ENSC 835-3: NETWORK PROTOCOLS AND PERFORMANCE
CMPT 885-3: SPECIAL TOPICS: HIGH-PERFORMANCE NETWORKS

Ljiljana Trajkovic

An Analysis of Constraint-based Routing in MPLS

Spring 2002

FINAL PROJECT REPORT

Tony Dongliang Feng
tdfeng@cs.sfu.ca

April 15, 2002

Abstract

This paper discusses Constraint-based Routing with Multi-Protocol Label Switching (MPLS) in a simulation context. We first briefly review MPLS, Traffic engineering and Constraint-based Routing to provide a background for our simulations. We then report a graduate course project that use simulation to demonstrate the MPLS with Constraint-based routing to improve the overall traffic delay. We also discuss the results we obtained and provide some extension work possible.

1.Introduction

IP networks offer unparalleled scalability and flexibility for rapid deployment of value-added IP services. However, with the increasing demand and explosive growth of the Internet, carriers require a network infrastructure that is dependable, predictable, and offers consistent network performance. Traffic engineering and differentiated services are the two cornerstones needed to achieve mission-critical networking.

The Multi-Protocol Label Switching (MPLS) approach proposed by the Internet Engineering Task Force (IETF) is to be the networking technology to deliver traffic engineering capability and QoS performance for carrier networks to support differentiated services. MPLS can deliver control and performance to IP data packets through the use of label switched paths (LSPs). One protocol used to implement the LSPs is Constraint-based routing using Label Distribution Protocol (CR-LDP)

In this paper, we describe the simulations using Constraint-based Routing capability of the MPLS model MNS2 in NS-2. We compare the overall packet delay from two scenarios: In scenario 1 LSPs are set up in the increasing importance order. While in scenario 2, LSPs are set up in the decreasing importance order. And show that the set up order of LSPs has important impact on the overall packet delay of all network traffics.

The organization of the rest of the paper is as follows. In Section 2, we describe the MPLS traffic engineering and Constraint-based Routing. Section 3 we describe our simulation and results. The paper concludes in Section 4 with a brief discussion of results, observations and possible future extensions.

2. MPLS Traffic Engineering and Constraint-based Routing.

2.1 MPLS Overview

MPLS is the latest technology in the evolution of routing and forwarding mechanisms for the core of the Internet. The "label" in MPLS is a short, fixed-length value carried in the packet's header to identify a *Forwarding Equivalence Class* (FEC). A FEC is a set of packets that are forwarded over the same path through a network. FECs can created from any combination of source and destination IP addresses, transport protocol, port numbers etc. Labels are assigned to incoming packets using a FEC to label mapping procedure at the edge routers. From that point on, it is only the labels that dictate how the network will treat these packets—i.e., what route to use, what priority to assign, and so on. MPLS defines *label-switched paths* (LSP), which are pre-configured to carry packets with specific labels. These LSPs can be used to forward specific packets through specific routes, thus facilitating traffic engineering.

2.2 MPLS Traffic Engineering

MPLS makes it easy to commit network resources in such a way as to balance the load in the face of a given demand and to commit to differential levels of support to meet various user traffic requirements. The ability to dynamically define routes, plan resource commitments on the basis of known demand, and optimize network utilization is referred to as *traffic engineering*.

With the basic IP mechanism, there is a primitive form of automated traffic engineering. Specifically, routing protocols such as OSPF enable routers to dynamically change the route to given destination on a packet-by-packet basis to try to balance load. But such dynamic routing reacts in a very simple meaner to congestion and does not provide a way to support QoS. All traffic between two endpoints follows the same route, which may be changed when congestion occurs. MPLS, on the other hand, is aware of not just individual packets, but flows of packets in which each flow has certain QoS requirements and a predictable traffic demand. With MPLS, it is possible to set up routes on the basis of these individual flows, with two different flows between the same endpoints perhaps following different routers. Further, when congestion threatens, MPLS paths can be rerouted intelligently. That is, instead of simply changing the route on a packet-packet basis, with MPLS, the routes are changed on a flow-by-flow basis, taking advantage of the known traffic demands of each flow. Effective use of traffic engineering can substantially increase usable network capacity.

2.3 Constraint-Based Routing

Constraint-based Routing (CBR) computes routes that are subject to constraints such as bandwidth and administrative policy. Because Constraint-based Routing considers more than network topology in computing routes, it may find a longer but

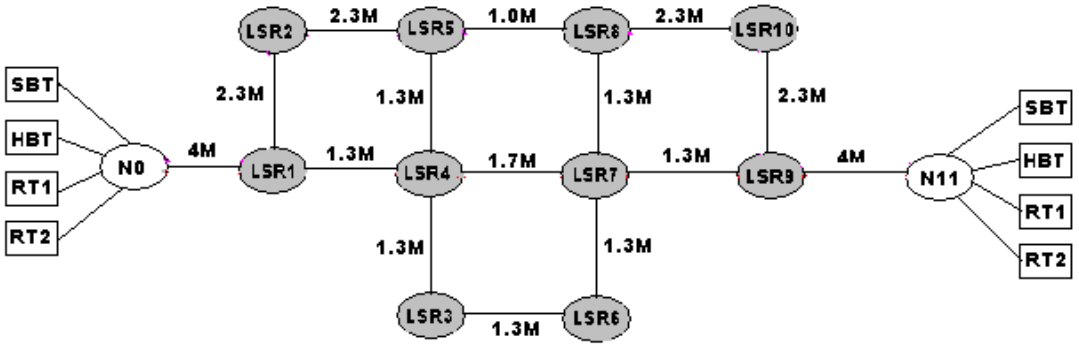
lightly loaded path better than the heavily loaded shortest path. Network traffic is hence distributed more evenly.

Although finding the optimal route using constraint-based routing is known to be computationally difficult for almost any realistic constraint-limited routing problem, a simple heuristic can be used to find a route satisfying a set of constraints—if one exists. The traffic engineer may simply prune resources that do not match the traffic trunk attributes and run a shortest path route computation on the residual graph. Other approaches may be used as well.

3. Simulation

3.1 Network Arrangements

The network topology is shown in Figure 1. All nodes can be considered IP routers and LSR1 to LSR8 are MPLS capable.



Firgure 1 Simulation Network Topology

All links were set up as duplex with 10ms delay and links between LSRs are using Class-based Queuing to support multiple service levels. The link data rates are shown on Figure 1. Note that link 1-4,7-9 and 5-8 are three bottlenecks. When the bandwidths on these three links are used up, the LSPs have to be routed through a longer path using the vertical links.

3.2 Traffic

The network was loaded with mix of four types of simulated “real-time” and “best-effort” traffics with different bandwidth requirements:

Real-time traffic 2 (RT2) was set up as a *constant bit rate* (CBR) traffic with packet size of 200 bytes and bandwidth requirement of 1000 kbps.
Real-time traffic 1 (RT1) was set up as a *constant bit rate* (CBR) traffic with packet size of 200 bytes and bandwidth requirement of 800 kbps.

High priority Best Effort traffic (HBT) was set up as an Exponential on/off traffic with packet size of 200 bytes, burst time of 500 ms, idle time of 500 ms, and bandwidth requirement of 300 kbps.
Simple Best Effort traffic (SBT) was set up as an Exponential on/off traffic with packet size of 200 bytes, burst time of 200 ms, idle time of 800 ms, and bandwidth requirement of 100 kbps.

Each traffic connection were setup between Node 0 and Node 11 with Node 0 as the source and Node 11 as the sink.

3.3 Performance Statistics

In the simulation, we use the animation tool "nam" to view the traffic behavior. By visualizing the packet flowing, the packet sizes and packet types; "nam" is an excellent tool to support simulation configuration decision and troubleshooting.

We concentrate on measure the packet delay of each type of traffic. We first create the simulator trace files from each scenario; then the statistics of packet delay was filtered out from the trace file using a custom script written in Perl (see Appendix-B); then the statistics data can be manipulated into tables and graphs for this report.

We measure and analyzed date form two scenarios. Details and result of these scenarios are described in the following sections:

3.5 Scenario 1

The importance order of LSPs means that the LSPs are sorted one by one, starting form high priority LSPs. For LSPs with the same priority, sort them in the order of decreasing bandwidth requirement.

In this scenario the LSPs are set up using Constraint-based routing in an increasing importance order, and the traffic started right after its LSP was set up. The path set up was illustrate in Figure2.

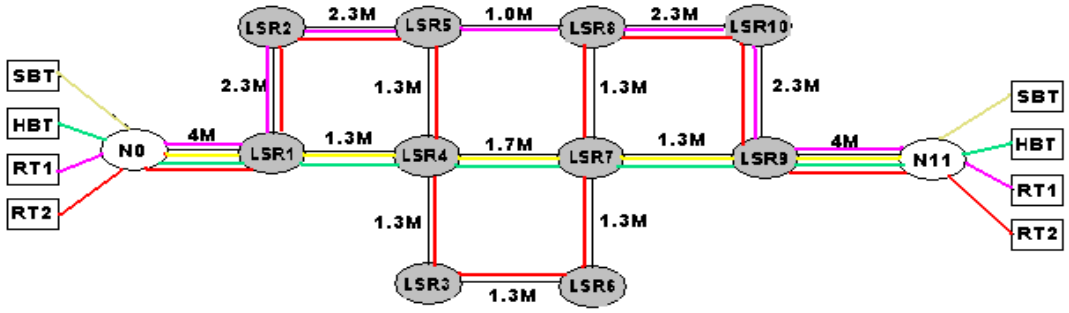


Figure 2 LSPs set up in Scenario 1

And the packet delay statistics is shown in Figure 3.

Traffic Type	Bandwidth (kbps)	Packets Sent	Packets Dropped	Packets lost	Average delay (ms)
SBT	100	82	0	0	54.3
HBT	300	224	0	0	54.4
RT1	800	1248	13	1.04%	78.2
RT2	1000	1381	35	2.53%	123.7

***Overall delay: 96.89 ms**

Figure 3 Packet delay statistics in Scenario 1

3.6 Scenario 2

In this scenario the LSPs are set up using Constraint-based routing in a decreasing importance order, and the traffic started right after its LSP was set up. The path set up was illustrated in Figure3.

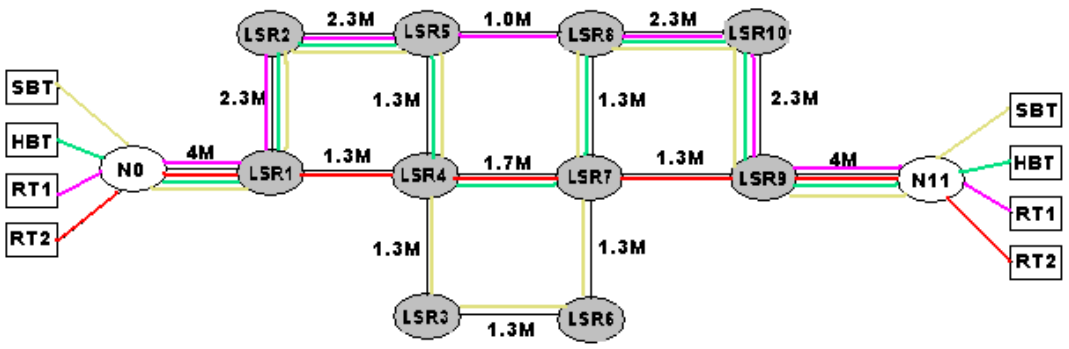


Figure 4 LSPs set up in Scenario 2

and the packet delay statistics is shown in Figure 5.

Traffic Type	Bandwidth (kbps)	Packets Sent	Packets Dropped	Packets lost	Average delay (ms)
SBT	100	30	0	0	120.5
HBT	300	166	0	0	97.7
RT1	800	1348	11	0.82%	77.8
RT2	1000	1823	15	0.82%	56.9

*Overall Delay: 67.88 ms.

Figure 3 Packet delay statistics in Scenario 1

4. Discussion

The network topology and the simulation scenario are decided to show the change of LSPs set up and its impact in the overall packet delay.

In Constraint-based Routing, the LSP, which is currently setting up, will use the shortest path that satisfies all the constraints. Confirm to this behaviour, in Scenario 1, we can see that the SBT and HBT traffic use the shortest path of 1-4-7-9 and RT1 using the second shortest path 1-2-5-8-10-9, while RT2 uses the longest path 1-2-5-4-3-6-4-8-10-9 (Figure 2). Since the lager LSP takes the longer path, it naturally that the overall delay will be higher.

On the contrast, in Scenario 2, the higher priority traffics with higher bandwidth take the shorter path (Figure 4). Thus the overall delay was optimized from 96.89 ms to 67.88 ms.

The paths for the LSPs can be computed by some offline Constraint-based Routing algorithm. By taking all the LSP requirements, link attributes and network topology information into consideration, an offline Constraint-based Routing server may be able to find better LSP placement than online Constraint-based Routing, where every router in the network finds paths for its LSPs separately based on its own information. Offline Constraint-based Routing will compute paths for LSPs periodically, e.g. daily. The path information of the LSPs is downloaded into the routers. Online Constraint-based Routing is still used in the routers, so that if some routers or links fails, new paths will be computed for those affected LSPs immediately. A simple offline Constraint-based Routing algorithm proposed by X. Xiao and L. Ni. In [5].

The outcome of our project was highly predictable- let the lager traffic use the shorter path to improve the overall packet delay. However, we submit the following course contributions:

(1) Incorporating the MPLS modules MNS2.0 into ns-2.1b8 and build an ns-2.1b8 a executable version that support MPLS Constraint-based Routing.
(2) Writing flexible TCL scripts of demonstrating the application of MNS 2.0 for Constraint-based Routing (See Appendix A).
(3) Writing flexible Perl scripts to filter the statistics data from the simulation trace files.

The ns-2 executable is available in /cs/grad1/tdfeng/ns-2/ns2.1b8/ns2.1b8a/ns.

A future extension to our project would be enlarging our network module and verifying the efficiency of scenario 2 in a more complex traffic environment. After this, we can implement an Offline Constraint-based routing algorithm to enable offline LSP computation and realize the LSPs set up of scenario 2 in an offline scheme.

Acknowledgements

We are indebted to Gaeil Ahn for the MNS2.0 [1] and Christian Glomb [7] for the adapted version to ns-2.1b8. Also, we greatly appreciated the timely advice and support from our course professor, Dr. Ljiljana Trajkovic and our TA, Mr. Wen Jin.

Reference:

[1] Gaeil Ahn, Woojik Chun “Design and Implementation of MPLS Network Simulator (MNS)”, March 2002
http://flower.ce.cnu.ac.kr/~fog1/mns/mns2.0/doc/MNS_v2.0_arch.pdf
[2] William Stallings, “MPLS”, the Internet Protocol Journal, September 2001, http://www.cisco.com/warp/public/759/ipj_4-3.pdf
[3] Paul Brittain, Adrian Farrel, “ MPLS traffic engineering: a choice of signalling protocols”, Jan. 2000. http://www.dataconnection.com/download/crldprsvp.pdf
[4] David Culley, Chris Fuchs, Duncan Sharp, “ An Investigation of MPLS traffic engineering capabilities using CR-LDP”, http://www.ensc.sfu.ca/~ljilja/ENSC833/Projects/ENSC833.projects.html, Spring 2001
[5] XiPeng Xiao, A. Hannan, B. Bailey, S. Carter, L. M. Ni, "Traffic Engineering with MPLS in the Internet", IEEE Network magazine, pp. 28-33, March 2000.
http://www.cse.msu.edu/~xiaoxipe/papers/mplsTE/mpls.te.pdf
[6] XiPeng Xiao, Thomas Telkamp, Lionel M. Ni, "A Practical Approach for Providing QoS in the Internet Backbone", Aug. 2001
[7] MNS-v2.0, christian.glomb@mchp.siemens.de
[8] B. Davie, Y. Rekhter, “MPLS Technology and Applications”, Morgan Kaufman Publishers Inc., US, 2000
[9] B. Jamoussi, Ed., L. Andersson, R. Callon, R. Dantu IETF RFC 3212 “Constraint-Based LSP Setup using LDP”,. January 2002.

[10] E. Rosen, A. Viswanathan, R. Callon, IETF RFC 3031 “Multiprotocol Label Switching Architecture”. January 2001

Appendixes

Appendix-A
Simulation Tcl Script for Scenario 2.

```
# Copyright (c) 2000 by Gaeil Ahn
# Everyone is permitted to copy and distribute this software.
# Please send mail to fog1@ce.cnu.ac.kr when you modify or distribute
# this sources.

#Adapted to ns-2.1b8 by Christian.Glomb@mchp.siemens.de
# If you have problems, found bugs or know improvements,
# please e-mail me (cc to Gaeil Ahn and Haobo Yu).


set ns [new Simulator]

set na [open test_OCPC.tr w]
set nf [open test_OCPC.nam w]
$ns trace-all $na
$ns namtrace-all $nf
proc finish {} {
        global ns na nf
        $ns flush-trace
        close $na
        close $nf
        exit 0
}

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

        set source [new Agent/CBR/UDP]
        $ns attach-agent $node $source
        set traffic [new Traffic/Expoo]
        $traffic set packet-size $size
        $traffic set burst-time $burst
        $traffic set idle-time $idle
        $traffic set rate $rate
        $source attach-traffic $traffic
        $ns connect $source $sink
        return $source
}

# routing protocol
$ns rtproto DV

# make nodes & MPLSnodes
```

```
set node0  [$ns node]
set LSR1   [$ns mpls-node]
set LSR2   [$ns mpls-node]
set LSR3   [$ns mpls-node]
set LSR4   [$ns mpls-node]
set LSR5   [$ns mpls-node]
set LSR6   [$ns mpls-node]
set LSR7   [$ns mpls-node]
set LSR8   [$ns mpls-node]
set LSR9   [$ns mpls-node]
set LSR10  [$ns mpls-node]
set node11 [$ns node]

# make links

$ns duplex-link $node0 $LSR1  4Mb   10ms DropTail

$ns duplex-link $LSR1  $LSR4  1.3Mb   10ms CBQ
$ns duplex-link $LSR4  $LSR7  1.7Mb   10ms CBQ
$ns duplex-link $LSR7  $LSR9  1.3Mb   10ms CBQ

$ns duplex-link $LSR1  $LSR2  2.3Mb  10ms CBQ
$ns duplex-link $LSR2  $LSR5  2.3Mb  10ms CBQ
$ns duplex-link $LSR5  $LSR8  1Mb  10ms CBQ
$ns duplex-link $LSR8  $LSR10 2.3Mb  10ms CBQ
$ns duplex-link $LSR10 $LSR9  2.3Mb  10ms CBQ

$ns duplex-link $LSR4  $LSR5  1.3Mb  10ms CBQ
$ns duplex-link $LSR7  $LSR8  1.3Mb  10ms CBQ

$ns duplex-link $LSR3  $LSR4  1.3Mb  10ms CBQ
$ns duplex-link $LSR3  $LSR6  1.3Mb  10ms CBQ
$ns duplex-link $LSR6  $LSR7  1.3Mb  10ms CBQ



$ns duplex-link $LSR9  $node11  4Mb  10ms DropTail

$ns duplex-link-op $LSR1 $LSR4 queuePos 0.8
$ns duplex-link-op $LSR4 $LSR7 queuePos 0.8
$ns duplex-link-op $LSR2 $LSR5 queuePos 0.8

#
# configure ldp agents on all mpls nodes
#
$ns configure-ldp-on-all-mpls-nodes

# configure-cbq-for-SBTS {qlim cbq_qtype okborrow bw maxidle extradelay}
$ns cfg-cbq-for-SBTS      10  DropTail    1     0.1    auto       0
$ns cfg-cbq-for-HBTS      10  DropTail    1     0.05  auto       0
$ns cfg-cbq-for-RTS       10  DropTail    0     0.8    auto       0
$ns cfg-cbq-for-STS       10  DropTail    1     0.05  auto       0

$ns bind-flowid-to-SBTS 0
```

```
$ns bind-flowid-to-SBTS 100
$ns bind-flowid-to-SBTS 200
$ns bind-flowid-to-SBTS 300
$ns bind-flowid-to-SBTS 400

$ns bind-ldp-to-STS

# set ldp-message clolr

$ns ldp-request-color       blue
$ns ldp-mapping-color       red
$ns ldp-withdraw-color      magenta
$ns ldp-release-color       orange
$ns ldp-notification-color  green

#
$ns collect-resource-info 4

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

#Create a traffic source
set src0 [attach-expoo-traffic $node0  $sink0 200 0 0 1000k]

$src0 set fid_ 100
$ns color 100  orange

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

#Create a traffic source
set src1 [attach-expoo-traffic $node0  $sink1 200 0 0 800k]

$src1 set fid_ 200
$ns color 200  magenta

#Create a traffic sink and attach it to the node node11
set sink2 [new Agent/LossMonitor]
$ns attach-agent $node11  $sink2
$sink2 clear

#Create a traffic source
set src2 [attach-expoo-traffic $node0  $sink2 200 500ms 500ms 300k]
$src2 set fid_ 300
$ns color 300  blue


#Create a traffic sink and attach it to the node node11
set sink3 [new Agent/LossMonitor]
$ns attach-agent $node11  $sink3
```

```
$sink3 clear

#Create a traffic source
set src3 [attach-expoo-traffic $node0  $sink3 200 200ms 800ms 100k]
$src3 set fid_ 400
$ns color 400  black


proc notify-erlsp-setup {node lspid} {
        global src0 src1 src2 src3

        set module [$node get-module "MPLS"]

        set ns [Simulator instance]
        if {[$node id] == 1} {
             #puts "      o The CR-LSP of lspid $lspid has been just established at [$ns now]"
              switch $lspid {
                    1100 {  $module bind-flow-erlsp   11 100  $lspid
                            $src0 start
                         }
                   1200 {  $module bind-flow-erlsp   11 200  $lspid
                            $src1 start
                         }
                    1300 {  $module bind-flow-erlsp   11 300  $lspid
                            $src2 start
                         }
                    1400 {  $module bind-flow-erlsp   11 400  $lspid
                            $src3 start
                         }
                default {
                            puts "error"
                            exit 1
                        }
             }
        }

}

proc notify-erlsp-fail {node status lspid tr} {

   puts "nodeid=[$node id] : status=$status  lspid=$lspid  tr=$tr"

}


proc constraint-based-routing { lspid sLSR dLSRid bw } {
   set sLSRmodule [$sLSR get-module "MPLS"]
   set er [$sLSRmodule constraint-based-routing $dLSRid $bw]
   if {$er != -1} {
           puts "--> The result of constraint-based routing for lspid $lspid : Explicit Route=$er"
           $sLSRmodule setup-crlsp  $dLSRid  $er  $lspid  $bw 400B 200B 7 3
    } else {
           puts "--> The result of constraint-based routing for lspid $lspid : Explicit Route= No path"
    }
```

```
}


$ns at 0.0   "constraint-based-routing 1100 $LSR1 9 1000K"
$ns at 0.2   "constraint-based-routing 1200 $LSR1 9 800K"
$ns at 0.4   "constraint-based-routing 1300 $LSR1 9 300K"
$ns at 0.6   "constraint-based-routing 1400 $LSR1 9 100K"

$ns at 3.0 "$src0 stop"
$ns at 3.0 "$src1 stop"
$ns at 3.0 "$src2 stop"
$ns at 3.0 "$src3 stop"

$ns at 3.1 "finish"

$ns run
```

Appendix-B
Perl Script to Filter Simulator Trace File

```
#!/usr/local/bin/perl
#
#

 @n =  split (/\/,$0);
 $0 =$n[$#n];
 $scr0 = "$0.1000k";
 $scr1 = "$0.800k";
 $scr2 = "$0.300k";
 $scr3 = "$0.100k";
 $debugfile = "$0.debug";

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

 open(DATA,"$outfil");
 open (SCR0,">$scr0")||die"$0: Can't open $scr0 for writing\n";
 open (SCR1,">$scr1")||die"$0: Can't open $scr1 for writing\n";
 open (SCR2,">$scr2")||die"$0: Can't open $scr2 for writing\n";
 open (SCR3,">$scr3")||die"$0: Can't open $scr3 for writing\n";
 open (DEBUG,">$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);

 next if ($typ ne 'exp');

          if ($src == 0  && $que eq '+' &&
          ($ipflw==100||$ipflw==200||$ipflw==300||$ipflw==400)) {
```

```
                $p{$id}{s}=$tim;
                $p{$id}{i}=$ipflw;
                print DEBUG "id:$id lspid:$ipflw s:$tim\n";
                next
        }
        elsif ($que eq 'r' && $dst ==  11 && exists ($p{$id})) {
                $p{$id}{r}= $tim;
                print DEBUG "id:$id lspid:$ipflw r:$tim\n";
        }
}

close (DATA);

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

$drop0 = 0;
$drop1 = 0;
$drop2 = 0;
$drop3 = 0;

foreach $key(@srt) {
  $transit = $p{$key}{r}-$p{$key}{s};
  $timein= $p{$key}{s};
  if       ($p{$key}{i}== 100 ) {
        if ($p{$key}{r} eq "") {
                $drop0++;
        }
        else {
                $pkt0++;
                $transitsum0 += $transit;
                print SCR0 "$timein $transit \n";
        }
  }
  elsif ($p{$key}{i}== 200 ) {
        if ($p{$key}{r} eq "") {
                 $drop1++;
        }
        else {
                $pkt1++;
                $transitsum1 += $transit;
                print SCR1 "$timein $transit \n";
        }
  }
  elsif ($p{$key}{i}== 300 ) {
        if ($p{$key}{r} eq "") {
                 $drop2++;
        }
        else {
                $pkt2++;
                $transitsum2 += $transit;
                print SCR2 "$timein $transit \n";
        }
  }
  elsif ($p{$key}{i}== 400 ) {
```

```
        if ($p{$key}{r} eq "") {
                $drop3++;
        }
        else {
                $pkt3++;
                $transitsum3 += $transit;
                print SCR3 "$timein $transit \n";
         }
  }



 };

 $at0 = $transitsum0/$pkt0;
 $at1 = $transitsum1/$pkt1;
 $at2 = $transitsum2/$pkt2;
 $at3 = $transitsum3/$pkt3;
 $st0 = $pkt0 + $drop0;
 $st1 = $pkt1 + $drop1;
 $st2 = $pkt2 + $drop2;
 $st3 = $pkt3 + $drop3;
 $lr0 = $drop0/$st0;
 $lr1 = $drop1/$st1;
 $lr2 = $drop2/$st2;
 $lr3 = $drop3/$st3;

$overallat=($transitsum0+$transitsum1+$transitsum2+$transitsum3)/
          ($pkt0+$pkt1+$pkt2+$pkt3);

 ###################################################################
 # Scenario 2
 $btp = $at0 * 1000 + $at1 * 800 + $at2 * 300 + $at3 * 100;
 print
 print      "source 1000k -- sent packet:$st0  dropped packet:$drop0
LostRate:$lr0 average delay:$at0 seconds\n";
 print      "source  800k -- sent packet:$st1  dropped packet:$drop1
LostRate:$lr1 average delay:$at1 seconds\n";
 print      "source  300k -- sent packet:$st2   dropped packet:$drop2
LostRate:$lr2  average delay:$at2 seconds\n";
 print      "source  100k -- sent packet:$st3    dropped packet:$drop3
LostRate:$lr3  average delay:$at3 seconds\n\n";


print " overall average delay = $overallat \n";
```