SCHEDULING MULTIMEDIA TRAFFIC ON IPV6 ENABLED REAL TIME LINUX OPERATING SYSTEM

By

Ms. Priyanka Sharma
(05MCE017)

DEPARTMENT OF COMPUTER ENGINEERING
Ahmedabad 382481
April 2007

SCHEDULING MULTIMEDIA TRAFFIC ON IPV6 ENABLED REAL TIME LINUX OPERATING SYSTEM

Major Project

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Computer Science and Engineering

By

Ms. Priyanka Sharma
(05MCE017)

Guide
Dr. S. N. Pradhan
and
Prof. Jaladhi Joshi

DEPARTMENT OF COMPUTER ENGINEERING
Ahmedabad 382481
April 2007

This is to certify that Dissertation entitled

Scheduling Multimedia Traffic on Ipv6 enabled Real Time Linux Operating System

Presented by

Ms. Priyanka Sharma

has been accepted toward fulfillment of the requirement

for the degree of

Master of Technology in Computer Science & Engineering

Prof. S. N. Pradhan	Prof. D. J. Patel
Professor In Charge	Head of The Department

Prof. A. B. Patel

Director, Institute of Technology

CERTIFICATE

This is to certify that the Major Project entitled “Scheduling Multimedia Traffic on Ipv6 enabled Real Time Linux Operating System” submitted by Ms. Priyanka Sharma (05MCE017), towards the partial fulfillment of the requirements for the degree of Master of Technology in Computer Science and Engineering of Nirma University of Science and Technology, Ahmedabad is the record of work carried out by her under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven’t been submitted to any other university or institution for award of any degree or diploma.

Dr. S. N. Pradhan
Guide,
Professor,
Department of Computer Science & Engineering,
Institute of Technology,
Nirma University,
Ahmedabad.

Prof. Jaladhi Joshi
Co-Guide,
Assistant Professor
Department of Computer Science & Engineering,
Institute of Technology,
Nirma University,
Ahmedabad.

Date: / /

ACKNOWLEDGEMENT

With immense pleasure, I would like to present this report on the dissertation work related to “Scheduling Multimedia Traffic on IPv6 enabled Real Time Linux Operating System”. I am very thankful to all those who helped me for the successful completion of the first phase of the dissertation and for providing valuable guidance throughout the project work.

I would first of all like to offer thanks to **Dr. S. N. Pradhan**, Programme Co-ordinator MTech CS&E, Institute of Technology, Nirma University, Ahmedabad whose keen interest and excellent knowledge base helped me to finalize the topic of the dissertation work. His constant support and interest in the subject equipped me with a great understanding of different aspects of the required architecture for the project work.

My sincere thanks and gratitude to **Prof. D.J. Patel**, Professor and Head Computer Engineering Department, Institute of Technology, Nirma University, Ahmedabad for his continual kind words of encouragement and motivation throughout the Dissertation work

I would also like to thank **Prof. Jaladhi Joshi,** Assistant Professor. CE Department, of Technology, Nirma University, whose knowledge and expertise in Computer Networks has been of a great pillar of support for my work. He has shown keen interest in this dissertation work right from beginning and has offered a great motivating factor in outlining the flow of my work.

I am thankful to Nirma University for providing all kind of required resources. I would like to thank The Almighty, my family, especially my husband and my daughter, for supporting and encouraging me in all possible ways. I would also like to thank all my friends who have directly or indirectly helped in making this dissertation work successful.

Ms. Priyanka Sharma
Roll No.: 05MCE017

ABSTRACT

Multimedia traffics are beginning to conquer computer networks. This thesis investigates into the scheduling aspects of real time multimedia streams with a real time test-bed environment. This will include the Operating System Scheduling Issues, as well as Routing at the Network Layer.

The architecture configured for the project work has been put forward with intricately defined requirements for each component. An Ipv6 enabled distributed system is designed for priority based scheduling of Multimedia stream generated by VLC's VideoLAN Software. The traffic is further routed based on Quality of Service requirements, through software based IP Router. The software router is configured to run on the Real Time kernel (RTLinux3.1). Out of the two software based routers (Zebra and XORP), which was configured and studied, XORP was selected was the project work, due to its extensibility and robustness. As compared to conventional hardware based routers, which are generally not open systems, XORP can be used as a router and can be extended through new XORP Processes, to implement new routing protocols, which can be integrated and tested in real time working environment.

The Forwarding Plane of the XORP Router uses the internal file structures like routing table of Linux kernel; hence the thesis work also deals with OS Kernel level Scheduling mechanisms.

Another aspect of the project work is QoS based scheduling of the multimedia traffic. This is because; the Multimedia Applications must adhere to stringent real-time constraints and Quality-of-Service (QoS) requirements. Therefore, a DiffServ based, QoS management approach is developed to orchestrate and guarantee the timely interaction between such applications and services. In this context, a kernel level module is developed to parse the network packets and identify the DSCP value, which is marked during multimedia streaming at the source. The further scheduling of the traffic is implemented based on this. The report also covers; several other approaches that can be implemented for QoS based scheduling of multimedia traffic.

CONTENTS

Certificate			I
Acknowledgement			II
Abstract			III
Contents			IV
List of Figures			VIII
List of Tables			X
Glossary			XI
Chapter 1	**Introduction**		**01**
	1.1	General	01
	1.2	Background	01
	1.3	Objective of Study	03
	1.4	Scope of Work	04
	1.5	Organization of Major Project	04
Chapter 2	**Literature survey**		**06**
	2.1	Introduction	06
	2.2	Real Time Operating Systems	07
	2.3	Multimedia and IPv6	08
	2.4	Clustered Middleware for Real Time Multimedia Routing	09
	2.5	QoS based routing of Multimedia Traffic	10
	2.6	IP Software Router	11
		2.6.1 Purpose	11
		2.6.2 Characteristics	12
		2.6.3 Zebra	14
		2.6.4 XORP	16
	2.7	Real Time Scheduling	25
		2.7.1 Scheduling Multimedia on RTOS	26
	2.8	Routing Multimedia	26
		2.8.1 General	26
		2.8.2 Multimedia Routing in Real Time	29

2.8.3 QoS based Routing 29
2.9 Summary 33

Chapter 3 Project Architecture 34
3.1 Introduction 34
3.2 Project Architecture 34
3.3 Installation and Configuration 36
3.3.1 RTLinux 3.1 with Linux kernel 2.4.20 and Ipv6 stack implementation 36
3.3.2 Ipv6 Layer as a router 38
3.3.3 Ipv6 to IP4 Tunneling 39
3.3.4 Streaming with VLC's VideoLAN's 43
3.3.5 Installation and Configuration of Zebra 45
3.3.6 Installation and Configuration of XORP with Ipv6 Stack 47
3.4 Summary 50

Chapter 4 QoS for Multimedia traffic in Ipv6 53
4.1 Introduction 53
4.1.1 Issues related to QoS 53
4.1.2 Parameters used to measure QoS at the network layer 54
4.1.3 Ipv4 header format 56
4.1.4 Ipv6 protocol header 57
4.1.5 Ipv6 flow label requirements 58
4.1.6 Bandwidth, delay and buffer requirements approach to Using flow labels 59
4.2 Approaches of Implementation 60
4.2.1 DiffServ Aware Video Streaming 61
4.2.2 Ipv6 and flow label usage 64
4.2.3 Real time linux kernel DiffServ Implementation 66
4.3 Summary 67

Chapter 5 Real Time Linux Kernel DiffServ Implementation 68
5.1 Introduction 68
5.2 Enabling QoS Support in the Linux Kernel 69
5.3 Installing Traffic Control (tc) utility of iproute2 70
5.4 Diffserv implementation overview 71
5.5 Resources 72
5.6 Queuing discipline 74
5.7 Classes 78
5.8 Module invocation from real time kernel 79
5.9 Summary 79

Chapter 6 Writing a XORP Process 80
6.1 Introduction 80
6.2 Creating a XORP process 80
6.3 Creating a XORP process for adding routes dynamically 81
6.3.1 Creating XRL interfaces 81
6.3.2 The Main Loop 84
6.3.3 Calling XRLs on the RIB 86
6.3.4 Returning values in XRLs 87
6.3.5 The XLOG logging facility 87
6.4 Summary 88

Chapter 7 Results 89
7.1 General 89
7.2 Experimental setup 89
7.3 Traffic generation 90
7.4 Measurement tools 91
7.5 Methodology 92
7.6 Time synchronization 92
7.7 Traffic generation 93
7.8 Test 95
7.9 Results 96
7.10 MPEG-2 TS dissector for ethereal 0.10.8 98

Chapter 8 Summary and Conclusion **101**

8.1 Summary 101

8.2 Conclusions 101

8.3 Future Work 102

References **104**

Appendix – A List of websites **106**

LIST OF FIGURES

Fig 2.1	Zebra Architecture	16
Fig 2.2	XORP Process model	18
Fig 2.3	Example XRL Targets and client	25
Fig 2.4	A queue for each egress line	27
Fig 2.5	Set of queues associated with an exit	28
Fig 2.6	Connection between a router and a host in IS Internet	29
Fig 3.1	Project Architecture (Higher Level)	36
Fig 4.1	Ipv4 Packet format	56
Fig 4.2	Ipv6 Protocol Header	57
Fig 4.3	Soft Real Time Application Traffic	59
Fig 4.4	Hard Real Time Application Traffic	59
Fig 4.5	Straightforward AF mapping strategy for MPEG Streams	62
Fig 4.6	IPv6 DiffServ-aware streaming system	63
Fig 5.1	Processing of network data	69
Fig 5.2	A simple queuing discipline with multiple classes	71
Fig 5.3	Combination of priority, TBF and FIFO queuing disciplines	72
Fig 5.4	Relation of elements of the IntServ and DiffServ architecture to traffic control in the Linux kernel	73
Fig 5.5	Flow chart showing the functions called when enqueuing and sending packets	76
Fig 7.1	IPv4 bandwidth consumption	96
Fig 7.2	IPv6 bandwidth consumption	96
Fig 7.3	Packet Loss for the High Priority Multimedia Traffic	96

Fig 7.4 Packet Loss for different packet sizes 97

Fig 7.5 Latency for different packet sizes 97

Fig 7.6 Ethereal's Open page for Transport stream file 98

Fig 7.7 Ethereal's Transport stream file details 99

Fig 7.8 Transport stream file, showing a High Transport priority 100

LIST OF TABLES

Table 5.1	Networking options during kernel configurations	70
Table 5.2	QoS and fair Queuing Options during kernel configurations	70
Table 7.1	Profile for Traffic Generation for both IPv6 and IPv4 network	95
Table 7.2	Comparison of IPv4 and IPv6	95

GLOSSARY

Quality-of-Service (QoS): A set of service requirements to be met by the network while transporting a flow.

Traffic Control (tc): Traffic control is a module of iproute2 package. It is related to network based Quality of Service (QoS) management issues.

Service class: The definitions of the semantics and parameters of a specific type of QoS.

Integrated services: The Integrated Services model for the Internet Defined in RFC 1633, allows for integration of QoS services with the best effort services of the Internet. The Integrated Services (IntServ) working group in the IETF has defined two service classes, Controlled Load Service and Guaranteed Service.

Differentiated services: The field in which the differentiated services class is encoded. It is the Type of Service (TOS) octet in the (DS) field IPv4 header or the traffic class octet in the IPv6 header.

QoS-based routing: A routing mechanism under which paths for flows are determined based on some knowledge of resource availability in the network as well as the QoS requirement of flows.

XORP: XORP, or Extensible Open Router Platform, is an open source routing software suite, aimed at being both stable and fully featured enough for production use and also extensible to support networking research. It is stable and secure and event driven. The router is developed on object oriented paradigm and can be extended through XORP processes. The software is coded in C++ and as a very active research area.

ZEBRA: Zebra is open source TCP/IP routing software that is similar to Cisco's Internetworking Operating System (IOS). The software is distributed

under GNU (General public License). Though the software works very much like CISCO routers, but lacks certain functionalities, like it is not multithreaded. The API of the software is still under development.

Routing Information Base (RIB): In XORP router, the RIB holds a user-space copy of the entire routing/forwarding table, complete with information about where each route came from (e.g., which protocol, and when). It communicates with the routing protocols such as BGP, RIP and OSPF to instantiate routes, and with the FEA to install the appropriate forwarding entries in the FEs

Forwarding Engine Abstraction (FEA): In XORP router, the FEA provides a platform independent interface to the basic routing and network interface management functionality. For example, get or set information about network interfaces, install or modify unicast forwarding entries or multicast routing support.

XORP Resource Locators (XRLs): The XORP Resource Locator is used to communication between the modules of a XORP router. This includes forwarding engine, routing daemons, router manager etc.

ipc finder: IPC (Inter-process Communication) finder process stores mappings between application requests, such as “What are this router’s interfaces?”, and the particular IPC calls necessary to answer those requests.

Xorpsh: The xorpsh is the process, that provides an interface; for interaction with the XORP router.

1. INTRODUCTION

1.1 GENERAL

The dissertation work related to 'Scheduling of multimedia traffic on IPv6 enabled Real Time Linux operating system' demands the setup of a distributed infrastructure. This requires to be done very meticulously, in order to meet the demands of real time multimedia traffic. The proposed architecture is inspired from a Video On Demand Server based application. The system includes one Real Time Linux server where files are stored. The client requests for a particular multimedia file from a remote host. The file is streamed on the server and sent to the client side. The server and the client lie on different networks. Hence routers come into picture and the multimedia traffic needs to be routed, with Quality of Service parameters. The routers used in the setup is XORP, a software based router. There are at least four routers each configured for different network is used. This forms the middleware of the implementation. Most of the results derived are for unicast transmission of data.

The protocol stack on which the architecture is implemented is IPv6, since Ipv6 has got inherent support for real time applications and multimedia communications. Here the client side could be a Windows based system also. But the source server and the software routers are all implemented on XORP router running over RTLinux3.1 kernel, further configured on Red Hat Linux or Fedora Core 6 nodes.

1.2 BACKGROUND

Multimedia has proven itself to be an efficient and effective medium for information communication. Recently there have been many developments in technologies, which have made multimedia information communication possible to reach the unapproachable, over the best-effort network i.e. the Internet, over the wireless devices i.e. through mobile networks etc. The prominent developments can be grouped in six parts.

First, the improving bandwidth of the internet link. The backbone as well as the last mile for internet has come out of its age. Growth from 56 kbps to Gbps link backbone and 28 kbps to 2 Mbps (ADSL) last mile provided the platform for the multimedia communications. The recent development Internet2 has been speed benchmarked for 9 Gbps data transfers over 20,000 miles.

Second, the newer CODECS. The development of excellent CODECS for video and audio greatly supported the cause. MPEG4, H264, Theora (open video codec, targeted at competing with MPEG-4) and many others could greatly shrink the heaviest component of multimedia, which includes; the video & audio and still provide acceptable quality.

Thirdly, the alternate mediums. The existence of alternate mediums provided the platform for wider reach. Wireless technology, the mobile communications have grown to 4G age, providing the right set of Quality of Service for the multimedia data communication.

Fourth, the newer better protocols. While addressing the shortcomings of ancestors, newer protocols also provide novel ways for delivering better services. Ipv6 have definite advantages over Ipv4. Likewise the advantage of availability of multicasting over unicasting.

Fifth, the development of newer hardware technologies delivered faster processing powers that enabled heavy data processing, the core requirement of multimedia data. Faster CPUs, larger & faster RAMs, HDDs made it possible to handle the heavy multimedia data in real-time environments.

Sixth, the age of open source development. The open source community has made it possible to assemble geographically distant groups of technocrats for the development of a concept/technology. Many open source tools/utilities have been developed which has enabled further R & D into the subject. The Linux & VideoLan project is the best thing that

happened to the OS & multimedia streaming technologies from R&D point of view, for academics.

The ball is rolling, but there are still many challenges to be met. The multimedia with its demanding Quality of Service, have always been providing the thrust/need for further developments. The continuous nature of data, heavy data, compression related issues, communication through heterogeneous systems, response times as well as efficient handling of intermediately routers/schedulers, compatibility issues among codecs & protocols are few related ongoing areas of active research.

1.3 OBJECTIVE OF STUDY

Apart from the above mentioned goals and vision, this report addresses the specific aspect of QoS based routing of multimedia traffic. The experimental setup has been created in a single Lab, with software based routers configured with various network configurations. Hence, the objective of the dissertation work can be summarized as:

Create architecture for streaming multimedia traffic on a real time operating system and schedule them to the destination host. The scheduling strategies should look into both operating system and network related issues.

Study and configuration of software based routers and implement them for the project work. This is a special feature of the thesis work. Study related to various methods for QoS based routing of Multimedia IP Traffic in Ipv6, hav also been made.

The next objective is, implementation of a kernel level module to read/set the DSCP value in the Traffic Class of IPv6 header and TOS field of Ipv4 field. The UDP traffic that is generated by streaming a multimedia file is to be scheduled as per the DiffServ architecture framework.

Finally, the objective is to test the performance of the network in routing

of multimedia traffic, with the developed module on the experimental test-bed. For this purpose, a heavy traffic is generated artificially using traffic Generation Tool.

1.4 SCOPE OF WORK

The experimental setup, prepared for the dissertation work, includes a relatively small network established in a Computer Lab. Atleast four routers with different network based configurations are used for this purpose. A special feature of the thesis work is study and implementation of XORP and ZEBRA, both software based routers. This is relatively a very new technology and is like a boon for the Computer Network fraternity. Out of the various QoS based implementation techniques for multimedia traffic, the thesis work includes a detailed implementation of DiffServ framework implementation in real time kernel.

1.5 ORGANIZATION OF THE PROJECT WORK

A large part of the project work comprised of the required infrastructure for implementation for scheduling of multimedia traffic in a real time environment. The next part was the development of a module for the implementation of DiffServ framework in Real Time kernel.

The representation of the project work has been described chapter-wise in total eight chapters. The **Chapter2** describes various issues related to literature survey for the project work. This includes a detailing of the various components of the architecture, XORP and ZEBRA as software based routers, and a theoretical analysis of the Multimedia QoS in Ipv6 and Ipv4 network. The **Chapter 3** describes the required configuration and installation of the required architecture. This includes the configuration of XORP and ZEBRA routers also. Then the next three chapters ie Chapter 4, Chapter 5 and Chapter 6 describe various implementation modules developed for the project work. **Chapter 4** describes a general view of the implementation strategies; **Chapter 5**

covers a detailed description of the development of a module at the Real Time Linux kernel. This is based on the DiffServ Framework. **Chapter6** covers a detailed description of the programming related to the extension of the XORP process for including the provision, of identifying the multimedia traffic based on the DSCP value and assigning it high priority, which is interpreted by XORP. The next chapter i.e. **Chapter 7**, covers a detailing of the experiments carried out and results obtained. The last chapter i.e. **Chapter 8** is the concluding chapter of the thesis work and covers Summary, Conclusion and Future Scope of the project.

2. LITERATURE SURVEY

2.1 GENERAL

The dissertation work related to 'Scheduling of multimedia traffic on IPv6 enabled Real Time Linux operating system' requires the setup of a distributed infrastructure. The proposed architecture describes a video on demand Server (VOD). The system includes one source where, Multimedia Files are stored. The selected Multimedia File is streamed using VideoLAN's VLC software, with DSCP field set with highest priority. The streamed data is processed by a module running the real time kernel, and then forwarded to IP based Software routers, based on the priority. Two Software routers have been selected for the purpose. This includes Zebra and XORP routers.

The XORP router is extensible in nature, and can be extended for priority based scheduling. The multimedia stream is passed onto one of the routers, which may be forwarded to the destination based on some Quality of Service parameters. The ZORP routers are software based routers hence can be clustered to increasing the performance of a router in processing the routing related jobs. This can be termed as the middleware of the proposed architecture of the project work. The protocol stack on which the architecture is implemented is IPv6. This is because Ipv6 has got inherent support for real time applications and multimedia communications. It can support Quality of Service related processing, through the eight bit Traffic Class field and the twenty bit Flow Labels. Here the client side initiating the request for the multimedia file could be on Microsoft Windows based system, but the software routers and the source machine/multimedia server has Real-Time Linux (RTlinux3.1) Operating System running over Fedora Core 6 (2.6.20) Operating systems.

A significant part of the thesis work is preparation of a real time environment for transmission of multimedia traffic and later scheduling them on priority basis. The architecture can be divided into various components which are covered in the following sections.

2.2 REAL TIME OPERATING SYSTEMS

A Real Time System is an information system whose performance does not only depend on the logical output of the algorithms but also on the moment in time when these output occurred. Real-time systems are typically implemented with multiple asynchronous threads of execution. This is dictated by the need to react to external events, and control asynchronous devices. Because of this characteristic, an RTOS must support multithreading. Also, because the criticality and rates of events are different, the RTOS must support a notion of priority so that a time-critical task is not delayed because of a non-critical task. Furthermore, tasks need to communicate. Therefore, the OS must provide synchronization and communication facilities.

An RTOS also needs to support timing features like high-resolution timers and clocks. Timers are used to support periodic processing and to detect system timeout errors. Clocks are needed to keep track of time. Typical real-time applications may need to be aware of time at a granularity of micro- or milliseconds.

The design of a real-time system goes through several phases: First the tasks to be performed and the temporal restrictions that must be satisfied are identified, Secondly the code is written and Finally the run-time of each task is measured and a schedulability test is done to ensure that the task will not miss its deadline while the system is running. The schedulability test consists on applying a number of tests to the whole set of tasks, and if they pass the tests then it will be possible to warranty that no task will loose its deadline. If the tests are not passed it is necessary to start our design again from the beginning, picking a faster CPU or using other algorithms to implement the tasks.

In summary, the tasks are identified with three time values: Pi stands for Period of Task i, Di stands for Deadline for task i and Ci stands for worse case Computational time for task i.

The objective of the system is to warranty that all tasks (in all their executions) will satisfy their deadlines. To warranty execution times the system must be predictable. Saying that a system is real-time or that the system is predictable is practically the same statement.

2.3 MULTIMEDIA AND IPV6

The transportation of multimedia traffic on IP networks is an important subject because multimedia is becoming cheaper and cheaper and therefore used more and more. All workstations and personal computers available today are equipped with sound boards for recording and reproducing sounds and with video boards for viewing MPEG images. Some of them are now equipped with video input and with small video cameras. Problems with bearing multimedia flows on IP networks are mainly related to the bandwidth they require and to the strict maximum delay requirements that must be met. This second point is particularly important when multimedia applications have to provide users with real-time interaction.

Mbone is a network layered on the Internet for multimedia applications. Multimedia traffic has a multicast nature intrinsically and therefore the need to improve the routing of multicast packets on IP networks. Some characteristics of IPv6 will improve the support of multimedia applications (in the following, also called real-time applications), such as the availability of the Priority field and of the Flow Label field on the IPv6 header [1] and the availability of a large addressing space reserved for multicast addresses. Moreover, other protocols of the stack introduce significant rationalizations in this field. ICMPv6 includes functions for the management of multicast groups, and OSPFv6 provides the treatment of multicast trees.

IPv6 is part of a more ambitious project called IS (Integrated Service) Internet, which is discussed in RFC 1633; it aims to extend the Internet architecture to allow the bearing of either best-effort or real-time traffic, as well as to control the use of transmission links (controlled link sharing). The best-effort traffic is

the only type of traffic that has been used on the Internet till now. It is based on the idea that the network's task is to do everything possible to deliver each IP packet, without guaranteeing the packet is delivered or the delivery time. Multimedia applications frequently generate real-time traffic—that is, a type of traffic sensitive to queuing delays and to losses due, for example, to network overloading. Moreover, this type of traffic frequently needs a guaranteed minimum bandwidth. The possibility of reserving a minimum bandwidth on links for particular classes of users, or protocol stacks, is in general a requirement understood by network administrators, also independently from multimedia applications.

Clearly, typical real-time applications—like the transmission of remote video images, multimedia conferences, and virtual reality—require the extension of IP by introducing the concept of QoS. The extension must in some way allow limited packet delays and must be designed, from the beginning, for IP multicast because most of the multimedia traffic is multicast.

2.4 CLUSTERED MIDDLEWARE FOR REAL TIME MULTIMEDIA ROUTING

The routers used for the implementation are software based. The efficiency of the software based routers can be combined, to extend the capability of individual routers. This way the processing that are required for processing at the forwarding plane, are combined through a cluster, to form a middleware. The activities on the nodes (running routers) are synchronized by the message passing libraries like LAM/MPI. The requirements for maintaining end to end Quality of Service parameters are managed on this middleware. Thus, the main reason behind having a clustered middleware is to manage the multimedia traffic as well as schedule them, both at operating system as well as network layer.

2.5 QOS BASED ROUTING OF MULTIMEDIA TRAFFIC

The advent of Quality of Service (QoS) in the Internet is being fostered by an

increasing need to provide adequate network services to a vast range of QoS-demanding applications. In the presence of distinct applications and traffic profiles, service providers need to differentiate customers so that an efficient and cost-effective network resource management can be achieved. In addition to admission control, reservation protocols, resource management solutions, or even when these are not present, acceptable QoS conditions can be obtained in the presence of an appropriated delay differentiation mechanism. In fact, delay differentiation can be extremely useful to integrate real-time applications and other delay sensitive applications. From the applications' perspective there are two crucial aspects for the integration of real-time applications in IP networks: the ability to satisfy end-to-end delay requirements and the capability to absorb excess delays. The former aspect is related to real-time applications, such as video conferencing systems and other interactive applications which are highly delay sensitive. In this context, the deployment of scheduling mechanisms providing queuing delay bounds plays a crucial role in the integration of real-time traffic in IP networks.

The latter aspect is related to mechanisms used by real-time applications in order to smooth excess delays or to adapt to network conditions. This means that in addition to end-to-end delay bounds it is useful to provide expectable differentiation mechanisms to handle excess queuing delays inside the network. In this context, the use of rigid admission control procedures and resource reservations protocols (e.g. in the IntServ architecture) play a relevant role. These solutions, suffering from well known lack of scalability and flexibility, led to lighter and easier to deploy solutions (e.g. DiffServ) [3]. As expected, relaxing QoS-guarantees in the network raises additional problems of integration of real-time applications mainly due to the absence of rigid admission control in the core routers. Even existing admission control at network edges the network core may cause feasibility problems to schedulers due to: i) the transient effect caused by dynamic flow aggregation; ii) traffic distortion and packet clustering caused by cascade queuing effects; iii) possible shaping inaccuracies at edge routers and iv) path changes caused by a route flip. In addition, delay-oriented scheduling mechanisms may show feasibility problems as regards their configuration parameters.

Hence in order to main QoS in multimedia networks, it is important to develop scheduling mechanisms able to react to particular congestion situations in order to achieve expectable differentiation behaviors among real-time traffic classes. Since, the dissertation work involves a real time linux operating system, running over linux kernel; the scheduling mechanisms will encompass both operating system based scheduling as well as proper routing strategies for multimedia traffic management for a video conferencing system.

2.6 IP SOFTWARE ROUTER

An IP software router has basically two tasks. Routing is generally accomplished by maintaining a routing table in each end system and each router, which gives, for each possible destination network, the next router to which the internet datagram should be sent.

2.6.1 Purpose

In general, routers open the IP packet to read the destination address, calculate the best route, and then send the packet toward the final destination. If the destination is on the same part of an intranet, the packet would be sent directly to the destination computer by the router. If the packet is destined for another intranet or subnetwork (or if the destination is on the Internet), the router considers factors like traffic congestion and the number of hops-a term that refers to the number of routers or gateways on any given path. The IP packet carries with it a segment that holds the hop count and a router will not use a path that would exceed a predefined number of hops

This long description of router tasks summarizes the two parts. Firstly, the router forwards the datagram from sender towards the destination in the network. Secondly, the router keeps the routing table synchronized between the other routers. In the following subsections we describe these two tasks in detail.

The thesis work discusses two major IP based Software router. The first among

them is ZEBRA Free routing software distributed under GNU General Public License. The GNU Zebra is maintained through the support of IP Fusion, a company which develops network software. The other software is XORP, an eXtensible Open Router Platform. The core team is based at the International Computer Science Institute in Berkeley, California, but contributors come from around the world.

2.6.2 Characteristics

The following sections will discuss the general characteristics of a software based router.

2.6.2.1 Traffic Forwarding

Forwarding is the process a router goes through, for each packet it receives. The packet may be consumed by the router, it may be output on one or more interfaces of the router, or both. Forwarding includes the process of deciding what to do with the packet as well as queuing it up for (possible) output or internal consumption.

We use the above definition for forwarding in this thesis. The process which decides where the packets should be forwarded uses specific rules for that. These rules are stored in the Forwarding Information Base (FIB). The table containing the information necessary to forward IP Datagrams, in this document, is called the Forwarding Information Base. At minimum, this contains the interface identifier and next hop information for each reachable destination network prefix.

Both forwarding and FIB are very important terms when we speak about routing. If a router was a customer servant, IP Datagrams would be its customers. The customer serving job would then be called forwarding. Even if the packets were coming to the router itself, they should be forwarded to a suitable receiving process. To continue more this example of customers and their servant, we can imagine the situation in a hotel's reception. The receptionist receives customers. The receptionist is now our forwarder. He/she

looks the customer room from booking list, or FIB in our terms. Finally the receptionist points the way to the room, or a next hop, to the customer. This was a different situation, but the same forwarding functionality.

2.6.3 Routing Table Management

The second task of a router is to keep its routing table up to date. Routing table can also be described by route database. The term "router" derives from the process of building this route database; routing protocols and configuration interact in a process called routing. This sentence tells the core idea behind the route database. Router gathers information of the network via routing protocols and saves that to the database. Another alternative is to modify the configuration by hand. Both these ways makes the router able to adapt to the network changes dynamically.

The hand made changes are easy to understand, but how can the router adjust its internal database when network changes? This case was above in the previous subsection where we thought about the case when one link goes down. This failure can be noticed by routing protocols. A router uses them to adjust its database automatically to the network changes. We have different kinds of routing protocols, but in this case their functionality is similar. Routing protocols constantly send polling messages over the link. Default time interval for the polling packets for example in Routing Information Protocol (RIP) is 30 seconds. Then, if the router has not received the polling packet from the link for 180 seconds, the link is marked as failed.

All these kinds of routing information is then in the routing table. We can assume that in the beginning, when the network is initially set up, the network has also other routing traffic. These messages include route add and probably also route remove messages. But after this initial routing table synchronization messaging, the network transfers only these link polling packets. This means, from the routing table perspective, that routing table has minor alternating needs after initialization.

2.6.3 ZEBRA

Zebra is open source TCP/IP routing software that is similar to Cisco's Internetworking Operating System (IOS). It is distributed under GNU (General public License), GNU Zebra is maintained through the support of IP Fusion. Flexible and powerful, it can handle routing protocols such as Routing Information Protocol (RIP), Open Shortest Path First (OSPF), Border Gateway Protocol (BGP), and all of their various flavors. It runs on Linux as well as other UNIX variants. Zebra is included in most modern distributions as routing software. The latest version, along with documentation, is available at the GNU Zebra Web site.

Kunihiro Ishiguro and Yoshinari Yoshikawa wrote the original Zebra package back in 1996. Today, primarily IP Infusion, of which Mr. Ishiguro is the CTO, maintains the package, with the assistance of networking engineers and open source volunteers. Zebra is unique in its design because it takes a modular approach to the protocols that it manages. Protocols can be enabled or disabled as network requirements dictate.

By far the most useful feature identified with Zebra was its close similarity to the Cisco IOS configuration formats. While there are some differences from IOS, the feel is close enough that network engineers already familiar with IOS will feel very comfortable in the environment.

2.6.3.1 Features

Some of the features, which make zebra routing software suitable for distributed routing platform, are discussed in the following section.

One of the main feature of Zebra, software based router is that, it provides a modular mechanism to add new protocols as they are developed. The zebra execution environment runs one process per protocol. Also, compared with CISCO hardware based routers, it is easy to add new protocols because of the API of zebra. The benefits of multiple servers are that, it increases modularity and extensibility, and is easy to debug the zebra code. In Zebra, each server

has a terminal interface that is similar to the Cisco terminal interface. Servers have their own configuration file.

But, the adverse side of this architecture is that, due to multiple processes, the router can be slow in processing, because of the time incurred in inter-process communication. Also, the kernel table updates and redistribution could be slow. One of the major drawback of Zebra is, its API is not complete.

2.6.3.2 Zebra Architecture

Zebra is made from a collection of several daemons that work together to build a routing table. There may be several protocol-specific routing daemons and zebra the kernel routing manager. The ripd daemon handles the RIP protocol, while ospfd is a daemon which supports OSPF protocol, bgpd supports the BGP-4 protocol. For changing the kernel routing table and for redistribution of routes between different routing protocols, there is a kernel routing table manager zebra daemon. Zebra is the routing manager. It provides kernel routing table updates, interface lookups and redistribution of routes between different routing protocols.

Each daemon has its own routing table. Zebra daemon is responsible for kernel routing table update (service) and its redistribution between different protocol (e.g. RIPd). Other daemons are for protocol handling.

Zebra protocol is used between protocol daemon and zebra. Each protocol daemon sends selected routes to the zebra daemon. The zebra daemon manages which route is installed into the forwarding table. Zebra protocol is a TCP based protocol.

All the daemons listen on a particular port for incoming VTY (Virtual Terminal Interface). VTY is a command line interface (e.g. telnet) to change or view the current configuration. Zebra's daemon's have their own VTY.

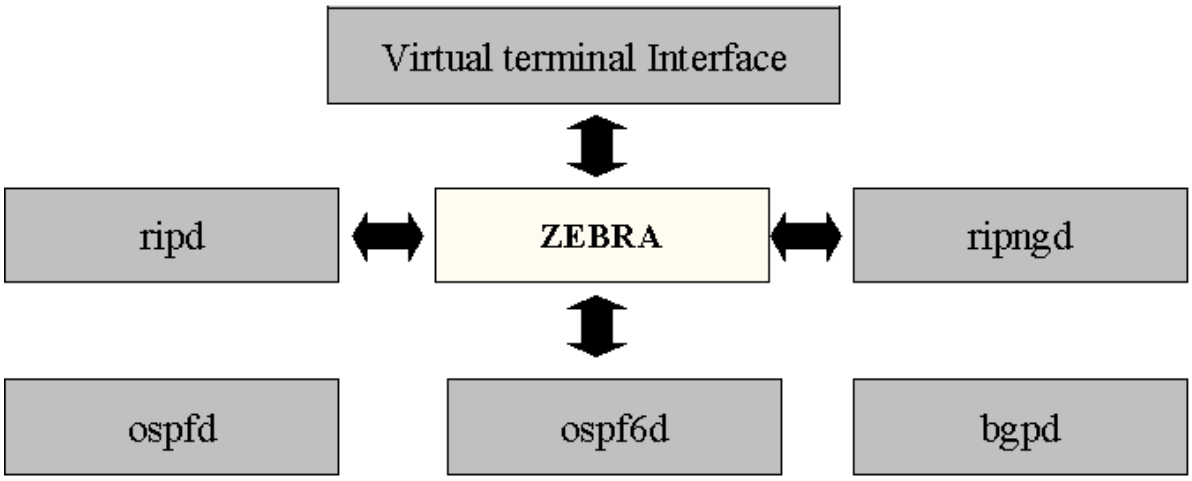

Fig 2.1: Zebra Architecture

2.6.4 XORP

At present, XORP does not implement its own forwarding system. It is reliant on the forwarding of the underlying host operating system. XORP can be deployed in commercial and research environments. It is stable and secure. XORP, or Extensible Open Router Platform, is an open source routing software suite, aimed at being both stable and fully featured enough for production use and also extensible to support networking research. The project was founded by Mark Handley in 2000, with first production release in July 2004. The project is now headed by Atanu Ghosh of the International Computer Science Institute, in Berkeley, California. The current XORP release is version 1.4.As of 2007, the code supports the several routing protocols like, BGP (including multiprotocol extensions for IPv6, Route Reflection, Confederations, Communities), RIP (v2 for IPv4, and RIPng for IPv6), OSPF (RFC2328 (OSPFv2) and RFC2740 (OSPFv3) for both IPv4 and IPv6), PIM (Sparse Mode for both IPv4 and IPv6), IGMP (v1, v2 and v3 (IPv4 only protocol)), MLD (v1 and v2 (IPv6 only protocol)).The XORP codebase consists of around 500,000 lines of C++, and is fairly portable running on FreeBSD, Linux, OpenBSD, DragonFlyBSD, NetBSD, Mac OS X and there is even a port to Windows Server 2003 which only supports IPv4.At present, XORP does not implement its own forwarding system. It is reliant on the forwarding of the underlying host operating system. XORP can be deployed in commercial and research environments. It is stable and secure.

2.6.4.1 Purpose

The basic purpose of the XORP router is specified by its basic functionalities like; it is open source, the XORP code runs on commodity PC hardware and operating systems. The modular design and separation of functional entities into distinct processes means that failure of one component does not cause the whole router to fail or malfunction. The modular design also enables future integration with hardware forwarding planes. XORP components communicate with a flexible IPC scheme. Components can run within a single host or be distributed across multiple hosts. There is scope for building a distributed router from low-cost PC hardware.

The router however has certain disadvantages like; its forwarding plane limits the forwarding rate. Also, the number of features is much less as compared to commercial hardware based routers. Support options do not exist like the big vendors. The project core does not have the resources to provide this level of support. If a company wants to commercialize XORP, this would be one area of differentiation for the commercial product.

2.6.4.2 XORP Architecture

The XORP design philosophy stresses extensibility, performance and robustness. For routing and management modules, the primary goals are extensibility and robustness. These goals are achieved by carefully separating functionality into independent modules, running in separate UNIX processes, with well-defined APIs between them. Clearly, there are performance penalties to pay for such architecture, but we believe that so long as careful attention is paid to computational complexity, the costs associated with inter-process communication will be acceptable given the obvious extensibility and robustness benefits.

For the forwarding path, the primary goals are extensibility and performance. Robustness here is primarily achieved through simplicity and a modular design that encourages re-use of well-tested components.

XORP can be divided into two subsystems. The higher-level i.e. User Space subsystem consists of the routing protocols and management mechanisms. The lower-level i.e. Kernel Space, provides the forwarding path, and provides APIs for the higher-level to access. User-level XORP uses a multi-process architecture with one process per routing protocol, and a novel inter-process communication mechanism known as XORP Resource Locators (XRLs). XRL communication is not limited to a single host, and so XORP can in principle run in a distributed fashion. For example, we can have a distributed router, with the forwarding engine running on one machine, and each of the routing protocols that update that forwarding engine running on a separate control processor system. The lower-level subsystem can use traditional UNIX kernel forwarding, the Click modular router or Windows kernel forwarding (Windows Server 2003). The modularity and minimal dependencies between the lower-level and user-level subsystems allow for many future possibilities for forwarding engines. As standards such as those being developed in the IETF For CES Working Group emerge, we expect to support them to provide true forwarding engine interchangeability.

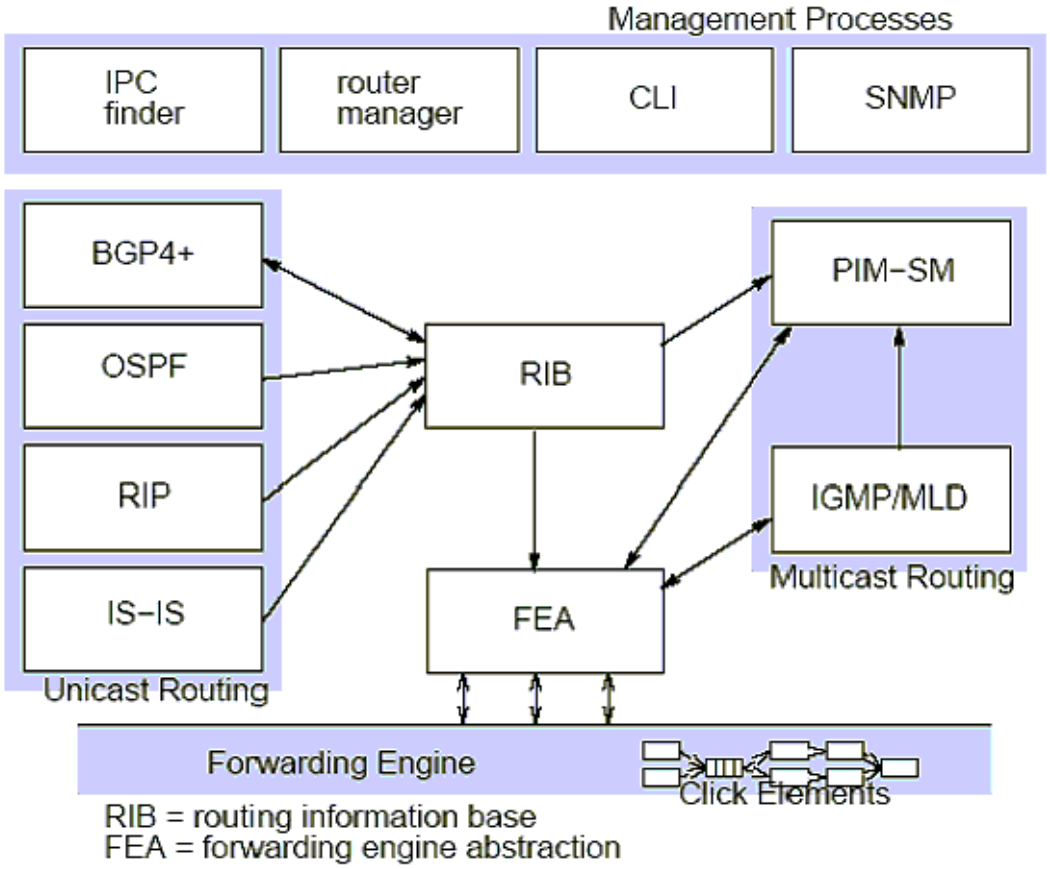

Fig 2.2: XORP Architecture

The Fig 2.2 shows the processes in XORP, although it should be noted that some of these modules use separate processes to handle IPv4 and IPv6. For simplicity, the arrows show only the main communication flows used for routing information. Control flows are not shown - for example, the FEA may need to inform the routing processes when an interface goes down.

We should note that even though our design philosophy is that each XORP component in Fig 2.2 will run as a separate process, it would also be possible to compile most of them together to run as one single process. Obviously, the robustness of such a router would suffer because the crash of a single component would bring down the whole router. However, the XORP architecture is designed to be flexible, and other developers building on this software can choose to use it in different ways. Let us take a look into the main components of XORP architecture.

FEA (Forwarding Engine Abstraction)

The FEA provides a platform independent interface to the basic routing and network interface management functionality. For example, get or set information about network interfaces, install or modify unicast forwarding entries, multicast routing support,

If the router is distributed, in the sense that some Forwarding Engines (FEs) are not in the same chassis as the control software, then the FEA will also handle control communications with the remote FEs. Note that, strictly speaking, it is not required that all communication with the FE must go through the FEA. For example, Click modules can communicate directly with a user-space process. However, the FEA abstracts all the details about the underlying system from the user-level XORP processes; therefore by using the common API provided by the FEA we can greatly simplify the rest of the XORP components. Note that the multicast-related functionalities are logically separated from the unicast-based functionalities in the MFEA (Multicast Forwarding Engine Abstraction), though the MFEA is part of the FEA process.

RIB (Routing Information Base)

The RIB holds a user-space copy of the entire routing/forwarding table, complete with information about where each route came from (e.g., which protocol, and when). It communicates with the routing protocols such as BGP, RIP and OSPF to instantiate routes, and with the FEA to install the appropriate forwarding entries in the FEs.

The RIB also holds the routing information for multicast-capable routes (MRIB) to be used for multicast Reverse-Path Forwarding (RPF) information. For example, PIM-SM uses the MRIB information to route joins/prunes and to determine the RPF interface for sources and for RPs. The MRIB is populated by whatever "unicast" protocol is used to provide multicast capable path information. Typically this is MBGP for inter-domain paths, where it is possible to tell the difference between unicast-capable and multicast-capable routers. For intra-domain routing, this usually is the regular unicast FIB information.

Note that, for unicast, routing and forwarding tables are practically the same. In case of multicast, the RIB provides the RPF information, while the forwarding information (the incoming and outgoing interfaces) is computed by the particular multicast routing protocol. The multicast forwarding information is kept by the multicast routing protocol itself, and installed directly through the FEA. In the future, when there is more than one multicast routing protocols, XORP may have the multicast equivalent of RIB that would be responsible for coordinating among the different multicast routing protocols running on the same router. On a router with multiple FEs, the RIB is responsible for splitting up the routing table amongst the FEs and for figuring out how to forward between FEs.

BGP4+

This is the BGP routing daemon. It implements IPv4 and IPv6 unicast routing in a single process, as well as MBGP for both IPv4 and IPv6 multicast RIBs for multicast routing purpose.

OSPF

This is the OSPF routing daemon. There are separate IPv4 and IPv6 daemons, because unlike BGP there is no real need to tie them together.

RIP

This is the RIP routing daemon. Similarly to OSPF, the IPv4 and IPv6 daemons are separate.

MLD/IGMP

This is Multicast Listener Discovery/Internet Group Management Protocol handler. It implements the router-side part of MLD and IGMP. Its main purpose is to discover local multicast members and propagate this information to multicast routing daemons such as PIM-SM. Similar to OSPF and RIP, the IGMP (IPv4) and MLD (IPv6) daemons are separate.

PIM-SM

This is the PIM-SM multicast routing daemon. Similar to OSPF and RIP, the PIM-SM IPv4 and IPv6 daemons are separate. The PIM-SM protocol requires information about local multicast members, and Reverse-Path Forwarding to operate properly. The former is obtained from the IGMP/MLD process; the latter is obtained from the RIB.

RTRMGR: XORP Router Manager

The rtrmgr is the process responsible for starting all components of the router, to configure each of them, and to monitor and restart any failing process. It also provides the interface for the CLI to change the router configuration.

CLI: Command Line Interface

The CLI can be used by an user to access the router, view its internal state, or to configure it on-the-fly. Its functionality is closely related to the rtrmgr. However, because the robustness of the rtrmgr itself is extremely important, all functionality that can be run as a separate CLI process are separated from the rtrmgr. The process implementing this CLI functionality is called xorpsh.

Inter-Process Communication Finder

The IPC finder is needed by the communication method used among all XORP components, i.e., the XRLs. Each of the XORP components registers with the IPC finder. The finder assists the XRL communications (more specifically, it knows the location of each XRL target), therefore a XORP process does not need to know explicitly the location of all other processes, or how to communicate with them. The router manager process (rtrmgr) incorporates a finder, so a separate finder process is only needed if the rtrmgr is not being used such as during testing.

2.6.4.3 XORP Router Manager

The rtrmgr is normally the only process explicitly started at router startup. The rtrmgr process includes a built-in XRL finder, so no external finder process is required. The basic operation of the XORP, with respect to can be described as follows:

The rtrmgr reads all the template files in the router's template directory. Typically there is one template file per XORP process that might be needed. A template file describes the functionality that is provided by the corresponding process in terms of all of the configuration parameters that may be set. It also describes the dependencies that need to be satisfied before the process can be started. After reading the template files, the rtrmgr knows all the configuration parameters currently supportable on this router, and it stores this information in its template tree. After all template files are read, the template tree is checked for errors (e.g., invalid variable names, etc). The rtrmgr will exit if there is an error.

The rtrmgr next reads the contents of the XRL directory to discover all the XRLs that are supported by the processes on this router. These XRLs are then checked against the XRLs in the template tree. As it is normal for the XRLs in the XRL directory to be used to generate stub code in the XORP processes, this forms the definitive version of a particular XRL. Checking against this version

detects if a template file has somehow become out of sync with the router's codebase. Doing this check at startup prevents subtle run time errors later. The rtrmgr will exit if a mismatch is discovered.

The rtrmgr then reads the router configuration file. All the configuration options in the config file must correspond to configurable functionality as described by the template files. As it reads the config file, the rtrmgr stores the intended configuration in its configuration tree. At this point, the nodes in the configuration tree are annotated as not existing - that is this part of the configuration has not yet been communicated to the process that will implement the functionality.

The rtrmgr next traverses the configuration tree to discover the list of processes that need to be started to provide the required functionality. Typically not all the available software on the router will be needed for a specific configuration. The rtrmgr traverses the template tree again to discover an order for starting the required processes that satisfies all their dependencies.

Thus, the rtrmgr starts the first process in the list of processes to be started. If no error occurs, the rtrmgr traverses the configuration tree to build the list of commands that need to be executed to configure the process just started. A command can be either an XRL or an external program. These commands are then called, one after another, with the successful completion of one command triggering the calling of the next. The commands are ordered according to the command semantics (e.g., see below the description of commands %create, %activate, etc). If the semantics of the commands do not specify the ordering, then the commands follow the order they are defined in the rtrmgr template files. Some processes may require calling a transaction start command before configuration, and a transaction complete command after configuration - the rtrmgr can do this if required. If no error occurred during configuration, the next process is started, and configured, and so forth, until all the required processes are started and configured.

At this point, the router is up and running. The rtrmgr will now allow connections from the xorpsh process to allow interactive operation.

2.6.4.4 XRL Interfaces

XRL Target as something that XRL requests can be directed to in order to be executed. Multiple XRL targets can exist within a single process, though there will typically be one target per process. Each XRL Target has an associated name and a set of IPC mechanisms that it supports. At start-up each target registers its name and IPC mechanisms with the Finder, so other processes are able to direct requests to it. We define an XRL Interface to be a set of related methods that an XRL Target would implement. XRL Interface Client is defined to be a process that accesses a particular XRL Interface

Each XRL Interface is uniquely identified by a name and version number. An XRL Target will nearly always implement multiple interfaces and potentially multiple versions of particular interface. For instance, every XRL Target could implement a process information interface that would return information about the process hosting the target. Each XRL Target will implement interfaces specific to their field of operation: a routing process would implement the “routing process interface” so that the RIB process can make identity-agnostic XRL calls for tasks like redistributing a route to another routing protocol.

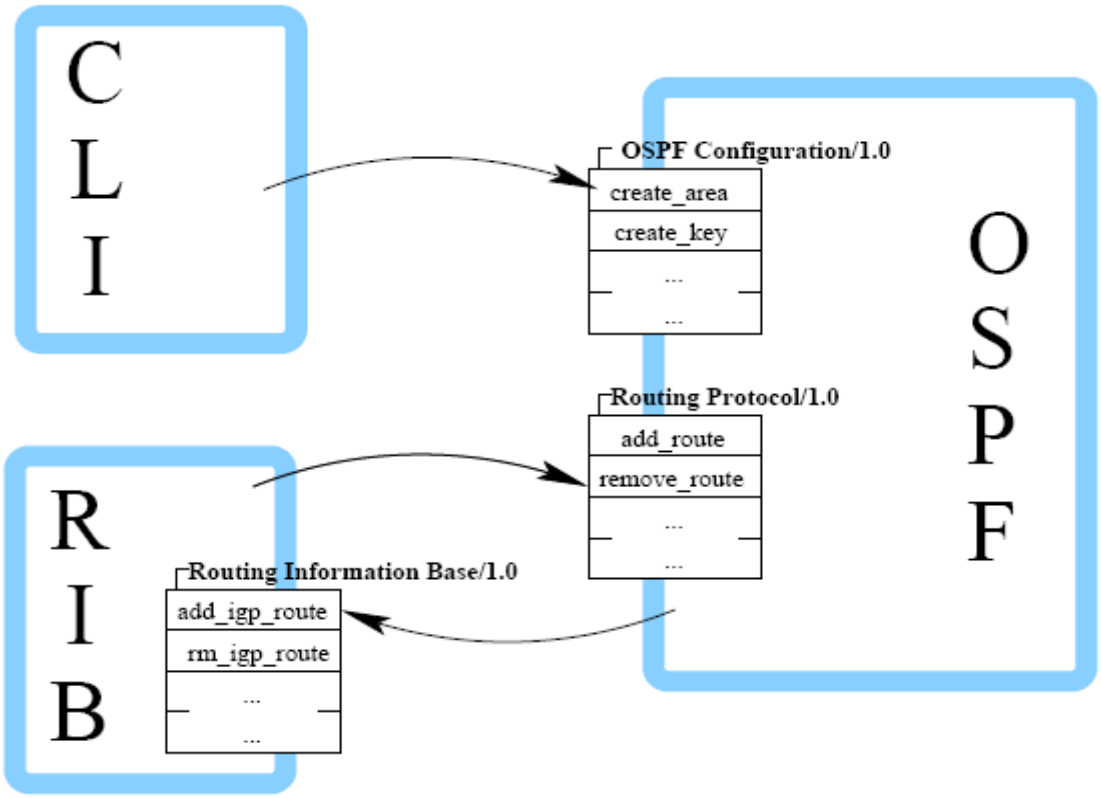

Figure 2.3: Example XRL Targets and clients: OSPF, RIB, and CLI. OSPF and RIB processes are XRL targets. OSPF supports XRL interfaces “OSPF Configuration/1.0” and “Routing Protocol/1.0”: the CLI is a client of the “OSPF Configuration/1.0” interface and the RIB a client of the “Routing Protocol/1.0” interface. RIB supports XRL interface “Routing Information Base/1.0”: the OSPF is a client of that interface.

2.7 REAL TIME SCHEDULING

Real-time systems are typically implemented with multiple asynchronous threads of execution. This is dictated by the need to react to external events, and control asynchronous devices. Because of this characteristic, an RTOS must support multithreading. Also, because the criticality and rates of events are different, the RTOS must support a notion of priority so that a time-critical task is not delayed because of a non-critical task. Furthermore, tasks need to communicate. Therefore, the OS must provide synchronization and communication facilities.

2.7.1 Scheduling Multimedia on RTOS

The most significant characteristic of multimedia applications is probably the real-time constraint. That is, a packet generated at the source should be received by the receiver before a specified time which is denoted as the deadline. Otherwise, this packet would become useless and will be dropped. Besides the real-time constraint, different applications may have different degrees of importance and hence will be assigned different levels of priority. Since it is impossible for any scheduling policy to avoid dropping packets for multimedia applications when the network load is heavy; hence, the good scheduler should drop packets with lower priorities first if packet dropping is unavoidable. We define normalized penalty denoted as NP[4], as

NP = sum of priorities of all lost packets / sum of priorities of all packets

Clearly, a good scheduling policy should be able to keep the value of NP as small as possible. There have been works on the design of scheduling algorithms for applications with real-time constraint. However, these works treat all packets with equal importance; hence, their scheduling policies depend only on the real-time constraints. The scheduling policy proposed in this paper will take into account both the real-time constraint and the priority values of packets.

2.8 ROUTING MULTIMEDIA

2.8.1 General

The basic implementation strategies of a router are subdivided into two parts: the forwarding path (lower part) and the background code (upper part). The additional blocks, in comparison with a common router, are the packet scheduler, the admission control agent, the classifier, and the reservation setup agent. These blocks operate on data flows, and this concept is clearly present in IPv6.

Current routers are designed for best-effort traffic; therefore, they treat packets with a simple FIFO (First In, First Out) queuing for each egress line. As for integrated services, a router must provide an appropriate QoS for each flow, and it must therefore be equipped with a module for the traffic control. This module consists of three sub-modules; packet scheduler, packet classifier and admission control. The packet scheduler guarantees the QoS administering the transmission of packets through a mechanism for the periodical visit of a set of queues. The packet classifier recognizes which flow a packet belongs to and queues it on the corresponding queue. A queue can be associated with a single flow or to a class of flows. The admission control decides, in response to a request of resource reservation from the reservation agent, whether this packet can be accepted. The decision is made on the basis of resources reserved by other flows, of the network administration policies set through the control agent, and of globally available resources. In practice, this module checks whether the requested QoS can be provided without colliding with the guarantees of service provided to other flows.

The presence of a classification module and of a packet scheduler requires that each egress line be associated with a set of queues.

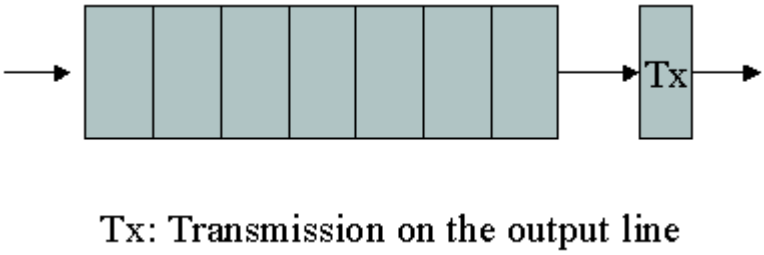

Fig 2.4: A queue for each egress line

The presence of a set of queues is a necessary, but not sufficient, characteristic to guarantee the QoS. It is, in fact, necessary that the scheduler guarantees that the frequency with which each queue is served is greater than or equal to that guaranteed during the resource reservation. This forces us to have a separate queue for each real-time flow (in the example, R1, R2, R3 e R4) and a shared queue for the best-effort traffic. The best-effort traffic will clearly be penalized, and it will be served only in the absence of real-time traffic.

The model of admission control is sometimes confused with the so called policing, a control mechanism that checks packet by packet that a host doesn't violate traffic characteristics agreed upon by a previous QoS agreement. In this case, the packet scheduler provides the policing. The fourth and last component is the resource reservation protocol, which is necessary to create and maintain the state of each flow on the routing path and which allows the interaction between reservation agents. The protocol chosen by the IETF is RSVP (Resource reservation Protocol)

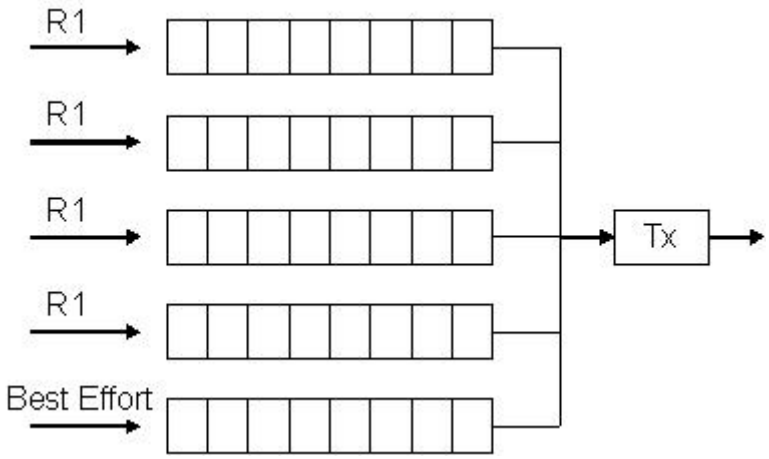

Fig 2.5: Set of queues associated with an exit

The implementation for hosts is usually similar to that of routers, but with the addition of applications. Fig 2.6 shows the interconnection between a host and a router.

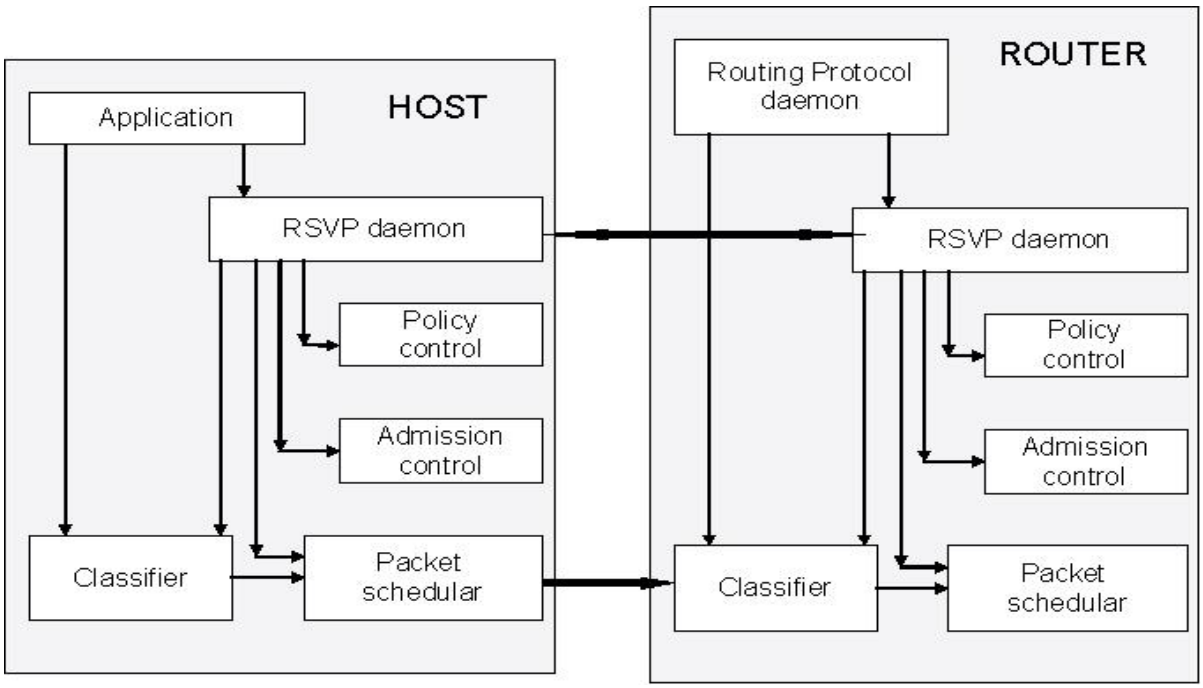

Fig 2.6: Connection between a router and a host in IS Internet

The host's data are received by an application that, if needing QoS for a flow, must request it from the local reservation agent (the RSVP agent)

2.8.2 Multimedia routing in Real Time

Multimedia applications have become some of the most important applications on high speed local and metropolitan area networks. One of the major characteristics of multimedia traffic is the real-time constraint of packets. Furthermore, in multimedia applications, packets of different applications may have different levels of priority. Hence, a proper scheduling policy is necessary to schedule packets with different priorities and real-time constraints. This paper presented a scheduling policy for real-time traffic with priorities on broadcast networks to reduce the probability of packet loss due to exceeding the real-time constraint. Moreover, if packet loss is unavoidable, this scheduling Policy will avoid dropping the packets with high priorities.

2.8.3 QOS based Routing

QoS-based routing was recognized as a missing piece in the evolution of QoS-based service offerings in the Internet, in the past. This section will include

some of the QoS based routing issues. The Internet Engineering Task Force (IETF) has proposed many service models and mechanisms to meet the demand for QoS. Notably among them are the integrated services/Resource Reservation Protocol (RSVP) model, the differentiated services (DS) model, multiprotocol label switching (MPLS), traffic engineering, and constraint-based routing.

The integrated services model is characterized by resource reservation. For real-time applications, before data are transmitted, the applications must first set up paths and reserve resources. RSVP is a signaling protocol for setting up paths and reserving resources. In differentiated services, packets are marked differently to create several packet classes. Packets in different classes receive different services. MPLS is a forwarding scheme. Packets are assigned labels at the ingress of an MPLS-capable domain. Subsequent classification, forwarding, and services for the packets are based on the labels. Traffic engineering is the process of arranging how traffic flows through the network. Constraint-based routing is to find routes that are subject to some constraints, such as bandwidth or delay requirement.

It is important to understand the difference between QoS-based routing and resource reservation. While resource reservation protocols such as RSVP provide a method for requesting and reserving network resources, they do not provide a mechanism for determining a network path that has adequate resources to accommodate the requested QoS. Conversely, QoS-based routing allows the determination of a path that has a good chance of accommodating the requested QoS, but it does not include a mechanism to reserve the required resources.

2.8.3.1 Objectives

Under QoS-based routing, paths for flows would be determined based on some knowledge of resource availability in the network, as well as the QoS requirement of flows. The main objectives of QoS-based routing are; dynamic determination of feasible paths, Optimization of resource usage, Graceful performance degradation.

In dynamic determination of feasible paths scheme, QoS-based routing can determine a path, from among possibly many choices, that has a good chance of accommodating the QoS of the given flow. Feasible path selection may be subject to policy constraints, such as path cost, provider selection, etc.

The Optimization of resource usage, refers to the fact that, a network state-dependent QoS based routing scheme can aid in the efficient utilization of network resources by improving the total network throughput. Such a routing scheme can be the basis for efficient network engineering.

The graceful performance degradation, means that, the state-dependent routing can compensate for transient inadequacies in network engineering (e.g., during focused overload conditions), giving better throughput and a more graceful performance degradation as compared to a state-insensitive routing scheme.

2.8.3.2 Real Time Flows

In real-time quality-of-service, delay variation is generally more critical than delay as long as the delay is not too high. Clearly, voice-based applications cannot tolerate more than a certain level of delay. The condition of varying delays may be expected to a greater degree in a shared medium environment with datagrams, than in a network implemented over a switched substrate. Routing a real-time flow therefore reduces to an exercise in allocating the required network resources while minimizing fragmentation of bandwidth. The resulting situation is a bandwidth-limited minimum hop path from a source to the destination. In other words, the router performs an ordered search through paths of increasing hop count until it finds one that meets all the bandwidth needs of the flow. To reduce contention and the probability of false probes (due to inaccuracy in route tables, the router could select a path randomly from a "window" of paths which meet the needs of the flow and satisfy one of three additional criteria: best-fit, first-fit or worst-fit. Note that there is a similarity between the allocation of bandwidth and the allocation of memory in a multiprocessing system. First-fit seems to be appropriate for a

system with a high real-time flow arrival rates; and worst-fit is ideal for real-time flows with high holding times.

2.8.3.3 *QoS based Routing for Multicast Flows*

QoS-based multicast routing is an important problem, especially if the notion of higher-level admission control is included. The dynamism in the receiver set allowed by IP multicast, and receiver heterogeneity adds to the problem. With straightforward implementation of distributed heuristic algorithms for multicast path computation, the difficulty is essentially one of scalability. To accommodate QoS, multicast path computation at a router must have knowledge of not only the id of subnets where group members are present, but also the identity of branches in the existing tree. In other words, routers must keep flow-specific state information. Also, computing optimal shared trees based on the shared reservation style, may require new algorithms.

2.8.3.4 Intra-domain Routing Requirements

At the intra-domain level, the objective is to allow as much latitude as possible in addressing the QoS-based routing issues. Indeed, there are many ideas about how QoS-based routing services can be provisioned within ASs. These range from on-demand path computation based on current state information, to statically provisioned paths supporting a few service classes.

Another aspect that might invite differing solutions is performance optimization. Based on the technique used for this, intradomain routing could be very sophisticated or rather simple. Finally, the service classes supported, as well as the specific QoS engineered for a service class, could differ from AS to AS. For instance, some ASs may not support guaranteed service, while others may. Also, some ASs supporting the service may be engineered for a better delay bound than others. Thus, it requires considerable thought to determine the high level requirements for intradomain routing that both supports the overall view of QoS-based routing in the Internet and allows maximum autonomy in developing solutions.

Hence, intra-domain routing in order to be qualified as “QoS-based” routing

must satisfy certain minimum requirements. These are, the routing scheme must route a flow along a path that can accommodate its QoS requirements, or indicate that the flow cannot be admitted with the QoS currently being requested. Also, the routing scheme must indicate disruptions to the current route of a flow due to topological changes. The routing scheme must also, accommodate best-effort flows without any resource reservation requirements. That is, present best effort applications and protocol stacks need not have to change to run in a domain employing QoS-based routing. Finally, the routing scheme may optionally support QoS-based multicasting with receiver heterogeneity and shared reservation styles. Thus, the intra-domain requirements are expected to be rather broad.

2.9 SUMMARY

The chapter discusses, literature survey related to various components of the project work. The first part dealt with, analysis of the subparts of the topic of the dissertation, like real time operating system, multimedia and IPv6, QoS of Multimedia in IPv6. The description regarding two software based routers studied was also discussed. The last section, which was core to the discussion, was related to issues governing routing of multimedia traffic.

3. PROJECT ARCHITECTURE

3.1 GENERAL

Multimedia traffic is known for its stringent quality of service requirements. The dissertation work that includes study work related to scheduling of multimedia traffic. The proposed architecture described in this chapter, is influenced by a Video On Demand Server (VOD) Architecture. The system comprises of one source where, Multimedia Files are stored. The operating platform identified for implementation of the source is a Real Time Linux operating system (RTLinux 3.1), which is configured to run on Linux Kernel (Fedora Core 6 with kernel 2.6.20). The multimedia file is streamed using VLC's VideoLAN software, from the source computer. This multimedia IP traffic, while going to the destination passes through software based routers (XORP, configured with iPv6 support)[4]. The network is also overloaded with heavy TCP traffic generated artificially by a Traffic Generator Tool called Mgen. The multimedia traffic is prioritized by a module developed to run in the Real Time Linux kernel. The XORP router is also extended, to identify the priority based traffic and route them accordingly to the destination host. The extension of the XORP router is through a XORP Process, created and running at the XORP routers. The requested file is later displayed at the client side. Various QoS Parameters, like Packet Loss, Latency, Throughput, and bandwidth occupied, are tested in the above configuration.

3.2 PROJECT ARCHITECTURE

As per the above discussion the proposed architecture is like a Video On Demand Server. The source computer is where multimedia files are stored. RTLinux kernel runs over Linux kernel in this node. The Real Time distribution which has been selected for work is RTLinux from FSMLabs. The base operating system on which RTLinux works is Fedora Core 6 and Red Hat Linux 9. IP Software router has been installed and configured on both the types of Linux platforms. The routers used for experimental purpose includes Zebra and

XORP. The features of both these routers have been discussed in Section 2.2.2 and Section 2.2.3 of the report.

Out of the two XORP router can be configured to work as a cluster, in which the routing based decisions and the forwarding plane can remain common. Hence the requirement of having a clustered middleware has been accomplished.

VideoLAN's VLC Software has been selected for streaming a multimedia file. The file is streamed and sent from the output interface with the specified destination address. An interface to send the request for a multimedia file to the VOD Server, from the client side is also created.

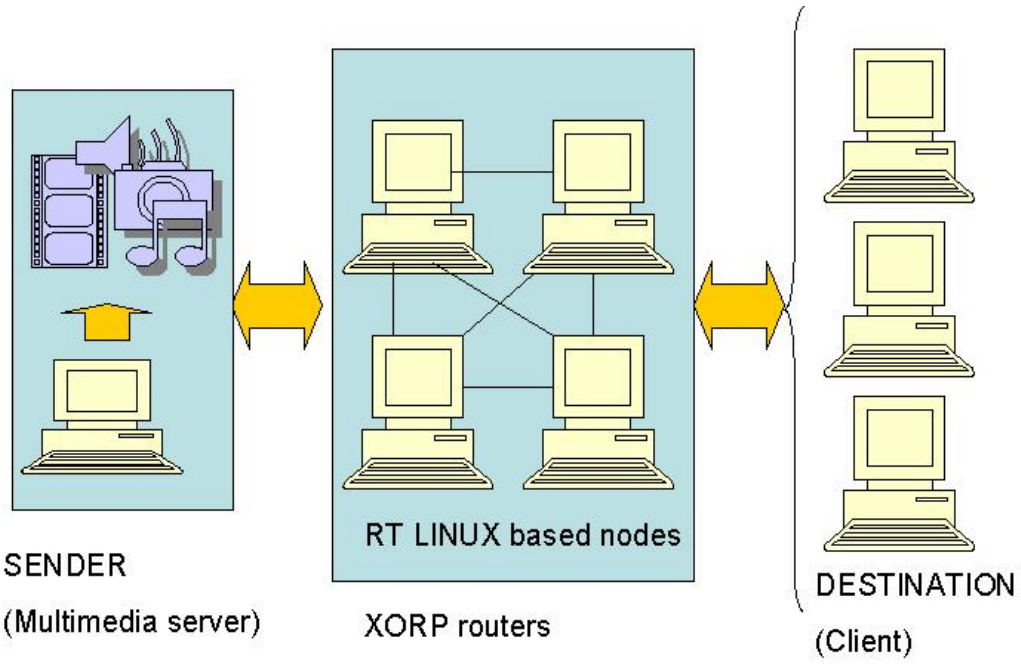

Fig 3.1: Project Architecture (Higher Level)

The streamed data is first sent to IP based Software routers, which works as a clustered middleware. The protocol stack on which the architecture is implemented is IPv6, since Ipv6 has got inherent support for real time applications and multimedia communications. Two strategies of managing quality of service of the multimedia stream have been implemented. The first strategy is related to developing a Multimedia Transport Dissector[] before the streamed traffic is out of the source server into the network stream. The other strategy is similar to the implementation of the Differentiated Services Architecture. In this technique the Type of Service field of the IPv4 and the

Traffic Class of the IPv6 header are updated. The multimedia traffic are forwarded to the destination based on some Quality of Service parameters.

Since the thesis work has various aspects to its overall architecture of implementation, the section below will discuss each in detail.

3.3 INSTALLATION AND CONFIGURATION

The Installation and configuration of the proposed architecture can be divided into various sections. This includes installation of Real Time Linux (RTLinux3.1) on Fedora Core 6 (kernel 2.6.20). The kernel would support both IPv6 and IPv4 stack. Steps required to tunneling traffic from IPv6 to IPv4 network are also discussed. The next part of the discussion is related to installation and configuration of Software routers, both Zebra and XORP, working on VLC's VideoLAN for streaming multimedia traffic on both Linux and Windows operating system. Let us look into the required configurations in detail in the following sections.

3.3.1 RTLinux 3.1 with Linux Kernel (2.6.20) , with IPv6 support

As mentioned in the previous discussion in Chapter 2, the network stack on which the implementation has been carried is Ipv6. This is due to Ipv6's inherent support for real time traffic management, especially the 8 bit field Traffic class and the 20 bit field FlowLabel. These fields will be used for Quality of Service of the multimedia stream. This will be discussed in the coming chapters. The present section discusses the steps required to configure Ipv6 support in the RTLinux kernel running on Linux kernel 2.4.20. In the RTKernel, running over fedora Core6, the option for Ipv6 stack support can be mentioned during installation itself.

The first step is to download the Prepatched Linux kernel from FSMLabs in /usr/src folder and uncompress it:
tar -xzf prepatched_linux_kernel-2.6.20.tar.gz

You will have a directory called linux.2.6.20-rtl. Create a link to this directory:

ln -s /usr/src/linux-2.6.20-rtl /usr/src/linux_ipv6

Having done that, you need to configure the new kernel to enable support for IPv6:

$make xconfig

We need to enable two options in the kernel configuration. First, go to Code Maturity Level and enable development/incomplete code/drivers:

"Prompt for development and/or incomplete code/drivers" YES

Then go to the Networking Options. There you will enable the IPv6 protocol:

IPv6 Protocol (EXPERIMENTAL) YES

Also select the Davicom Fast Ethernet Driver under 10 B Fast Ethernet section, from Load drivers section. This is the configuration required at the kernel level. The next step is to save the configuration and exit by clicking on the Save and Exit button. This will create a .config file in /usr/src/linux, which is the kernel configuration file. Now the kernel is to be compiled, by following these steps:

make clean

make dep

make bzImage

The result will be a new kernel image created in /usr/src/linux/arch/i386/boot/. If some other features are added as modules, then the compilation has to be done by the following procedure.

make modules

make modules_install

At this point, copy the new IPv6-enabled boot image to /boot, and update the System.map file:

cp /usr/src/linux/arch/i386/boot/bzImage /boot/bzImage.ipv6

cp /usr/src/linux/System.map /boot/System.map-2.6.20-ipv6

ln -fs /boot/System.map-2.6.20-ipv6 /boot/System.map

The only remaining step is to update /etc/grub.conf.conf file to add an entry for the new IPv6-enabled kernel. Edit the /etc/grub.conf file and add a new entry as follows:

title Real Time Linux (Ipv6)
kernel /boot/System.map ro root/dev/hda5

Then the grub loader is to be reinstalled, by the following command.
$grub-install /dev/hda0

Reboot the System, by selecting the option of 'Real Time Linux (Ipv6)'. The system should boot properly.

3.3.2 Ipv6 Layer as a Router

Datagrams flow through the IPv6 layer in two directions: from the network *up* to user processes and from user processes *down* to the network. Using this orientation, IPv6 is layered *above* the network interface drivers and *below* the transport protocols such as UDP and TCP.

The IPv6 layer will normally act as a router (forwarding datagrams that are not addressed to it, among other things) when the machine has two or more IPv6 interfaces that are up. This behavior can be overridden by using ndd(1M) to set the /dev/ip6 variable, ip6_forwarding. The value 0 means, do not forward; the value 1 means forward. The initialization scripts (see /etc/init.d/inetinit) set this value at boot time based on the number of "up" interfaces and whether or not the neighbor discovery protocol daemon configuration file /etc/inet/ndpd.conf exists. The default value is zero; ip6_forwarding is set to 1 only if more than one interface has been configured for IPv6 and if /etc/inet/ndpd.conf exists.

Additionally, finer grained forwarding can be configured in IPv6. Each interface will create an ifname:ip6_forwarding /dev/ip6 variable that can be modified using ndd(1M). If a per-interface: ip6_forwarding variable is set to 0, packets will neither be forwarded from this interface to others, nor forwarded to this interface. Setting the ip6_forwarding variable will toggle all of the per-interface: ip6_forwarding variables to the setting of ip6_forwarding.

3.3.3 IPv6 TO IPv4 Tunneling

Before we start of with this discussion, it is worth mentioning that, all the networking related files, required at kernel level are available in the /usr/src/linux/Documentation/networking folder of any linux installation. There are more than one possibility to tunnel IPv6 packets over IPv4-only link. They are mentioned below:

Displaying the Existing tunnels

Using ip

/sbin/ip -6 tunnel show [<device>]

Using route

/sbin/route -A inet6

The output displayed is:

sit0: ipv6/ip remote any local any ttl 64 nopmtudisc

Kernel IPv6 routing table

Destination	Next Hop	Flags	Metric	Ref	Use	Iface
::1/128	::	U	0	0	0	lo
fe80::280:adff:fe83:559/128	::	U	0	4	1	lo
fe80::/10	::	UA	256	0	0	eth0
ff00::/8	::	UA	256	0	0	eth0
::/0	::	UDA	256	0	0	eth0

Here lo and etho are the available network interfaces.

Adding a Tunnel (Point to Point)

/sbin/ip tunnel add sit1 mode sit ttl 64 remote 222.222.3.6 – creates a tunnel

But, this can throw error:

ioctl: No buffer space available, this may be thrown if the tunnel sit1 already exists. Therefore check using ifconfig -a option (...). The optput displayed, after this is:

eth0 Link encap:Ethernet HWaddr 00:80:AD:83:05:59

inet addr:222.222.3.5 Bcast:222.222.255.255 Mask:255.255.0.0

inet6 addr: fe80::280:adff:fe83:559/10 Scope:Link

```
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:419000 errors:0 dropped:0 overruns:0 frame:0
        TX packets:2686 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:100
        RX bytes:45657975 (43.5 Mb)  TX bytes:336243 (328.3 Kb)
        Interrupt:3 Base address:0xd800

lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING  MTU:16436  Metric:1
        RX packets:152 errors:0 dropped:0 overruns:0 frame:0
        TX packets:152 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:15381 (15.0 Kb)  TX bytes:15381 (15.0 Kb)
sit0    Link encap:IPv6-in-IPv4
        NOARP  MTU:1480  Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
sit1    Link encap:IPv6-in-IPv4
        POINTOPOINT NOARP  MTU:1480  Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

Creating tunnels using "ifconfig" and "route" (deprecated)

This not very recommended way to add a tunnel because it's a little bit strange. No problem if adding only one, but if you setup more than one, you cannot easy shutdown the first ones and leave the others running.

Removing a tunnel using "ip"

Usage for removing a tunnel device:

/sbin/ip tunnel del <device>

Configuration of kernel parameters at runtime

The sysctl system call is used to modify kernel parameters at runtime. The parameters available are those listed under /proc/sys/. You can use sysctl(8) to both read and write sysctl data. The location of the various networking related configuration files are at: /proc/sys/net/ipv6/

For example, let us take the example of forwarding related details. This enables global IPv6 forwarding between all interfaces. In IPv6 you can't control forwarding per device, forwarding control has to be done using IPv6-netfilter (controlled with ip6tables) rulesets and specify input and output devices. This is different to IPv4, where you are able to control forwarding per device (decision is made on interface where packet came in). This also sets all interfaces' Host/Router setting 'forwarding' to the specified value. See below for details. This referred to as global forwarding. If this value is 0, no IPv6 forwarding is enabled, packets never leave another interface, neither physical nor logical like e.g. tunnels.

Retrieving the value of forwarding parameter:

sysctl net.ipv6.conf.all.forwarding

net.ipv6.conf.all.forwarding = 0

Setting a value

A new value can be set (if entry is writable):

sysctl -w net.ipv6.conf.all.forwarding=1

net.ipv6.conf.all.forwarding = 1

Test for IPv6 support of network configuration scripts

You can test, whether your Linux distribution contain support for persistent IPv6 configuration using the following command

/etc/sysconfig/network-scripts/network-functions-ipv6

Auto-magically test:
test -f /etc/sysconfig/network-scripts/network-unctions-ipv6 && echo "Main IPv6 script library exists"

Short hint for enabling IPv6 on current RHL 7.1, 7.2, 7.3, ...
Check whether running system has already IPv6 module loaded
modprobe -c | grep net-pf-10
alias net-pf-10 off

If result is "off", then enable IPv6 networking by editing /etc/sysconfig/network, add following new line
NETWORKING_IPV6=yes

Reboot or restart networking using
service network restart

Now IPv6 module should be loaded
modprobe -c | grep ipv6
alias net-pf-10 ipv6

Displaying neighbors using "ip"
With following command you can display the learnt or configured IPv6 neighbors
ip -6 neigh show [dev <device>]

3.3.4 Streaming with VideoLan's VLC

Streaming media is Multimedia, that is continuously received by, and normally displayed to, the Enad User while it is being delivered by the provider. When audio or video is streamed, a small buffer space is created on the user's computer, and data starts downloading into it. As soon as the buffer is full (usually just a matter of seconds), the file starts to play. As the file plays, it uses up information in the buffer, but while it is playing, more data is being

downloaded. As long as the data can be downloaded as fast as it is used up in playback, the file will play smoothly

3.3.4.1 Streaming with Ipv6 support on Windows

The file to be streamed can be from a disc, network or some capture device. The 'Stream/Save' needs to be checked first. The tool also supports various protocols like RTP, HTTP, MMSH, UDP. The address selected for unicasting or multicasting is al mentioned with the required port number. Under that Networking options, the Ipv6 option can be enabled, and also. Once the above options are specified, the streaming starts. The receiver side can be accordingly configured to receive IP stream with the help of source Ip address and port number. The file can be simply played or also buffered, to be saved at the same time on the receiver side

3.3.4.2 Streaming with Ipv6 support on RTLinux

This includes the following steps:
vlc -vvv videofile.mpeg –ipv6 –sout udp:\[ff08::1%eth0\] –ttl 12

Here videofile.mpeg is the file that is to be streamed.
Ff08::1 is either the Ipv6 address of the machine you need to unicast to, or the multicast address.
Here 12 is the TTL, (Time To Live), the stream will be able to cross 12 routers.
eth0 is the name of the output network interface.

3.3.4.3 Receive a stream from VLC

An active stream from VLC is received in the following way in Linux
vlc -vvv udp (unicast stream) OR
vlc -vvv udp:@239.255.12.42 (multicast stream), where 239.255.12.42 is the multicast address.

3.3.5.4 Saving the stream to the disk

VLC can save the stream to the disk. In order to do this, use the following command

vlc --sout file/ts:filestream.mpeg

Here ts stands for the format supported by VLC for MPEG2-TS format.

The filestream is the name of the file, the data is to be saved to.

3.3.5 Installation and configuration of Zebra

The installation of Zebra includes configuring and then installing the software from the tar file available from [].

Starting Zebra

```
# zebra –b zebra.conf
```

The next step is to verify whether the IP Forwarding option is set. This is done by the following command.

```
#cat /proc/sys/net/ipv4/ip_forward
```

If the result is 0 then the following is executed to enable the router with IP packet forwarding support:

echo 1 >> /proc/sys/net/ipv4/ip_forward

The configuration of various routing details on zebra can be doen through the VTY interface. VTY stands for Virtual TeletYpe interface. It means you can connect to the daemon via the telnet protocol.

To enable a VTY interface, you have to setup a VTY password. If there is no VTY password, one cannot connect to the VTY interface at all. The zebra username and password needs to be written into zebra.conf file. The file is to be copied to /usr/local/etc/ folder. This file is to be given as an input to zebra command, while invoking the server. This is done as follows:

root#zebra -b zebra.conf &

The zebra server is now running on port number 2600, hence a telnet onto this port will invoke the VTY interface for communication with the routing server.

```
root# telnet localhost 2601
    Trying 127.0.0.1...
    Connected to localhost.
    Escape character is '^]'.

    Hello, this is zebra (version 0.94)
    Copyright 1997-2000 Kunihiro Ishiguro

    User Access Verification
    Password: XXXXX
    Router> ?
      enable          Turn on privileged commands
      exit            Exit current mode and down to previous mode
      help            Description of the interactive help system
      list            Print command list
      show            Show running system information
      who             Display who is on a vty
    Router> enable
    Password: XXXXX
    Router# configure terminal
    Router(config)# interface eth0
    Router(config-if)# ip address 10.0.0.1/8
    Router(config-if)# ^Z
    Router#
```

The configuration options available from the router# prompt is similar to the one available with CISCO's Router.

Configuration of OSPF with IPv6 support in zebra

The Open Shortest Path First (OSPF) protocol is a link-state, hierarchical interior gateway protocol (IGP) for network routing. Dijkstra's algorithm is used to calculate the shortest path tree. It uses path cost as its routing metric. Path cost is determined generally by the speed (bandwidth) of the interface addressing the given route.

Starting and Stopping ripd

Copy /ZEBRA_HOME/ospf6d/ospf6d.conf.sample to /usr/local/etc/ospf6d.conf and execute the command,

root@priyanka zebra]# ospf6d -d ospf6

The command starts the OSPF protocol as the daemon process. To stop ospf6d, Please use kill `cat /var/run/ospf6d.pid`. Certain signals have special meaningless to ospf6d. The various options required to further configure OSPF6, is available with ZEBRA []

3.3.6 Installation and Configuration of XORP with IPv6 Stack

There is a single XORP process that manages the whole XORP router - this is called xorp_rtrmgr (short for XORP Router Manager). The xorp_rtrmgr will expect to find a configuration file in the same directory it is started from. By default this configuration file is called config.boot. Typically xorp_rtrmgr, must be run as root. This is because it starts up processes that need privileged access to insert routes into the forwarding path in the kernel.

A XORP router must be configured to perform the desired operations. The configuration information can be provided, by using the config.boot file or though XORP Shell.

Use a configuration file when the rtrmgr is started. The config.boot file needs to be used after tailoring the IP addresses, interfaces and routing protocols to match the required local network environment.

xorp_rtrmgr -b config.boot

Use the xorpsh command line interface after the rtrmgr is started. The xorpsh has to be in configurational mode:

user@hostname> configure
Entering configuration mode.
There are no other users in configuration mode.
[edit]
user@hostname#

On UNIX, only an user who belongs to group "xorp" can run xorpsh in configuration mode. It should be noted that command line completion in the xorpsh does greatly simplify configuration. The mixture of both methods is permissible. For example, a configuration file can also be loaded from within the xorpsh.

3.3.6.1 Network Interfaces

A XORP router will only use interfaces that it has been explicitly configured to use. Even for protocols such as BGP that are agnostic to interfaces, if the next-hop router for a routing entry is not through a configured interface routes the route will not be installed. For protocols that are explicitly aware of interfaces only configured interfaces will be used.

Every physical network device in the system is considered to be an "interface". Every interface can contain a number of virtual interfaces ("vif"s). In the majority of cases the interface name and vif name will be identical and will map to the name given to the interface by the operating system. A virtual interface is configured with the address or addresses that should be used. At each level in the configuration hierarchy ("interface", "vif" and "address") this part of the configuration is implicitly enabled.

```
interfaces {
    restore-original-config-on-shutdown: false
    interface eth0 {
        description: "data interface"
        disable: false
        /* default-system-config */
        vif eth0 {
            disable: false
            address ::10.10.10.10 {
                prefix-length: 24
                broadcast: 10.10.10.255
                disable: false
```

```
        }
    interface eth1 {
        description: "data interface"
        disable: false
        /* default-system-config */
        vif eth0 {
            disable: false
            address ::222.222.10.87 {
                prefix-length: 16
                broadcast: 222.222.255.255
                disable: false
            }
        }
    }
}
```

Note: Prior to XORP Release-1.1, the "enabled" flag was used instead of "disable".

It is recommended that you select the interfaces that you want to use on your system and configure them as above. If you are configuring an interface that is currently being used by the the system make sure that there is no mismatch in the "address", "prefix-length" and "broadcast" arguments. If the "default-system-config" statement is used, it instructs the FEA that the interface should be configured by using the existing interface information from the underlying system. In that case, the "vif's" and "addresses" must not be configured.

3.3.6.2 Forwarding Engine Abstraction

It is a requirement to explicitly enable forwarding for each protocol family.

```
fea {
    unicast-forwarding4 {
        disable: false
    }

    unicast-forwarding6 {
```

```
        disable: false
    }
}
```

If IPv4 and IPv6 forwarding are required you will require the configuration above.

3.3.6.3 Static Routes

This is the simplest routing protocol in XORP. It allows the installation of unicast or multicast static routes (either IPv4 or IPv6). Note that in case of multicast the routes are installed only in the user-level Multicast Routing Information Base and are used for multicast-specific reverse-path information by multicast routing protocols such as PIM-SM.

```
protocols {
   static {
       route ::10.10.10.0/24 {
          next-hop: ::10.10.10.10
          metric: 1
       }
       route ::222.222.10.0/16 {
          next-hop: ::222.222.10.87
          metric: 1
       }
     } //static
}//protocols
```

3.4 SUMMARY

The basic theme of the chapter is to cover details related to the architecture of the project work. The system comprises of one source where, multimedia files are stored. The operating platform identified for implementation of the source is a Real Time Linux operating system (RTLinux 3.1), which is configured to run on Linux Kernel (Fedora Core 6 with kernel 2.6.20). The multimedia file is streamed using VLC's VideoLAN software, from the source computer and passed through a group of XORP routers that are configured with Ipv6 support.

The configuration of the architecture was a quite a challenging experience, and required an in depth knowledge of Linux system intricacies.

4. QOS FOR MULTIMEDIA TRAFFIC IN IPV6

4.1 INTRODUCTION

Quality of Service refers to the ability of the network, its elements and its providers to provide for a certain assurance and consistency to transport a users' traffic. QoS can be defined as "any mechanism that provides distinction of traffic types, which can be classified and administered differently throughout the network" [5]. The above definition can be explained further with a brief example. Consider an organization, which intends to have a remote videoconference over the Internet with its remote branch situated at a different geographical location. The nature of this traffic begin real-time intolerant requires that the network it passes through, transports the traffic with limited or no delay and jitter. Also the network transporting this traffic needs to control all the packet loss parameters in order to ensure that none of the packets in the transmission are lost. The network also could ensure that in the case where the bandwidth is scarce, the network has to reserve enough bandwidth for this traffic to pass through. QoS mechanisms allow the application or the user to request for a certain guarantee thereby classifying the packets of this transmission into a separate class or flow.

QoS mechanisms also allow the network administrator to administer and control the resources necessary for the successful transmission of these packets. Finally QoS mechanisms allow the intervening routers to process this request by reserving network resources such as bandwidth and controlling delay, jitter and packet loss.

4.1.1 Issues related to QoS

QoS has not been an area that has been widely understood and accepted with various individuals and organizations defining it to suit their requirement. "Quality of Service (QoS) is one of the most elusive, confounding, and confusing topics in data networking today" [5]. Efforts by organizations such as the IETF are underway to better define QoS and the underlying principles and

practices that govern this technology. Along with this QoS faces various issues. Various network protocols offering QoS support such as IP, ATM, MPLS operate in the Internet which give rise to the problem of choosing the correct protocol for a specific setup and the ability to integrate the various protocol to offer an end-to-end QoS support [6]. Various QoS architectures such as IntServ and DiffServ have been developed but there has not been a common consensus on which technology to use. Bandwidth is not infinite and introducing QoS in a bandwidth deprived network does not offer any solution. QoS does not increase the bandwidth but only aims at utilizing it as effectively as possible [6]. Reliable QoS measuring tools have not been developed which would provide the Service Provider and the customer to determine whether adequate QoS capability is being provided [5]. The IPPM charter of the IETF is currently working on providing a baseline for defining QoS measurements. Vendor support to QoS is mostly through integrated software that offers QoS support. This might relatively cause a performance drop due to the delay introduced in the processing and hence it is essential that QoS be supported at the hardware level too [6].

4.1.2 Parameters used to Measure Qos at the Network Layer

The traditional parameters used to measure Network QoS have been bandwidth, delay, buffer requirements, packet loss, latency and jitter. Bandwidth is also commonly known as throughput. The bandwidth of a link refers to the ability of the link to transfer data at a rate expressed commonly in bits per second. Higher bandwidth provides the link with a higher capacity to transport data at a faster rate. Bandwidth is an important factor in evaluating and providing for QoS since lower bandwidth links lead to congestion, packet loss and packet transmission delays when there is too much data to be transported on the link. Delays in packet transmission affect the quality of real-time traffic. Real time traffic can be categorized on Real-Time-Tolerant (RTT) and Real Time Intolerant (RTI) traffic. Live video feeds and multimedia traffic are examples of real-time intolerant traffic. Packets belonging to such traffic require minimal delay across networks. Real-time-tolerant traffic such as an audio streaming session can tolerate a little delay in transmission due to the buffering and reconstruction capabilities offered by intelligent end devices. To

assure adequate QoS, delay has to be kept minimal. Packet loss results due to various factors such as low bandwidth and congestion on links. Intelligent end devices are capable of reconstructing data using various algorithms. But extreme packet losses may result in the inability to reconstruct data. Also RTI applications are very sensitive to packet losses. Latency is defined as the time taken for a packet of data to move from the source to the destination. Latency is a combination of all delays accrued during the packet transmission such as the propagation delay, the transmission delay, the processing delay and other induced delays. Latency is commonly measured by the round-trip-time which is the time taken for a packet to travel from the source to the destination and back to the source. Latency can be introduced at various points in the path. While offering end-to-end QoS this is an important parameter that has to be considered. The QoS enabled devices have to negotiate such that the latency in-between various points along the path are kept minimal. Jitter is defined as the variation in the delay introduced between different packets of a single transmission. Jitter may be caused due to timing issues, bandwidth constraints, network congestion or a synchronization problem. End systems are capable of buffering data and handling delays in order to provide the packets in a synchronized format to the upper layers. Jitter introduces uncertainty and makes it difficult for the intelligent algorithms to buffer and reconstruct data. This is very important for RTI traffic. Jitter is hence considered an important parameter in evaluating QoS.

To provide for efficient QoS on the Internet the quality, performance, and reliability of Internet data delivery services has to be studied and known. Parameters have to be identified which allow us to study and measure the Internet. The IETF IPPM working group (IPPM-WG) is involved in defining specific metrics and procedures for accurately measuring and documenting these metrics2.

4.1.3 IPv4 Header Format

The Type of Service field in IPv4 is a key mechanism of the IP service and is used to indicate the quality of service desired by the user or an application. The IP packets can be marked with an abstract or generalized set of parameters

which can be used to guide the routers to indicate the choice of service that are provided by the networks forming the Internet. The service types are based on the factors of delay, throughput and reliability. The IPv4 header consists of an 8 bit Type of Service (TOS) field to provide for the classification of service. Originally, the first three bits in this field were used to set the precedence, the next three were used to signify delay, throughput and reliability parameters and the last 2 bits were reserved for future use. The packet format is as below:

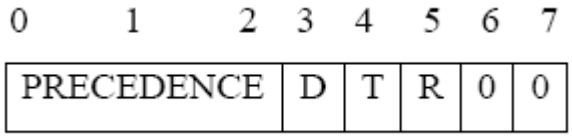

Fig 4.1: Ipv4 Packet format

Bits 0-2: Precedence.
Bit 3: 0 = Normal Delay, 1 = Low Delay.
Bit 4: 0 = Normal Throughput, 1 = High Throughput.
Bit 5: 0 = Normal Reliability, 1 = High Reliability.
Bits 6-7: Reserved for Future Use.

Precedence

111 - Network Control
110 - Internetwork Control
101 - CRITIC/ECP
100 - Flash Override
011 - Flash
010 - Immediate
001 - Priority
000 - Routine

Currently the 3 bit precedence field8 is ignored [Stevens99] and the next 4 bits are used to denote the QoS parameters necessary to classify the packets into different classes. The current implementation uses bits 3 through 6 to set values for delay, throughput, reliability, monetary cost and normal service.

The semantics for the 4 bits is as follows:

1000 -- minimize delay

0100 -- maximize throughput

0010 -- maximize reliability

0001 -- minimize monetary cost

0000 -- normal service

4.1.4 IPv6 Protocol Header

Consider the figure below to illustrate the Ipv6 header format

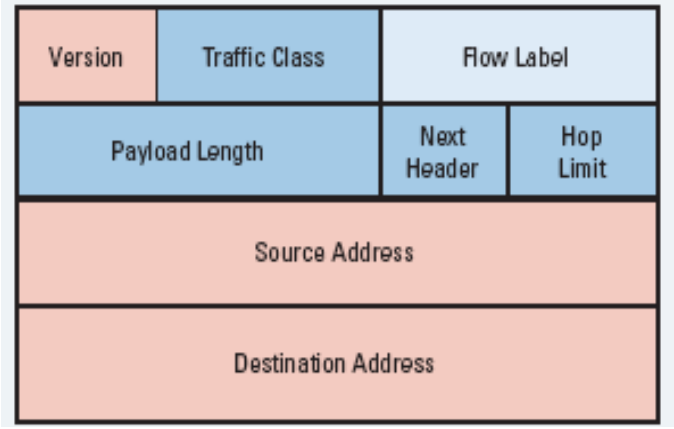

Fig 4.2: Ipv6 Protocol Header

The IPv6 header consists of an 8 bit Traffic Class field which allows for similar functionality as the IPv4 TOS field. The Traffic Class field in IPv6 provides for originating nodes to classify packets into different classes or priorities based on the QoS requirements. It also enables the forwarding routers to identify packets belonging to different classes. [RFC2460] lists a few general requirements that need to be applied to the Traffic Class field.

The IPv6 header also defines a 20 bit field called the Flow Label field which is to be used for providing QoS in IPv6. The flow label field enables an IPv6 enabled host to label a sequence of packets for which the host requests special handling by the IPv6 routers [RFC2460]. This enables the host to request non-default quality of service from the IPv6 network.

4.1.5 IPv6 Flow Label Requirements

The RFC 3697 specifies the flow labeling requirements for the IPv6 flow label field. A summary of the specification as listed in [RFC3697] is give below.

Flows should be unique for different transport connections and application data streams in order for proper flow label based classification. A source node which does not assign traffic to flows must set the Flow Label to zero. Application and transport layer protocols should be able to specify Flow Label values so that they can define what packets constitute a flow. The source node should be able to select unused Flow Label values for flows not requesting a specific value to be used. A source node must ensure that it does not unintentionally reuse Flow Label values it is currently using or has recently used when creating new flows

Flow Label values previously used with a specific pair of source and destination addresses must not be assigned to new flows with the same address pair within the flow state lifetime of 120 seconds. Accidental Flow Label value reuse must be avoided by providing for sequential or pseudo-random generation of new flow values. The method for flow state establishment must provide the means for flow state cleanup from the IPv6 nodes providing the flow-specific treatment. Flow state establishment methods must be able to recover from the case where the requested flow state cannot be supported

4.1.6 Bandwidth, delay and buffer requirements approach to using flow labels

Several Approaches to modifying the flow labels has been defined. The above is just one of them. This approach lists the important QoS parameters as Bandwidth, Delay or Latency, Jitter, Packet Loss and Buffer Requirements. Jitter and Packet Loss values are desired to be a minimum by any application and hence need not be defined in the flow label. It suggests the use of the Hop-by-Hop extension header as an alternative to specify these values. There are three parameters, to be used to define the value of the flow label field. They are, Bandwidth (to be expressed in multiples of kbps), Delay (to be expressed

in nanoseconds) and Buffer requirements (to be expressed in bytes)

The first bit of the remaining 17 bits of the flow label field to be used to differentiate between Soft Real Time (RTT - Real Time Tolerant) and Hard Real Time applications (RTI - Real Time Intolerant) applications.

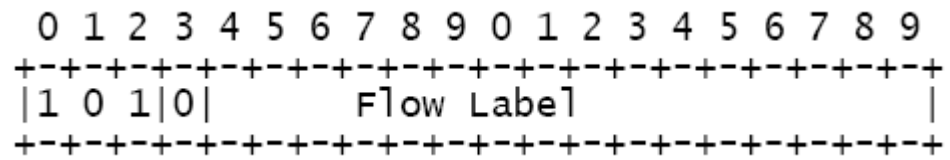

Fig 4.3: Soft Real Time Application Traffic

This implementation sets the fourth bit in the flow label to 0 to indicate average bandwidth requirement and intermediate end-to-end delay for an arbitrary packet. This implementation for soft real time applications indicates that the application can manage even if QoS requirements are not strictly met.

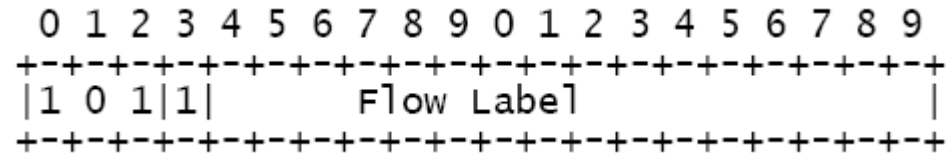

Figure 4.4: Hard Real Time Application Traffic

This implementation sets the fourth bit in the flow label to 1 to demand minimal latency and jitter and indicate that any delay is unacceptable for hard real time application traffic. The implementation proposes that the applications can decrease delay by increasing demands for bandwidth. In this scenario the minimum or maximum values specified in the Flow Label have to be exactly met.

The last 16 bits of the 20 bit flow label field are used to denote parameters such as Bandwidth, Buffer and Delay requirements. Bandwidth requirements are indicated by using the 6 bits of the remaining 16 bits. The next 5 bits are used to indicate the Buffer requirements and the final 5 bits are used to indicate the delay requirements. This approach supports DiffServ based model where applications are able to provide exact values of Bandwidth, Delay and

Buffer requirements.

4.2 APPROACHES OF IMPLEMENTATION

The above section discussed various intricacies, which played a key role in selecting the implementation strategy for the implementation of Quality of Service based scheduling of Multimedia traffic in real time environment. Three implementation strategies were selected, to suit the kind of experimental setup that was created, especially the inclusion of Software based IP routers. The first approach is the Diffserv aware video streaming, which is an implementation, to mark the multimedia steams generated through VLC's VideoLAN software. A multimedia stream parser needs to be created, by extending the VideoLAN's internal library. The other strategy is using the 20 bit FlowLabel field of the IPv6 header. This module captures the multimedia packets at the network interface of the source machine, where video streaming is being conducted and marks the Flow label field as per the bandwidth, delay and buffer requirements approach discussed above. The third approach is for real time kernel diffserv implementation. This satisfies the basic goal of scheduling multimedia traffic as per the goal of the project work. The module is implemented successfully. The module is inserted at every node, that is participating in the routing of the multimedia traffic. It works at the real time kernel level. Let us discuss each of the approaches introduced above in detail in the following sections.

4.2.1 DiffServ aware video streaming

Differentiated Services (Diffserv) is architecture for providing different types or levels of service for network traffic. One key characteristic of Diffserv is that flows are aggregated in the network, so that core routers only need to distinguish a comparably small number of aggregated flows, even if those flows contain thousands or millions of individual flows.

Support for Differentiated Services on Linux is part of the more general Traffic Control architecture. But the kernel in its minimal form does not have this support. The kernel needs to be recompiled to include this support.

4.2.1.1 General

The term DiffServ-aware video streaming is used here to describe a type of streaming system characterized by the integration of two main components: a DiffServ-enabled IP network and a DiffServ-aware video streaming [7] application. The general idea behind this concept is that performance may be improved if applications are aware (make use) of the enhanced network capabilities offered by the DiffServ architecture. Thus, a DiffServ-aware application marks its packets in order for them to take advantage of the advanced rate, delay, jitter and/or loss features offered by the network. This approach is called application-driven (or application-controlled) marking, and is different from the typical DiffServ router-based marking which is done by edge routers based on rate-metering or flow identification.

The compressed streams generated by video codecs are usually organized in a hierarchical structure. For example, at the frame level, an MPEG-compressed video stream contains three types of frames: I, P and B. The loss of each type of frame has a different impact on the visual quality of the received signal. The loss of an I frame has a bigger impact than that of a B frame. If an I frame is lost, in addition to the loss of the frame itself, some neighboring frames can also be considered lost for practical reasons. This is due to the fact that they cannot be properly decoded even if they are correctly received, because the reference information from the lost I frame is not available. One can speak then of error propagation.

In the case of layered coding, a video stream is made up of one base layer and one or more enhancement layers. The base layer represents the video sequence with the lowest quality but also the lowest required bit-rate. Each successive enhancement layer adds up quality and bit-rate. In a multicast environment, clients with low access rates may ask only for the base layer, while clients with higher access rates may additionally ask for as many available enhancements layers as their links can accept. In any of the cases above, a DiffServ-aware video application marks the outgoing packets taking

into account the type of video information they carry. This way, the differentiated treatment inside the DiffServ network should yield a better visual quality by assuring a particular rate, limiting delay and jitter, or discarding first less important packets when congestion arises. This type of marking has also been called semantic marking. Note that when we speak of DiffServ-aware streaming, we refer to methods in which a video stream is somehow separated in "segments" (layers, frame types, partitions, etc.) that are marked and forwarded using different DiffServ classes or drop priorities. This means that we do not regard as DiffServ-aware streaming the case in which a complete video flow is isolated and transported in a single class (like a premium class, for instance).

4.2.1.2 Mapping strategy and Architecture

The figure below shows the DiffServ mapping strategy behind the implemented IPv6 DiffServ-aware video streaming. It's a straightforward mapping that consists of forwarding an MPEG-coded stream in a single three-precedence AF class where the discard priorities (the *colors*) of each packet are set according to the type of frame they carry. This way, packets carrying I frames are marked green, packets carrying P frames are marked yellow and packets carrying B frames are marked red. The goal of this mapping strategy is to protect the reference frames, highly relevant for quality, under heavy network congestion. The prioritized discard of the AF network (enforced by Active Queue Management in the routers) should assure that packets carrying I frames are the last to be discarded if congestion builds up.

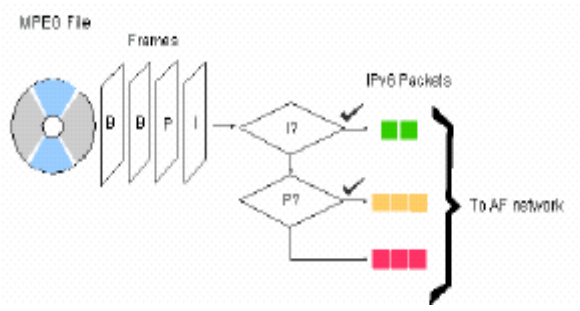

Fig 4.5: Straightforward AF mapping strategy for MPEG streams

Based on the general mapping strategy described above, the architecture for IPv6 DiffServ-aware streaming would look like the one shown in Figure 14: a video client-server system that sends the MPEG source-marked stream through an IPv6 AF network.

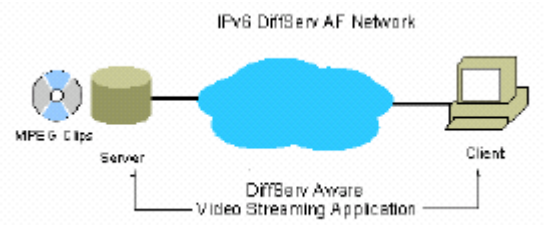

Figure 4.6: IPv6 DiffServ-aware streaming system

The VideoLAN streaming system for the implementation of the DiffServ-aware features because it is free, open source, multi-platform and is already IPv6-enabled. The DS-aware features are now incorporated to the VideoLAN's VLC software. The above implementation requires a MPEG-2 Transport Stream (TS) Parser, which should the following: identify the frame type carried in the transport packets and assign them the corresponding priority. This parser is under development.

4.2.2 Ipv6 and flow label usage

The Flow Label is a 20 bit field in the IPv6 header that has been planned to operate for per flow QoS treatment. General rules for the Flow Label field were proposed recently in RFC 3697. The RFC 3595 defines textual conventions to represent the Flow Label field. Actually the flow label tries to integrate the classic Diffserv operation where traffic is aggregated into classes with the flow establishment. Therefore RFC 3697 defines that the 20-bit Flow Label field is used by a source to label packets of a flow and the zero value is used to indicate packets that are not part of any flow. Packet classifiers use the triplet of Flow Label, Source Address, and Destination Address fields to identify which flow a particular packet belongs to. Packets are processed in a flow-specific manner by the nodes (routers) that have been set up with that flow-specific state. In all cases the Flow Label value set by the source must be delivered

unchanged to the destination node.

If an IPv6 node is not providing flow-specific treatment, it must ignore the field when receiving or forwarding a packet. Each established flow should expire when the flow is idle in order to increase router performance. Therefore, the RFC defines that nodes should not assume that packets arriving 120 seconds or more after the previous packet of a flow still belong to the same flow (unless a flow state establishment method defines a longer flow state lifetime or the flow state has been explicitly refreshed). Also, this flow expiration allows the re-use of Flow Label values. Flow Label values previously used with a specific pair of source and destination addresses must not be assigned to new flows with the same address pair within 120 seconds of the termination of the previous flow. Finally, to avoid accidental Flow Label value reuse, the source node should select new Flow Label values in a well-defined sequence (e.g., sequential or pseudo-random) and use an initial value that avoids reuse of recently used Flow Label values each time the system restarts. In addition, a major issue at the deployment of QoS services that uses the Flow Label field is the security aspect. In particular, it is crucial to avoid theft of service attacks by unauthorized traffic. Such attacks are possible in two ways: by spoofing the Flow Label value (only on valid nodes that uses the correct source address) or by spoofing the whole 3-tuple (Flow Label, Source Address, Destination Address). The latter can be done in an intermediate router or in a host that is not subject to ingress filtering. Also, we should notice that IPsec protocol does not include the IPv6 header's Flow Label in any of its cryptographic calculations (in the case of tunnel mode, it is the outer IPv6 header's Flow Label that is not included). As a consequence IPsec does not provide any defense against an adversary's modification of the Flow Label. Finally it is recommended that applications with an appropriate privilege in a sending host should be entitled to set a non zero Flow Label. But this is an issue that depends on the host’s operating system or on the used policy and authorization mechanisms.

The implementation of the concept was made using the libpcap library, and several network programming based header files like ipv6.h. The basic process

of implementation, was identification of the network interfaces status, and reading packets from them. The packets are read in two parts, ie header and payload. The header is identified using the ipv6.h file and values set for the 20 bit Flow label.

4.2.3 Real Time linux kernel DiffServ Implementation

The key characteristic of DiffServ architecture is that, flows are aggregated in the network, so that core routers only need to distinguish a comparably small number of aggregated flows, even if those flows contain thousands or millions of individual flows. Support for Differentiated Services in Linux kernel, is part of the more general Traffic Control architecture. But the kernel in its minimal form does not have this support. The kernel needs to be recompiled to include this support. This will be covered in the following sections

The next chapter describes how new control functions can be added to the Linux kernel traffic. The implementation captures the multimedia packets received at the network interface and categorizes their flow as per the traffic class parameters. This parameter is read from the Ipv6 header's Traffic class field. At present it is assumed, that the UDP traffic coming at the network interface is the multimedia traffic and it is prioritized at the output network interface. The multimedia stream is released into the network. The module that is created for the above implementation is a kernel module, inserted in the Real Time kernel. The module is invoked whenever there is a packet at the network interface. A heavy TCP traffic is also created using the Mgen tool. The description regarding this tool will be given in the coming chapter. The kernel module described above, only marks the UDP traffic from a particular source, ie the VOD Server, with DSCP value = 16. This signifies the highest priority. The DSCP value is also set while streaming the multimedia file from the VLC Server. The option is specified, when VLC is invoked from the command line parameter. Note that this facility is available in the latest version of the Video LAN software ie. 0.9.0-svn-20070112-0001. The TCP traffic is handled with low priority; marked by DSCP value of zero, and may be queued for longer times, till a *resent* routine is called to clear all the logged TCP traffic.

4.3 SUMMARY

The major focus of the chapter is on the approaches that were actually implemented for QoS based scheduling of multimedia traffic. However, the initial focus was also on the IPv header's 8 bit Traffic class field and 20 bit Flow Label field, that is required for attaching various QoS parameters with multimedia IP traffic. The first technique of implementation was related to DissServ aware Video streaming that dealt with marking the I-Frame of multimedia stream with highest priority. The next technique was related to implementation of a module at the Real Time Linux Kernel level for scheduling the IP traffic based on their values in TOS (Ipv4), TC or Flow Label(IPv6) field.

5. REAL TIME LINUX KERNEL DIFFSERV IMPLEMENTATION

5.1 INTRODUCTION

Differentiated Services (DiffServ) is architecture for providing different types or levels of service for network traffic. One key characteristic of DiffServ is that flows are aggregated in the network, so that core routers only need to distinguish a comparably small number of aggregated flows, even if those flows contain thousands or millions of individual flows.

Support for Differentiated Services on Linux is part of the more general Traffic Control architecture. But the kernel in its minimal form does not have this support. The kernel needs to be recompiled to include this support. This will be covered in the following sections

This chapter describes how new control functions can be added to the Linux kernel traffic. The implementation captures the multimedia packets received at the network interface and categorizes their flow as per the traffic class parameters. This parameter is read from the Ipv6 header's Traffic class field. At present it is assumed, that the UDP traffic coming at the network interface is the multimedia traffic and it is prioritized at the output network interface. The multimedia stream is released into the network. The module that is created for the above implementation is a kernel module, inserted in the Real Time kernel. The module is invoked whenever there is a packet at the network interface. A heavy TCP traffic is also created using the Mgen tool. The description regarding this tool will be given in the coming chapter. The kernel module described above, only marks the UDP traffic with DSCP value = "16", with highest priority. The DSCP value is set while streaming the multimedia file from the VLC Server. The option can be specified, when VLC is invoked from the command line parameter. Note that this facility is available in the latest version of the Video LAN software ie. 0.9.0-svn-20070112-0001. The TCP traffic is handled with low priority and may be queued for longer times, till a resent routine is called to clear all the logged TCP traffic.

The Figure 5.1 shown below, gives a broader picture of, how the kernel processes data received from the network and how it generates new data to be sent on the network: incoming packets are examined and then either directly forwarded to the network (eg, on a different interface, if the machine is acting as a router or a bridge), or they are passed up to higher layers in the protocol stack (e.g) to a transport protocol like UDP or TCP) for further processing. Those higher layers may also generate data on their own and hand it to the lower layers for tasks like encapsulation, routing and eventually transmission.

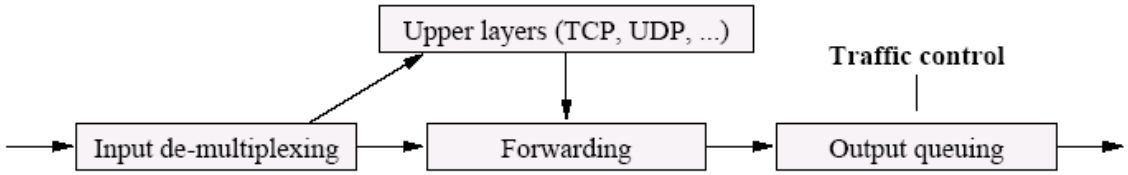

Fig 5.1: Processing of network data

Forwarding includes the selection of the output interface, the selection of the next hop, encapsulation, etc. Once all this is done, packets are queued on the respective output interfaces. This is the point where traffic control comes into play. Traffic control can, among other things, decide if packets are queued or if they are dropped, it can send the packets in a specific order, it can delay the sending of the packets. Once the packet is released, the network device driver picks it up and emits it on the network.

5.2 ENABLING QOS SUPPORT IN THE LINUX KERNEL

The steps required for kernel compilation have been covered in Chapter2. Apart from the options selected before, the kernel compilation should also include the options for setting Quality of Service related details. The following kernel configuration options have to be enabled in the **Networking options**[8] section:

Table 5.1: Networking options during Kernel configuration

Sr. No.	Options
1	Kernel/User netlink socket (CONFIG_NETLINK)
2	Network packet filtering (CONFIG_NETFILTER)
3	QoS and/or fair queueing (CONFIG_NET_SCHED)

The following kernel configuration options should be enabled in the section Networking options, QoS and/or fair queuing:

Table 5.2: QoS and fair Queuing Options during kernel configuration

Sr. No.	Options
1	CBQ packet scheduler (CONFIG_NET_SCH_CBQ
2	The simplest PRIO pseudoscheduler (CONFIG_NET_SCH_PRIO)
3	RED queue (CONFIG_NET_SCH_RED)
4	GRED queue (CONFIG_NET_SCH_GRED)
5	Diffserv field marker (CONFIG_NET_SCH_DSMARK)
6	Ingress Qdisc (CONFIG_NET_SCH_INGRESS)
7	QoS support (CONFIG_NET_QOS
8	Packet classifier API (CONFIG_NET_CLS
9	TC index classifier (CONFIG_NET_CLS_TCINDEX)
10	Firewall based classifier (CONFIG_NET_CLS_FW)
11	U32 classifier (CONFIG_NET_CLS_U32)
12	Traffic policing (CONFIG_NET_CLS_POLICE)

5.3 INSTALLING TRAFFIC CONTROL (TC) UTILITY OF IPROUTE2

Linux Traffic Control is configured with the tc utility, which comes with iproute2 package. Fedora Core 6 kernel includes the iproute2 package. But in Red Hat Linux the package can be installed by first extracting the iproute2 package and then editing the iproute2/config file to enable DiffServ. i.e. tar -xvzf iproute2-

current.tar.gz, followed by TC_CONFIG_DIFFSERV=y. Finally run make command in iproute2 folder.

5.4 DIFFSERV IMPLEMENTATION OVERVIEW

The traffic control code in the Linux kernel consists of the four major parts, Queuing disciplines, Classes, Filters and Policing. Each network device has a queuing discipline associated with it, which controls how packets queued on that device are treated. The simple queuing discipline is FIFO. But it could as well use filters to distinguish among different classes of packets and process each class in a specific way, e.g. by giving one class priority over other classes. Figure 5.2 gives example of such a queuing discipline. It is possible that multiple filters map onto the same class.

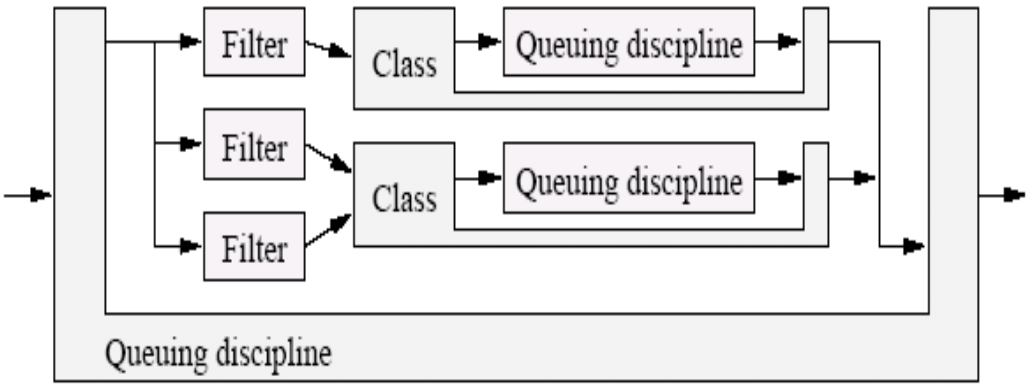

Fig 5.2: A simple queuing discipline with multiple classes.

Filters can be combined arbitrarily with queuing disciplines and classes as long as the queuing discipline has classes at all. The figure, Fig 5.3 shows a stack: first, there is queuing discipline with two delay priorities. Packets, which are selected by the filter, go to the 'high' priority class, while all other packets go to the 'low' priority class. The sch_prio queue works this way.

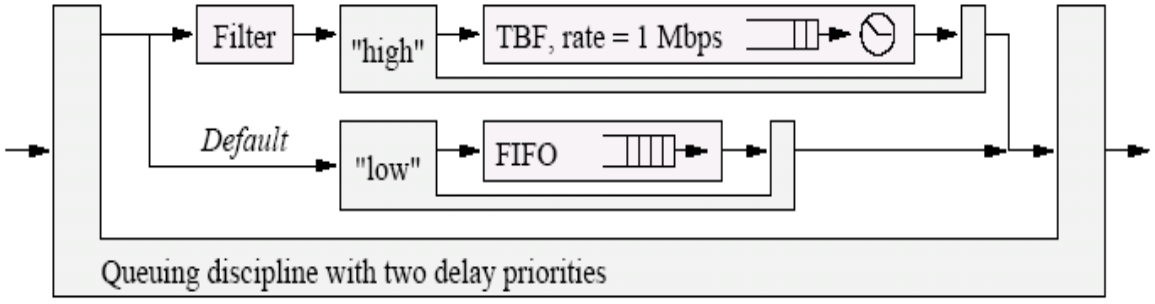

Fig 5.3: Combination of priority, TBF and FIFO queuing disciplines.

Packets are enqueued as follows: when the enqueue function of a queuing discipline is called, it runs one filter after the other until one of them indicates a match. It then queues, the packet for the corresponding class, which usually means to invoke the enqueue function of the queuing discipline “owned” by that class. Packets that which do not match any of the filters are typically attributed to some default class.

5.5 RESOURCES

The tc is a user space program used to manipulate individual traffic control elements. Its source is in the iproute2/tc package. The kernel code resides in the net/sched/folder. The interfaces between kernel traffic control [9] elements and the user space programs using them are declared in include/linux/pgk_cls.h and include/linux/pkt_sched.h. Declarations used only inside the kernel and the definitions of some inline functions can be found in include/net/pkt_cls.h and include/net/pkt_sched.h

The rtnetlink mechanism used for communication between traffic control elements in user space and in the kernel is implemented in net/core/rtnetlink.c and include/linux/rtnetlink.h. The rtnetlink is based on netlink, which can be found in net/netlink/ and include/linux/netlink.h

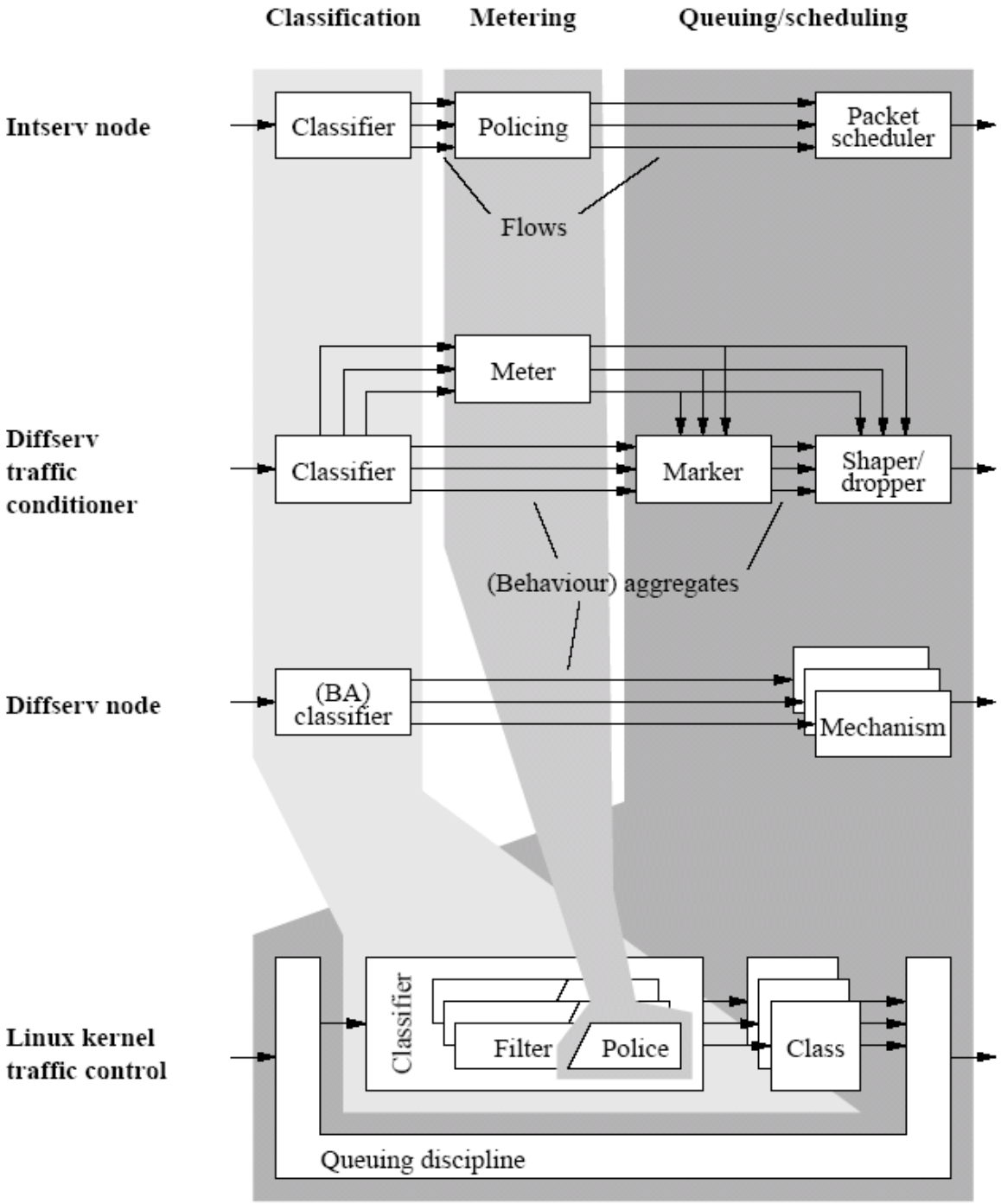

Fig 5.4: Relation of elements of the intserv and diffserv architecture to traffic control in the Linux kernel.

The Figure 5.4 describes the architectural models of IntServ and Diffserv, and how the elements of Linux Traffic Control systems are related to them. The classes play an ambivalent role, because they determine the final outcome of a

classification and they can also be part of the mechanism that implements a certain queuing or scheduling behavior.

5.6 QUEUEING DISCIPLINE

Each queuing discipline, provides the following set of functions to control its operations. The structure Qdisc_ops defines a queuing discipline. It is located in /net/pkt_sched.h file. The structure is displayed below:

```
struct Qdisc_ops
{
struct Qdisc_ops        *next;
struct Qdisc_class_ops  *cl_ops;
char                    id[IFNAMSIZ];
int                     priv_size;
int                     (*enqueue)(struct sk_buff *, struct Qdisc *);
struct sk_buff *        (*dequeue)(struct Qdisc *);
int                     (*requeue)(struct sk_buff *, struct Qdisc *);
unsigned int            (*drop)(struct Qdisc *);
int                     (*init)(struct Qdisc *, struct rtattr *arg);
void                    (*reset)(struct Qdisc *);
void                    (*destroy)(struct Qdisc *);
int                     (*change)(struct Qdisc *, struct rtattr *arg);
int                     (*dump)(struct Qdisc *, struct sk_buff *);
int                     (*dump_stats)(struct Qdisc *, struct gnet_dump *);
struct module           *owner;
};
```

The main components of the structure are:

1. init: initializes the queuing discipline. The queuing displines are usually referenced by a pointer to the corresponding struct Qdisc.
2. change: changes the configuration of a queuing discipline

3. destroy: removes a queuing discipline.
4. enqueue: enqueues a packet with the queuing discipline. It removes all classes and possible also all filters, cancels all pending events and returns all resources held by the queuing discipline.
5. dequeue: returns the next packet eligible for sending. It returns NULL, when there are no packets to be sent.
6. requeue: puts a packet back into the queue.
7. drop: drops one packet from the queue

When a packet is enqueued on an interface (dev-queue_xmit in net/core/dev.c), the enqueue function of the device's queuing discipline (field qdisc of struct device in include/linux/netdevice.c) is invoked. Afterwards, dev_queue_xmit calls qdisc_wakeup in include/net/pkt_sched.h on that device to try sending the packet that was just enqueued.

The qdisc_wakeup immediately calls qdisc_restart in net/sched/sch_generic.c, which is the main function to poll queuing disciplines and to send packets. Qdisc_restart first tries to obtain a packet from the queuing discipline of the device, and if it succeeds, it invokes the device's hard_start_xmit function to actually send the packet. If sending fails for some reason, then the packets are retuned back to the queuing discipline by the requeue function.

It is worth noting that queuing disciplines never make call to delivery functions. Rather, they have to wait until they are polled. The queuing discipline has to be registered using register-qdisc, from the init-module function, which is invoked from the kernel space. This can be done since the queuing discipline is developed as a kernel module. This kernel module can be inserted in the Real Time Linux kernel, and it gets activated whenever a packet is sniffed from the User Space program.

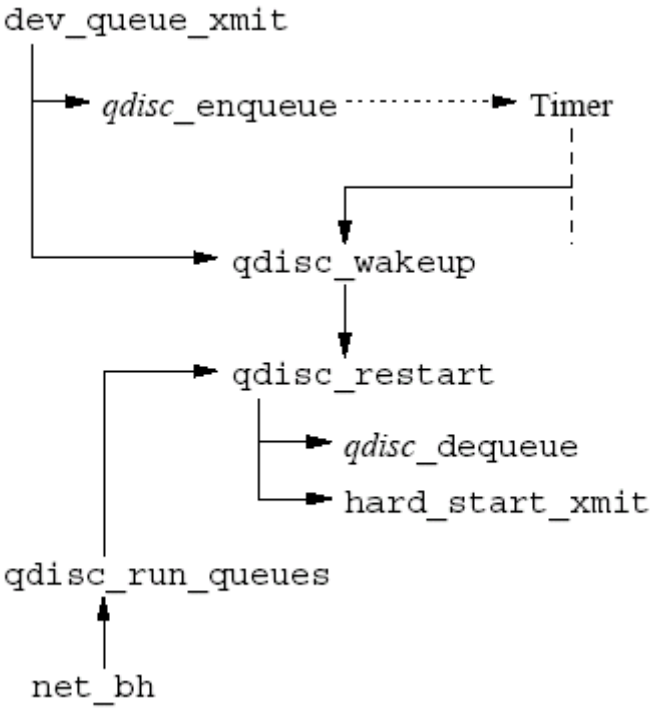

Fig 5.5: Flow chart showing the functions called when enqueuing and sending packets.

The include/linux/rtnetlink.h has declarations of the following types:

```
#define RTPROT_ZEBRA    11      /* Zebra */
#define RTPROT_XORP     14      /* XORP */
#define RTPROT_STATIC   4       /* Route installed by administrator
```

Ideally, the values of protocol >= RTPROTO_STATIC are not interpreted by kernel. But when the routed is disabled, inorder to route packets manually, the value is read, to send packets to XORP or ZEBRA routing daemons. The rtnetlink.h [10] file also contains, a structure (struct rtattr*) to control vector of options, that are passed while creating or changing an instance of queuing disciplines. Each value is encoded with its type, the length of the value and the value(eg zero or other data types). Option types and the data structures used for values are declared in include/linux/pkt_sched.h. The option vector is parsed by calling rtattr_parse, which returns an array of pointers to the individual elements, indexed by the option type. The length and the content of the option can be accessed via the macros RTA_PAYLOAD and RTA_DATA,

respectively.

In the realtime kernel, the option vectors are passed between user-space programs and the kernel using the rtnetlink mechanism. The synopsis of the rtnetlink method is as follows:

#include<asm/types.h>
#include<linux/netlink.h>
#include<linux/rtnetlink.h>
#include <sys/socket.h>
rtnetlink_socket = socket(PF_NETLINK, int socket_type, NETLINK_ROUTE);

Here, the first argument specifies that the address family is netlink. The second argument speciefies that raw network protocol access is being sought. The third argument contains information related to the type of netlink socket.activity. It can contain the following values

1. NETLINK_ROUTE Modify most of IPv4 control parameters
2. NETLINK_FIREWALL Receive packets sent by IPv4 firewall
3. NETLINK_ARPD Manage ARP table from user space
4. NETLINK_ROUTE6 IPv6 routing table update
5. NETLINK_IP6_FW Currently not implemented
6. NETLINK_TAPBASE Ethertap device
7. NETLINK_SKIP Reserved for Enskip

The value used in the implementation is NETLINK_ROUTE6 for Ipv6 traffic support. The rtnetlink utility allows the kernel's routing tables to be read and altered. Network routes, ip addresses, link parameters, neighbour setups, queueing disciplines, traffic classes and packet classifiers are controlled through NETLINK_ROUTE sockets. It is based on netlink messages.

5.7 CLASSES

Queuing disciplines with classes provide the following set of functions to manipulate classes (ie struct Qdisc_class_ops in include/net/pkt_sched.h).

Classes can be identified in two ways: (1) by the class ID, which is assigned by user and (2) by the internal ID, which is assigned by the queuing discipline. The internal Id is of type unsigned long, whereas class ID is of type u32 [11]. The implementation uses internal Id, because that is the way by which kernel identifies classes internally.

Two classes have been identified for the implementation. The first one is the IP Premium Traffic, ie for the UDP Traffic, or the IP traffic, in which the DSCP value is 16. This value is set while streaming the Multimedia traffic through VLC's VideoLAN software. The other traffic is represented by the Best Effort Class, which has a DSCP value of zero. This is otherwise the default value, of a non prioritized IP traffic. The Best Effort traffic is the TCP traffic created through Mgen Traffic generator tool.

The traffic can be prioritized by the specifications in the Class Id, if skb -> priority (struct sk_buff in include/linux/skbuff.h)is equal to a particular ClassId. If this is not true, the skb -> priority is set to skb -> priority (struct sock in include/linux/skbuff.h). The sk -> priority can be set with the SO_PRIORITY socket option (sock_setsockopt in net/core/sock.c). The SO_PRIORITY can be set to a number greater than 10, to mark a relatively higher PRIORITY. In this case the value used is 16. After this the enqueue function of the respective inner queuing discipline is invoked.

5.8 MODULE INVOCATION FROM REAL TIME KERNEL

The most common communication model between a Real Time Code/kernel code and User Space code is FIFO. A process on one end writes to a FIFO, which appears as a normal file, while another one reads from the other end. With RTLinux, the reader might be a real-time process, while the writer is a user-space program shuttling directives to the real-time code through the FIFO or vice versa. In either case, the FIFO devices are normal character devices (/dev/rtf*), and both ends can interact with the device through normal POSIX calls, such as open(), close(), read() and write().

In this implementation, the User Space code comprises of a program to read the configured network interface for outgoing packets. This is done using the pcap utility available in the pcap.h header file.

The moment there is a packet at the network interface, the information is passed onto the kernel module, which starts reading the processing the packets as mentioned above. This is a preferred technique for the implementation, because it is a wrong practice to keep the kernel module engaged in wait operations. Hence the work is divided between the User Space and Kernel Space for better execution and performance. The module is inserted at all the computers where XORP is running. The above implementation adds Priority based Scheduling or Class based Scheduling, which was otherwise lacking in XORP.

5.9 SUMMARY

The major focus of the chapter was description of the work related to implementation of priority based scheduling of multimedia traffic at the Linux kernel level. The implementation uses libraries of iproute2 package for traffic control mechanisms. The module is later inserted into the Real Time Linux kernel and invoked whenever there is a packet at the network interface.

6. WRITING A XORP PROCESS

6.1 INTRODUCTION

The software router is extensible in nature. It is implemented in C++ and can be extended to include desired facilities in the router. One such implementation has been written for the router to verify the incoming packets, for its required QoS fields marked Ipv6 header. If the packets require a high QoS, then they are buffered on the router for a very short time comparatively, and immediately sent to the best possible routes [11]. The routes are calculated dynamically and communicated to the RIB. This feature was already available with XORP in the static routes program. [12] But the logic for scheduling based on priority have been added. The implementation uses a basic characteristic of XORP routers, ie it is event driven.

A distinguishing factor between implementations of software router, is based on the fact, that a router is *event-driven* or it uses a periodic *route scanner* to resolve dependencies between routes. The scanner-based approach is simpler, but has a rather high latency before a route change actually takes effect. Cisco IOS and Zebra both use route scanners, with a significant latency cost; MRTD and BIRD are event-driven, but this is easier given a single monolithic process. In XORP, the decision that everything is event-driven is [13] fundamental and has been reflected in the design and implementation of all protocols, and of the IPC mechanism. Note that the router is running with certain initial configuration, which is coded in the config.boot file under rtrmgr folder of XORP distribution. Refer Appendix B for configuration made in the config.boot file.

6.2 CREATING A XORP PROCESS

The basic process of creating a XORP process includes creating XRL interface (.xif) file, parsing the XRL Interface specifications using clnt-gen and generating a header and related library source file to simplify invoking the XRL's of that interface. The next process is to create target interface (.tgt) file
And the generate a header and related library source file, using tgt-gen. This is

later used to simplify invoking the XRL's of that interface.

6.3 CREATING A XORP PROCESS FOR PRIORITY BASED ROUTING

The *static routes* program is a very simple XORP process. To a first approximation, it receives XRL configuration requests from the *xorp rtrmgr* to set up static routing entries, store the entries, and communicate them to the RIB using XRLs.

It exports an XRL interface to other processes (typically the xorp rtrmgr) and calls XRLs on the XRL interface of another XORP process (the RIB). The major segments of the code are the XRL interface of static routes, implementation of the XRL interface of static routes, coding the main loop of static routes and then calling XRLs on the RIB.

6.3.1 Creating XRL interfaces

An interface is declared using the interface keyword and a set of XRL methods associated with it. There are two types of interfaces defined for a XORP process, ie client interface and the target interface. The interfaces consist of XRLs. Each item in the XRL Method list consists of a method name, its dispatch arguments, and return arguments specified in the default XRL text form.

6.3.1.1 Client Interface

The client interface is saved as a .xif file and stored in xorp/xrl/interfaces. The interface is basically defined as:
interface <*interface name*>/<*interface version*> { . . . *xrl method list* . . . }
The code snippet showing the interface definition for the static routes process is shown below:

```
interface static_routes/0.1 {
enable_static_routes ? enable:bool
start_static_routes
```

stop_static_routes
add_route4 ? unicast:bool & multicast:bool & network:ipv4net & nexthop:ipv4 & metric:u32
add_route6 ? unicast:bool & multicast:bool & network:ipv6net & nexthop:ipv6 & metric:u32
}

Thus the first XRL in this file is:
static routes/0.1/enable static routes?enable:bool

When this XRL is actually called, it would look like:
finder://static routes/static routes/0.1/enable static routes?enable:bool=true

The finder part is on the target side. In the call, the first static routes indicates the name of the target process, and the second static routes is the name of the interface, taken from the XIF file. A process can support more than one interface, and an interface definition can be used by more than one process, hence the duplication in a process as simple as *static routes*.

Generating stub code for the caller

The clnt-gen (XRL Interface Client Generator) parses Xrl Interface specifications and generates a header and related library source file to simplify invoking the XRL's of that interface. Each XRL Interface specification generates an XRL Interface Client C++ class. The generated C++ class has a send method for each XRL in the XRL interface specification and takes care of argument marshalling when the XRL is dispatched and when it returns. The call required to invoke clnt-gen is mentioned in the Makefile.am (the automake Makefile) in xorp/xrl/interfaces

static_routes.xif is processed by clnt-gen to produce static routes xif.hh and static routes xif.cc, which are then compiled and linked into the library libstaticroutesxif.la. Any process that wants to call the *static routes* interface can link with this library.

6.3.1.2 Target Interface

The basic synopsis of the XRL targets is declared in *.tgt* files using:
target < target name > implements <*interface*> [, <*interface*> . . .]

Target specification files use the standard C include primitive to include the appropriate interface specifications. The file is saved with .tgt extension and stored in xorp/xrl/targets. A sample code snippet showing the Interface file is as follows:

```
#include "common.xif"
#include "test.xif"
target test implements common/0.1, test/1.0
```

Generating stub code for the target

A XORP process can implement more than one interface. In fact most XORP processes implement a special-purpose interface. The MakeFile.am includes the procedure to add static_routes.tgt to the list of tgt files. The tgt-gen (XRL Interface Target Generator) takes an XRL target specification and generates list of XRL's supported by the target and a class to be sub-classed that takes care of argument marshalling and unmarshalling and sanity checks.

From each .tgt file, a .xrls file will be generated using the python script tgt-gen. The libstaticroutesbase.la is a library, which is going to be used to link the *static routes* process, to get access to all the stub code to implement the target part of this interface. The libstaticroutesbase.la defines a class called XrlStaticRoutesTargetBase which will be used to receive XRL requests. The constructor for XrlStaticRoutesTargetBase takes a single parameter which is typically the XrlRouter for the process. An XrlRouter is an object that is bound to an *EventLoop* and which sends and receives XRL requests.

6.3.2 The main loop

The main loop of a XORP process includes a header file that includes

information like XORP_MODULE_NAME and XORP_MODULE_VERSION.

Then we include the functionality from libxorp that we'll need:

- libxorp/xorp.h: generic headers that should always be included.
- libxorp/xlog.h: XORP logging functionality. The convention is to use XLOG macros to log warnings and error messages, so we can redefine how logging if implemented in future without rewriting the code that uses logging. See Section 7 for more information about the XLOG facility.
- libxorp/debug.h: XORP debugging functionality.
- libxorp/callback.hh: XORP callback templates, needed to pass a handle into event handling code to be called later when an event occurs.
- libxorp/eventloop.hh: the main XORP eventloop.
- libxorp/exceptions.hh: standard exceptions for standard stuff - useful as a debugging aid.

The other main class is the static_routes_module.cc file. The most important function of the class is the static_routes_main method.

```
static void static_routes_main(const string& finder_hostname, uint16_t
finder_port)
{
    // Init stuff
    EventLoop eventloop;

    // StaticRoutes node
    XrlStaticRoutesNode xrl_static_routes_node(
        eventloop,
        "static_routes",
        finder_hostname,
        finder_port,
        "finder",
        "fea",
        "rib");
    wait_until_xrl_router_is_ready(eventloop,
```

```
xrl_static_routes_node.xrl_router());

    // Startup
    xrl_static_routes_node.startup();

    // Main loop
    while (! xrl_static_routes_node.is_done()) {
        eventloop.run();
    }
}
```

Then we create an instance of the XrlStaticRoutesNode class we defined earlier to receive XRLs on the static routesXRL target interface. Inside this object there will be the corresponding XrlStdRouter object for sending and receiving XRLs from this process. The following parameters are passed to the XrlStaticRoutesNode.

- The EventLoop object.
- The XRL target name of this process: in this case "static routes".
- Information about the host and port where the XRL finder is located.
- Information about the names of other XRL targets we need to communicate with: the Finder, the FEA, and the RIB.

Before we proceed any further, we must give the XrlStdRouter time to register our existence with the Finder. Thus we call wait until XRL router is ready().

6.3.3 CALLING XRLS ON THE RIB

The *static routes* process will do some checks and internal processing on these routes (such as checking that they go out over a network interface that is currently up). Finally it will communicate the remaining routes to the RIB process for use by the forwarding plane. We will now see how we send these routes to the RIB.

The rib.xif file is processed by a python script to produce the files rib xif.hh and rib xif.cc in the xorp/xrl/interfaces directory which are then compiled and linked

to produce the libribxif.la library. rib xif.hh. To use this code, we must first create an instance of this class, calling the constructor and supplying a pointer to an XrlSender. Typically such an XrlSender is an instance of an XrlRouter object.

The class XrlStaticRoutesNode actually defined an instance of XrlRibV0p1Client called xrl rib client as a member variable, so this object is created automatically when our main loop creates xrl static routes node The xrl router into the constructor for xrl rib client. So, once everything else has been initialized, we'll have access to xrl rib client from within xrl static routes node. In order to send a route to the RIB we simply call xrl rib client.send add route4() with the appropriate parameters,

The only real complication here is related to how we get the response back from the XRL. Recall that xrl rib client.send add route4() will return immediately with a local success or failure response, before the XRL has actually been transmitted to the RIB. Thus we need to pass a *callback* in to send add route4(). This callback will wrap up enough state so that when the response finally returns to the XrlRouter in the *static routes* process, it will know which method to call on which object with which parameters so as to send the response to the right place. Eg,
XorpCallback1<void, const XrlError&>::RefPtr

This defines a callback function that returns void and which takes one argument of type const XrlError.

6.3.4 Returning values in XRLs

In the rib.xif file, the XRL lookup route by dest4 returns one value of type ipv4 called nexthop. XRLs can actually return multiple parameters, eg.

lookup_route_by_dest4 ? addr:ipv4 & unicast:bool & multicast:bool & nexthop:ipv4

6.3.5 The XLOG logging facility

The XORP XLOG facility is used for log messages generation, similar to syslog. The log messages may be output to multiple output streams simultaneously.

```
int main(int argc, char *argv[])
{
// Initialize and start xlog
xlog_init(argv[0], NULL);
xlog_set_verbose(XLOG_VERBOSE_LOW); // Least verbose messages

// Increase verbosity of the error messages
xlog_level_set_verbose(XLOG_LEVEL_ERROR, XLOG_VERBOSE_HIGH);
xlog_add_default_output();
xlog_start();
// Do something
// Gracefully stop and exit xlog
xlog_stop();
xlog_exit();
exit (0);
}
```

Typically, a developer would use the macros described below to print a message, add an assert statement, place a marker, etc. If a macro accepts a message to print, the format of the message is same as printf(3). The only difference is that the XLOG utility automatically adds '\n', (i.e. end-of-line) at the end of each string specified by format:

6.4 SUMMARY

The main focus of the chapter was discussion regarding creating a new XORP process. This includes writing code for XRL interfaces, both client and target, generating required libraries, class files and header files using. The methodology also includes writing the main loop of execution, and implementing logging facility. The process requires a delicate balance of processes, and is to be coded very meticulously.

7. RESULTS

7.1 GENERAL

This chapter covers a quick overview of the intricacies of the development of the infrastructure for the implementation of the scheduling of multimedia traffic in real time environment. After the experimental setup tests have been conducted by streaming a multimedia file through VLC's VideoLAN software and also generating a heavy background traffic (TCP), through MGEN Traffic generator tool. The results are derived and displayed in the following sections.

7.2 EXPERIMENTAL SETUP

The experimental set up created for the implementation of priority based scheduling of multimedia traffic through a real time linux operating system, and routing the stream through software based routers; required tremendous effort. The installation and configuration of the real time kernel on Red Hat kernel 2.4.20 and Fedora Core 6 kernel 2.6.18, with IPv6 stack, and Quality of Service support was done successfully. The next major configuration was the installation of the XORP and Zebra, a Software based router.

XORP was considered more preferable from implementation point of view, due to its extensibility features, powerful API, event driven approach. It is worth noting that, a distinguishing factor between implementations of software router, is based on the fact, that a router is event-driven or it uses a periodic route scanner to resolve dependencies between routes. The scanner-based approach is simpler, but has a rather high latency before a route change actually takes effect. Cisco IOS and Zebra both use route scanners, with a significant latency cost; MRTD and BIRD are event-driven, but this is easier given a single monolithic process. In XORP, the decision that everything is event-driven is fundamental and has been reflected in the design and implementation of all protocols, and of the IPC mechanism.

The next part of the project work was related to streaming of a multimedia file, through VLC's VideoLAN software. The streaming is done through command line interface on Fedora core, with the –-dscp option.

Note: The option is concatenated with the vlc command line statements with –dscp =<integer> DiffServ Code Point. The Differentiated Services Code point for outgoing UDP streams (or IPv4 Type Of Service, or IPv6 Traffic Class). This is used for network Quality of Service –packetize implementation.

The core part of the implementation was development of a module at the kernel level, which gets invoked, which is event driven and gets invoked when there is a packet at the network interface. The module parses the packets and reads the dscp value. A DSCP value of 16 is identified as a high priority multimedia traffic and if put in the High Priority queue. The other traffic, which is mostly TCP traffic is kept in the Low Priority queue. The packets are released to the XORP router. On the routers, the DSCP value is checked propagated to the Routing Information Base (RIB) at the Router. The RIB, in turn notifies this to the forwarding plane and release the traffic at a high priority. This mechanism has been added as a new process to the XORP.

7.3 TRAFFIC GENERATION

Apart from the multimedia stream generated from VLC, the project work also required a heavy TCP traffic on the network. The traffic is generated to create a real time scenario of loads of traffic in network, with different bandwidth and delay requirements; within a Computer Lab. The traffic was generated using the Mgen tool.

The Multi-Generator (MGEN) is open source software by the Naval Research Laboratory (NRL) PROTocol Engineering Advanced Networking (PROTEAN) group which provides the ability to perform IP network performance tests and measurements using UDP/IP traffic (TCP is currently being developed).

The toolset generates real-time traffic patterns so that the network can be loaded in a variety of ways. The generated traffic can also be received and logged for analyses. Script files are used to drive the generated loading

patterns over the course of time. These script files can be used to emulate the traffic patterns of unicast and/or multicast UDP/IP applications. The receive portion of this tool set can be scripted to dynamically join and leave IP multicast groups. MGEN log data can be used to calculate performance statistics on throughput, packet loss rates, communication delay, and more. MGEN currently runs on various Unix-based (including MacOS X) and WIN32 platforms.

7.4 MEASEREMENT TOOLS

The tools have been chosen keeping in mind their open-source origins and flexibility of use.

1. OpenIMP :The Open*IMP* tool which is used to measure one way delay and jitter follows the IETF measurement standards and is compliant for both IPv6 anf IPv4.

2. MGEN : This tool is used to generate traffic flows. Its log files are also helpful in gauging packet loss.

3. Ethereal 0.10.8: This tool integrated with support for MPEG2 TS Parser was highly useful in analyzing the multimedia packets.

4. Ping: ICMP packets were used when setting up the testbed to check the link characteristics of loss and delay.

7.5 METHODOLOGY

The OpenIMP clients were run on the routers and the data was collected at the end of the experiment at the monitoring host C where the tool used the captured statistics to derive one-way delay, one-way delay distribution and jitter with separate graphs for traffic with different TOS field.. The test-bed never demonstrated any sort of transmission errors during the experiments.

Fragmentation was also not detected since the packet size that we used was below the link MTUs. MGEN was run at the transmitting host to generate traffic with different characteristics. A MGEN client was run at the receiver to receive the transmitted data. Since the measurements are relative to time, we had to make sure the skew between the clocks at the routers and hosts did not affect the results obtained. The Network Time Protocol daemon process, NTPD was used to synchronize time and reduce skew. More of it is described later. NTPD was run at regular intervals of 1 second when the time at Host C was used as reference point for all the other test-bed entities.

7.6 TIME SYNCHRONIZATION

NTP is a protocol designed to synchronize the clocks of a computers over a network. NTP version 3 is an internet draft standard, formalized in RFC 1305.
A key component of any computer analysis may be it is in terms of security or network performance analysis is time. If computers are running on different times, it becomes almost impossible to accurately log packets timings, and hence results in errors in calculation.

The Network Time Protocol (NTP) is used to synchronize the time of a computer client or server to another server or reference time source, in our case a machine in our network. It provides accuracy typically within a millisecond on LAN. On request, the server sends a message including its current clock value or timestamp and the client records its own timestamp upon arrival of the message. For the best accuracy, the client needs to measure the server-client propagation delay to determine its clock offset relative to the server. There are few security issues concerned with the server to prevent any malicious attack which is not considered on our case.

To update from a server:
ntpdate –u 192.168.5.3
To constantly update each two seconds:
watch ntpdate –u 192.168.5.3

7.7 TRAFFIC GENERATION

The Multi-Generator (MGEN) is open source software by the Naval Research Laboratory (NRL) Protocol Engineering Advanced Networking (PROTEAN) group which provides the ability to perform IP network performance tests and measurements using UDP/IP traffic. The toolset generates real-time traffic patterns so that the network can be loaded in a variety of ways. The generated traffic can also be received and logged for analyses. Script files are used to drive the generated loading patterns over the course of time. These script files can be used to emulate the traffic patterns of unicast and/or multicast applications. The receive portion of this tool set can be scripted to dynamically join and leave IP multicast groups. MGEN log data can be used to calculate performance statistics on throughput, packet loss rates, communication delay, and more. MGEN currently runs on various Unix-based and WIN32 platforms. The principal tool is the mgen program which can generate, receive, and log test traffic.

Syntax:
Mgen[ipv4][ipv6][input<scriptFile>][save<saveFile>][outputlogFile>][log<logFile>][binary][txlog][nolog][flush][hostAddr{on|off}][event"<mgenevent>"][portrecvPortList>][instance<name>][command<cmdInput>][sink<sinkFile>][block][source<sourceFile>][interface<interfaceName>][ttl<timeToLive>][tos<typeOfService>][label<value>][txbuffer<txSocketBufferSize>][rxbuffer<rxSocketBufferSize>]

Since MGEN allows us to set various parameters that define a particular traffic, like video, audio, VOIP, bulk data etc we used this tool to generate the required traffic with varying characteristics. This allowed us the flexibility to try out various types of streams for our experiments and validation. For example, a multimedia video traffic as having random bursts at irregular intervals through MGEN's parameters. The bursts can be generated with varying parameters. The BURST command which is used to write the burst traffic activity scripts is as follows :
BURST [REGULAR|RANDOM <aveInterval (sec)> <patternType>

[<patternParams>] FIXED|EXPONENTIAL <aveDuration (sec)>]

The BURST pattern generates bursts of other MGEN pattern types at a specified average interval. The first parameter of the BURST pattern is either "REGULAR" resulting in periodic burst uniformly distributed in time by the <aveInterval> value, or "RANDOM" which exponentially distributes the traffic generation bursts in time with an average burst interval as specified by the <aveInterval> parameter value. The characteristics of the MGEN messages generated during a burst is given by the <patternType> and associated <patternParams> parameters. The <patternType> may any MGEN pattern type including PERIODIC, POISSON, or, yes, even BURST. The <patternParams> must be appropriate for the given <patternType>. When a traffic generation burst occurs, its duration is either of a FIXED value as given by the <aveDuration> or a randomly varying duration with EXPONENTIAL statistics and an average duration as given by the <aveDuration> parameter. An example use of the BURST pattern would be to roughly emulate the "talk spurts" which might result from video conferencing system. As a voice conversation commences, a user's burst of activity (talk spurts) might be RANDOM with some average interval and the duration talk spurts approximate EXPONENTIAL statistics. > When the talk spurt (burst) occurs, the voice compression codec might generate messages following something like a PERIODIC flow with packet rates and packet sizes dependent upon the voice codec in use. Other uses of the BURST pattern might be to roughly model message/packet generation occurring with random use of a network such as web browsing, etc. The BURST model provided by MGEN does not presuppose any specific traffic model, but might be useful in approximating some models of regular or intermittent network activity.

7.8 TEST

This experiment was conducted using 2 streams, of which one was multimedia traffic while the other was normal periodic traffic. The bandwidth required is 120Mbps which is greater than 100Mbps link. The multimedia traffic being Periodic with regular bursts, it was expected the delays would increase abruptly

at burst instances when the traffic bandwidth required would be much greater than that available [14]. The queuing delays increase and average delay shoots up. Table 7.1 describes the traffic profile of the multimedia traffic's characteristics.

Table 7.1: Profile for Traffic Generation for both IPv6 and IPv4 network

Stream	Type	Rate (packet/sec)	Size (in bytes)
Multimedia (UDP)	Periodic Burst (every 4.5 sec)	4500 4500	1024 1024
Normal (TCP)	Periodic	3000	1024

For some experiments to calculate latency for different packet size, different packet size has been considered.

7.9 RESULTS

The experiments were conducted for both IPv6 and IPv4 networks. The comparison between the two, with reference to, Packet Loss, Average and Maximum Jitter and Average receive time, is displayed in Table 7.2. Note that jitter is defined to be the difference between the minimal and maximal datagram delays.

Table 7.2: Comparison of IPv4 and IPv6

	Packets Lost (%)	Average jitter (msec)	Maximum jitter (msec)	Average receive time(msec)
IPv4	5	40	81	98
Ipv6	1	21	65	20

The IPv6 network showed a better quality, as compared to the IPv4 network. Also the bandwidth consumption of the IPv6 network is much more stable as compared to the IPv4 network. This is shown in the figure Fig 7.1 and Fig 7.2.

Also, the UDP foreground traffic, ie, multimedia traffic, suffers less packet loss w.r.t the TCP background traffic (generated artificially through MGEN Traffic Generator Tool). The results derived after the experiment was used to display packet loss, delay and latency.

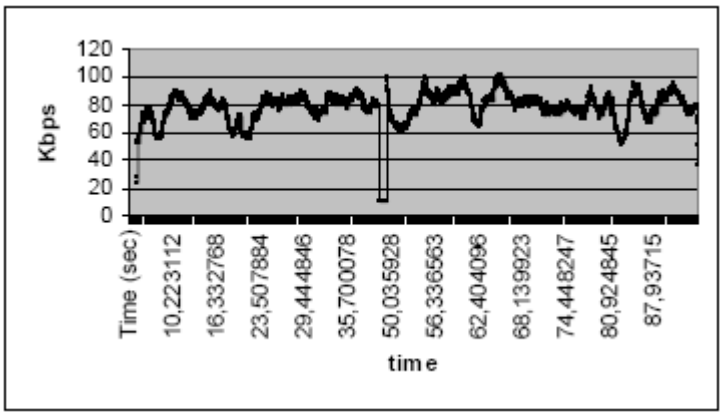

Fig 7.1: IPv4 bandwidth consumption

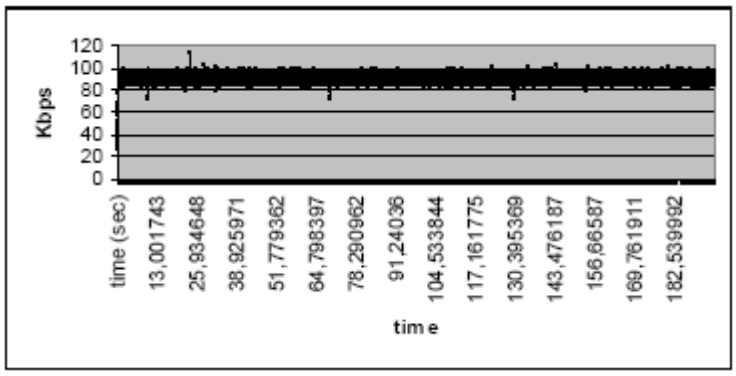

Fig 7.2: IPv6 bandwidth consumption

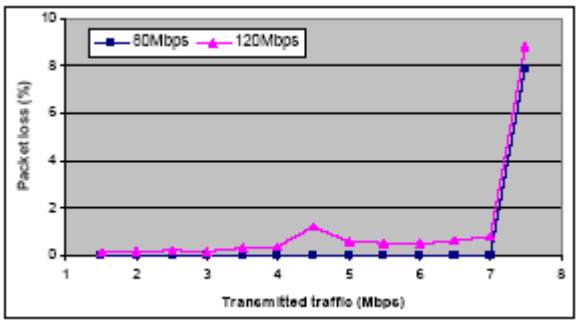

Fig 7.3: Packet Loss for the High Priority Multimedia Traffic.

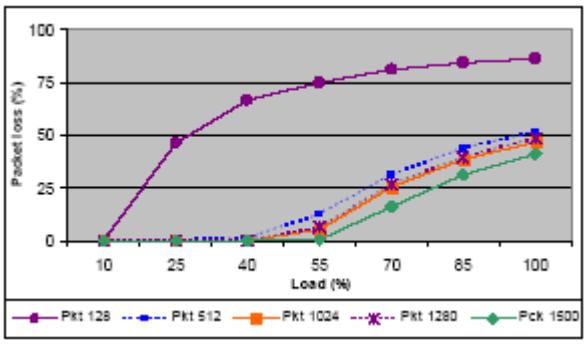

Fig 7.4: Packet Loss for different packet sizes.

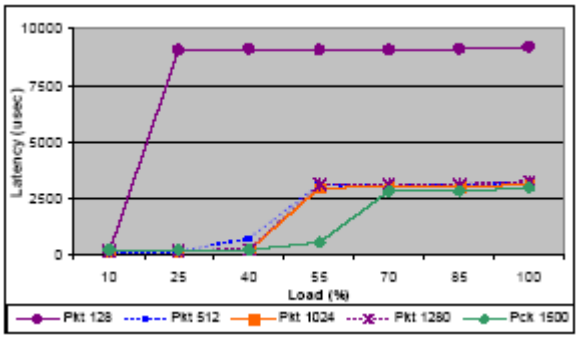

Fig 7.5: Latency for different packet sizes

7.9 MPEG-2 TS DISSECTOR FOR ETHEREAL 0.10.8

The MPEG-2 TS (Transport Stream) dissector for Ethereal can display transport stream and section heading information, reassemble the sections and analyze the PSI (Program Specific Information) data. The following section covers the procedure required to open a MPEG2 TS Stream and analyze its fields. Open the .ts file (created through VideoLAN), from Ethereal (version 0.10.8). This is required to dissect the MPEG2 TS stream sections contained in the .ts file.

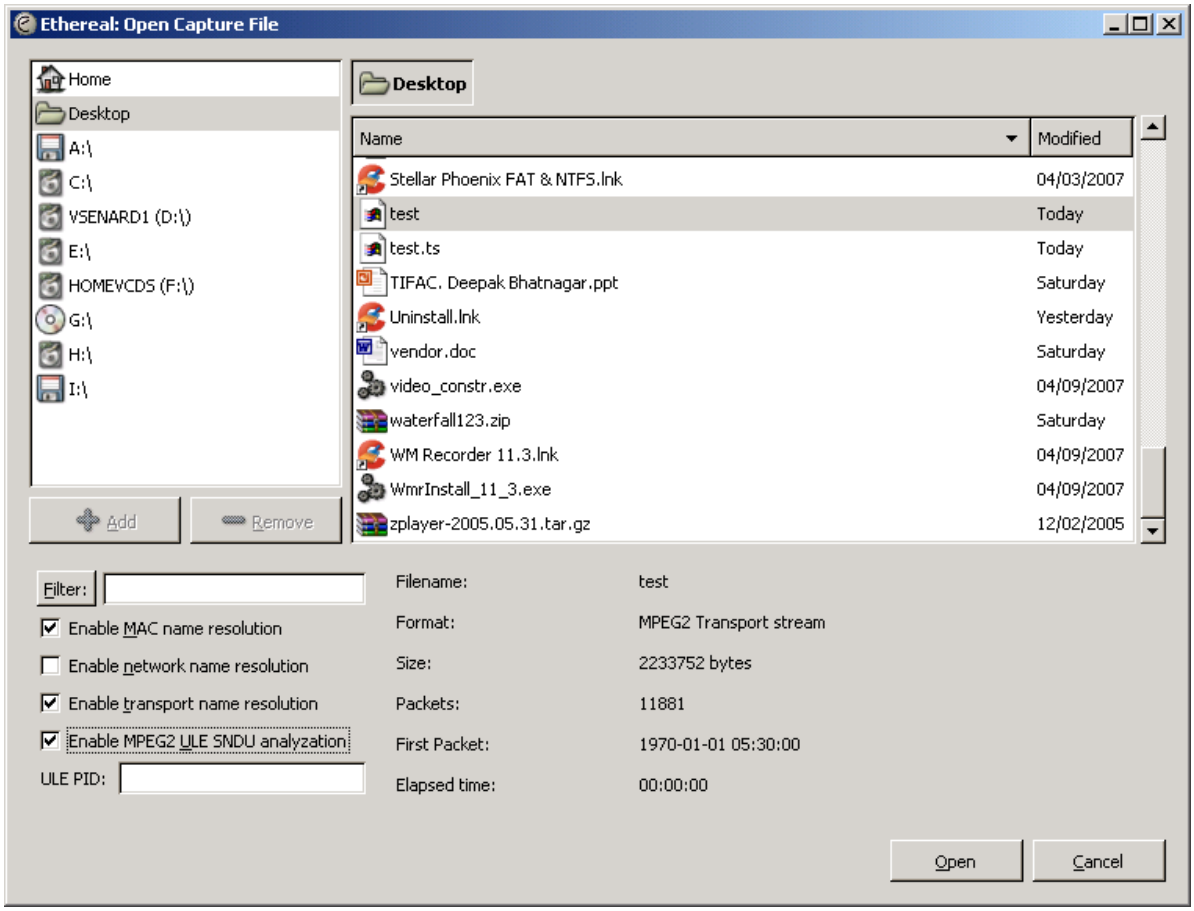

Fig 7.6: Ethereal's Open page for Transport stream file

There are two possibilities to dissect MPEG 2 ULE, ie analyze ULE datagrams with specific PIDs, analyze a Stream only containing ULE SNDUs. In order to analyze a transport stream carrying both ULE SNDUs and normal sections click the check box "Enable MPEG2 ULE SNDU analyzation" and enter the ULE PIDs into the "ULE PID" text field separated with blanks either in hexadecimal or decimal format (e.g. 0x12 25 0x34 ...). If the value is correct the text field background will turn green. To analyze a stream only consisting of ULE SNDUs just click the check box "Enable MPEG2 ULE SNDU analyzation" and the whole transport stream will be analyzed as a stream only containing ULE SNDUs as shown in Figure 7.7. If these settings are wrong the analysis will be incorrect.

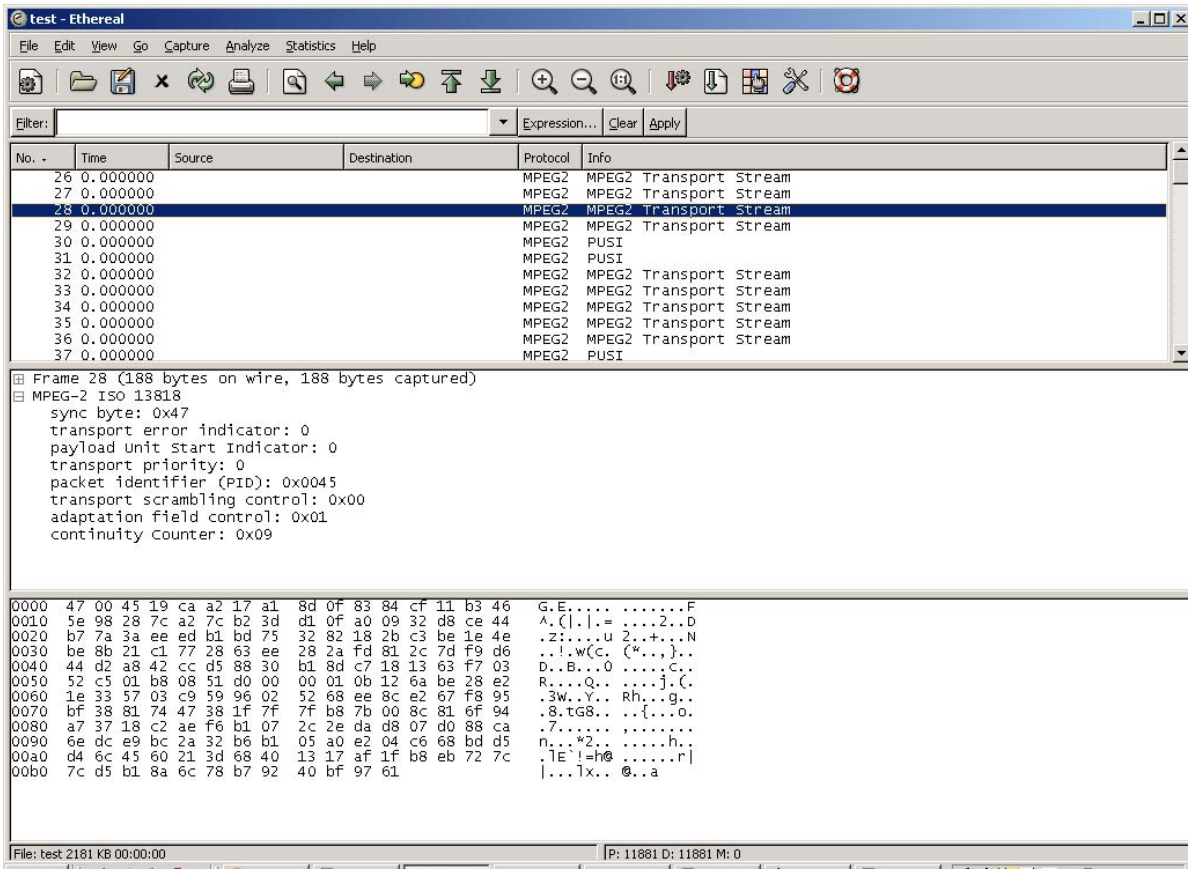

Fig 7.7: Ethereal's Transport stream file details

The figure below shows the Transport priority marked with red circle. The priority was marked at source while streaming the multimedia stream.

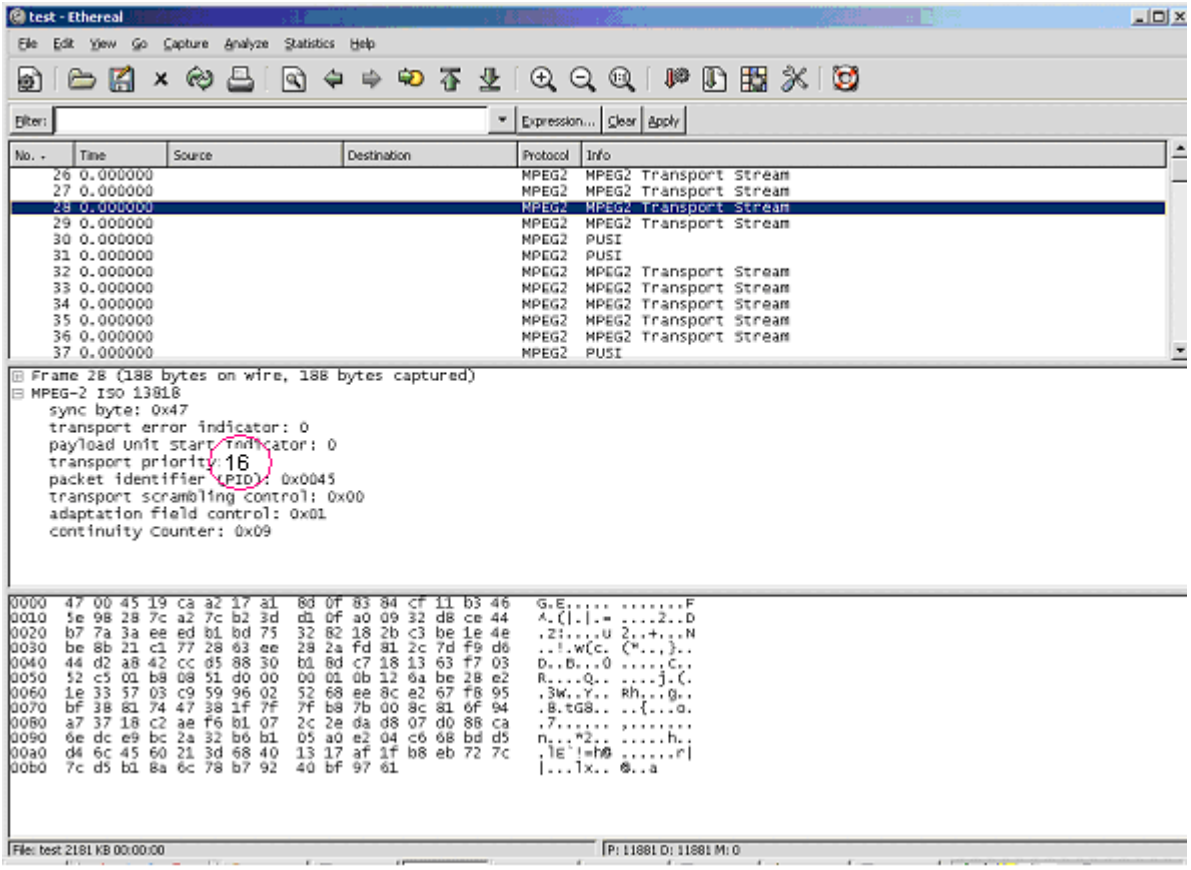

Fig 7.8: Transport stream file, showing a High Transport priority

8. SUMMARY AND CONCLUSION

8.1 SUMMARY

The experimental set up created for the implementation of priority based scheduling of multimedia traffic through a real time linux operating system, and routing the stream through software based routers; required tremendous effort. The installation and configuration of the real time kernel on Red Hat kernel 2.4.20 and Fedora Core 6 kernel 2.6.18, with IPv6 stack, and Quality of Service support was done successfully. The next major configuration was the installation of the XORP and Zebra, a Software based router.

The next part of the project work was related to streaming of a multimedia file, through VLC's VideoLAN software. The streaming is done through command line interface on Fedora core, with the –-dscp option.

However, the core part of the implementation was development of a module at the kernel level, which gets invoked, which is event driven and gets invoked when there is a packet at the network interface. The module parses the packets and reads the DSCP value. A DSCP value of 16 is identified as a high priority multimedia traffic and if put in the High Priority queue. The other traffic, which is mostly TCP traffic is kept in the Low Priority queue. The packets are released to the XORP router. On the routers, the DSCP value is checked propagated to the Routing Information Base (RIB) at the Router. The RIB, in turn notifies this to the forwarding plane and release the traffic at a high priority. This mechanism has been added as a new process to the XORP.

8.2 CONCLUSION

XORP was considered more preferable from implementation point of view,

due to its extensibility features, powerful API, event driven approach. It is worth noting that, a distinguishing factor between implementations of software router, is based on the fact, that a router is event-driven or it uses a periodic route scanner to resolve dependencies between routes. The scanner-based approach is simpler, but has a rather high latency before a route change actually takes effect. Cisco IOS and Zebra both use route scanners, with a significant latency cost; MRTD and BIRD are event-driven, but this is easier given a single monolithic process. In XORP, the decision that everything is event-driven is fundamental and has been reflected in the design and implementation of all protocols, and of the IPC mechanism.

The experimental results indicated that IPv6 traffic has less of packet loss, latency and jitter problems as compared to the IPv4 network. One reason could be the overall packet flow strategies of IPv6 and the other could be that IPv6 network are relatively less congested, due to less usage, w.r.t the IPv4 network. Also the foreground traffic, ie multimedia (high priority) traffic showed less jitter and latency w.r.t the background traffic (TCP traffic) generated through the MGEN traffic generator.

8.3 FUTURE SCOPE

The methodology that is implemented successfully for the project work, is related to implementation of a module for DiffServ architecture in the real time kernel.

The other strategy which is under implementation is DiffServ-aware Video Streaming. This refers to a streaming system characterized by the integration of two main components: a DiffServ-enabled IP network and a DiffServ-aware video streaming application. The general idea behind this concept is that performance may be improved if applications are aware (make use) of the enhanced network capabilities offered by the DiffServ architecture. Thus, a DiffServ-aware application marks its packets in order

for them to take advantage of the advanced rate, delay, jitter and/or loss features offered by the network. This approach is called application-driven (or application-controlled) marking, and is different from the typical DiffServ router-based marking which is done by edge routers based on rate-metering or flow identification.

The compressed streams generated by video codecs are usually organized in a hierarchical structure. For example, at the frame level, an MPEG-compressed video stream contains three types of frames: I, P and B. The loss of each type of frame has a different impact on the visual quality of the received signal. The loss of an I frame has a bigger impact than that of a B frame. If an I frame is lost, in addition to the loss of the frame itself, some neighboring frames can also be considered lost for practical reasons. This is due to the fact that they cannot be properly decoded even if they are correctly received, because the reference information from the lost I frame is not available. One can speak then of error propagation.

Thus overall the dissertation work is a mix of exhaustive configurations in Linux environment and core level network programming, which also looks into the operating system level intricacies for scheduling of multimedia traffic, in a real time environment.

REFERENCES

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

2. Rajahalme, J., Conta, A., Carpenter, B., & Deering, S.*, RFC3697: IPv6 Flow Label Specification*, March 2004

3. Manish Mahajan and Manish Parashar, *Managing QoS for Multimedia Applications in the Differentiated Services Environment*, The Applied Software Systems Laboratory (TASSL), 94 Brett Road, Piscataway, New Jersey, October 2004

4. Ashwin Kumar Chimata, *A Route Control Platform using XORP Design Considerations*, July 27, 2006

5. Ferguson, P., & Huston, G., *Quality of Service: Delivering QoS on the Internet and in Corporate Networks*. New York, NY: Wiley Press, 1998.

6. Almquist, P, *RFC1349: Type of Service in the Internet Protocol Suite.*, July 1998

7. Johann Max Hofmann Magalhaes, Paulo Roberto Guardieiro, *A New QoS Mapping for streamed MPEG Video over a DiffServ Domain* Larisa Rizvanovic, Gerhar Fohler, Federal University of Uberlandia, 2002

8. W Stevens, M. Thomas, *RFC 2292: Advanced Socket API for IPv6*, AltaVista, February 1998

9. R. Gilligan, S. Thomson, Bellcore, J. Bound, Compaq W. Stevens, *RFC 2553 - Basic Socket Interface Extensions for IPv6*, March 1999

10. Jau-Hsiung Huang and Ing-Chau Chang, *Communications and Multimedia Lab, A real-time scheduling policy for Multimedia applications on broadcast Network,* Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, *2003*

11. Ahmed, T. Nafaa, A. Mehaoua, A. , *An object-based MPEG-4 multimedia content classification model for IP QoS differentiation,* CNRS-PRiSM Lab., Versailles Univ., France; July 2003

12. Michael Barabano*, A Linux based real time operating system,* New Maxico Institute of Mining and Technology, 1998

13. Mark Handley Adam Greenhalgh, XORP: Breaking the Mould in Router Software, Dept of Computer Science, University College London, 2004

14. Reinder J. Brill. *Real-time scheduling for media processing using conditionally guaranteed budgets*. Ph.D Thesis, Technical University Eindhoven, 2004.

APPENDIX A LIST OF WEBSITES

1. XORP Website: http://www.xorp.org/

2. ZEBRA Website: http:// www.zebra.org

3. Linux kernel: http:// www.kernel.org

4. FSMLsbs: http://www.fsmlabs.com

5. IP QoS Project: http://dpnm.postech.ac.kr/research/01/ipqos/

6. Ipv6 Website: http://www.ipv6.org/

7. OpenIMP: http://www.ip-measurement.org/openimp/

8. Fedora Project: http://fedoraproject.org/wiki/

9. Linux Ipv6 HowTo: http://tldp.org/HOWTO/Linux+IPv6-HOWTO/index.html

10. VideoLAN HOW TO: http://tldp.org/HOWTO/VideoLAN-HOWTO/ and http://www.videolan.org/vlc

11. RFC's repository: http://www.rfc-archive.org/getrfc.php