

# **DESIGN OF MANAGEMENT FRAMEWORK FOR MPLS BASED NETWORKS**

## **PROJECT REPORT**

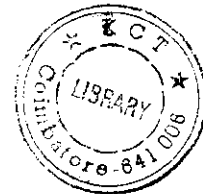
submitted in partial fulfillment of the requirements for the award  
of the degree of **BACHELOR OF ENGINEERING-**  
**Information Technology** of Bharathiar University, Coimbatore.

submitted by

P-888

**V. Kavitha**

**S.Geetha**



under the guidance of

Mrs.L.S.Jayashree , ME ,

Department of Computer Science and Engineering

Kumaraguru College of Technology,

Coimbatore-641006.

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**Kumaraguru College of Technology**

**Coimbatore , Tamilnadu – 641 006.**

**MARCH 2003**

**KUMARAGURU COLLEGE OF TECHNOLOGY**

**COIMBATORE, TAMILNADU-641 006.**

**Department of Computer Science and Engineering**

**CERTIFICATE**

This is to certify that the Project Report entitled

**DESIGN OF MANAGEMENT FRAMEWORK FOR  
MPLS BASED NETWORKS**

is a bonafide record of work done by

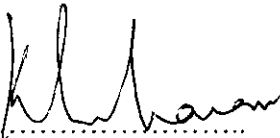
**V.Kavitha(9927S0055)**

**S.Geetha(9927S0506)**

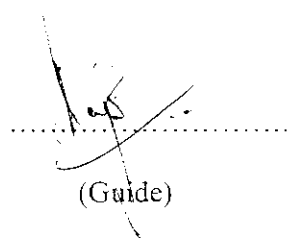
in partial fulfillment of the requirements for the award of the degree of

**BACHELOR OF ENGINEERING – INFORMATION TECHNOLOGY**

of Bharathiar University , Coimbatore during the academic year 2002-2003.



(Head of the Department)



(Guide)

Certified that the candidate was examined by us in the project work

Viva Voce examination held on 20.12.2003 and the

University Register Number is 9927S0055.

.....  
(Internal Examiner)

.....  
(External Examiner)

---

## ACKNOWLEDGEMENT

---

## ACKNOWLEDGEMENT

We take this opportunity to express our gratitude to all those people without whom this project would be incomplete.

We are extremely grateful to **Dr.K.K.Padmanabhan**, Principal , Kumaraguru College of Technology for having given us a golden opportunity to embark on this project.

We are deeply obliged to **Prof.Dr.S.Thangasamy**, Head of the Department of Computer Science and Engineering for his valuable guidance and useful suggestions during the course of this project.

We are indebted to our project guide **Mrs. L.S.Jayashree., ME** and our class advisor **Mrs.N .Chitra Devi .,ME** Kumaraguru College of Technology for their helpful guidance and valuable support given to us throughout the duration of this project.

We also extend our heartfelt thanks to **Mr.K.R.Bhaskaran., M.S.**, Assistant Professor, Kumaraguru College of Technology for providing us support which really helped us make this project a success.

Above all we owe our gratitude to our parents for their support and God Almighty for showering his abundant blessings on us.

## SYNOPSIS

MPLS is a forwarding paradigm applicable to Internet Protocol (IP) networks, which assigns packet flows to Label Switched Paths (LSPs). Unlike traditional hop-by-hop IP routing, packets are classified once at the network edge and then transported over a switched virtual path.

Network Monitoring is an important tool in fault tracing and elimination. It is used to monitor and control the network devices and evaluate the performance of the network.

The main purpose of gathering data and statistics is the evaluation of current network conditions.

Management Information Base is the database of network management information that is used and maintained by a network management protocol such as SNMP. The value of a MIB object can be changed or retrieved by means of SNMP commands, usually through a network management system. MIB objects are organized in a tree structure that include public(standard) and private(proprietary) branches.

The MPLS Label Switching Router MIB (MPLS-LSR-MIB) allows to use the Simple Network Management Protocol (SNMP) to remotely monitor a label switching router (LSR) that is using the Multiprotocol Label Switching (MPLS) technology. The MPLS-LSR-MIB contains managed objects that support the retrieval of label switching information from a router.

The objects which we implemented are the `mplsVersion`, `mplsLspFrom`, `mplsLspTo` and `mplsPathBandwidth` and the packet loss, departures and arrivals at a particular node. Each object in the MIB incorporates a DESCRIPTION field that includes an explanation

of the object's meaning and usage, which together with the other characteristics of the object (SYNTAX , MAX - ACCESS and INDEX) provides sufficient information for management application development, as well as for documentation and testing.

# **INDEX**

## **TITLE**

### **1. Introduction**

#### **1.1 MPLS Architecture**

#### **1.2 MPLS Path Establishment**

#### **1.3 Network Management Using SNMP**

### **2. Platform Used**

#### **2.1 LINUX Operating System**

#### **2.2 TCL Script**

#### **2.3 NS 2 Simulator**

### **3. MPLS LSR MIB Objects**

### **4. Details of Implementation**

### **5. Output**

### **6. Conclusion**

#### **Future Enhancements**

### **7. References**

### **8. Appendices**

#### **A: Acronyms used**

#### **B: Source Code**

# 1.INTRODUCTION

## 1.1 MPLS ARCHITECTURE – AN INTRODUCTION

Multi - Protocol Label Switching ( MPLS ) was originally presented as a way of improving the forwarding speed of routers but is now emerging as a crucial standard technology that offers new capabilities for large scale IP networks. The essence of MPLS is the generation of a short fixed-length label that acts as a shorthand representation of an IP packet's header.

IP packets have a field in their 'header' that contains the address to which the packet is to be routed. Traditional routed networks process this information at every router in a packet's path through the network (hop by hop routing).

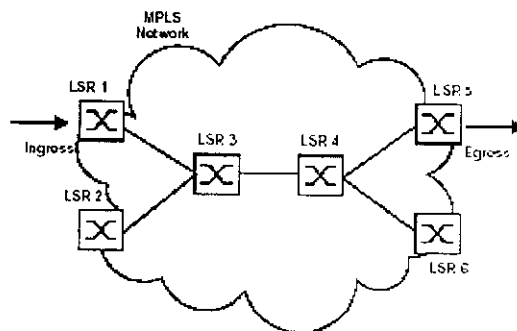


fig 1 MPLS ARCHITECTURE

MPLS is an accelerated data transmission technology that uses a method of label lookup. A short, fixed-length label is appended/overlaid to



the packets that traverse through an MPLS domain. Once that packet reaches the destination edge of the network, the label is stripped away. The MPLS protocol derives its performance by simply switching labels on packets within the network. MPLS is an efficient way of integrating IP and ATM networks because it allows high-volume IP traffic to traverse core ATM infrastructures. A Label Edge Router (LER) is an LSR placed at the edge of an MPLS domain and passes traffic into and out of the MPLS domain. An ingress LER is responsible for classifying data and assigning it to a suitable LSP.

When unlabelled packets need to traverse the same path between an ingress and an egress LSR (packets from an aggregate of one or more flows are said to belong to a stream) belonging to the same MPLS domain, a Label Switched Path (LSP) – a LSP is similar to a unidirectional ATM Virtual Circuit (VC) – needs to be set-up. This will allow the packet to be forwarded from one MPLS node to another just by using the assigned label as an index to a forwarding table. The LSP set-up can be traffic, request, or topology-driven. In the traffic-driven scheme the label assignment is triggered by the arrival of data at an LSR, whereas with the request-driven scheme the label is assigned in response to normal processing of request based control traffic. In the case of a topology-driven scheme the labels are pre - assigned according to existing routing protocol information.

In the MPLS forwarding paradigm, once a packet is assigned to a FEC, no further header analysis is done by subsequent routers; all forwarding is driven by the labels. This has a number of advantages over conventional network layer forwarding.

- MPLS forwarding can be done by switches which are capable of doing label lookup and replacement, but are either not capable of analyzing the network layer headers, or are not capable of analyzing the network layer headers at adequate speed.

- Since a packet is assigned to a FEC when it enters the network, the ingress router may use, in determining the assignment, any information it has about the packet, even if that information cannot be gleaned from the network layer header.
- A packet that enters the network at a particular router can be labeled differently than the same packet entering the network at a different router, and as a result forwarding decisions that depend on the ingress router can be easily made. This cannot be done with conventional forwarding, since the identity of a packet's ingress router does not travel with the packet.
- Sometimes it is desirable to force a packet to follow a particular route which is explicitly chosen at or before the time the packet enters the network, rather than being chosen by the normal dynamic routing algorithm as the packet travels through the network . This may be done as a matter of policy, or to support traffic engineering. In conventional forwarding, this requires the packet to carry an encoding of its route along with it ("source routing"). In MPLS, a label can be used to represent the route, so that the identity of the explicit route need not be carried with the packet.

Some routers analyze a packet's network layer header not merely to choose the packet's next hop, but also to determine a packet's "precedence" or "class of service". They may then apply different discard thresholds or scheduling disciplines to different packets. MPLS allows (but does not require) the precedence or class of service to be fully or partially inferred from the label. In this case, one may say that the label represents the combination of a FEC and a precedence or class of service.

## **MPLS network structure:**

A typical structure of a MPLS network consists of several basic elements.

### Label Edge Router (Ingress & Egress):

Label Edge Routers are located at the boundaries of MPLS network performing value added network services and assigning / removing labels to the packets ie., Ingress LSR assign Labels to the packets and Egress LSR removes label from the packet.

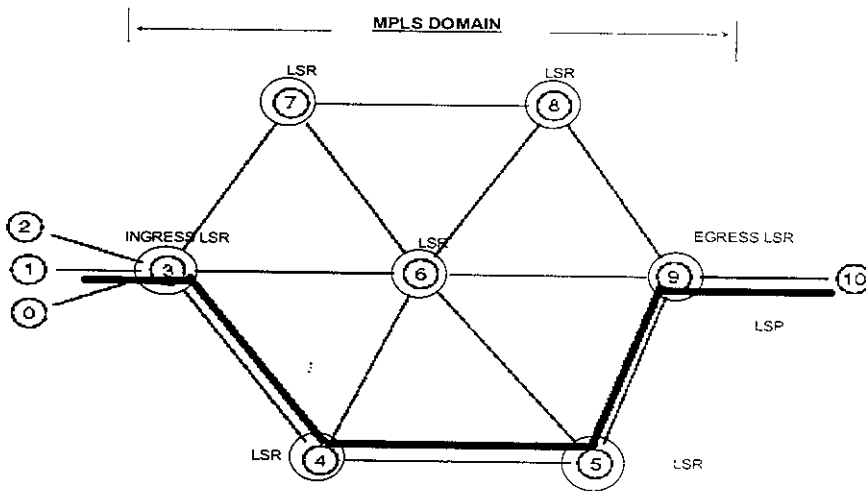


fig 2. MPLS NETWORK STRUCTURE

### Label Switch Router:

With reference to the labels assigned to the packets and LIB, PFT & ERB packets are forwarded through the established LSP (Label Switched Path) towards the Egress LSR by swapping the incoming label with suitable out going label.

### **Label Distribution Protocol:**

To ensure the capacity for the transmission in the reserved end-to-end the LER uses a Label Distribution Protocol (LDP) such as Constrained-Based Routing LDP (CR-LDP) or the RSVP-TE to distribute the necessary labels that direct traffic along this reserved route and thus label switched path (LSP) is established.

### **Tables (LIB, PFT, ERB):**

There are three tables to manage information related to LSP.

#### **Partial Forwarding Table (PFT):**

-a subset of Forwarding Table and consists of FEC, Per-Hop-Behavior (PHB) and LIB pointer.

#### **Label Information Base (LIB):**

- has the information about the LSP such as incoming interface, incoming label, outgoing interface, outgoing label and LIB pointer.

#### **Explicit Routing Information Base (ERB):**

- has information for the explicit routing path such as LSP-ID, FEC and LIB pointer.

## 1.2 MPLS LSP Establishment

The following are three major processes in the MPLS functionality:

### **Path selection:**

The best path determination through a network using either a hop-by-hop or an explicit route methodology. The hop-by-hop method allows the path selection to follow the normal underlying IGP best path. Each node in the path is responsible for determining the best next hop on the link state database. Alternatively, an explicit route is a path through the network that is specified by the instantiating router. The explicitly routed path has administratively configured criteria, like constraints, to influence the path selection through the underlying network. It is very possible an explicit route will deviate from a path that would have been selected using the hop-by-hop IGP method.

### **Path establishment:**

Once the path has been determined, a signaling protocol is used to inform all the routers in the path that a new label switch path, or LSP is required. The signaling protocol is responsible for indicating the specifications of the path, including the session id, resource reservations and the like, to all other routers in the path. This process also includes the label-mapping request for all the data that will use the label switched path. Following the successful establishment of the path the signaling is responsible for ensuring the integrity of the peering session.

### **Packet forwarding:**

At the very highest level the data flow toward an MPLS network occurs at the ingress label switch router. The ingress LSR classifies a packet or a flow

to a specific path and pushes the applicable label on the packet. Routers along the label switched path perform forwarding based on the top-level label.

### 1.3 Network Management using SNMP

This section explains the basic architecture of reading MIB values of the LSP object in lsp path between agent and management station.

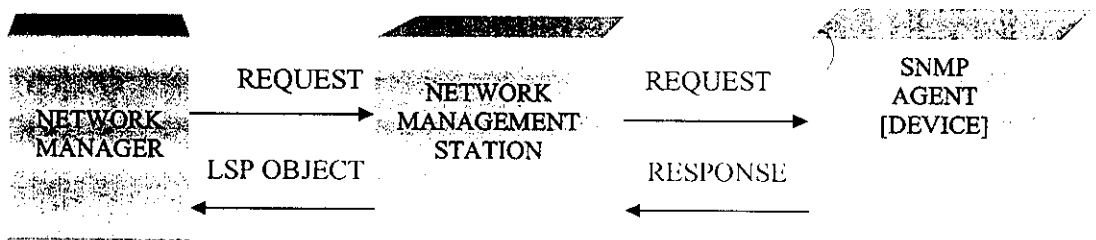


Fig 3 SNMP Architecture

#### SNMP Architecture

The SNMP architecture is based on the very simple concept of the Query / response model. The client, which sends out Queries, is generally described as the Manager. The SNMP server is referred to as the 'agent'. The SNMP protocol enables a network management station to read and to change an agent's parameters according to the rules of SNMP. SNMP also allows the agents to send an unrequested message to the management station.

The simplicity of the SNMP system reduces the complexity of the overall range of functions and contributes significantly to the fulfillment of the following goals :

- 1) Development costs for the implementation of the overall system are reduced.

- 2) The range of functions can be reached from every device via a system structure thereby reducing acceptance of network management applications.
- 3) The simple protocol structure makes the network management very easy to understand. It is cost efficient to run the network.
- 4) The entire architectural model was defined as simply as possible and is completely independent of hardware and software structures.

The SNMP architecture establishes the following fixed points :

- 1) The protocol and its range of functions
- 2) The information and data that can be transferred via the protocol.
- 3) The protocol mechanisms necessary for the transportation of management functions.
- 4) The functions and tasks of the individual devices participating in the transfer of management data.

### **SNMP Commands**

SNMP supports five message types. The names of the individual commands are given:

- 1) Get Request
- 2) Get Next
- 3) Set Request



4) Get Response

5) Event/Trap

### **Get Request**

The Get Request command enables a management system (client) to request a given variable in the MIB of an SNMP agent(server).An Object identifier is set as an argument with this type of message.The client always receives a Get Response message in reply to a Get Request.

### **Set Request**

The Set Request command enables a management system (client) to set certain specific variables in the MIB of an SNMP agent (server) .An object identifier is sent as an argument with this type of message. If the Set Request command can be processed with the specified value from the agent, a Get Response packet is sent back . If there is an error , a Get Response is created and sent back to the requester (management station) with the relevant error message.

### **Get Next Request**

The Get Next Request command enables a management station (client) to request a value for the next object in the MIB tree hierarchy . The Get Next operation is ideally suited for traversing tables and for requesting consecutive objects quickly.In response to a Get Next Request, the client always receives a message of the type Get Response.

## **Get Response**

The Get Response command enables an agent (server) to respond to all Get Next Request, Set Request and Get request queries from a management System (client). If the relevant command can be processed with the specified value from the agent, a Get Response packet is returned and the operation is positively confirmed.

## **Event/Trap**

SNMP is essentially based on a simple polling mechanism, according to which every network management station (client) has to implement all variable and status requests explicitly. If an agent determines a particular situation, it sends a trap message to the management station. This method enables the network management station to react immediately to the information sent.

---

PLATFORM USED

---

## 2. PLATFORM USED

### 2.1 LINUX OPERATING SYSTEM

This project works on the Red Hat LINUX operating system, Version 6 and above. Linux, a UNIX clone, is an operating system that embodies the concept of complete transparency. One of the most important aspects of Linux is that it has been developed and supported by its users, making it relatively easy to get your hands on one of the many free Linux distributions. But although Linux is growing in popularity, there are still people who consider this operating system a toy or pet; something to play with, but not to be taken seriously.

Until recently, Linux was reserved for self-avowed hackers and enthusiasts. This was mainly because Linux was not very user-friendly. Now, with an intuitive graphical user interface or GUI, Linux is as user friendly as Windows. You do not need to be a rocket Scientist to use Linux with the GUI. In fact, a Linux Desktop looks much like a Windows Desktop (with a few added features). Linux has at least a dozen different highly configurable graphical interfaces, which runs on top of a Xfree86.

#### FEATURES OF LINUX :

1. *Multitasking* : several programs running at the same time.
2. *Multiuser* : several users on the same machine at the same time.
3. *Multiplatform* : runs on many different CPU 's , not just Intel .
4. *Multiprocessor* : SMP support is available on the Intel and SPARC platforms (with work currently in progress on other platforms)

5. *Multithreading* : has native kernel support for multiple independent threads of control within a single process memory space.

## 2.2 TCL Script (Tool Command Language)

This project works on Tcl . Tcl was originally intended to be a reusable command language. Its developers had been creating a number of interactive tools, each requiring its own command language. Since they were more interested in the tools themselves than the command languages they would employ, these command languages were constructed quickly, without regard to proper design . After implementing several such "quick-and-dirty" command languages and experiencing problems with each one, they decided to concentrate on implementing a general-purpose, robust command language that could easily be integrated into new applications. Thus Tcl (Tool Command Language) was born. Since that time, Tcl has been widely used as a scripting language. In most cases, Tcl is used in combination with the Tk ("Tool Kit") library, a set of commands and procedures that make it relatively easy to program graphical user interfaces in Tcl.

One of Tcl's most useful features is its extensibility. If an application requires some functionality not offered by standard Tcl, new Tcl commands can be implemented using the C language, and integrated fairly easily. Since Tcl is so easy to extend, many people have written extension packages for some common tasks, and made these freely available on the internet.

The Tcl script is compiled with the (.tcl extension).

## FEATURES OF TCL SCRIPTING LANGUAGE

The main difference between Tcl and languages such as C, is that Tcl is an *interpreted* rather than a *compiled* language. Tcl programs are simply scripts consisting of Tcl commands that are processed by a Tcl interpreter at run time. One advantage that this offers is that Tcl programs can themselves generate Tcl scripts

that can be evaluated at a later time. This can be useful, for example, when creating a graphical user interface with a command button that needs to perform different actions at different times.

## 2.3 NS2 SIMULATOR

*ns* is an object oriented simulator, written in C++, with an OTcl interpreter as a frontend. The simulator supports a class hierarchy in C++ (also called the compiled hierarchy in this document), and a similar class hierarchy within the OTcl interpreter (also called the interpreted hierarchy in this document). The two hierarchies are closely related to each other; from the user's perspective, there is a one-to-one correspondence between a class in the interpreted hierarchy and one in the compiled hierarchy. The root of this hierarchy is the class TclObject. Users create new simulator objects through the interpreter; these objects are instantiated within the interpreter, and are closely mirrored by a corresponding object in the compiled hierarchy. The interpreted class hierarchy is automatically established through methods defined in the class TclClass. User instantiated objects are mirrored through methods defined in the class TclObject. There are other hierarchies in the C++ code and Otcl scripts; these other hierarchies are not mirrored in the manner of TclObject.

*ns* uses two languages because simulator has two different kinds of things it needs to do. On one hand, detailed simulations of protocols requires a systems programming language which can efficiently manipulate bytes, packet headers, and implement algorithms that run over large data sets. For these tasks run-time speed is important and turn-around time (run simulation, find bug, fix bug, recompile, re-run) is less important. On the other hand, a large part of network research involves slightly varying parameters or configurations, or quickly exploring a number of scenarios. In these cases, iteration time (change the model and re-run) is more important. Since configuration runs once (at the beginning of the simulation), run-time of this part of the task is less important. *Ns* meets both of these needs with two languages, C++ and OTcl. C++ is fast to run but slower to change, making it suitable for detailed protocol implementation. OTcl runs much slower but can be changed very quickly (and interactively), making it ideal for simulation configuration. *ns* (via tclcl) provides glue to make objects and variables appear on both languages.



---

MPLS LSR MIB OBJECTS

---

### **3.MPLS LSR MIB OBJECTS**

The MIB objects which are present in the MPLS domain which includes the following:

- 1) MPLS-TC-MIB
- 2) MPLS-TE-MIB
- 3) MPLS-LDP-MIB
- 4) MPLS-LSR-MIB

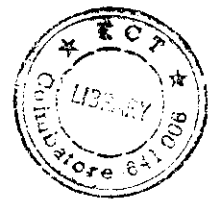
But here we deal with only MPLS-LSR-MIB objects.

The MPLS Label Switching Router MIB (MPLS-LSR-MIB) allows to use the Simple Network Management Protocol (SNMP) to remotely monitor a label switching router (LSR) that is using the Multiprotocol Label Switching (MPLS) technology.

The notation used in the MPLS – LSR – MIB follows the conventions defined in Abstract Syntax Notation One (ASN.1). ASN.1 defines an Open System Interconnection (OSI) language used to describe data types independently from particular computer structures and presentation techniques. Each object in the MIB incorporates a DESCRIPTION field that includes an explanation of the object's meaning and usage, which, together with the other characteristics of the object (SYNTAX, MAX-ACCESS, and INDEX) provides sufficient information for management application development, as well as for documentation and testing. The MPLS-LSR-MIB represents an ASN.1 notation reflecting an idealized MPLS LSR.

A network administrator can access the entries (objects) in the MPLS-LSR-MIB by means of any SNMP-based network management system (NMS). The network administrator can retrieve information in the MPLS-LSR-MIB using standard SNMP **get** and **getnext** operations.

Typically, SNMP runs as a low-priority process. The response time for the MPLS-LSR-MIB is expected to be similar to that for other MIBs. The size and structure of the MIB and other MIBs in the system influence response time when you retrieve information from the management database. Traffic through the LSR also affects SNMP performance.



### MPLS-LSR-MIB Structure

MIB structure is represented by a tree hierarchy. Branches along the tree have short text strings and integers to identify them. Text strings describe object names, and integers allow computer software to encode compact representations of the names. The MPLS-LSR-MIB falls on the experimental branch of the Internet MIB hierarchy. The experimental branch of the Internet MIB hierarchy is represented by object identifier 1.3.6.1.3. This branch can also be represented by its object name *iso.org.dod.internet.experimental*. The MPLS-LSR-MIB is identified by the object name *mplsLsrMIB*, which is denoted by the number 96. Therefore, objects in the MPLS-LSR-MIB can be identified by either of the following:

- The object identifier—1.3.6.1.3.96.[MIB-variable]
- The object name—*iso.org.dod.internet.experimental.mplsLsrMIB*. [MIB-variable]

To display a *MIB-variable*, enter an SNMP **get** command with an object identifier. Object identifiers are defined by the MPLS-LSR-MIB.

The MPLS LSR MIB objects are

mplsLspName	DisplayString,
mplsLspState	INTEGER,
mplsLspOctets	Counter64,
mplsLspPackets	Counter64,
mplsLspAge	TimeStamp,
mplsLspTimeUp	TimeStamp,
mplsLspPrimaryTimeUp	TimeStamp,
mplsLspTransitions	Counter32,
mplsLspLastTransition	TimeStamp,
mplsLspPathChanges	Counter32,
mplsLspLastPathChange	TimeStamp,
mplsLspConfiguredPaths	Integer32,
mplsLspStandbyPaths	Integer32,
mplsLspOperationalPaths	Integer32,

mplsLspFrom	IpAddress,
mplsLspTo	IpAddress,
mplsPathName	DisplayString,
mplsPathType	INTEGER,
mplsPathExplicitRoute	OCTET STRING,
mplsPathRecordRoute	OCTET STRING,
mplsPathBandwidth	Integer32,
mplsPathCOS	INTEGER,
mplsPathInclude	Integer32,
mplsPathExclude	Integer32,
mplsPathSetupPriority	INTEGER,
mplsPathHoldPriority	INTEGER,
mplsPathProperties	INTEGER

## 4. DETAILS OF IMPLEMENTATION

### LSR objects implemented

The **MPLS LSR MIB** objects we implemented are `mplsVersion`, `mplsLspFrom`, `mplsLspTo`, `mplsPathBandwidth`, `packet loss`, `arrivals` and `departures` at a particular LSP path.

These objects are queried by the user and the result is thus obtained. The queried objects may then be retrieved or changed by the user.

### MPLS LSR MIB object structure

#### 1) `mplsVersion`

<code>mplsVersion</code>	OBJECT-TYPE
SYNTAX	Integer32
MAX-ACCESS	read-only
STATUS	current
DESCRIPTION	

"MPLS version number."

## 2) mplsLspFrom

mplsLspFrom                      OBJECT-TYPE

SYNTAX                              IpAddress

MAX-ACCESS                        read-only

STATUS                              current

DESCRIPTION

"Source IP address of this LSP."

## 3) mplsLspTo

mplsLspTo                        OBJECT-TYPE

SYNTAX                              IpAddress

MAX-ACCESS                        read-only

STATUS                              current

DESCRIPTION

"Destination IP address of this LSP."

#### 4) **mplsPathBandwidth**

mplsPathBandwidth                      OBJECT-TYPE

SYNTAX                                      Integer32

MAX-ACCESS                                read-only

STATUS                                      current

#### DESCRIPTION

"The configured bandwidth for this LSP, in units of thousands of bits per second (Kbps). This field is meaningless unless mplsPathName is not empty"



Retrieval of objects

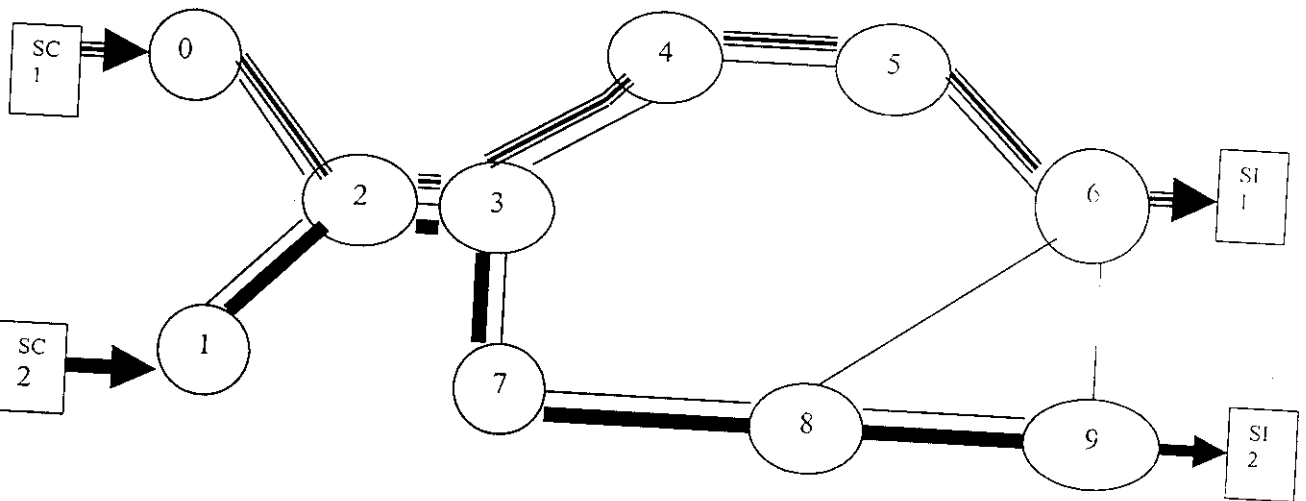


Fig 4. Path Establishment

SC1-SOURCES ONE

SC2-SOURCES TWO

SI1-SINK ONE

SI2-SINK TWO

— LSP-1

≡≡≡ LSP-2

— NETWORK ROUTE

LSP1 0-2-3-4-5-6

LSP2 1-2-3-7-8-9

## Configuration

Configuration menu provides the following options:

1. View a lsp object
2. Get a lsp object
3. Getnext lsp object
4. Evaluate a bandwidth
5. View the network structure

When the particular LSP path is activated then all agent in that path are monitored and the LSP MIB objects are generated. When the manager wants the object by querying the agent the response is given to management station.

The network manager refers to the path selection, path establishment and path forwarding are decided by ingress router. Before running the application the management station should know the bandwidth is available for the application.

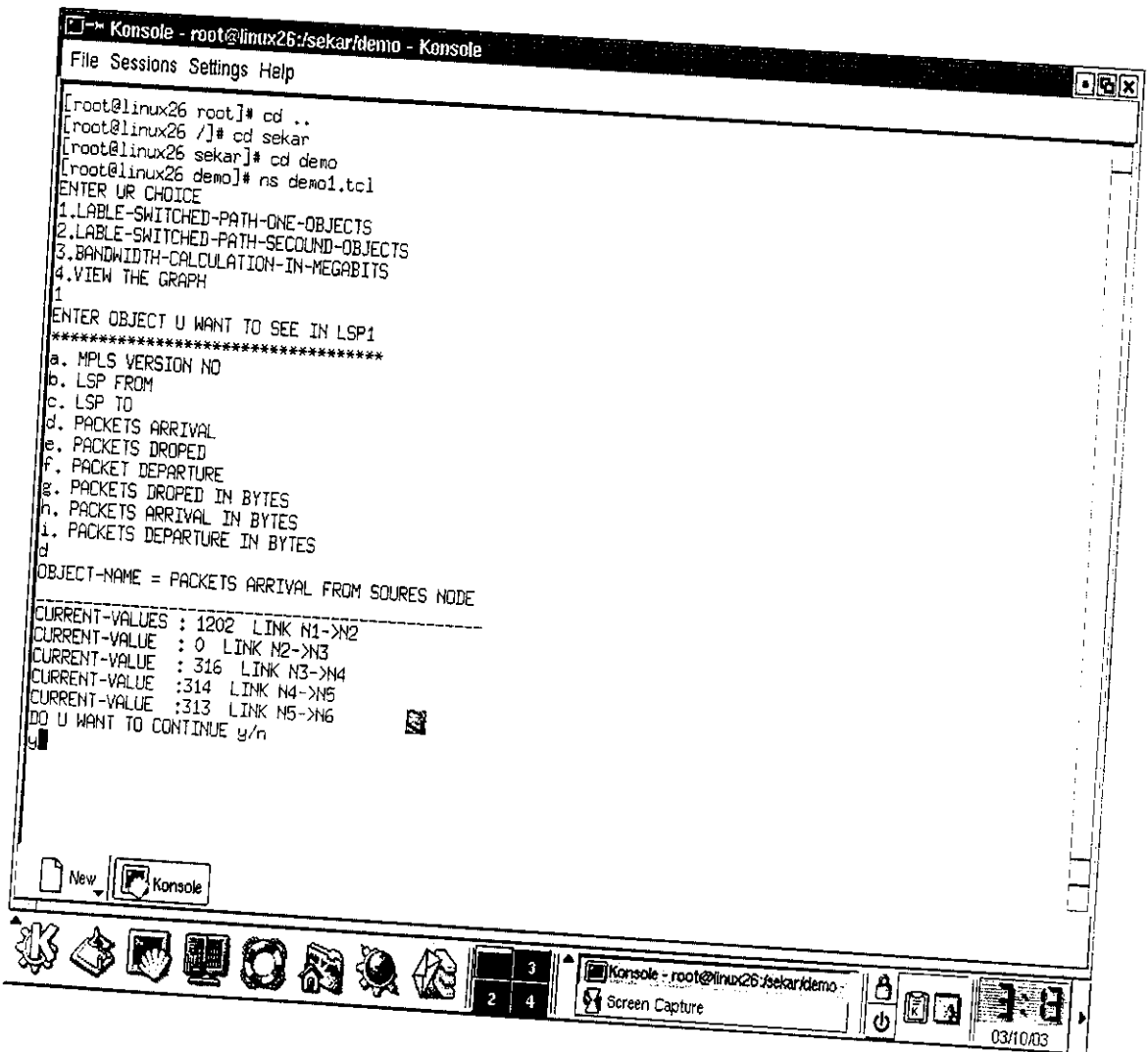
This frame work senses the link capacity between nodes ,from which node LSP path starts, destination node, packets in quantity arrived, deparutured ,dropped is calculated in bytes between nodes in LSP . In this frame work nam animator is used which simulates the movement of packets between the nodes in network and also displays if any drop has occurred.

## 5.OUTPUT

In this framework we used a nam animator which simulates the packets between the ingress router to the egress router through the pre-established LSP path. The nam animator output shows the packets which has been dropped during the transmission. It also shows where the LSP path starts, the number of packets arrived at a particular node , the packets departed and the packets dropped.

The ingress router decides the path establishment , selection and the packet forwarding in a particular LSP path. The Bandwidth is calculated between the nodes in bytes.

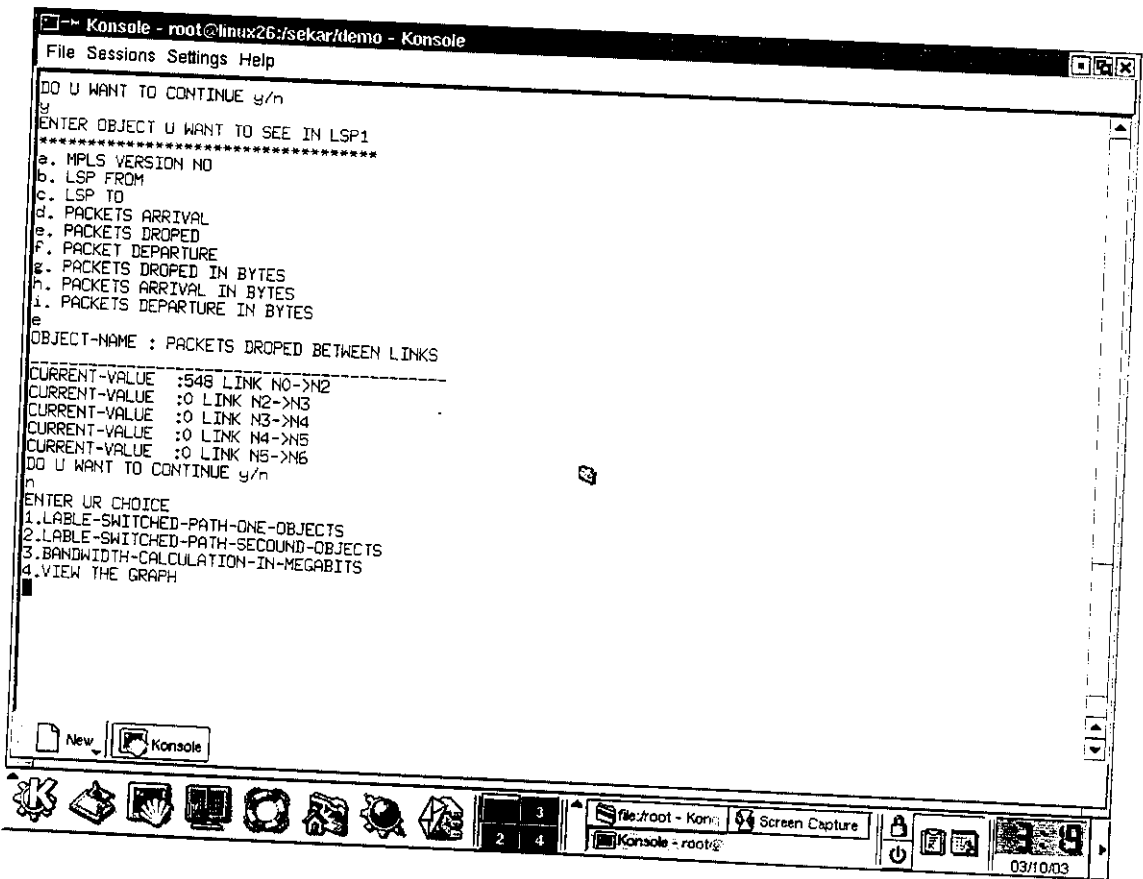
## SCREEN SHOTS

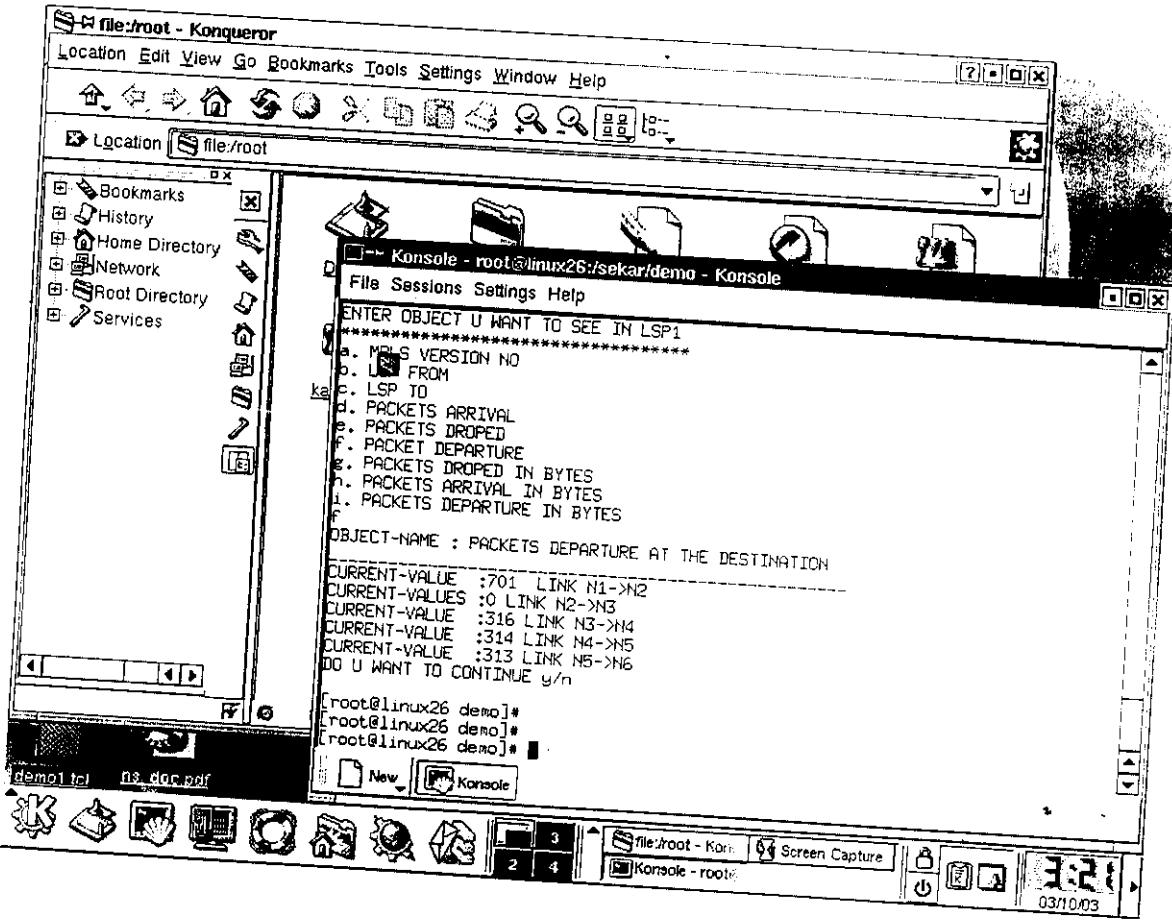


The screenshot shows a terminal window titled "Konsole - root@linux26:/sekar/demo - Konsole". The terminal content is as follows:

```
[root@linux26 root]* cd ..
[root@linux26 /]# cd sekar
[root@linux26 sekar]* cd demo
[root@linux26 demo]* ns demo1.tcl
ENTER UR CHOICE
1.LABLE-SWITCHED-PATH-ONE-OBJECTS
2.LABLE-SWITCHED-PATH-SECOUND-OBJECTS
3.BANDWIDTH-CALCULATION-IN-MEGABITS
4.VIEW THE GRAPH
1
ENTER OBJECT U WANT TO SEE IN LSP1
*****
a. MPLS VERSION NO
b. LSP FROM
c. LSP TO
d. PACKETS ARRIVAL
e. PACKETS DROPED
f. PACKET DEPARTURE
g. PACKETS DROPED IN BYTES
h. PACKETS ARRIVAL IN BYTES
i. PACKETS DEPARTURE IN BYTES
d
OBJECT-NAME = PACKETS ARRIVAL FROM SOURES NODE
-----
CURRENT-VALUES : 1202 LINK N1->N2
CURRENT-VALUE  : 0 LINK N2->N3
CURRENT-VALUE  : 316 LINK N3->N4
CURRENT-VALUE  : 314 LINK N4->N5
CURRENT-VALUE  : 313 LINK N5->N6
DO U WANT TO CONTINUE y/n
y
```

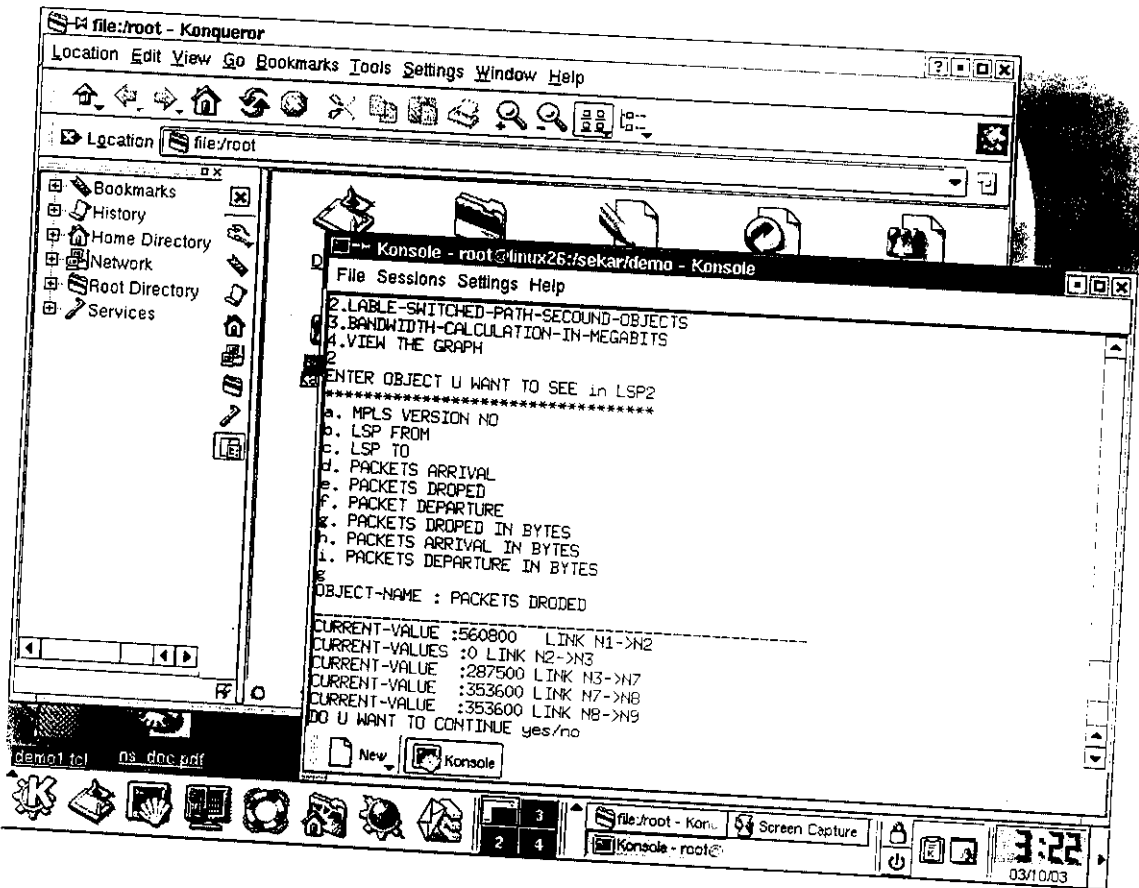
The terminal window includes a menu bar with "File Sessions Settings Help" and a taskbar at the bottom with various system icons and a date/time display of "03/10/03".

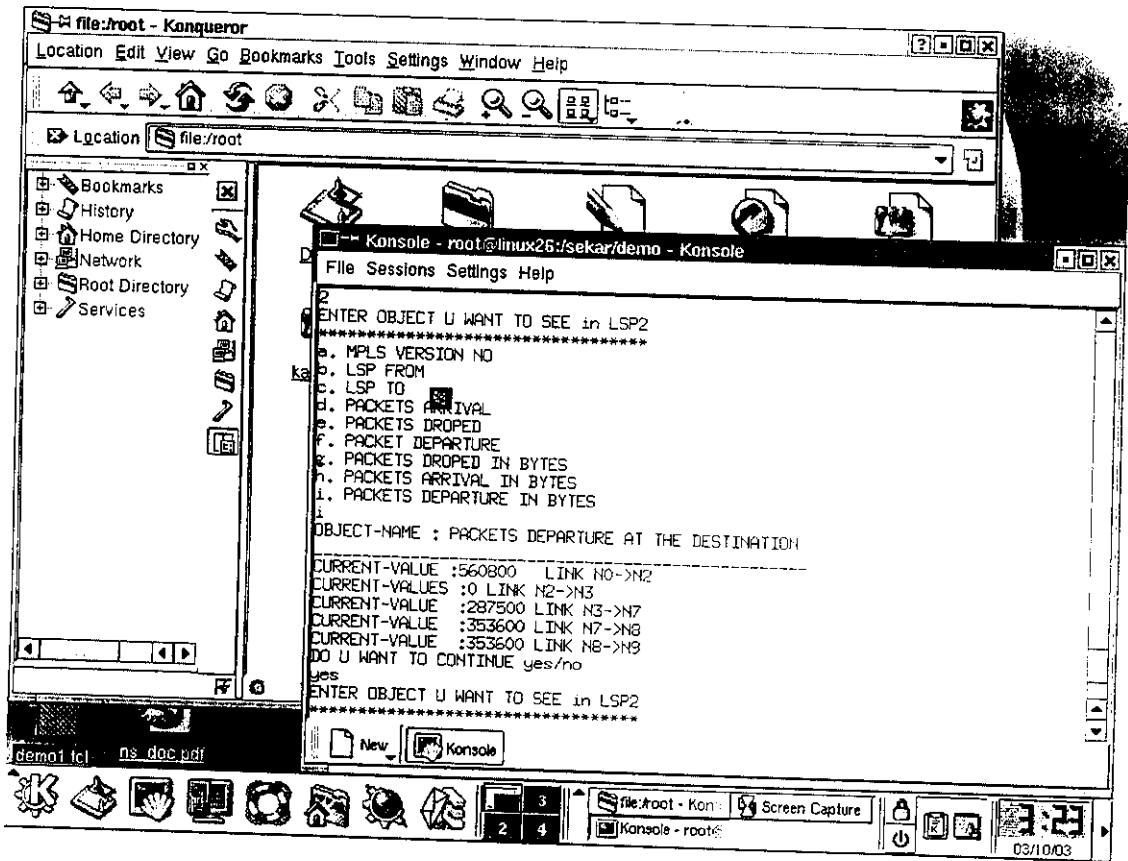




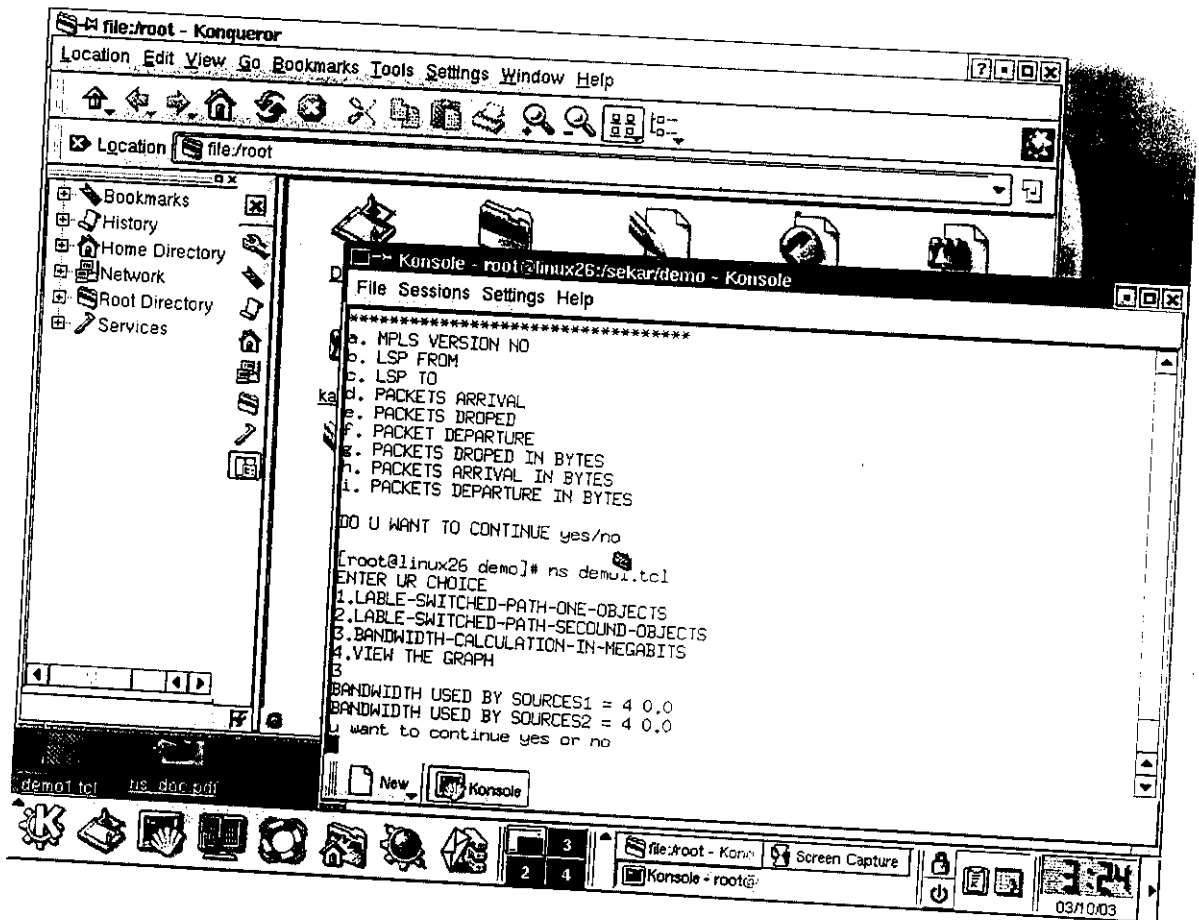
```
Konsole - root@linux26:/sekar/demo - Konsole
File Sessions Settings Help
ENTER OBJECT U WANT TO SEE IN LSP1
*****
a. MFLS VERSION NO
b.  FROM
c. LSP TO
d. PACKETS ARRIVAL
e. PACKETS DROPE
f. PACKET DEPARTURE
g. PACKETS DROPE IN BYTES
h. PACKETS ARRIVAL IN BYTES
i. PACKETS DEPARTURE IN BYTES
f
OBJECT-NAME : PACKET DEPARTURE AT THE DESTINATION
-----
CURRENT-VALUE :701 LINK N1->N2
CURRENT-VALUES :0 LINK N2->N3
CURRENT-VALUE :316 LINK N3->N4
CURRENT-VALUE :314 LINK N4->N5
CURRENT-VALUE :313 LINK N5->N6
DO U WANT TO CONTINUE y/n

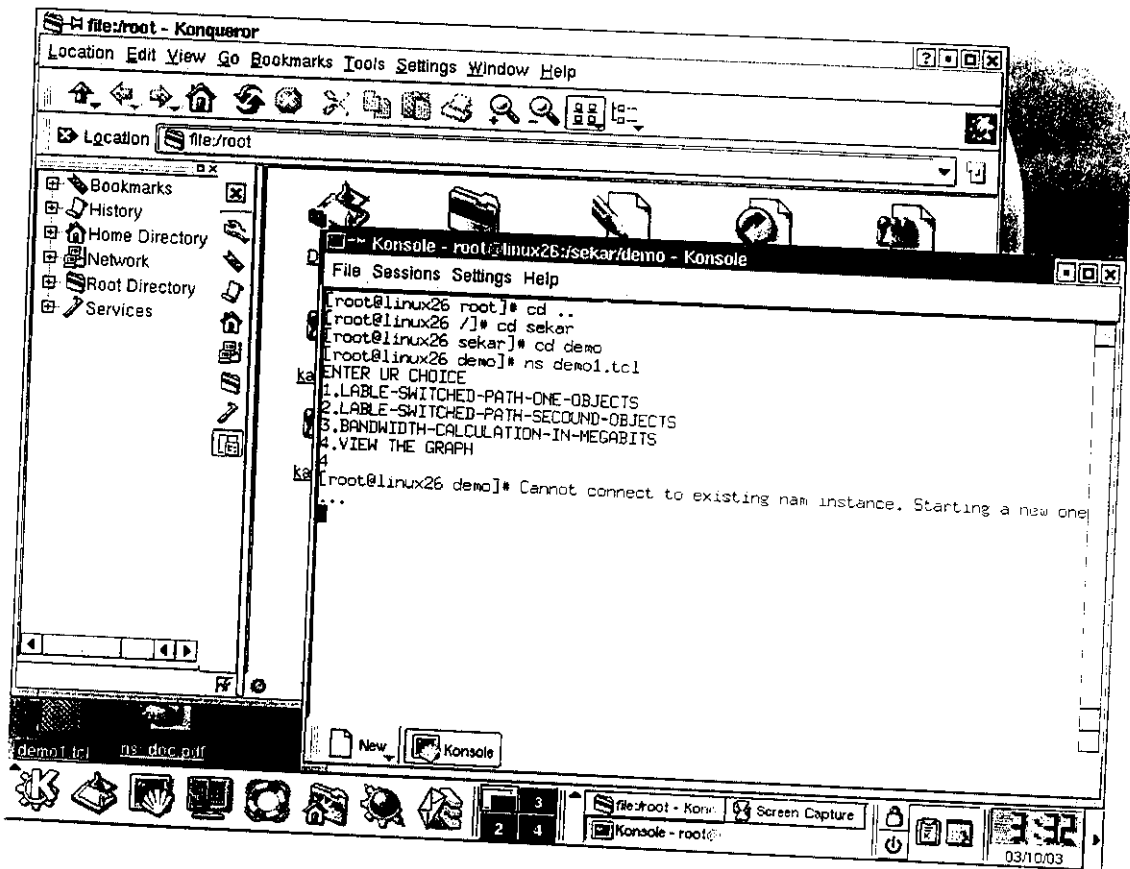
[root@linux26 demo]#
[root@linux26 demo]#
[root@linux26 demo]#
```

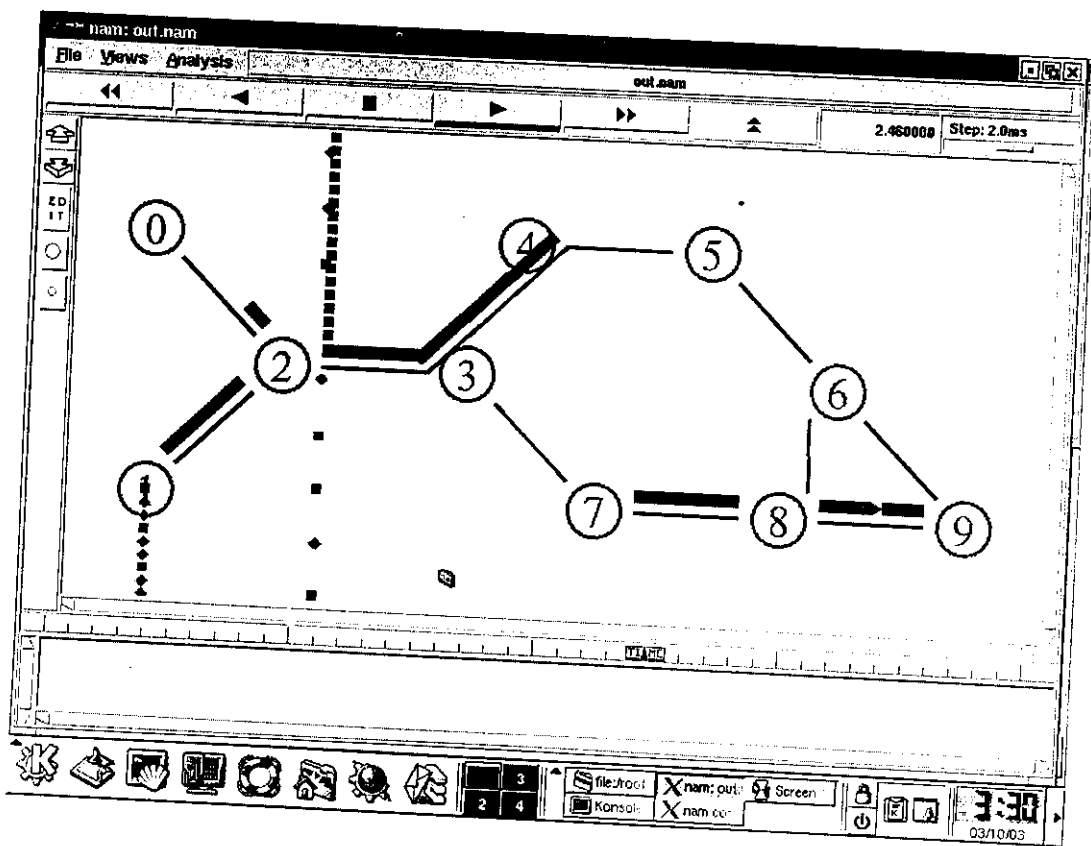












---

CONCLUSION

---

## 6.CONCLUSION

The sole purpose of this project was to study the behaviour of MPLS based networks and designing the management framework which could be used by a network administrator to more closely monitor the performance of their MPLS domain during their operation. Since designing a complete management framework is exhaustive and could not be completed within the stipulated period of 4 months, this project work mainly focuses on MPLS-MIB-LSR, which is used in monitoring the behaviour of LSP's once they are established.

Using the NS2 simulator , querying and retrieving up of the objects is successfully completed . The packet loss , departures and arrivals are calculated between the established LSP paths.

### **Future Enhancements**

The MPLS-LSR –MIB objects has been done only for a limited set of objects. The objects which has been queried and retrieved are the mplsVersion, mplsLspFrom , mplsLspTo and mplsPathBandwidth . There are some objects which cannot be implemented namely MplsAdminGroup , MplsAdminGroupList, in the MPLS-LSR-MIB.

These features can be implemented for other MPLS domain namely MPLS-TC-MIB , MPLS – LDP- MIB , MPLS –TE-MIB .

## 7.REFERENCES

- 1) Feldman, et al., “Evolution of Multiprotocol Label Switching “,  
IEEE Communications Magazine , Vol .36 , No. 5 ,May 1998.
- 2) David Greenfield , “MPLS in Brief “ ,  
Network magazine – Feb . 2002
- 3) E.Rosen , A.Viswanathan , R.Callon “MultiProtocol Label Switching  
Architecture” , RFC 3031 , January 2001.
- 4) S.McCanne and S.Floyd.ns—NetworkSimulator.  
<http://www-mash.cs.berkeley.edu/ns/>.
- 5) [www.juniper.net/techpubs/software/junos53/swconfig53-net-mgmt/html/mib-mpls.txt](http://www.juniper.net/techpubs/software/junos53/swconfig53-net-mgmt/html/mib-mpls.txt).
- 6) [www.mplssrc.com/whitepapers/marconi.pdf](http://www.mplssrc.com/whitepapers/marconi.pdf)
- 7) <http://www.trillium.com/assets/broadband!mpls/datasheet/8743023.pdf>.

---

## APPENDICES

---

## 8. Appendices

### A. Acronyms used:

ATM	ASYNCHRONOUS TRANSFER MODE
FEC	FORWARDING EQUIVALENCE CLASS
IGP	INTERIOR GATEWAY PROTOCOL
IP	INTERNET PROTOCOL
LDP	LABEL DISTRIBUTION PROTOCOL
LSP	LABEL SWITCHED PATH
LSR	LABEL SWITCHING ROUTER
MPLS	MULTI PROTOCOL LABEL SWITCHING
OSPF	OPEN SHORTEST PATH FIRST
QOS	QUALITY OF SERVICE
TE	TRAFFIC ENGINEERING
TTL	TIME TO LIVE



## **B. Source Code**

The following source of code is stored as Demol.tcl . Its purpose is to monitor the MPLS domain , to create nodes in the network and monitor the packet loss from source to destination. The bandwidth used by the particular LSP path is monitored.

The MPLS objects are queried and the results are obtained . It also gives the results of the packet arrivals , packets dropped and packet departures.

- /\* Core portion of the Network monitoring source code*
- \* Create the nodes from source to destination link
- \* Create links between the nodes
- \* Set the packet size for each node and the interval to be transferred from each node.
- \* Makes relevant calls to MPLS version number, MPLS from , MPLS to and the packet departures , arrivals and loss between the nodes.

```
#Create a simulator object
```

```
set ns [new Simulator]
```

```
global bw
```

```
#Define different colors for data flows
```

```
$ns color 1 Blue
```

```
$ns color 2 Red
```

```
$ns color 3 Black
```

```
#Open the nam trace file
```

```
set nf [open out.nam w]
```

```
$ns namtrace-all $nf
```

```
#Define a 'finish' procedure
```

```
proc finish {} {
```

```
    global ns nf
```

```
    $ns flush-trace
```

```
#Close the trace file
```

```
    close $nf
```

```
#Execute nam on the trace file
```

```
    exec nam out.nam &
```

```
    exit 0
```

```
}
```

#Create four node

set n0 [\$ns node]

set n1 [\$ns node]

set n2 [\$ns node]

set n3 [\$ns node]

set n4 [\$ns node]

set n5 [\$ns node]

set n6 [\$ns node]

set n7 [\$ns node]

set n8 [\$ns node]

set n9 [\$ns node]

#Create links between the nodes

\$ns duplex-link \$n0 \$n2 1.5Mb 1ms DropTail

\$ns duplex-link \$n1 \$n2 1.5Mb 1ms DropTail

\$ns duplex-link \$n3 \$n2 1.5Mb 1ms DropTail

\$ns duplex-link \$n4 \$n3 1.5Mb 2ms DropTail

\$ns duplex-link \$n4 \$n5 1.5Mb 2ms DropTail

\$ns duplex-link \$n5 \$n6 1.5Mb 2ms DropTail

\$ns duplex-link \$n6 \$n9 1.5Mb 2ms DropTail

\$ns duplex-link \$n3 \$n7 1.5Mb 2ms DropTail

\$ns duplex-link \$n7 \$n8 1.5Mb 2ms DropTail

\$ns duplex-link \$n8 \$n6 1.5Mb 2ms DropTail

\$ns duplex-link \$n8 \$n9 1.5Mb 2ms DropTail

\$ns duplex-link-op \$n0 \$n2 orient right-down

\$ns duplex-link-op \$n1 \$n2 orient right-up

\$ns duplex-link-op \$n2 \$n3 orient right

```
$ns duplex-link-op $n4 $n3 orient right
$ns duplex-link-op $n4 $n5 orient right
$ns duplex-link-op $n5 $n6 orient right-down
$ns duplex-link-op $n6 $n9 orient right-down
$ns duplex-link-op $n3 $n7 orient right-down
$ns duplex-link-op $n7 $n8 orient right
$ns duplex-link-op $n8 $n6 orient right-up
$ns duplex-link-op $n8 $n9 orient right
```

```
set qmon [$ns monitor-queue $n0 $n2 stdout]
set qmon1 [$ns monitor-queue $n1 $n2 stdout]
set qmon2 [$ns monitor-queue $n2 $n3 stdout]
set qmon3 [$ns monitor-queue $n3 $n4 stdout]
set qmon4 [$ns monitor-queue $n4 $n5 stdout]
set qmon5 [$ns monitor-queue $n5 $n6 stdout]
set qmon6 [$ns monitor-queue $n6 $n9 stdout]
set qmon7 [$ns monitor-queue $n3 $n7 stdout]
set qmon8 [$ns monitor-queue $n7 $n8 stdout]
set qmon9 [$ns monitor-queue $n8 $n9 stdout]
set qmon2 [$ns monitor-queue $n8 $n6 stdout]
```

```
#Monitor the queue for the link between node 2 and node 3
```

```
$ns duplex-link-op $n2 $n3 queuePos 0.5
$ns duplex-link-op $n6 $n9 queuePos 0.5
```

```
#Create a UDP agent and attach it to node n0
```

```
set udp0 [new Agent/UDP]
Sudp0 set class_ 1
```

```
$ns attach-agent $n0 $udp0
```

```
# Create a CBR traffic source and attach it to udp0
```

```
set cbr0 [new Application/Traffic/CBR]
```

```
$cbr0 set packetSize_ 800
```

```
$cbr0 set interval_ 0.005
```

```
$cbr0 attach-agent $udp0
```

```
#Create a UDP agent and attach it to node n1
```

```
set udp1 [new Agent/UDP]
```

```
$udp1 set class_ 2
```

```
$ns attach-agent $n1 $udp1
```

```
# Create a CBR traffic source and attach it to udp1
```

```
set cbr1 [new Application/Traffic/CBR]
```

```
$cbr1 set packetSize_ 1500
```

```
$cbr1 set interval_ 0.005
```

```
$cbr1 attach-agent $udp1
```

```
#Create a Null agent (a traffic sink) and attach it to node n3
```

```
set null0 [new Agent/Null]
```

```
$ns attach-agent $n9 $null0
```

```
#Create a Null agent (a traffic sink) and attach it to node n6
```

```
set null1 [new Agent/Null]
```

```
$ns attach-agent $n6 $null1
```

```
#Connect the traffic sources with the traffic sink
```

```
$ns connect $udp0 $null0
```

```
$ns connect $udp1 $null1
```

```
set sink0 [new Agent/LossMonitor]
```

```
set sink1 [new Agent/LossMonitor]
```

```
$ns attach-agent $n6 $sink0
```

```
$ns attach-agent $n9 $sink1
```

```
#mplsversion
```

```
proc abc1 {} {
```

```
puts "NAME OF OBJECT : mplsversion"
```

```
puts "CURRENT VALUE : MPLSV2"
```

```
puts "DESCRIPTION : Tells which version is used in this path"
```

```
}
```

```
#mplsversion
```

```
proc ab1 {} {
```

```
puts "NAME OF OBJECT : mplsversion"
```

```
puts "CURRENT VALUE : MPLSV2"
```

```
puts "DESCRIPTION : Tells which version is used in this path"
```

```
}
```

```
#mpls from
```

```
proc abc2 {}
```

```
{
```

```
puts "NAME OF THE OBJECT :LSP FROM "
```

```
puts "_____"
```

```
puts "CURRENT VALUE : 126.00.00.00"
```

```
puts "DESCRIPTION : IT DESCRIBES FROM WHICH NODE THE LABELED  
SWITCHED PATH STARTS"
```

```
}
```

```
#mpls from
```

```
proc ab2 {}
```

```
{
```

```
puts "NAME OF THE OBJECT :LSP FROM "
```

```
puts "_____"
```

```
puts "CURRENT VALUE : 127.00.00.00"
```

```
puts "DESCRIPTION : IT DESCRIBES FROM WHICH NODE THE LABELED  
SWITCHED PATH STARTS"
```

```
}
```

```
#mpls to
```

```
proc abc3 {}
```

```
{
```

```
puts "NAME OF THE OBJECT :LSP TO "
```

```
puts "_____"
```

```
puts "CURRENT VALUE : 128.00.100.00 "
```

```
puts "DESCRIPTION : IT DESCRIBES TO WHICH NODE THE PATH DESTINES"
```

```
puts "ACTS AS SINK IN LABEL SWITCH PATH"
```

```
}
```

```

#mpls to
proc ab3 {}
{
puts "NAME OF THE OBJECT :LSP TO "
puts "_____ "
puts "CURRENT VALUE : 128.00.100.00 "
puts "DESCRIPTION : IT DESCRIBES TO WHICH NODE THE PATH DESTINES"
puts "ACTS AS SINK IN LABEL SWITCH PATH"
}

```

```

#packetsarrival
proc abc4 {}
{
global qmon1 qmon2 qmon3 qmon4 qmon5 qmon6
puts "OBJECT-NAME PACKETS ARRIVED FROM SOURCE NODE"
puts "_____ "
puts "CURRENT-VALUE : [$qmon1 set parrivals_] LINK N1->N2"
puts "CURRENT-VALUE : [$qmon2 set parrivals_] LINK N2->N3"
puts "CURRENT-VALUE : [$qmon3 set parrivals_] LINK N3->N4"
puts "CURRENT-VALUE : [$qmon4 set parrivals_] LINK N4->N5"
puts "CURRENT-VALUE : [$qmon5 set parrivals_] LINK N5->N6"
}

```

```

#packetsarrival
proc ab4 {}
{
global qmon qmon2 qmon7 qmon8 qmon9
puts "OBJECT-NAME : PACKETS ARRIVED FROM SOURCE NODE"
puts "_____ "
puts "CURRENT-VALUE : [$qmon set parrivals_] LINK N0->N2"

```



```
puts "CURRENT-VALUE :[$qmon2 set parrivals_] LINK N2->N3"  
puts "CURRENT-VALUE :[$qmon7 set parrivals_] LINK N3->N7"  
puts "CURRENT-VALUE :[$qmon8 set parrivals_] LINK N7->N8"  
puts "CURRENT-VALUE :[$qmon9 set parrivals_] LINK N8->N9"  
}
```

```
#packetdropped
```

```
proc abc5 {} {
```

```
global qmon1 qmon2 qmon3 qmon4 qmon5 qmon6
```

```
puts "OBJECT-NAME : PACKETS DROPPED BETWEEN LINKS"
```

```
puts "_____"
```

```
puts "CURRENT-VALUE :[$qmon1 set pdrops_] LINK N0->N2"
```

```
puts "CURRENT-VALUE :[$qmon2 set pdrops_] LINK N2->N3"
```

```
puts "CURRENT-VALUE :[$qmon3 set pdrops_] LINK N3->N4"
```

```
puts "CURRENT-VALUE :[$qmon4 set pdrops_] LINK N4->N5"
```

```
puts "CURRENT-VALUE :[$qmon5 set pdrops_] LINK N5->N6"
```

```
}
```

```
#packetdropped
```

```
proc ab5 {} {
```

```
global qmon qmon2 qmon7 qmon8 qmon9
```

```
puts "OBJECT-NAME : PACKETS DROPPED BETWEEN LINKS"
```

```
puts "_____"
```

```
puts "CURRENT-VALUE :[$qmon set pdrops_] LINK N0->N2"
```

```
puts "CURRENT-VALUES :[$qmon2 set pdrops_] LINK N2->N3"
```

```
puts "CURRENT-VALUE :[$qmon7 set pdrops_] LINK N3->N7"
```

```
puts "CURRENT-VALUE :[$qmon8 set pdrops_] LINK N7->N8"
```

```
puts "CURRENT-VALUE :[$qmon9 set pdrops_] LINK N8->N9"
```

```
}
```

```

#packets departure
proc abc6 {} {
global qmon qmon2 qmon3 qmon4 qmon5
puts "OBJECT-NAME : PACKET DEPARTURE AT THE DESTINATION "
puts "_____ "
puts "CURRENT-VALUE :[$qmon set pdepartures_] LINK N1->N2"
puts "CURRENT-VALUE :[$qmon2 set pdepartures_] LINK N2->N3"
puts "CURRENT-VALUE :[$qmon3 set pdepartures_] LINK N3->N4"
puts "CURRENT-VALUE :[$qmon4 set pdepartures_] LINK N4->N5"
puts "CURRENT-VALUE :[$qmon5 set pdepartures_] LINK N5->N6"
}

```

```

#packets departure
proc ab6 {} {
global qmon qmon2 qmon7 qmon8 qmon9
puts "OBJECT-NAME : PACKETS DEPARTURE AT THE DESTINATION"
puts "_____ "
puts "CURRENT-VALUE :[$qmon set pdepartures_] LINK N0->N2"
puts "CURRENT-VALUES :[$qmon2 set pdepartures_] LINK N2->N3"
puts "CURRENT-VALUE :[$qmon7 set pdepartures_] LINK N3->N7"
puts "CURRENT-VALUE :[$qmon8 set pdepartures_] LINK N7->N8"
puts "CURRENT-VALUE :[$qmon9 set pdepartures_] LINK N8->N9"
}

```

```

proc abc8 {} {
global qmon qmon2 qmon3 qmon4 qmon5
puts "OBJECT-NAME : PACKETS ARRIVED FROM SOURCES"
puts "_____ "
puts "CURRENT-VALUE :[$qmon set barrivals_] LINK N1->N2"
puts "CURRENT-VALUES :[$qmon2 set barrivals_] LINK N2->N3"

```

```
puts "CURRENT-VALUE :[$qmon3 set barrivals_] LINK N3->N4"  
puts "CURRENT-VALUE :[$qmon4 set barrivals_] LINK N4->N5"  
puts "CURRENT-VALUE :[$qmon5 set barrivals_] LINK N5->N6"  
}
```

```
proc ab8 {}
```

```
{  
global qmon qmon2 qmon3 qmon7 qmon8  
puts "OBJECT-NAME : PACKETS ARRIVED FROM SOURCES"  
puts "_____  
puts "CURRENT-VALUE :[$qmon set barrivals_] LINK N0->N2"  
puts "CURRENT-VALUES :[$qmon2 set barrivals_] LINK N2->N3"  
puts "CURRENT-VALUE :[$qmon3 set barrivals_] LINK N3->N7"  
puts "CURRENT-VALUE :[$qmon7 set barrivals_] LINK N7->N8"  
puts "CURRENT-VALUE :[$qmon8 set barrivals_] LINK N8->N9"  
}
```

```
proc abc9 {}
```

```
{  
global qmon qmon2 qmon3 qmon4 qmon5  
puts "OBJECT-NAME : PACKETS DEPARTURE AT THE DESTINATION"  
puts "_____  
puts "CURRENT-VALUE :[$qmon set bdepartures_] LINK N1->N2"  
puts "CURRENT-VALUES :[$qmon2 set bdepartures_] LINK N2->N3"  
puts "CURRENT-VALUE :[$qmon3 set bdepartures_] LINK N3->N4"  
puts "CURRENT-VALUE :[$qmon4 set bdepartures_] LINK N4->N5"  
puts "CURRENT-VALUE :[$qmon5 set bdepartures_] LINK N5->N6"  
}
```

```

proc ab9 {} {
global qmon qmon2 qmon3 qmon7 qmon8
puts "OBJECT-NAME : PACKETS DEPARTURE AT THE DESTINATION "
puts "_____ "
puts "CURRENT-VALUE :[$qmon set bdepartures_] LINK N0->N2"
puts "CURRENT-VALUES :[$qmon2 set bdepartures_] LINK N2->N3"
puts "CURRENT-VALUE :[$qmon3 set bdepartures_] LINK N3->N7"
puts "CURRENT-VALUE :[$qmon7 set bdepartures_] LINK N7->N8"
puts "CURRENT-VALUE :[$qmon8 set bdepartures_] LINK N8->N9"
}

```

```

proc abc7 {}
{
global qmon1 qmon2 qmon3 qmon4 qmon5
puts "OBJECT-NAME : PACKETS DROPPED "
puts "_____ "
puts "CURRENT-VALUE :[$qmon1 set bdepartures_] LINK N1->N2"
puts "CURRENT-VALUES :[$qmon2 set bdepartures_] LINK N2->N3"
puts "CURRENT-VALUE :[$qmon3 set bdepartures_] LINK N3->N4"
puts "CURRENT-VALUE :[$qmon4 set bdepartures_] LINK N4->N5"
puts "CURRENT-VALUE :[$qmon5 set bdepartures_] LINK N5->N6"
}

```

```

proc ab7 {}
{
global qmon qmon2 qmon3 qmon7 qmon8
puts "OBJECT-NAME : PACKETS DROPPED "
puts "_____ "
puts "CURRENT-VALUE :[$qmon set bdepartures_] LINK N1->N2"
puts "CURRENT-VALUES :[$qmon2 set bdepartures_] LINK N2->N3"
puts "CURRENT-VALUE :[$qmon3 set bdepartures_] LINK N3->N7"

```

```
puts "CURRENT-VALUE :[$qmon7 set bdepartures_] LINK N7->N8"  
puts "CURRENT-VALUE :[$qmon8 set bdepartures_] LINK N8->N9"  
}
```

```
#Schedule events for the CBR agents
```

```
$ns at 0.5 "$cbr0 start"  
$ns at 1.0 "$cbr1 start"  
$ns at 4.0 "$cbr1 stop"  
$ns at 4.5 "$cbr0 stop"
```

```
#Call the finish procedure after 5 seconds of simulation time
```

```
proc mainfunc {}  
{  
puts "ENTER UR CHOICE"  
puts "1.LABEL-SWITCHED-PATH-ONE-OBJECTS"  
puts "2.LABEL-SWITCHED-PATH-SECOND-OBJECTS"  
puts "3.BANDWIDTH-CALCULATION-IN-MEGABITS"  
puts "4.VIEW THE GRAPH"
```

```
gets stdin first
```

```
if { $first == 1 } {  
"callinglsp"  
}
```

```
if { $first == 2 } {  
"startingslsp"  
}
```

```
I
```

```
if { $first == 3 } {  
  "setup"  
}
```

```
if { $first == 4 } {  
  "finish"  
}  
}
```

```
proc callingslp {} {  
  global qmon qmon1 qmon2  
  puts "ENTER OBJECT U WANT TO SEE IN LSP1"  
  puts "*****"  
  puts "a. MPLS VERSION NO"  
  puts "b. LSP FROM"  
  puts "c. LSP TO"  
  puts "d. PACKETS ARRIVAL"  
  puts "e. PACKETS DROPPED"  
  puts "f. PACKETS DEPARTURE"  
  puts "g. PACKETS DROPPED IN BYTES"  
  puts "h. PACKETS ARRIVED IN BYTES"  
  puts "i. PACKETS DEPARTURED IN BYTES"
```

```
  gets stdin objno  
  if { $objno == "a" } {  
    "abc1"  
  }  
  if { $objno == "b" } {  
    "abc2"  
  }  
}
```

```
if {$objno == "c"} {  
  "abc3"  
}
```

```
if {$objno == "d"} {  
  "abc4"  
}
```

```
if {$objno == "e"} {  
  "abc5"  
}
```

```
if {$objno == "f"} {  
  "abc6"  
}
```

```
if {$objno == "g"} {  
  "abc7"  
}
```

```
if {$objno == "h"} {  
  "abc8"  
}
```

```
if {$objno == "i"} {  
  "abc9"  
}
```

```
puts "DO U WANT TO CONTINUE y/n "
```

```
gets stdin ch
```

```

if { $Sch == "y" } {
"callinglsp"
}

if { $Sch == "n" } {
"mainfunc"
}
}

proc startinglsp {} {
global qmon qmon1 qmon2
puts "ENTER OBJECT U WANT TO SEE in LSP2"
puts "*****"
puts "a. MPLS VERSION NO"
puts "b. LSP FROM"
puts "c. LSP TO"
puts "d. PACKETS ARRIVAL"
puts "e. PACKETS DROPED"
puts "f. PACKET DEPARTURE"
puts "g. PACKETS DROPED IN BYTES"
puts "h. PACKETS ARRIVAL IN BYTES"
puts "i. PACKETS DEPARTURE IN BYTES"

gets stdin objno1
if { $objno1 == "a" } {
"ab1"
}
if { $objno1 == "b" } {
"ab2"
}
}

```



```
if {$objno1 == "c"} {  
  "ab3"  
}
```

```
if {$objno1 == "d"} {  
  "ab4"  
}
```

```
if {$objno1 == "e"} {  
  "ab5"  
}
```

```
if {$objno1 == "f"} {  
  "ab6"  
}
```

```
if {$objno1 == "g"} {  
  "ab7"  
}
```

```
if {$objno1 == "h"} {  
  "ab8"  
}
```

```
if {$objno1 == "i"} {  
  "ab9"  
}
```

```

puts "DO U WANT TO CONTINUE yes/no "
gets stdin ch
if { $ch == "yes" } {
"startingsp"
}
if { $ch == "no" } {
"mainfunc"
}
}

proc setup { } {
global sink0 sink1
    #Get an instance of the simulator
    set ns [Simulator instance]
    #Set the time after which the procedure should be called again
    set time 0.5
    #How many bytes have been received by the traffic sinks?
    set bw0 [$sink0 set bytes_]
    set bw1 [$sink1 set bytes_]

    #Get the current time

    set now [$ns now]
    #Calculate the bandwidth (in MBit/s) and write it to the files

    puts "BANDWIDTH USED BY SOURCE1 = $now [expr $bw0/$time*8/1000000]"
    puts "BANDWIDTH USED BY SOURCE2 = $now [expr $bw1/$time*8/1000000]"

puts "u want to continue yes or no "
gets stdin g2

```

```
if { $g2 == "yes" } {  
  "setup"  
}  
if { $g2 == "no" } {  
  "mainfunc"  
}  
}
```

```
#$ns at 1000.0 "finish"  
$ns at 4.0 "mainfunc"
```

```
#Run the simulation  
$ns run
```