TRAFFIC ENGINEERING PRIORITIZED IP PACKETS OVER MULTI-PROTOCOL LABEL SWITCHING NETWORKS

by

Danny Yip

BASc, University of British Columbia, 1999

PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF ENGINEERING

in the School

of

Engineering Science

© Danny Yip 2002

SIMON FRASER UNIVERSITY

April 2002

All rights reserved. This work may not be
reproduced in whole or in part, by photocopy
or other means, without permission of the author.

Approval

Name:	Danny Yip
Degree:	Master of Engineering
Title of thesis:	Traffic Engineering Prioritized IP Packets over Multi-Protocol Label Switching Networks

Examining Committee:

Chair: Dr. Glenn H. Chapman
Professor
School of Engineering Science
Simon Fraser University

Dr. Ljiljana Trajkovic
Senior Supervisor
Associate Professor
School of Engineering Science
Simon Fraser University

Dr. William A. Gruver
Supervisor
Professor
School of Engineering Science
Simon Fraser University

Date approved: March 26, 2002

Abstract

An emerging technology called Multi-Protocol Label Switching (MPLS) has been gaining considerable attention among Internet Service Providers (ISPs). MPLS enables ISPs to consolidate networks using different protocols into a unified label-switching network, and most importantly, to achieve better traffic engineering. The challenge of traffic engineering is how to make the most effective use of the available bandwidth in a large Internet Protocol (IP) backbone network so that the best performance characteristics from the same IP networks are achieved even in the event of congestion.

MPLS addresses the traffic engineering issue by setting up explicit paths through the network using constraint based routing. With constraint-based routing, a pre-determined path can be established based on the traffic's constraints, and packets are label-switched onto the pre-determined path to reach their destinations. Since the path selected via MPLS is established based on traffic's constraints, sufficient network resources are reserved along the selected path to handle the traffic. As a result, ISPs with MPLS can have higher control over their networks and provide better Quality-of-Service (QoS) to packet senders than using plain IP routing. Although MPLS supports two types of QoS: differentiated services and integrated services to packet senders, none of them offer QoS treatments based on a packet's Type of Service (ToS) field inside the Internet Protocol version 4 (IPv4) header field (or the Traffic Class (TC) inside the Internet Protocol version 6 (IPv6) header).

This report is focused on presenting the importance of prioritizing packets in traffic engineering and describes MPLS' deficiency in providing

QoS according to a packet's ToS (or TC) header value. In order to improve MPLS' capability in engineering prioritized IP traffic, I proposed a solution called "Priority-Bound and Explicit-Routed Label Switched Path" (PBER-LSP). PBER-LSP allows ISPs to force packets of different ToS (or TC) values to reach their destinations through some predetermined paths, and therefore ISPs can better allocate their network resources based on the priority of their traffic and offer more Classes of Service (CoS) to their customers. I implemented this solution into a network simulation tool called Network simulator, version 2 (ns-2). The simulation results demonstrate that MPLS-enabled network with PBER-LSP can better engineer prioritized IP traffic. Advantages and disadvantages of using PBER-LSP are discussed at the end of this report.

Acknowledgement

I would like to thankfully acknowledge the valuable comments and suggestions made from Dr. Trajkovic in preparation of this report. I would also like to express my gratitude to Winnie Woo for her support throughout the course of this project, and to Dr. Gruver and Dr. Chapman for spending their valuable time on reading the report and for being the members of the examining committee. This research is funded by Redback Networks Inc.

Table of Contents

List of Tables

List of Figures

Acronyms and Abbreviations

ATM	Asynchronous Transfer Mode
BW	Bandwidth
CoS	Class of Service
CR-LSP	Constraint-Based Routing – Label Switched Path
DSCP	Differentiated Services Code Point
ECMP	Equal-Cost Multipath
E-LSP	Experimental Inferred Label Switch Path
ERB	Explicit Route Information Base
ER-LSP	Explicit-Routed Label Switch Path
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IGP	Internet Gateway Protocol
IP	Internet Protocol (commonly refer to Internet Protocol version 4)
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISP	Internet Serviced Provider
L2	Layer 2
L3	Layer 3
LIB	Label Information Base
L-LSP	Label Inferred Label Switched Path
LSP	Label Switched Path
LSR	Label Switch Router
Mbps	Mega bits per second (1×10^6 bits / second)

MPLS	Multi-Protocol Label Switching
NAM	Network Animator
NFS	Network File System
ns-2	Network simulator, version 2
OPSF	Open Shortest Path First
PBER-LSP	Priority-Bound and Explicit-Routed – Label Switched Path
PDU	Packet Data Unit
PFT	Partial Forwarding Table
PHB	Per Hop Behaviour
QoS	Quality of Service
SNMP	Simple Network Management Protocol
TC	Traffic Class
TCP	Transmission Control Protocol
ToS	Type of Service
TTL	Time-to-Live

Introduction

Internet Service Providers (ISPs) around the world are struggling to keep up with the growth of the Internet traffic and customers' demands for more reliable and differentiated services. These demands require ISPs to maintain strict control over resource allocation and IP traffic flows throughout their networks. The process of mapping traffic flows onto physical topology of a network and allocating resources to these flows is called traffic engineering, and it is one of the most difficult tasks facing ISPs today [2, p. 2].

Whether a network is well traffic-engineered or not is determined by how efficiently ISPs utilize their infrastructures and, specifically, the available bandwidth. A well traffic-engineered network should be able to address the following three items:

- Gain better network utilization by avoiding situations in which some parts of network are congested while other parts are underutilized [1, p. 191].
- Create a more manageable network [2, p. 2].
- Segment different traffic types and provide them with appropriate Quality of Service (QoS) by allocating network resources in response to traffic requirements [2, p. 2].

Many ISPs today have been relying on the QoS capabilities offered by their Asynchronous Transfer Mode (ATM) backbones and running IP-over-ATM as a mean to engineer their networks. However, running IP-over-ATM is not an optimal solution because traffic engineering is limited to the ATM portion of the network and does not provide traffic-engineering benefits for the IP portion. Moreover, the combination of IP

and ATM within the same network requires the establishment of two independent topologies and hence presents many management and operational problems for ISPs [2, p. 2; 4, p. 16].

Another traffic engineering solution used by many ISPs is to over-engineer a plain IP network through manipulating link metrics and using equal-cost multipath (ECMP) capabilities. However, over-engineering a plain IP network is not a true traffic engineering solution, but rather a load sharing technique to minimize network congestion and to increase network utilization [2, p. 2; 1, p. 193].

In order to address the need for IP traffic engineering, the Internet Engineering Task Force (IETF) introduced a protocol known as "Multi-Protocol Label Switching" (MPLS) [3]. MPLS combines the deterministic traffic engineering control of Layer 2 ATM switching along with the flexible topology management of IP routing. It enables ISPs to consolidate networks using different technologies into an unified network, and most importantly allows ISPs to achieve true traffic engineering at the IP (Layer 3) level by:

- Employing "constraint-based routing" to determine path that meets the resource requirements of a traffic flow.
- Re-routing IP-traffic flows to other pre-determined paths when the network experiences any link or node failures. As a result, impact to those traffic flows of higher resource requirements is minimized.
- Providing two different models of QoS: Integrated Services and Differentiated Services to ISPs for segmenting different types of traffic and allocating network resource accordingly and efficiently.

Although MPLS offers many IP traffic engineering capabilities that IP-over-ATM and plain IP routing cannot, MPLS lacks the ability to provide QoS to packets based on their priority levels assigned by their senders. In this report, prioritized packets are called IP packets whose 8-bit wide "Type of Service" (ToS) field in their IPv4 headers (or 8-bit wide "Traffic Class" (TC) field in their IPv6 headers) are assigned to certain values by the senders and are not replaced by Differentiated Services Code Point (DSCP) before entering an MPLS-enabled network.

Scope of Report

This report is divided into four sections: Section 1 discusses different priorities defined in the IP, with emphasis on IPv6, and their importance in traffic engineering. Section 2 briefly reviews the MPLS protocol, constraint-based routing, and how networks that use the combination of MPLS and constraint-based routing can offer better traffic engineering than plain IP routing. Section 2 also examines how MPLS currently treats prioritized IP packets in order to achieve QoS, and lists some of MPLS' limitations in providing appropriate QoS treatments based on ToS (or TC) header field of each packet. In Section 3, I propose a solution called "Priority-Bound and Explicit-Routed Label Switched Path" (PBER-LSP), which can enhance MPLS' capability in providing different QoS to packets according to their priority levels. This solution was incorporated into a network simulation tool called "Network simulator, version 2" (ns-2) and modifications to ns-2 are briefly discussed in Section 3. Finally, in Section 4, the effectiveness of PBER-LSP was demonstrated via simulations using the modified ns-2, and the simulation scenarios and results are discussed. Advantages and

disadvantages of using PBER-LSP are also discussed at the end of Section 4.

It should be noted that readers of this report are expected to have basic knowledge of networking and IP routing protocols. In addition, this report assumes that MPLS' underlying backbone is neither an ATM nor Frame Relay, although deploying MPLS over ATM or Frame Relay is a valid engineering approach.

1 Internet Protocol and Prioritized IP Packets

Internet Protocol version 4 (commonly referred as "IPv4" or simply "IP") has been the underlying protocol of today's Internet. The IPv4 header has an 8-bit wide field called "Type of Service" (ToS) that allows senders to prioritize and classify their packets according to their traffic types. However, at the time when IPv4 was widely implemented and deployed, the ToS field was not well defined. As a result, today's networking equipment either 1) ignores this field and treats all IPv4 packets equally, or 2) relies on other mechanisms, such as integrated services and differentiated services, to offer QoS to their users. The introduction of the Traffic Class (TC) field in Internet Protocol version 6 (IPv6) addresses this deficiency. The TC field enables senders to prioritize packets that belong to particular traffic flows and allows ISPs to provide QoS according to the priority level assigned to each packet, rather than relying on other mechanisms like integrated services or differentiated services that IPv4 uses. In order to understand the importance of prioritizing packets in the context of traffic engineering, it is important to understand the proposed definition of each priority defined in IPv6.

1.1 Properties of Prioritized Packets

Initially, the IPv6 Specification – RFC 1883 [4] defines the provisional description of the internal structure and semantics of the TC field, however the latest version of the IPv6 Specification – RFC 2460 [4] removes the descriptions and it states that such descriptions will be provided in separate documents. Nevertheless, such documents have

not yet been released when this report is prepared. The following outlines what the TC field was initially defined in IPv6. According to RFC 1883, the TC field is separated into two categories: 1) Congestion-controlled traffic and 2) Non-congestion-controlled traffic:

1.1.1 Congestion-Controlled Traffic:

Congestion-controlled traffic refers to traffic for which the source decreases its transmission rate in response to congestion. Such mechanism exists in protocols such as Transmission Control Protocol (TCP). Because of the nature of congestion-controlled traffic, a variable delay in the delivery of packets is acceptable. IPv6 further classify congestion-controlled traffic into the following six categories:

- **TC = 0, Uncharacterized traffic:**
  If the upper-layer application gives IPv6 no guidance about traffic priority, then those packets should be assigned the lowest-priority value and will be delivered based on the available bandwidth.
- **TC = 1, Filler traffic:**
  These applications generate traffic handled in the background, while other types of traffic are delivered. Those applications include newsgroups' messages and electronic mail.
- **TC = 2, Unattended data transfer:**
  These applications generate packets that are not delivered instantly. The best example of this category is electronic mail. Generally, the user does not wait for the transfer to be complete, and therefore, those packets can experience longer delay.
- **TC = 4, Attended bulk transfer:**

These applications may involve the transfer of a large amount of data. Those applications include File Transfer Protocol (FTP) and Hypertext Transfer Protocol (HTTP). During these application sessions, users are generally prepared to accept a larger amount of delay, and therefore, packets whose TCs equal to 4 can be delivered at a lower priority than packets whose TCs set to 6 and higher.

- **TC = 6, Interactive traffic:**

  Interactive traffic, such as telnet, requires rapid response time. It is crucial to minimize the packet delay during interactive session because a user is interacting with the host in real-time.

- **TC = 7, Internet control traffic:**

  This is the most important traffic to deliver in the congestion-controlled traffic category, especially during times of high congestion, because the content of this traffic type contains information and instructions regarding the network status and any topology changes. For example, protocols such as Open Shortest Path First (OSPF) and Simple Network Management Protocol (SNMP) constantly transmit packets to network equipment peers concerning traffic conditions. The network equipment will then use the information to adjust routing path and to perform dynamic reconfiguration in response to congestion conditions.

Table 1 summarizes examples of applications that correspond to each TC in the congestion-controlled traffic category. It should be noted that Traffic Classes 3 and 5 have been reserved and nothing was yet defined for these two classes.

TC	Description
0	Uncharacterized traffic
1	Filler traffic (Netnews)
2	Unattended data transfer (email)
3	Reserved
4	Attended bulk transfer (FTP, NFS)
5	Reserved
6	Interactive traffic (telnet)
7	Internet control traffic (routing protocols, SNMP)

Table 1: Traffic class descriptions for congestion-controlled traffic.

1.1.2 Non-Congestion-Controlled Traffic:

Non-congestion-controlled traffic has smooth data rates and requires relatively constant delivery delay. Examples are real-time video and audio applications, which maintain smooth delivery flow and do not require retransmission because it is useless to re-transmit voice or real-time video signal that was dropped few seconds earlier and to replay it in the middle of a real-time conversation.

Eight levels of priority have been allocated for this type of traffic, from the lowest priority 8 (most willing to discard) to the highest priority 15 (least willing to discard). In general, the criterion is how severely the application is affected when packets are dropped. For example, packets

carrying data of a telephone voice conversation would typically be assigned a high priority because human ears are more sensitive to audio signal loss. On the other hand, video signal contains redundant information between frames, and the loss of a few packets carrying video data will most likely not be noticeable. Therefore, video traffic is assigned a relatively low priority. Table 2 summarizes the different TC values for non-congestion-controlled traffic and examples of applications that corresponds to each TC categories.

TC	Description
8	Lowest priority, most willing to discard (video traffic).
9–14	…
15	Highest priority, least willing to discard (audio traffic).

Table 2: Traffic class descriptions for non-congestion-controlled traffic.

Although there are distinctions between congestion-controlled traffic and the non-congestion-controlled traffic, there is no priority relationship implied between the two traffic categories. Priorities are relative only within each category.

1.2 Importance of Prioritizing Packets in Traffic Engineering

The importance of prioritizing packets in traffic engineering is apparent because, given that packet senders fairly classify and prioritize their packets based on their traffic requirements, ISPs can use priority information of each packet to:

1. Provide end-to-end QoS between the sender and the receiver;
2. Maximize their network utilization and offer more Classes of Services (CoS) to their customers by allocating resources to packets based on their priorities;
3. Segment traffic of different priorities and provide appropriate QoS to them accordingly;
4. Minimize impact to packets of higher priorities during network failures.

2 Multi-Protocol Label Switching

It has been demonstrated the different priorities defined in IPv6 and their importance in the context of traffic engineering, but before looking into MPLS' limitations in providing QoS treatments to packets based on their assigned priorities, a brief overview on how MPLS achieves basic traffic engineering and QoS is discussed. In order to facilitate the explanation, Figure 1 is used throughout this section:

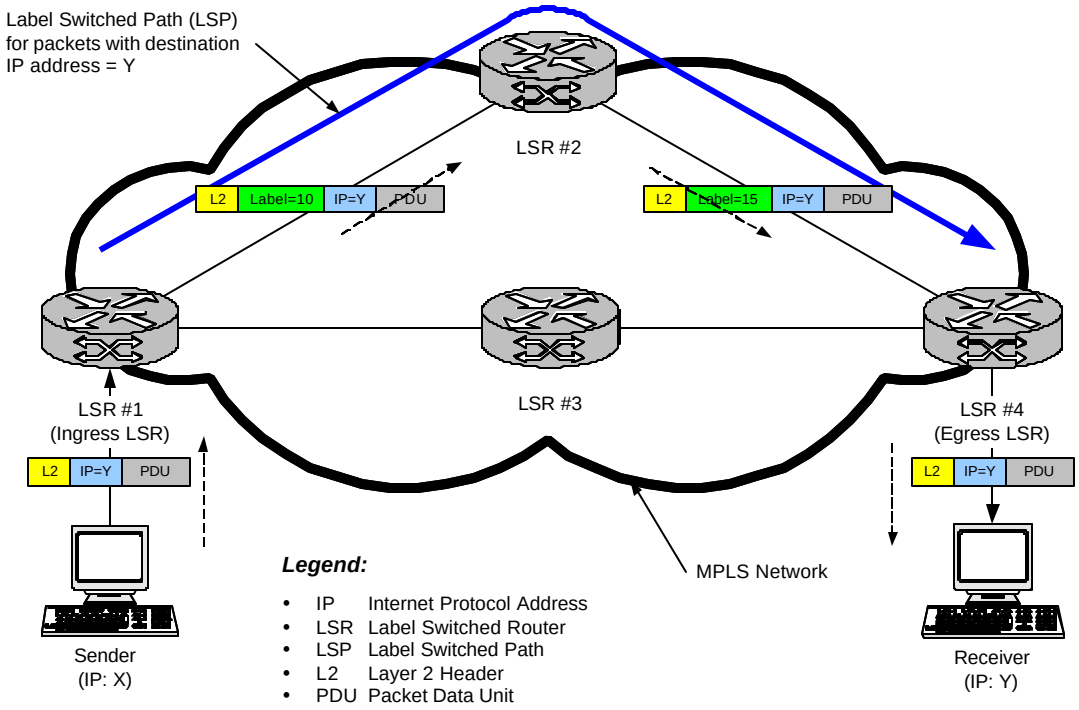
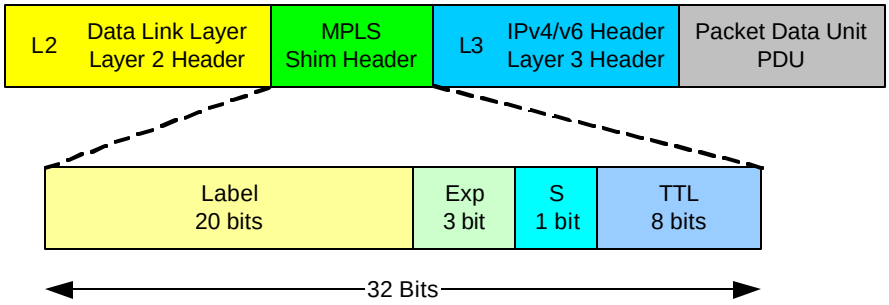


Figure 1: A simplified network for MPLS demonstration.

2.1 MPLS – A Label Switching Technology

Multi-Protocol Label Switching (MPLS) is a label switching technology introduced by IETF and has been gaining considerable attention among ISPs. MPLS enables ISPs to consolidate networks using different technologies into an unified network, and most importantly allows ISPs to achieve true traffic engineering at the IP (Layer 3) level. MPLS uses constraint-based routing to determine paths that a traffic flow will traverse and label-switch packets to their destinations using the pre-determined paths. These paths, known as Label Switched Path (LSP), may have dedicated resources associated with them and can be explicitly determined (known as explicit routing) or implicitly created (known as implicit routing). Once LSPs have been established, packets received by Label Switch Router (LSR) are mapped onto the paths according to 1) their traffic requirements, 2) the information carried in their IP headers, and/or 3) the local routing information.

Before an IP packet can be label-switched onto a LSP, the LSR at the head of the LSP (known as Ingress LSR) assigns a label representing the LSP to the packet by inserting a shim header to it. The shim header is shown in Figure 2. Each shim header contains a 20-bit label, a 3-bit Experimental field, a 1-bit label stack indicator and an 8-bit Time-to-Live (TTL) field. Once a packet is labeled, the label inside the shim header, instead of the traditional IP destination address, is used as the reference to deliver the packet along the selected path.



- The Label field stores the label number associated to Label Switch Path (LSP) that the packet will traverse.
- The Exp field is "Experimental", though it has been proposed to indicate Per Hop Behavior (PHB) of labeled packets.
- The Stack (S) bit indicates the presence of stacked label.
- The Time-to-Live (TTL) field is initially set to a pre-determined value and is decremented at each LSR hop. It is used to avoid looping packets.

Figure 2: MPLS shim header.

As an MPLS-labeled packet arrives at a LSR, the "label" field inside the shim header is used to determine the next hop of each packet by looking up the forwarding table resided in each LSR. Before switching the packet to the next hop, LSRs can replace the incoming label by another label or stack a new shim header on top of the incoming shim header, depending on the forwarding table in each LSR. In the case of stacking a new shim header on top of the incoming shim header, the stack (S) bit inside the newly inserted shim header will be set as an indication of such an event. In addition, the TTL field in the shim header is decremented at every hop.

Finally, if the destination of an MPLS-labeled packet is inside the MPLS-network, the shim header of the packet will be removed as it reaches its destination. However, if an MPLS-labeled packet's TTL field is decremented to zero before it reaches its destination or when an MPLS-

labeled packet leaves the MPLS-network, the egress LSR will remove the shim header from the packet and the packet will be dropped to avoid looping or routed as an ordinary packet, respectively.

2.2 Constraint-Based Routing

By label switching itself, MPLS does not satisfy the need for traffic engineering. In order to achieve IP traffic engineering, MPLS relies on constraint-based routing to create LSPs in an IP network by applying a series of constraints to them. The constraints that are used to establish LSPs vary widely, but they can be divided into three categories: explicit, implicit and mixed routing.

2.2.1 Explicit Routing

Explicit routing is a method of pre-selecting and specifying an exact path for traffic to flow through the network. The benefit is that ISPs determine exactly how traffic will flow through their networks rather than relying on routing protocols, such as OSPF and Internet Gateway Protocol (IGP), which tend to send all traffic across common links while potentially leaving other links idle. In addition, resources can be allocated to the explicit routes based on the traffic requirements. One potential drawback is that explicit routing is primarily a manual process, which requires a fair amount of effort and is hence not dynamic in nature [2; p. 3].

2.2.2 Implicit Routing

Alternatively, implicit routing is a method of routing traffic through the network based on the forwarding requirements of the traffic. The

forwarding requirements typically take the form of a quantitative requirement, such as a specific amount of bandwidth. The implicit routing process then selects a path through the network that is capable of satisfying the specified forwarding requirements and then dedicates resources along the path on behalf of the traffic.

The benefit to the implicit routing method is that the manual effort is limited to specifying requirements for different traffic types, without having to specify specific paths through the network. Implicit routing calculates the paths by an automated process, thus providing a dynamic method to traffic engineering a network [2; p. 3].

2.2.3 Mixed Routing

Constraint-based routing may also include a combination of both explicit and implicit routing. The mechanism of establishing a LSP using a combination of both explicit and implicit routing is called mixed routing. For example, an ISP may explicitly specify which LSRs the traffic should not flow through, and then may rely on the implicit routing to determine the best LSP that satisfies the constraints.

2.3 Traffic Engineering with Constraint-Based Routing

Because MPLS constraint-based routing considers more than network topology in computing routes, an MPLS-enabled network can avoid situations in which some parts of its network are congested while other parts are underutilized. The following sections discuss how ISPs can better traffic engineer their networks with MPLS constraint-base routing.

2.3.1 Maximizing Network Utilization

In order to maximize the network resource utilization, MPLS constraint-based routing may switch traffic through a longer but lightly loaded path rather than the shortest but heavily loaded path, and yet meeting all forwarding requirements. In the example shown in Figure 3, packets destined to receiver (Y) are sent to LSR #1 by sender (X) at a rate of 40 Mbps. If plain IP routing were used, the traffic flow would be routed through the link 1-2 because of the following two reasons:

- It is the shortest path between LSR #1 and the receiver, as the traffic flow requires only two hops to reach its destination (based on OSPF).
- The sum of all link metrics (m) of this path is equal to one (based on Internet Gateway Protocol (IGP)), which is the smallest among all possible paths. With plain IP routing, the traffic will experience high packet losses and end-to-end delay because the path between LSR #1 and LSR #3 does not have sufficient.

With MPLS and constraint-based routing, ISPs can easily resolve this problem by using explicit routing. ISPs can explicitly force the traffic flow to always traverse LSR #2, as the link 1-2-3 have sufficient reservable bandwidth to handle the bandwidth constraints. Alternately, if implicit routing is used, since the amount of reservable bandwidth on the shortest path is only (622-600) = 22 Mbps, the implicit routing mechanism will select the link 1-2-3 instead, because the shortest path (link 1-3) does not meet the bandwidth constraint.

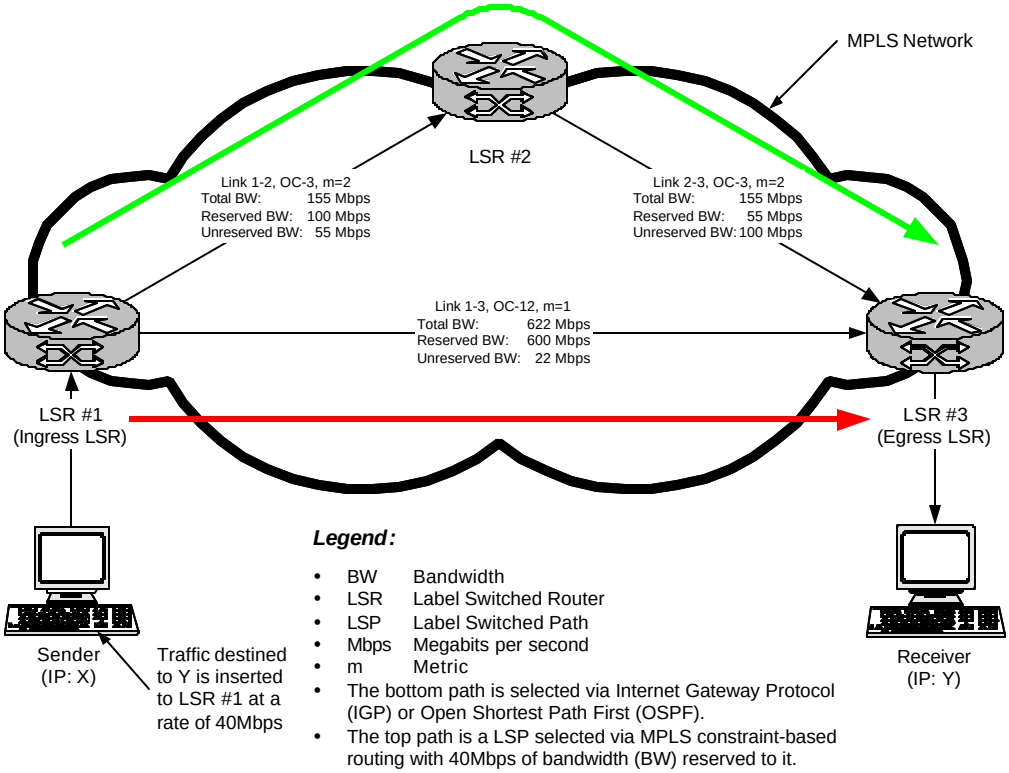


Figure 3: Illustration of constraint-based routing.

2.3.2 Improving Network Robustness

Plain IP network often experiences high packet losses because of links or routers (or both) failures. The amount of packet losses due to a link or router failures mainly depends on the amount of time it takes to distribute the failure event among all the routers and to re-compute an alternative path [1; p. 201]. Typically, with OSPF, the time to converge and determine an alternative path is on the order of seconds, which means packet losses may last at least several seconds. Even when an alternative path is recomputed, the path might be heavily loaded and the

network may continue to experience high packet losses because the alternative path does not have sufficient bandwidth to handle the extra traffic.

A well traffic-engineered network should be able to minimize impacts to traffic in the events of link or router failures, and to maintain a robust and deterministic network during the failures. MPLS constraint-based routing addresses the issue by explicitly pre-establishing a "protection" LSP around a "working" LSP. When any links or LSRs along the working LSP fail, traffic used to traverse the working LSP will take the protection LSP instead.

For example in Figure 4, packets destined to receiver (Y) are sent to LSR #1 by sender (X) at a rate of 50 Mbps. LSP #1 (link 1-4-5) is assigned for delivering the packets to their destinations. In order to handle failures along LSP #1 (link 1-4-5), a protection LSP (labeled as LSP #2 in Figure 4) is pre-established along link 1-2-3-5 and a bandwidth of 50 Mbps is reserved along the link via explicit routing.

If the link 1-2 goes down, LSR #1 will detect the failure and will change the label-forwarding table such that traffic used to traverse LSP #1 will use LSP #2 instead. Although packets will take one more hop to reach their destination, the network is robust and deterministic. However, if there was not another LSP protecting LSP #1 when the link failure occurred, LSR #1 would select the shortest path between LSR #1 and LSR #5 (the link 1-5 in Figure 4) computed by the OSPF protocol. Since link 1-5 does not have sufficient available bandwidth to handle the extra traffic, the link could be congested while leaving link 1-2-3-5 underutilized.

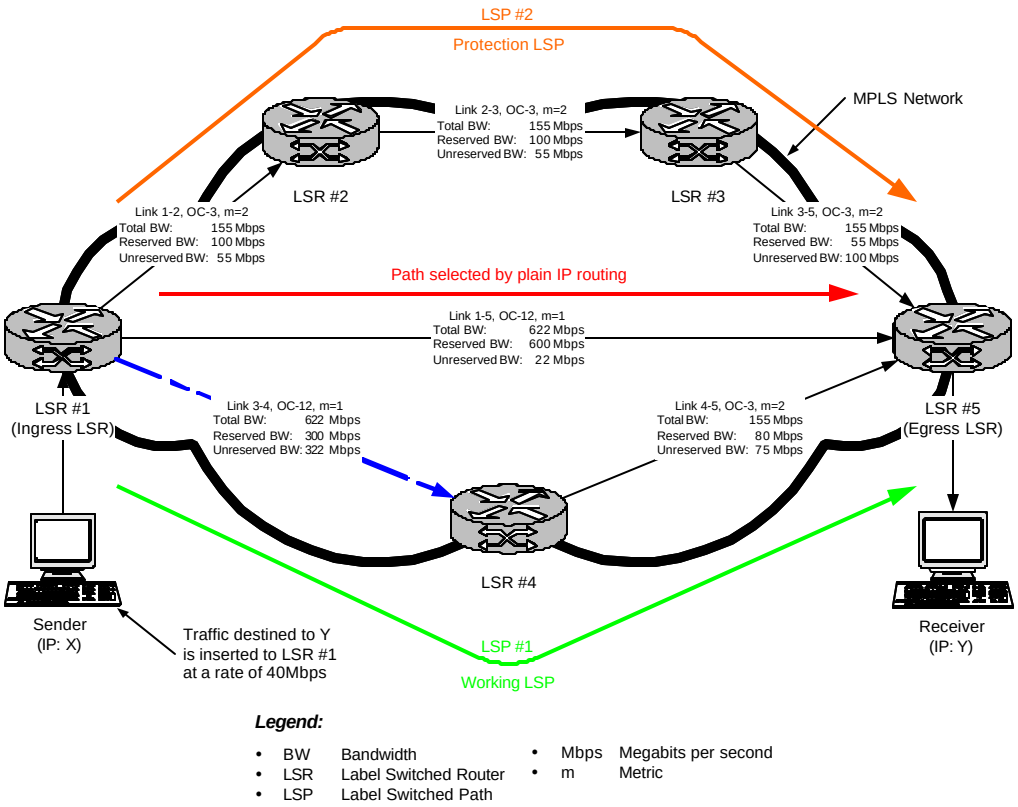


Figure 4: Traffic protection with MPLS constraint-based routing.

2.4 MPLS' QoS Capabilities and their Limitations

It has been demonstrated that MPLS is suited for traffic engineering, but much more can be accomplished when MPLS is tied to Quality-of-Service (QoS). MPLS supports two QoS capabilities commonly used in today's best effort IP network: Integrated Services and Differentiated Services. This section discusses MPLS' two QoS capabilities and their limitations in providing appropriate QoS treatments to prioritized packets.

2.4.1 MPLS and Integrated Services

Integrated Services was introduced by the IETF in the mid-1990 with the goal of enabling flow based QoS to IP traffic. Flow based QoS allows ISPs to offer end-to-end QoS to applications by reserving sufficient resource and bandwidth between the sender and the receiver. Once the reservation is done, routers between the sender and the receiver recognize packets that belong to the reservation by inspecting the IP and transport protocol headers. MPLS supports integrated services by establishing one LSP per traffic flow via constraint-based routing, and since the LSP is established according to the traffic flow's specifications, QoS is always guaranteed along the LSP.

Although integrated services can guarantee end-to-end QoS to packet senders, it can only be guaranteed on a per-flow basis, and packets of different priorities within the same flow are treated equally. As a result, higher priority packets within a flow cannot be distinguished from those that can tolerate higher delay, thus ISPs using integrated services cannot provide different QoS treatments to prioritized packets within a traffic flow.

2.4.2 MPLS and Differentiated Services

Since Integrated Services can only guarantee QoS at the traffic flow level, another mechanism called Differentiated Services was introduced to offer QoS treatment at the packet level, and such mechanism is supported by MPLS. Differentiated services is known as class based QoS and it involves marking each packet with a tag indicating the requested QoS treatment on a per-hop basis. The tag is called Differentiated

Services Code Point (DSCP), and is carried in the IP header by replacing the Type of Service field (ToS) in IPv4 or the Traffic Class (TC) field in IPv6 (please refer to Figure 5).

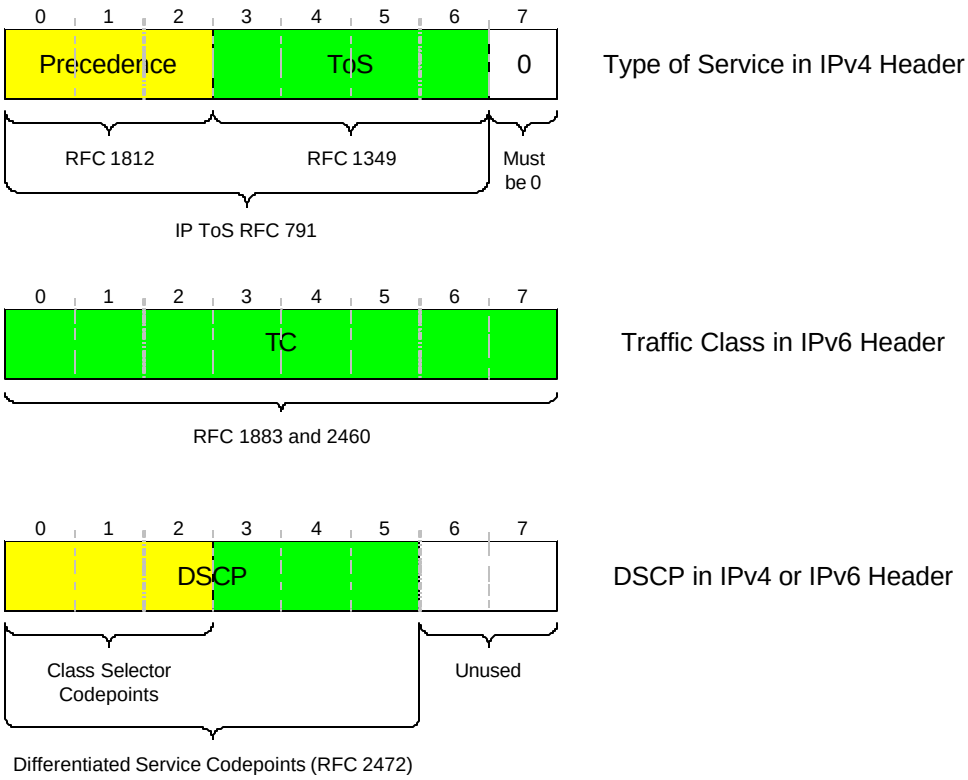


Figure 5: DSCP in IPv4 and IPv6 header.

Currently there are four proposed mappings of DSCP which together equal twenty-two unique code points [5]:

- **DE: Default**

  No special treatment is required and it is equivalent to the best effort class (one code point).

- **CS: Class Selector**

  Backward Compatible with IP Precedence (eight code points);

- **EF: Expedited Forwarding**

  An approximation of constant bit services, with minimal delay and low packet loss (one code point);

- **AF: Assured Forwarding**

  Four hierarchical classes are define and there are three drop-precedences within each class (for a total twelve code points).

When a router receives a DSCP-labeled packet, the router classifies the packet by looking up the DSCP in the IP header and determines the QoS treatment accordingly. This approach provides Per Hop Behaviour (PHB) as opposed to integrated services with explicit reservation where QoS is guaranteed on a per-flow basis.

MPLS supports differentiated service (class based QoS) by providing two kinds of LSPs for aggregating DSCP marked packets into MPLS LSP:

- Experimental Inferred Label Switched Path (E-LSP):

  E-LSP is a label switched path where DSCP code is mapped to the experimental field of the MPLS shim header and QoS treatment to MPLS labeled packets is determined individually by examining the experimental (Exp) field of the MPLS shim header (please refer to Figure 2: MPLS shim header.). Since the Exp field is 3-bit wide, each E-LSP can map up to eight possible DSCP. The Ingress LSR makes the mapping of DSCP to Exp and scheduling of MPLS-labeled packets is determined individually at each hop along the E-LSP. Since E-LSP can only map at most eight different DSCPs, L-LSP is used when more than eight different levels of QoS is to be offered.

- Label Inferred Label Switched Path (L-LSP):

  Instead of using the MPLS shim header to classify different QoS requirements of each packet, ingress LSR examines the DSCP in the IP header and selects a LSP that meets the specified QoS. The selected LSP is called Label Inferred Label Switched Path (L-LSP) because the label is used to reference packet's QoS requirement. At the egress LSR, the last label is removed and the packet is sent to the next IP hop with its original DSCP. This method requires that an association of specific DSCP to LSPs be pre-established prior to traffic flow.

Although differentiated-services provide class-based QoS and allow ISPs to offer some levels of traffic engineering on a per-packets basis, there are still some limitations. Consider the example shown in Figure 6. In order to support differentiated services, those packets that require special QoS treatments must have the ToS field or TC field in their IPv4 or IPv6 headers replaced by a "proper" DSCP before entering the MPLS network. Since the ToS (or TC) field is permanently replaced by DSCP, the originally assigned priority of each packet is lost as it reaches its destination. This might not be significant in IPv4 because the ToS header field was not well defined at the time when IPv4 was widely deployed and implemented. However, as discussed in "Section 1 – Internet Protocol and Prioritized IP Packets", more effort was put in defining IPv6's TC header field for identifying a packet's nature and priority level, and permanently replacing the TC field by DSCP can be significant in the IPv6 domain. In addition, since QoS treatment to each packet is decided at each LSR along the LSP, the quality level provided by

each LSR might not be consistent and therefore end-to-end QoS cannot be guaranteed to customers.

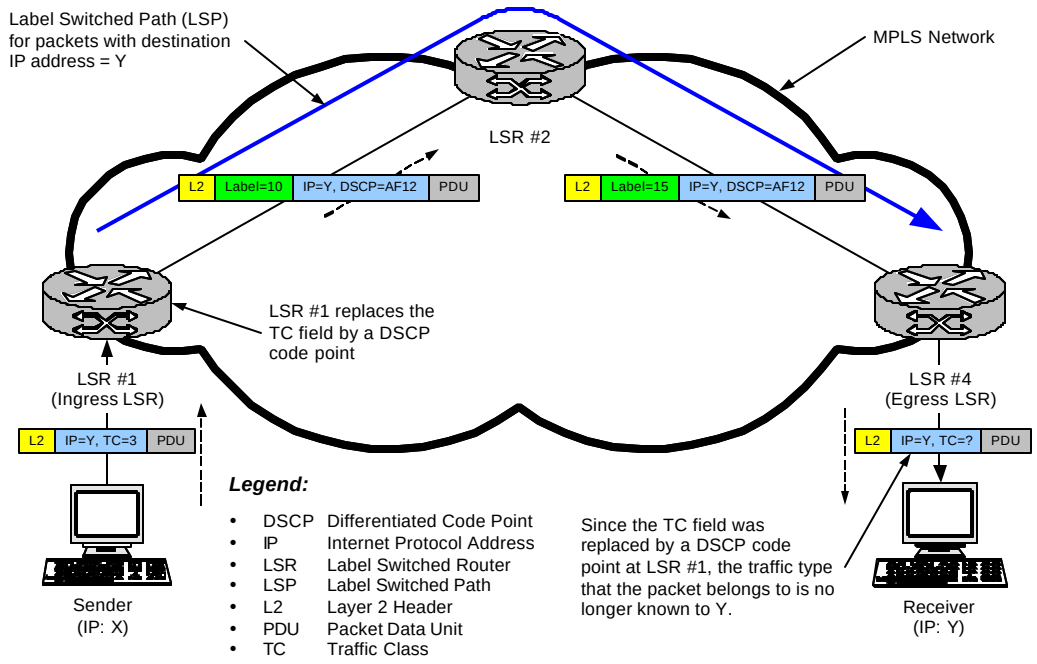


Figure 6: MPLS network with differentiated services.

2.4.3 Ideal Solution for Traffic Engineering Prioritized IP Packets

An ideal solution of traffic engineering prioritized packets should enable ISPs to achieve the following three goals:

- Offer end-to-end and appropriate levels of QoS to prioritized packets within a given traffic flow;
- Determine appropriate QoS treatment to each prioritized packet based on the original priority level or traffic class assigned to the packet;

- Improve network robustness by minimizing impacts to packets of higher priority during node or link failures;

I propose a solution called “Priority-Bound and Explicit-Routed – Label Switched Path”, which can achieve all three goals listed above, and it is discussed in the following section.

3 Implementation of Priority-Bound and Explicit-Routed Label Switched Path (PBER-LSP) in ns-2

In order to address MPLS' limitations in providing proper QoS treatments to prioritized packets, especially in the IPv6 domain, I propose a solution called "Priority-Bound and Explicit-Routed – Label Switch Path" (PBER-LSP). By integrating a new attribute "priority" into the existing explicit-routing mechanism, PBER-LSP enables ISPs to allocate network resource and to offer end-to-end QoS based on packet's priority assigned by its sender.

3.1 PBER-LSP – Enhancement of MPLS' Existing Explicit-Routed Label Switched Path

As discussed in Section 2.4.3, a well traffic-engineered network that carries prioritized traffic should be able to do the following three things: 1) offer end-to-end and appropriate levels of QoS to prioritized packets within a given traffic flow, 2) determine appropriate QoS treatment to each prioritized packet based on the original priority level (or traffic class) assigned to the packet, and 3) improve network robustness by minimizing impacts to packets of higher priority during node or link failures. These are precisely the capabilities that can be provided by enhancing MPLS' explicit-routing mechanism. Recalling the description of explicit-routing in Section 2.2.1, explicit-routing is a method of pre-selecting a LSP for traffic to flow through the network. Since the LSP is pre-selected via constraint-based routing, network resources (e.g., bandwidth) are reserved based on the traffic's constraints along the explicit-routed LSP (ER-LSP), and packets that traverse the ER-

LSP are treated equally regardless of their priority levels. In order to enhance MPLS' capability in providing appropriate QoS treatments to prioritized packets, I proposed a modified version of ER-LSP called "Priority-Bound and Explicit-Routed LSP" (PBER-LSP) where the ER-LSP is bound to certain priority levels and only those packets whose priorities match the LSP's bound priorities can traverse through the LSP. With this enhancement, ISPs can route prioritized packets to pre-determined LSPs that meet the QoS requirement of each packet, and minimize impacts to those higher priority packets during node or link failures.

3.2 Incorporating PBER-LSP into "Network Simulator, version 2"

In order to illustrate the PBER-LSP concept, I incorporated this idea into the Network simulator, version 2.1b8a-win (ns-2) for demonstration. The selection of ns-2 is obvious because it already supports some of MPLS' basic features such as explicit routing. This section briefly discusses the MPLS architecture in ns-2 and what I modified in ns-2 to implement the PBER-LSR concept.

3.2.1 MPLS Architecture in ns-2

The MPLS architecture in ns-2 consists of three classifiers: MPLS Classifier, Address Classifier, Service Classifier, and three information tables: Partial Forwarding Table (PFT), Label Information Base (LIB), Explicit Route Information Base (ERB).

Figure 7 and Figure 8 illustrate the relationships and interactions between the classifiers and information tables.

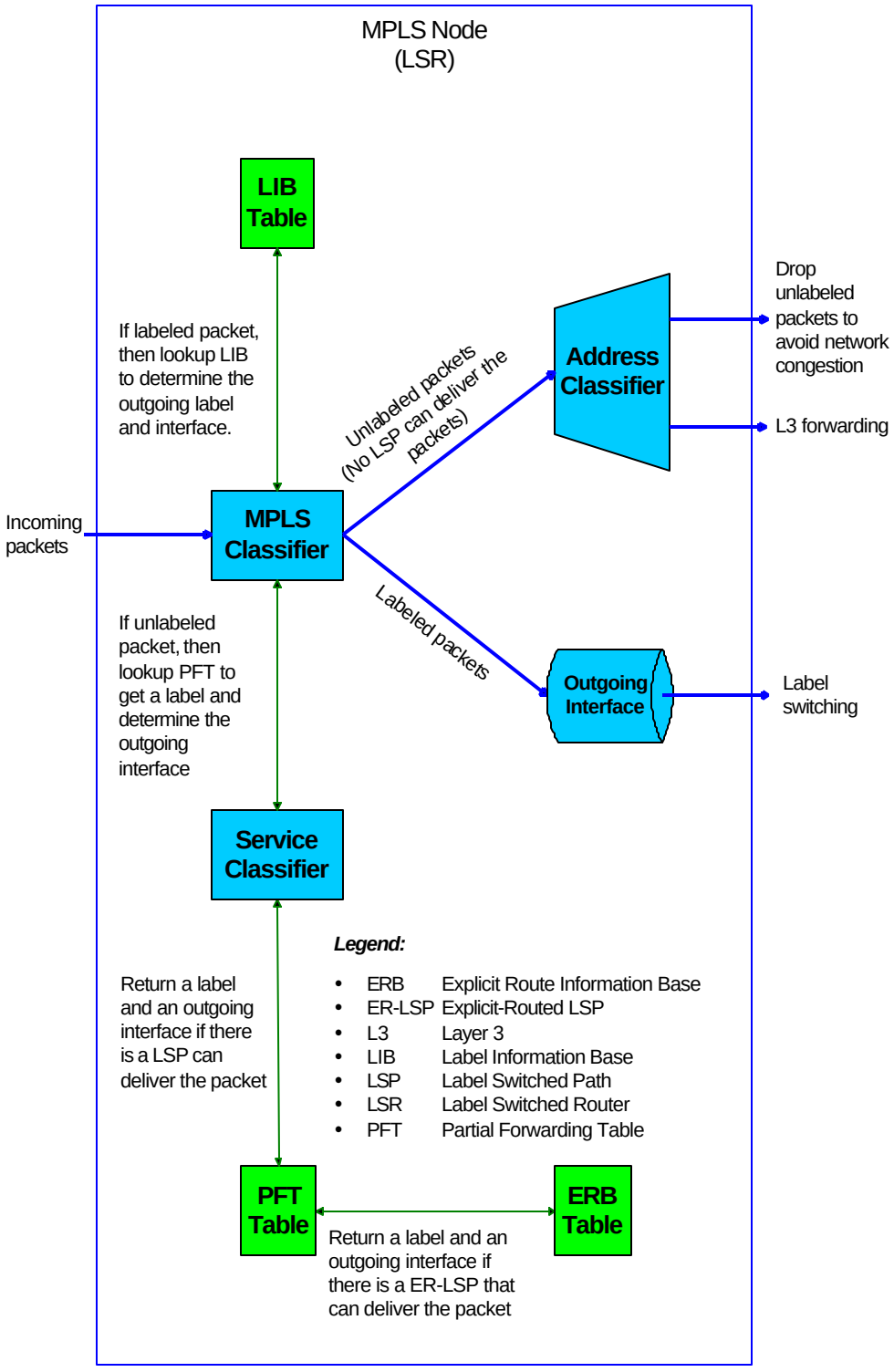


Figure 7: Components and their relationships of the MPLS architecture in ns-2.

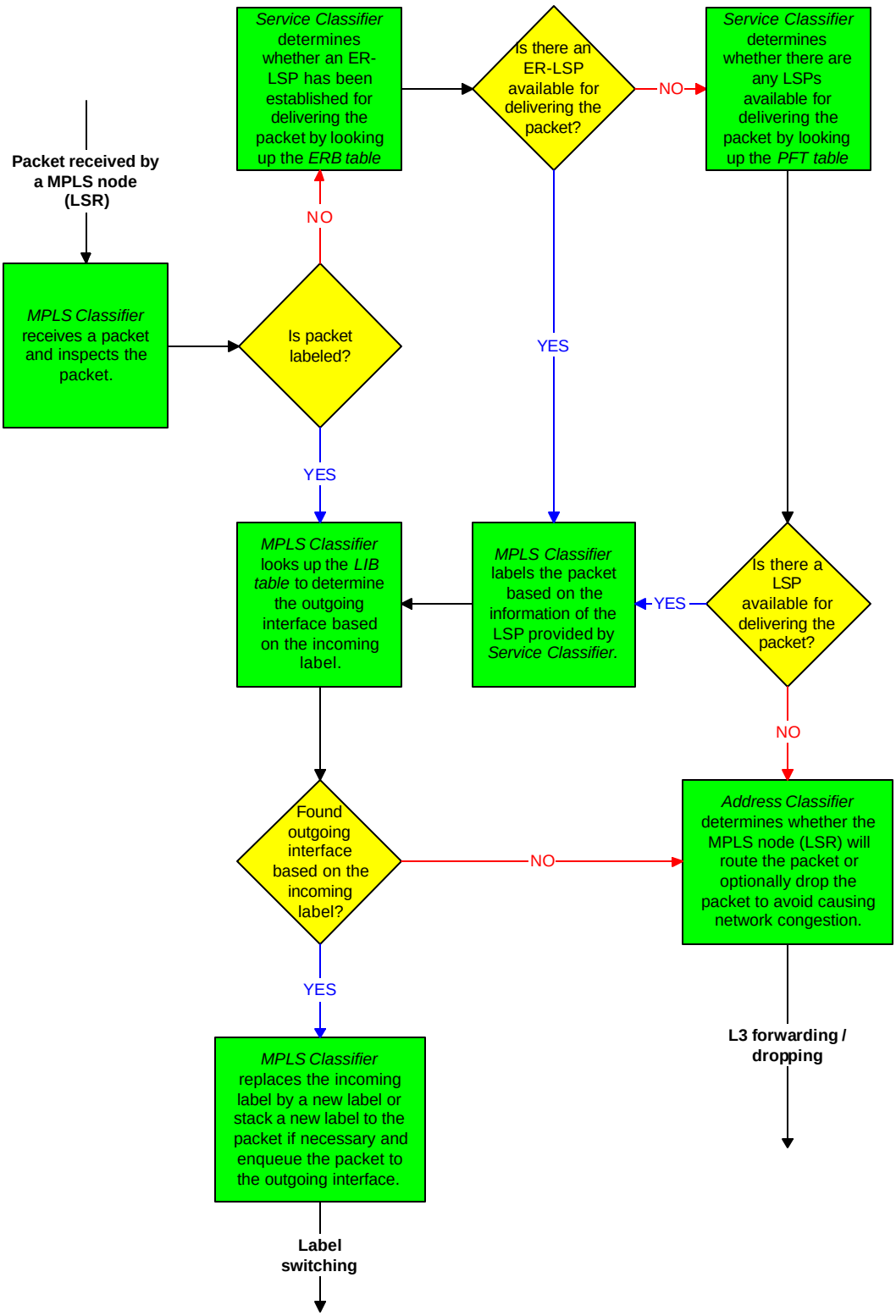


Figure 8: Operational flow-chart of the MPLS node in ns-2.

When a MPLS node (LSR) receives a packet, the LSR will perform the following operation [9]:

1. The *MPLS Classifier* determines whether the received packet is labeled with a shim header.
2. If it is labeled, the *MPLS Classifier* determines the outgoing interface and label of the packet by looking up the *LIB table*, and then executes label switching for the packet. The *LIB table* has information of all established LSPs. It consists of in/out-label and in/out-interface.
3. If it is not labeled but an explicit-routed LSP (ER-LSP) was pre-established for routing the packets, *Service Classifier* would return information about the ER-LSP to the *MPLS classifier* by looking up the *ERB table*, and the *MPLS classifier* would label the packet based on the information. The *ERB table* maintains information of all established ER-LSP.
4. If it is not labeled but an LSP whose path can deliver the packet, the packet will be labeled with a proper shim header after looking up the *PFT table* and then will be handled like a labeled packet by the *MPLS Classifier*.
5. Otherwise, the MPLS node will behave as an IP router and the *Address Classifier* will forward the packet using layer 3 routing protocol or optionally drops the packet to avoid causing traffic congestion to the network

3.2.2 Modifications Made to ns-2

In order to implement the PBER-LSP in ns-2, I modified ns-2's existing mechanism that routes packets to an ER-LSP. The following

discusses some of the major modifications to the ns-2 source code. For details of all the modifications to the ns-2 source code, please refer to "Appendix A – Modifications to ns-2 Source Code" on page 50.

- **agent.h and agent.cc**

  These two files were modified so that users can use the ns-2 simulator to send prioritized packets into the simulation network. Packets' priorities are user-configurable, and they can either be fixed to an user-predefined value, or uniformly distributed between 0 and 15, except 3 and 5 (please refer to "Section 1 – Internet Protocol and Prioritized IP Packets" for details about the different priority levels and why priority levels 3 and 5 are not part of the distribution).

- **trace.cc**

  This file was updated so that the priority of each packet is logged to the trace file when the packet is enqueued, dequeued, or dropped by a node. With this change, more information is collected and therefore more analysis can be performed with the simulation results.

- **classifier-addr-mpls.h and classifier-addr-mpls.cc**

  In order to incorporate the PBER-LSP concept into the existing explicit-routing mechanism, a new attribute called "priority" is added to the *PFT* and *ERB* tables. With this change, when a prioritized packet is received by *MPLS Classifier*, the classifier can lookup the *ERB* and *PFT* to determine whether an ER-LSP whose bound-priorities match the packet's priority, and switch the packet to an appropriate LSP. Also, additional functions are defined for binding and unbinding different priorities to ER-LSP.

In order to support the feature of rerouting prioritized packets upon failures, similar functions are defined for binding and unbinding priorities to protection ER-LSP.

Besides the changes made above, a new conditional statement is introduced into the operational flow of ns-2 MPLS node. When a non MPLS-labeled packet is received by a MPLS node, the node will not only check whether there is an explicit-routed LSP (ER-LSP) pre-established for routing the packet, but also determines whether there are any priorities bound to the ER-LSP. If 1) no priority is bound to the ER-LSP or 2) one of the ER-LSP's bound priorities matches the packet's priority, the packet can then switch onto the LSP. Otherwise, *Service Classifier* will determine a non explicit-routed LSP to deliver the packet. If *Service Classifier* cannot find a LSP to deliver the packet, it will forward the packet to *Address Classifier* and *Address Classifier* will decide whether the packet should be dropped or routed via plain IP routing. Figure 9 shows the operational behaviour of the modified MPLS node.

To summarize, after making the modifications described above, the ns-2 simulator can now generate IP traffic of different priorities. In addition, different priority levels can be bound to an ER-LSP (to become an PBER-LSP).

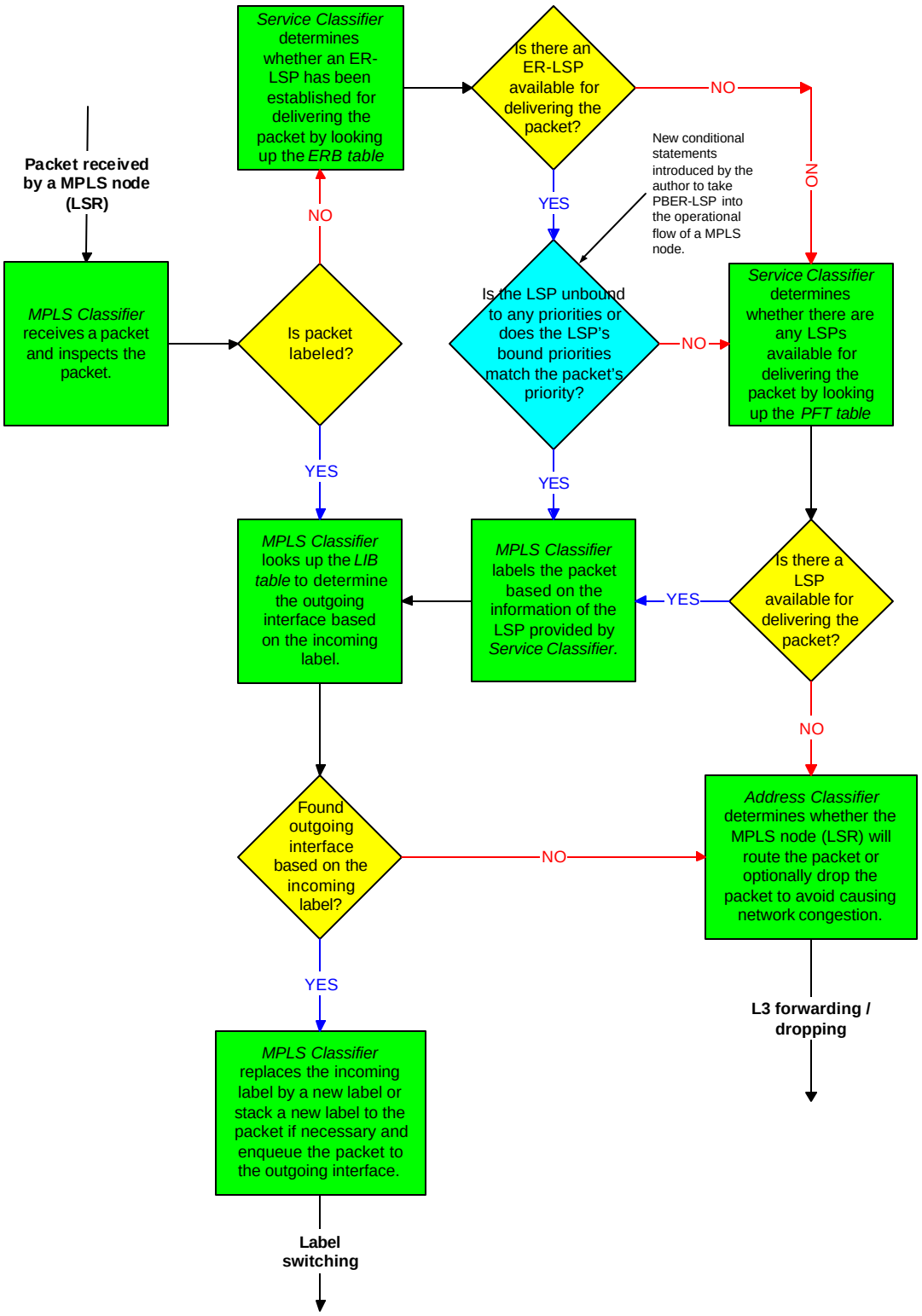


Figure 9: Operational flow-chart of ns-2 MPLS node with PBER-LSP capability incorporated.

When a prioritized packet is received by a LSR, the LSR determines whether a pre-establish ER-LSP or PBER-LSP can deliver the packet to its destination. If there is one, the LSR then checks whether the LSP is bound to any priorities levels and label-switch the packet to the LSP if 1) the packet's priority matches one of the LSP's bound-priorities, or 2) no priority is bound to the LSP. If the LSP is bound to certain priority levels but none of them matches the packet's priority, the LSR will determine another LSP to deliver the packet or will execute layer 3 forwarding if there is not another LSP suitable for delivering the packet.

4 Demonstration of PBER-LSP and Simulation Results

Two network topologies were used to demonstrate the PBER-LSP concept. The first topology is used to demonstrate that, by forwarding prioritized packets over PBER-LSP, ISPs can improve their network utilization, and offer end-to-end QoS and more CoS. The second topology is used to demonstrate how PBER-LSP improves network robustness and minimizes impacts to higher priority packets upon link failure.

4.1 Forwarding Prioritized Packets onto an PBER-LSP

In order to demonstrate the advantage of using PBER-LSP, the modified version of ns-2 was used to simulate the following network topology and please refer to "Appendix B – ns-2 Scripts Used for Demonstrating PBER-LSP Concept" for more details on the script used to generate the topology via the modified ns-2. It should be noted that Figure 10 was captured from Network Animator (NAM), a tool that visualizes the simulated network and its behaviour by parsing the output trace file generated by ns-2 after each simulation run.

There are five routers (numbered from 2 to 6, inclusively) in this simulated network. Traffic destined to a receiver (labeled as 7) is sent to router #2 by a source (labeled as 0) at a rate of 0.8Mbps. Half of the traffic is assumed to be real-time Voice-over-IP packets and the priority level of those packets is set to 15. The other half of the packets is assumed to be carrying non real-time traffic whose priority level is set to 0. In this simulated network, there are 0.5 Mbps and 1.0 Mbps reservable-bandwidth available along links 2-5-6 and 2-3-4-6, respectively.

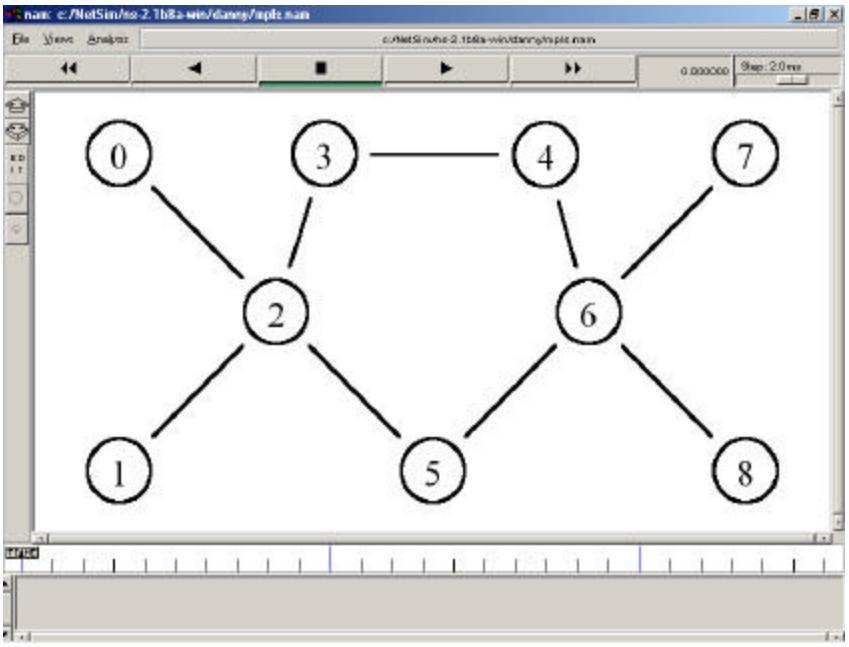


Figure 10: Topology used for simulation and performance comparison between Plain IP Routing, MPLS' ER-LSP and PBER-LSP.

Three simulation runs were performed using this configuration, and each simulation run uses a different routing method. The first simulation run uses plain IP routing (OSPF) to route the traffic. The second simulation run uses an ER-LSP to label-switch the traffic. The third simulation run uses the newly introduced PBER-LSP method. Packet end-to-end delay and drop ratio are collected after each simulation run, and their results are compared at the end of this section.

4.1.1 Simulation of Plain IP Routing

Since plain IP routing is used to determine each packet's route, the routing protocol would select the shortest path between the source and destination, which is link 2-5-6. Figure 11 is captured from NAM during

simulation. As illustrated in Figure 11, since the data arrival rate to router #2 is 0.8 Mbps but link 2-5-6 has only 0.5 Mbps available bandwidth, the link does not have sufficient bandwidth to handle the traffic, and hence a high percentage of packets were dropped by router #2 and the end-to-end delay experienced by each packet varies.

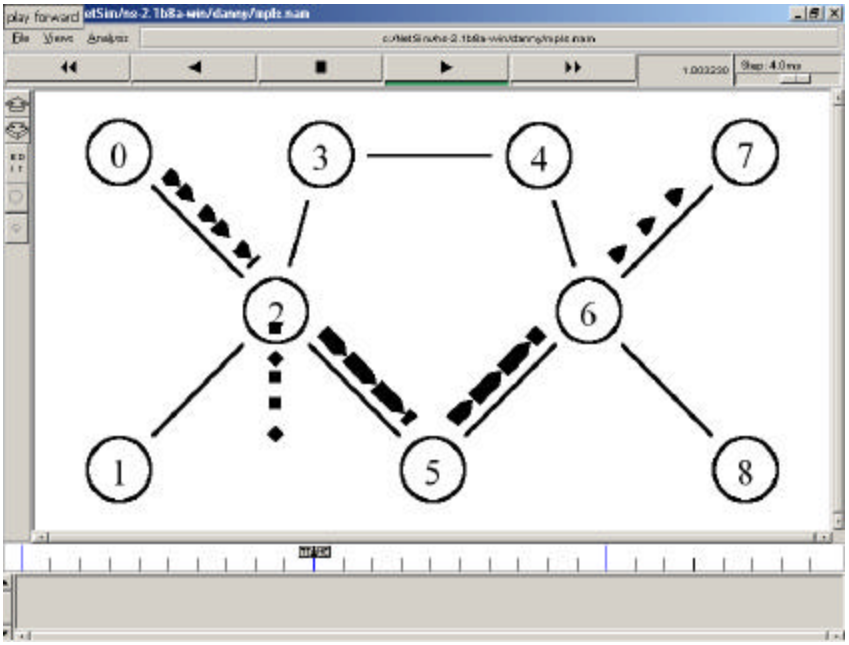


Figure 11: The shortest path (link 2-5-6) is selected by Plain IP Routing to deliver traffic from Source #0 to Destination #7, and packets are being dropped by Router #2 as there are not sufficient bandwidth at link 2-5.

Table 3 summarizes the end-to-end delay and the drop ratio experienced by the two groups of prioritized packets. Since Plain IP Routing treats all packets equally, the packet end-to-end delay experienced by those higher priority packets is similar to those lower priority packets. In addition, their standard deviations of the packet

end-to-end delay are large, which imply that delay experienced by each packet varies a lot.

Packet priority	Number of packets sent	Number of packets received	Drop ratio	Average end-to-end delay (sec)	Standard deviation of end-to-end delay
0	449	258	43%	0.187	0.0405
15	449	352	22%	0.190	0.036137565

Table 3: Plain IP Routing: Packet drop ratio and end-to-end delay.

4.1.2 Simulation of Label Switching via ER-LSP

According to the description of the network topology outlined earlier, link 2-3-4-6 has 1.0 Mbps bandwidth available and it is the only path that has sufficient bandwidth to handle the traffic. In this simulation run, an ER-LSP is pre-established along link 2-3-4-6 and prioritized packets are label-switched to the LSP. Figure 12 was captured when running NAM. It demonstrates that packets are label-switched to the ER-LSP instead of the path (link 2-5-6) chosen by plain IP routing

As sufficient bandwidth were reserved at each LSR along the ER-LSP, the traffic flow did not experience any packet drops. Since all packets follow the same path to reach their destinations, the end-to-end delays experienced by packets in both traffic classes are relatively consistent. Table 4 shows the packet end-to-end delay and the drop ratio experienced by the two groups of prioritized packets.

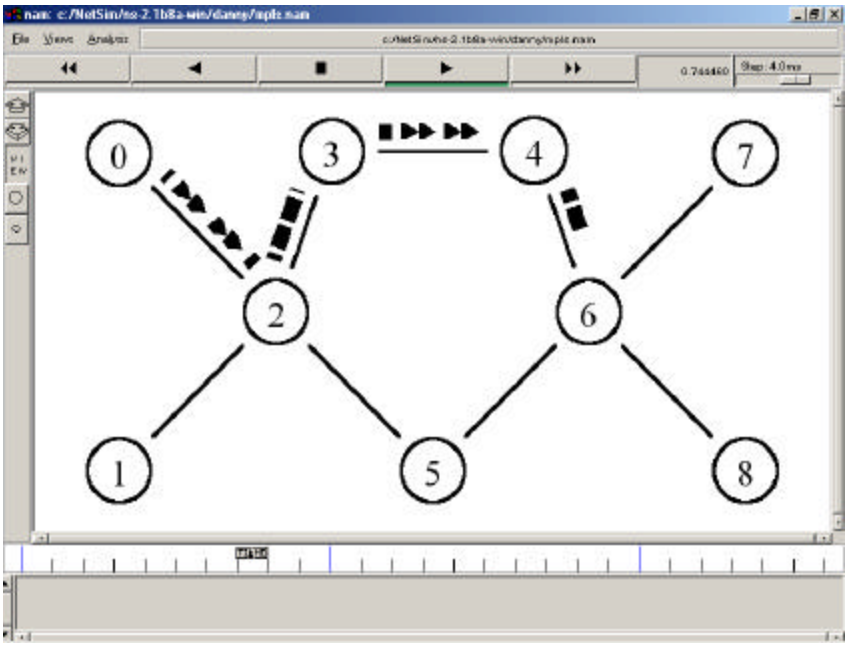


Figure 12: ER-LSP along link 2-3-4 is pre-established to deliver traffic from Source #0 to Destination #7.

Packet priority	Number of packets sent	Number of packets received	Drop ratio	Average end-to-end delay (sec)	Standard deviation of end-to-end Delay
0	449	449	0%	0. 0596	4.95057E-06
15	449	449	0%	0. 0580	1.22702E-06

Table 4: ER-LSP: Packet drop ratio and end-to-end delay.

4.1.3 Simulation of Label Switching via PBER-LSP

A well traffic-engineered network should be able to minimize the end-to-end delay experienced by those packets of higher priority, and

this is exactly what can be accomplished with PBER-LSP. In order to minimize end-to-end delay to those packets carrying audio data, two PBER-LSPs were pre-established in this simulation run. The first LSP is established along link 2-3-4-6 with priority bound to level 0. A second LSP is established along link 2-5-6 with priority bound to level 15. Packets that can tolerate higher delay (packets whose priorities are equal to 0) will follow the first LSP (link 2-3-4-6), and those packets carrying real-time packet will follow the second LSP (link 2-5-6). Figure 13 was captured when running NAM and it illustrates that packets of different priorities are label-switched to an appropriate PBER-LSP.

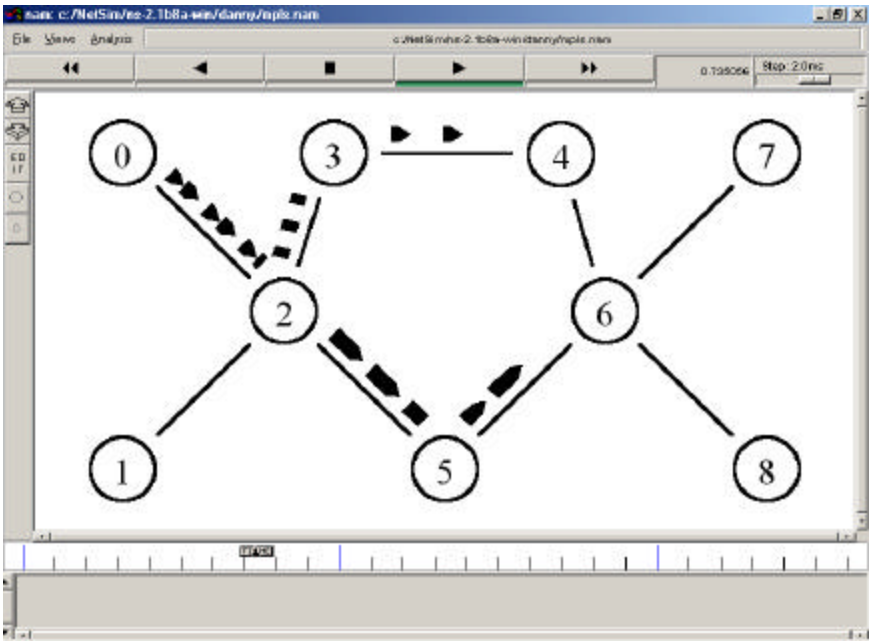


Figure 13: Two PBER-LSPs, one along link 2-3-4-6 and the other along link 2-5-6, are pre-established to deliver packets whose priorities are assigned to 0 and 15, respectively.

Since there are fewer hops along the second LSP (link 2-5-6), the average end-to-end delay experienced by those higher priority packets is less than those lower priority packets. Table 5 summarizes the packet end-to-end delay, the standard deviation of the end-to-end delay, and the drop ratio experienced by the two groups of prioritized packets.

Packet priority	Number of packets sent	Number of packets received	Drop ratio	Average end-to-end delay (sec)	Standard deviation of end-to-end Delay
0	449	449	0%	0. 0596	6.21768E-09
15	449	449	0%	0. 0496	4.88E-09

Table 5: PBER-LSP: Packet drop ratio and end-to-end delay.

4.1.4 Comparison of Simulation Results

Figure 14 and Figure 15 consolidate the results from the three simulation runs and they demonstrate that PBER-LSP can achieve better traffic engineering to prioritized packets over plain IP routing and ER-LSP. With PBER-LSP, the average end-to-end delay experienced by those higher priority packets is 10 ms smaller than the ER-LSP's. In addition, PBER-LSP's standard deviation of the end-to-end delay is smaller than ER-LSP's. Although it might not seem significant, the maximum tolerable round-trip delay of a toll-quality real-time communication is around 200 ms, and with PBER-LSP, a 10 ms improvement can make a real-time conversation smoother. [10]

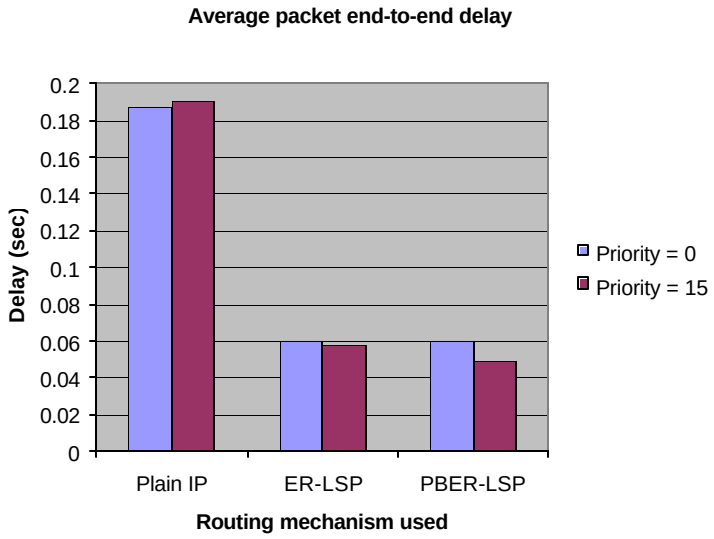


Figure 14: Comparison of the average packet end-to-end delay experienced by packets in plain IP routing, ER-LSP and PBER-LSP.

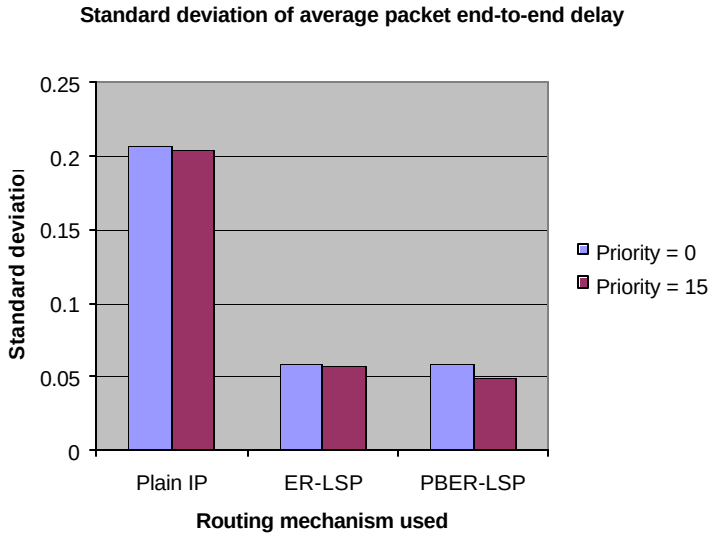


Figure 15: Comparison of the standard deviation of the average end-to-end delay experienced by packets in Plain IP Routing, ER-LSP and PBER-LSP.

4.2 Rerouting Prioritized Packets upon Failures

In order to demonstrate how PBER-LSP can minimize impacts to higher priority packets during node or link failure, the topology shown in Figure 16 was simulated using the modified version of ns-2. Please refer to "Appendix B – ns-2 Scripts Used for Demonstrating PBER-LSP Concept" for more details on the script used to generate the topology via the modified ns-2.

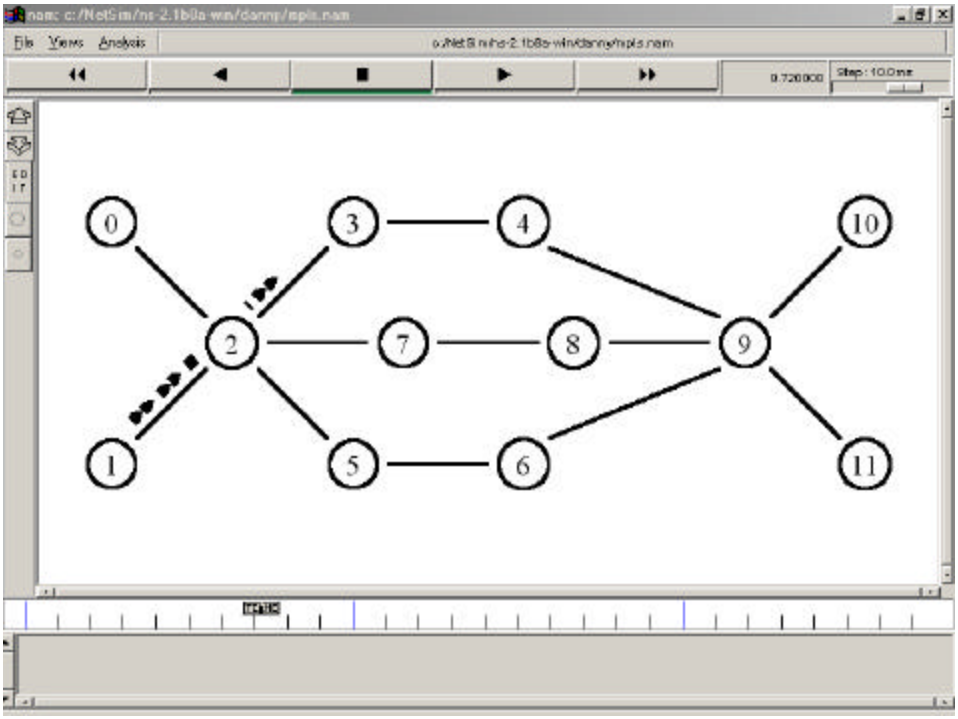


Figure 16: Topology used for demonstrating PBER-LSP's superior traffic-rerouting capability.

In this topology, traffic destined to a receiver (labeled as 10) is sent to router #2 by a source (labeled as 1) at a rate of 0.8 Mbps starting at t = 0.7 sec. The traffic consists of two classes of packet:

- The first class of packet is assumed to be carrying real-time audio data and composes half of the traffic flow. Packets within this class have their priority levels set to 10.
- The second class of packet is assumed to be carrying real-time video data and composes the other half of the traffic flow. These lower priority packets have their priority levels set to 8.

In this simulated network, there are 1.0 Mbps, 0.2 Mbps and 0.5 Mbps reservable-bandwidth available along links 2-3-4-9, 2-7-8-9 and 2-5-6-9, respectively. An ER-LSP is pre-established and sufficient bandwidth (0.8 Mbps) is reserved along the link 2-3-4-9 for label-switching these two classes of packets. In order to protect the traffic in case of node or link failures along the ER-LSP, a protection PBER-LSP is pre-established along the link 2-5-6-9. Since network resources (especially bandwidth) are scarce, pre-establishing a protection LSP requires bandwidth reservation in all LSRs along the path, and those reserved bandwidth are wasted when the protection LSP is idle. In order to maximize network utilization, the PBER-LSP is pre-established for protecting the audio traffic only, because human ears are more sensitive to audio signal loss. In addition, a real-time telephone conversation can still carry on even without video image. The protection PBER-LSP is bound to priority level 10 and only sufficient bandwidth (0.4 Mbps) is reserved along the LSP to handle the audio data. Therefore, as illustrated in Figure 17, when the link 2-3 goes down at t = 1.0 second, those packets carrying audio data are switched to the protection PBER-LSP, whereas those packets carrying video data are dropped to avoid causing congestions to other parts of the network. Figure 17 was captured from NAM at the instance when link 2-3 went down.

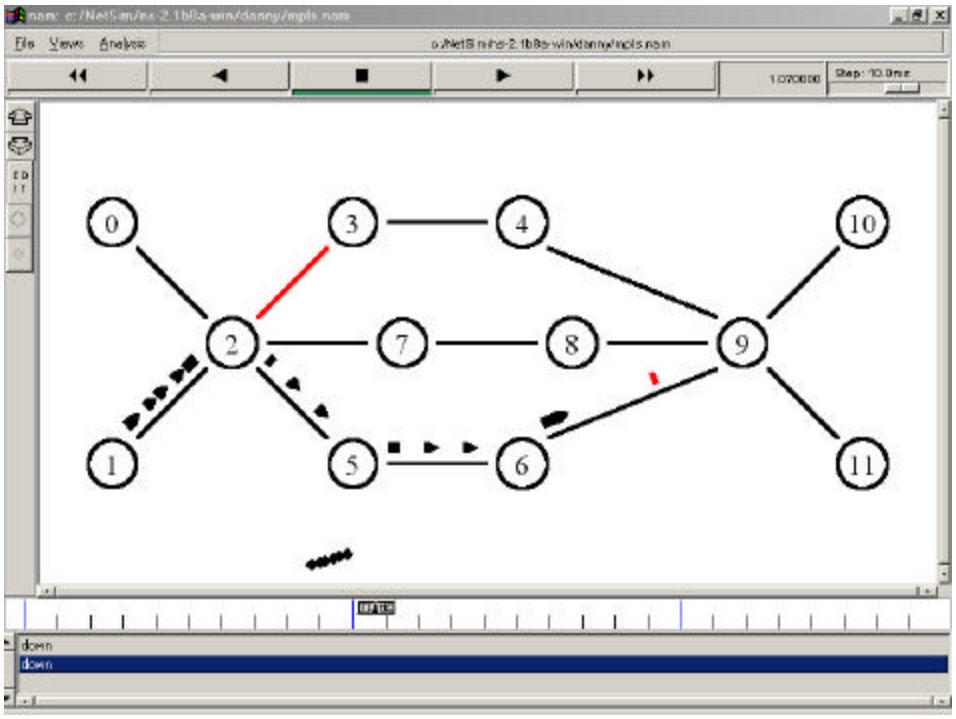


Figure 17: Higher priority packets are label-switched to the protection PBER-LSP when link 2-3 fails (at t = 1.0ms).

When the link between router #2 and #3 is restored at 1.5 second, both classes of packet follow the original (working) ER-LSP and those packets carrying video data are no longer dropped. Figure 18 was captured from NAM at the instance when the link 2-3 is restored and it shows that all packets now follow the working LSP to reach their destinations. The simulation demonstrates that, by using PBER-LSP, ISPs can have higher control over the type(s) of traffic they want to protect upon failures, and hence they can offer more classes of service to their customers. Moreover, it can be shown that, by selectively protecting a subset of packets based on their priorities, more network resource can allocated to many working LSPs, and therefore the amount

of idle bandwidth reserved for protection LSPs is minimized. As a result, ISPs can have more bandwidth to offer their customers and achieve more profits.

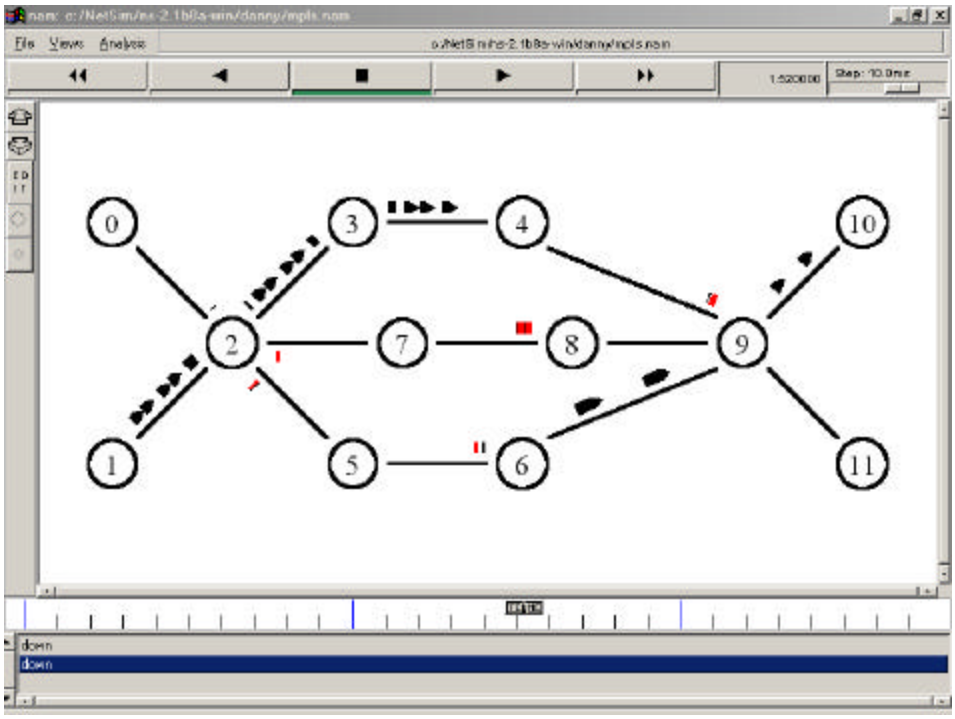


Figure 18: Both low and higher priority packets are label-switched to the working LSP when link 2-3 is restored (at t = 1.5ms).

4.3 Advantages and Disadvantages of PBER-LSP

As it has been demonstrated via ns-2 simulations, there are many advantages using PBER-LSP. With PBER-LSP, ISPs can:

1. Offer appropriate QoS treatments to packets based on their priorities;
2. Offer more services classes to their customers;

3. Maximize their network utilizations by forcing packets of different priorities to follow different LSPs;
4. Have higher control over the types of traffic they want to protect upon link and/or node failures;
5. Minimize the amount of idle bandwidth reserved to protection paths;
6. Have complete knowledge about how prioritized packets are labeled-switched across the network.
7. Achieve more profits as ISPs can have higher control over their networks.

Although PBER-LSP can achieve better traffic engineering to prioritized packets, there are some disadvantages associated with it:

1. PBER-LSP can only be used to label-switch prioritized packets received from a trust boundary. Obviously, ISPs do not want an un-trusted entity to send undetermined amounts of high priority packets to the network trying to gain higher network access;
2. Resource requirements and priority distribution of a traffic flow must be known ahead in order to take full advantage of PBER-LSP. Without the information, ISPs would not be able to pre-establish different PBER-LSPs to service packets of different priorities accordingly.
3. Since packets' priorities are bound to PBER-LSP, more labels that are distinct are required to classify packets that need different QoS treatments, and hence LSRs that support PBER-LSP will require more memory to store more labels and a larger label-lookup table.

Conclusions

An emerging technology called Multi-Protocol Label Switching (MPLS) has been gaining considerable attention among Internet Service Providers (ISPs), and one of the main reasons is that MPLS enables ISPs to better traffic engineer their networks. MPLS uses constraint-based routing to determine a path based on the traffic's constraints and label-switch packets onto the pre-determined path to reach their destinations. Since the path selected via MPLS is established according to traffic's constraints, sufficient network resources are reserved along the selected path to handle the traffic. As a result, ISPs can have higher control over their networks and provides better QoS to their customers.

Two QoS capabilities are supported by MPLS and they are: differentiated services and integrated services. None of these capabilities can provide QoS treatments to packets based on their Type of Service (ToS) field (or Traffic Class (TC) field) inside the IPv4 header (or IPv6 header). In order to address this limitation, I proposed a solution called "Priority-Bound and Explicit-Routed Label Switch Path" (PBER-LSP). PBER-LSP is a path explicitly established across the network via MPLS' constraint-based routing for delivering prioritized packet. This path is bound to different priority level(s) and only those packets whose values in their ToS (TC) field match the priority level(s) bound to the PBER-LSP can use the reserved resource along the path.

Such solution was implemented in Network simulator, version 2 (ns-2). Simulation results show that, with PBER-LSP, ISPs can offer different QoS treatment to packets based on their priorities, and hence can offer more service-classes to their customers. In addition, by forcing

packets of different priorities to follow different PBER-LSPs, network utilizations can be maximized. PBER-LSP can also allow ISPs to selectively protect higher priority packets during node or link failures, and hence the amount of idle bandwidth reserved to protection paths can be minimized. Although there are many advantages of using PBER-LSP to traffic engineer prioritized packets, application of PBER-LSP is limited to network where packets are received from a trusted source, otherwise paths reserved for those trusted traffic sources might be congested by those un-trusted sources trying to gain higher network access. Use of PBER-LSP also requires ISPs to have knowledge about the priority distribution of the traffic in addition of other traffic constraints. Moreover, label switch router (LSR) that supports PBER-LSP will require more memory to store a larger label-lookup table, as more labels are required to classify packets of different QoS treatments.

List of References

1. B. Davie and Y. Rekhter, *MPLS: Technology and Applications*, Morgan Kaufmann Publishers, Inc., May 2000.
2. Unisphere Networks, "MPLS-Based Traffic Engineering and Virtual Private Networks," URL: http://www.unisphere networks.com/downloads/products/mpls_based_traffic_whitepaper.pdf, (last access: February 21, 2002).
3. E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture," RFC 3031, January 2001.
4. S. Deering and R. Hinden, "Internet protocol, version 6 (IPv6) specification," RFC 2460, December 1998, RFC 1883, December 1995.
5. Avici Systems Inc., "Traffic Engineering with Multiprotocol Label Switching," March 16, 2000, URL: http://www.avici.com/ technology/whitepapers/ mpls_wp.pdf, (last access: February 21, 2002).
6. CoSine Communications, "MPLS and the Virtual Router Architecture: Profiting from Traffic Engineering and Network-based Services," July 2001, URL: http://www.cosinecom.com/ library/downloads/mpls_wp.pdf, (last access: February 21, 2002).
7. Cisco Systems Inc., "MPLS Traffic Engineering," December 13, 1999, URL: http://www.cisco.com/univercd/cc/td/doc/ product/software/ios120/120newft/120t/120t7/te120_7t.pdf, (last access: February 21, 2002).

8. K. Nichols, S. Blake, F. Baker, and D. Black, “Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers,” RFC 2474, December 1998.
9. G. Ahn and W. Chun, “Architecture of MPLS Network Simulator (MNS) for the setup of CR-LSP,” October 11, 2001, URL: http://flower.ce.cnu.ac.kr/~fog1/mns/mns2.0/doc/MNS_v2.0_arch.pdf, (last access: February 21, 2002).
10. M. Karam and F. Tobagi, “Analysis of the Delay and Jitter of Voice Traffic Over the Internet”, 2001, URL: http://www.ieee-infocom.org/2001/paper/307.pdf, *Proceedings of IEEE Infocom 2001*, (last access: April 02, 2002).

Appendix A – Modifications to ns-2 Source Code

The following lists all the files changed to Network Simulator version 2.1b8a-win in order to implement the PBER-LSP concept:

/ns-2.1b8a-win/agent.h

/ns-2.1b8a-win/agent.c

/ns-2.1b8a-win/mpls/classifier-addr-mpsl.h

/ns-2.1b8a-win/mpls/classifier-addr-mpsl.cc

/ns-2.1b8a-win/ip.h

/ns-2.1b8a-win/trace.cc

/ns-2.1b8a-win/mpls/ns-mpls-node.tcl

Modifications made to the above are implemented such that all interfaces to the ns-2 scripting are backwardly compatible. The following are the output files after running Unix's "diff" command to the modified version and the original version of each file listed above:

/ns-2.1b8a-win/agent.h

```
95d94
<      inline int& prio() { return (prio_); }
137,143d135
<      void setFixedPrio(int prio);
<      void setUniformPrio();
<      void clearPrio();
<
<      int fixedPrio_;
<      int uniformPrio_;
<
```

/ns-2.1b8a-win/agent.cc

```
49d48
< #include "rng.h"
63d61
<
77c75
<      oldValueList_(NULL), app_(0), fixedPrio_(-1), uniformPrio_(-1)
---
>      oldValueList_(NULL), app_(0)
```

```
147,152d144
<             } else if (strcmp(argv[1], "set-uniform-prio") == 0) {
<                    setUniformPrio();
<                    return (TCL_OK);
<             } else if (strcmp(argv[1], "clear-prio") == 0) {
<                    clearPrio();
<                    return (TCL_OK);
153a146
>
182,184d174
<             } else if (strcmp(argv[1], "set-fixed-prio") == 0) {
<                    setFixedPrio(atoi(argv[2]));
<                    return (TCL_OK);
480,494c470
<      int localPrio = prio_;
<      if (uniformPrio_ != -1) {
<             localPrio = RNG::defaultrng()->uniform(IP_MAX_NUM_OF_PRIO);
<             if (3 <= localPrio) {
<                    localPrio++;
<                    if (5 <= localPrio) {
<                           localPrio++;
<                    }
<             }
<      } else {
<             if (fixedPrio_ != -1) {
<                    localPrio = fixedPrio_;
<             }
<      }
<      iph->prio() = localPrio;
---
>      iph->prio() = prio_;
552,572d527
< }
<
< void
< Agent::setFixedPrio(int fixedPrio)
< {
<      fixedPrio_ = fixedPrio;
<      uniformPrio_ = -1;
< }
<
< void
< Agent::setUniformPrio()
< {
<      fixedPrio_ = -1;
<      uniformPrio_ = 0;
< }
<
< void
< Agent::clearPrio()
< {
<      fixedPrio_ = -1;
<      uniformPrio_ = -1;
```

/ns-2.1b8a-win/mpls/classifier-addr-mpsl.h

```
90,91d89
<      int  prio_;
<      int  prioBound_;
94,95d91
<      int  aPATHprio_;
```

```
<      int  aPATHprioBound_;
106d101
<      int prio_;
108d102
<      int prioBound_;
134d127
<      int prio_;
161,162c154
<      void GetIPInfo(Packet* p, ns_addr_t& dstaddr, int& phb,
<                     int& srcnode, int& prio);
---
>      void GetIPInfo(Packet* p, ns_addr_t& dstaddr, int& phb, int& srcnode);
173c165
<      void PFTinsert(int FEC, int PHB, int prio, int LIBptr);
---
>      void PFTinsert(int FEC, int PHB, int LIBptr);
177,180c169,170
<      void PFTbindPrio(int entrynb, int prio);
<      void PFTunbindPrio(int entrynb, int prio);
<      int PFTlocate(int FEC, int PHB, int prio, int &LIBptr);
<      int PFTlookup(int FEC, int PHB, int prio, int &oIface,
---
>      int PFTlocate(int FEC, int PHB, int &LIBptr);
>      int PFTlookup(int FEC, int PHB, int &oIface,
183c173
<      void ERBinsert(int LSPid, int FEC, int prio, int LIBptr);
---
>      void ERBinsert(int LSPid, int FEC, int LIBptr);
186,188c176
<      void ERBbindPrio(int entrynb, int prio);
<      void ERBunbindPrio(int entrynb, int prio);
<      int ERBlocate(int LSPid, int FEC, int prio, int &LIBptr);
---
>      int ERBlocate(int LSPid, int FEC, int &LIBptr);
199,203c187,190
<      int ErLspBinding(int FEC,int PHB, int erFEC, int LSPid, int prio);
<      int ErLspStacking(int erFEC0, int erLSPid0, int prio0,
<                        int erFEC, int erLSPid, int prio);
<      int FlowAggregation(int fineFEC, int finePHB, int finePrio,
<                          int coarseFEC, int coarsePHB, int coarsePrio);
---
>      int ErLspBinding(int FEC,int PHB, int erFEC, int LSPid);
>      int ErLspStacking(int erFEC0, int erLSPid0, int erFEC, int erLSPid);
>      int FlowAggregation(int fineFEC, int finePHB, int coarseFEC,
>                          int coarsePHB);
205,206c192,193
<      int aPathBinding(int FEC, int PHB, int erFEC, int LSPid, int prio);
<      int aPathLookup(int FEC, int PHB, int prio,
---
>      int aPathBinding(int FEC, int PHB, int erFEC,int LSPid);
>      int aPathLookup(int FEC, int PHB,
```

/ns-2.1b8a-win/mpls/classifier-addr-mpsl.cc

```
192,193c192
<      if (aPathLookup(PI_.dst_.addr_, PI_.phb_, PI_.prio_,
<                      oIface, oLabel, LIBptr) == 0)
---
>      if (aPathLookup(PI_.dst_.addr_, PI_.phb_, oIface, oLabel, LIBptr) == 0)
301c300
<
```

```
---
>
303,304c302
<      if (PFTlookup(PI_.dst_.addr_, PI_.phb_, PI_.prio_,
<                    oIface, oLabel, LIBptr) == 0)
---
>      if (PFTlookup(PI_.dst_.addr_, PI_.phb_, oIface, oLabel, LIBptr) == 0)
340c338
<      GetIPInfo(p, PI_.dst_, PI_.phb_, PI_.srcnode_, PI_.prio_);
---
>      GetIPInfo(p, PI_.dst_, PI_.phb_, PI_.srcnode_);
374c372
<                                       int &phb, int &srcnode, int& prio)
---
>                                  int &phb, int &srcnode)
379d376
<      prio = iphdr->prio_;
476c473
<      } else if (argc == 3) {
---
>      } else if (argc == 3) {
487,490c484,485
<             }
<      } else if (argc == 4) {
<             if (strcmp(argv[1], "get-fec-for-lspid") == 0) {
<                    // <classifier get-fec-for-lspid LSPid prio
---
>             } else if (strcmp(argv[1], "get-fec-for-lspid") == 0) {
>                    // <classifier get-fec-for-lspid LSPid
492d486
<                    int prio   = atoi(argv[3]);
495c489
<                    ERBnb = ERBlocate(LSPid, -1, prio, LIBptr);
---
>                    ERBnb = ERBlocate(LSPid, -1, LIBptr);
501,528c495,496
<             } else if (strcmp(argv[1], "install") == 0) {
<                    int slot = atoi(argv[2]);
<                    //if ((slot >= 0) && (slot < nslot_) &&
<                    //  (slot_[slot] != NULL)) {
<                    //     if (strcmp(slot_[slot]->name(),argv[3]) != 0)
<                    //            tcl.evalf("%s routing-update %s %.15g",
<                    //                   name(), argv[2],
<                    //                   Scheduler::instance().clock());
<                    //     else
<                    //           tcl.evalf("%s routing-nochange %s %.15g",
<                    //                  name(), argv[2],
<                    //                  Scheduler::instance().clock());
<                    //} else
<                    //     tcl.evalf("%s routing-new %s %.15g",
<                    //              name(), argv[2],
<                    //              Scheduler::instance().clock());
<                    // Then the control is passed on the the base
<                    // address classifier!
<                    //return AddressClassifier::command(argc, argv);
<                    //not a good idea since it would start an infinite loop
incase MPLSAddressClassifier::install is defined;
<                    //so call Classifier::install explicitly.
<                    NsObject* target = (NsObject*)TclObject::lookup(argv[3]);
<                    //Classifier::install(slot, target);
<
<                    install(slot,target);
<                    return (TCL_OK);
```

```
<             }
<      }else if (argc == 5) {
---
>             }
>      } else if (argc == 4) {
530c498
<                     // <classifier> exist-fec <fec> <phb> [prio]
---
>                     // <classifier> exist-fec <fec> <phb>
532,533c500,501
<                     int PFTnb = PFTlocate(atoi(argv[2]), atoi(argv[3]),
<                                           atoi(argv[4]), LIBptr);
---
>                     int PFTnb = PFTlocate(atoi(argv[2]), atoi(argv[3]),
>                                           LIBptr);
545d512
<             int prio  = atoi(argv[4]);
548c515
<                     PFTnb = PFTlocate(fec,PHB, prio, LIBptr);
---
>                     PFTnb = PFTlocate(fec,PHB, LIBptr);
550c517
<                     ERBlocate(LSPid,fec, prio,LIBptr);
---
>                     ERBlocate(LSPid,fec, LIBptr);
583a551,576
>             } else if (strcmp(argv[1], "install") == 0) {
>                     int slot = atoi(argv[2]);
>                     //if ((slot >= 0) && (slot < nslot_) &&
>                     //  (slot_[slot] != NULL)) {
>                     //     if (strcmp(slot_[slot]->name(),argv[3]) != 0)
>                     //            tcl.evalf("%s routing-update %s %.15g",
>                     //                      name(), argv[2],
>                     //                     Scheduler::instance().clock());
>                     //     else
>                     //           tcl.evalf("%s routing-nochange %s %.15g",
>                     //                     name(), argv[2],
>                     //                     Scheduler::instance().clock());
>                     //} else
>                     //     tcl.evalf("%s routing-new %s %.15g",
>                     //               name(), argv[2],
>                     //               Scheduler::instance().clock());
>                     // Then the control is passed on the the base
>                     // address classifier!
>                     //return AddressClassifier::command(argc, argv);
>                     //not a good idea since it would start an infinite loop
incase MPLSAddressClassifier::install is defined;
>                     //so call Classifier::install explicitly.
>                     NsObject* target = (NsObject*)TclObject::lookup(argv[3]);
>                     //Classifier::install(slot, target);
>
>                     install(slot,target);
>                     return (TCL_OK);
585c578
<      } else if (argc == 6) {
---
>      } else if (argc == 5) {
587c580
<                     // <classifier> ErLspBinding <FEC> <PHB> <lspid> [prio]
---
>                     // <classifier> ErLspBinding <FEC> <PHB> <lspid>
591,607c584
<                     int prio   = atoi(argv[5]);
```

```
<                   if ( !ErLspBinding(addr, PHB, -1, LSPid, prio) )
<                         tcl.result("1");
<                   else
<                         tcl.result("-1");
<                   return (TCL_OK);
<             }
<      } else if (argc == 7) {
<             if (strcmp(argv[1], "aPathBinding") == 0) {
<                   // <classifier> aPathBinding <FEC> <PHB>
<                   // <erFEC> <LSPid>
<                   int FEC   = atoi(argv[2]);
<                   int PHB   = atoi(argv[3]);
<                   int erFEC = atoi(argv[4]);
<                   int LSPid = atoi(argv[5]);
<                   int prio  = atoi(argv[6]);
<                   if (aPathBinding(FEC, PHB, erFEC, LSPid, prio) == 0)
---
>                   if ( !ErLspBinding(addr, PHB,-1, LSPid) )
613c590,591
<      } else if (argc == 8) {
---
>      } else if (argc == 6) {
>             // <classifier> ErLspStacking fec0 erlspid0 fec erfecid
615d592
<                   // <classifier> ErLspStacking fec0 erlspid0 prio0 fec
erfecid prio
618,622c595,597
<                   int prio0    = atoi(argv[4]);
<                   int erfec    = atoi(argv[5]);
<                   int erlspid  = atoi(argv[6]);
<                   int prio     = atoi(argv[7]);
<                   if (ErLspStacking(erfec0,erlspid0,prio0,erfec,erlspid,prio)
== 0)
---
>                   int erfec    = atoi(argv[4]);
>                   int erlspid  = atoi(argv[5]);
>                   if (ErLspStacking(erfec0,erlspid0,erfec,erlspid) == 0)
632,637c607,610
<                   int finePrio   = atoi(argv[4]);
<                   int coarseaddr = atoi(argv[5]);
<                   int coarsePHB  = atoi(argv[6]);
<                   int coarsePrio = atoi(argv[7]);
<                   if (FlowAggregation(fineaddr, finePHB, finePrio, coarseaddr,
<                                     coarsePHB, coarsePrio) == 0)
---
>                   int coarseaddr = atoi(argv[4]);
>                   int coarsePHB  = atoi(argv[5]);
>                   if (FlowAggregation(fineaddr, finePHB, coarseaddr,
>                                     coarsePHB) == 0)
642,643c615,628
<             }
<      } else if (argc == 9) {
---
>             } else if (strcmp(argv[1], "aPathBinding") == 0) {
>                   // <classifier> aPathBinding <FEC> <PHB>
>                   // <erFEC> <LSPid>
>                   int FEC   = atoi(argv[2]);
>                   int PHB   = atoi(argv[3]);
>                   int erFEC = atoi(argv[4]);
>                   int LSPid = atoi(argv[5]);
>                   if (aPathBinding(FEC, PHB, erFEC, LSPid) == 0)
>                         tcl.result("1");
>                   else
```

```
>                         tcl.result("-1");
>                 return (TCL_OK);
>         }
>     } else if (argc == 8) {
646d630
<         int prio  = atoi(argv[8]);
650c634
<                 PFTnb = PFTlocate(addr,PHB,prio,LIBptr);
---
>                 PFTnb = PFTlocate(addr,PHB,LIBptr);
652c636
<                 ERBnb = ERBlocate(LSPid,addr,prio,LIBptr);
---
>                 ERBnb = ERBlocate(LSPid,addr,LIBptr);
694c678
<                                                 MPLS_DEFAULT_PHB, prio,
---
>                                                 MPLS_DEFAULT_PHB,
750c734
<                                 ERBinsert(LSPid,addr,prio,ptr);
---
>                                 ERBinsert(LSPid,addr,ptr);
780c764
< void MPLSAddressClassifier::PFTinsert(int FEC, int PHB, int prio, int LIBptr)
---
> void MPLSAddressClassifier::PFTinsert(int FEC, int PHB, int LIBptr)
786,794d769
<     PFT_.Entry_[PFT_.NB_].aPATHprio_  = 0;
<     PFT_.Entry_[PFT_.NB_].aPATHprioBound_  = 0;
<     if (prio >= 0) {
<             PFT_.Entry_[PFT_.NB_].prio_      = prio;
<             PFT_.Entry_[PFT_.NB_].prioBound_ = 1;
<     } else {
<             PFT_.Entry_[PFT_.NB_].prio_      = 0;
<             PFT_.Entry_[PFT_.NB_].prioBound_ = 0;
<     }
800,804c775,777
<     PFT_.Entry_[entrynb].FEC_       = -1;
<     PFT_.Entry_[entrynb].PHB_       = -1;
<     PFT_.Entry_[entrynb].prio_      = 0;
<     PFT_.Entry_[entrynb].prioBound_ = 0;
<     PFT_.Entry_[entrynb].LIBptr_    = -1;
---
>     PFT_.Entry_[entrynb].FEC_    = -1;
>     PFT_.Entry_[entrynb].PHB_    = -1;
>     PFT_.Entry_[entrynb].LIBptr_ = -1;
812,816c785,787
<             PFT_.Entry_[i].FEC_       = -1;
<             PFT_.Entry_[i].PHB_       = -1;
<             PFT_.Entry_[i].prio_      = 0;
<             PFT_.Entry_[i].prioBound_ = 0;
<             PFT_.Entry_[i].LIBptr_    = -1;
---
>             PFT_.Entry_[i].FEC_    = -1;
>             PFT_.Entry_[i].PHB_    = -1;
>             PFT_.Entry_[i].LIBptr_ = -1;
825,839c796
< void MPLSAddressClassifier::PFTbindPrio(int entrynb, int prio)
< {
<     //printf("PFTbindPrio: entrynb %d, prio %d\n", entrynb, prio);
<     PFT_.Entry_[entrynb].prio_      |= prio;
<     PFT_.Entry_[entrynb].prioBound_ = 1;
< }
```

```
<
< void MPLSAddressClassifier::PFTunbindPrio(int entrynb, int prio)
< {
<      //printf("PFTunbindPrio: entrynb %d, prio %d\n", entrynb, prio);
<      PFT_.Entry_[entrynb].prio_      = 0;
<      PFT_.Entry_[entrynb].prioBound_ = 0;
< }
<
< int MPLSAddressClassifier::PFTlocate(int FEC, int PHB, int prio, int &LIBptr)
---
> int MPLSAddressClassifier::PFTlocate(int FEC, int PHB, int &LIBptr)
845,855c802,804
<             if ((PFT_.Entry_[i].FEC_  == FEC) &&
<                 (PFT_.Entry_[i].PHB_  == PHB)) {
<                    if (PFT_.Entry_[i].prioBound_) {
<                                 if ((prio != -1) && (PFT_.Entry_[i].prio_ & (1
<< prio))) {
<                                        LIBptr = PFT_.Entry_[i].LIBptr_;
<                                        return i;
<                                 }
<                    } else {
<                                 LIBptr = PFT_.Entry_[i].LIBptr_;
<                                 return i;
<                          }
---
>             if ((PFT_.Entry_[i].FEC_ == FEC) && (PFT_.Entry_[i].PHB_ == PHB))
{
>                    LIBptr = PFT_.Entry_[i].LIBptr_;
>                    return i;
859,862c808,810
< /*
< int MPLSAddressClassifier::PFTlookup(int FEC, int PHB,
<                                      int prio, int &oIface,
<                                      int &oLabel, int &LIBptr)
---
>
> int MPLSAddressClassifier::PFTlookup(int FEC, int PHB, int &oIface,
>                                int &oLabel, int &LIBptr)
868,869c816
<             if ((PFT_.Entry_[i].FEC_ == FEC) && (PFT_.Entry_[i].PHB_ == PHB)
&&
<                 (PFT_.Entry_[i].prio_ == prio))
---
>             if ((PFT_.Entry_[i].FEC_ == FEC) && (PFT_.Entry_[i].PHB_ == PHB))
874,896d820
< */
<
< int MPLSAddressClassifier::PFTlookup(int FEC, int PHB,
<                                      int prio, int &oIface,
<                                      int &oLabel, int &LIBptr)
< {
<      oIface = oLabel = LIBptr = -1;
<      if (FEC < 0)
<             return -1;
<      for (int i = 0; i < PFT_.NB_; i++)
<             if ((PFT_.Entry_[i].FEC_ == FEC) && (PFT_.Entry_[i].PHB_ == PHB))
{
<                    if (PFT_.Entry_[i].prioBound_) {
<                           if ((prio != -1) && (PFT_.Entry_[i].prio_ & (1 <<
prio))) {
<                                  return LIBlookup(PFT_.Entry_[i].LIBptr_,
<                                                   oIface, oLabel, LIBptr);
<                           }
```

```
<                   } else {
<                          return LIBlookup(PFT_.Entry_[i].LIBptr_,
<                                           oIface, oLabel, LIBptr);
<                   }
<             }
<      return -1;
< }
919c843
< void MPLSAddressClassifier::ERBinsert(int LSPid, int FEC, int prio, int LIBptr)
---
> void MPLSAddressClassifier::ERBinsert(int LSPid, int FEC, int LIBptr)
924,930d847
<      if (prio >= 0) {
<             ERB_.Entry_[ERB_.NB_].prio_      = prio;
<             ERB_.Entry_[ERB_.NB_].prioBound_ = 1;
<      } else {
<             ERB_.Entry_[ERB_.NB_].prio_      = 0;
<             ERB_.Entry_[ERB_.NB_].prioBound_ = 0;
<      }
936,940c853,855
<      ERB_.Entry_[entrynb].FEC_       = -1;
<      ERB_.Entry_[entrynb].LSPid_     = -1;
<      ERB_.Entry_[entrynb].prio_      = 0;
<      ERB_.Entry_[entrynb].prioBound_ = 0;
<      ERB_.Entry_[entrynb].LIBptr_    = -1;
---
>      ERB_.Entry_[entrynb].FEC_    = -1;
>      ERB_.Entry_[entrynb].LSPid_  = -1;
>      ERB_.Entry_[entrynb].LIBptr_ = -1;
948,967c863
< void MPLSAddressClassifier::ERBbindPrio(int entrynb, int prio)
< {
<      //printf("ERBbindPrio: entrynb %d, prio %d\n", entrynb, prio);
<      if (prio >= 0) {
<             ERB_.Entry_[entrynb].prio_      |= prio;
<             ERB_.Entry_[entrynb].prioBound_ = 1;
<      } else {
<             ERB_.Entry_[entrynb].prio_      = 0;
<             ERB_.Entry_[entrynb].prioBound_ = 0;
<      }
< }
<
< void MPLSAddressClassifier::ERBunbindPrio(int entrynb, int prio)
< {
<      //printf("ERBunbindPrio: entrynb %d, prio %d\n", entrynb, prio);
<      ERB_.Entry_[entrynb].prio_ = 0;
<      ERB_.Entry_[entrynb].prioBound_ = 0;
< }
<
< int  MPLSAddressClassifier::ERBlocate(int LSPid, int FEC, int prio, int
&LIBptr)
---
> int  MPLSAddressClassifier::ERBlocate(int LSPid, int FEC, int &LIBptr)
971,980c867,869
<             if ((ERB_.Entry_[i].LSPid_ == LSPid)) {
<                    if (ERB_.Entry_[i].prioBound_) {
<                           if ((prio != -1) && (ERB_.Entry_[i].prio_ & (1 <<
prio))) {
<                                  LIBptr = ERB_.Entry_[i].LIBptr_;
<                                  return(i);
<                           }
<                    } else {
<                           LIBptr = ERB_.Entry_[i].LIBptr_;
```

```
<                         return(i);
<                 }
---
>             if (ERB_.Entry_[i].LSPid_ == LSPid) {
>                   LIBptr = ERB_.Entry_[i].LIBptr_;
>                   return(i);
1125,1126c1014,1015
< int MPLSAddressClassifier::ErLspStacking(int erFEC0, int erLSPid0, int prio0,
<                                          int erFEC, int erLSPid, int prio)
---
> int MPLSAddressClassifier::ErLspStacking(int erFEC0,int erLSPid0,
>                                    int erFEC, int erLSPid)
1130c1019
<      if ((ERBlocate(erLSPid0,erFEC0,prio0,erLIBptr0) < 0) || (erLIBptr0 < 0))
---
>      if ((ERBlocate(erLSPid0,erFEC0,erLIBptr0) < 0) || (erLIBptr0 < 0))
1132c1021
<      if ((ERBlocate(erLSPid,erFEC,prio,erLIBptr) < 0) || (erLIBptr < 0))
---
>      if ((ERBlocate(erLSPid,erFEC,erLIBptr) < 0) || (erLIBptr < 0))
1138,1139c1027
< int MPLSAddressClassifier::ErLspBinding(int FEC,int PHB, int erFEC,
<                                         int LSPid, int prio)
---
> int MPLSAddressClassifier::ErLspBinding(int FEC,int PHB, int erFEC, int LSPid)
1143,1146d1030
<      int ERBnb = -1;
<
<      //printf("ErLspBinding: FEC %d, PHB %d, erFEC %d, LSPid %d, prio %d\n",
<      //       FEC, PHB, erFEC, LSPid, prio);
1148,1149c1032
<      ERBnb = ERBlocate(LSPid, erFEC, -1, erLIBptr);
<      if ((ERBnb < 0) || (erLIBptr < 0))
---
>      if ((ERBlocate(LSPid, erFEC, erLIBptr) < 0) || (erLIBptr < 0))
1152,1154c1035
<      ERBbindPrio(ERBnb, prio);
<
<      int PFTnb = PFTlocate(FEC, PHB, -1, LIBptr);
---
>      int PFTnb = PFTlocate(FEC,PHB, LIBptr);
1156c1037
<             PFTinsert(FEC, PHB, prio, erLIBptr);
---
>             PFTinsert(FEC,PHB, erLIBptr);
1158c1039
<             if (LIBptr < 0) {
---
>             if (LIBptr < 0)
1160,1161c1041
<                   PFTbindPrio(PFTnb, prio);
<             } else {
---
>             else {
1174,1176c1054
<                                         int finePrio,
<                                         int coarseFEC, int coarsePHB,
<                                         int coarsePrio)
---
>                                   int coarseFEC, int coarsePHB)
1181c1059
<      if ((PFTlocate(coarseFEC, coarsePHB, coarsePrio, cLIBptr) < 0) ||
(cLIBptr < 0))
```

```
---
>      if ((PFTlocate(coarseFEC,coarsePHB, cLIBptr) < 0) || (cLIBptr < 0))
1184c1062
<      int PFTnb = PFTlocate(fineFEC, finePHB, finePrio, fLIBptr);
---
>      int PFTnb = PFTlocate(fineFEC,finePHB, fLIBptr);
1186c1064
<             PFTinsert(fineFEC, finePHB, finePrio, cLIBptr);
---
>             PFTinsert(fineFEC,finePHB, cLIBptr);
1202,1203c1080
< int MPLSAddressClassifier::aPathBinding(int FEC, int PHB, int erFEC,
<                                         int LSPid, int prio)
---
> int MPLSAddressClassifier::aPathBinding(int FEC, int PHB, int erFEC, int LSPid)
1205c1082
<      int entrynb, tmp, erLIBptr;
---
>      int entrynb,tmp,erLIBptr;
1207,1208c1084,1085
<      entrynb = PFTlocate(FEC,PHB,prio,tmp);
<      if ((entrynb < 0) || (ERBlocate(LSPid, erFEC, prio, erLIBptr) < 0))
---
>      entrynb = PFTlocate(FEC,PHB,tmp);
>      if ((entrynb < 0) || (ERBlocate(LSPid,erFEC,erLIBptr) < 0))
1211,1214d1087
<      if (prio >= 0) {
<             PFT_.Entry_[entrynb].aPATHprio_ |= (1 << prio);
<             PFT_.Entry_[entrynb].aPATHprioBound_ = 1;
<      }
1218,1219c1091,1092
< int MPLSAddressClassifier::aPathLookup(int FEC, int PHB, int prio,
<                                        int &oIface, int &oLabel, int &LIBptr)
---
> int MPLSAddressClassifier::aPathLookup(int FEC,int PHB, int &oIface,
>                                int &oLabel, int &LIBptr)
1227,1238c1100,1102
<                 (PFT_.Entry_[i].PHB_ == PHB)) {
<                    if (PFT_.Entry_[i].aPATHprioBound_) {
<                           if ((prio >= 0) &&
<                               (PFT_.Entry_[i].aPATHprio_ & (1 << prio))) {
<                              return LIBlookup(PFT_.Entry_[i].aPATHptr_,
<                                           oIface, oLabel, LIBptr);
<                           }
<                    } else {
<                           return LIBlookup(PFT_.Entry_[i].aPATHptr_,
<                                        oIface, oLabel, LIBptr);

<                    }
<             }
---
>                 (PFT_.Entry_[i].PHB_ == PHB))
>                    return LIBlookup(PFT_.Entry_[i].aPATHptr_,
>                                     oIface, oLabel, LIBptr);
```

/ns-2.1b8a-win/ip.h

```
55,56d54
< #define IP_MAX_NUM_OF_PRIO 14
< #define IP_NUM_OF_RESERVED_PRIO 2
```

/ns-2.1b8a-win/trace.cc

```
247c247
<             sprintf(pt_->buffer(), "%c "TIME_FORMAT" %d %d %s %d %s %d %s.%s
%s.%s %d %d %d",
---
>             sprintf(pt_->buffer(), "%c "TIME_FORMAT" %d %d %s %d %s %d %s.%s
%s.%s %d %d",
265,266c265
<                     th->uid(), /* was p->uid_ */
<                     iph->prio()); // Added by Danny Yip
---
>                     th->uid() /* was p->uid_ */);
269c268
<                     "%c "TIME_FORMAT" %d %d %s %d %s %d %s.%s %s.%s %d %d %d
0x%x %d %d %d",
---
>                     "%c "TIME_FORMAT" %d %d %s %d %s %d %s.%s %s.%s %d %d %d
0x%x %d %d",
291,292c290
<                     tcph->sa_length(),
<                     iph->prio()); // Added by Danny Yip
---
>                     tcph->sa_length());
```

/ns-2.1b8a-win/tcl/mpls/ns-mpls-node.tcl

```
84,90c84,85
< RtModule/MPLS instproc exist-fec {fec phb args} {
<      if {[llength $args] == 0} {
<             set prio -1
<      } else {
<             set prio [lindex $args 0]
<      }
<      return [[$self set classifier_] exist-fec $fec $phb $prio]
---
> RtModule/MPLS instproc exist-fec {fec phb} {
>         return [[$self set classifier_] exist-fec $fec $phb]
93,99c88,89
< RtModule/MPLS instproc get-incoming-iface {fec lspid args} {
<      if {[llength $args] == 0} {
<             set prio -1
<      } else {
<             set prio [lindex $args 0]
<      }
<      return [[$self set classifier_] GetInIface $fec $lspid $prio]
---
> RtModule/MPLS instproc get-incoming-iface {fec lspid} {
>         return [[$self set classifier_] GetInIface $fec $lspid]
102,108c92,93
< RtModule/MPLS instproc get-incoming-label {fec lspid args} {
<      if {[llength $args] == 0} {
<             set prio -1
<      } else {
<             set prio [lindex $args 0]
<      }
<      return [[$self set classifier_] GetInLabel $fec $lspid $prio]
---
```

```
> RtModule/MPLS instproc get-incoming-label {fec lspid} {
>         return [[$self set classifier_] GetInLabel $fec $lspid]
111,117c96,97
< RtModule/MPLS instproc get-outgoing-label {fec lspid args} {
<      if {[llength $args] == 0} {
<             set prio -1
<      } else {
<             set prio [lindex $args 0]
<      }
<      return [[$self set classifier_] GetOutLabel $fec $lspid $prio]
---
> RtModule/MPLS instproc get-outgoing-label {fec lspid} {
>         return [[$self set classifier_] GetOutLabel $fec $lspid]
120,126c100,101
< RtModule/MPLS instproc get-outgoing-iface {fec lspid args} {
<      if {[llength $args] == 0} {
<             set prio -1
<      } else {
<             set prio [lindex $args 0]
<      }
<      return [[$self set classifier_] GetOutIface $fec $lspid $prio]
---
> RtModule/MPLS instproc get-outgoing-iface {fec lspid} {
>         return [[$self set classifier_] GetOutIface $fec $lspid]
129,135c104,105
< RtModule/MPLS instproc get-fec-for-lspid {lspid args} {
<      if {[llength $args] == 0} {
<             set prio -1
<      } else {
<             set prio [lindex $args 0]
<      }
<      return [[$self set classifier_] get-fec-for-lspid $lspid $prio]
---
> RtModule/MPLS instproc get-fec-for-lspid {lspid} {
>         return [[$self set classifier_] get-fec-for-lspid $lspid]
138c108
< RtModule/MPLS instproc in-label-install {fec lspid iface label args} {
---
> RtModule/MPLS instproc in-label-install {fec lspid iface label} {
140,145c110
<      if {[llength $args] == 0} {
<             set prio -1
<      } else {
<             set prio [lindex $args 0]
<      }
<      $self label-install $fec $lspid $iface $label $dontcare $dontcare $prio
---
>      $self label-install $fec $lspid $iface $label $dontcare $dontcare
148c113
< RtModule/MPLS instproc out-label-install {fec lspid iface label args} {
---
> RtModule/MPLS instproc out-label-install {fec lspid iface label} {
150,155c115
<      if {[llength $args] == 0} {
<             set prio -1
<      } else {
<             set prio [lindex $args 0]
<      }
<      $self label-install $fec $lspid $dontcare $dontcare $iface $label $prio
---
>      $self label-install $fec $lspid $dontcare $dontcare $iface $label
158c118
< RtModule/MPLS instproc in-label-clear {fec lspid args} {
```

```
---
> RtModule/MPLS instproc in-label-clear {fec lspid} {
160,165c120
<      if {[llength $args] == 0} {
<             set prio -1
<      } else {
<             set prio [lindex $args 0]
<      }
<      $self label-clear $fec $lspid -1 -1 $dontcare $dontcare $prio
---
>      $self label-clear $fec $lspid -1 -1 $dontcare $dontcare
168c123
< RtModule/MPLS instproc out-label-clear {fec lspid args} {
---
> RtModule/MPLS instproc out-label-clear {fec lspid} {
170,175c125
<      if {[llength $args] == 0} {
<             set prio -1
<      } else {
<             set prio [lindex $args 0]
<      }
<      $self label-clear $fec $lspid $dontcare $dontcare -1 -1 $prio
---
>      $self label-clear $fec $lspid $dontcare $dontcare -1 -1
178,184c128,129
< RtModule/MPLS instproc label-install {fec lspid iif ilbl oif olbl args} {
<      if {[llength $args] == 0} {
<             set prio -1
<      } else {
<             set prio [lindex $args 0]
<      }
<      [$self set classifier_] LSPsetup $fec $lspid $iif $ilbl $oif $olbl $prio
---
> RtModule/MPLS instproc label-install {fec lspid iif ilbl oif olbl} {
>         [$self set classifier_] LSPsetup $fec $lspid $iif $ilbl $oif $olbl
187,193c132,133
< RtModule/MPLS instproc label-clear {fec lspid iif ilbl oif olbl args} {
<      if {[llength $args] == 0} {
<             set prio -1
<      } else {
<             set prio [lindex $args 0]
<      }
<      [$self set classifier_] LSPrelease $fec $lspid $iif $ilbl $oif $olbl
$prio
---
> RtModule/MPLS instproc label-clear {fec lspid iif ilbl oif olbl} {
>         [$self set classifier_] LSPrelease $fec $lspid $iif $ilbl $oif $olbl
196,208c136,137
< RtModule/MPLS instproc flow-erlsp-install {fec phb lspid args} {
<      if {[llength $args] == 0} {
<             set prio -1
<      } else {
<             set prioToBind [lindex $args 0]
<             set prioToBind [split $prioToBind "_"]
<             set prioLen [llength $prioToBind]
<             set prio 0
<             for {set i 0} {$i < $prioLen} {incr i 1} {
<                    set prio [expr $prio | (1 << [lindex $prioToBind $i])]
<             }
<      }
<      [$self set classifier_] ErLspBinding $fec $phb $lspid $prio
---
> RtModule/MPLS instproc flow-erlsp-install {fec phb lspid} {
```

```
>         [$self set classifier_] ErLspBinding $fec $phb $lspid
211,217c140,141
< RtModule/MPLS instproc erlsp-stacking {erlspid tunnelid args} {
<      if {[llength $args] == 0} {
<             set prio -1
<      } else {
<             set prio [lindex $args 0]
<      }
<      [$self set classifier_] ErLspStacking -1 $erlspid $prio -1 $tunnelid
$prio
---
> RtModule/MPLS instproc erlsp-stacking {erlspid tunnelid} {
>         [$self set classifier_] ErLspStacking -1 $erlspid -1 $tunnelid
220,228c144,146
< RtModule/MPLS instproc flow-aggregation {fineFec finePhb \
<                                          coarseFec coarsePhb args} {
<      if {[llength $args] == 0} {
<             set prio -1
<      } else {
<             set prio [lindex $args 0]
<      }
<      [$self set classifier_] FlowAggregation $fineFec $finePhb $prio \
<          $coarseFec $coarsePhb $prio
---
> RtModule/MPLS instproc flow-aggregation {fineFec finePhb coarseFec coarsePhb}
{
>         [$self set classifier_] FlowAggregation $fineFec $finePhb $coarseFec \
>                $coarsePhb
245,258c163,164
< RtModule/MPLS instproc reroute-binding {fec phb lspid args} {
<      if {[llength $args] == 0} {
<             set prio -1
<             [$self set classifier_] aPathBinding $fec $phb -1 $lspid $prio
<      } else {
<             set prioToBind [lindex $args 0]
<             set prioToBind [split $prioToBind "_"]
<             set prioLen [llength $prioToBind]
<             set prio 0
<             for {set i 0} {$i < $prioLen} {incr i 1} {
<                    set prio [lindex $prioToBind $i]
<                    [$self set classifier_] aPathBinding $fec $phb -1 $lspid
$prio
<             }
<      }
---
> RtModule/MPLS instproc reroute-binding {fec phb lspid} {
>         [$self set classifier_] aPathBinding $fec $phb -1 $lspid
398c304
<             pathvec er lspid rc args} {
---
>             pathvec er lspid rc} {
426,431c332
<                    if { [llength $args] == 0 } {
<                           set prio -1
<                    } else {
<                           set prio [lindex $args 0]
<                    }
<                    set lspidFEC [$self get-fec-for-lspid $erhop $prio]
---
>                    set lspidFEC [$self get-fec-for-lspid $erhop]
558,563c459
<      for {set prio 0} {$prio < 16} {incr prio 1} {
<             set outlabel  [$self get-outgoing-label $fec $lspid $prio]
```

```
<             if {$outlabel >= 0} {
<                    break;
<             }
<      }
---
>      set outlabel  [$self get-outgoing-label $fec $lspid]
567,568c463,464
<      set nexthop [$self get-outgoing-iface $fec $lspid $prio]
<      $self out-label-clear $fec $lspid $prio
---
>      set nexthop [$self get-outgoing-iface $fec $lspid]
>      $self out-label-clear $fec $lspid
```

Appendix B – ns-2 Scripts Used for Demonstrating PBER-LSP Concept

The following is the script used to simulate the topology described in Section 4.1 “Forwarding Prioritized Packets onto an PBER-LSP”:

```
#========================
# Create simulator object
#========================
set ns [new Simulator]

#================================================
# Open files to write trace-data for NAM and Xgraph
#================================================
set nf [open mpls.nam w]
$ns namtrace-all $nf
set f0 [open mpls.tr w]
$ns trace-all $f0

#============================================================================
# Finish procedure which closes the trace file and opens Xgraph and NAM
#============================================================================
proc finish {} {
       global ns nf f0
       $ns flush-trace
       close $nf
       close $f0
       exec ../../nam/nam mpls.nam &
       exit 0
}

#============================================
# Set dynamic distance-vector routing protocol
#============================================
$ns rtproto DV

#============================================================
# Define trigger strategy, Label Distribution Control Mode
# and Label Allocation and Distribution Scheme
#============================================================
Classifier/Addr/MPLS set control_driven_ 1

#
# Turn on all traces to stdout
#
#Agent/LDP set trace_ldp_ 1
#Classifier/Addr/MPLS set trace_mpls_ 1

#
# use 'List' scheduling of events
#
$ns use-scheduler List

#================================================================
# define nodes and MPLS LSRs (in case of a LSR, the [$ns node]
# command has to be preceded by node-config -MPLS ON
# and succeeded by node-config -MPLS OFF
```

```
#================================================================
set node0   [$ns node]
set node1   [$ns node]
$ns node-config -MPLS ON
set LSR2    [$ns node]
set LSR3    [$ns node]
set LSR4    [$ns node]
set LSR5    [$ns node]
set LSR6    [$ns node]
$ns node-config -MPLS OFF
set node7   [$ns node]
set node8   [$ns node]

#================================================================
# Define links, bandwidth 1 Mb, delay 10ms, queue managementDropTail
#================================================================
$ns duplex-link $node0 $LSR2  1Mb 10ms DropTail
$ns duplex-link $node1 $LSR2  1Mb 10ms DropTail
$ns duplex-link $LSR2  $LSR3  1Mb 10ms DropTail
$ns duplex-link $LSR3  $LSR4  1Mb 10ms DropTail
$ns duplex-link $LSR4  $LSR6  1Mb 10ms DropTail
$ns duplex-link $LSR2  $LSR5  0.5Mb 10ms DropTail
$ns duplex-link $LSR5  $LSR6  0.5Mb 10ms DropTail
$ns duplex-link $LSR6  $node7 1Mb 10ms DropTail
$ns duplex-link $LSR6  $node8 1Mb 10ms DropTail

#================================
# Control layout of the network
#================================
$ns duplex-link-op $node0 $LSR2   orient right-down
$ns duplex-link-op $node1 $LSR2   orient right-up
$ns duplex-link-op $LSR2  $LSR3   orient right-up
$ns duplex-link-op $LSR3  $LSR4   orient right
$ns duplex-link-op $LSR4  $LSR6   orient right-down
$ns duplex-link-op $LSR2  $LSR5   orient right-down
$ns duplex-link-op $LSR5  $LSR6   orient right-up
$ns duplex-link-op $LSR6  $node7  orient right-up
$ns duplex-link-op $LSR6  $node8  orient right-down

#================================================================
# The default value of a link cost (1) can be adjusted
# Notice that the procedure sets the cost along one direction only!
#================================================================
#$ns cost $LSR3 $LSR4 3
#$ns cost $LSR4 $LSR3 3

#=========================================================
# Install/configure LDP agents on all MPLS nodes,
# and set path restoration function that reroutes traffic
# around a link failure in a LSP to an alternative LSP.
# There are 2 options as follows:
# "new": create new alternative path if one doesn't exist
# "drop": do not create any new alternative path
#
# Adjust loop length to address all LSRs (MPLS nodes).
#=========================================================
for {set i 2} {$i < 7} {incr i} {
       set a LSR$i
       for {set j [expr $i+1]} {$j < 7} {incr j} {
              set b LSR$j
              eval $ns LDP-peer $$a $$b
       }
       set m [eval $$a get-module "MPLS"]
```

```
	$m enable-reroute "drop"
}

#==============================
# Set ldp-message color in NAM
#==============================
$ns ldp-request-color      blue
$ns ldp-mapping-color      red
$ns ldp-withdraw-color     magenta
$ns ldp-release-color      orange
$ns ldp-notification-color yellow

#
# Define procedure to create a CBR traffic flow and connect it to a UDP agent
#
proc attach-expoo-traffic { node sink size burst idle rate} {
	global ns

	set udp [new Agent/UDP]
	$ns attach-agent $node $udp

	set traffic [new Application/Traffic/Exponential]
	$traffic set packetSize_ $size
	$traffic set burst_time_ $burst
	$traffic set idle_time_ $idle
	$traffic set rate_ $rate
	$traffic attach-agent $udp

	$ns connect $udp $sink
	return $traffic
}

proc attach-expoo-traffic-with-fixed-prio { node sink size burst idle rate prio}
{
	global ns

	set udp [new Agent/UDP]
	$udp set-fixed-prio $prio
	$ns attach-agent $node $udp

	set traffic [new Application/Traffic/Exponential]
	$traffic set packetSize_ $size
	$traffic set burst_time_ $burst
	$traffic set idle_time_ $idle
	$traffic set rate_ $rate
	$traffic attach-agent $udp

	$ns connect $udp $sink
	return $traffic
}

proc attach-expoo-traffic-with-uniform-prio { node sink size burst idle rate} {
	global ns

	set udp [new Agent/UDP]
	$udp set-uniform-prio
	$ns attach-agent $node $udp

	set traffic [new Application/Traffic/Exponential]
	$traffic set packetSize_ $size
	$traffic set burst_time_ $burst
	$traffic set idle_time_ $idle
	$traffic set rate_ $rate
```

```
        $traffic attach-agent $udp

        $ns connect $udp $sink
        return $traffic
}

#
# Create a traffic sink and attach it to the node node8
#
set sink0 [new Agent/LossMonitor]
$ns attach-agent $node7  $sink0
set sink1 [new Agent/LossMonitor]
$ns attach-agent $node7  $sink1

#
# Create a traffic source
#
set src0 [attach-expoo-traffic-with-fixed-prio $node0  $sink0 200 0 0 400k 15]
set src1 [attach-expoo-traffic-with-fixed-prio $node0  $sink1 200 0 0 400k 8]

$ns at 0.10 "[$LSR6 get-module MPLS] ldp-trigger-by-withdraw 7 -1"
$ns at 0.30 "[$LSR2 get-module MPLS] make-explicit-route 6 2_3_4_6 1000 -1"
$ns at 0.30 "[$LSR2 get-module MPLS] make-explicit-route 6 2_5_6 1001 -1"

# comment the following line when simulating plain IP routing
# only uncomment the following line if simulating ER-LSP
#$ns at 0.50 "[$LSR2 get-module MPLS] flow-erlsp-install 7 -1 1000"

# comment the following two lines when simulating plain IP routing
# only uncomment the following lines if simulating PBER-LSP
$ns at 0.50 "[$LSR2 get-module MPLS] flow-erlsp-install 7 -1 1000 8"
$ns at 0.50 "[$LSR2 get-module MPLS] flow-erlsp-install 7 -1 1001 15"

$ns at 0.70 "$src0 start"
$ns at 0.70 "$src1 start"
$ns at 2.50 "$src0 stop"
$ns at 2.50 "$src1 stop"
$ns at 3.00 "finish"

#
# The last line finally starts the simulation
#
$ns run
```

The following is the script used to simulate the topology described in Section 4.2 “Rerouting Prioritized Packets upon Failures”:

```
#=========================
# Create simulator object
#=========================
set ns [new Simulator]

#================================================
# Open files to write trace-data for NAM and Xgraph
#================================================
set nf [open mpls.nam w]
$ns namtrace-all $nf
set f0 [open mpls.tr w]
$ns trace-all $f0

#=============================================================================
# Finish procedure which closes the trace file and opens Xgraph and NAM
#=============================================================================
proc finish {} {
       global ns nf f0
       $ns flush-trace
       close $nf
       close $f0
       exec ../../nam/nam mpls.nam &
       exit 0
}

#============================================
# Set dynamic distance-vector routing protocol
#============================================
$ns rtproto DV

#==========================================================
# Define trigger strategy, Label Distribution Control Mode
# and Label Allocation and Distribution Scheme
#==========================================================
Classifier/Addr/MPLS set control_driven_ 1

#
# Turn on all traces to stdout
#
#Agent/LDP set trace_ldp_ 1
#Classifier/Addr/MPLS set trace_mpls_ 1

#
# use 'List' scheduling of events
#
$ns use-scheduler List

#================================================================
# define nodes and MPLS LSRs (in case of a LSR, the [$ns node]
# command has to be preceded by node-config -MPLS ON
# and succeeded by node-config -MPLS OFF
#================================================================
set node0   [$ns node]
set node1   [$ns node]
$ns node-config -MPLS ON
set LSR2    [$ns node]
set LSR3    [$ns node]
set LSR4    [$ns node]
```

```
set LSR5    [$ns node]
set LSR6    [$ns node]
set LSR7    [$ns node]
set LSR8    [$ns node]
set LSR9    [$ns node]
$ns node-config -MPLS OFF
set node10   [$ns node]
set node11   [$ns node]

#===================================================================
# Define links, bandwidth 1 Mb, delay 10ms, queue managementDropTail
#===================================================================
$ns duplex-link $node0 $LSR2  1Mb 10ms DropTail
$ns duplex-link $node1 $LSR2  1Mb 10ms DropTail
$ns duplex-link $LSR2  $LSR3  1Mb 10ms DropTail
$ns duplex-link $LSR3  $LSR4  1Mb 10ms DropTail
$ns duplex-link $LSR4  $LSR9  1Mb 10ms DropTail
$ns duplex-link $LSR2  $LSR5  1Mb 10ms DropTail
$ns duplex-link $LSR5  $LSR6  1Mb 10ms DropTail
$ns duplex-link $LSR6  $LSR9  1Mb 10ms DropTail
$ns duplex-link $LSR2  $LSR7  1Mb 10ms DropTail
$ns duplex-link $LSR7  $LSR8  0.5Mb 10ms DropTail
$ns duplex-link $LSR8  $LSR9  1Mb 10ms DropTail
$ns duplex-link $LSR9  $node10 1Mb 10ms DropTail
$ns duplex-link $LSR9  $node11 1Mb 10ms DropTail

#================================
# Control layout of the network
#================================
$ns duplex-link-op $node0 $LSR2   orient right-down
$ns duplex-link-op $node1 $LSR2   orient right-up
$ns duplex-link-op $LSR2  $LSR3   orient right-up
$ns duplex-link-op $LSR3  $LSR4   orient right
$ns duplex-link-op $LSR4  $LSR9   orient right-down
$ns duplex-link-op $LSR2  $LSR5   orient right-down
$ns duplex-link-op $LSR5  $LSR6   orient right
$ns duplex-link-op $LSR6  $LSR9   orient right-up
$ns duplex-link-op $LSR2  $LSR7   orient right
$ns duplex-link-op $LSR7  $LSR8   orient right
$ns duplex-link-op $LSR8  $LSR9   orient right
$ns duplex-link-op $LSR9  $node10  orient right-up
$ns duplex-link-op $LSR9  $node11  orient right-down

#===================================================================
# The default value of a link cost (1) can be adjusted
# Notice that the procedure sets the cost along one direction only!
#===================================================================
$ns cost $LSR3 $LSR4 3
$ns cost $LSR4 $LSR3 3
$ns cost $LSR5 $LSR6 3
$ns cost $LSR6 $LSR5 3

#=========================================================
# Install/configure LDP agents on all MPLS nodes,
# and set path restoration function that reroutes traffic
# around a link failure in a LSP to an alternative LSP.
# There are 2 options as follows:
# "new": create new alternative path if one doesn't exist
# "drop": do not create any new alternative path
#
# Adjust loop length to address all LSRs (MPLS nodes).
#=========================================================
for {set i 2} {$i < 10} {incr i} {
```

```
        set a LSR$i
        for {set j [expr $i+1]} {$j < 10} {incr j} {
                set b LSR$j
                eval $ns LDP-peer $$a $$b
        }
        set m [eval $$a get-module "MPLS"]
        $m enable-reroute "drop"
}

#==============================
# Set ldp-message color in NAM
#==============================
$ns ldp-request-color      blue
$ns ldp-mapping-color      red
$ns ldp-withdraw-color     magenta
$ns ldp-release-color      orange
$ns ldp-notification-color yellow

#
# Define procedure to create a CBR traffic flow and connect it to a UDP agent
#
proc attach-expoo-traffic { node sink size burst idle rate} {
        global ns

        set udp [new Agent/UDP]
        $ns attach-agent $node $udp

        set traffic [new Application/Traffic/Exponential]
        $traffic set packetSize_ $size
        $traffic set burst_time_ $burst
        $traffic set idle_time_ $idle
        $traffic set rate_ $rate
        $traffic attach-agent $udp

        $ns connect $udp $sink
        return $traffic
}

proc attach-expoo-traffic-with-fixed-prio { node sink size burst idle rate prio}
{
        global ns

        set udp [new Agent/UDP]
        $udp set-fixed-prio $prio
        $ns attach-agent $node $udp

        set traffic [new Application/Traffic/Exponential]
        $traffic set packetSize_ $size
        $traffic set burst_time_ $burst
        $traffic set idle_time_ $idle
        $traffic set rate_ $rate
        $traffic attach-agent $udp

        $ns connect $udp $sink
        return $traffic
}

proc attach-expoo-traffic-with-uniform-prio { node sink size burst idle rate} {
        global ns

        set udp [new Agent/UDP]
        $udp set-uniform-prio
        $ns attach-agent $node $udp
```

```
        set traffic [new Application/Traffic/Exponential]
        $traffic set packetSize_ $size
        $traffic set burst_time_ $burst
        $traffic set idle_time_ $idle
        $traffic set rate_ $rate
        $traffic attach-agent $udp

        $ns connect $udp $sink
        return $traffic
}

#
# Create a traffic sink and attach it to the node node8
#
set sink0 [new Agent/LossMonitor]
$ns attach-agent $node10  $sink0
set sink1 [new Agent/LossMonitor]
$ns attach-agent $node11  $sink1

#
# Create a traffic source
#
set src0 [attach-expoo-traffic-with-fixed-prio $node1  $sink0 200 0 0 400k 1]
set src1 [attach-expoo-traffic-with-fixed-prio $node1  $sink0 200 0 0 400k 2]

$ns at 0.10 "[$LSR9 get-module MPLS] ldp-trigger-by-withdraw 10 -1"
$ns at 0.30 "[$LSR2 get-module MPLS] make-explicit-route 9 2_3_4_9 1000 -1"
$ns at 0.30 "[$LSR2 get-module MPLS] make-explicit-route 9 2_5_6_9 1001 -1"
$ns at 0.50 "[$LSR2 get-module MPLS] flow-erlsp-install 10 -1 1000 1_2"
$ns at 0.60 "[$LSR2 get-module MPLS] reroute-binding 10 -1 1001 2"
$ns at 0.70 "$src0 start"
$ns at 0.70 "$src1 start"
$ns rtmodel-at 1.00 down $LSR2 $LSR3
$ns rtmodel-at 1.50 up   $LSR3 $LSR2
$ns at 2.50 "$src0 stop"
$ns at 2.50 "$src1 stop"
$ns at 3.00 "finish"

#
# The last line finally starts the simulation
#
$ns run
```