }
else if (strcmp(argv[1], "addPolicyEntry") == 0) {
              for(vector<FgeeMarker *>::iterator fmi = markers.begin();
              fmi != markers.end(); fmi++)
                      (*fmi)->command(argc, argv);
      return(TCL_OK);
  }
else if (strcmp(argv[1], "addPolicerEntry") == 0) {
              for(vector<FgeeMarker *>::iterator fmi = markers.begin();
              fmi != markers.end(); fmi++)
                      (*fmi)->command(argc, argv);
      return(TCL_OK);
  }


else if(strcmp(argv[1], "set-point") == 0) {
      if (argc != 4) {
              printf("FgeeManager (%s): not correct number of arguments for
              set-point!\n", name());
              return (TCLERROR);
      }
      return set_point(atoi(argv[2]), atof(argv[3]));
}

else if(strcmp(argv[1], "set-price") == 0) {
      if (argc != 4) {
              printf("FgeeManager (%s): not correct number of arguments
              for set-price!\n", name());
              return (TCLERROR);
      }
      return set_price(atoi(argv[2]), atof(argv[3]));
}

else if(strcmp(argv[1], "get-price") == 0) {
```

```
        if (argc != 3) {
                printf("FgeeManager (%s): not correct number of arguments
                for get-price!\n", name());
                return (TCL_ERROR);
        }
printf("PRICE: %d %.3lf %.3lf\n", atoi(argv[2]), Scheduler::instance().clock(),
        get_price(MAX_BW, TOTAL_BW_USED, atoi(argv[2])));
return (TCL_OK);
}

else if(strcmp(argv[1], "request") == 0) {

        if (argc != 6) {
                printf("FgeeManager (%s): not correct number of arguments
                for request!\n", name());
                return (TCL_ERROR);
        }
        return request(atoi(argv[2]), atof(argv[3])/8.0, atoi(argv[4]), atoi(argv[5]));
}
else if(strcmp(argv[1], "release") == 0) {

        if (argc != 3) {
                printf("FgeeManager (%s): not correct number of arguments
                for release!\n", name());
                return (TCL_ERROR);
        }
        return release(atoi(argv[2]));
}


else if(strcmp(argv[1], "besteffort") == 0) {
        return besteffort(atoi(argv[2]));
}

else if ((argc == 4) && (strcmp(argv[1], "add-range") == 0)) {
        return add_range(atoi(argv[2]), atoi(argv[3]), atof(argv[4]));
}

else if ((argc == 5) && strcmp(argv[1], "pushedge") == 0) {
        return push_edge(argv[2], atoi(argv[3]), atoi(argv[4]));
}
else if (strcmp(argv[1], "pushmang") == 0) {
        return push_mang(argv[2]);
}
else if (strcmp(argv[1], "addnodepair") == 0) {
```

```
        return add_nodepair(atoi(argv[2]), atoi(argv[3]), atoi(argv[4]));
    }
    printf("FgeeMarker (%s): Unknown function %s\n", name(), argv[1]);
    return (TCL_ERROR);
}

//--------------------------------------------------

int FgeeManager::add_nodepair(int fid, int n1, int n2) {
    struct nodepair ntmp;
    ntmp.n1 = n1;
    ntmp.n2 = n2;
    npairs[fid] = ntmp;
    return (TCL_OK);
}

//--------------------------------------------------

int FgeeManager::besteffort(int fid) {
        //int fid = atoi(argv[2]);
        ftop[fid] = 8;
        flows[fid] = 1;

        for(vector<FgeeMarker *>::iterator fmi = markers.begin();
            fmi != markers.end(); fmi++)
                (*fmi)->set_flow(fid, 0);

        return TCL_OK;
}

//--------------------------------------------------
int FgeeManager::push_edge(const char* s, int n1, int n2) {
        if(strlen(s) == 0) {
                printf("FgeeManager (%s): Wrong Number of arguments??", name());
                return (TCL_ERROR);
        }
        printf("FgeeManager (%s): Adding %s to the end of the stack\n", name(), s);
        FgeeMarker *fm = dynamic_cast<FgeeMarker*>(TclObject::lookup(s));
        if(fm != 0) {
                markers.push_back(fm);
                if(find(nodeids.begin(), nodeids.end(), n1) == nodeids.end())
                        nodeids.push_back(n1);
        }
        else {
                printf("FgeeManager (%s): %s is not an object of type
```

```
                    FgeeMarker* !\n", name(), s);
            printf("Exiting\n");
            return  (TCL_ERROR);
        }
        return (TCL_OK);
}

//-------------------------------------------------
int FgeeManager::push_mang(const char* s) {
        if(strlen(s) == 0) {
            printf("FgeeManager (%s): Wrong Number of arguments??", name());
            return (TCL_ERROR);
        }
        printf("FgeeManager (%s): Adding %s to the end of the MANAGER stack\n", name(), s);
        FgeeManager *fm = dynamic_cast<FgeeManager*>(TclObject::lookup(s));
        if(fm != 0)
            managers.push_back(fm);
        else {
            printf("FgeeManager (%s): %s is not an object of type FgeeManager* !\n",
                name(), s);
            printf("Exiting\n");
            return  (TCL_ERROR);
        }
        return (TCL_OK);
}

//-------------------------------------------------
int FgeeManager::add_range(int i, int j, double k)
{
        //int i = atoi(argv[2]);
        //int j = atoi(argv[3]);
        //float k = atof(argv[4]);
        if(i > j) {
            fprintf(stderr, "Whoa! I think you meant:\n");
            fprintf(stderr, "FgeeManager add-range %d %d instead of:\n",
                        j, i);
            fprintf(stderr, "FgeeManager add-range %d %d, I'll try to fix
                         it for you",
                        i, j);
            int tmp = i;
            i = j;
            j = tmp;
        }
```

```
if(k < 0.0) {
        fprintf(stderr, "Whoa! you can't have a negative weight,
                        I will reverse the sign");
        k = -k;
}
if(j > 15) {
        printf("FgeeManager (%s): %d is too large for the priority field!
                (4 bits) Ignoring!\n", name(), j);
        return (TCL_ERROR);
}
if(beginnings[0] == 0 && ends[0] == 15) {
        // Only best effort exists, delete it;
        beginnings[0] = i;
        ends[0] = j;
        classes[0] = 1; // entire range is best effort, no?
        weights[0] = .99; // entire range is best effort, no?
} else {
        for(unsigned int t = 0; t < ends.size(); t++) {
                if(i >= beginnings[t] && i <= ends[t]) {
                        printf("FgeeManager (%s): ERROR: I can't overlap
                                priorities!\n", name());
                        printf("FgeeManager (%s): %d is between beginnings[%d] = %d
                                and ends[%d] = %d]\n", name(), i, t, beginnings[t], t,
                                ends[t]);
                        abort();
                } else if(j >= beginnings[t] && j <= ends[t]) {
                        printf("FgeeManager (%s): ERROR: I can't overlap priorities!\n",
                               name());
                        printf("FgeeManager (%s): %d is between beginnings[%d] = %d and
                               ends[%d] = %d]\n", name(), j, t, beginnings[t], t, ends[t]);
                        abort();
                }
        }
        beginnings.push_back(i);
        ends.push_back(j);
        classes.push_back(classes.size());
        weights.push_back(k);
}
printf("FgeeManager (%s): Setup new traffic class %d -> %d! (class %d)\n",
                name(), i, j, classes[classes.size()-1]);
for(vector<FgeeMarker *>::iterator fmi = markers.begin();
   fmi != markers.end(); fmi++) {
        (*fmi)->replicate(i, j);
}
return (TCL_OK);
```

```
}

//--------------------------------------------------
double FgeeManager::check_price(int fid)
{
        if(flowprice.count(fid) == 0)
        {
                fprintf(stderr, "FgeeManager (%s): No prices for fid %d
                        found!\n", name(), fid);
                return -1000.0;
        }
        else
        return flowprice[fid];
}
//--------------------------------------------------
int FgeeManager::request(int fid, double band, int s, int d)
{
        //double band = atof(argv[3])/8.0;
        //int fid = atoi(argv[2]);

        if(npairs.find(fid) == npairs.end()) {
                printf("FgeeManager (%s): flow id: %d not found! Well, shit!\n",
                        name(), fid);
                return (TCL_ERROR);
        }

        int shere = 0;
        int dhere = 0;
        for(vector<int>::iterator i = nodeids.begin(); i != nodeids.end(); i++) {
                printf("%d == %d ? and %d == %d ?\n", *i, s, *i, d);
                if(s == *i)
                        shere = 1;
                if(d == *i)
                        dhere = 1;
        }

        if(shere == 0) {
                printf("FgeeManager (%s): flow id: %d nodeid: %d not originating here!\n",
                         name(), fid, s);
                return (TCL_ERROR);
        }

        if ((MAX_BW - TOTAL_BW_USED - band) < 0.0) {
                printf("FgeeManager (%s): flow id: %d requested too much bandwidth,
                        denying!\n", name(), fid);
```

```
                printf("Math: MAX_BW - TOTAL_BW_USED - band = ?\n");
                printf("Math: %f - %f - %f = %f\n", MAX_BW, TOTAL_BW_USED, band,
                        MAX_BW - TOTAL_BW_USED - band);
                return TCL_OK;
        }

        if(dhere == 0)
        {
                vector<FgeeManager *>::iterator fmani;
                for(fmani = managers.begin(); fmani != managers.end(); fmani++)
                {
                        if(!(*fmani)->check_bw(fid, band, d))
                                break;
                }
                if(fmani == managers.end()) {
                        printf("FgeeManager (%s): flow id: %d can't find endpoint %d
                                in any of my other managers\n", name(), fid, d);
                        return (TCL_ERROR);
                }
        }

        TOTAL_BW_USED += band;

        printf("FgeeManager (%s): flow id: %d requested bandwidth: %f , accepting!\n",
                 name(), fid, band);
        printf("FgeeManager (%s): used  bandwidth: %f, bandwidth remaining: %f!\n",
                name(), TOTAL_BW_USED, MAX_BW - TOTAL_BW_USED);

        flows[fid] = band;
        ftop[fid] = -1;

        for(vector<FgeeMarker *>::iterator fmi = markers.begin();
            fmi != markers.end(); fmi++)
                (*fmi)->set_flow(fid, band);

        return TCL_OK;
}
//--------------------------------------------------
int FgeeManager::release(int fid)
{

        if(flows.find(fid) != flows.end())
                TOTAL_BW_USED -= flows[fid];

        flows.erase(fid);
```

```
        ftop.erase(fid);

        for(vector<FgeeMarker *>::iterator fmi = markers.begin();
           fmi != markers.end(); fmi++)
                (*fmi)->unset_flow(fid);

        return (TCL_OK);
}


//-----------------------------------------------
int FgeeManager::check_bw(int fid, double band, int d) {

        if(npairs.find(fid) == npairs.end()) {
                printf("FgeeManager (%s): flow id: %d not found! Well !\n", name(), fid);
                return 1;
        }

        int dhere = 0;
        for(vector<int>::iterator i = nodeids.begin(); i != nodeids.end(); i++) {
                if(d == *i)
                        dhere = 1;
        }
        if (dhere == 0) {
                printf("FgeeManager (%s): flow id: %d Can't find endpoint %d\n", name(), fid, d);
                return 1;
        }

        if ((MAX_BW - TOTAL_BW_USED - band) < 0.0) {
                printf("FgeeManager (%s): flow id: %d requested too much bandwidth,
                        denying!\n", name(), fid);
                printf("Math: MAX_BW - TOTAL_BW_USED - band = ?\n");
                printf("Math: %f - %f - %f = %f\n", MAX_BW, TOTAL_BW_USED, band,
                        MAX_BW - TOTAL_BW_USED - band);
                return 1;
        }

        TOTAL_BW_USED += band;

        flows[fid] = band;
        ftop[fid] = -1;

        for(vector<FgeeMarker *>::iterator fmi = markers.begin();
           fmi != markers.end(); fmi++)
                (*fmi)->set_flow(fid, band);
```

```
        return 0;
}

/*-------------------------------------------------------------------*/


void FgeeManager::addNewPolicy(nsaddr_t SNode, nsaddr_t DNode, policerType policer,
                               int cp, double cir, double cbs, double ebs, double pir, double pbs){

  if (MAX_BW > TOTAL_BW_USED)
    {
      Tcl& tcl = Tcl::instance();

      switch(policer)
        {
        case EF:
          if (pir > (MAX_EF_BW - EF_BW_USED))
            {
              printf("ERROR: NO BANDWIDTH TO ALLOCATE\n");
              Tcl_SetVar(tcl.interp(), "allocate", "0",TCL_GLOBAL_ONLY );
            } else
              {
                Tcl_SetVar(tcl.interp(), "allocate", "1", TCL_GLOBAL_ONLY);
                EF_BW_USED += pir;
              }
          break;
        default:
          if (cir > (MAX_AF_BW - AF_BW_USED))
              {
                printf("ERROR: NO BANDWIDTH TO ALLOCATE\n");
                Tcl_SetVar(tcl.interp(), "allocate", "0",TCL_GLOBAL_ONLY );
              } else
                {
                  Tcl_SetVar(tcl.interp(), "allocate", "1 ", TCL_GLOBAL_ONLY);
                  AF_BW_USED += cir;
                }
          break;
        }
      TOTAL_BW_USED = EF_BW_USED + AF_BW_USED + BE_BW_USED;
    } else
      {
        printf("NO MORE ALLOCATIONS POSSIBLE \n");
      }
  return;
```

```
}

/*--------------------------------------------------------------------------*/

void FgeeManager::addNewPHB(policerType policer)
{
  Tcl& tcl = Tcl::instance();

  if (NumQ == 4)
    {
      if (AF1 < AF2)
        {
          switch(policer)
            {
            case EF:
              Tcl_SetVar(tcl.interp(), "setPHB", "0", TCL_GLOBAL_ONLY);
              break;
            default:
              Tcl_SetVar(tcl.interp(), "setPHB", "1", TCL_GLOBAL_ONLY);
              AF1++;
              break;
            }
        }
      else
        {
          switch(policer)
            {
            case EF:
              Tcl_SetVar(tcl.interp(), "setPHB", "0", TCL_GLOBAL_ONLY);
              break;
            default:
              Tcl_SetVar(tcl.interp(), "setPHB", "2", TCL_GLOBAL_ONLY);
              AF2++;
              break;
            }
        }
      return;
    }
  else
    {
      return;
    }
}
```

```
/*-------------------------------------------------------------------------

void FgeeManager::addEdgePolicyObj(edgeQueue* edgePol)
{
  if (numEdgePolicyObj > 9)
    {
      printf("ERROR: No more policy objects can be added\n");
    } else
      {
        edgePolicyObj[numEdgePolicyObj] =edgePol;
        numEdgePolicyObj++;
      }
  return;
}

/*-------------------------------------------------------------------------

void FgeeManager::addCorePolicyObj(coreQueue* corePol)
{
  if (numCorePolicyObj > 9)
    {
      printf("ERROR: No more policy objects can be added\n");
    } else
      {
        corePolicyObj[numCorePolicyObj] =corePol;
        numCorePolicyObj++;
      }
  //  printf("Successfully added policy Objects\n");
  return;
}


#if 0
/*-------------------------------------------------------------------------

void FgeeManager::addPolicyObj(edgeQueue* pol)
{
  if (numPolicyObj > 9)
    {
      printf("ERROR: No more policy objects can be added\n");
    } else
      {
        policyObj[numPolicyObj] =pol;
        numPolicyObj++;
```

```
    }
  return;
}


/*-------------------------------------------------------------------------
a method to add policys to all the policy objects

-------------------------------------------------------------------------*/

void FgeeManager::policyObjEntries(nsaddr_t SNode, nsaddr_t DNode, policerType policer,
                                   int cp, double cir, double cbs, double ebs, double pir, double p
{
  int first, second, third, fourth;

  for(int i=0; i<numPolicyObj; i++)
    {
      policyObj[i]->addNewPolicyEntry(SNode, DNode, policer, cp, cir, cbs, ebs, pir, pbs);
      printf("The policies have been added\n");
      policyObj[i]->addNewPolicer(policer, first, second, third, fourth);
      printf("The Policer has been added\n");

    }

  return;
}

#endif

/*-------------------------------------------------------------------------
setMaxBw() sets the maximum bandwidth available to allocate.
need to check on the sense behind converting it between bits and bytes???
-------------------------------------------------------------------------*/

void FgeeManager::setMaxBw(int bw)
{
  MAX_BW = (double) bw/8.0;
  MAX_EF_BW = MAX_BW/4.0;
  MAX_AF_BW = MAX_BW*(2.0/4.0);
  MAX_BE_BW = MAX_BW - (MAX_EF_BW + MAX_AF_BW);
  printf("MAX_BW     is %f\nMAX_EF_BW is %f\nMAX_AF_BW is %f\nMAX_BE_BW is %f\n",
         MAX_BW*8.0, MAX_EF_BW*8.0, MAX_AF_BW*8.0, MAX_BE_BW*8.0);

                for(vector<FgeeMarker*>::iterator fmi = markers.begin();
```

```
                fmi != markers.end(); fmi++)
            (*fmi)->set_maxbw(MAXBW);

  return;
}

/*-----------------------------------------------------------------------
setNumQueues(NumQ)
sets the Number of queues in the diffserv domain.
This lets the BB decide based on the number of queues tp which queue the next
codepoint should be assigned.
-----------------------------------------------------------------------*/

void FgeeManager::setNumQueues(int nQ)
{
  NumQ = nQ;
  return;
}

/*-----------------------------------------------------------------------
  void FgeeManager::dynamicAllocation()

method provides for dynamic allocation. checks the meter, depending on the type of policer and the
code points, decides the new codepoint to be used.

-----------------------------------------------------------------------*/

void FgeeManager::dynamicAllocation(nsaddr_t source, nsaddr_t dest)
{
  int i;

  for(i = 0; i < numEdgePolicyObj; i++) {
    tableEntry = edgePolicyObj[i]->getPolicyTableEntry(source, dest);
    if(tableEntry != NULL) break;
  }

  if(i == numEdgePolicyObj) {
    printf("ERROR!!! No whatever found from source %d to destination %d\n", source, dest);
    return;
  }

  switch(tableEntry->policer)
    {
    case EF:
```

```
if (tableEntry->avgRate <= ((tableEntry->pir)/2) + 1000.0)
  {
    if (tableEntry->cir <= ((tableEntry->pir)*2/3 + 500.0))
      {
      }
    else if (tableEntry->cir <= ((tableEntry->pir)*3/4 + 500.0))
      {
        upgradeTwo(tableEntry);
        EF_BW_USED = EF_BW_USED - (tableEntry->pir)/12;
      }
    else
      {
        upgradeTwo(tableEntry);
        EF_BW_USED = EF_BW_USED - (tableEntry->pir)/3;
      }
  }
else if (tableEntry->avgRate <= ((tableEntry->pir)*2/3) + 1000.0)
  {
    if (tableEntry->cir <= ((tableEntry->pir)*2/3 + 500.0))
      {
        if (MAX_EF_BW >= EF_BW_USED + (tableEntry->pir)/12) {
          downgradeOne(tableEntry);
          EF_BW_USED = EF_BW_USED + (tableEntry->pir)/12;
        }
        else {
        }
      }
    else if (tableEntry->cir <= ((tableEntry->pir)*3/4 + 500.0))
      {
      }
    else
      {
        upgradeOne(tableEntry);
        EF_BW_USED = EF_BW_USED - (tableEntry->pir)/4;
      }
  }
else
  {
    if (tableEntry->cir <= ((tableEntry->pir)*2/3 + 500.0))
      {
        downgradeTwo(tableEntry);
        EF_BW_USED = EF_BW_USED + (tableEntry->pir)/3;
      }
    else if (tableEntry->cir <= ((tableEntry->pir)*3/4 + 500.0))
      {
```

```
            if (MAX_EF_BW >= EF_BW_USED + (tableEntry->pir)/4) {
              downgradeTwo(tableEntry);
              EF_BW_USED = EF_BW_USED + (tableEntry->pir)/4;
            }
            else {
            }
          }
        else
          {
          }
      } break;
    default:
      if (tableEntry->avgRate < tableEntry->cir)
        {
          AF_BW_USED = AF_BW_USED - (tableEntry->cir - tableEntry->avgRate)/2.0;
          tableEntry->cir = tableEntry->cir - (tableEntry->cir - tableEntry->avgRate)/2.0;
        }
      else if (tableEntry->avgRate < tableEntry->pir)
        {
          if (MAX_AF_BW >= AF_BW_USED + (tableEntry->avgRate - tableEntry->cir))
            {
              AF_BW_USED = AF_BW_USED + (tableEntry->avgRate - tableEntry->cir);
              tableEntry->cir = tableEntry->avgRate;
            }
          else
            {
            }
        }
      else
        {
        }
      break;
    }
  TOTAL_BW_USED = EF_BW_USED + AF_BW_USED + BE_BW_USED;
  return;
}

/*-----------------------------------------------------------------------

void FgeeManager::addReallocTableEntry(int cp, double cir)

This method changes the policy marked in a a pkt during runtime
to regain bw.

-----------------------------------------------------------------------
```

```
void FgeeManager::addReallocTableEntry(int cp, double initialCir, double first, double second)
{
  if (reallocTableSize == MAX_REALLOC)
    {
      printf("No more policies accepted\n");
    }
  else
    {
      reallocTable[reallocTableSize].codePt = cp;
      reallocTable[reallocTableSize].initialCir = initialCir;
      reallocTable[reallocTableSize].firstChange = first;
      reallocTable[reallocTableSize].secondChange = second;
      reallocTableSize++;
    }
  return;
}

/*-------------------------------------------------------------------------

void FgeeManager::upgradeOne()

-------------------------------------------------------------------------*/

void FgeeManager::upgradeOne(policyTableEntry * tableEntry)
{
  for (int i=0; i < reallocTableSize; i++)
    {
      if (reallocTable[i].codePt == tableEntry->codePt)
        tableEntry->cir = reallocTable[i].firstChange;
    }
  return;
}

/*-------------------------------------------------------------------------

void FgeeManager::upgradeTwo()

-------------------------------------------------------------------------*/

void FgeeManager::upgradeTwo(policyTableEntry * tableEntry)
{
  for (int i=0; i < reallocTableSize; i++)
    {
      if (reallocTable[i].codePt == tableEntry->codePt)
```

```
        tableEntry->cir = reallocTable[i].secondChange;
    }
  return;
}

/*--------------------------------------------------------------------

void FgeeManager::downgradeTwo()

--------------------------------------------------------------------*/

void FgeeManager::downgradeTwo(policyTableEntry* tableEntry)
{
  for (int i=0; i < reallocTableSize; i++)
    {
      if (reallocTable[i].codePt == tableEntry->codePt)
        tableEntry->cir = reallocTable[i].initialCir;
    }
  return;
}

/*--------------------------------------------------------------------

void FgeeManager::downgradeOne()

--------------------------------------------------------------------*/

void FgeeManager::downgradeOne(policyTableEntry* tableEntry)
{
  for (int i=0; i < reallocTableSize; i++)
    {
      if (reallocTable[i].codePt == tableEntry->codePt)
        tableEntry->cir = reallocTable[i].firstChange;
    }
  return;
}

/*--------------------------------------------------------------------

void FgeeManager::endFlow()

--------------------------------------------------------------------*/

void FgeeManager::endFlow(nsaddr_t SNode, nsaddr_t DNode)
{
```

```
  double bw;
  policerType policer;

  for(int i = 0; i < numEdgePolicyObj; i++) {
    tableEntry = edgePolicyObj[i]->getPolicyTableEntry(SNode, DNode);
    bw = tableEntry->cir;

    policer = tableEntry->policer;
    edgePolicyObj[i]->removePolicyTableEntry(SNode, DNode);

    switch(policer)
      {
      case EF:
        EF_BW_USED = EF_BW_USED - bw;
      default:
        AF_BW_USED = AF_BW_USED - bw;
      }

    TOTAL_BW_USED = EF_BW_USED + AF_BW_USED + BE_BW_USED;

    if(tableEntry != NULL) break;
  }
  return;
}

int FgeeManager::get_class(int prio)
{
        if(prio > 15)
        {
                printf("FgeeMarker (%s): ERROR: Invalid priority field: %d !\n", name(), prio);
                abort();
        }

        for(unsigned int i = 0; i < beginnings.size(); i++)
        {
                if(prio >= beginnings[i] && prio <= ends[i])
                        return classes[i];
        }
        return 0;       // Best effort if it doesn't fit a specified class
}

void FgeeManager::classify_flow(int fid, int prio)
{
        if(flows.count(fid) == 0) {
                int c = get_class(prio);
```

```
                //flows[fid] = c;
                for(vector<FgeeMarker*>::iterator fmi = markers.begin();
                    fmi != markers.end(); fmi++)
                        //(*fmi)->set_flow(fid, c);
                printf("FgeeManager (%s): Added flow id %d to class %d\n",
                        name(), fid, c);
        }

}

void FgeeManager::downgrade_flow(int fid, int after) {
                for(vector<FgeeMarker*>::iterator fmi = markers.begin();
                   fmi != markers.end(); fmi++)
                (*fmi)->downgrade_flow(fid, after);
}

int FgeeManager::set_price(int prio, double baseprice)
{
        priotobaseprice[prio] = baseprice;
        return (TCL_OK);
}

int FgeeManager::set_point(int prio, double fcrit)
{
        if(fcrit >= 0.0 && fcrit <= 1.0)
                priotocriticalpoint[prio] = fcrit;
        else {
                fprintf(stderr, "FgeeManager(%s): CRITICAL POINT (%lf) MUST BETWEEN
                        0.0 and 1.0\n", name(), fcrit);
                return (TCL_ERROR);
        }
        return (TCL_OK);
}

double FgeeManager::get_price(double total, double used, int p)
{
        double fcrit = 0.0;
        double baseprice = 0.0;
        if(priotocriticalpoint.find(p) != priotocriticalpoint.end())
                fcrit = priotocriticalpoint[p];
        else
                fprintf(stderr, "FgeeManager(%s): UNABLE TO FIND PRICING POLICY
                        FOR PRIORITY: %d, setting critical point to 0.0!", name(), p);

        if(priotobaseprice.find(p) != priotobaseprice.end())
```

```
                baseprice = priotobaseprice[p];
        else
                fprintf(stderr, "FgeeManager(%s): UNABLE TO FIND PRICING POLICY
                        FOR PRIORITY: %d, setting baseprice to 0.0!", name(), p);

        double load = used/total;
        double price;
        if(load <= fcrit)
                price =  baseprice;
        else
                price = (baseprice*(1.0-fcrit)/(1.0-load));

        if(isinf(price))
                price = (baseprice*(1.0-fcrit)/(0.0001));

        return price;
}

/*-----------------------------------------------------------------------
int FgeeManager::canYouAddPolicy(

work on stuff like domain numbers and stuff.
-----------------------------------------------------------------------*/

#if 0

  switch(tableEntry->policer)
    {
    case EF:
      if (tableEntry->avgRate > tableEntry->pir) {
        printf("avgRate greater than EF Peak\n");
        return;
      }
      else if (tableEntry->avgRate < tableEntry->cir)
        {
          printf("avgRate lesser than EF CIR\n");
          //      printf("EF_USED is  %f\n", EF_BW_USED);
          EF_BW_USED = EF_BW_USED - (tableEntry->cir - tableEntry->avgRate);
          //(tableEntry->cir + (tableEntry->cir - tableEntry->avgRate)/2.0);
          //      printf("EF_USED is now %f\n", EF_BW_USED);
          //      printf("TOTAL_BW_USED is %f\n", TOTAL_BW_USED);
          TOTAL_BW_USED = EF_BW_USED + AF_BW_USED + BE_BW_USED;
          //      printf("TOTAL_BW_USED is now %f\n", TOTAL_BW_USED);
          //  tableEntry->cir = tableEntry->cir - (tableEntry->cir - tableEntry->avgRate)/2.0;
          tableEntry->cir=tableEntry->avgRate;
```

```
    }
  else if (tableEntry->avgRate < tableEntry->cir)
    {
      if ( MAX_EF_BW > EF_BW_USED + (tableEntry->avgRate - tableEntry->cir)/1.0)
        {
          printf("avgRate greater than EF CIR\n");
          //              printf("EF_USED is  %f\n", EF_BW_USED);
          EF_BW_USED = EF_BW_USED + (tableEntry->avgRate - tableEntry->cir)/1.0;
          //              printf("EF_USED is now %f\n", EF_BW_USED);
          //              printf("TOTAL_BW_USED is %f\n", TOTAL_BW_USED);
          TOTAL_BW_USED = EF_BW_USED + AF_BW_USED + BE_BW_USED;
          //              printf("TOTAL_BW_USED is now %f\n", TOTAL_BW_USED);
          tableEntry->cir = tableEntry->cir + (tableEntry->avgRate - tableEntry->cir)/1.0;
        } else return;
    }
  else {
    printf("avgRate is same as cir\n");
      return;
  }
  break;
case TSW3CM:
  printf("I am in AF\n");
  if (tableEntry->avgRate < tableEntry->cir)
    {
      printf("avgRate less than AF CIR\n");
      //      printf("AF_USED is  %f\n", AF_BW_USED);
      AF_BW_USED = AF_BW_USED - (tableEntry->cir - tableEntry->avgRate);
      //(tableEntry->cir + (tableEntry->cir - tableEntry->avgRate)/2.0);
      //      printf("AF_USED is now %f\n", AF_BW_USED);
      //      printf("TOTAL_BW_USED is %f\n", TOTAL_BW_USED);
      TOTAL_BW_USED = EF_BW_USED + AF_BW_USED + BE_BW_USED;
      //      printf("TOTAL_BW_USED is now %f\n", TOTAL_BW_USED);
      // tableEntry->cir = tableEntry->cir - (tableEntry->cir - tableEntry->avgRate)/2.0;
      tableEntry->cir=tableEntry->avgRate;
    }
  else if (tableEntry->avgRate > tableEntry->pir)
    {
      printf("avgRate greater than AF Peak\n");
      return;
    }
  else if (tableEntry->avgRate > tableEntry->cir)
    {
      printf("avgRate greater than AF CIR \n");
      if (MAX_AF_BW > AF_BW_USED + (tableEntry->avgRate - tableEntry->cir)/1.0)
        {
```

```
            AF_BW_USED = AF_BW_USED + (tableEntry->avgRate - tableEntry->cir)/1.0;
            TOTAL_BW_USED = EF_BW_USED + AF_BW_USED + BE_BW_USED;
            tableEntry->cir = tableEntry->cir + (tableEntry->avgRate - tableEntry->cir)/1.0;
          } else {
            printf("I am somewhere here \n");
            return;
          }
        }
      else {
        printf("avgRate is %f, same as AF CIR \n", tableEntry->avgRate);

        return;
      }
      break;
    default:
      printf("default case. value of policer = %d\n", tableEntry->policer);
      break;
    }
#endif
```

D.1.2 IP Edge Router Code

Listing D.3: fgee-marker.cc

```
/*
 * FgeeMarker module for IP Edge routers.
 *
 * Authors: El-Bahul Fgee <fgeeee@dal.ca>,
 *          Jason Kenney <jdkenney@dal.ca>, 2003
 */

#include <string.h>
#include <queue.h>
#include "random.h"
#include "fgee-marker.h"


static class FgeeMarkerClass : public TclClass {
 public:
        FgeeMarkerClass() : TclClass("Queue/FgeeMarker") {}
        TclObject* create(int, const char*const*) {
                return (new FgeeMarker);
        }
} class_fgee_marker;
```

```
FgeeMarker::FgeeMarker() {
        q_ = new WFQ;
        fqueues = 0;
        setqweights();
        stats.drops = 0;
        stats.degrades = 0;
        stats.edrops = 0;
        stats.pkts = 0;
   // Some initial values for the random marking fractions
        marker_frc_[0]=0.0; // class-0 is not used
        marker_frc_[1]=0.4;
        marker_frc_[2]=0.7;
        marker_frc_[3]=0.9;
        marker_frc_[4]=1.0;

        // initially set up one class: best-effort
        int i = 0;
        classes.push_back(0);
        beginnings.push_back(i);
        i = 15;
        ends.push_back(i);
        fmanager = 0;
        maxbw = 0.0;
    queueprice = 0.0;
}

int FgeeMarker::get_class(int prio)
{
        if(prio > 15)
        {
                printf("FgeeMarker (%s): ERROR: Invalid priority field: %d !\n",
                        name(), prio);
                abort();
        }

        for(unsigned int i = 0; i < beginnings.size(); i++)
        {
                if(prio >= beginnings[i] && prio <= ends[i])
                        return classes[i];
        }
        return 0;       // Best effort if it doesn't fit a specified class
}

int FgeeMarker::command(int argc, const char*const* argv) {
```

```
    if (strcmp(argv[1], "report-length") == 0) {
            double t = atof(argv[2]);
            fprintf(stderr, "%f %d\n", t, q_->length());
            return (TCL_OK);
    }
    if(strcmp(argv[1], "setqueueprice") == 0) {
            queueprice = atof(argv[2]);
            return (TCL_OK);
    }
    if (strcmp(argv[1], "printStats") == 0) {
    printStats();
    return (TCL_OK);
}
    if (strcmp(argv[1], "addPolicyEntry") == 0) {
            policy.addPolicyEntry(argc, argv);
            return(TCL_OK);
    }
    if (strcmp(argv[1], "addPolicerEntry") == 0) {
            policy.addPolicerEntry(argc, argv);
            return(TCL_OK);
    }
    if (argc == 3) {
            if (!strcmp(argv[1], "marker_type")) {
                    marker_type_ = atoi(argv[2]);
                    if ((marker_type_ != DETERM) && (marker_type_ != STATIS)) {
                            printf("Wrong Marker Type\n");
                            abort();
                    }
                    return (TCL_OK);
            }
            if (!strcmp(argv[1], "marker_class")) {
                    marker_class_ = atoi(argv[2]);
                    if (marker_class_ <1 || marker_class_>NO_CLASSES) {
                            printf("Wrong Marker Class:%d\n", marker_class_);
                            abort();
                    }
                    return (TCL_OK);
            }
            if (!strcmp(argv[1], "init-seed")) {
                    rn_seed_ = atoi(argv[2]);
                    Random::seed(rn_seed_);
                    srand48((long)(rn_seed_));
                    return (TCL_OK);
            }
            if (!strcmp(argv[1], "attach-manager")) {
```

```
                        fmanager = dynamic_cast<FgeeManager*>(TclObject::lookup(argv[2]));
                        if(fmanager == 0) {
                                printf("FgeeMarker (%s): %s is not an object of type
                                        FgeeManager* n", name(), argv[2]);
                                printf("Called from: %s %s %s\n", argv[0], argv[1], argv[2]);
                                return (TCL_ERROR);
                        }
                        printf("FgeeMarker (%s): Attached FgeeManager %s\n", name(), argv[2]);
                        return (TCL_OK);
                }
        }
        return Queue::command(argc, argv);
}

void FgeeMarker::statenque(int fid) {
        if(stats.pkts_CP.size() == 0)
                stats.pkts_CP[fid] = 0;

        if(stats.drops_CP.count(fid) == 0) {
                stats.drops_CP[fid] = 0;
        }

        stats.pkts++;
        stats.pkts_CP[fid]++;
}


void FgeeMarker::statdrop(int fid) {
        if(stats.drops_CP.size() == 0)
                stats.drops_CP[fid] = 0;

        stats.drops++;
        stats.drops_CP[fid]++;

}

void FgeeMarker::statdegrade(int fid) {
        if(stats.degrades_CP.size() == 0)
                stats.degrades_CP[fid] = 0;

        stats.degrades++;
        stats.degrades_CP[fid]++;
}
```

```
void FgeeMarker::enque(Packet* p) {
        hdr_tcp*  tcph = hdr_tcp::access(p);
        hdr_ip*   iph = hdr_ip::access(p);
        hdr_cmn* cm_h = hdr_cmn::access(p);

        statenque(iph->fid_);

        if(q_->qlen(iph->fid_) == q_->limit()) {
                fprintf(stderr, "PKT: %f %d DROP FID:%d PRI:%d\n",
                        Scheduler::instance().clock(), tcph->seqno() ,iph->fid_, iph->prio_);
                statdrop(iph->fid_);
                drop(p);
                return ;
        }

        iph->fgeeprice() += queueprice;
        int cp;
        double  cur_time  = Scheduler::instance().clock();
        cm_h->ts_arr_ = cur_time;

  if(fmanager == 0) {
        printf("FgeeMarker (%s): Next time try attaching a Manager first\n", name());
        abort();
  }

        if(ftop[iph->fid_] == -1) {
                ftop[iph->fid_] = iph->prio_;
                setqweights();
        }

        int before = iph->prio_;

        if(fmanager != NULL) {
                cp = policy.mark(p);
                //int after = (hdr_ip::access(p))->prio_;
                int after = iph->prio_;


                if(after < 8) {
                        fmanager->downgrade_flow(iph->fid_, 8);
                        fprintf(stdout, "PKT: %f %d DROP FID:%d PRI:%d\n",
                                Scheduler::instance().clock(), tcph->seqno() ,iph->fid_,
                                 iph->prio_);
                        statdrop(iph->fid_);
                        drop(p);
```

```
                        fprintf(stderr, "Dropping packet because it's best-effort now:
                                flowid: %d\n", iph->fid_);
                        return;
                }

                if(before == 8)
                        iph->prio_ = after = before; // don't let best effort creep up
                //fprintf(stderr, "(%s) before:%d cp:%d bling:%d\n", name(), before, after, ftop[iph
                if(after < before) {
                        statdegrade(iph->fid_);
                }

                if(before != after && ftop[iph->fid_] != after) {
                        //printf("ftop[%d] == %d != %d\n", iph->fid_, before, after);
                        if(ftop[iph->fid_] != -1)
                                fprintf(stderr, "FGEE: %f %d MOD FID:%d PRI:%d\n",
                                        Scheduler::instance().clock(), tcph->seqno() ,
                                        iph->fid_, iph->prio_);
                        fmanager->downgrade_flow(iph->fid_, after);
                }
        }

        q_->enque(p);

}

void FgeeMarker::downgrade_flow(int fid, int after) {
        //if(ftop.count(fid) > 0) {
                if(after > ftop[fid])
                        fprintf(stderr, "(%s) %f Moving priority UP (%d -> %d)\n", name(),
                                Scheduler::instance().clock(),ftop[fid], after);
                else if(after < ftop[fid]) {
                        fprintf(stderr, "(%s) %f Moving priority DOWN (%d -> %d)\n",
                                name(), Scheduler::instance().clock(),ftop[fid], after);
                }
                ftop[fid] = after;
                setqweights();
        //}
}

// Nothing interesting here
Packet* FgeeMarker::deque() {
        Packet *p;
        p = q_->deque();
        return p;
```

```
}

void FgeeMarker::replicate(int newb, int newe)
{
        if(fmanager == 0)
                printf("FgeeMarker (%s): Replicating beginnings and ends from FgeeManager\n", name(
        else
                printf("FgeeMarker (%s): Replicating beginnings and ends from FgeeManager (%s)\n",
        if(beginnings[0] == 0 && ends[0] == 15) {
                beginnings[0] = newb;
                ends[0] = newe;
                classes[0] = 1;
        } else {
                beginnings.push_back(newb);
                ends.push_back(newe);
                classes.push_back((int) classes.size() + 1);
        }
}

void FgeeMarker::set_flow(int fid, double b)
{
        flows[fid] = b;
        if(b != 0)
                ftop[fid] = -1;
        else
                ftop[fid] = 8;

        q_->set_flow(fid, ++fqueues);

        for(std::map<int, double>::iterator mi = flows.begin(); mi != flows.end(); mi++) {
                //printf("fid: %d has weight: %f\n", (*mi).first, (*mi).second/maxbw);
                //q_->setweight((*mi).first, (*mi).second/maxbw);
        }
}

void FgeeMarker::unset_flow(int fid)
{
        flows.erase(fid);
        ftop.erase(fid);

        q_->unset_flow(fid);
}


void FgeeMarker::printStats() {
```

```
    printf("\nPackets Statistics\n");
    printf("=============================================\n");
    printf(" CP  TotPkts   TxPkts   ldrops   degrades\n");
    printf(" --  -------  ------  ------  --------\n");
    printf(" All %8ld %8ld %8ld %8ld\n",stats.pkts,stats.pkts - stats.drops,
          stats.drops,stats.degrades);

        for (map<int, long>::iterator i = stats.drops_CP.begin(); i != stats.drops_CP.end(); i++) {
        printf("%3d %8ld %8ld %8ld %8ld\n",i->first,stats.pkts_CP[i->first],
              stats.pkts_CP[i->first] - stats.drops_CP[i->first],stats.drops_CP[i->first],
               stats.degrades_CP[i->first]);
        }
}

void FgeeMarker::setqweights() {

        if(ftop.size() == 0)
                return;

        double total = 0.0;

        for(map<int, int>::iterator i = ftop.begin(); i != ftop.end(); i++) {
                if(i->second < 8 || i->second > 15) {
                        //printf("skipping ftop[%d] -> %d\n", i->first, i->second);
                        continue;
                }
                //printf("(%s)      : %d\n", name(), i->second - 7);
                total += (double) (i->second - 7);
        }

        //printf("(%s) Total: %f\n", name(), total);

        double blam = 0.0;

        for(map<int, int>::iterator i = ftop.begin(); i != ftop.end(); i++) {
                if(i->second < 8 || i->second > 15)
                        continue;
                q_->setweight(i->first, ((double) i->second - 7)/total);
                blam += ((double) i->second - 7)/total;
                //printf("(%s) Flowid: %d (%d) has weight %f \n", name(), i->first, i->second, ((do
        }
        //printf("(%s) TOTAL: %f \n", name(), blam);
}
```

Listing D.4: fgee-demarker.cc

```
/*
 * Fgee-demarker module for IP Edge routers.
 *
 * Authors: El-Bahul Fgee <fgeeee@dal.ca>,
 *          Jason Kenney <jdkenney@dal.ca>, 2003
 */

#include <string.h>
#include <queue.h>
#include "fgee-demarker.h"
#include "tcp.h"

static class FgeeDemarkerClass : public TclClass {
public:
        FgeeDemarkerClass() : TclClass("Queue/FgeeDemarker") {}
        TclObject* create(int, const char*const*) {
                return (new FgeeDemarker);
        }
} class_demarker;

FgeeDemarker::FgeeDemarker() {
        q_ = new PacketQueue;

        f2price.clear();
        last_monitor_update_=0.0;
        monitoring_window_ = 0.1;

        for (int i=0; i<=NO_CLASSES; i++) {
                demarker_arrvs_[i]=0;
                arrived_Bits_[i] = 0;
        }

  bind("demarker_arrvs1_",    &(demarker_arrvs_[1]));
  bind("demarker_arrvs2_",    &(demarker_arrvs_[2]));
  bind("demarker_arrvs3_",    &(demarker_arrvs_[3]));
  bind("demarker_arrvs4_",    &(demarker_arrvs_[4]));
}



int FgeeDemarker::command(int argc, const char*const* argv) {
        if (argc == 3) {
                if (strcmp(argv[1], "trace-file") == 0) {
```

```
                        file_name_ = new(char[500]);
                        strcpy(file_name_,argv[2]);
                        if (strcmp(file_name_,"null") != 0) {
                                demarker_type_ = VERBOSE;
                                for (int i=1; i<=NO_CLASSES; i++) {
                                        char filename[500];
                                        sprintf(filename,"%s.%d", file_name_,i);
                                        if ((delay_tr_[i] = fopen(filename,"w"))==NULL) {
                                            printf("Problem with opening the trace files\n");
                                                abort();
                                        }
                                }
                        } else {
                                demarker_type_ = QUIET;
                        }
                        return (TCL_OK);
                } else if (strcmp(argv[1], "id") == 0) {
                        link_id_ = (int)atof(argv[2]);
                        return (TCL_OK);
                }
        }
        if(strcmp(argv[1], "printPrices") == 0) {
                return printPrices();
        }
        return Queue::command(argc, argv);
}


void FgeeDemarker::enque(Packet* p) {
        q_->enque(p);
        if (q_->length() >= qlim_) {
            q_->remove(p);
            drop(p);
            printf("Packet drops in FgeeDemarker of type:%d\n", demarker_type_);
        }
}


Packet* FgeeDemarker::deque() {
  Packet* p= q_->deque();
  if (p==NULL) return p;

  hdr_ip*  iph = hdr_ip::access(p);
```

```
  hdr_cmn* cm_h = hdr_cmn::access(p);
  double   cur_time  = Scheduler::instance().clock();

  //printf("PACKETPRICE: %lf %d\n", iph->fgeeprice_, iph->flowid());
  if(f2price.find(iph->flowid()) == f2price.end()) {
        f2price[iph->flowid()] = iph->fgeeprice_;
        //printf("NEW FLOW\n");
        }
  else {
        f2price[iph->flowid()] += iph->fgeeprice_;
        //printf("ADDING COST\n");
  }

  int cls = iph->prio_;
  if ((cls<1) || ((cls - 8)>NO_CLASSES)) {
    printf("Wrong class type in FgeeDemarker-deque (S=%d, D=%d, FID=%d, Class=%d)\n",
           (int)(iph->src().addr_), (int)(iph->dst().addr_),
           iph->fid_, iph->prio_);

    fflush(stdout);
    abort();
  }



  return p;
}

int FgeeDemarker::printPrices(void)
{
        printf("Flow Id:          Price:\n");
        printf("-------------------------\n");
        for(std::map<int,double>::iterator i = f2price.begin(); i != f2price.end(); i++) {
        printf("%d                $%lf\n", i->first, i->second);
        }
        return (TCL_OK);
}
```

D.2 TCL Code Scripets

D.2.1 QoS Manager Simulation Code

Listing D.5: IPv6-Model-Simulation.tcl

```
Simulator instproc get-link { node1 node2 } {
        $self instvar link_
        set id1 [$node1 id]
        set id2 [$node2 id]
        return $link_($id1:$id2)
}

Simulator instproc get-queue { node1 node2 } {
        global ns
        set l [$ns get-link $node1 $node2]
        set q [$l queue]
        return $q
}

# main
puts " "
set ns [new Simulator]

#ns-random 3298
ns-random 4327

set nf [open out1.tr w]
set qf1 [open bw1 w]
set qf2 [open bw2 w]
set qf3 [open bw3 w]
set nt [open out1.nam w]
$ns trace-all $nf
$ns namtrace-all $nt
Queue/JoBS set drop_front_ false
Queue/JoBS set trace_hop_ true
Queue/JoBS set adc_resolution_type_ 0
Queue/JoBS set shared_buffer_ 1
Queue/JoBS set mean_pkt_size_ 4000
Queue/Demarker set demarker_arrvs1_ 0
Queue/Demarker set demarker_arrvs2_ 0
Queue/Demarker set demarker_arrvs3_ 0
Queue/Demarker set demarker_arrvs4_ 0
#
```

```
set mang [new FgeeManager]


puts "\nTOPOLOGY SETUP"

set hops 2

# Link  latency (ms)
set DELAY       1.0
puts "Links latency: $DELAY (ms)"

# links: bandwidth (kbps)
set BW          1000.0
puts "Links capacity: $BW (kbps)"

Queue/FgeeMarker set bandwidth_ $BW
Queue/WFQ set bandwidth_ $BW

# Buffer size in gateways (in packets)
set GW_BUFF     50
puts "Gateway Buffer Size: $GW_BUFF (packs)"


# Packet Size (in bytes); assume common for all sources
set PKTSZ       500
set MAXWIN      50
puts "Packet Size: $PKTSZ (bytes)"

# Number of monitored flows (equal to # of classes)
set N_CL        4
puts "Number of classes: $N_CL"

set N_USERS     1
puts "Number of flows: $N_USERS"

set START_TM    0.0
puts "Monitored flows start at: $START_TM
"

set max_time    20.0
puts "Max time: $max_time (sec)"

# Statistics-related parameters
set STATS_INTR  2; # interval between reporting statistics
set START_STATS 0; # start-time for reporting statistics
```

```
# Marker Types
# Deterministic
set DETERM      1

# Demarker Types
# Create a trace file for each class (for user traffic)
set VERBOSE     1
# Do not create trace files
set QUIET       2

proc print_time {interval} {
        global ns
        puts ""
        puts -nonewline stdout [format "%.2f\t" [$ns now]]
        $ns at [expr [$ns now]+$interval] "print_time $interval"
}


# 1. Core nodes ($hops nodes)

set edge1 [$ns node] ;#id 0
set edge2 [$ns node] ;#id 1

set core1 [$ns node] ;#id 2
set core2 [$ns node] ;#id 3
set core3 [$ns node] ;#id 4
set core4 [$ns node] ;#id 5

$edge1 color "black"
$edge1 shape "square"
$edge1 label Edge1
$edge2 color "black"
$edge2 shape "square"
$edge2 label Edge2
$core1 label Core1
$core2 label Core2
$core3 label Core3
$core4 label Core4

# 2. User traffic nodes and sinks (N_USERS sources and sinks)

set s1 [$ns node] ;#id 6
$s1 color blue
set s2 [$ns node] ;#id 7
```

```
$s2 color green
set s3 [$ns node] ;#id 8
$s3 color red
set s4 [$ns node] ;#id 9
$s4 color orange

set d [$ns node] ;#id 10

$s1 label "Highest Priority"
$s2 label "Source 2"
$s3 label "Source 3"
$s4 label "Best Effort"

puts "Created Nodes"

$ns duplex-link $edge2 $d [expr $BW*1000.0] [expr $DELAY/1000.0] DropTail

$ns duplex-link $edge1 $s1 [expr $BW*1000.0] [expr $DELAY/1000.0] DropTail
$ns duplex-link $edge1 $s2 [expr $BW*1000.0] [expr $DELAY/1000.0] DropTail
$ns duplex-link $edge1 $s3 [expr $BW*1000.0] [expr $DELAY/1000.0] DropTail
$ns duplex-link $edge1 $s4 [expr $BW*1000.0] [expr $DELAY/1000.0] DropTail

$ns duplex-link $core1 $core2 [expr $BW*1000.0] [expr $DELAY/1000.0] DropTail
$ns duplex-link $core3 $core4 [expr $BW*1000.0] [expr $DELAY/1000.0] DropTail

$ns simplex-link $edge1 $core1 [expr $BW*1000.0] [expr $DELAY/1000.0] FgeeMarker
$ns simplex-link $edge1 $core3 [expr $BW*1000.0] [expr $DELAY/1000.0] FgeeMarker
$ns simplex-link $core1 $edge1 [expr $BW*1000.0] [expr $DELAY/1000.0] DropTail
$ns simplex-link $core3 $edge1 [expr $BW*1000.0] [expr $DELAY/1000.0] DropTail
$ns queue-limit $edge1 $core1 20
$ns queue-limit $edge1 $core3 20
#$ns trace-queue $edge1 $core1 $qf
#set qm [$ns monitor-queue $edge1 $core1 $qf]

$ns simplex-link $edge2 $core2 [expr $BW*1000.0] [expr $DELAY/1000.0] FgeeMarker
$ns simplex-link $edge2 $core4 [expr $BW*1000.0] [expr $DELAY/1000.0] FgeeMarker
$ns simplex-link $core2 $edge2 [expr $BW*1000.0] [expr $DELAY/1000.0] DropTail
$ns simplex-link $core4 $edge2 [expr $BW*1000.0] [expr $DELAY/1000.0] DropTail
$ns queue-limit $edge2 $core2 50
$ns queue-limit $edge2 $core4 50
set qE1C1 [[$ns link $edge1 $core1] queue]

puts "Created Connections"

set udp1 [new Agent/UDP]
```

```
set null1 [new Agent/LossMonitor]

set udp2 [new Agent/UDP]
set null2 [new Agent/LossMonitor]

set udp3 [new Agent/UDP]
set null3 [new Agent/LossMonitor]

set udp4 [new Agent/UDP]
set null4 [new Agent/LossMonitor]

set cbr1 [new Application/Traffic/CBR]
set exp2 [new Application/Traffic/CBR]
set exp3 [new Application/Traffic/CBR]
set exp4 [new Application/Traffic/Exponential]

$ns attach-agent $s1 $udp1
$ns attach-agent $s2 $udp2
$ns attach-agent $s3 $udp3
$ns attach-agent $s4 $udp4
$ns attach-agent $d $null1
$ns attach-agent $d $null2
$ns attach-agent $d $null3
$ns attach-agent $d $null4

$ns connect $udp1 $null1
$ns connect $udp2 $null2
$ns connect $udp3 $null3
$ns connect $udp4 $null4

#$udp1 set packetSize_ $PKTSZ
$cbr1 set fid_ 15
$udp1 set prio_ 15
$udp1 set fid_ 15
$null1 set prio_ 15
$cbr1 attach-agent $udp1

#$udp2 set packetSize_ $PKTSZ
$udp2 set fid_ 12
$udp2 set prio_ 12
$null2 set prio_ 12
$exp2 attach-agent $udp2

#$udp3 set packetSize_ $PKTSZ
$udp3 set fid_ 8
```

```
$udp3 set prio_ 8
$null3 set prio_ 8
$exp3 attach-agent $udp3

#$udp4 set packetSize_ $PKTSZ
$udp4 set fid_ 8
$udp4 set prio_ 8
$null4 set prio_ 8
$exp4 attach-agent $udp4

$cbr1 set rate_ [expr $BW*0.5*1000.0]
$ns at 0.0 "$cbr1 start"
#$ns at 4.0 "$cbr1 set rate_ [expr $BW*0.50*1.3*1000.0]"

$exp2 set rate_ [expr $BW*0.25*1000.0]
$ns at 0.0 "$exp2 start"
$ns at 3.0 "$exp2 set rate_ [expr $BW*0.25*1.3*1000.0]"

$exp3 set rate_ [expr $BW*0.25*1000.0]
$ns at 0.0 "$exp3 start"
#$ns at 2.0 "$exp3 set rate_ [expr $BW*0.25*1.0*1000.0]"

#$exp4 set rate_ [expr $BW*0.2*1000.0]
#$ns at 0.0 "$exp4 start"
#$ns at 1.0 "$exp4 set rate_ [expr $BW*0.2*1.1*1000.0]"

#puts [format "\tCBR-1 starts at %.4fsec" $START_TM]

puts "Created Flows"


set q [$ns get-queue $edge1 $core1]
$mang pushedge $q
$q attach-manager $mang

set q [$ns get-queue $edge1 $core3]
$mang pushedge $q
$q attach-manager $mang

set q [$ns get-queue $edge2 $core2]
$mang pushedge $q
$q attach-manager $mang
set q [$ns get-queue $edge2 $core4]
$mang pushedge $q
$q attach-manager $mang
```

```
$mang setMaxBw 1000000
$mang request 15 400000
$mang request 12 200000
$mang request 10 200000
$mang besteffort 8

$mang addPolicyEntry [$s1 id] [$d id] TokenBucket 15 [expr $BW*0.5*1000] 1000
$mang addPolicyEntry [$d id] [$s1 id] TokenBucket 15 [expr $BW*0.5*1000] 1000
$mang addPolicyEntry [$s2 id] [$d id] TokenBucket 12 [expr $BW*0.25*1000] 1000
$mang addPolicyEntry [$d id] [$s2 id] TokenBucket 12 [expr $BW*0.25*1000] 1000
$mang addPolicyEntry [$s3 id] [$d id] TokenBucket 10 [expr $BW*0.25*1000] 1000
$mang addPolicyEntry [$d id] [$s3 id] TokenBucket 10 [expr $BW*0.25*1000] 1000
#$mang addPolicyEntry [$d id] [$s4 id] TokenBucket  8 200000 200
#$mang addPolicyEntry [$s4 id] [$d id] TokenBucket  8 200000 200
$mang addPolicerEntry TokenBucket 15 12
$mang addPolicerEntry TokenBucket 12 10

puts "Created Policies"

#
# Queue Monitors
#

$ns at $STATS_INTR "print_time $STATS_INTR"

$ns at [expr $max_time+.000000001] "puts \"Finishing Simulation...\""
$ns at [expr $max_time+1] "finish"
proc finish {} {
        global ns nf nt qf1 qf2 qf3
    puts "\nSimulation End"
        $ns flush-trace
        close $nf
        close $nt
        close $qf1
        close $qf2
        close $qf3
        exit 0
}

proc record {} {
        global null1 null2 null3 qf1 qf2 qf3 ns
        set bw1 [$null1 set bytes_]
        set bw2 [$null2 set bytes_]
```

```
        set bw3 [$null3 set bytes_]

        puts $qf1 "[$ns now] [expr $bw1/.01/1000000*8]"
        puts $qf2 "[$ns now] [expr $bw2/.01/1000000*8]"
        puts $qf3 "[$ns now] [expr $bw3/.01/1000000*8]"

        $null1 set bytes_ 0
        $null2 set bytes_ 0
        $null3 set bytes_ 0
        $ns at [expr [$ns now] + 0.01] "record"
}

$ns at 0.0 "record"
$ns at 2.0 "$qE1C1 printStats"
$ns at 4.0 "$qE1C1 printStats"
$ns at 6.0 "$qE1C1 printStats"
$ns at 8.0 "$qE1C1 printStats"
$ns at 10.0 "$qE1C1 printStats"
$ns at 15.0 "$qE1C1 printStats"
$ns at 20.0 "$qE1C1 printStats"

# nam stuff
$ns color 8 red
$ns color 12 green
$ns color 10 blue
$ns color 15 purple
puts "\ngo!\n"
#$ns gen-map
$ns run
```

D.2.2 Pricing Integration Simulation Code

Listing D.6: IPv6-Pricing-Simulation.tcl

```
Simulator instproc get-link { node1 node2 } {
        $self instvar link_
        set id1 [$node1 id]
        set id2 [$node2 id]
        return $link_($id1:$id2)
}

Simulator instproc get-queue { node1 node2 } {
        global ns
        set l [$ns get-link $node1 $node2]
        set q [$l queue]
```

```
        return $q
}

set ns [new Simulator]

set nf [open test_all.tr w]
set nt [open test_all.nam w]
set bw1 [open bw1.out w]
$ns trace-all $nf
$ns namtrace-all $nt

set rng [new RNG]
set uniform [new RandomVariable/Uniform]
$uniform use-rng $rng
$uniform set min_ 0.0
$uniform set max_ 1.0

#$ns namtrace-all $nt
Queue/JoBS set drop_front_ false
Queue/JoBS set trace_hop_ true
Queue/JoBS set adc_resolution_type_ 0
Queue/JoBS set shared_buffer_ 1
Queue/JoBS set mean_pkt_size_ 4000
Queue/Demarker set demarker_arrvs1_ 0
Queue/Demarker set demarker_arrvs2_ 0
Queue/Demarker set demarker_arrvs3_ 0
Queue/Demarker set demarker_arrvs4_ 0

set mang [new FgeeManager]

set DELAY        1.0
set BW           1000.0
set GW_BUFF      50
set PKTSZ        500
set MAXWIN       50
set N_CL         4
set N_USERS      1
set START_TM     0.0
set max_time     40.0
set STATS_INTR   2; # interval between reporting statistics
set START_STATS  0; # start-time for reporting statistics
set DETERM       1
set VERBOSE      1
set QUIET        2
```

```
#####################################################
#
#
set BAND(1) 500000
set BAND(2) 250000
set BAND(3) 250000
set PRIO(1) 15
set PRIO(2) 12
set PRIO(3) 8
set FID(1) 15
set FID(2) 12
set FID(3) 8
#
#
#
#
#
#####################################################

Queue/FgeeMarker set bandwidth_ $BW
Queue/WFQ set bandwidth_ $BW

proc print_time {interval} {
        global ns
        puts ""
        puts -nonewline stdout [format "%.2f\t" [$ns now]]
        $ns at [expr [$ns now]+$interval] "print_time $interval"
}

proc print_price {} {
        global ns mang
        $mang get-price 12
        $mang get-price 15
        $mang get-price 8
        $ns at [expr [$ns now]+0.01] "print_price"
}

proc update_req1 {} {
        global uniform mang FID BAND edge ns
        $mang release $FID(1)
        set b [expr [expr [$uniform value]*.20+0.80]*$BAND(1)]
        $mang request $FID(1) $b [$edge(1) id] [$edge(2) id]
        $ns at [expr [$ns now]+0.25] "update_req1"
}
proc update_req2 {} {
```

```
        global uniform mang FID BAND edge ns
        $mang release $FID(2)
        set b [expr [expr [$uniform value]*.20+0.80]*$BAND(2)]
        #set b [expr [$uniform value]*$BAND(2)]
        $mang request $FID(2) $b [$edge(1) id] [$edge(2) id]
        $ns at [expr [$ns now]+0.25] "update_req2"
}

# 1. Core nodes ($hops nodes)

for {set i 1} {$i <= 2} {incr i} {
        set edge($i) [$ns node] ; #id 0,1,2,3
        $edge($i) label Edge$i
        $edge($i) color "blue"
        $edge($i) shape "square"
}

for {set i 1} {$i <= 4} {incr i} {
        set core($i) [$ns node] ; # id 4,5,6,7, 8,9,10,11
        $core($i) label Core$i
        $core($i) color "black"
        $core($i) shape "circle"
}

#2. User traffic nodes and sinks (N_USERS sources and sinks)

set s(1) [$ns node] ;#id 12
$s(1) color green
set s(2) [$ns node] ;#id 13
$s(2) color red
set s(3) [$ns node] ;#id 14
$s(3) color blue
set d(1) [$ns node] ;#id 14
$d(1) color blue
set d(2) [$ns node] ;#id 15
$d(2) color orange
set d(3) [$ns node] ;#id 16
$d(2) color green
$s(1) label "Source 1"
$s(2) label "Source 2"
$s(3) label "Source 3"
$d(1) label "Destination 1"
$d(2) label "Destination 2"
$d(3) label "Destination 3"
```

```
puts "Created Nodes"

set btmp "1Mb"
set dtmp [expr $DELAY/1000.0]

#Core links
$ns duplex-link $core(1) $core(2) $btmp $dtmp DropTail
$ns duplex-link $core(3) $core(4) $btmp $dtmp DropTail

#Left Side
$ns simplex-link $edge(1) $core(1) $btmp $dtmp FgeeMarker
$ns simplex-link $edge(1) $core(3) $btmp $dtmp FgeeMarker
$ns simplex-link $core(1) $edge(1) $btmp $dtmp DropTail
$ns simplex-link $core(3) $edge(1) $btmp $dtmp DropTail
$ns queue-limit $edge(1) $core(1) 20
$ns queue-limit $edge(1) $core(3) 20

#Right Side
$ns simplex-link $edge(2) $core(2) $btmp $dtmp FgeeMarker
$ns simplex-link $edge(2) $core(4) $btmp $dtmp FgeeMarker
$ns simplex-link $core(2) $edge(2) $btmp $dtmp DropTail
$ns simplex-link $core(4) $edge(2) $btmp $dtmp DropTail
$ns queue-limit $edge(2) $core(2) 20
$ns queue-limit $edge(2) $core(4) 20


$ns duplex-link $edge(2) $d(1) $btmp $dtmp DropTail
$ns duplex-link $edge(2) $d(2) $btmp $dtmp DropTail
$ns duplex-link $edge(2) $d(3) $btmp $dtmp DropTail
$ns duplex-link $edge(1) $s(1) $btmp $dtmp DropTail
$ns duplex-link $edge(1) $s(2) $btmp $dtmp DropTail
$ns duplex-link $edge(1) $s(3) $btmp $dtmp DropTail
set qE1C1 [[$ns link $edge(1) $core(1)] queue]

puts "Created Connections"


        set udp(1) [new Agent/UDP]
        set null(1) [new Agent/LossMonitor]


        $ns attach-agent $s(1) $udp(1)
        $ns attach-agent $d(1) $null(1)

        $ns connect $udp(1) $null(1)
```

```
$udp(1) set fid_ $FID(1)
$null(1) set prio_ $PRIO(1)

set exp(1) [new Application/Traffic/CBR]
$exp(1) set fid_ $FID(1)
$exp(1) set prio_ $PRIO(1)
$exp(1) attach-agent $udp(1)
$exp(1) set rate_ $BAND(1)
$ns at 0.0 "$exp(1) start"


set udp(2) [new Agent/UDP]
set null(2) [new Agent/LossMonitor]


$ns attach-agent $s(2) $udp(2)
$ns attach-agent $d(2) $null(2)

$ns connect $udp(2) $null(2)

$udp(2) set fid_ $FID(2)
$null(2) set prio_ $PRIO(2)

set exp(2) [new Application/Traffic/CBR]
$exp(2) set fid_ $FID(2)
$exp(2) set prio_ $PRIO(2)
$exp(2) attach-agent $udp(2)
$exp(2) set rate_ $BAND(2)
$ns at 0.0 "$exp(2) start"

set udp(3) [new Agent/UDP]
set null(3) [new Agent/LossMonitor]


$ns attach-agent $s(3) $udp(3)
$ns attach-agent $d(3) $null(3)

$ns connect $udp(3) $null(3)

$udp(3) set fid_ $FID(3)
$null(3) set fid_ $FID(3)
$udp(3) set prio_ $PRIO(3)
$null(3) set prio_ $PRIO(3)
```

```
        set exp(3) [new Application/Traffic/CBR]
        $exp(3) set fid_ $FID(3)
        $exp(3) set prio_ $PRIO(3)
        $exp(3) attach-agent $udp(3)
        $exp(3) set rate_ $BAND(3)

        $ns at 0.0 "$exp(3) start"


puts "Created Flows"

        set q [$ns get-queue $edge(1) $core(1)]
        $mang pushedge $q [$edge(1) id] [$core(1) id]
        $q attach-manager $mang
        set q [$ns get-queue $edge(1) $core(3)]
        $mang pushedge $q [$edge(1) id] [$core(3) id]
        $q attach-manager $mang
        set q [$ns get-queue $edge(2) $core(2)]
        $mang pushedge $q [$edge(2) id] [$core(2) id]
        $q attach-manager $mang
        set q [$ns get-queue $edge(2) $core(4)]
        $mang pushedge $q [$edge(2) id] [$core(4) id]
        $q attach-manager $mang

$mang set-price 8  0.04
$mang set-price 12 0.09
$mang set-price 15 0.16

$mang set-point 8 1.0
$mang set-point 12 0.7
$mang set-point 15 0.5

$mang addnodepair $FID(1) [$s(1) id] [$d(1) id]
$mang addnodepair $FID(2) [$s(2) id] [$d(2) id]
$mang addnodepair $FID(3) [$s(3) id] [$d(3) id]

$mang setMaxBw [expr $BW*1000.0]
$mang besteffort 8

$mang addPolicerEntry TokenBucket $PRIO(1) $PRIO(2)
$mang addPolicerEntry TokenBucket $PRIO(2) 8

set b [expr [expr [$uniform value]*.20+0.80]*$BAND(1)]
$exp(1) set rate_ $BAND(1)
```

```
$mang request $FID(1) $b [$edge(1) id] [$edge(2) id]
set b [expr [expr [$uniform value]*.20+0.80]*$BAND(2)]
$exp(2) set rate_ $BAND(2)
$mang request $FID(2) $b [$edge(1) id] [$edge(2) id]


puts "Requesting single domain path"
$mang addPolicyEntry [$s(1) id] [$d(1) id] TokenBucket $FID(1) [expr $BAND(1)] 1000
$mang addPolicyEntry [$d(1) id] [$s(1) id] TokenBucket $FID(1) [expr $BAND(1)] 1000


puts "Should have been ok."
puts "Requesting multiple domain path"
$mang addPolicyEntry [$s(2) id] [$d(2) id] TokenBucket $FID(2) $BAND(2) 1000
$mang addPolicyEntry [$d(2) id] [$s(2) id] TokenBucket $FID(2) $BAND(2) 1000

puts "Requesting single domain path"
$mang addPolicyEntry [$s(3) id] [$d(3) id] TokenBucket $FID(3) [expr $BAND(3)] 1000
$mang addPolicyEntry [$d(3) id] [$s(3) id] TokenBucket $FID(3) [expr $BAND(3)] 1000


puts "Hope it was ok!"

puts "Created Policies"

$ns at $STATS_INTR "print_time $STATS_INTR"

$ns at 2.0 {
        foreach {i} {1 1} {
                set a [$exp($i) set rate_]
                $exp($i) set rate_ [expr $BAND($i)*.50]
                puts "Changing rate from: $a to [$exp($i) set rate_]"
        }
}

$ns at 10.0 {
        foreach {i} {1 1} {
                set a [$exp($i) set rate_]
                $exp($i) set rate_ [expr $BAND($i)*1.0]
                puts "Now Changing rate from: $a to [$exp($i) set rate_]"
        }
}
```

```
$ns at [expr $max_time+.000000001] "puts \"Finishing Simulation...\""
$ns at [expr $max_time+1] "finish"
$ns at 0.0 "print_price"
$ns at 1.125 "update_req1"
$ns at 1.0 "update_req2"

proc finish {} {
        global ns nf nt qE1C1
        $qE1C1 printStats
    puts "\nSimulation End"
        $ns flush-trace
        close $nf
        close $nt
        exit 0
}
$ns color 12 green
$ns color 15 purple
puts "\ngo!\n"
#$ns gen-map
$ns run
```