Setting up an MPLS Network in Lab

CS680 Course Project Report

Submitted in partial fulfillment of the requirements
for the degree of

Master of Technology

by

Rahul Mittal, Seshuraju P. and Election Reddy
Roll No: 07305003,07305028,07305054

under the guidance of

Prof. Anirudha Sahoo

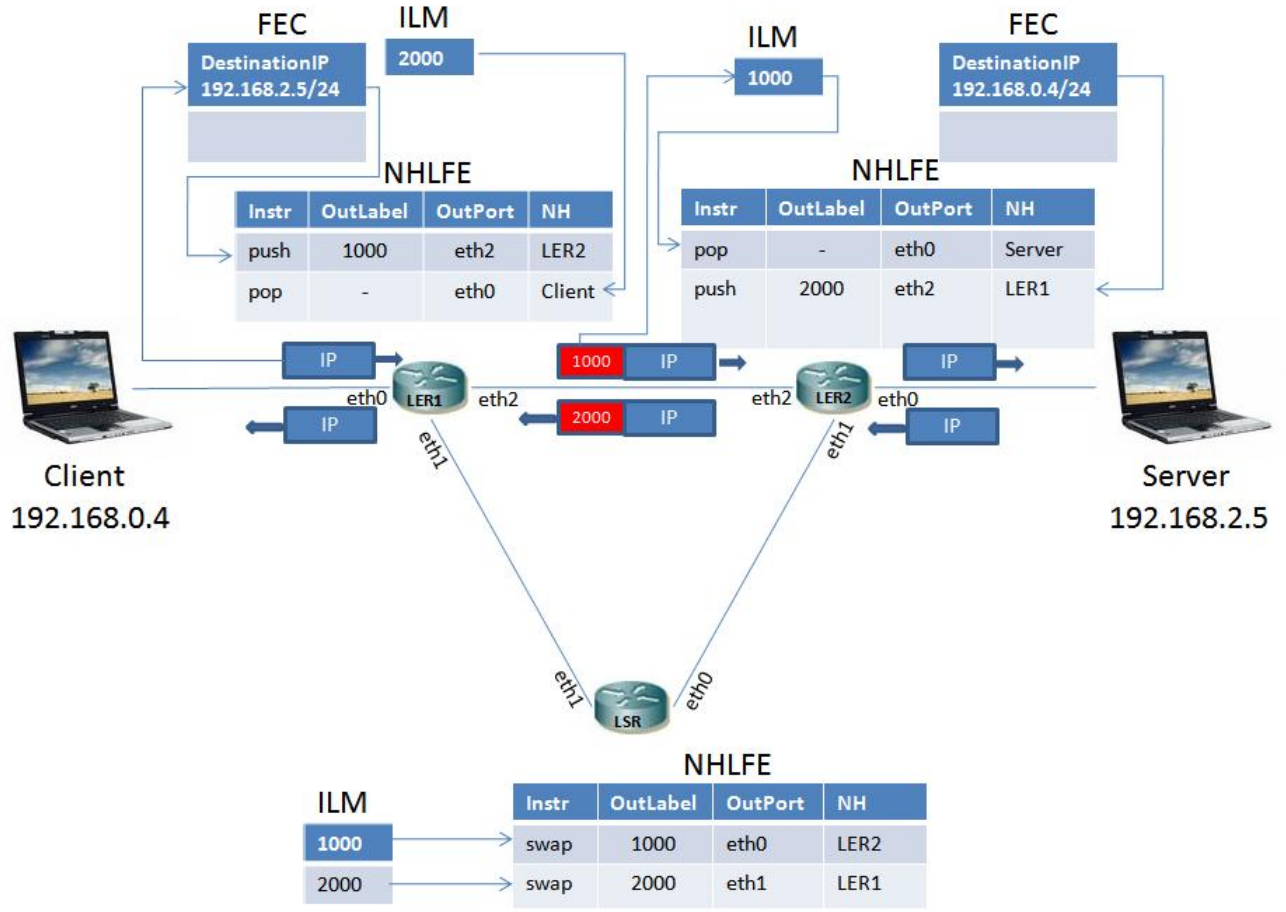
Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai

Contents

1 Introduction

Multiprotocol Label Switching Label Switching (MPLS) is a technology for speeding up network traffic flow and making it easier to manage. MPLS involves setting up a specific path for a given sequence of packets, identified by a label put in each packet, which saves the time needed for a router to look up the address to the next node to forward the packet to. In addition to moving traffic faster overall, MPLS makes it easy to manage a network for quality of service (QoS). MPLS is called multiprotocol because it can be implemented over multiple different networks and protocols of IP network, ATM network. MPLS allows most packets to be forwarded at the layer 2 (switching) level rather than at the layer 3 (routing) level. Infact the MPLS layer is in between the layer 2 and layer 3.

Instead of ip routing the packet, a router in the mpls network uses the label switching table (**LST**) to forward the packet. If a router in mpls network is a edge router and receives a ip packet from other network, it uses the label switching table to map the forwarding equivalence class (**FEC**) to a next hop label forwarding entry (**NHLFE**). The NHLFE contains the outgoing label, the interface through which the packet will be forwarded and the next hop ip address. If a router in the mpls network receives a mpls packet containing a label it maps the incoming label to a NHLFE using the LST which is also called as the incoming label mapping (**ILM**).



2 MPLS Installation

2.1 Requirements

2.1.1 Hardware Requirements

1. Two Computers with three LAN cards each (LERs)
2. One Computer with two LAN (LSR)
3. Two Computers with one LAN card each (one Server other Client)
4. Five Cross Cables

2.1.2 Software Requirements

1. Linux Operating System, Fedora core 4
2. Following Fedora core-4 rpms, available at `http://rpmfind.net/linux/rpm2html/`

   kernel-2.6.18-1.2257.fc4.mpls.1.955.i686.rpm

   kernel-devel-2.6.18-1.2257.fc4.mpls.1.955.i686.rpm

   ebtables-2.0.6-7_mpls_1.950d.i386.rpm

   iproute-2.6.11-1_mpls_1.950d.i386.rpm

   iptables-1.3.0-2_mpls_1.950d.i386.rpm

   iptables-ipv6-1.3.0-2_mpls_1.950d.i386.rpm
3. Tcpdump
4. Wireshark
5. PureFtp Server
6. Ethtool

2.2 Procedure

Perform the following steps for the systems in the mpls network (two LERs and one LSR)

1. Update the yum repository address by adding the iitb ftp server address.
2. Update the yum repository and perform the following installation

   yum update

   yum install udev.i386 071-0.FC4.3

   yum install hardlink

   yum install bridge-utils
3. Install the rpms to update the kernel with mpls patch and various tools

   rpm -ivh kernel-2.6.18-1.2257.fc4.mpls.1.955.i686.rpm

   rpm -ivh kernel-devel-2.6.18-1.2257.fc4.mpls.1.955.i686.rpm

rpm -ivh iproute-2.6.11-1_mpls_1.950d.i386.rpm

rpm -ivh –force iproute-2.6.11-1_mpls_1.950d.i386.rpm

rpm -ivh iptables-1.3.0-2_mpls_1.950d.i386.rpm

rpm -ivh –force iptables-1.3.0-2_mpls_1.950d.i386.rpm

rpm -ivh ebtables-2.0.6-7_mpls_1.950d.i386.rpm

4. Remove the older version of the tools

   rpm -e iptables-1.3.0-2

   rpm -e iproute-2.6.11-1

5. Enable the ip forwarding by editing the /etc/sysctl.conf file

   ipv4.forward=1

6. Install the tcpdump used for debugging.

7. Install the wireshark on host systems

8. Update the routing table of hosts

9. Install and configure a FTP server on the server machine.

2.3 Cautions

1. The kernel version with mpls patch used should be **newer** than the **version** of Fedora Core-4 used.

2. Installation of kernel with mpls patch updates the menu.lst of the **boot loader** and the boot loader shows one more option at the time of booting i.e. to boot with mpls patched kernel. This option will not be there if you compile the kernel from a linux over which some other operating system is installed. This is because system then uses the boot loader of the new operating system, which is not updated to show the new option.

3. While the installation of Fedora, **firewall** should not be selected which stops the forwarding of various request like ssh, http, ftp etc. If already a Fedora is installed, firewall can be checked by using iptables -L command. If firewall is enabled already, it can be flushed using **iptables -F** command.

4. Do not forget to remove the **older versions of iptables and iproute** form the system.

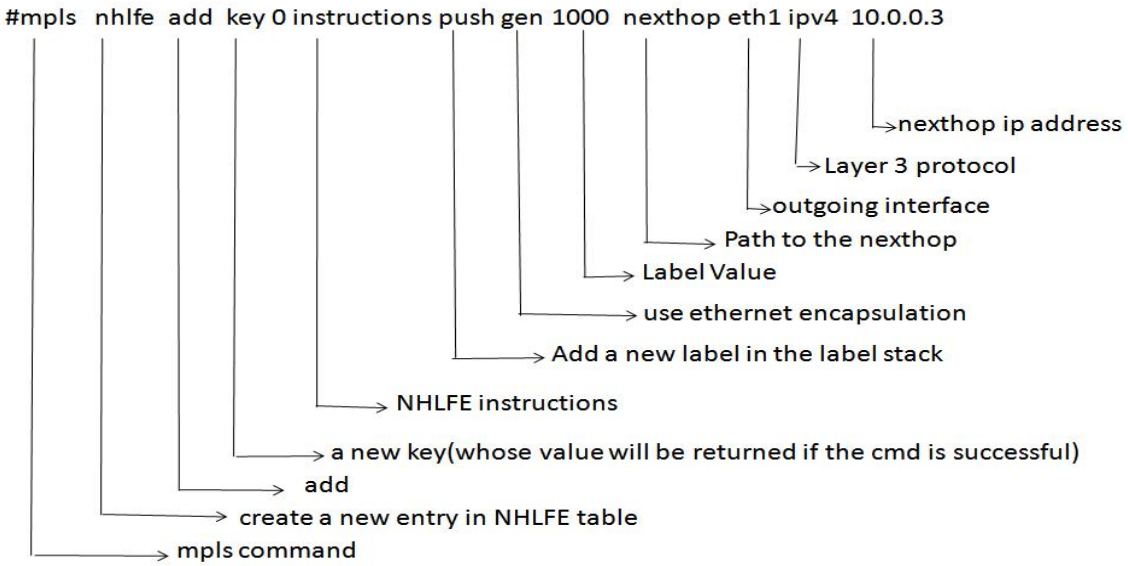
5. Do not forget to ON the **ipforward** option.

3 Commands and Tools

Some basic commands to setup the Label switching path (LSP) in the mpls network and there explanation.

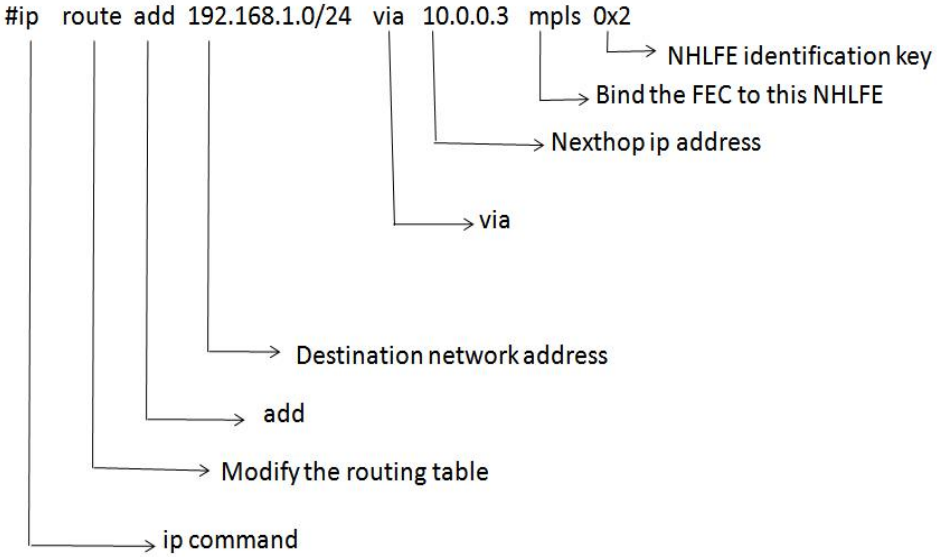
3.1 mpls nhlfe add

This command is executed at the edge routers to create a NHLFE entry to add label 1000 and forward the packets to 10.0.0.3 using outgoing interface eth1.



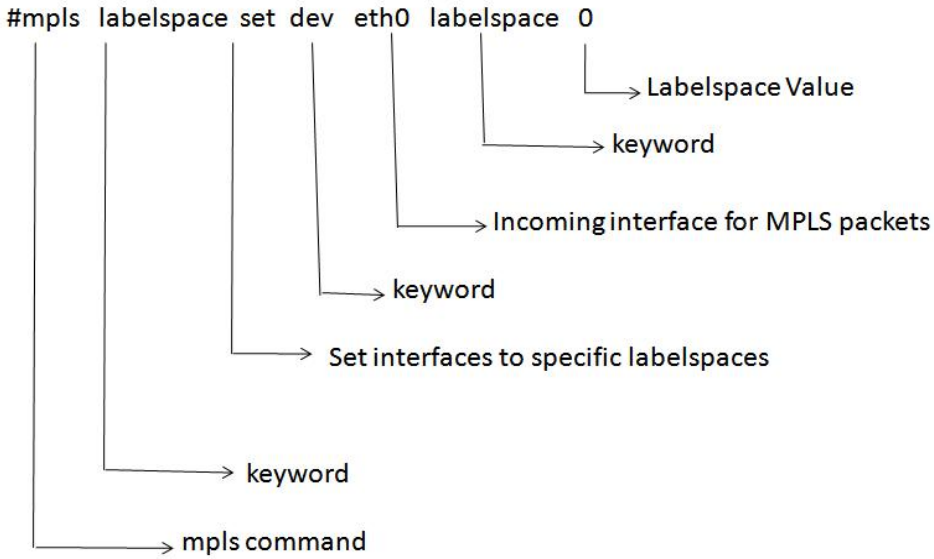
3.2 ip route add

This command is executed at mpls routers to map a FEC to a NHLFE, to actually use the created NHLFE.



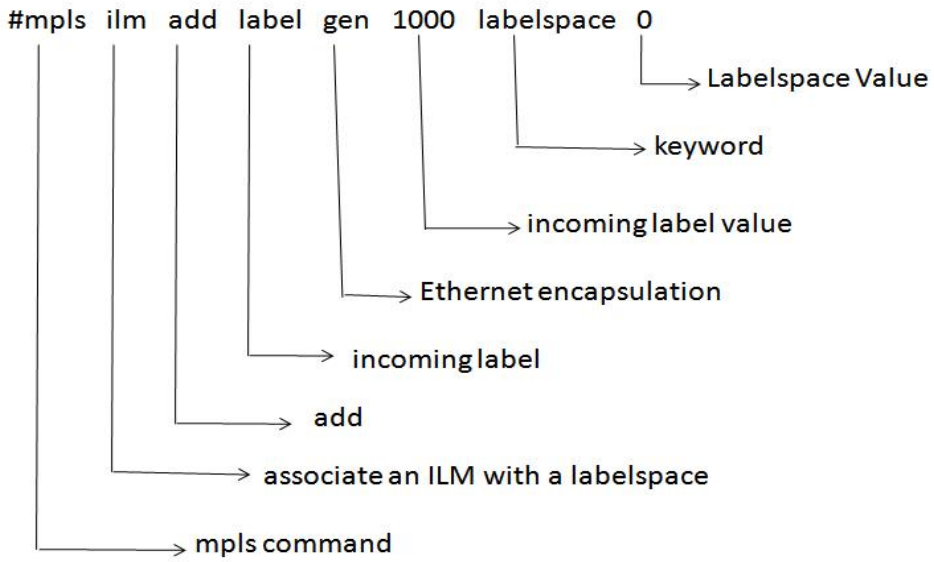
3.3 mpls labelspace

This command is executed at the label switching router or the egress router to set a labelspace so that the router expects MPLS packets through an interface.



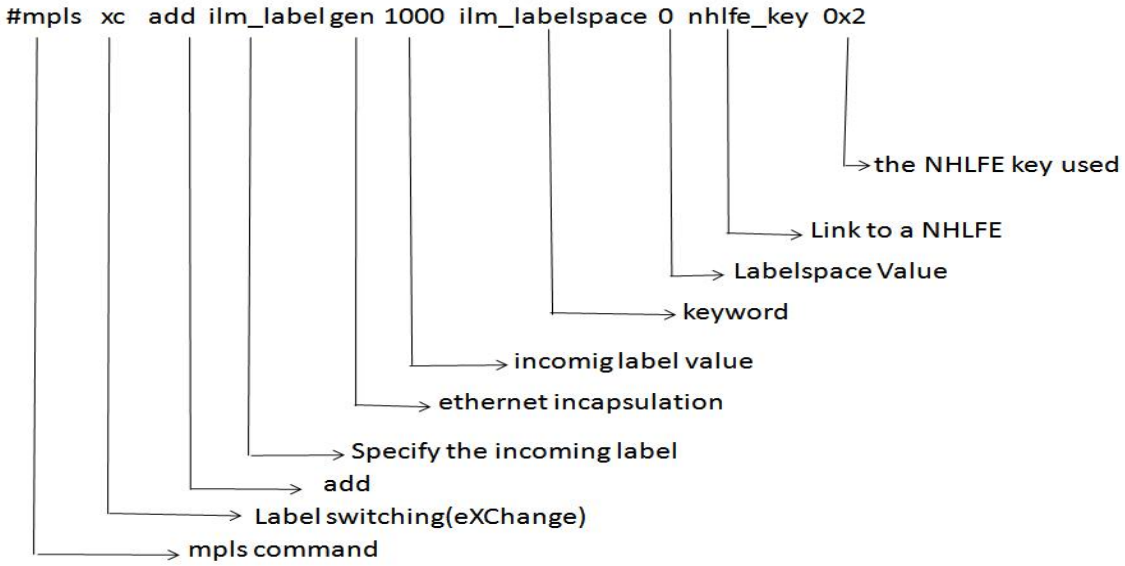
3.4 mpls ilm add

This command is executed at the label switching router or the egress router to add an entry to the ILM table to list an expected label.



3.5 mpls xc add

Command to do the actual switching.



3.6 tcpdump

tcpdump command is used on the mpls routers for all the interfaces they are having. One tcpdump command for on interface on a router. tcpdump displays anything that comes to that interface. This is useful when we what to know the label of the packet, source and destination ip address, protocol id, ttl etc. Even if the packet is rejected at any layer because of any reasons like failing CRC, the content will be displayed on the screen.

```
tcpdump -nv -i eth0
```

The option nv in the command displays the label of the packet due to mpls on the screen. The i option is specify which interface it is meant for.

3.7 iptables

iptables is used to filter the type of traffic and transfer them through different path or interface.

```
iptables -A OUTPUT -d 192.168.4.5 -p tcp --dport 80 -j mpls --nhlfe $key
```

Running this command will route any packet of a tcp traffic coming on port 80 destined for 192.168.4.5 as specified by the mpls NHLFE having key as $key.

3.8 ethtool

This command gives us the details of the settings of ethernet card. We can also know using this command whether the link on this interface is active or inactive.

```
ethtool eth0
```

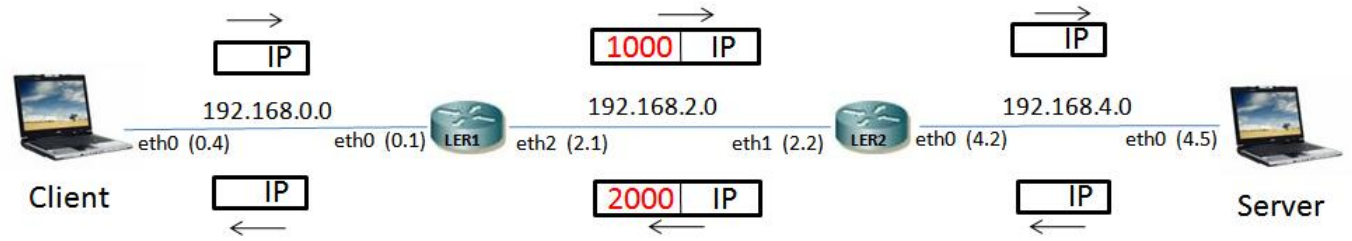
This command displays the setting of the eth0 interface. This displays a *link detected* flag as yes or no which is used by us in experiment 6.

4 Experiments and Results

4.1 Experiment 1

4.1.1 Scenario

The fig shows the experimental setup. This experiment is the most basic one in which we have just two routers in the mpls network and we ping from server to client and interpret the result.



4.1.2 Result and Explanation

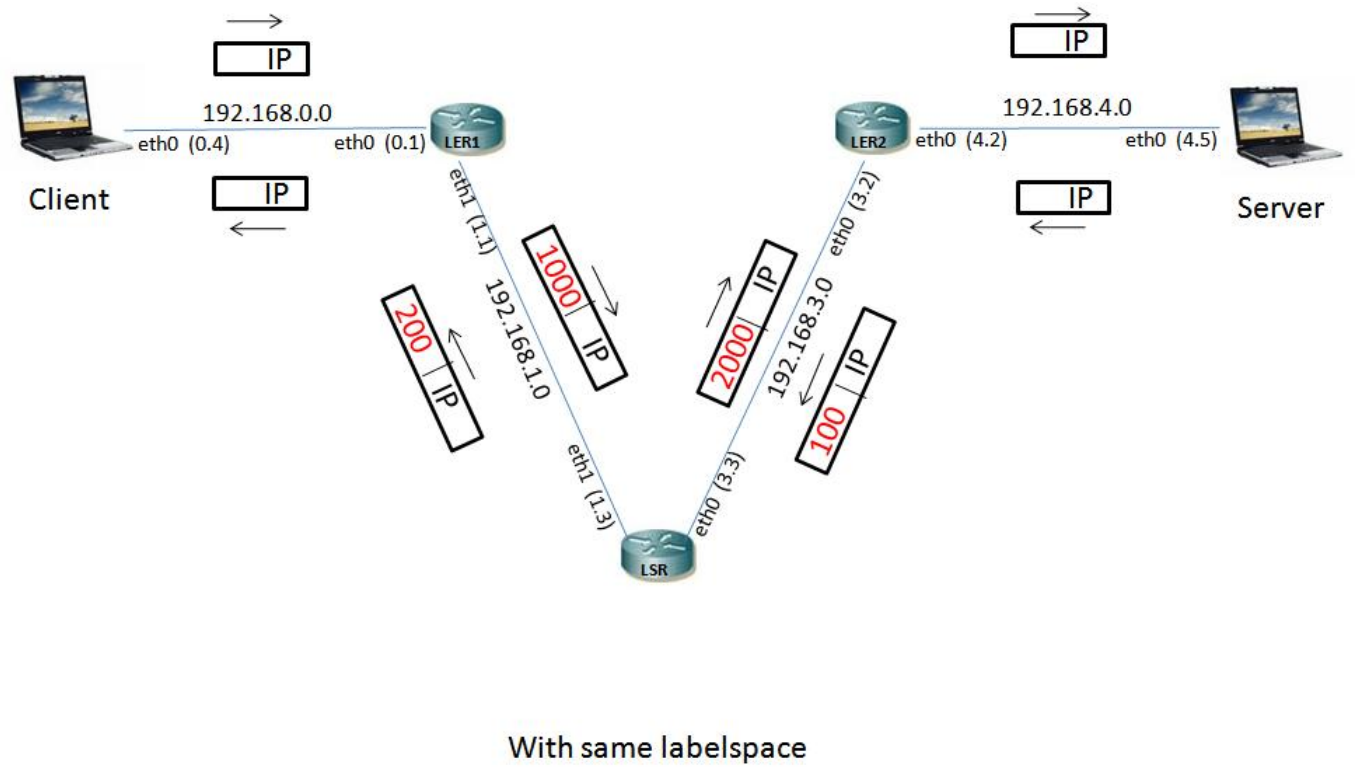
```
rahul@SUPERCOMPUTER:~$ ping -R 192.168.0.4
PING 192.168.0.4 (192.168.0.4) 56(124) bytes of data.
64 bytes from 192.168.0.4: icmp_seq=1 ttl=63 time=5.41 ms
RR:     192.168.4.5
        192.168.2.2
        192.168.0.4
        192.168.0.4
        192.168.2.1
        192.168.4.5

64 bytes from 192.168.0.4: icmp_seq=2 ttl=63 time=0.505 ms      (same route)
64 bytes from 192.168.0.4: icmp_seq=3 ttl=63 time=1.62 ms       (same route)
64 bytes from 192.168.0.4: icmp_seq=4 ttl=63 time=0.452 ms      (same route)
64 bytes from 192.168.0.4: icmp_seq=5 ttl=63 time=1.55 ms       (same route)
64 bytes from 192.168.0.4: icmp_seq=6 ttl=63 time=1.53 ms       (same route)
64 bytes from 192.168.0.4: icmp_seq=7 ttl=63 time=0.553 ms      (same route)
64 bytes from 192.168.0.4: icmp_seq=8 ttl=63 time=1.64 ms       (same route)

--- 192.168.0.4 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 6999ms
rtt min/avg/max/mdev = 0.452/1.660/5.418/1.507 ms
```

We have noticed here that when we `ping -R` from server to the client we do not find addresses of all the interfaces through which the ping message goes. This is because the ping only shows the addresses of the interfaces following the ip routing and in our network some interfaces are following mpls routing, so there addresses are not displayed.

4.2 Experiment 2

4.2.1 Scenario

The fig. shows the experimental setup having three nodes in the mpls network. It's a linear topology having two LERs and a LSR. We used the ping, ssh, scp and FTP traffic.



4.2.2 Result and Explanation

Successful transmission of all traffic in this scenario, but the LSR here needs to have two different label mapping for request and response. The next experiment solves this. Round Trip Time measured using *ping -R*.

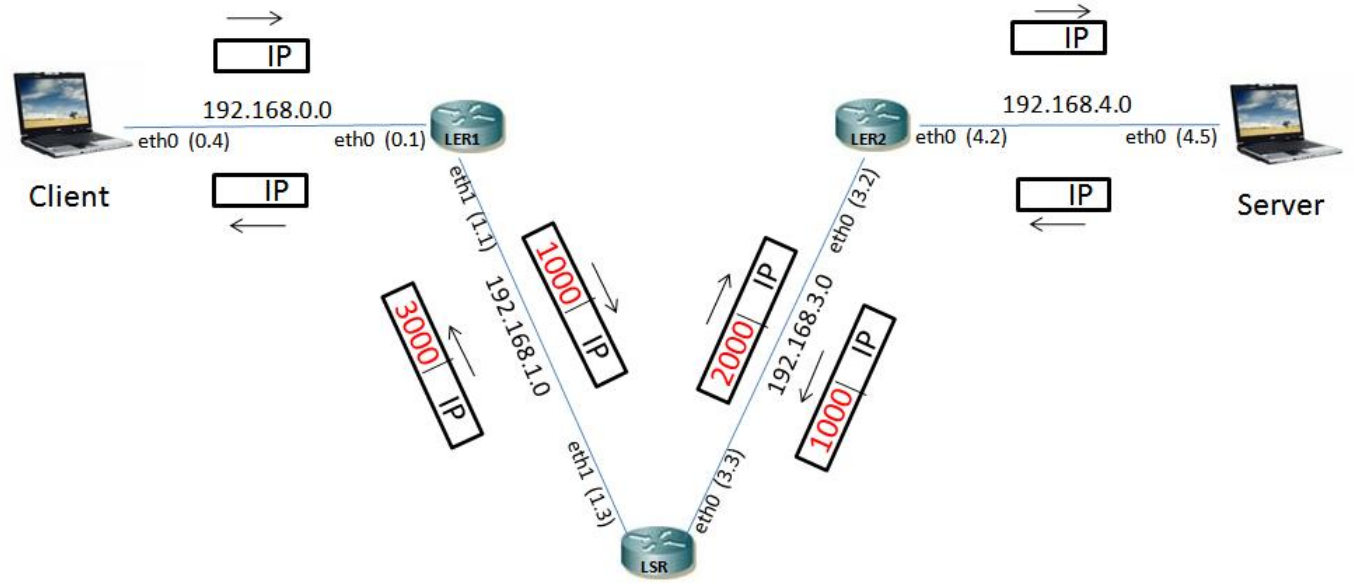
```
root@SUPERCOMPUTER:~# ping -R 192.168.0.4
PING 192.168.0.4 (192.168.0.4) 56(124) bytes of data.
64 bytes from 192.168.0.4: icmp_seq=1 ttl=63 time=3.87 ms
RR:     192.168.4.5
        192.168.3.2
        192.168.0.4
        192.168.0.4
        192.168.1.1
        192.168.4.5

64 bytes from 192.168.0.4: icmp_seq=2 ttl=63 time=0.639 ms      (same route)
64 bytes from 192.168.0.4: icmp_seq=3 ttl=63 time=3.88 ms       (same route)

--- 192.168.0.4 ping statistics ---
```

```
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 0.639/2.798/3.881/1.528 ms
```

4.3 Experiment 3

4.3.1 Scenario

Experimental setup is the same as the previous experiment. But here the LSR uses the same label mapping which can be seen from the fig.



With different labelspace

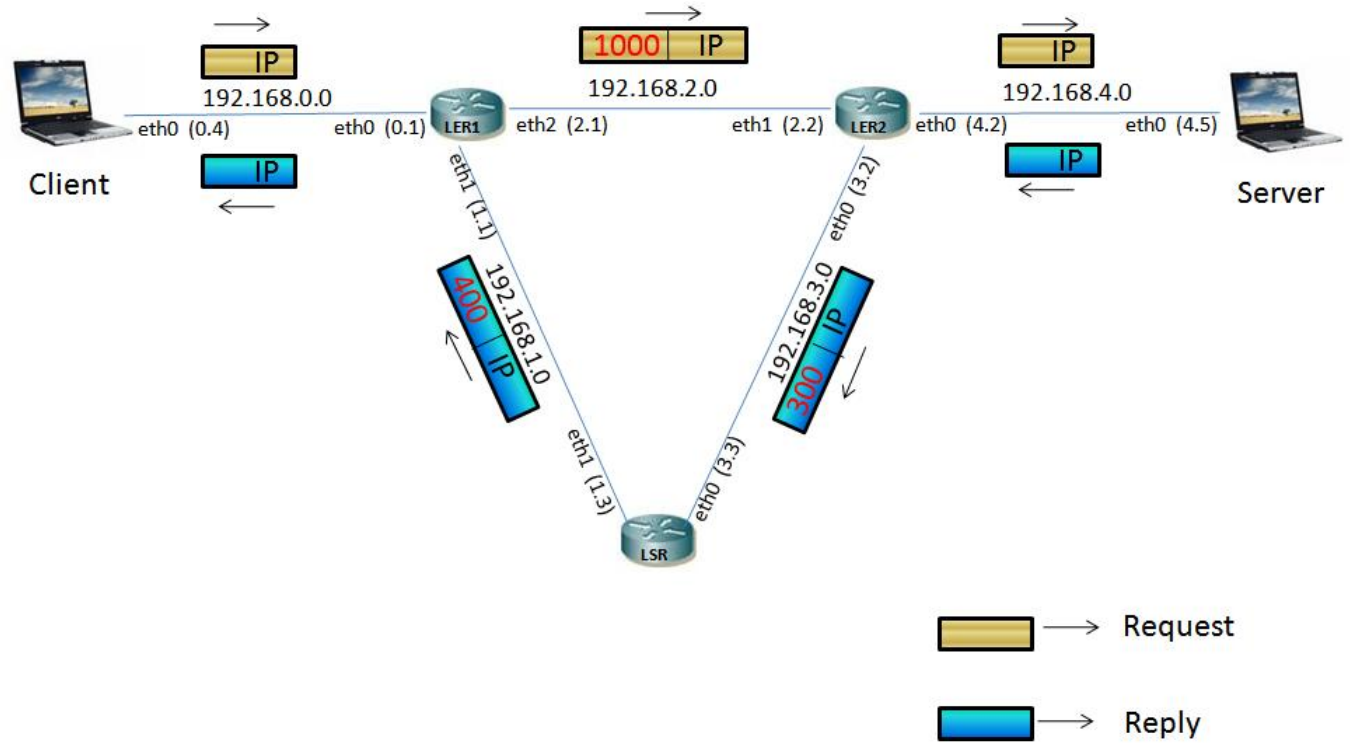
4.3.2 Result and Explanation

The difference here is that we used two different label spaces. Different label spaces can have same label mapping which enabled the LSR to have same label mapping for both request and response.

4.4 Experiment 4

4.4.1 Scenario

The fig. shows the experimental setup. Here we have three systems in the mpls network. But the difference is that, here we have two different paths through the network. Here we make the request to travel through on path and the response to come from the other path and check if there is any problem.



4.4.2 Result and Explanation

We were able to perform the experiment successfully and verified the path using *ping -R*.

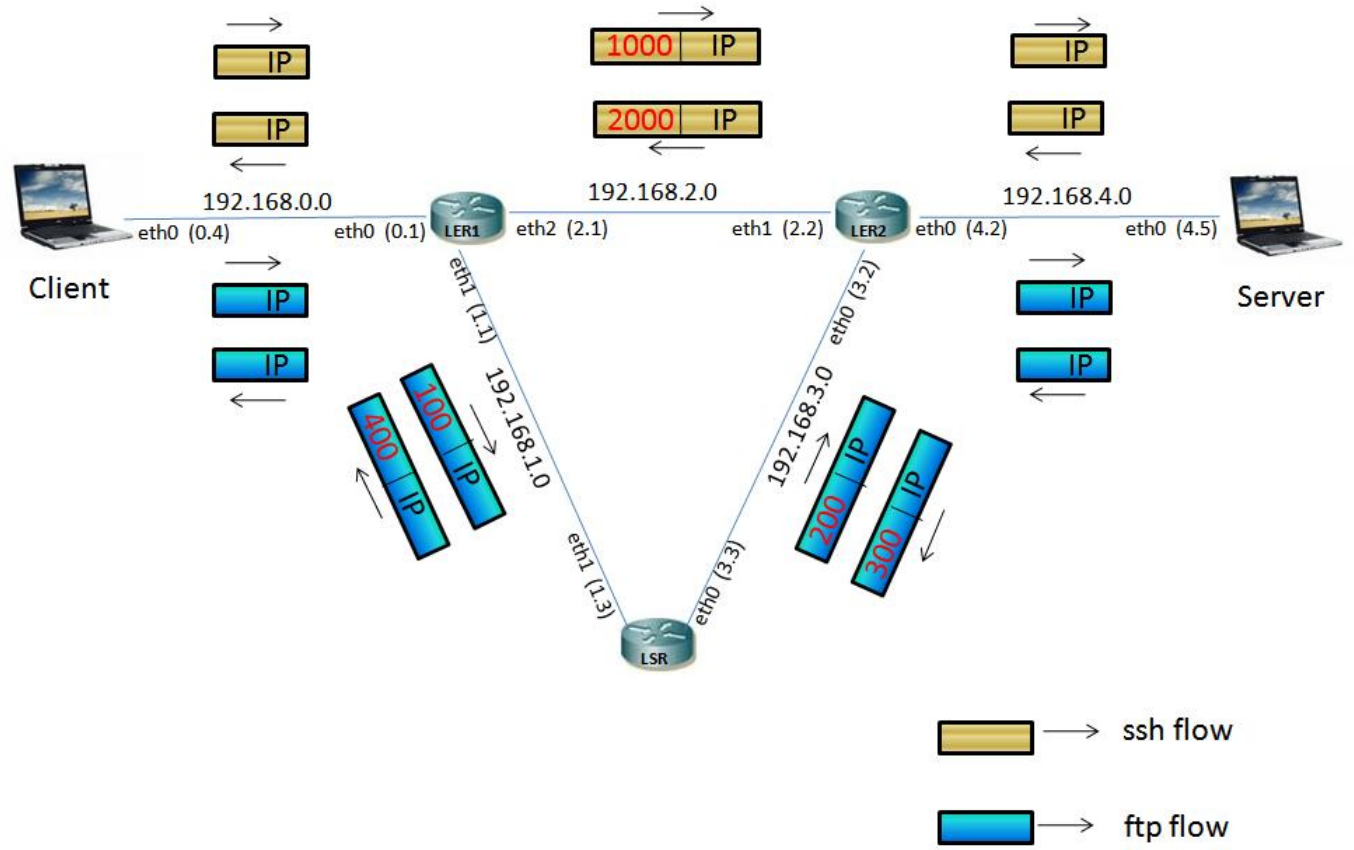
```
root@SUPERCOMPUTER:~# ping -R 192.168.0.4
PING 192.168.0.4 (192.168.0.4) 56(124) bytes of data.
64 bytes from 192.168.0.4: icmp_seq=1 ttl=63 time=2.50 ms
RR:     192.168.4.5
        192.168.3.2
        192.168.0.4
        192.168.0.4
        192.168.2.1
        192.168.4.5

64 bytes from 192.168.0.4: icmp_seq=2 ttl=63 time=1.73 ms       (same route)
64 bytes from 192.168.0.4: icmp_seq=3 ttl=63 time=0.683 ms      (same route)

--- 192.168.0.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
```

```
rtt min/avg/max/mdev = 0.683/1.638/2.502/0.746 ms
```

4.5 Experiment 5

4.5.1 Scenario

The fig. shows the experimental setup. Here we make use of the iptables to filter different types of traffics and route them through different links.



4.5.2 Result and Explanation

In this experiment using the iptables command we filtered different types of incoming traffic (different incoming ports) on the node and mapped them to different NHLFE. But this resulted in having multiple entries in the routing table (*ip route* and *route -n*) of the node. This resulted in all the traffic to be routed using the first entry of the destination in the routing table.

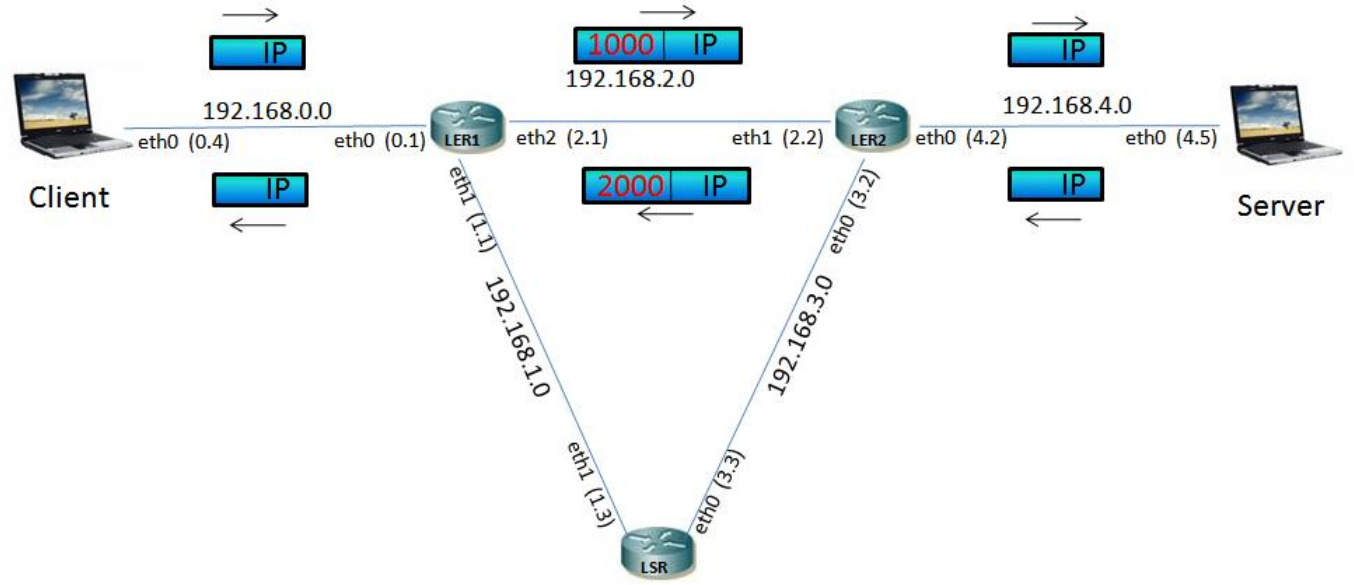
```
[root@localhost Ler1]# ip route
192.168.4.5 via 192.168.2.2 dev eth2 mpls 0x3f
192.168.4.5 via 192.168.1.3 dev eth1 mpls 0x41
192.168.2.0/24 dev eth2  proto kernel  scope link  src 192.168.2.1
192.168.1.0/24 dev eth1  proto kernel  scope link  src 192.168.1.1
192.168.0.0/24 dev eth0  proto kernel  scope link  src 192.168.0.1
[root@localhost Ler1]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
```

```
192.168.4.5     192.168.2.2     255.255.255.255 UGH   0      0        0 eth2
192.168.4.5     192.168.1.3     255.255.255.255 UGH   0      0        0 eth1
192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 eth2
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 eth1
192.168.0.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
```

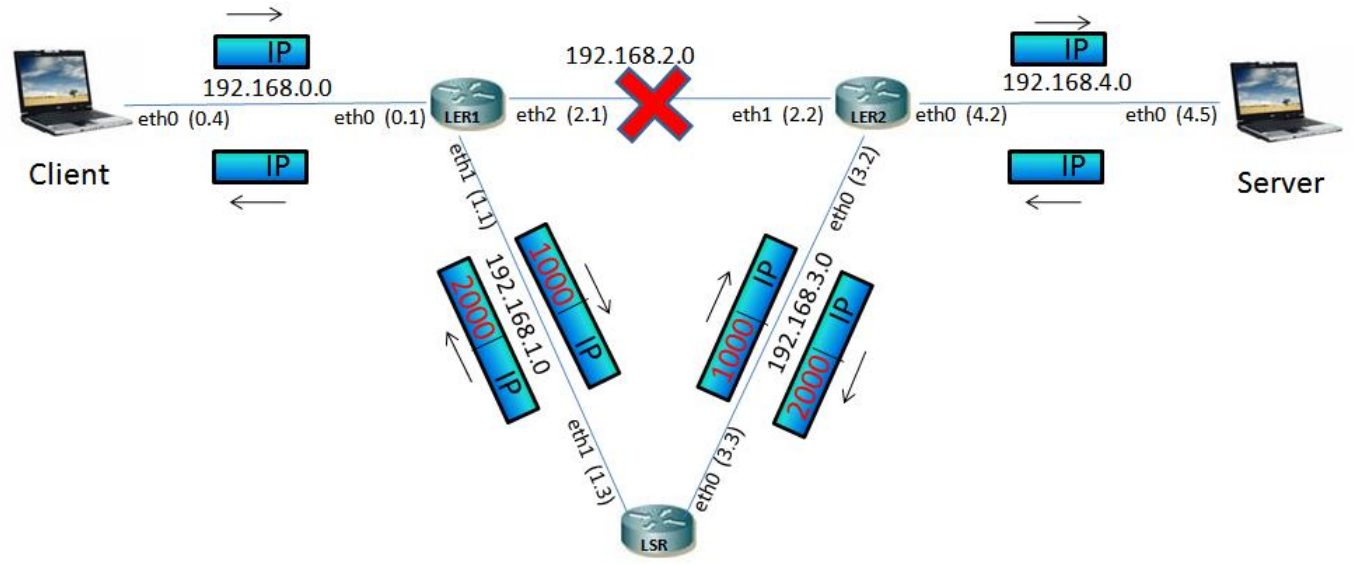
4.6 Experiment 6

4.6.1 Scenario

The fig. shows the experimental setup. There are two path available in the mpls network to route the data from client to server. In the event of failure of one path, it automatically routes the packets from the alternate path.



Before Link Failure



After Link Failure

4.6.2 Result and Explanation

```
root@SUPERCOMPUTER:~# ping -R 192.168.0.4
PING 192.168.0.4 (192.168.0.4) 56(124) bytes of data.
64 bytes from 192.168.0.4: icmp_seq=1 ttl=63 time=11.5 ms
RR:     192.168.4.5
        192.168.2.2
        192.168.0.4
        192.168.0.4
        192.168.2.1
        192.168.4.5

64 bytes from 192.168.0.4: icmp_seq=2 ttl=63 time=0.517 ms      (same route)
64 bytes from 192.168.0.4: icmp_seq=4 ttl=63 time=10.7 ms
RR:     192.168.4.5
        192.168.3.2
        192.168.0.4
        192.168.0.4
        192.168.1.1
        192.168.4.5

64 bytes from 192.168.0.4: icmp_seq=5 ttl=63 time=1.72 ms       (same route)
64 bytes from 192.168.0.4: icmp_seq=10 ttl=63 time=1.70 ms      (same route)
64 bytes from 192.168.0.4: icmp_seq=11 ttl=63 time=2.80 ms
RR:     192.168.4.5
        192.168.3.2
        192.168.0.4
        192.168.0.4
        192.168.2.1
        192.168.4.5

64 bytes from 192.168.0.4: icmp_seq=12 ttl=63 time=4.13 ms
RR:     192.168.4.5
        192.168.2.2
        192.168.0.4
        192.168.0.4
        192.168.2.1
        192.168.4.5

64 bytes from 192.168.0.4: icmp_seq=13 ttl=63 time=1.60 ms      (same route)
64 bytes from 192.168.0.4: icmp_seq=18 ttl=63 time=0.542 ms     (same route)

--- 192.168.0.4 ping statistics ---
18 packets transmitted, 17 received, 5% packet loss, time 16999ms
rtt min/avg/max/mdev = 0.517/2.856/11.548/3.173 ms
```

We verified every 0.5sec whether the link is active or not in our shell script. If the link becomes inactive the the incoming packets are mapped to different NHLFE. If the original link is active the packets go through original link else takes the alternate path till the original link is up. Ethtool command is used to check whether a link is active or inactive.

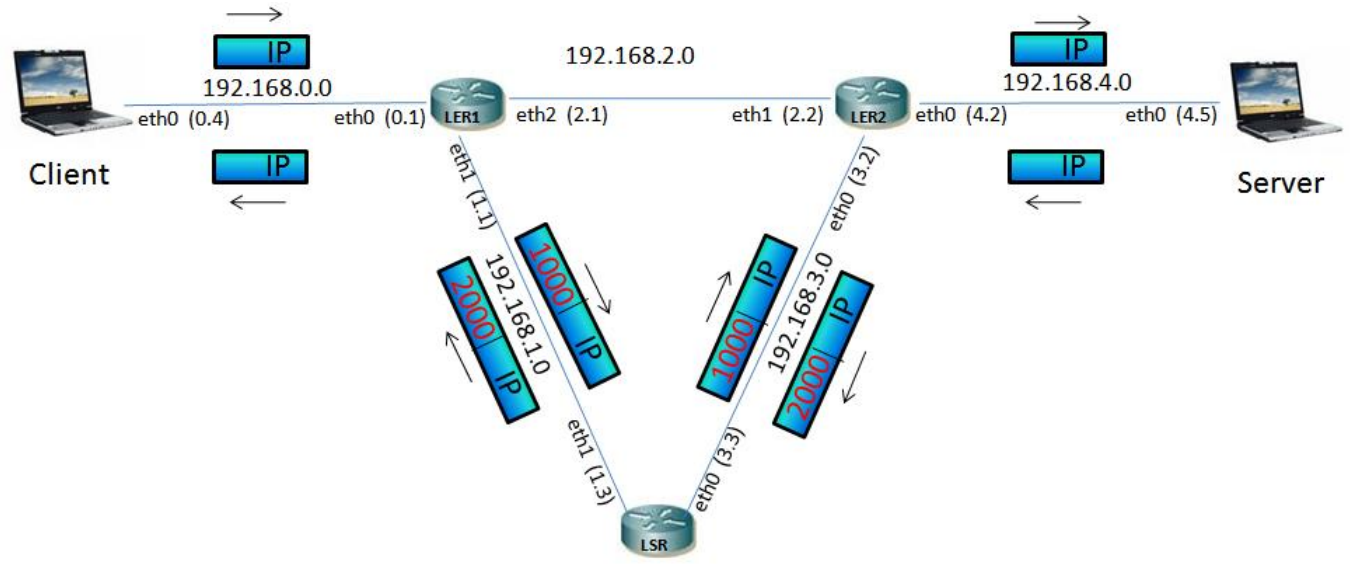
```
ethtool eth0
```

It displays the setting of the ethernet card. Through it we extract the link detected flag whether it is yes or no.

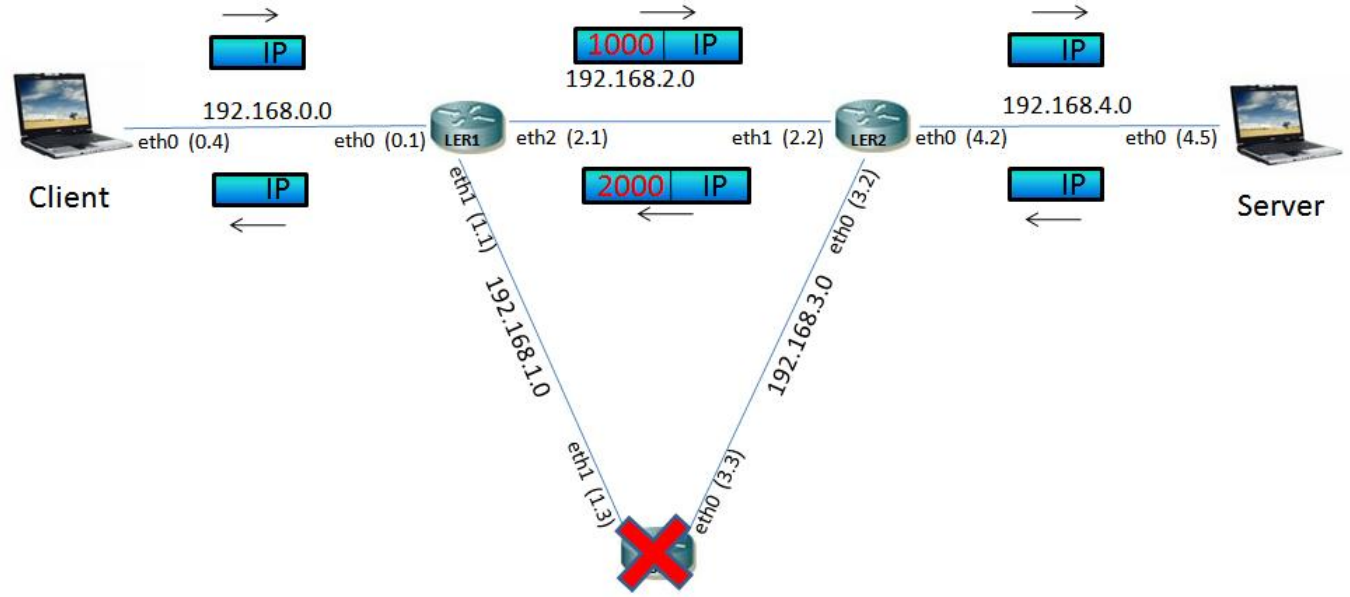
4.7 Experiment 7

4.7.1 Scenario

The fig.shows the experimental setup. On the event of a node failure in on of the routes in the mpls network, it shifts to alternate path to route the packets.



Before Node Failure



After Node Failure

4.7.2 Result and Explanation

```
root@SUPERCOMPUTER:~# ping -R 192.168.0.4
PING 192.168.0.4 (192.168.0.4) 56(124) bytes of data.
64 bytes from 192.168.0.4: icmp_seq=1 ttl=63 time=4.73 ms
RR:     192.168.4.5
        192.168.3.2
        192.168.0.4
        192.168.0.4
        192.168.1.1
        192.168.4.5

64 bytes from 192.168.0.4: icmp_seq=2 ttl=63 time=1.77 ms       (same route)
64 bytes from 192.168.0.4: icmp_seq=21 ttl=63 time=0.785 ms
RR:     192.168.4.5
        192.168.2.2
        192.168.0.4
        192.168.0.4
        192.168.2.1
        192.168.4.5

64 bytes from 192.168.0.4: icmp_seq=22 ttl=63 time=1.65 ms      (same route)
. . . . .
64 bytes from 192.168.0.4: icmp_seq=34 ttl=63 time=1.64 ms      (same route)
64 bytes from 192.168.0.4: icmp_seq=35 ttl=63 time=0.549 ms
RR:     192.168.4.5
        192.168.3.2
        192.168.0.4
        192.168.0.4
        192.168.2.1
        192.168.4.5

64 bytes from 192.168.0.4: icmp_seq=36 ttl=63 time=1.64 ms
RR:     192.168.4.5
        192.168.3.2
        192.168.0.4
        192.168.0.4
        192.168.1.1
        192.168.4.5

64 bytes from 192.168.0.4: icmp_seq=37 ttl=63 time=0.620 ms     (same route)
64 bytes from 192.168.0.4: icmp_seq=40 ttl=63 time=1.76 ms      (same route)

--- 192.168.0.4 ping statistics ---
40 packets transmitted, 22 received, 45% packet loss, time 38997ms
rtt min/avg/max/mdev = 0.486/1.316/4.731/0.893 ms
```

First the packets follow initial path then we make the two interfaces of the middle node down using

```
ifconfig eth0 down
ifconfig eth1 down
```

this gets the the packet to follow different path. Then again we get the middle node up to get the packets follow the original path.
We verified every 0.5sec whether the node is active or not using ping to that node in our shell script. If the node is not active the packets takes alternate path till the node is active again.
Lesser throughput is seen with this experiment in comparison to experiment 6 because ethtool is faster then manually pinging a dead node.

5 Comparison of Experiments and Conclusion

In this section we compare the results obtained in experiment 6 and 7.
We got less throughput with the experiment 7 (node failure) then with experiment 6 (link failure) beacuse it takes more time to detect a node failure then to detect a link failure. This is beacuse the node failure detection is done using ping command which waits for the acknowlegement and the the link failure is detected using the *ethtool* which is much faster then manually pinging a dead node.

Secondly, in experiment 2 (linear topology four hop) we measured more avg. round trip delay (rtt) then in experiment 4 (data and ack different path) because in experiment 2, both ack and data follows the same path which is the shortest path.

6 General problems faced and Causes of Time wastage

1. Initially very less number of system could comply with the requirements.
2. Some system had no working **graphical interface**.
3. Initially all the system contain **ubuntu version 5**.
4. Not all the **LAN cards** were working.
5. System were not having **DVD players** on them, required for installation of FEDORA 4.
6. Finally all the **working LAN cards** were **collected** and three system were identified to full fill the requirement.
7. A single **external DVD player** available in the networks lab was used for installation of FEDORA 4 on three systems.
8. Installation of operating system by **other group** did cost us installation of operating system multiple times.
9. Getting only three systems working did not allow us to work with **tunneling** and **label merging**.
10. **Using DVD** in the CD player did cost us writing new DVD and some time.

11. **Firewall** in linux was the major reason of loosing time.

12. **Time sharing** the test bed was a issue at peak time.

7 Conclusion

8 Future Work

1. Implementation of mpls using the computers having **ubuntu** installed on it using the deb packages.

2. Implementation of mpls network having more number of alternate path, and the ingress nodes selects the mpls appropriate path in case of any link or node failure.

3. Implementation of an algorithm like Label Distribution Protocol (LDP) to have the nhlfe in the label switching table by a protocol and not manually.

4. Implement of some features like label merging and tunneling.

5. Implementation of some link state monitor to know about the link failure as early as possible.

6. Experiments to perform the comparison of ip routing and label switching. To know the scenarios where ip routing is better than label switching and vice-versa.

7. Implementation of MPLS over ATM network.

9 References

1. `http://rpmfind.net/linux/rpm2html/` for various rpms.

2. `http://sourceforge.net/projects/mpls-linux/` for mpls documentation and command reference.

3. *Internet QoS: Architectures and Mechanisms for Quality of Service* by Zheng Wang.

4. `https://help.ubuntu.com/community/PureFTP` for the configuration of pureftp.

5. `http://mpls-linux.sourceforge.net/` for installation.