

Department Of Computer Science & Engineering

Kumaraguru College of Technology

Coimbatore -641 006

CERTIFICATE

This is to certify that the project work entitled

IMPLEMENTATION OF
MPLS FOR REAL TIME IP TRAFFIC

Done by

R.PATHMA, Reg No 9927K0147

S.PAVITHRA, Reg No 9927K0150

Submitted in partial fulfillment of the requirements for the award of the

degree of

BACHELOR OF ENGINEERING

OF BHARATHIAR UNIVERSITY, COIMBATORE

Professor and Head

Internal Guide

Submitted for University Examination held on 18/05/07

Internal Examiner

External Examiner

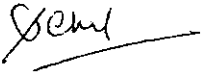
भारतीय इलेक्ट्रॉनिकी अनुसंधान एवं विकास केन्द्र
(भारत सरकार सूचना प्रौद्योगिकी मंत्रालय
का स्वायत्त वैज्ञानिक संस्थान)
तिरुवनन्तपुरम यूनिट
डा.पे.सं. 6520, वेळयंबलम
तिरुवनन्तपुरम 695 033, भारत

**ELECTRONICS RESEARCH AND
DEVELOPMENT CENTRE OF INDIA**
(An Autonomous Scientific Society of Ministry of
Information Technology, Government of India)
Thiruvananthapuram Unit
P.B. No. 6520, Vellayambalam
Thiruvananthapuram 695 033, India

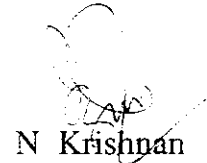
Phone: +91-471-320116 Fax: +91-471-331654, 332230 email: erdc@erdcitym.org Web: www.erdcitym.org

CERTIFICATE

Certified that Ms. S Pavithra, Ms. R Pathma, Kumaraguru College of
Technology, Coimbatore, have successfully completed the project titled,
“**Implementation of MPLS for Real Time IP Traffic**” using **Linux** at the
Centre during the period November 2002 to March 2003, towards the partial
fulfillment of the requirements for the award of B.E degree in Computer Science
and Engineering from Bharathiar University, Coimbatore.



Saramma Chacko
Joint Director



N Krishnan
Additional Director

(Project Guide)

Thiruvananthapuram
10th March 2003

N. KRISHNAN
ADDITIONAL DIRECTOR
CONTROL & INSTRUMENTATION
ELECTRONICS RESEARCH & DEV. CENTRE OF INDIA
MINISTRY OF INFORMATION TECHNOLOGY, GOVT. OF INDIA
VELLAYAMBALAM, THIRUVANANTHAPURAM-695 033, INDIA

SYNOPSIS

The most widely used internet architecture offers a very straightforward point to point delivery service, which is based on the best effort delivery model. In this method, the highest pledge the network provides is the reliable data delivery. This is adequate for traditional applications like ftp and telnet, which requires correct data delivery than prompt delivery.

But issues in the current development of Internet seem to be its capability to scale and to support new real-time or near real-time applications like video and audio conferencing. These application experience quality degradation if the packets are delayed for that long. There are two factors that affect these qualities: one is the ability to distinguish which connections should be switched and the other is the effective control over network resources.

P-867

To classify and handle the huge number of applications, a network requires Quality of Service in addition to Best effort service. Thus before these applications can be widely used, the Internet infrastructure must be modified to support real time QoS and controlled end-to-end delays.

Label switching is a natural evolutionary step for the Internet. Over the last few years many companies have proposed different techniques to implement label switching. Notable ones for IP packet handling for real time applications are Cell Switching Router (CSR) by Toshiba, IP switching by Ipsilon, Tag Switching by Cisco and Aggregate Route –based IP Switching (SRIS) by IBM. **Multi Protocol Label Switching (MPLS)** an improved version of the Tag Switching has emerged as an industry-standard approach to switching and forwarding. MPLS reduces the complexity of forwarding using encapsulated fixed-length labels for making high speed forwarding decisions, thus eliminating conventional network-layer header processing. It simplifies the routing of packets and facilitates the selection of optimal paths through the Internet labyrinth.

MPLS also supports Quality of Service (QoS) by allowing bandwidth reservations and traffic prioritization through the so-called “best effort” Internet.

The main objective of this project is to build a MPLS framework model for network infrastructure incorporating real time applications in addition to traditional applications.

CONTENTS

	<u>Pg.No</u>
1. Introduction	1
1.1 Current Status of the problem	
1.2 Relevance and Importance of the topic	
2. Literature survey	5
3. System Requirements	8
3.1 Product Definition	
3.2 Functional Specification	
3.3 Developing and Processing Environment	
3.3.1 Software	
3.3.2 Hardware	
4. Line of Attack	11
4.1 Linux Kernel with MPLS support	
4.2 Implementing MPLS over PPP	
4.3 Implementing MPLS over Ethernet	
4.4 Implementing MPLS over ATM	
4.5 Providing different classes of service using MPLS –enhanced Software Switch	
5. Details of proposed methodology	14
5.1 MPLS - Introduction	
5.2 MPLS - Functions	
5.3 MPLS –Operational Overview	
5.4 Proposed System Architecture	
5.5 Implementation Details	
6. Results	27
7. Conclusion and Future outlook	37
8. References	40
9. Appendices	42

1. INTRODUCTION

1.1 CURRENT STATUS OF THE PROBLEM

There are various limitations in legacy IP networking technologies which make it impossible, predicting the bandwidth available on a link at any given time and there are ample difficulties in controlling the allocation of shared Bandwidth. This also paved way for congestion in the network as the packets are routed through a common path to the destination. Routing was perceived as processor oriented. Though early solutions developed across the globe address the need for wire-speed transfer of packets as they traversed the network, they did not address the service requirements of the information contained in the packets. Over the last few years, the new applications that have been developed increased the demand for guaranteed bandwidth management in the backbone of the network. Most of routing protocols deployed today are based on algorithms designed to obtain shortest path in the network for packet traversal and do not take into account additional metric such as delay, jitter and traffic congestion, which can further diminish network performance. MPLS is a lucid solution to these problems which enable multiple network links to be setup as required, and not only the congestion problems are addressed but also load balancing acts are done much efficiently.

IP forwarding is done based on longest prefix match of the destination address. Either a longest match or a default route should be present in the forward table. Transit providers don't do default routing and they need full routing table in every core router. In MPLS packets are forwarded based on the label value. Packet forwarding is de-coupled from IP. This is the key benefit of MPLS. Other paradigms may be used to forward traffic. There is no need to strictly follow unicast destination based routing, which enables MPLS applications (VPN, Traffic Engineering etc...)

In a normally environment, frames pass from a source to destination in a hop by hop basis. Transit routers evaluate each frame's layer 3 headers and perform a route table lookup to determine the next hop toward the destination. This tends to reduce throughput in a network because of the intensive CPU requirements to process each frame. Routing protocols have little, if any, visibility into the Layer 2 characteristics of the network, particularly in regard to QoS. The primary goal of QoS is to provide priority including dedicated bandwidth, controlled jitter and improved loss characteristics. Also important is making sure that providing priority for one or more flows does not make other flows fail.

To meet these new demands MPLS changes the hop - by - hop paradigm by enabling devices to specify path in the networks using a label, this path will be based upon Quality of Service and bandwidths need of an application. MPLS address performance issues related to emerging Internet applications such as real-time voice and video streaming by giving such applications higher priority and more bandwidth than data packets.

1.2 RELEVANCE AND IMPORTANCE OF THE TOPIC

Internet Protocol (IP) traffic on the Internet and enterprise networks has been growing exponentially for some time. This growth is beginning to stress the traditional processor based design of current-day routers. There have been in recent years, many announcements of new technologies that promise to change the way data is forwarded or switched in the Internet and other networks based on the IP suite. **Many techniques strive to use IP addresses and standard Internet routing protocols** such as **OSPF and BGP**, while maintaining the control paradigm of the Internet. In many aspects, this approach combines **the simplicity, scalability, and robustness of IP with the speed, capacity, and multi-service capabilities of ATM**. In the past, routing functionality was notoriously difficult to evolve, and the reason for this is the close coupling between routing and forwarding in IP networks.

This **data-driven approach** has a drawback in terms of performance because it is difficult to predict performance since it depends so much on the detailed characteristics of offered traffic. A small change in the length or number of flows passing through a point may cause a large change in the percentage of traffic that can be switched, resulting in **overload of the control processor**. In traditional IP routing full IP header analysis occurs at every node, unicast and multicast support requires multiple complex forwarding algorithms and routing decisions is based only on address. Most service providers have implemented connection-oriented ATM networks in the core. ATM provides performance, bandwidth, and the ability to perform traffic engineering. However, an **ATM network does not scale well**, because it relies on virtual circuit state information in the core.

The Internet has experienced remarkable growth. Label switching is a response to this growth and these challenges. **Label switching** is an advanced form of packet forwarding that replaces conventional longest address match forwarding with a more efficient label swapping algorithm. Many of these technologies are based on a set of common ideas which is the use of a label swapping technique for forwarding data, the same technique that is used to forward data in ATM switches. Development of the Label switching field not only stemmed from the need for fast, cheap routers, but as well as other factors. As well as **increased speed, the networks need to deal with increased numbers of nodes, more routes in routing tables, more flows passing through a given point** and so forth.

Label switching's attraction is that the forwarding algorithm is fixed and that new control paradigms can be deployed without making any changes to it. It is possible to put the **forwarding algorithm in hardware** or to tune **the fast path software once** without concern that it will need to be re-optimized every time a new piece of routing functionality is required. Some believe that label

switching (control driven) is likely to form the foundation for the next generation of routing architecture. In Label switching full IP header analysis only once at the network edge when label is assigned, unicast and multicast support requires just one forwarding algorithm and is sufficient and the routing decisions can be based on any number of parameters such as **QoS, VPN membership**, etc, MPLS enhanced ATM scales well the provider must configure only the edge device (provider edge) router, as opposed to all the routers in the network.

2. LITERATURE SURVEY

Development of the Label switching field not only stemmed from the need for fast, cheap routers, but as well as other factors. As mentioned above, the Internet has experienced remarkable growth. Label switching is a response to this growth and these. Over the last few years many companies have attempted to blend the high speed operation of ATM-based switching with the routing processes of the Internet's IP-based network layer. Four of these are noteworthy:

Cell Switching Router (CSR)

CSR approach was developed by Toshiba and presented to the IETF in 1994. It was one of the earliest public proposals for using IP protocols to control an ATM switching fabric. CSR is designed to function as a router for connecting logical IP subnets in a classical 'IP over ATM' environment. Label switching devices communicate over standard ATM virtual circuits. CSR labeling is data-driven (i.e., labels are assigned on the basis of flows that are locally identified). The Flow Attribute Notification Protocol (FANP) is used to identify the dedicated VCs between CSR's and to establish the association between individual flows and individual dedicated VCs. The objective of the CSR is to allow 'cut through' forwarding of flows, i.e., to switch the ATM cell flow that constitutes the packet rather than reassembling it and making an IP level forwarding decision on it. CSRs have been deployed in commercial and academic networks in Japan.

IP Switching

It was developed by Ipsilon (who are now part of Nokia), was announced in early 1996 and has been delivered in commercial products. IP Switching enables a device with the performance of an ATM switch to act as a router.

thereby overcoming the limited packet throughput of traditional routers. The basic goal of IP Switching is to integrate ATM switches and IP routing in a simple and efficient way (by eliminating the ATM control plane). IP Switching uses the presence of data traffic to drive the establishment of a label. A label binding protocol (called the Ipsilon Flow Management Protocol or GSMP) are defined. GSMP is used solely to control an ATM switch and the virtual circuits made across it.

Tag Switching

It is the label switching approach developed by Cisco Systems. In contrast to CSR and IP Switching, Tag Switching is a control-driven technique that does not depend on the flow of data to stimulate setting up of label forwarding tables in the router. A Tag Switching network consists of Tag Edge Routers and Tag Switching Routers, with packet tagging being the responsibility of the edge router. Standard IP routing protocols are used to determine the next hop for traffic. Tags are 'bound' to routes in a routing table and distributed to peers via a Tag Distribution Protocol. Tag switching is available on a number of products from Cisco.

Aggregate Route-based IP Switching (ARIS),

IBM's label switching approach is similar architecturally to Tag Switching. ARIS binds labels to aggregate routes (groups of address prefixes) rather than to flows (unlike CSR or IP Switching). Label bindings and label switched paths are set up in response to control traffic (such as routing updates) rather than data flows, with the egress router generally the initiator. Routers that are ARIS-capable are called Integrated Switch Routers. ARIS was designed with a focus on ATM as the Data Link Layer of choice (it provides loop prevention mechanisms that are not available in ATM). The ARIS Protocol is a peer-to-

peer protocol that runs between ISRs directly over IP and provides a means to establish neighbors and to exchange label bindings. A key concept in ARIS is the “egress identifier”. Label distribution begins at egress router and propagates in an orderly fashion towards the ingress router.

3. SYSTEM REQUIREMENTS

3.1 Product definition

Traditional telecommunications services and more recent data services, such as Intranets/Internet access, can be converged on a single network; enterprises are increasingly looking for more advanced and specialized services tailored to their specific needs. This will enable enterprises to dynamically adjust their network requirements based upon factors such as traffic loading per application (bandwidth allocation) and application performance (QoS/CoS).

Current IP networks are a long way from meeting the requirements of service providers and their customers. The capabilities embodied in MPLS are designed to meet the unified transport requirements of large-scale IP networks and build on the existing concepts of IP networking.

3.2 Functional specification

The aim of the project is to develop robust & reliable backbone which provides an effective class of service environment. The system should be able to provide 3 classes of service viz gold, silver & bronze. Each service class is assigned to a typical user on his priority i.e.

Gold → high priority

Silver → medium priority

Bronze → low priority

Prioritization of client requests by this backbone should be able to provide specific QoS parameters viz bandwidth, speed & time, for different clients based on priority.

3.3 Developing and Processing Environment:

3.3.1 Software

- Operating System

Red Hat Linux: version -7.2

Kernel Version - 2.4.19

Linux is a free, open-source, POSIX compliant, UNIX clone operating system. Its true preemptive multitasking, multi-user support, memory protection and symmetric multiprocessing support characteristics together with its networking, graphical user interface, speed and stability make Linux a preferred tool for research and development.

- Programming Language

C Programming in Linux Environment

It is a robust language whose rich set of built-in functions and operations can be used to write any complex program. The C compiler combines the capabilities of an assembly language with features of high level language and therefore it is well-suited for writing both system software and business packages. Programs written in C are efficient and fast. C is highly portable. C language is well-suited for structured program. Its important feature is its ability to extend itself. With its availability of large number of functions the programming task becomes simple.

- GUI Interface

Ethereal Packet Analyser (available under Linux Platform)

3.3.2 Hardware

- RS-232 for serial communication (PPP)
- Network Interface Cards (NICs) with cross connect cables for Ethernet Interface.
- ATM NICs for with cross-connect cables for ATM Interface
- 4 GB or more hard disk.
- 64 MB RAM



4. LINE OF ATTACK

The project aims at implementing MPLS framework in a 4 node domain network in Linux .The primary goals are to assign different interfaces for various services based on the priority of the client.

4.1 Linux kernel with MPLS support

This module of the system is to ensure that a node in order to be a part of an MPLS domain has its LINUX kernel with MPLS functionality. There are 2 LER's and 2 LSR's. Ingress LER is used to add the MPLS label to an incoming packet from a Non- MPLS domain, if its destination is within the MPLS domain. Egress LER is used to remove the label from the packet and forward it to the Non- MPLS domain. LSR's are label switched routers which are used for MPLS label look-up and forwarding to a node in MPLS domain.

4.2 Implementing MPLS over PPP

PPP(Point to Point Protocol) is a mechanism for creating and running (the Internet Protocol) and other network protocols over a serial link- direct serial connection (using null modem cable).This module involves enabling PPP support for the MPLS kernel in each LSR, which performed routing and forwarding in MPLS domain. The point-to-point protocol is designed for simple links which transport packets between two peers. The PPP daemon (pppd), negotiates with the peer to establish the link and set up the PPP network interface. Thus the Bronze class of service can be provided for the respective client.

4.3 Implementing MPLS over Ethernet

MPLS improves Ethernet not by changing what is good about Ethernet, but by adding desired capabilities at a higher layer (Layer 2).This module involves enabling Ethernet support for the MPLS kernel in each LSR, which performed routing and forwarding in MPLS domain. When MPLS and Ethernet are used

together, Ethernet is called to focus on only what it does best- Transport between point -to-point peers, MPLS meanwhile adds the connection -oriented capabilities that are a tool for creating service level Agreement-backed services. By this phase the 'Silver' class of service i.e., mission critical, guaranteed service is provided to the client.

4.4 Implementing MPLS over ATM

ATM is Asynchronous Transfer Mode. ATM allows for data, voice, and video all to share the same medium. ATM can connect to a variety of other network types varying from a 25Mbps connections to a 455 Mbps backbone. This module specifies the MPLS encapsulation to be used when sending labeled packets from ATM - LSR. The specified encapsulation is to be used for multicast or explicitly routed label packets as well. The goal is to provide High-Priority, low latency, Premium -class Gold service for the typical user.

4.5 Providing different classes of service using MPLS - enhanced Software Switch

One major difference between PPP, Ethernet and an ATM connection is of course speed- a standard ATM connection operates at 155 Mbps, an Ethernet connection operates at 10Mbps whereas an analogue modem operates at speeds up to 56 Kbps (maximum theoretical throughput).

The above mentioned configurations will act as a universal router/switch by supporting any Layer 2 protocol (ATM, Ethernet, etc.) and by being able to encapsulate any Layer 3 packets (IP, MPLS, etc). Thus the idea of selecting one of the three interfaces (PPP, Ethernet, and ATM) and providing different level of services is incorporated in one single switch. As the packets arrive to it, they will be filtered and classified. The classification in this project is done based on the clients' priority. Those classes of packets will each follow a different

interface for their requested services, purely based on their source and assigned priority.

5. DETAILS OF THE PROPOSED METHODOLOGY

5.1 MPLS -- Introduction

MPLS is a switching method used by special routers to forward IP packets. Those routers forward IP traffic using a label (usually between Layer 2 and Layer 3) to instruct other routers where to forward packets based on pre-established IP routing information. MPLS also supports the capability of the network to provide better service and higher priority to some selected network traffic. As the name suggests it is called multi-protocol because its techniques are applicable to any network protocol. MPLS provides the ability to support any type of traffic on a large IP network without having to subordinate the design to the limitations of different routing protocols, transport layers and addressing schemes.

MPLS is an IETF (Internet Engineering Task Force) - specified frame work that addresses performance issues related to emerging internet applications such as real-time voice and video streaming by giving such applications higher priority and more bandwidth than data packets, for example using the same network. Because routers switch the frame based upon the labels and without need to perform usual routing operations, frames are usually handled more quickly.

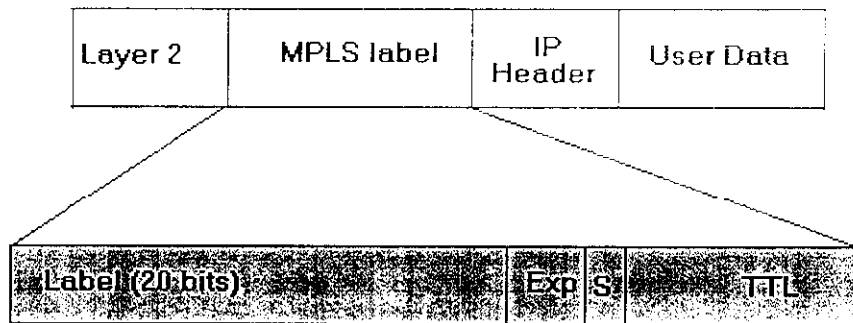
Multi-Protocol Label Switching (MPLS) is a new technology that will be used by many future core networks, including converged data and voice networks. MPLS does not replace IP routing, but will work alongside existing and future routing technologies to provide very high-speed data forwarding between Label-Switched Routers (LSRs) together with reservation of bandwidth for traffic flows with differing Quality of Service (QoS) requirements. MPLS enhances the services that can be provided by IP networks, offering scope for Traffic Engineering, guaranteed QoS and Virtual Private Networks (VPNs).

5.2 MPLS- Functions

- Specifies mechanisms to manage traffic flows of various granularities, such as flows between different hardware, machines or even flows between different applications.
- Remains independent of the layer-2 and Layer-3 protocols.
- Provides a means to map IP addresses to simple, fixed length labels used by different packet-forwarding and packet-switching technologies.
- Interfaces to existing routing protocols such as Resource Reservation Protocol (RSVP) and Open Shortest Path First (OSPF).
- Supports the IP, ATM and frame-relay Layer-2 protocols.

5.3 MPLS- Operational Overview

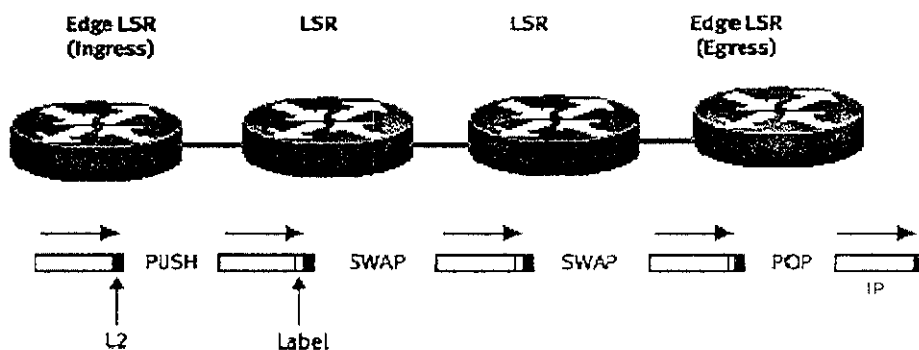
Conventional wisdom in the data communications industry suggests that “switching” is simple and that “routing” is complex. MPLS allows IP packets to be “switched” through the Internet. To switch the packets, a short (4-byte) label is inserted in the packet header. This label is then used to forward the packet between MPLS routers throughout the Internet. This label is inserted at the ingress to the MPLS domain and then removed at the egress. Hence, an IP packet is generated at the source (a workstation, for example) and an IP packet is delivered to the destination (i.e., a remote file server). The fact that an MPLS label temporarily exists between the source and destination is completely transparent to the users, the applications, and even the customer’s networking equipment.



MPLS label format

As indicated above, the MPLS Label (often called a *shim*) is placed between the Layer 2 (e.g., Ethernet) and Layer 3 (IP) headers in the packet. Packets are then switched based on the label values. The labels only have local significance (similar to Frame Relay DLCIs or ATM VPI/VCI) and will change from hop to hop as the packet traverses the Internet. In *Figure 1 on page 6*, four MPLS routers, also called *Label Switching Routers (LSRs)* are shown. These routers all reside in the Internet, or within the “cloud.” Therefore, the terms “ingress” and “egress” refer to the entry and exit points of the Internet. As a packet enters the Internet, a label is inserted or “pushed” onto the packet header. This label is then exchanged or “swapped” at each intermediate router hop. This is accomplished by mapping the incoming label to an outgoing label based on the entries in a table contained inside the router. For example, a packet might arrive via port six with a label value of 45.

The router’s label table may indicate that such a packet then needs to be forwarded via port ten with a label of 61. In this case, the appropriate Layer 2 header will be built for the outbound link layer protocol associated with port ten and the incoming label value of 45 will be replaced with an outbound label of 61.



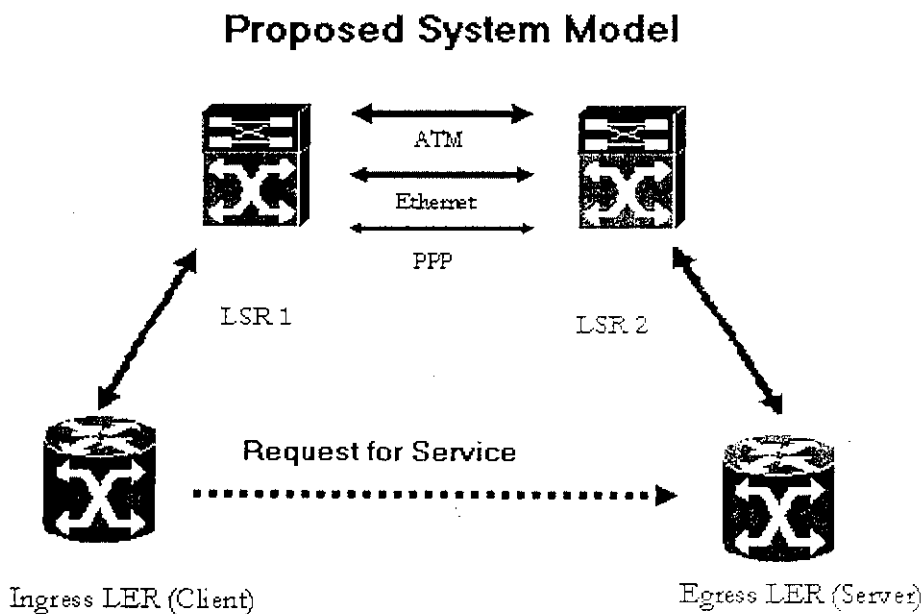
The label is eventually removed or “popped” from the packet at the egress of the Internet and a traditional IP packet is delivered to the customer’s destination network. This packet can then be forwarded to its final destination via standard IP routing procedures. The MPLS label field will support over one million unique labels. However, the creation and maintenance of massive label tables will require considerable routing hardware resources. Therefore, it is highly recommended that MPLS networks be designed based on limited quantities of labels in the routers’ tables. A label table can be as simple as having one entry per physical port on the router. Hence, if a packet is received with a label value of 22, the router will know to forward that particular packet to interface 22. The fundamental axiom for label tables is that small is better. This reduces the latency, simplifies the routers’ lookup procedures, and also simplifies the overall network. Simplicity is further realized based on the fact that MPLS is a connection-oriented service. This is the antithesis of the current connection-less Internet! Packets are routed through today’s Internet on a hop-by-hop basis. Each individual router reads a packet’s destination IP address and then searches its entire Internet routing table (which currently has approximately 120,000 entries) to determine the next router hop for that particular packet. This process is repeated at every intermediate router hop, for every single IP packet.

Packets with the same destination address do not necessarily follow the same path since independent decisions are made at each router hop. MPLS, on the

other hand, establishes a fixed path for communications based on the first packet of a stream. Once the path is established, all subsequent packets will follow the same route (called a *Label Switch Path* or *LSP*). Routers will then only need to switch the packets along a predefined LSP, thereby simplifying the process and expediting delivery.

5.4 PROPOSED SYSTEM ARCHITECTURE

Our proposed architecture consists of four systems viz., Ingress LER (Client), Egress LER (Server), and 2 LSRs' (performs the process of switching between PPP, Ethernet, ATM services) as explained below.



Module I : Linux Kernel with MPLS Support:

The goals of this module are to provide MPLS support for the Linux Kernel. This capability is enabled by patching the Linux-kernel file with MPLS support to the existing kernel. In order to perform the desired operations with the new kernel, the following functions incorporated.

- Resolving the hostname on providing the IP address and vice-versa → `resolve_it ()`
- Parsing the command line arguments → `parse_nh_info()`
- Label look up and forwarding, which involves swapping and pushing
label → `fill_label()`
- Generating and executing instructions such as POP, PUSH, PEEK, FWD for debug messages → `fill_instructions()`

The command line argument options provided here are,

A → Add Modifier, instructs the above functions to add a label as specified.

D → Delete Modifier instructs the above functions to delete a label as Specified.

B → Bind Modifier instructs the above functions to bind the label with the selected key.

d → Toggle debug, change the status of the debug option

L → Set Label Space instructs the above functions to create and set the label space for the specified interface.

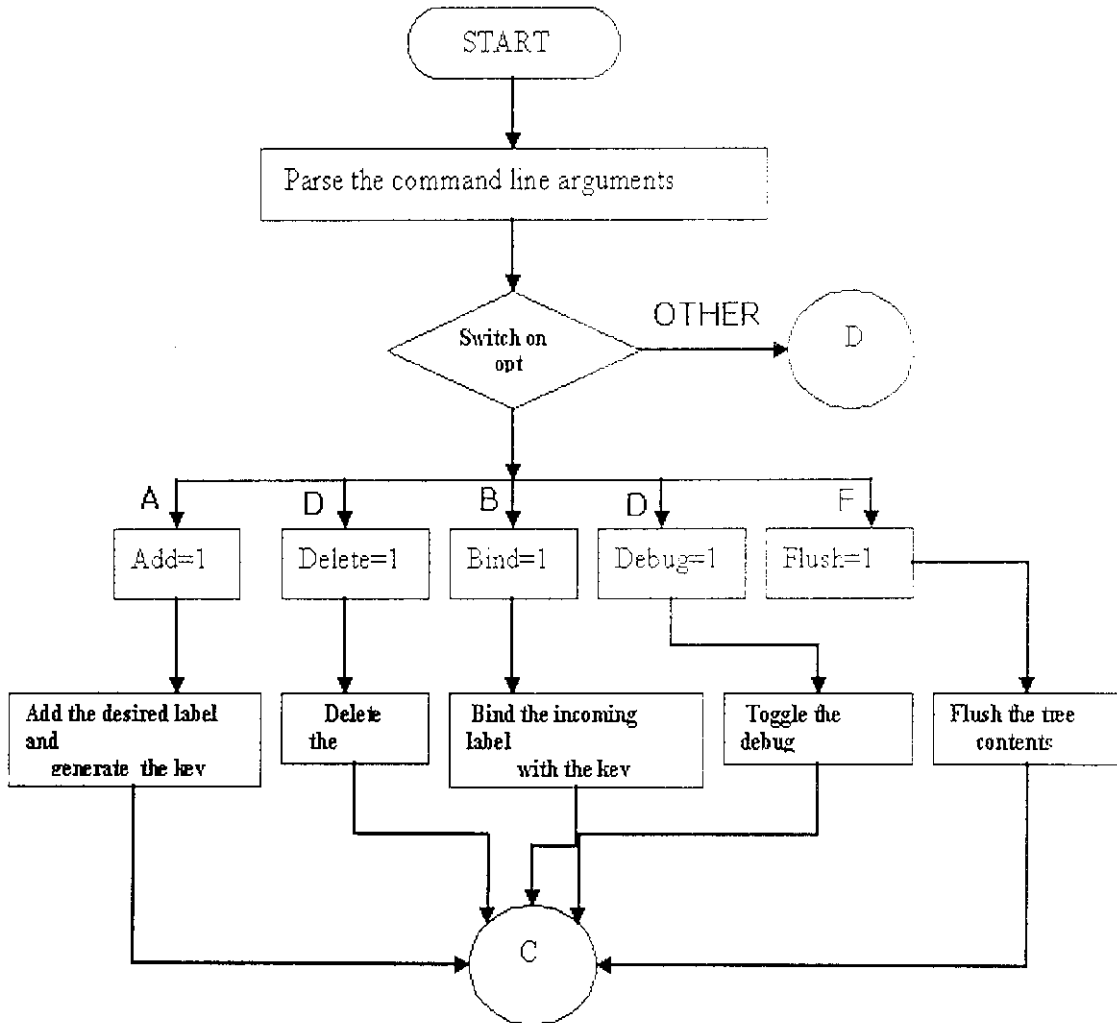
I → Create/ Delete an incoming label

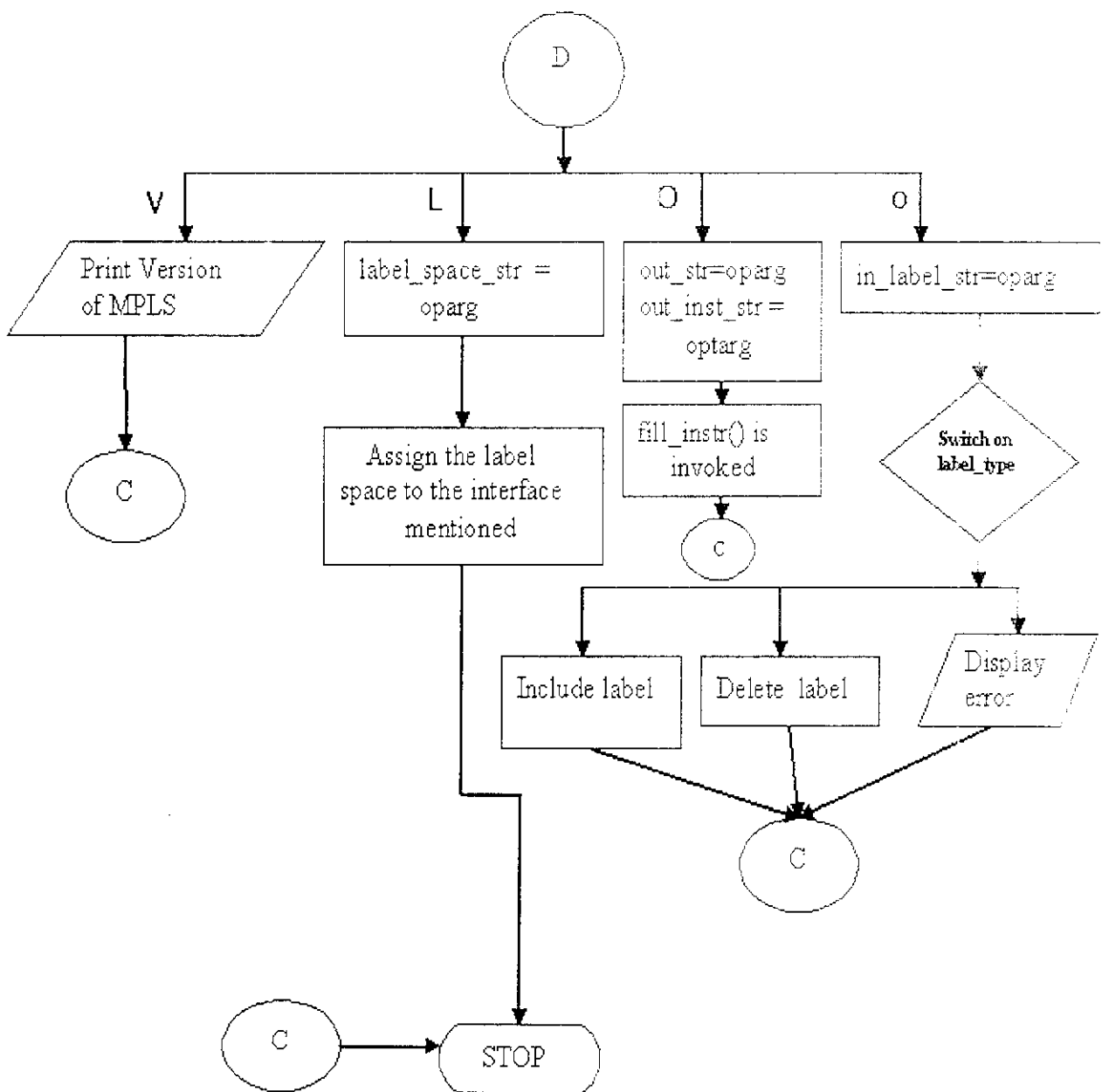
O → Generate label with a key

o → To generate the opcodes, PUSH, POP, PEEK

V → Output the MPLS kernel version at the compile time

FLOWCHART





ALGORITHM

The flow of the algorithm is as follows.

- 1) Start.
- 2) Get the parameters in command line as specified.
- 3) Parse the arguments thus obtained from command line and store each token separately.
- 4) Analyze all the tokens and perform the different functions associated with it, display errors otherwise.
- 5) Output the debug messages at the appropriate instances
- 6) Stop

This implementation makes the MPLS router hot configurable. By example, a Linux box running as an IP router becomes an MPLS router once we load the MPLS module. If we unload the MPLS module (for debugging or modifications) the system will remain IP router. This reconfiguration can be done without recompiling the kernel or restarting the system.

Module II : PPP over MPLS

The goal of this module is to enable PPP support over MPLS. The actions performed in the first module allocate memory space for the Point-to-Point Protocol. The label is obtained amongst the tokens in the command line, then the label is verified for correct values and if it is within the desired range. The label is then copied to the space in memory corresponding to the PPP interfaces' label space.

Once a PPP packet is received at the kernel, the label in the PPP interface's memory is encapsulated to the packet as shown in the figure below.

PPP Header	Label (Shim Header) 32 bits	Layer 3 Header
-------------------	--	-----------------------

Thus the label encapsulated packet follows the next-hop entry according to the entries in Label Forwarding Information Base (LFIB) via RS-232 cable. This takes place only at the LSR's.

Module III : Ethernet Over MPLS

This module aims at encapsulating label(s) with the packet arriving via Ethernet interface. A node in the MPLS domain is configured by the first module of this project, in such a way to allocate memory space for the Ethernet interface in the Linux kernel. Once a label- add instruction is executed at that node, the label assigned is stored in the memory space allocated for the Ethernet interface. Thus when an Ethernet packet is picked up by the kernel, it is encapsulated with the label in the following way.

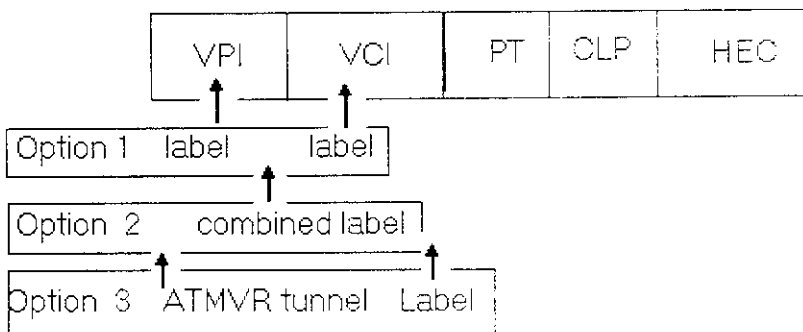
MAC Header	Label (Shim Header) 32 bits	Layer 3 Header
-------------------	--	-----------------------

Thus the packet that arrives in the Ethernet interface is encapsulated with a label as specified in Label Forwarding Information Base (LFIB) in a particular node. As per the instructions available for each label in LIB, the destiny of this label encapsulated packet is determined.

Module IV: ATM over MPLS

In general there are three ways to encapsulate MPLS label over ATM. One way is to place the label in the VCI field or in the VPI field. In this project the second way is adopted where in the label is assigned in the place of VPI and VCI fields combined. The third option is to integrate the tunneling and the MPLS concepts.

ATM LSR constrained by the cell format imposed by existing ATM standards



ATM Over MPLS- Encapsulation

Module V: Providing different classes of service using MPLS - enhanced Software Switch

There are three different classes of services defined in the system. They are provided upon request and authorization of the respective client. The above explained four modules are **integrated in this step and the different classes of service are provided**. The following algorithm illustrates the function of this module.

ALGORITHM:

- 1) Start
- 2) Client (Ingress LER) sends request to the server (egress LER) for a service.
- 3) The requested is gated via the LSR 1 and routed to the LSR 2 and its destination as obvious is Egress-LER
- 4) Once the LSR1 receives the request packet, upon label look up available in the packet, it determines the LSP to be followed and the class of service to be provided. (Gold, Silver, Bronze).
- 5) LSR1 swaps the incoming label with another label, if the former is valid and that it contains a entry in the Label Forwarding Information Base (LFIB).
- 6) The above is repeated at LSR2 which forwards the packets to Egress - LER, thereby the request reaches the Server via MPLS switching and routing operations.
- 7) The Egress-LER in turn performs the aforesaid operations as defined in its Label Forwarding Information Base (LFIB) and proceeds as per the instructions in LFIB.
- 8) The reply to the client is also sent via the LSR's which are defined as gateway for all the packets transferred between Ingress-LER and Egress- LER.
- 9) Stop.

5.6 IMPLEMENTATION DETAILS

Our proposed system is implemented as given below.

The Linux kernel was added the MPLS capabilities using the Patch files. The routes and the labels to be added were statically defined. PPP support for the newly formed Linux kernel was implemented. The above set up was tested for MPLS label encapsulation in PPP packets using the RS-232 cables connecting 2 LSR's end to end. The next step involved enabling Ethernet over MPLS. The newly formed kernel was analyzed and tested if it added the MPLS 32 bit header information for the outgoing packets as defined in the Label Forwarding Information Base (LFIB), through the cross-connect Ethernet 10/100 Mbps cables. A sample file was successfully downloaded via both the PPP interface and the Ethernet interface and a comparative study was made. The next step was to implement ATM over MPLS, which was achieved by the options in the configurations file. Sample packets were tested if MPLS encapsulation was taking place over the ATM interface. Finally the setup was integrated for MPLS label- lookup and forwarding. The end nodes in MPLS domain were the Edge Routers , Ingress LER (Client) which requests for a service from the another edge router nameiy the Egress LER (server). The 2 LSR's in between the edge routers, acted as switches providing 3 different classes of service, namely Gold, Silver, Bronze (ATM,Ethernet,PPP) depending on the incoming clients' request. The above setup was tested for various files of different sizes. The Speed, Time and size of the files were recorded.

The comparative study adapts the following steps.

- The number of files to be compared is specified.
- The debug messages of MPLS were stored and analyzed.
- These messages were examined for the correct labels assigned to appropriate class of service.
- The speed, time and size of the files were recorded.

6. RESULTS

The four nodes mentioned in the proposed architecture were configured with the following routing tables.

NODE A

Kernel IP routing table (Ingress-LER); CLIENT

Ethernet 0 interface IP Address: 172.16.10.17 (Low priority User)

Ethernet 0 interface IP Address: 172.16.10.18 (Medium priority user)

Ethernet 0 interface IP Address: 172.16.10.16 (High Priority User)

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
172.16.10.23	172.16.10.20	255.255.255.255	UGH	0	0	0	eth0
172.16.10.24	172.16.10.20	255.255.255.255	UGH	0	0	0	eth0
172.16.10.25	172.16.10.20	255.255.255.255	UGH	0	0	0	eth0
192.168.34.35	172.16.10.20	255.255.255.255	UGH	0	0	0	eth0
192.168.34.34	172.16.10.20	255.255.255.255	UGH	0	0	0	eth0
172.16.0.0	0.0.0.0	255.255.0.0	U	0	0	0	eth0
127.0.0.0	0.0.0.0	255.0.0.0	U	0	0	0	lo

NODE B

Kernel IP routing table for LSR 1

Ethernet 0 interface IP Address : 172.16.10.19

Ethernet 1 interface IP Address : 172.16.10.20

PPP Interface IP Address : 172.16.10.21

ATM Interface IP Address : 192.168.34.34

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
192.168.34.35	192.168.34.34	255.255.255.255	UGH	0	0	0	atm0
172.16.10.23	172.16.10.19	255.255.255.255	UGH	0	0	0	eth0
172.16.10.22	172.16.10.21	255.255.255.255	UGH	0	0	0	ppp0
172.16.10.22	0.0.0.0	255.255.255.255	UH	0	0	0	ppp0
172.16.10.18	172.16.10.20	255.255.255.255	UGH	0	0	0	eth1
172.16.10.17	172.16.10.20	255.255.255.255	UGH	0	0	0	eth1
172.16.10.16	172.16.10.20	255.255.255.255	UGH	0	0	0	eth1
172.16.10.25	192.168.34.35	255.255.255.255	UGH	0	0	0	atm0
172.16.10.24	172.16.10.19	255.255.255.255	UGH	0	0	0	eth0
172.16.0.0	0.0.0.0	255.255.0.0	U	0	0	0	eth0
172.16.0.0	0.0.0.0	255.255.0.0	U	0	0	0	eth1
192.168.0.0	192.168.34.34	255.255.255.0	U	0	0	0	atm0
127.0.0.0	0.0.0.0	255.0.0.0	U	0	0	0	lo
0.0.0.0	172.16.10.22	0.0.0.0	UG	0	0	0	ppp0

NODE C

Kernel IP routing table for LSR 2

Ethernet 0 interface IP Address : 172.16.10.23

Ethernet 1 interface IP Address : 172.16.10.24

PPP Interface IP Address : 172.16.10.22

ATM Interface IP Address : 192.168.34.35

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
192.168.34.34	192.168.34.35	255.255.255.255	UGH	0	0	0	atm0
172.16.10.21	172.16.10.22	255.255.255.255	UGH	0	0	0	ppp0
172.16.10.21	0.0.0.0	255.255.255.255	UH	0	0	0	ppp0
172.16.10.20	172.16.10.23	255.255.255.255	UGH	0	0	0	eth0
172.16.10.19	172.16.10.23	255.255.255.255	UGH	0	0	0	eth0
172.16.10.18	172.16.10.20	255.255.255.255	UGH	0	0	0	eth0
172.16.10.17	172.16.10.21	255.255.255.255	UGH	0	0	0	ppp0
172.16.10.16	192.168.34.34	255.255.255.255	UGH	0	0	0	atm0
172.16.10.25	172.16.10.24	255.255.255.255	UGH	0	0	0	eth1
172.16.0.0	0.0.0.0	255.255.0.0	U	0	0	0	eth0
172.16.0.0	0.0.0.0	255.255.0.0	U	0	0	0	eth1
192.168.0.0	192.168.34.35	255.255.255.0	U	0	0	0	atm0
127.0.0.0	0.0.0.0	255.0.0.0	U	0	0	0	lo
0.0.0.0	172.16.10.21	0.0.0.0	UG	0	0	0	ppp0

NODE D

Kernel IP routing table (Egress-LER); SERVER

Ethernet 0 interface IP Address: 172.16.10.25

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
172.16.10.23	172.16.10.24	255.255.255.255	UGH	0	0	0	eth0
172.16.10.22	172.16.10.24	255.255.255.255	UGH	0	0	0	eth0
172.16.10.16	172.16.10.24	255.255.255.255	UGH	0	0	0	eth0
192.168.34.34	172.16.10.24	255.255.255.255	UGH	0	0	0	eth0
192.168.34.35	172.16.10.24	255.255.255.255	UGH	0	0	0	eth0
172.16.10.21	172.16.10.24	255.255.255.255	UGH	0	0	0	eth0
172.16.10.20	172.16.10.24	255.255.255.255	UGH	0	0	0	eth0

```

172.16.10.19 172.16.10.24 255.255.255.255 UGH 0 0 0 eth0
172.16.10.18 172.16.10.24 255.255.255.255 UGH 0 0 0 eth0
172.16.10.17 172.16.10.24 255.255.255.255 UGH 0 0 0 eth0
172.16.0.0 0.0.0.0 255.255.0.0 U 0 0 0 eth0
127.0.0.0 0.0.0.0 255.0.0.0 U 0 0 0 lo

```

MPLS Settings sample for LSR1

[mpls-linux]: displaying MPLS settings for Linsys88 (LSR 1)

```

mpls_in {
0x0001e005 100/8800/0 gen 30 1 1 POP FWD(0x00000002)
0x00032001 76/6688/0 gen 50 0 1 POP FWD(0x00000004)
0x0003c005 66/5808/0 gen 60 1 1 POP FWD(0x00000003)
0x00050009 22/1936/0 gen 80 2 1 POP FWD(0x00000005)
0x00060010 10/1000/0 gen 90 1 1 POP FWD(0x00000006)
0x00070009 23/2044/0 gen 110 3 1 POP FWD(0x00000007)

};

mpls_labelspace {
eth0 0 13
eth1 1 10
ppp0 2 11
atm0 3 12
};

mpls_out {
0x00000002 100/8400/0 2 PUSH(gen 35) SET(eth0,172.16.10.23)
0x00000003 66/5544/0 2 PUSH(gen 65) SET(ppp0,172.16.10.22)
0x00000004 76/6384/0 2 PUSH(gen 55) SET(eth1,172.16.10.18)

```

```
0x00000005 22/1848/0 2 PUSH(gen 85) SET(eth1,172.16.10.17)
0x00000006 33/1425/0 2 PUSH(gen 95) SET(atm0,192.168.34.35)
0x00000007 22/1320/0 2 PUSH(gen 115) SET(eth1,172.16.10.16)
```

```
};
```

```
mpls_version {
```

```
01010702
```

```
};
```

```
done
```

SAMPLE DEBUG MESSAGES AT THE INGRESS LER

```
Feb 24 17:25:46 Linsys89 kernel: mpls_send: GEN
```

```
Feb 24 17:25:46 Linsys89 kernel: mpls_send: using hh
```

```
Feb 24 17:25:46 Linsys89 kernel: mpls_skb_dump: from eth0 with len 78 (348)
headroom=86 tailroom=28
```

```
Feb 24 17:25:46 Linsys89 kernel:
06000000340000003480040834800408*c0000000c0000000050000000400000
003000000f4000000f4800408f480040813000000130000000400000001000000
0100000000000000080040800800408863b01000000{0002441193980002441
193848847|0001e1404500003c24cc40004006a982ac100a12#ac100a190403001
53795b25b00000000a00216c040400000020405b00402080a000196b50000000
00103030008}
```

```
Feb 24 17:25:46 Linsys89 kernel: mpls_send: mpls_send result 0
```

```
Feb 24 17:25:46 Linsys89 kernel: mpls_output2: exit
```

```
Feb 24 17:25:46 Linsys89 kernel: mpls_output: exit
```

```
Feb 24 17:25:46 Linsys89 kernel: mpls_rcv: enter
```

```
Feb 24 17:25:46 Linsys89 kernel: mpls_skb_dump: from eth0 with len 64 (252)
headroom=32 tailroom=0
```

Feb 24 17:25:46 Linsys89 kernel:
3c343e4665622032342031373a32353a3235*000244119384000244119398884
7{#|0003713e4500003c000040004006ce70ac100a19ac100a1200150403399ba3
003795b25ca0121690b11b0000020405b00402080a0001b2a6000196b5010303
0001}

Feb 24 17:25:46 Linsys89 kernel: mpls_input: enter

Feb 24 17:25:46 Linsys89 kernel: mpls_input:
label=0,label=55,exp=0,B.O.S=1,TTL=62

Feb 24 17:25:46 Linsys89 kernel: mpls_skb_dump: from eth0 with len 64 (252)
headroom=32 tailroom=0

Feb 24 17:25:46 Linsys89 kernel:
3c343e4665622032342031373a32353a3235*000244119384000244119398884
7{#|0003713e4500003c000040004006ce70ac100a19ac100a1200150403399ba3
003795b25ca0121690b11b0000020405b00402080a0001b2a6000196b5010303
0001}

Feb 24 17:25:46 Linsys89 kernel: mpls_in_info_hold: enter

Feb 24 17:25:46 Linsys89 kernel: mpls_in_info_hold: new count 2

Feb 24 17:25:46 Linsys89 kernel: mpls_in_info_hold: exit

Feb 24 17:25:46 Linsys89 kernel: mpls_input: opcode POP

Feb 24 17:25:46 Linsys89 kernel: mpls_input: opcode PEEK

Feb 24 17:25:46 Linsys89 kernel: mpls_in_info_release: enter

Feb 24 17:25:46 Linsys89 kernel: mpls_in_info_release: new count 1

Feb 24 17:25:46 Linsys89 kernel: mpls_in_info_release: exit

Feb 24 17:25:46 Linsys89 kernel: mpls_input: delivering

Feb 24 17:25:46 Linsys89 kernel: mpls_finish: enter

Feb 24 17:25:46 Linsys89 kernel: mpls_finish: exit

Feb 24 17:25:46 Linsys89 kernel: mpls_dlv: setting ttl 62

Feb 24 17:25:46 Linsys89 kernel: mpls_output: enter

Feb 24 17:25:46 Linsys89 kernel: mpls_output2: enter

Feb 24 17:25:46 Linsys89 kernel: mpls_output2: opcode PUSH

Feb 24 17:25:46 Linsys89 kernel: mpls_push: enter

Feb 24 17:25:46 Linsys89 kernel: mpls_push: creating larger packet

Feb 24 17:25:46 Linsys89 kernel: mpls_push: dump old packet

Feb 24 17:25:46 Linsys89 kernel: mpls_skb_dump: from net stack with len 52
(316) headroom=108 tailroom=0

```
Feb          24          17:25:46          Linsys89          kernel:
06000000340000003480040834800408*c0000000c0000000050000000400000
003000000f4000000f4800408f480040813000000130000000400000001000000
0100000000000000080040800800408863b010000000002441193980002441:
938488470001e1404500003c{|4500003424ef40004006a989ac100a12ac100a19
#040300153795b25c399ba301801016c0dfac00000101080a000196b50001b2a6
01}
```

Feb 24 17:25:46 Linsys89 kernel: mpls_push: dump new packet

Feb 24 17:25:46 Linsys89 kernel: mpls_skb_dump: from net stack with len 52
(284) headroom=32 tailroom=44

```
Feb          24          17:25:46          Linsys89          kernel:
3c343e4665622032342031373a32353a3235206b65726e656c3a206d706c735f
{|4500003424cf40004006a989ac100a12ac100a19#040300153795b25c399ba30
1801016c0dfac00000101080a000196b50001b2a669}
```

Feb 24 17:25:46 Linsys89 kernel: mpls_push: using tailroom

Feb 24 17:25:46 Linsys89 kernel: mpls_push: done using tailroom

Feb 24 17:25:46 Linsys89 kernel: mpls_push: exit

Feb 24 17:25:46 Linsys89 kernel: mpls_output2: opcode SET

Feb 24 17:25:46 Linsys89 kernel: mpls_send: output device = eth0

Feb 24 17:25:46 Linsys89 kernel: mpls_finish: enter

Feb 24 17:25:46 Linsys89 kernel: mpls_finish: exit

Feb 24 17:25:46 Linsys89 kernel: mpls_send: GEN

Feb 24 17:25:46 Linsys89 kernel: mpls_send: using hh

Feb 24 17:25:46 Linsys89 kernel: mpls_skb_dump: from eth0 with len 70 (284)
headroom=18 tailroom=40

```
Feb          24          17:25:46          Linsys89          kernel:
3c343e4665622032342031373a32353a0000{000244119398000244119384884
```

7|0001e1404500003424ef40004006a989ac100a12#ac100a19040300153795.25
c399ba301801016c0dfac00000101080a000196b50001b2a66f}

Feb 24 17:25:46 Linsys89 kernel: mpls_send: mpis_send result 0

Feb 24 17:25:46 Linsys89 kernel: mpls_output2: exit

Feb 24 17:25:46 Linsys89 kernel: mpls_output: exit

COMPARISON BASED ON THE RESULTS OBTAINED

**PPP
Interface**

**Ethernet
Interface**

**ATM
Interface**

FILE SIZE	TIME TAKEN	SPEED	TIME TAKEN	SPEED	TIME TAKEN	SPEED
	(Seconds)	(Kbytes/s)	(Seconds)	(Kbytes/s)	(Seconds)	(Kbytes/s)
3 KB	0.097	36	0.00079	4500	0.00063	6100
1.24 MB	120	11	0.16	8100	0.101	16000
2.59 MB	250	11	0.4	6600	0.15	16000
3.25 MB	360	11	0.35	9600	0.191	16000
4.8 MB	620	11	0.45	11000	0.284	16000

As seen from the results above the idea of selecting one of the three different interfaces PPP, Ethernet, ATM, defined in the system architecture provide their typical class of service. These are incorporated in one single switch.

As the packets arrives at the switch they will be filtered and classified based on their priority; PPP interface is used for simple links which transport packets between two peers, thus providing the bronze class of service. For mission critical, guaranteed services the Ethernet interface is used and thus the Silver class of service is endowed with. High-Priority, low latency, Premium -class Gold service is bestowed upon the typical user with the highest priority in the network.

The results are thus compared above from the time taken for different files obtained through the usage of file transfer protocol (FTP). The tabular column clearly indicates the differences in class of services provided through the project for different clients.

7. CONCLUSION

The internet protocol (IP) has enabled a comprehensive network between a never-ending variety of systems and communication, transmission media. From the basic e-mail, web browsing; which have become part of the daily routine, to the emerging video conferencing, phone radio and televisions, all indicate that they congregate on IP.

There is a clear need for relatively simple coarse method of providing different classes of service for internet traffic, support various types of applications, and specific business requirements.

Multi Protocol Label Switching is an elegant solution to these issues. MPLS is destined to provide a new technical foundation for the next generation of multi-user, multi-service inter-networks. The promise is for higher performance, another order of magnitude increase in scalability, improved and expanded functionality, and the flexibility to match the user's quality of service requirements more closely. While the expansion of the Internet has been a major driver for development of label switching, it is not the only, or even the most important factor.

Label switching provides significant improvements in the packet forwarding process by simplifying the processing, avoiding the need to duplicate header processing at every step in the path, and creating an environment that can support controlled QoS. Several vendor-specific solutions exist today and IETF MPLS standards are expected within a year. Deployment of MPLS allows a closer integration of IP and ATM, supports service convergence, and offers new opportunities for traffic engineering and VPN support.

By adding fixed size labels to packet flows, the way we add ZIP codes to mail to help with sorting, packet processing performance can be improved. CoS

controls can be more easily applied and very large global public networks can be built. All of this results in better networks with more functions at lower cost. MPLS is a new technology that is just beginning to be recognized as beneficial. The basic standards will soon be completed and products will be delivered quickly afterward. It is fully expected that MPLS will see wide-spread deployment in both public and private IP networks, paving the way for true convergence of telephony, video, and computing services.

FUTURE WORK

1. MPLS is implemented in this project with static label assignments. Instead the LDP (Label Distribution Protocol) can be implemented to perform dynamic label assignments, upon request and requirements in the given network.
2. The switching technology implemented here uses a software version of MPLS. The performance of the same setup can be enhanced using a hardware switch. This will enable better performance in terms of speed, usage and cost.

8. REFERENCES

Journals

1. Rekhter, Y.B., David E., Rosen, G. Swallow, D. Farinacci, and D. Katz. "Tag Switching Architecture Overview". In Proceedings of the IEEE 82, no.12, December 1997, 1973-1983.
2. Newman, Peter, Minshall, Greg, Lyon, Thomas. "IP Switching-ATM". IEEE/ACM Transactions on Networking. 1998. 6(2).

Websites

1. www.agilent.com
2. www.cisco.com
3. www.cnnovatenetworks.com
4. www.ethereal.com
5. www.icc.org
6. www.ietf.org
7. www.marconi.com
8. www.mplsforum.org
9. www.mplssrc.com
10. www.netplane.com
11. www.riverstonenet.com
12. www.techguide.com
13. www.trillium.com

Books

1. L. Peterson and B. Davie, *Computer Networks: A systems approach*, p. 163, Morgan & Kaufmann, USA, 1996.

2. Andrew.S.Tanenbaum, Computer Networks, Third Edition, Prentice Hall of India, 2001.
3. Black, Ulysses. "ATM: Foundation for Broadband Networks". Prentice Hall, NJ. 1995.
4. Michael K.Johnson, Erik.W.Troan, Linux Application Development, Low priced Edition Pearson Education India.
5. Cary Lu, The Race for Bandwidth: Understanding Data Transmission, Prentice Hall Of India, 2001.
6. Sumitabha Das, Unix Concepts and Applications, 2nd edition, Tata-McGraw Hill, 1997.
7. Mathias Hein, David Griffiths, Switching Technology in the Local Area Network, from LAN to switched LAN to Virtual LAN, Thomson Computer Press, 1997.

MPLSADM.C

```
/* INCLUDING THE REQUIRED HEADER FILES */
```

```
#include <ctype.h>
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <unistd.h>
```

```
#include <string.h>
```

```
#include <netdb.h>
```

```
#include <sys/socket.h>
```

```
#include <netinet/in.h>
```

```
#include <net/if.h>
```

```
#include <arpa/inet.h>
```

```
#include <linux/mpls.h>
```

```
#include <sys/ioctl.h>
```

```
#include <linux/atm.h>
```

```
#include <asm/types.h>
```

```
int fd = 0;
```

```
char verbose = 0;
```

```
unsigned int resolve_it(char *host);
```

```
int parse_nh_info(struct mpls_nexthop_info *mni, char **arg, int start);
```

```
int fill_label(struct mpls_label *lbl, char **args, int start);
```

```
int fill_instructions(struct mpls_instruction_req *mir, char** arg, int start,  
int num);
```



```
/* GENERATION OF ERROR MESSAGES */
```

```
void usage() {  
    fprintf(stderr,"usage: mplsadm [ADBUdhvT:L:f:O:i:o:2:]\n\n");  
    fprintf(stderr,"-A add modifier\n");  
    fprintf(stderr,"-B bind modifier\n");  
    fprintf(stderr,"-D delete modifier\n");  
    fprintf(stderr,"-U unbind modifier\n");  
    fprintf(stderr,"-d toggle debug\n");  
    fprintf(stderr,"-h this message\n");  
    fprintf(stderr,"-v verbose info\n");  
    fprintf(stderr,"-F flush all ILMs and NHLFEs\n");  
    fprintf(stderr,"-L <interface name>:<label space> set the label space  
for"  
        " an interface (-1 disables)\n");  
    fprintf(stderr,"-I <gen | atm | fr>:<label>:<label space> create | delete an"  
        " incoming label\n");  
    fprintf(stderr,"-O <key> (create with a key of 0)\n");  
    fprintf(stderr,"-o <opcode:opcode_data>+\n");  
    fprintf(stderr,"-V output the MPLS kernel version used at compile  
time\n");  
    fprintf(stderr,"\\n\n");  
}
```

```
/* FUNCTION TO GET THE NEXT HOP INFORMATION */
```

```

int parse_nh_info(struct mpls_nexthop_info *mni, char **arg, int start) {
    struct ifreq ifr;
    int result;
    int len;
    int i = 0;

    len = strlen(arg[start]);
    i++;
    if(len > 0 && len < 10) {
        memset(&ifr, 0, sizeof(struct ifreq));
        strncpy(ifr.ifr_name, arg[start], len+1);
        if((result = ioctl(fd, SIOCGIFINDEX, &ifr)) == 0) {
            struct sockaddr_in sin;
            memset(&sin, 0, sizeof(sin));
            mni->mni_if = ifr.ifr_ifindex;
            if(arg[start+1] && !strcmp(arg[start+1], "ipv4")) {
                i++;

                if(verbose) fprintf(stderr, "Nexthop protocol: ipv4\n");
                sin.sin_addr.s_addr = resolve_it(arg[start+2]);
                sin.sin_family = AF_INET;
                i++;
            }
            memcpy(&mni->mni_addr, &sin, sizeof(struct sockaddr));
        } else {
            fprintf(stderr, "Unable to resolve ifindex for %s\n", ifr.ifr_name);
        }
    }
    return i;
}

```

```
}
```

```
/* FUNCTION TO ADD THE LABEL AT THE DESIRED POSITON  
OF THE IP PACKET */
```

```
int fill_label(struct mpls_label *lbl,char **args,int start) {
```

```
int result = 0;
```

```
if(verbose) fprintf(stderr,"Label type: %s\n",args[start]);
```

```
/* IDENTIFYING THE TYPE OF LABEL */
```

```
if(!strncmp(args[start],"atm",3)) {
```

```
char *vpi_str,*vci_str;
```

```
int vpi,vci;
```

```
vpi_str = strtok(args[start+1],"/");
```

```
vci_str = strtok(NULL,"\\0");
```

```
if(vpi_str && vci_str) {
```

```
vpi = atoi(vpi_str);
```

```
vci = atoi(vci_str);
```

```
if(vpi > -1 && vpi < ATM_MAX_VPI && vci > -1 && vci <  
ATM_MAX_VCI) {
```

```
lbl->ml_type = MPLS_LABEL_ATM;
```

```
lbl->u.ml_atm.mla_vpi = vpi;
```

```
lbl->u.ml_atm.mla_vci = vci;
```

```
 }
```

```
}
```

```
result = 1;
```

```
}
```

```
else if(!strncmp(args[start],"gen",3)) {
```

```

lbl->ml_type = MPLS_LABEL_GEN;
lbl->u.ml_gen = atol(args[start+1]);
result = 1;
} else if(!strcmp(args[start],"fr",2)) {
    lbl->ml_type = MPLS_LABEL_FR;
    lbl->u.ml_fr = atol(args[start+1]);
    result = 1;
} else if(!strcmp(args[start],"key",3)) {
    lbl->ml_type = MPLS_LABEL_KEY;
    lbl->u.ml_key = strtol(args[start+1],NULL,0);
    result = 1;
}
return result;
}

```

/* FUNCTION TO GIVE OUT THE SUITABLE INSTRUCTIONS */

```

int fill_instructions(struct mpls_instruction_req *mir, char** arg,int
start,
int num) {
    struct ifreq ifr;
    int length = 0;
    int result;
    int i;

    for(i=start;i<(num+start);i++) {
        if(verbose) fprintf(stderr,"Instruction: %s\n",arg[i]);
        if(!strcmp("pop",arg[i],3)) {
            mir->mir_instruction[length].mir_opcode = MPLS_OP_POP;

```

```

} else if(!strncmp("peek",arg[i],4)) {
    mir->mir_instruction[length].mir_opcode = MPLS_OP_PEEK;
} else if(!strncmp("push",arg[i],4)) {
    i++;
    mir->mir_instruction[length].mir_opcode = MPLS_OP_PUSH;
    if(!(result=fill_label(&(mir->mir_instruction[length].mir_data.push),arg,i)))
    {
        fprintf(stderr,"Error while parsing label\n");
        exit(-1);
    }
    i += result;
}
if(!(result = fill_label(&(mir->mir_instruction[length].mir_data.fwd),
    arg,i))) {
    fprintf(stderr,"Error while parsing label\n");
    exit(-1);
}
i += result;
}

if(verbose) fprintf(stderr,"Length: %d\n",length);
return 0;
}

```

```

unsigned int resolve_it(char *host) {
    unsigned int result = -1;
    struct hostent *hp;

```

```

if(isdigit(host[0])) {
    result = inet_addr(host);
} else {
    if((hp = gethostbyname(host)) {
        memcpy(&result, hp->h_addr, sizeof(unsigned int));
    }
}
return result;
}

```

```

/* FUNCTION TO PARSE THE ARGUMENT IN THE COMMAND
LINE */

```

```

int parse_args(int argc, char **argv, char *args) {
    int i = 1;

    argv[0] = strtok(args, ":");
    while((argv[i] = strtok(NULL, ":"))) {
        i++;
        if(i > (argc - 1)) {
            return -i;
        }
    }
    return i;
}

```

```

/* MAIN */

```

```

int main(int argc, char **argv) {

```

```
struct mpls_instruction_req mir_req;
struct mpls_labelspace_req mls_req;
struct mpls_out_label_req mol_req;
struct mpls_in_label_req mil_req;
struct mpls_xconnect_req mx_req;
```

```
struct ifreq ifr;
```

```
char *in_instr_str = NULL;
char *out_instr_str = NULL;
char *in_label_str = NULL;
char *out_str = NULL;
char *proto_str = NULL;
char *mtu_str = NULL;
char *label_space_str = NULL;
char *larg[1024];
int result = -1;
int num = 0;
int opt;
```

```
char delete = 0;
char add = 0;
char bind = 0;
char unbind = 0;
char debug = 0;
char flush = 0;
```

```
fd = socket(AF_INET,SOCK_DGRAM,0);
```

```

if(fd < 0) {
    perror("Socket");
    exit(fd);
}

memset(&mil_req,0,sizeof(struct mpls_in_label_req));
memset(&mils_req,0,sizeof(struct mpls_labelspace_req));
memset(&mol_req,0,sizeof(struct mpls_out_label_req));
memset(&mx_req,0,sizeof(struct mpls_xconnect_req));
memset(&mir_req,0,sizeof(struct mpls_instruction_req));

while((opt = getopt(argc,argv,"ADBUdhvFT:L:I:O:i:o:m:p:V")) != EOF) {
    switch(opt) {
        case 'A':    /* ADD MODIFIER */
            add = 1;
            break;
        case 'B':    /* BIND MODIFIER */
            bind = 1;
            break;
        case 'D':    /* DELETE MODIFIER */
            delete = 1;
            break;
        case 'U':    /* UNBIND MODIFIER */
            unbind = 1;
            break;
        case 'd':    /* TOGGLE DEBUG */
            debug = 1;
            break;
        case 'h':    /* VERBOSE INFO */

```



```

    verbose = 1;
    break;
case 'F':      /* FLUSH */
    flush = 1;
    break;
case 'L':     /* <interface name>:<label space> set the label space for
*/
    label_space_str = optarg;
    if(verbose) fprintf(stderr,"Label Space input: %s\n",label_space_str);
    break;
case 'o':    /* <opcode:opcode_data> */
    out_instr_str = optarg;
    if(verbose) fprintf(stderr,"Out instr input: %s\n",out_instr_str);
    break;
case 'O':    /* O <key> (create with a key of 0) */
    out_str = optarg;
    if(verbose) fprintf(stderr,"Out segment input: %s\n",out_str);
    break;
case 'V':    /* PRINT THE VERSION */
    fprintf(stdout, "\n\tMPLS version %d.%d%d%d\n\n",
        (MPLS_LINUX_VERSION >> 24) & 0xFF,
        (MPLS_LINUX_VERSION >> 16) & 0xFF,
        (MPLS_LINUX_VERSION >> 8) & 0xFF,
        MPLS_LINUX_VERSION & 0xFF);
    break;
case 'h':    /* THIS MESSAGE */
default:
    usage();
    exit(result);

```

```

        break;
    }
}

/* TOGGLE DEBUG */
if(debug) {
    result = ioctl(fd,SIOCMPLSDEBUG,&ifr);
    perror("Debug");
}

/* FLUSH THE ENTIRE CONTENTS OF THE LABEL FORWARD
INFORMATION BASE */
if (flush) {
    result = ioctl(fd,SIOCMPLSILMFLUSH,&ifr);
    result = ioctl(fd,SIOCMPLSNHLFEFLUSH,&ifr);
    perror("Flush");
}

/* ALLOCATE LABEL SPACE FOR THE INTERFACES*/

if(label_space_str) {
    int len;

    num = parse_args(1024,larg,label_space_str);
    if(num == 2) {
        len = strlen(larg[0]);
        if(verbose) fprintf(stderr,"If: %s LS: %s\n",larg[0],larg[1]);
        if(len > 0 && len < IFNAMSIZ) {
            memset(&ifr,0,sizeof(struct ifreq));

```

```

strcpy(ifr.ifr_name,larg[0]);
if((result = ioctl(fd,SIOCGIFINDEX,&ifr)) != 0) {
    perror("SIOCGIFINDEX");
    exit(result);
}
mls_req.mls_ifindex = ifr.ifr_ifindex;
mls_req.mls_labelSpace = atoi(larg[1]);
result = ioctl(fd,SIOCSSLABELSPACEMPLS,&mls_req);
}
}
perror("Label Space");
}

```

```

if(tunnel_str) {
    num = parse_args(1024,larg,tunnel_str);
    if(add && num == 1) {
        strncpy(ifr.ifr_name,larg[0],IFNAMSIZ);
        result = ioctl(fd,SIOCMPLSTUNNELADD,&ifr);
        if(bind) {
            result = ioctl(fd,SIOCGIFINDEX,&ifr);
        }
    } else if((delete && num == 1) || (bind && num == 1)) {
        strcpy(ifr.ifr_name,larg[0]);
        if(delete) {
            result = ioctl(fd,SIOCMPLSTUNNELDEL,&ifr);
        } else {
            result = ioctl(fd,SIOCGIFINDEX,&ifr);
        }
    }
}

```

```

} else {
    fprintf(stderr,"Tunnel: wrong number of paramters(%d)\n",num);
    fprintf(stderr,"Tunnel: -A -T <name>\n");
    fprintf(stderr,"Tunnel: -D -T <name>\n");
    fprintf(stderr,"Tunnel: -B ... -T <name>\n");
    tunnel_str = NULL;
}
}

```

**/* GET THE INCOMMING LABEL FROM DESIRED INTERFACE
AND ASSIGN TO THE CORRESPONDING LABEL SPACE */**

```

if(in_label_str) {
    num = parse_args(1024,larg,in_label_str);
    if(num == 3) {
        fill_label(&mil_req.mil_label,larg,0);
        if (mil_req.mil_label.ml_type == MPLS_LABEL_KEY) {
            fprintf(stderr,"In labels cannot be specified via 'key'");
            exit(-1);
        }
        mil_req.mil_label.ml_index = atoi(larg[2]);
        if(delete) {
            result = ioctl(fd,SIOCMPLSILMDEL,&mil_req);
            perror("In Label del");
        } else if(add) {
            result = ioctl(fd,SIOCMPLSILMADD,&mil_req);
            perror("In Label add");
        }
    } else {

```

```

    fprintf(stderr,"In Label: wrong number of paramters(%d)\n",num);
    in_label_str = NULL;
}
}

```

/* GENERATE THE CORRESPONDING KEY FOR THE LABEL GENERATED AND ACCEPTED */

```

if(out_str) {
    mol_req.mol_label.ml_type = MPLS_LABEL_KEY;
    mol_req.mol_label.u.ml_key = strtol(out_str,NULL,0);

    if(delete) {
        result = ioctl(fd,SIOCMPLSNHLFEDEL,&mol_req);
        perror("Out Segment del");
    } else if(add) {
        result = ioctl(fd,SIOCMPLSNHLFEADD,&mol_req);
        printf("Key: 0x%08x\n",mol_req.mol_label.u.ml_key);
        perror("Out Segment add");
    }
}
}

```

```

if(in_label_str && in_instr_str) {
    num = parse_args(1024,larg,in_instr_str);
    if(num >= 1) {
        mir_req.mir_direction = MPLS_IN;
        if((result = fill_instructions(&mir_req,larg,0,num)) >= 0) {
            memcpy(&mir_req.mir_label,&mil_req.mil_label,sizeof(struct
mpls_label));
            mir_req.mir_index = mil_req.mil_label.mi_index;

```

```

    result = ioctl(fd,SIOCSMPLSININSTR,&mir_req);
}
perror("In Instr");
} else {
    fprintf(stderr,"In Instr: wrong number of parameters(%d)\n",num);
}
}

```

```

if(out_str && out_instr_str) {
    num = parse_args(1024,larg,out_instr_str);
    if(num >= 1) {
        mir_req.mir_direction = MPLS_OUT;
        if((result = fill_instructions(&mir_req,larg,0,num)) >= 0) {
            memcpy(&mir_req.mir_label,&mol_req.mol_label,sizeof(struct
mpls_label));
            result = ioctl(fd,SIOCSMPLSOUTINSTR,&mir_req);
        }
        perror("Out Instr");
    } else {
        fprintf(stderr,"Out Instr: wrong number of parameters(%d)\n",num);
    }
}
}

```

```

if(out_str && tunnel_str) {
    if(ifr.ifr_ifindex == 0) {
        fprintf(stderr,"%s: no such interface\n",tunnel_str);
    } else {
        memcpy(&ifr.ifr_data,&mol_req.mol_label,sizeof(struct mpls_label));
    }
}

```

```
/* BIND AND UNBIND OPERATIONS */
```

```
if(bind && !delete) {  
    result = ioctl(fd,SIOCMPLSTUNNELADDOUT,&ifr);  
    perror("Bind Tunnel2Out add");  
} else if(unbind && !add) {  
    result = ioctl(fd,SIOCMPLSTUNNELDELOUT,&ifr);  
    perror("Bind Tunnel2Out del");  
} else {  
    fprintf(stderr,"Bind Tunnel2Out: No modifier specified\n");  
}  
}  
}
```

```
if(out_str && in_label_str) {  
    if(in_instr_str || out_instr_str) {  
        fprintf(stderr,"Bind In2Out: not at same time as In or Out Instr\n");  
    } else {  
        memcpy(&mx_req.mx_in,&mil_req.mil_label,sizeof(struct  
mpls_label));  
        memcpy(&mx_req.mx_out,&mol_req.mol_label,sizeof(struct  
mpls_label));  
    }  
}
```

```
if(bind && !delete) {  
    result = ioctl(fd,SIOCMPLSXCADD,&mx_req);  
    perror("Bind In2Out add");  
} else if(unbind && !add) {  
    result = ioctl(fd,SIOCMPLSXCDEL,&mx_req);
```

```
    perror("Bind In2Out del");
} else {
    fprintf(stderr,"Bind In2Out: No modifier specified\n");
}
}
}
```

```
if(proto_str && in_label_str && !delete) {
    mil_req.mil_proto = strtol(proto_str,NULL,0);
    result = ioctl(fd,SIOCMPLSILMSETPROTO,&mil_req);
    perror("Set ILM Proto");
}
```

```
if(mtu_str && out_str && !delete) {
    mol_req.mol_mtu = strtol(mtu_str,NULL,0);
    result = ioctl(fd,SIOCMPLSNHLFEDEL,&mol_req);
    perror("Set NHLFE MTU");
}
```

```
return result;
}
```

*** END OF THE PROGRAM */**

CONFIGURING MPLS AND LABEL FORWARD INFORMATION BASE IN THE LSRS AND LERS

INGRESS LER

IP : 172.16.10.16 (ATM)

```
mplsadm -L eth0:0
mplsadm -A -O 0 -o push:gen:90:set:eth0:ipv4:172.16.10.20
iptables -A OUTPUT -t mangle -d 172.16.10.25 -j MPLS --set-key 0x00000004
mplsadm -A -I gen:115:0
```

IP : 172.16.10.18 (SILVER)

```
mplsadm -L eth0:0
mplsadm -A -O 0 -o push:gen:30:set:eth0:ipv4:172.16.10.20
iptables -A OUTPUT -t mangle -d 172.16.10.25 -j MPLS --set-key
0x00000002
mplsadm -A -I gen:55:0
```

IP : 172.16.10.17 (BRONZE)

```
mplsadm -L eth0:0
mplsadm -A -O 0 -o push:gen:60:set:eth0:ipv4:172.16.10.20
iptables -A OUTPUT -t mangle -d 172.16.10.25 -j MPLS --set-key 0x00000003
mplsadm -A -I gen:85:0
```

LSR 1:

```
mplsadm -L eth0:0
```

```
mplsadm -L eth1:1
mplsadm -L ppp0:2
mplsadm -L atm0:3
```

```
mplsadm -A -I gen:30:1
mplsadm -A -O 0 -o push:gen:35:set:eth0:ipv4:172.16.10.23
iptables -A OUTPUT -t mangle -d 172.16.10.25 -j MPLS --set-key
0x00000002
mplsadm -B -I gen:30:1 -O 0x00000002
```

```
mplsadm -A -I gen:60:1
mplsadm -A -O 0 -o push:gen:65:set:ppp0:ipv4:172.16.10.22
iptables -A OUTPUT -t mangle -d 172.16.10.25 -j MPLS --set-key
0x00000003
mplsadm -B -I gen:60:1 -O 0x00000003
```

```
mplsadm -A -I gen:50:0
mplsadm -A -O 0 -o push:gen:55:set:eth1:ipv4:172.16.10.18
iptables -A OUTPUT -t mangle -d 172.16.10.18 -j MPLS --set-key
0x00000004
mplsadm -B -I gen:50:0 -O 0x00000004
```

```
mplsadm -A -I gen:80:2
mplsadm -A -O 0 -o push:gen:85:set:eth1:ipv4:172.16.10.17
iptables -A OUTPUT -t mangle -d 172.16.10.17 -j MPLS --set-key
0x00000005
mplsadm -B -I gen:80:2 -O 0x00000005
```

```
mplsadm -A -I gen:90:1
mplsadm -A -O 0 -o push:gen:95:set:atm0:ipv4:192.168.34.35
```

```
iptables -A OUTPUT -t mangle -d 172.16.10.25 -j MPLS --set-key  
0x00000006  
mplsadm -B -I gen:90:1 -O 0x00000006
```

```
mplsadm -A -I gen:110:3  
mplsadm -A -O 0 -o push:gen:115:set:eth1:ipv4:172.16.10.16  
iptables -A OUTPUT -t mangle -d 172.16.10.16 -j MPLS --set-key  
0x00000007  
mplsadm -B -I gen:110:3 -O 0x00000007
```

LSR 2

```
mplsadm -L eth0:0  
mplsadm -L eth1:1  
mplsadm -L ppp0:2  
mplsadm -L atm0:3
```

```
mplsadm -A -I gen:35:0  
mplsadm -A -O 0 -o push:gen:40:set:eth1:ipv4:172.16.10.25  
iptables -A OUTPUT -t mangle -d 172.16.10.25 -j MPLS --set-key  
0x00000002  
mplsadm -B -I gen:35:0 -O 0x00000002
```

```
mplsadm -A -I gen:65:2  
mplsadm -A -O 0 -o push:gen:70:set:eth1:ipv4:172.16.10.25  
iptables -A OUTPUT -t mangle -d 172.16.10.25 -j MPLS --set-key  
0x00000003  
mplsadm -B -I gen:65:2 -O 0x00000003
```

```
mplsadm -A -I gen:45:1
mplsadm -A -O 0 -o push:gen:50:set:eth0:ipv4:172.16.10.19
iptables -A OUTPUT -t mangle -d 172.16.10.18 -j MPLS --set-key
0x00000004
mplsadm -B -I gen:45:1 -O 0x00000004
```

```
mplsadm -A -I gen:75:1
mplsadm -A -O 0 -o push:gen:80:set:ppp0:ipv4:172.16.10.21
iptables -A OUTPUT -t mangle -d 172.16.10.17 -j MPLS --set-key
0x00000005
mplsadm -B -I gen:75:1 -O 0x00000005
```

```
mplsadm -A -I gen:95:3
mplsadm -A -O 0 -o push:gen:100:set:eth1:ipv4:172.16.10.25
iptables -A OUTPUT -t mangle -d 172.16.10.25 -j MPLS --set-key
0x00000006
mplsadm -B -I gen:95:3 -O 0x00000006
```

```
mplsadm -A -I gen:105:1
mplsadm -A -O 0 -o push:gen:110:set:atm0:ipv4:192.168.34.34
iptables -A OUTPUT -t mangle -d 172.16.10.16 -j MPLS --set-key
0x00000007
mplsadm -B -I gen:105:1 -O 0x00000007
```

EGRESS LER

```
mplsadm -L eth0:0
```

```
mplsadm -A -I gen:40:0
mplsadm -A -O 0 -o push:gen:45:set:eth0:ipv4:172.16.10.24
iptables -A OUTPUT -t mangle -d 172.16.10.18 -j MPLS --set-key 0x00000002
mplsadm -A -I gen:70:0
mplsadm -A -O 0 -o push:gen:75:set:eth0:ipv4:172.16.10.24
iptables -A OUTPUT -t mangle -d 172.16.10.17 -j MPLS --set-key 0x00000003
mplsadm -A -I gen:100:0
mplsadm -A -O 0 -o push:gen:105:set:eth0:ipv4:172.16.10.24
iptables -A OUTPUT -t mangle -d 172.16.10.16 -j MPLS --set-key 0x00000004
```

GLOSSARY

ATM	Asynchronous Transfer Mode
ARIS	Aggregate Route-based IP Switching.
BGP	Border Gateway protocol
CSR	Cell Switching Router
FEC	Forward Equivalence class
IETF	Internet Engineering Task Force.
IP	Internet Protocol.
ISP	Internet Service Provider
LAN	Local Area Network
LDP	Label Distribution Protocol.
LER	Label Edge Router
LSP	Label Switched Path
LSR	Label Switching Router.
LIFB	Label information forwarding base
MPLS	Multiple Protocol Label Switching.
OSPF	Open Shortest Path First.
PPP	Point to point protocol
TOS	Type of Service.
TCP	Transmission Control Protocol.
Unicast	Equivalent to point-to-point transmission.
VoD	Video on Demand