ENSC 833-3: NETWORK PROTOCOLS and PERFORMANCE
CMPT 885-3: SPECIAL TOPICS: HIGH-PERFORMANCE NETWORKS

An INVESTIGATION of MPLS TRAFFIC ENGINEERING CAPABILITIES using CR-LDP

Spring 2001

FINAL PROJECT REPORT

David Culley
<culleyd@pmc-sierra.com>

Chris Fuchs
<chris.fuchs@bchydro.com>

Duncan Sharp
<dsharp@planetworks.ca>

April 12, 2001

Abstract
Multi Protocol Label Switching (MPLS) was initially proposed to overcome the bottleneck of IP routing over ATM while retaining the efficiency of ATM's label swapping and forwarding abilities. Now with the advent of gigabit routers the issue of connection oriented forwarding and IP routing integration is more focused on the additional advantages that MPLS provides: to manage traffic not necessarily based on shortest path measures, to provide QoS routing, the ability to set up Virtual Private Networks (VPN) and to implement congestion management control strategies. One control standard that allows MPLS to provide QoS based routing is the Constraint based Routing Label Distribution Protocol (CR-LDP). This paper reports on a graduate course project that uses simulation to demonstrate MPLS with CR-LDP to improve real-time traffic service quality while reducing congestion and improving network utilization.

1. INTRODUCTION

Internet Protocol (IP) networks have proven to be highly scalable and efficient platforms for delivering e-mail, Web and other classic Internet traffic where the delay of "best-effort" delivery are tolerable. It is widely believed that in order for the Internet to evolve into our future "universal information infrastructure" the Internet must be able to deliver real-time multi-media traffic – such as voice and video. Unfortunately by its nature, real-time traffic is sensitive to delay – for example; delay over 100 to 200 ms on a telephone connection impedes normal conversation.

In the current literature, achieving low delay with some probability of success (e.g., 98% of packets delivered within 100 ms) is referred to as "traffic engineering". Essentially this means availability of network resources to carry the traffic within the specified constraints. Which means, in turn, that network resources must be provisioned ahead of the offered real-time traffic and/or some connection admission policy must reject new connections that will, if otherwise accepted, overload and congest the network.

Multi Protocol Labeled Switching (MPLS) is a packet forwarding mechanism that is currently receiving considerable attention. One virtue of MPLS is its ability to support traffic engineering. The objective of our project is to demonstrate through simulation how MPLS can improve the quality of service (QoS) for real-time traffic on an IP network that is carrying a mix of real-time and best-effort traffic. Secondary objectives include: (i) to incorporate the MPLS and Label Distribution Protocol (LDP) software written for "network simulator" version 2.5b into version 2.6b; (ii) to extend the authors' understanding of the capabilities, limitations and implementation issues of MPLS; and (iii) for the authors to gain a working knowledge of an important network simulation tool.

This paper is organized to follow our project approach and methodology. Specifically, in Section 2, we present a concise description of MPLS and CR-LDP supported by key references. Section 3 describes our simulation in terms of the network, traffic, performance measurement arrangements, and results. The paper concludes in Section 4 with a brief discussion of results, observations, and possible future extensions to this work.

2. OVERVIEW of MPLS & CR-LDP

MPLS grew out of the efforts to make the Internet more scalable by reducing the complexity (cost) and increasing the speed of forwarding packets through a core (backbone) Internet router. The basic concept of MPLS is to forward packets using a short label instead of using the IP destination address to look-up the next hop in a routing table. Because the routing table is set up by a prearranged routing protocol (e.g. OSPF), the current "paradigm" combines packet forwarding with network routing. MPLS is a packet forwarding mechanism. The routing decisions may be made, and distributed in a variety of ways. One way, and the way explored in our project, is using Constraint-based Routing over the Label Distribution Protocol (CR-LDP).

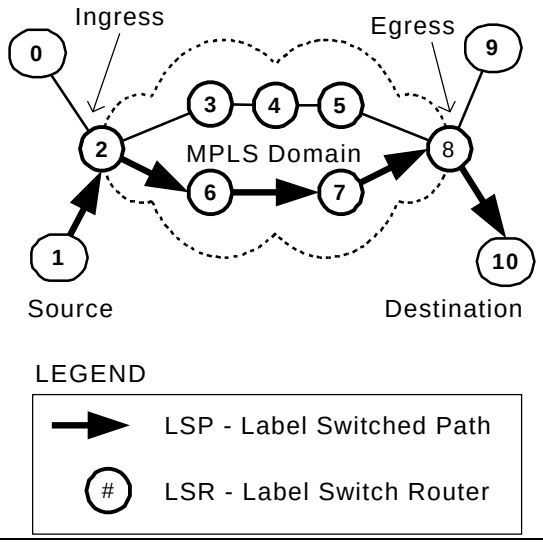

Figure-1 Basic elements and terminology for Multi Protocol Label Switching (MPLS)

As shown in Figure-1, MPLS works along what is called a label switched path (LSP) that is set up through a set of label switched routers (LSRs) – i.e., routers supporting MPLS. After the LSP is set up, packets follow this path and this traffic becomes "flow" or connection oriented. Path set up requires a suitable signaling protocol such as the Reservation Protocol (RSVP) or Label Distribution Protocol (LDP). An LSP originates at the edge of an MPLS domain where the first LSR maps the incoming traffic into forward equivalent classes (FECs). FECs are flexibly defined by a set of attributes such as a destination IP address, and for constraint-based routing (CR), class of service (CoS) or quality of service (QoS) parameters. Packets with the same FEC classification are given the same label and sent to the next LSR based on that label. Each LSR forwards the packet based only on the label and without using any other packet header information. At the last LSR in the MPLS domain, the label is removed and the packet is forwarded as a normal IP packet.

In the absence of routing constraints, the LSP is set up hop by hop using the routers IP forwarding table and thus the LSP is the path an IP packet would have followed using whatever default IP routing protocol is used. The LSP is "explicitly routed" if it is set up based on constraints specified by the network operator or computed using some network management algorithm that is independent of the default IP routing protocol. When LDP is used, this becomes constraint-based routing over LDP or CR-LDP. LDP provides LSRs with the following functions: (i) peer discovery (used to identify other LSRs in the MPLS domain), (ii) session (used to establish, maintain and delete sessions between LSR peers), (iii) advertisement (used to create, change and delete label mappings for FECs), and (iv) notification (used to provide status, diagnostic and error information). Advertisement messages include "requests" to set up an LSP which propagate forward, and label "mappings" which propagate back through the network.

For CR-LDP, an LSP is set up when a series of label request messages propagate forward from the ingress to the egress LSR and then, if the requested path satisfies the constraints (e.g., sufficient resources available), then labels are allocated and distributed (mapped) by a set of label-mapping messages that propagate backward from the egress LSR to the ingress LSR.

As of February 2001, when our project was developing, MPLS and LDP had become Internet standards [4,5] and CR-LDP was the subject of several Internet Drafts [1,2]. For additional information on MPLS and CR-LDP, including simulation and the application of MPLS for traffic engineering, refer to [3,6,7,8,9].

3. SIMULATION

Although MPLS has several desirable capabilities, we have chosen to demonstrate its ability to provide "traffic engineering" – i.e., to deliver adequate QoS for real-time traffic in a network that is carrying both best-effort and real-time traffic. To demonstrate this capability, we set up and ran the simulation described below using the network simulator tool "ns-2" including the animator "nam" [10]. See Appendix-A for a copy of the "Tcl" script. The simulation parameters were finalized for our demonstration runs after calculations and experimentation to ensure reasonable and illustrative results (see Section 4, Discussion).

3.1 Network Arrangements

The basic network topology, shown in Figure-2, was chosen with two paths such that default routing would be along the shortest path (i.e., through nodes 6 and 7). All nodes can be considered IP routers with nodes 2 through 8 also being MPLS capable – i.e., LSRs.

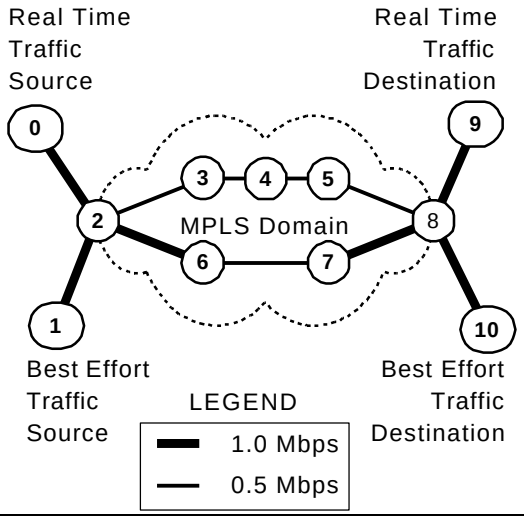

Figure-2 Network arrangements for our simulation of MPLS with CR-LDP.

All links were set up as duplex with 10 ms delay and using statistical fair queuing. The link data rates are shown on Figure-2. Note that (i) link 6-7 is a bottleneck on the shortest path between nodes 2 and 8; and (ii) the total network capacity between nodes 2 and 8 is 1 Mbps.

Queue monitors were set at each node to enable queuing delay to be observed during the network animations and to enable data to be dumped to file for performance analysis.

3.2 Traffic

The network was loaded with a mix of simulated "real-time" and "best effort" traffic. The best effort traffic provides background traffic, with the real-time traffic being of interest in terms of quality of service.

The real-time traffic connection was set up between node 0 and node 9 using User Datagram Protocol (UDP). We assumed constant bit rate (CBR) voice traffic with 48 byte packets and 3 ms inter arrival time. This represents, for example, 2 voice channels running at 64 kbps (G.711 PCM) with 6 ms packet delay. The 48 byte packets are indicative of an underlying Asynchronous Transfer Mode (ATM) cell relay bearer.

The best effort traffic connection was set up between node 1 and 10 using Transmission Control Protocol (TCP). We used the default ns-2 version of TCP (Tahoe) and packet size (1000 bytes). For authenticity, we used a genuine traffic trace exhibiting long range dependency (self-similar distribution).

Specifically, we used the pOct98_2000.TL trace from the course Web site [13]. This trace was collected starting at 11:00 on October 5, 1989 from local Ethernet traffic at Bellcore Morristown Research and Engineering facility [14]. The traffic trace contains packet arrival time and packet size in an ASCII format. Since ns-2 requires traffic traces to be in binary format, we wrote a trace conversion program in C, which is included in Appendix-B[1].

3.3 Performance Measurements

The animation tool "nam" was used to view the simulation. Nam proved to be an excellent means of visualizing network behavior for supporting simulation configuration decisions and for troubleshooting.

Measured performance included packet delay (for real-time traffic), packet loss and network utilization. We obtained the data for these parameters by saving the simulator trace from each simulation run in an output file. The output file was then post processed by, first filtering for the relevant data using a custom script written in Perl (see Appendix-C); then the filtered data was ported to a spreadsheet and manipulated into the tables and graphs for this report.

We captured and analyzed data from three basic simulation scenarios. These scenarios

[1] Note that together with an inverse program, that we also wrote, there is a slight accumulated error in arrival times between the original trace file and the one we translated into and back out of binary. This error is small and deemed inconsequential for the purposes of this simulation.

and the results are presented in the following sub sections.

3.4 Demonstration Scenario

To illustrate the impact of using CR-LDP to invoke explicit routing for a specific class of traffic (i.e., providing traffic engineering), the network in Figure-2 was operated (i) without any explicit routing for a short initial period during which all traffic takes the default (shortest path) route through nodes 6 and 7; then (ii) after this period, CR-LDP was used to invoke an explicit route for the real-time traffic through nodes 3, 4 and 5. The time line for this demonstration is shown in Figure-3 and the actual "Tcl" script has been appended as Appendix-A.

Figure-4 illustrates the results of a typical run, showing severe delay, delay variation and packet loss being experienced by the real-time traffic prior to the invocation of an "engineered" LSP for the real-time traffic.

Time (sec)	Action
0.00	Start simulation run
0.05	Start real-time traffic source
0.10	Start best effort traffic source
1.60	Send withdraw real-time traffic route message using CR-LDP
1.80	Send explicit route set up for real-time traffic to take LSP thru 3, 4 & 5
2.00	Issue LSP install directive at node 2
2.80	Send withdraw real-time traffic route
3.00	Stop traffic sources
3.20	Stop simulation run

Figure-3 Demonstration Script Time Line

3.5 Default (Non Engineered) Scenario

To collect sufficient simulation data to illustrate the consequences of the uncontrolled mixing of real-time and best effort traffic on a busy network, The network was simulated for 60 seconds using MPLS, however the routing for all traffic was determined by the default routing protocol and thus followed the shortest path through nodes 6 and 7. The results were aggregated and presented in Figure-5 (showing real-time packet delay distribution) and Figure-6 (showing packet loss rates and network utilization). Note that 69% of the real-time traffic packets were dropped under these congestion situations.

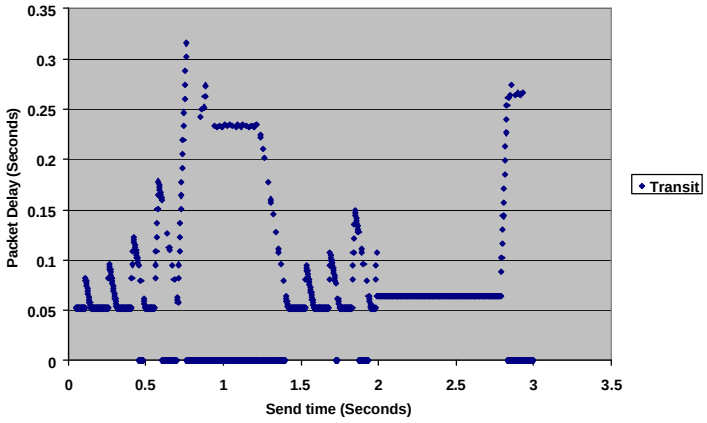

Figure-4 Scatter plot graph of delay to simulated real-time traffic versus time. At time t = 2, traffic engineering is implemented by routing the real-time and best effort traffic onto separate LSPs.

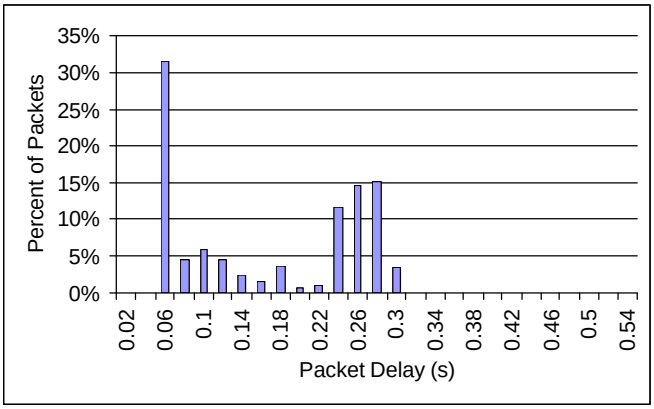

Figure-5 Real-time packet delay distribution for loaded network operation without traffic engineering. Note – 69% of packets dropped.

3.6 Traffic Engineered Scenario

After CR-LDP sets up an explicit route that has been engineered for the real-time traffic, the quality of service problem disappears and the performance measurements become stable and uninteresting (as may be seen from the right hand side of the graph in Figure-4). Simulation data with traffic engineering was used as a reference baseline for comparison. A 60 second simulation was run with all real-time traffic explicitly routed through nodes 3, 4 and 5.

The results are presented in Figure-6 (packet loss rates and network utilization comparisons). The delay for real-time traffic stabilized at 63.84 ms (as is seen from Figure-4 and is consistent

with expectations based on the calculated delay).

Parameter (units)		WITHOUT Traffic Engineering	WITH Traffic Engineering
Offered Traffic (Erl)		0.555	0.624
Packet Loss Rate (%)	TCP	1.0%	0.5%
	CBR	69.4%	0.0%
Network Utilization (Erl)		0.462	0.622

Figure-6 Comparison table of packet loss rate and network utilization (based on averages across the total simulation periods).

4. DISCUSSION

The network topology and other simulation parameters were chosen to demonstrate a seriously degraded quality of service condition. The configuration was arrived at after some calculation and experimentation with network scale (number of nodes, link capacity and delay) and traffic arrangements (sources and packet sizes, and CBR packet arrival rates). In particular, and as expected, smaller TCP packets improve the performance of the CBR traffic (in terms of both delay and loss) but at the expense of slower TCP start up. Although the chosen parameters can be argued to be artificially extreme, the mechanisms affecting performance, the general delay and loss effects do illustrate why, for example, voice over IP on the Internet suffers from unpredictable and often unusable quality.

The affect of best effort traffic over TCP on real-time traffic is amply demonstrated by our example network. Figure-4 is particularly illustrative. The series of delay escalations (and gaps where packets are dropped) is graphic evidence of the TCP window opening to sense the allowable capacity, then closing in response to congestion.

Using "nam" to observe network behavior during the non-engineered scenario runs, we witness (i) the default routing of all traffic on the shortest path, (ii) the build up of queue depth at node 6 where the capacity bottleneck starts, and (iii) the stranding and overflow of CBR (real-time) traffic in proportion to the number of packets[2]. When traffic engineering is invoked by the set up of an explicit route for the CBR traffic, we noted that there was a period during which CBR packets were arriving out of order as the buffers on the shortest path route cleared their queues. The comparison table in Figure-6 summarizes the performance gains achieved through traffic engineering for our specific network and traffic situation.

The outcome of our project was highly predictable – separating the real-time traffic onto routes that can be engineered for adequate resources to deliver a good quality of service, improves overall network performance in terms of packet loss rates and delay. In this sense, our project does not extend the state of the art. It was, however, instructive for the authors and we submit the following course contributions:

(a) Incorporating the MPLS and CR-LDP modules from [11] into ns-2.6b and building an ns-2.6b executable version.
(b) Writing a flexible Tcl script for demonstrating the operation and application of MPLS and CR-LDP for traffic engineering (Appendix-A).
(c) Developing a routine in C to convert the Bellcore Ethernet traffic traces [13,14] into a form compatible with ns-2 (Appendix-B).
(d) Writing a flexible script in Perl to filter the relevant data from the full simulation trace (Appendix-C).

The ns-2 executable is available in /ensc/grad1/cwfuchs/proj/ns-2.1b6/ns . The other contributions are in the appendices to this report and will be made available for download from the course project Web site [13].

The use of CR-LDP to provide traffic engineering by invoking an explicit route for designated traffic classes was useful for demonstration. This method involves intervention by an intelligent operator to engineer the network. While this is not as elegant as a fully automated process that uses a suitable policy based routing algorithm,

[2] In ns-2, statistical fair queuing appears to be packet based and, since the CBR constitutes a relatively large number of packets, it suffers proportionally higher packet loss in the buffer overflows.

"manual" route assignment can be extremely useful. Specifically, voice traffic is generally well behaved statistically and engineering for it is a well developed and understood discipline. Thus, with MPLS and CR-LDP, it is easy to visualize the rapid evolution of efficient and effective network architectures to support real-time multi-media traffic.

Of course the end game should be to provide a fully automated and self organizing network capability. A future extension to our project would be to implement policy based CR-LDP such that flows (LSPs) are set up autonomously. Implementing this as a simulation in ns-2 was investigated and the following steps identified:

- Implementing the ability to negotiate resources at LSRs and to maintain these resource reservations (hold state).
- Implementing a suitable routing algorithm, for example, a type of distance vector algorithm where vector parameters include link capacity, traffic loading, delay, etc.

Although we investigated the implementation of these enhancements, they require a significant amount of customization to several ns-2 modules and were thus considered unattainable in the course time frame. In addition to customizing the ns-2 code, it would also be necessary to implement a more complex network with more routing choices and more complex background traffic arrangements.

ACKNOWLEDGEMENTS

We are indebted to Gaeil Ahn (at fog1@ce.cnu.ac.kr) in Korea for the MPLS node, CR- LDP and associated software modules [11]. We also sincerely appreciated the timely advice and support from our course professor, Dr. Ljiljana Trajkovic, and our TA, Mr. Milan Nikolic.

REFERENCES

[1] "Constraint-Based LSP Setup using LDP", *IETF Internet Draft*, July 2000, [http://search.ietf.org/internet-drafts/draft-ietf-mpls-cr-ldp-04.txt]

[2] "LDP State Machine", *IETF Internet Draft*, January 2000, [http://search.ietf.org/internet-drafts/draft-ietf-mpls-ldp-state-03.txt]

[3] B. Davie & Y. Rekhter, "MPLS Technology and Applications, *Morgan Kaufman Publishers Inc.*, US, 2000

[4] "Multiprotocol Label Switching Architecture", *IETF Internet Request for Comments*, RFC 3031, January 2001 [http://www.ietf.org/rfc/rfc3031.txt?number=3031]

[5] "LDP Specification", *IETF Request for Comments*, RFC 3036, January 2001 [http://www.ietf.org/rfc/rfc3036.txt?number=3036]

[6] T. Chen & T. Oh, "Reliable Services in MPLS", *IEEE Communications Magazine*, Vol.37, No.12, pp.58-62, Dec 1999

[7] E. Lim, H. Shin, Y. Kim, "Implementation of the Simulation Model for the MPLS Signaling Protocol and OAM Functions With OPNET", [http://www.mil3.com/products/modeler/biblio.html]

[8] "Using CR-LDP for Service Interworking, Traffic Engineering, and Quality of Service in Carrier Networks", *Nortel Networks*, White Paper, September 2000

[9] A. Ghanwani et al, "Traffic Engineering Standards in IP Networks Using MPLS", *IEEE Communications Magazine*, Vol.37, No.12, pp.49-53, December 1999]

[10] ns-2 network simulator, [http://www.isi.edu/nsnam/ns/]

[11] MPLS nodes & CR-LDP modules, [http://www.raonet.com]

[12] T. Nguyen, et al, "Voice over IP Service and Performance in Satellite Networks", *IEEE Communications Magazine*, Vol.39, No.3, pp.164-171, March 2001

[13] SFU ENSC-833, Course Web site [http://www.ensc.sfu.ca/people/faculty/ljilja/ENSC833]

[14] W. Leland et al, "On the Self-Similar Nature of Ethernet Traffic", *ACM SIGComm '93*, San Francisco, USA, September 1993

APPENDIXES

Appendix-A
Simulation Tcl Script

```
# ENSC-833 Project by Fuchs, Culley & Sharp
# Spring Semester 2001
# MPLS Simulation Script

# Current File Name: z5.tcl
```

```
# Developed to run on ns-2.6b
# Using MPLS & LDP code from
# Gaeil Ahn (fog1@ce.cnu.ac.kr), Jan. 2000


# SIMULATOR PRELIMINARIES

# Create a simulator object
set ns [new Simulator]

# Open a name trace file
set nf [open test-mpls.nam w]
$ns namtrace-all $nf

# Open a parameter trace file
set nfz [open z3.out w]
$ns trace-all $nfz

# Define a 'finish' procedure
proc finish {} {
        global ns nf nfz
        $ns flush-trace
        close $nf
        close $nfz
        exec nam test-mpls.nam &
        exit 0
}

# DEFINE the NETWORK
# as 2 source nodes connecting 2 destination
# nodes thru a network of MPLS LSRs

# Create network nodes
set n0    [$ns node]
set n1    [$ns node]
set LSR2  [$ns MPLSnode]
set LSR3  [$ns MPLSnode]
set LSR4  [$ns MPLSnode]
set LSR5  [$ns MPLSnode]
set LSR6  [$ns MPLSnode]
set LSR7  [$ns MPLSnode]
set LSR8  [$ns MPLSnode]
set n9    [$ns node]
set n10 [$ns node]

# Create network links
$ns duplex-link $n0 $LSR2  1Mb  10ms DropTail
$ns duplex-link $n1 $LSR2  1Mb  10ms DropTail
$ns duplex-link $LSR2  $LSR3  0.5Mb  10ms DropTail
$ns duplex-link $LSR3  $LSR4  0.5Mb  10ms DropTail
$ns duplex-link $LSR4  $LSR5  0.5Mb  10ms DropTail
$ns duplex-link $LSR5  $LSR8  0.5Mb  10ms DropTail
$ns duplex-link $LSR2  $LSR6  1Mb  10ms SFQ
$ns duplex-link $LSR6  $LSR7  0.5Mb  10ms SFQ
$ns duplex-link $LSR7  $LSR8  1Mb  10ms SFQ
$ns duplex-link $LSR8  $n9  1Mb  10ms DropTail
$ns duplex-link $LSR8  $n10  1Mb 10ms DropTail

# Set queue monitors
$ns duplex-link-op $n0 $LSR2 queuePos 0.5
$ns duplex-link-op $n1 $LSR2 queuePos 0.5
$ns duplex-link-op $LSR2 $LSR3 queuePos 0.5
$ns duplex-link-op $LSR3 $LSR4 queuePos 0.5
$ns duplex-link-op $LSR4 $LSR5 queuePos 0.5
$ns duplex-link-op $LSR2 $LSR6 queuePos 0.5
$ns duplex-link-op $LSR6 $LSR7 queuePos 0.5
$ns duplex-link-op $LSR5 $LSR8 queuePos 0.5
$ns duplex-link-op $LSR7 $LSR8 queuePos 0.5
$ns duplex-link-op $LSR8 $n9 queuePos 0.5
$ns duplex-link-op $LSR8 $n10 queuePos 0.5

# Configure LDP agents on all MPLS nodes
$ns configure-ldp-on-all-mpls-nodes

# Set LDP message color for animation
$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

# Set colors for traffic animation
$ns color 1 Hotpink
$ns color 2 Navyblue

# Set LDP events
$ns enable-control-driven

# Other LDP event options
#$ns enable-data-driven
#$ns enable-on-demand
#$ns enable-ordered-control

# Set up default routing protocol
$ns rtproto DV

# DEFINE the TRAFFIC
# by setting up a UDP "connection" for CBR traffic
# and a TCP connection for trace traffic

# Create UDP "connection" between n0 & n9
set udp0 [new Agent/UDP]
set null [new Agent/Null]
$ns attach-agent $n0 $udp0
$ns attach-agent $n9 $null
$ns connect $udp0 $null
$udp0 set class_ 1

# Put CBR traffic on UDP connection
set Src0 [new Application/Traffic/CBR]
$Src0 set packetSize_ 48
$Src0 set interval_ 0.003
$Src0 attach-agent $udp0

# Create TCP connection between n1 & n10
# and set packet (segment) size to 1000 (ns default)
set tcp [new Agent/TCP]
$tcp set packetSize_ 1000
set tcpsink [new Agent/TCPSink]
$ns attach-agent $n1 $tcp
$ns attach-agent $n10 $tcpsink
$ns connect $tcp $tcpsink
$tcp set class_ 2

# Put trace driven traffic over the TCP connection
set tfile [new Tracefile]
$tfile filename pOct89_2000_TL.bin
set Src1 [new Application/Traffic/Trace]
$Src1 attach-tracefile $tfile
```

```
$Src1 attach-agent $tcp

# SCHEDULE the SIMULATION
$ns at 0.05  "$Src0 start"
$ns at 0.1  "$Src1 start"
$ns at 1.6  "$LSR8 ldp-trigger-by-withdraw 9 -1"
$ns at 1.8  "$LSR2 make-explicit-route 8  2_3_4_5_8
3000  -1"
$ns at 2.0  "$LSR2 flow-erlsp-install  9 -1   3000"
#The following line turns off our explicit routing
$ns at 2.8  "$LSR2 ldp-trigger-by-release  9 3000"
$ns at 3.0 "$Src1 stop"
$ns at 3.0 "$Src0 stop"

# EXCECUTE the SIMULATION

# Call finish procedure after 3.2 secs of simulation time
$ns at 3.2 "finish"

# Run the simulation
$ns run
```

Appendix-B
Program to Convert Traffic Trace

```
#include <stdio.h>

typedef struct _trec {
  unsigned int trec_time;
  unsigned int trec_len;
} trec;

main (int argc, char** argv)
{ trec t;
  double last_time;
  double this_time;
  void exit();
  char *pt;
  char i;
  FILE *infile;
  FILE *outfile;

  if (argc == 3)
  { infile = fopen (argv[1], "r");
    if (infile == NULL)
    { fprintf (stderr, "Unable to open %s for reading\n",
argv[1]);
      exit (0);
    } else
    { fprintf (stderr, "Opening %s for reading\n", argv[1]);
    }
    outfile = fopen (argv[2], "w");
    if (outfile == NULL)
    { fprintf (stderr, "Unable to open %s for writing\n",
argv[2]);
      exit (0);
    } else
    { fprintf (stderr, "Opening %s for writing\n", argv[2]);
    }
  } else
  { fprintf (stderr, "Wrong number of args: %d\n", argc);
    fprintf (stderr, "  USAGE: %s infile outfile\n", argv[0]);
```

Appendix-C
Program to Filter Simulator Trace File

```
#!/usr/local/bin/perl
#
#
@n = split (/\//, $0);
$0 = $n[$#n];
$debugfile = "$0.debug";

die "usage: $0 data-filename \n" unless (-e $ARGV[0]);
$outfil=$ARGV[0];

open(DATA,"$outfil");
open (DEBUG, ">$debugfile") || die "$0: Can't open
$debugfile for writing\n";

while ($line=<DATA>) {

($que,$tim,$src,$dst,$typ,$siz,$flg,$ipflw,$ipsrc,$ipdst,$
seq,$id)=
  split(/\s+/,$line);

  print DEBUG;
  next if ($typ ne 'cbr');

  if ($src == 0 && $que eq '+' ) {
   $p{$id}{s}=$tim;
   print DEBUG "id:$id s:$tim\n";
   next
  } elsif ($que eq 'r' && $dst == 9 && exists($p{$id}) ) {
       $p{$id}{r}=$tim;
```

```
        print DEBUG "id:$id r:$tim\n";
   }
}
close(DATA);

@srt=sort {$p{$a}{s} <=> $p{$b}{s}} (keys(%p));

$drop = 0;
foreach $key (@srt) {
  $transit=$p{$key}{r}-$p{$key}{s};
  $timein=$p{$key}{s};
  if ($p{$key}{r} eq "")
  { $drop += 1;
    print "id = $key send_time=$timein DROP=$drop\n";
  }
  else
  { print "id = $key send_time=$timein transit=$transit
(s=$p{$key}{s} r=$p{$key
}{r})\n";
  }
};
```