**SAN JOSE STATE UNIVERSITY**

**dEPARTMENT OF ELECTRICAL ENGINEERING**

**SPRING 2016**

**EE283 - Broadband Communication Networking**

**PROJECT ON SIMULATION OF LABEL-SWITCHING NETWORKS USING NS3 TOOL**

**SUPERVISOR: Prof. NADER MIR**

**TEAM MEMBERS:**

**PALAK JOSHI – 010755199 (EE283-02)**

**INDUJA MARAMRAJ – 010756980 (EE283-01)**

**TABLE OF CONTENTS**

1. Objective ………………………………………………………………………..1
2. Abstract …………………………………………………………………………1
3. Introduction ……………………………………………………………………..1
4. Procedure ……………………………………………………………………….2
5. Working and Simulation of the program ……………………………………….6
6. Program code with explanation ……………………………………………….. 7
7. Output ………………………………………………………………………….11
8. Conclusion ……………………………………………………………………..22

**LIST OF FIGURES**

Fig. 1 Topology of the given network

Fig. 2 Ubuntu install

Fig. 3 python, mercurial install

Fig. 4 ns-3-allinone file download

Fig. 5 ns-3-allinone scripts

Fig. 6 MPLS module install

Fig. 7 war command use

Fig. 8 simulation of the program indicating build successfully

Fig. 9 Node0 Interface

Fig. 10 Node1 Interface

Fig. 11 Node2 Interface

Fig. 12 Node3 Interface

Fig. 13 Node4 Interface

Fig. 14 Swapping of label at node0 and node1(ingress LSR1 and LSR2)

Fig. 15 Swapping of label at node2 (core LSR3)

Fig. 16 Swapping of label at node3 (core LSR4)

Fig. 17 pop action at node4 (Egress LSR5)

**OBJECTIVE**

The objectives of this project are:

* To learn a widely-used computer communication network simulation tool, NS3
* To understand how to simulate computer networks learned in class with case study of label switching networks

**ABSTRACT**

This project aims at understanding how label switching works in MPLS i.e., how different labels are being assigned to Label Switched Routers and how the swapping of the labels takes place in an MPLS based environment. A tool called NS3 (Network Simulation tool) is being used for configuring the routers to support MPLS and observing the switching of routers using labels. A sample C++ code is run on the Ubuntu Linux Operating System to analyze a network, assign labels and study the switching.

**INTRODUCTION**

Multi-Protocol Label Switching (MPLS) is a technology that is used to integrate layer 2 and layer 3 information of a network in an autonomous system. It makes use of short labels instead of long addresses for mapping. The flexibility of MPLS is observed while diverting and routing traffic during congestion and link failures, which improves user experience. It is due to this efficiency that many companies are now opting for MPLS networks.

MPLS relies on assigning labels to the different Label Switched Routers (LSR) in a predetermined Label Switched Path (LSP). LSRs forward incoming packets by switching using assigned labels and following an LSP that leads to the destination. Each of the LSRs have the necessary forwarding data in their Forwarding Information Base (FIB).

Here a network with five LSRs of which two are Ingress LSRs, two are core LSRs and one Egress LSR is assumed. Topology of the network is shown below



*Figure 1*

Each of the LSRs are assigned different labels for different paths based on which the packets are routed to their destination. All of the routing is done at LSRs using the push, pop and swap operations. The swap operation is used for switching LSRs. The top most label in the label stack is replaced with another label while the packet is being forwarded. The push operation is used to add a new label to the label stack and it will keep the newly added label at the top of the stack. The pop operation is used to remove the top most label in the label stack.

The NS3 and MPLS module is installed on a Linux OS to simulate the given network. Then, the program is run and the output is analyzed. The FIB concepts such as NHLFE, ILM and FTN are studied.

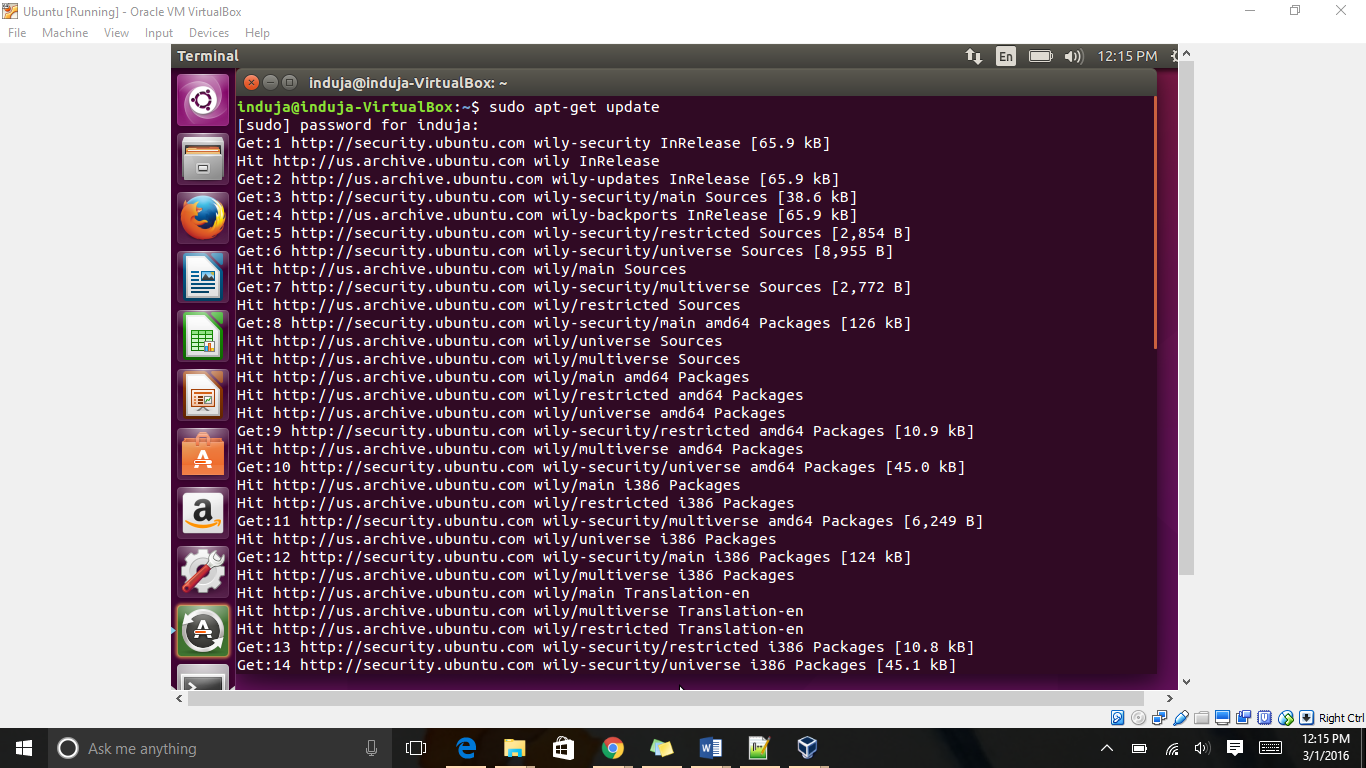
Through this project, the given Label Switched Networks are studied and different labels for the given Label Switched Routers (LSRs) are assigned to accordingly perform the Label Operations. The Label Switched Network is simulated and the structure of the simulation program has been studied.

.

**PROCEDURE**

The initial steps to follow to observe the simulation of the given Label Switched Network are

1. The Ubuntu Linux Operating System is first installed an updated.  
   The command used for updating is *sudo apt-get update*



*Figure 2*

1. Python, Mercurial and Build-essential are then installed

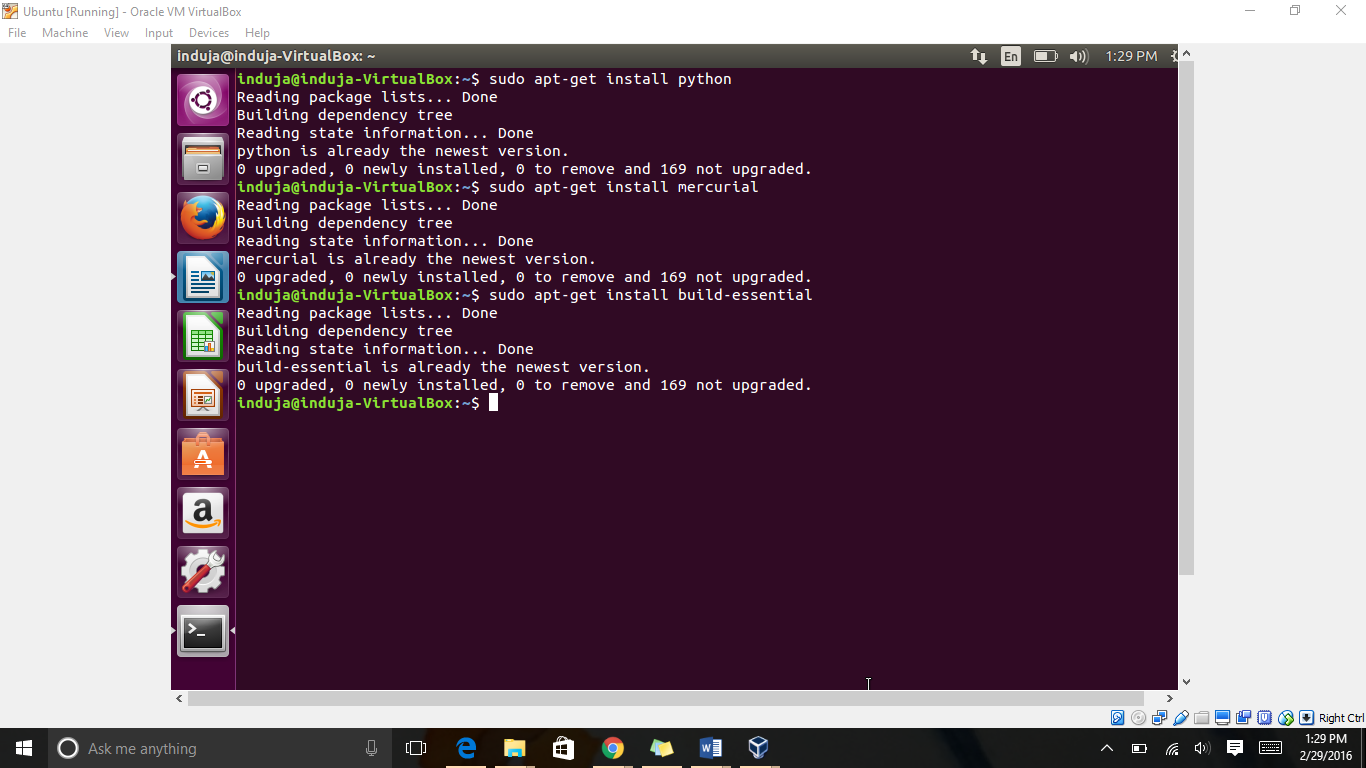
Python is used to execute the sample code, Mercurial for managing the source code and build-essential to build the library files.

The commands used for the installation on Python, Mercurial and Build-essential respectively are:

*sudo apt-get install python*

*sudo apt-get install mercurial*

*sudo apt-get install build-essential*

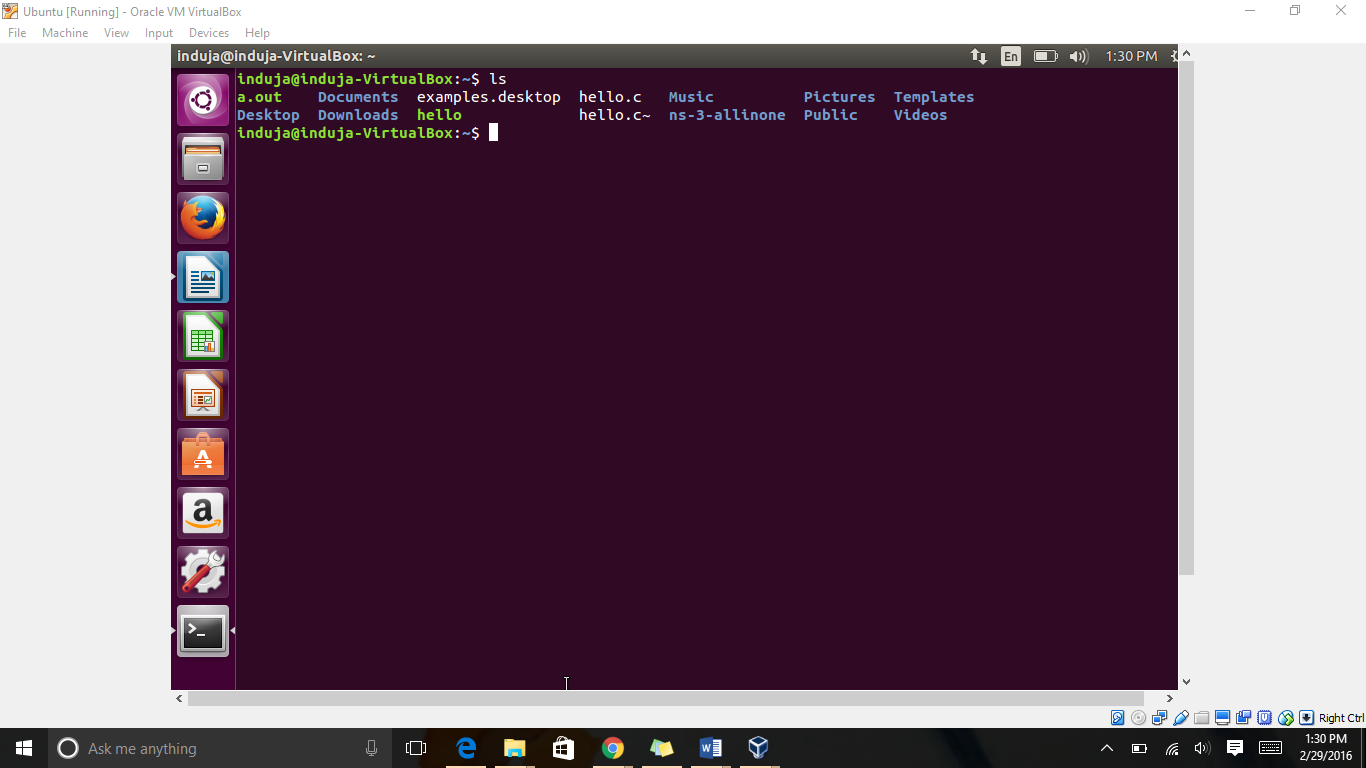
**

*Figure 3*

1. The following command is then typed to get NS3

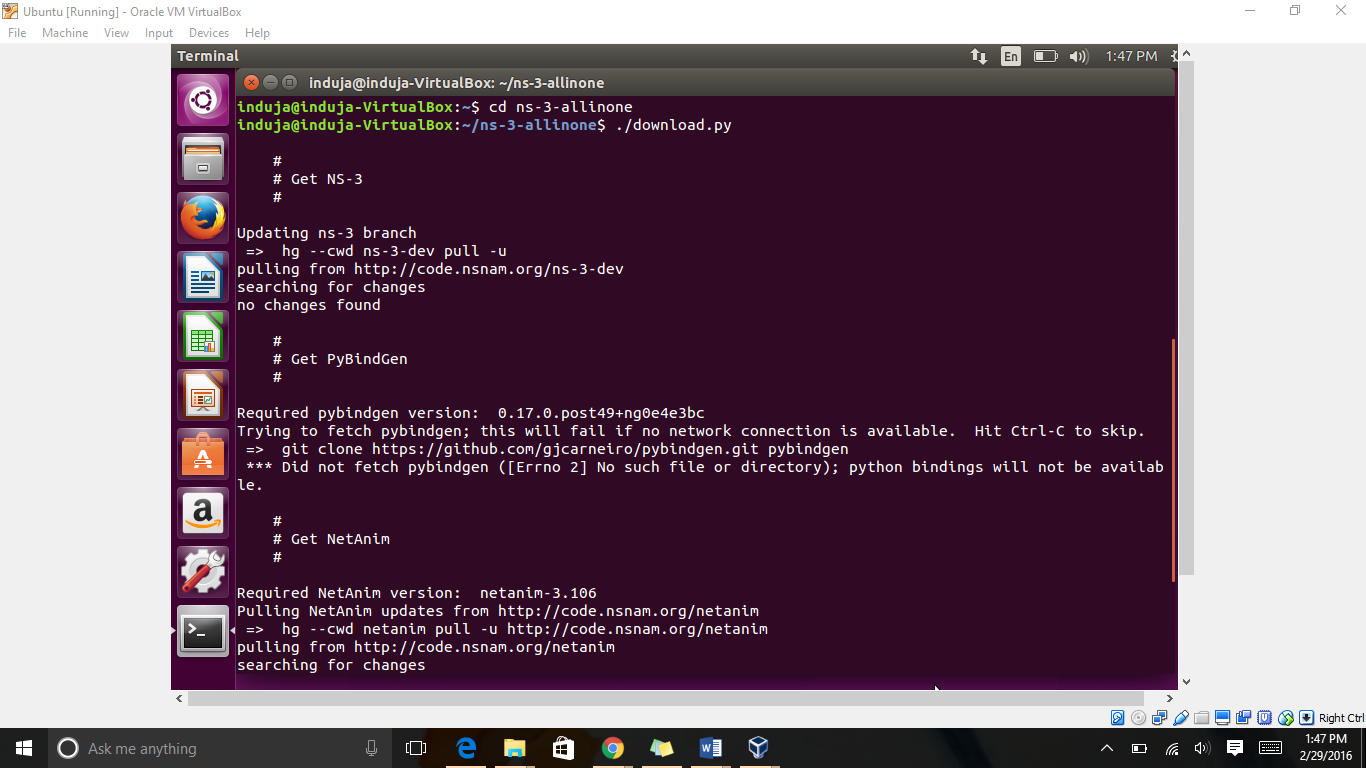
*hg clone* [*http://code.nsam.org/ns-3-allinone*](http://code.nsam.org/ns-3-allinone)

A file called ns-3-allinone is downloaded and can be viewed as



*Figure 4*

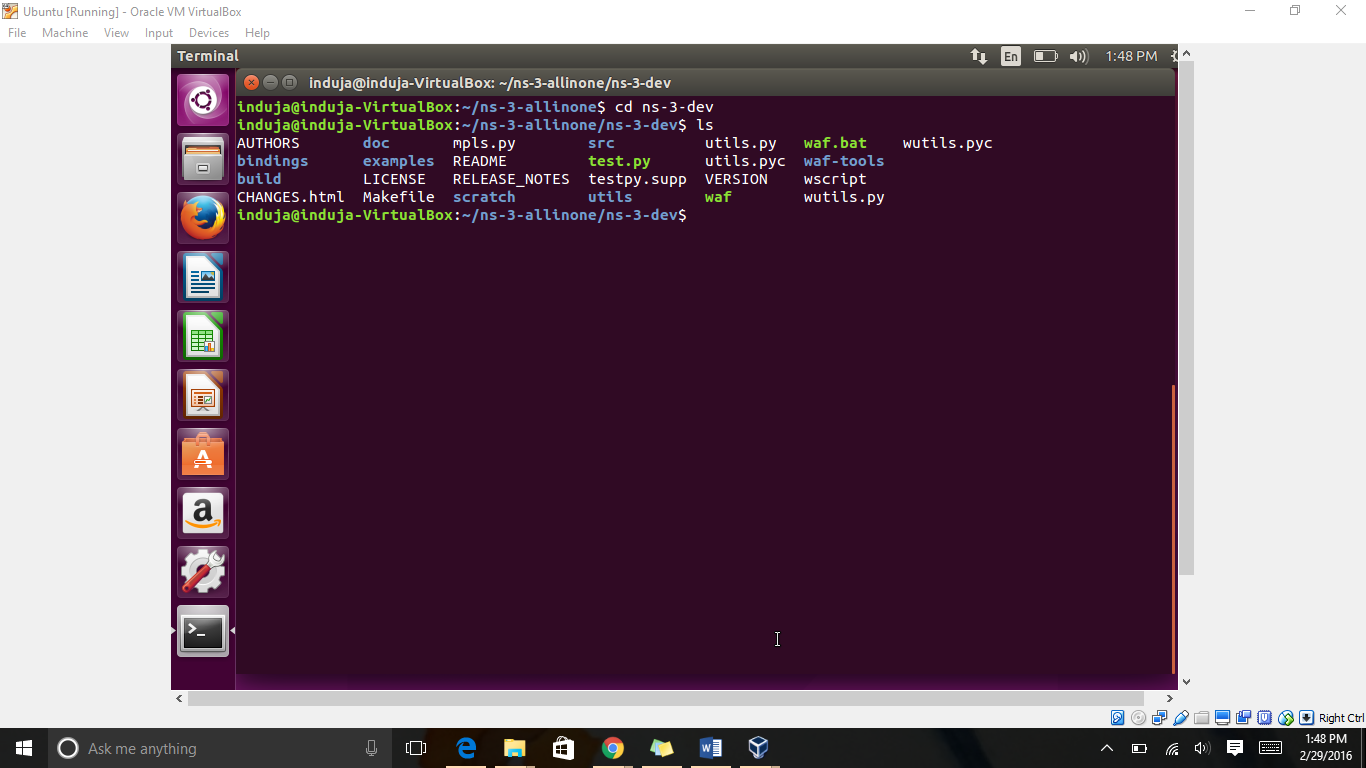
1. NS-3-allinone has various subsystems that need to be managed by a set of scripts. The next step is to download these scripts.

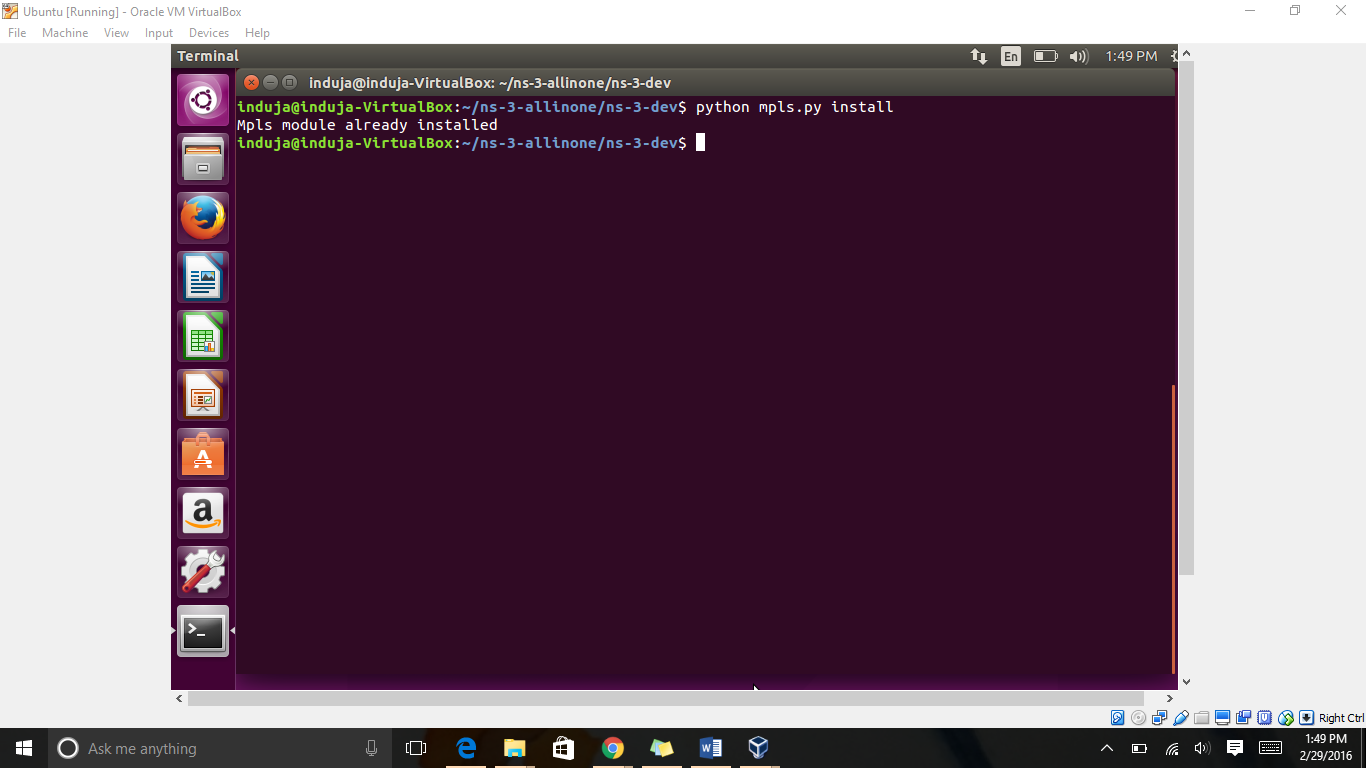


*Figure 5*

1. For the installation on the MPLS module, navigate to the ns-3-dev folder and then type

*python mpls.py install*

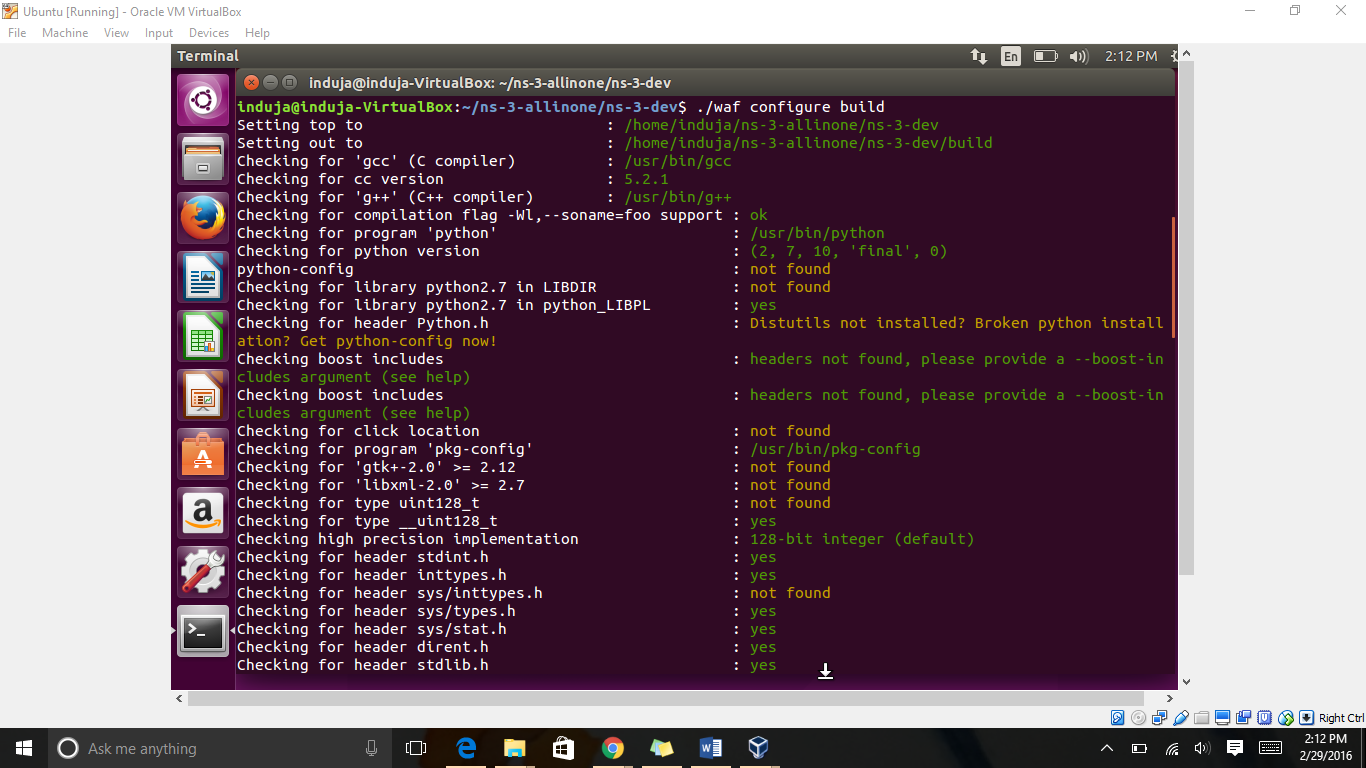




*Figure 6*

1. Following this, all other recommended MPLS modules should be built by typing the command mentioned below

*./waf configure build*



*Figure 7*

1. Then a few changes, mentioned below, are made in the 283.cc, mpls-test.cc and the wscript files

**283.cc:**

The line “UdpEchoClientHelper client ("192.168.4.2",port);” on line number 67 is replaced with “UdpEchoClientHelper client (Ipv4Address("192.168.4.2"), port);”

**Mpls.py:**

The line “AddTestCase (new NhlfeTestCase (),);” on line number 74 is replaced with “AddTestCase (new NhlfeTestCase (), QUICK);”

**Wscript:**

The line **“**bld.new\_task\_gen(features=['ns3header'])” is replaced with headers = bld(features='ns3header')

1. Now, the given Label Switched Network has to be simulated where the flow of packets from Ingress LSR1 and Ingress LSR2 and move to Egress LSR5. We need to assign labels for the two LSPs where the two LSPs merge at LSR3.

In the topology the LABEL 100 is assigned to outgoing packet at LSR 1 with the IP address 192.168.1.1 and destination LSR 5 with IP address 192.168.4.2. LABEL 200 is assigned to the incoming packet at LSR 2 with the IP address 192.168.2.1 and the same destination with the IP address 192.168.4.2.

1. After assigning these labels in every LSR, the following command is typed to start simulating:

*./waf –run scratch/283*

**WORKING AND SIMULATION OF THE PROGRAM**

**NHLFE, ILM and FTN:**

The MPLS architecture document (Rosen, Viswanathan, and Callon 2001) defines the components of the forwarding information base (FIB)as follows:

* Next Hop Label Forwarding Entry (NHLFE): An entry containing next-hop information (interface and next-hop address) and label manipulation instructions; it may also include label encoding, L2 encapsulation information, and other information required for processing packets in the associated stream.
* Incoming Label Map (ILM): A mapping from incoming labels to corresponding NHLFEs.
* FEC-to-NHLFE map (FTN): A mapping from the FEC of any incoming packets to corresponding NHLFEs.

When a packet with no label first comes to an LSR, LSR determines the FEC to route that packet, then it will look at FIB to find the FTN that matches the FEC. This FTN has a label and an NHLFE that contains the next hop for routing the packet.

When a packet that has already a label comes to an LSR, it will search for the ILM part of FIB that matches the incoming label. The associated NHLFE to that ILM has the action LSR should take about that label and the next hop information. If the NHLFE indicates that the label should be swapped, LSR will swap the label and will send the packet to the next hop. If the corresponding NHLFE says the label should be popped, LSR will remove the label and route the packet to the next hop.

**PROGRAM CODE WITH EXPLAINATION**

#include "ns3/core-module.h"

#include "ns3/network-module.h"

#include "ns3/internet-module.h"

#include "ns3/point-to-point-module.h"

#include "ns3/applications-module.h"

#include "ns3/mpls-module.h"

#include "ns3/ipv4-global-routing-helper.h"

//Above are the modules included in the program. It makes the code implementation easy. When we build the file waf automatically generates a module to load the public header files.

#include <iostream>

using namespace ns3;

//This helps in creating a global namespace scope which includes all ns-3 related declarations.

using namespace mpls;

int //This part of the code is the main function

main (int argc, char \*argv[])

{

LogComponentEnable ("UdpEchoClientApplication", LOG\_LEVEL\_INFO);

// “LogComponentEnable” allows us to keep a record of all the functions the server and clients use and the packets they receive.

LogComponentEnable ("UdpEchoServerApplication", LOG\_LEVEL\_INFO);

//LOG\_LEVEL\_INFO displays informational messages in the communication between nodes

LogComponentEnable ("mpls::MplsProtocol", LOG\_LEVEL\_DEBUG);

LogComponentEnable ("mpls::Ipv4Routing", LOG\_LEVEL\_DEBUG);

LogComponentEnable ("MplsNetworkDiscoverer", LOG\_LEVEL\_DEBUG);

//LOG\_LEVEL\_INFO and LOG\_LEVEL\_DEBUG are used for the clients and server

NodeContainer hosts;

NodeContainer routers;

NodeContainer routers2;

PointToPointHelper pointToPoint; // Point To Point helpers are used to create channels for communication

Ipv4AddressHelper address; //Helps us associate the devices on our nodes with IPv4 address

NetDeviceContainer devices; //Installs devices on the nodes

InternetStackHelper internet;

MplsNetworkConfigurator network; //MPLS network configured

routers = network.CreateAndInstall (5); //routers created

hosts.Create (3); //hosts created

internet.Install (hosts);

pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("100Mbps"));

pointToPoint.SetChannelAttribute ("Delay", StringValue ("1ms"));

//The delay and bandwidth are set as 1ms and 100Mbps for each link respectively

uint16\_t port = 9; //port number 9 is used for communication

UdpEchoServerHelper server (port);

ApplicationContainer apps = server.Install (hosts.Get(1));

apps.Start (Seconds (0.0));

apps.Stop (Seconds (10.0));

//For the installation of the application on the server node the following commands are used

UdpEchoClientHelper client ("192.168.4.2", port);

client.SetAttribute ("MaxPackets", UintegerValue (1));

client.SetAttribute ("Interval", TimeValue (MilliSeconds (0.00)));

client.SetAttribute ("PacketSize", UintegerValue (1024));

apps = client.Install (hosts.Get (0));

apps = client.Install (hosts.Get (2));

apps.Start (Seconds (0.0));

apps.Stop (Seconds (2.0));

// Hosts configuration

devices = pointToPoint.Install (hosts.Get(0), routers.Get(0));

address.SetBase ("192.168.1.0", "255.255.255.0");

address.Assign (devices);

devices = pointToPoint.Install (hosts.Get(2), routers.Get(1));

address.SetBase ("192.168.2.0", "255.255.255.0");

address.Assign (devices);

// Routers configuration

devices = pointToPoint.Install (routers.Get(0), routers.Get(2));

address.SetBase ("10.1.1.0", "255.255.255.0");

address.Assign (devices);

devices = pointToPoint.Install (routers.Get(1), routers.Get(2));

address.SetBase ("10.1.3.0", "255.255.255.0");

address.Assign (devices);

devices = pointToPoint.Install (routers.Get(2), routers.Get(3));

address.SetBase ("10.1.4.0", "255.255.255.0");

address.Assign (devices);

devices = pointToPoint.Install (routers.Get(3), routers.Get(4));

address.SetBase ("10.1.5.0", "255.255.255.0");

address.Assign (devices);

// Address is not specified. Mpls interfaces will be disabled.

devices = pointToPoint.Install (routers.Get(4), routers.Get(0));

NhlfeSelectionPolicyHelper policy; // Creating the policy and assigning it to the routers

policy.SetAttribute ("MaxPacketsInTxQueue", IntegerValue (0));

MplsSwitch LSR1 (routers.Get (0));

MplsSwitch LSR2 (routers.Get (1));

MplsSwitch LSR3 (routers.Get (2));

MplsSwitch LSR4 (routers.Get (3));

MplsSwitch LSR5 (routers.Get (4));

LSR1.SetSelectionPolicy (policy);

//The below code explains the ingress LSR Label swapping. The ingress LSR1 and LSR2 contain the source and destination IPv4 address of the incoming packet, the incoming label value and the next hop address. The ingress node should add an entry in the routing table to redirect certain flows to the NHLFE entry.

LSR1.AddFtn (Ipv4Source ("192.168.1.1") && Ipv4Destination ("192.168.4.2"),

Nhlfe (Swap (100), Ipv4Address ("10.1.1.2"))

);

LSR2.AddFtn (Ipv4Source ("192.168.2.1") && Ipv4Destination ("192.168.4.2"),

Nhlfe (Swap (200), Ipv4Address ("10.1.3.2"))

);

//The core LSRs 3 and 4 map the incoming labels to the NHLFE entry (ILM) and swap labels. The incoming label values are swapped.

LSR3.AddIlm (100,

Nhlfe(Swap (300), Ipv4Address ("10.1.4.2"))

);

LSR3.AddIlm (200,

Nhlfe(Swap (300), Ipv4Address ("10.1.4.2"))

);

LSR4.AddIlm (300,

Nhlfe(Swap (400), Ipv4Address ("10.1.5.2"))

);

//The egress LSR5 only adds an entry in the ILM map indicating the label that has to be popped out

LSR5.AddIlm (400,

Nhlfe(Pop ())

);

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

network.DiscoverNetwork ();

network.ShowConfig ();

Simulator::Run (); //Runs the simulation and aborts incase no events are present

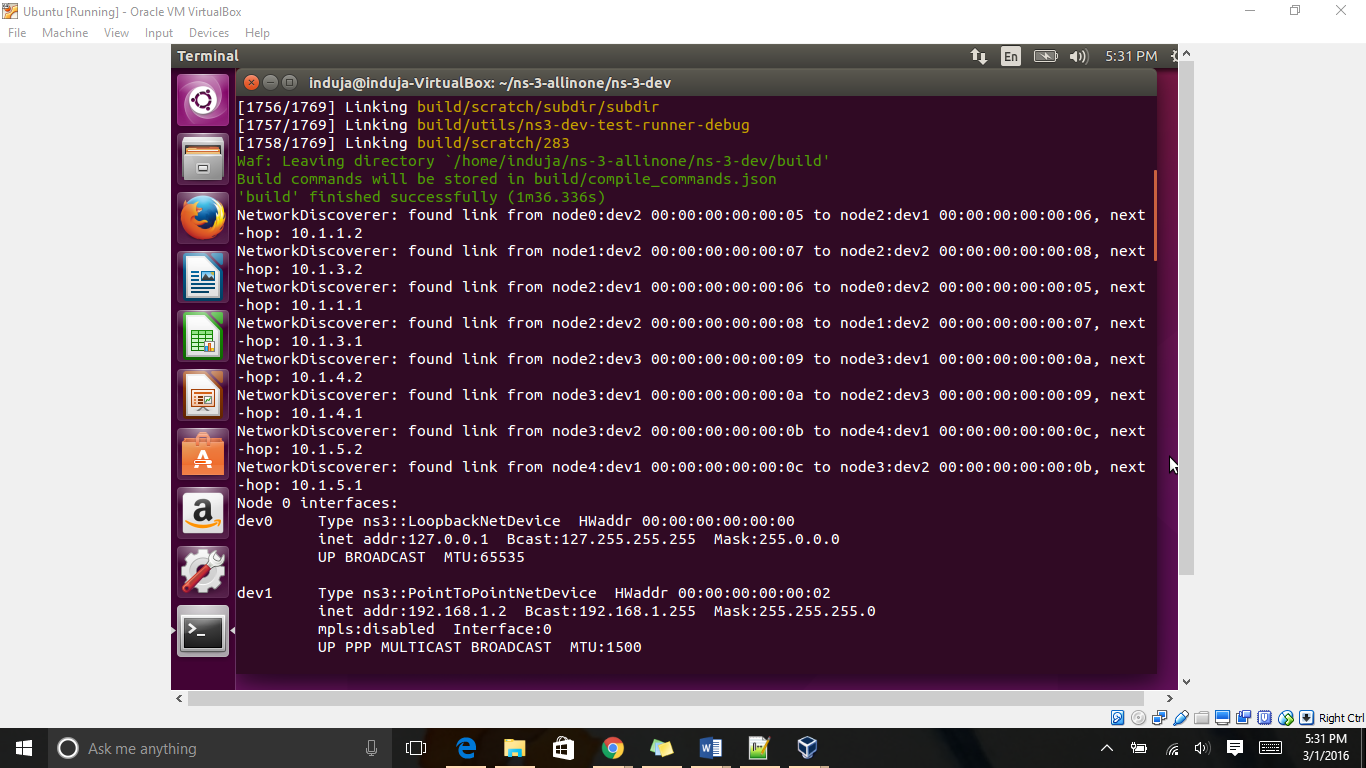
Simulator::Destroy (); //Frees memory by destroying the objects after simulation

return 0;

}

**OUTPUT**

In this part of the report, the output is explained which contains the simulation of the network showing the nodes, interfaces and the links between the nodes as well as the procedure of assigning the labels.



*Figure 8*

NetworkDiscoverer: found link from node0:dev2 00:00:00:00:00:05 to node2:dev1 00:00:00:00:00:06, next-hop: 10.1.1.2

NetworkDiscoverer: found link from node1:dev2 00:00:00:00:00:07 to node2:dev2 00:00:00:00:00:08, next-hop: 10.1.3.2

NetworkDiscoverer: found link from node2:dev1 00:00:00:00:00:06 to node0:dev2 00:00:00:00:00:05, next-hop: 10.1.1.1

NetworkDiscoverer: found link from node2:dev2 00:00:00:00:00:08 to node1:dev2 00:00:00:00:00:07, next-hop: 10.1.3.1

NetworkDiscoverer: found link from node2:dev3 00:00:00:00:00:09 to node3:dev1 00:00:00:00:00:0a, next-hop: 10.1.4.2

NetworkDiscoverer: found link from node3:dev1 00:00:00:00:00:0a to node2:dev3 00:00:00:00:00:09, next-hop: 10.1.4.1

NetworkDiscoverer: found link from node3:dev2 00:00:00:00:00:0b to node4:dev1 00:00:00:00:00:0c, next-hop: 10.1.5.2

NetworkDiscoverer: found link from node4:dev1 00:00:00:00:00:0c to node3:dev2 00:00:00:00:00:0b, next-hop: 10.1.5.1

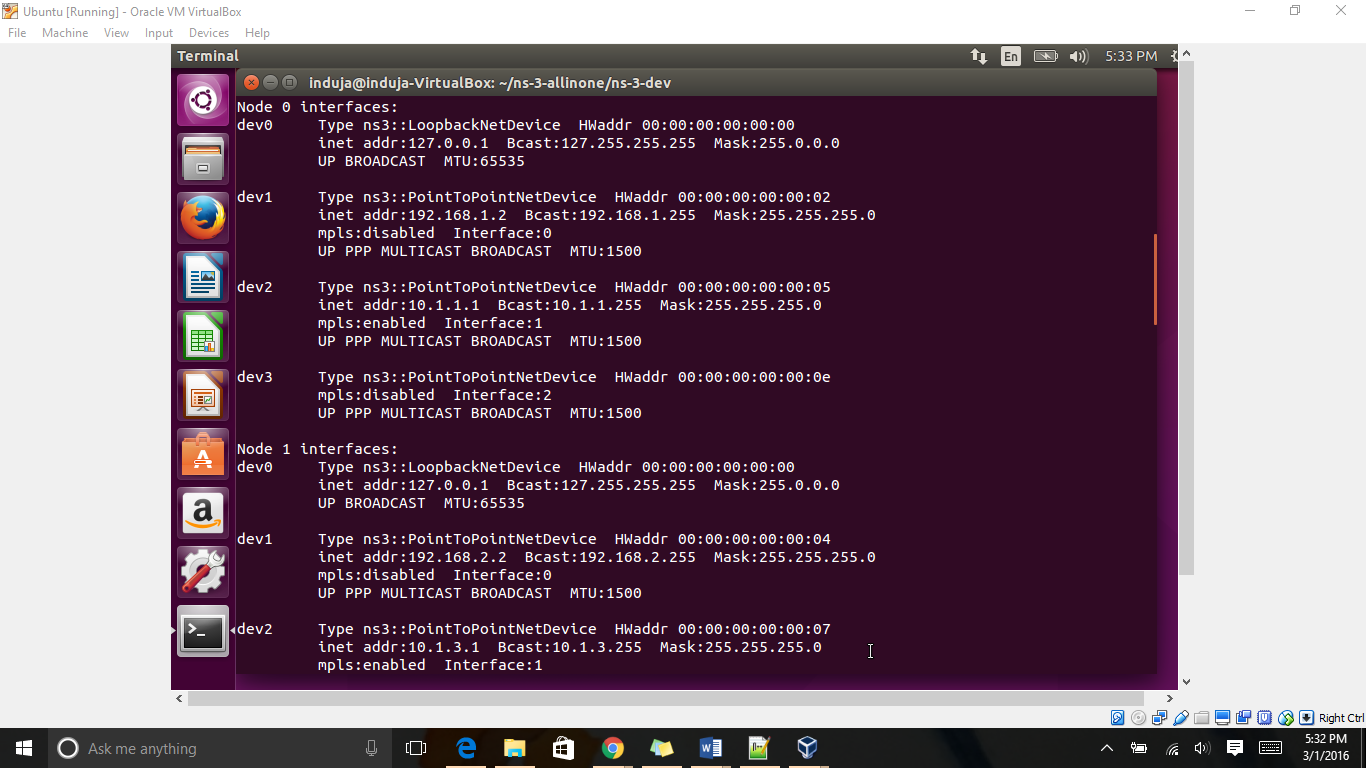
NetworkDiscoverer is used to find links between routers in our network and assign

MAC addresses to the interfaces.

The propagation of packet can be understood by the following table.

|  |  |  |
| --- | --- | --- |
| Path travelled | IP Address | Next Hop IP Address |
| Node0 to node2 | 00:00:00:00:00:05 to 00:00:00:00:00:06 | 10.1.1.2 |
| Node1 to node2 | 00:00:00:00:00:07 to 00:00:00:00:00:08 | 10.1.3.2 |
| Node 2 to node0 | 00:00:00:00:00:06 to 00:00:00:00:00:05 | 10.1.1.1 |
| Node2 to node1 | 00:00:00:00:00:08 to 00:00:00:00:00:07 | 10.1.3.1 |
| Node2 to node3 | 00:00:00:00:00:09 to 00:00:00:00:00:0a | 10.1.4.2 |
| Node3 to node2 | 00:00:00:00:00:0a to 00:00:00:00:00:09 | 10.1.4.1 |
| Node 3 to node4 | 00:00:00:00:00:0b to 00:00:00:00:00:0c | 10.1.5.2 |
| Node4 to node3 | 00:00:00:00:00:0c to 00:00:00:00:00:0b | 10.1.5.1 |

LSR1:



*Figure 9*

Interfaces for each node is shown. Each node has an interface with local host (itself) which always has IP address of 127.0.0.1.

Node0 has local host MAC address as 00:00:00:00:00:00 and IP address 127.0.0.1 and Maximum transmission size of 65535 bytes and broadcast IP of 127.255.255.255 and net-mask of 255.0.0.0

Node 0 interfaces:

dev0 Type ns3::LoopbackNetDevice HWaddr 00:00:00:00:00:00

inet addr:127.0.0.1 Bcast:127.255.255.255 Mask:255.0.0.0

UP BROADCAST MTU:65535

Node0 also have an interface with a host with IP address 192.168.1.2 and MAC address 00:00:00:00:02. broadcast address 192.186.1.255 and MTU 1500. MPLS is disable and this link is part of an IP network, So node0 (LSR1) is an ingress.

dev1 Type ns3::PointToPointNetDevice HWaddr 00:00:00:00:00:02

inet addr:192.168.1.2 Bcast:192.168.1.255 Mask:255.255.255.0

mpls:disabled Interface:0

UP PPP MULTICAST BROADCAST MTU:1500

Node0 has another interface with a MPLS network with interface address 10.1.1.1 and broadcast address of 10.1.1.255 and MTU 1500.

dev2 Type ns3::PointToPointNetDevice HWaddr 00:00:00:00:00:05

inet addr:10.1.1.1 Bcast:10.1.1.255 Mask:255.255.255.0

mpls:enabled Interface:1

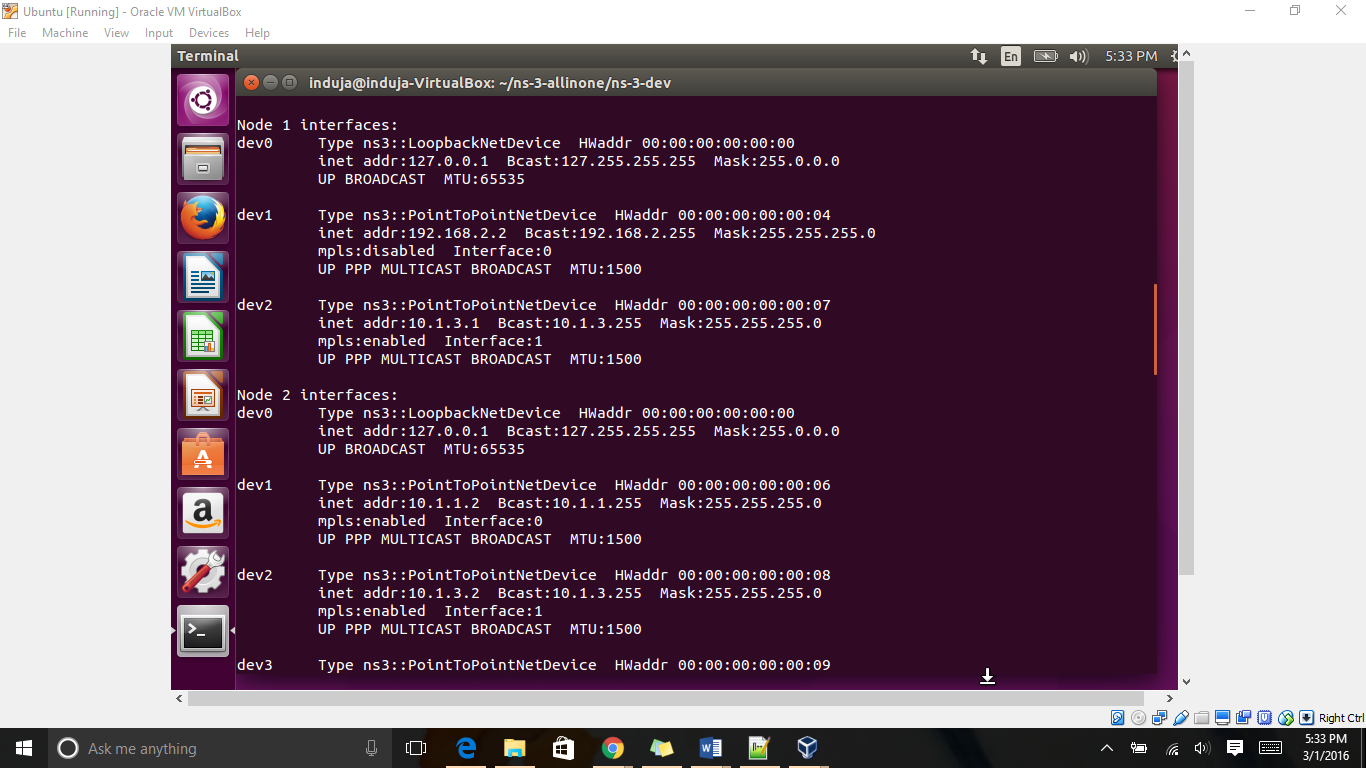
UP PPP MULTICAST BROADCAST MTU:1500

dev3 Type ns3::PointToPointNetDevice HWaddr 00:00:00:00:00:0e

mpls:disabled Interface:2

UP PPP MULTICAST BROADCAST MTU:1500

LSR2:



*Figure 10*

This shows LSR2 (node 1) interface with IP address 192.168.2.2. It is also connected to another LSR with interface IP 10.1.3.1.

Node 1 interfaces:

dev0 Type ns3::LoopbackNetDevice HWaddr 00:00:00:00:00:00

inet addr:127.0.0.1 Bcast:127.255.255.255 Mask:255.0.0.0

UP BROADCAST MTU:65535

dev1 Type ns3::PointToPointNetDevice HWaddr 00:00:00:00:00:04

inet addr:192.168.2.2 Bcast:192.168.2.255 Mask:255.255.255.0

mpls:disabled Interface:0

UP PPP MULTICAST BROADCAST MTU:1500

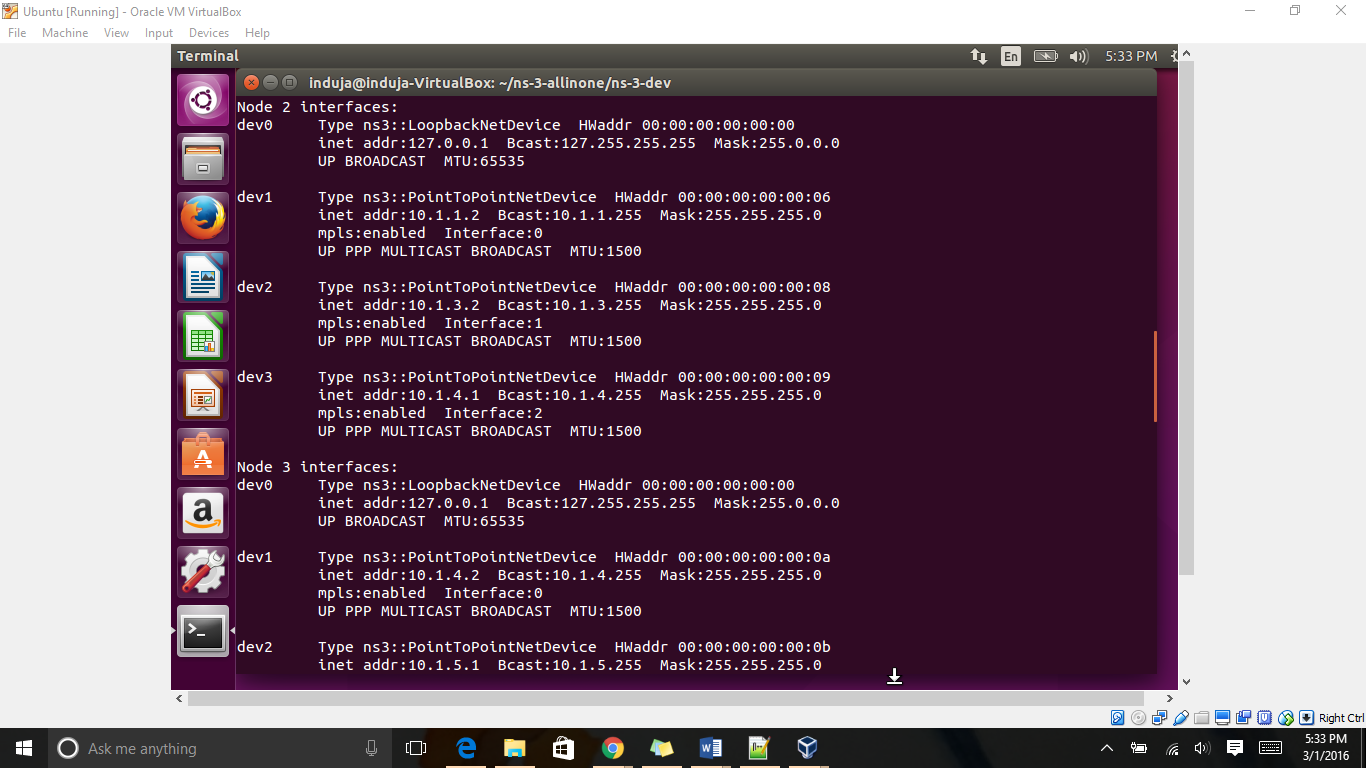
dev2 Type ns3::PointToPointNetDevice HWaddr 00:00:00:00:00:07

inet addr:10.1.3.1 Bcast:10.1.3.255 Mask:255.255.255.0

mpls:enabled Interface:1

UP PPP MULTICAST BROADCAST MTU:1500

LSR3:



*Figure 11*

Node2 has three interfaces with three LSRs in MPLS network with interface IP address 10.1.1.2 (LSR1) and 10.1.3.2(LSR2) and 10.1.4.1 (LSR4) with MTU 1500.

Node 2 interfaces:

dev0 Type ns3::LoopbackNetDevice HWaddr 00:00:00:00:00:00

inet addr:127.0.0.1 Bcast:127.255.255.255 Mask:255.0.0.0

UP BROADCAST MTU:65535

dev1 Type ns3::PointToPointNetDevice HWaddr 00:00:00:00:00:06

inet addr:10.1.1.2 Bcast:10.1.1.255 Mask:255.255.255.0

mpls:enabled Interface:0

UP PPP MULTICAST BROADCAST MTU:1500

dev2 Type ns3::PointToPointNetDevice HWaddr 00:00:00:00:00:08

inet addr:10.1.3.2 Bcast:10.1.3.255 Mask:255.255.255.0

mpls:enabled Interface:1

UP PPP MULTICAST BROADCAST MTU:1500

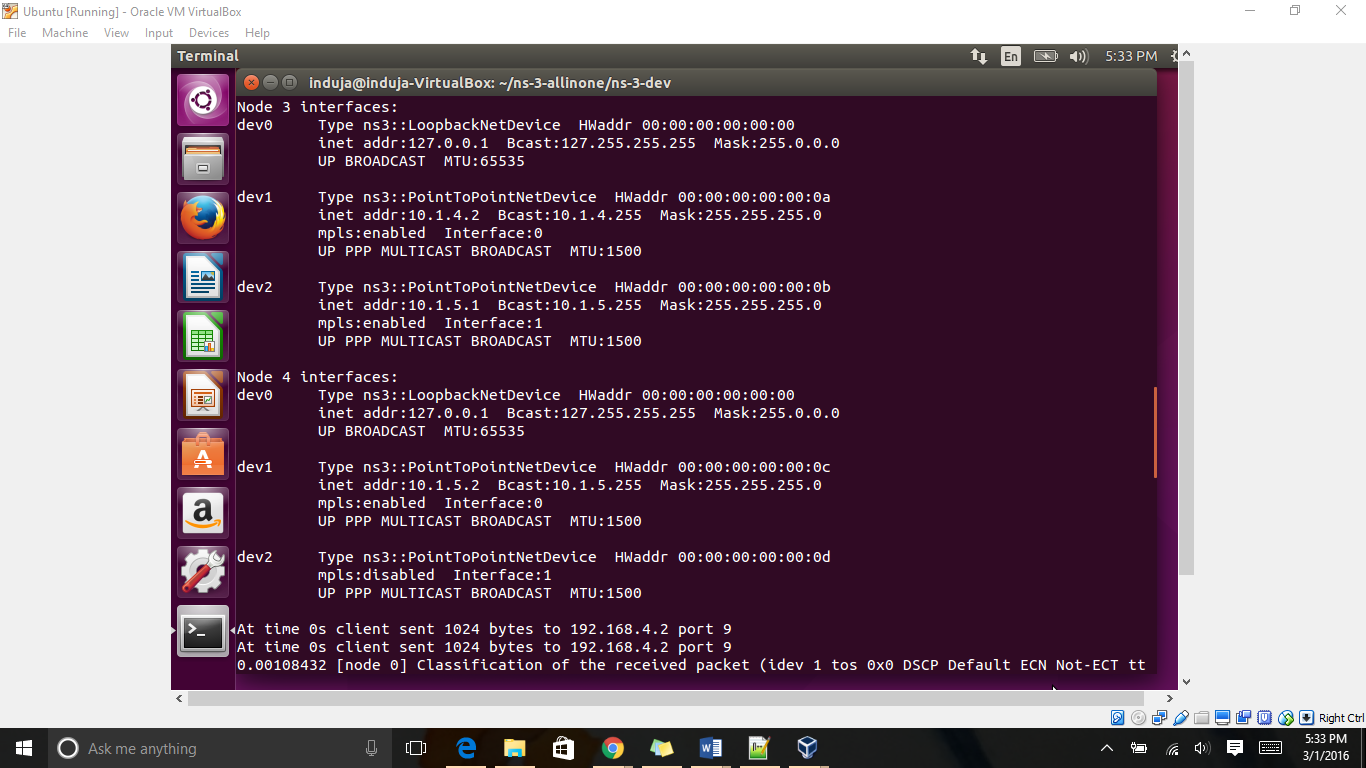
dev3 Type ns3::PointToPointNetDevice HWaddr 00:00:00:00:00:09

inet addr:10.1.4.1 Bcast:10.1.4.255 Mask:255.255.255.0

mpls:enabled Interface:2

UP PPP MULTICAST BROADCAST MTU:1500

LSR4:



*Figure 12*

Node 3 interfaces:

dev0 Type ns3::LoopbackNetDevice HWaddr 00:00:00:00:00:00

inet addr:127.0.0.1 Bcast:127.255.255.255 Mask:255.0.0.0

UP BROADCAST MTU:65535

dev1 Type ns3::PointToPointNetDevice HWaddr 00:00:00:00:00:0a

inet addr:10.1.4.2 Bcast:10.1.4.255 Mask:255.255.255.0

mpls:enabled Interface:0

UP PPP MULTICAST BROADCAST MTU:1500

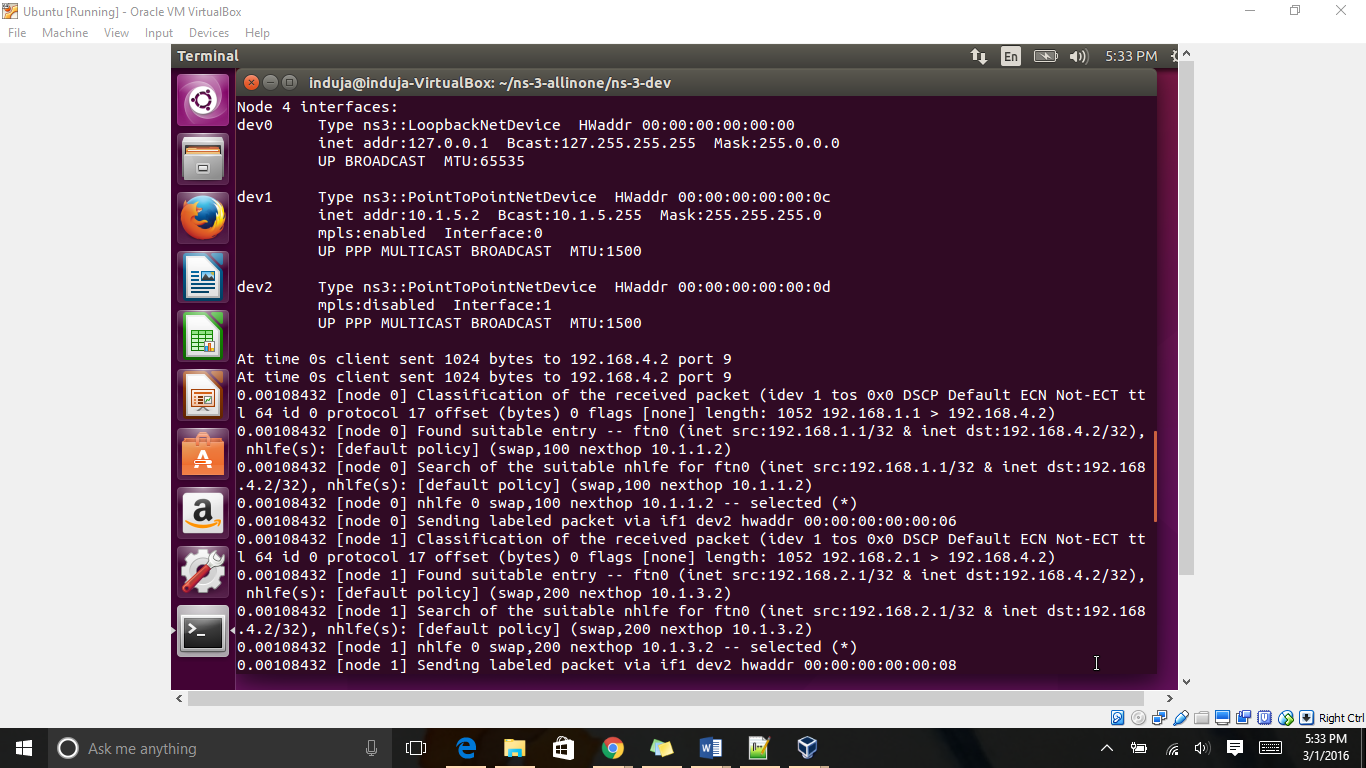
dev2 Type ns3::PointToPointNetDevice HWaddr 00:00:00:00:00:0b

inet addr:10.1.5.1 Bcast:10.1.5.255 Mask:255.255.255.0

mpls:enabled Interface:1

UP PPP MULTICAST BROADCAST MTU:1500

LSR5:



*Figure 13*

Node4 (LSR5) interfaces with IP address 10.1.5.2 with broadcast address 10.1.5.255 and MTU 1500. It is an egress LSR.

Node 4 interfaces:

dev0 Type ns3::LoopbackNetDevice HWaddr 00:00:00:00:00:00

inet addr:127.0.0.1 Bcast:127.255.255.255 Mask:255.0.0.0

UP BROADCAST MTU:65535

dev1 Type ns3::PointToPointNetDevice HWaddr 00:00:00:00:00:0c

inet addr:10.1.5.2 Bcast:10.1.5.255 Mask:255.255.255.0

mpls:enabled Interface:0

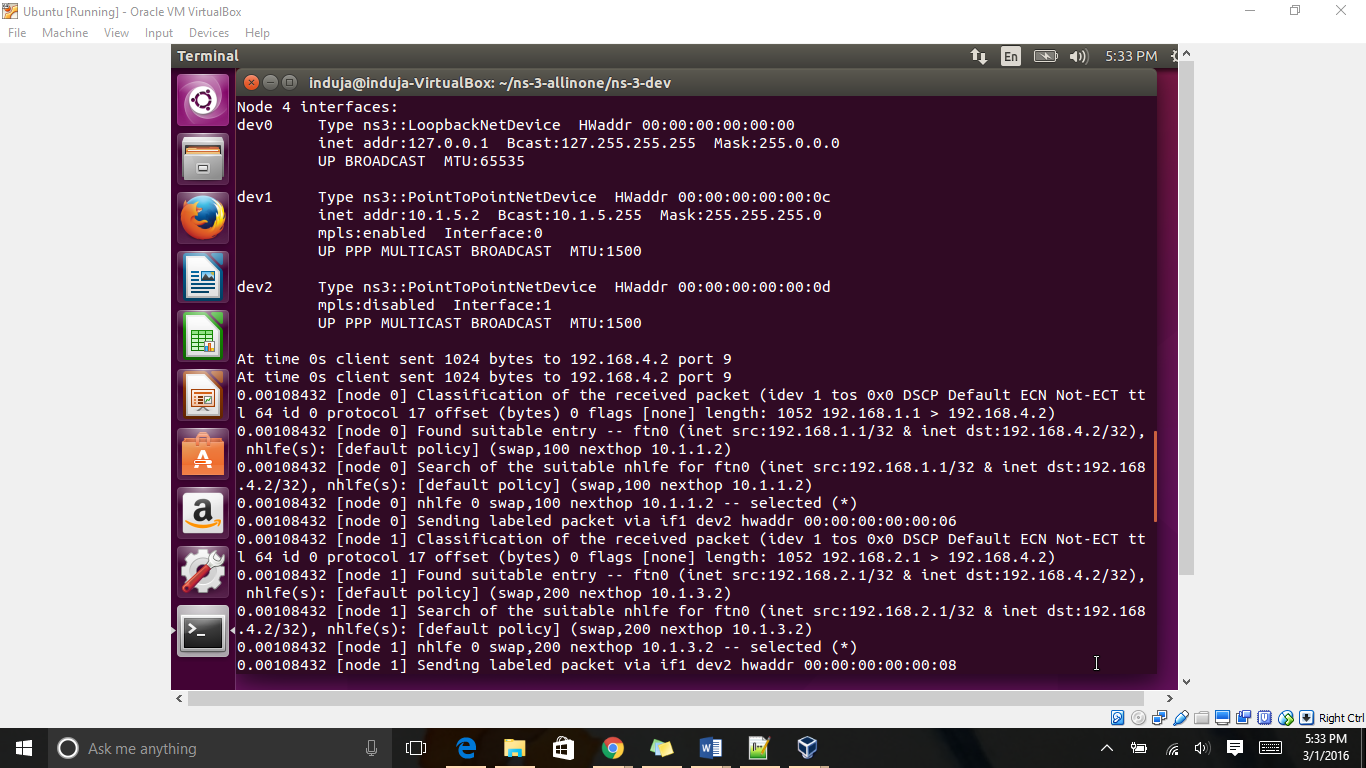
UP PPP MULTICAST BROADCAST MTU:1500

dev2 Type ns3::PointToPointNetDevice HWaddr 00:00:00:00:00:0d

mpls:disabled Interface:1

UP PPP MULTICAST BROADCAST MTU:1500

For Ingress LSR1 and LSR2:



*Figure 14*

At time 0s client sent 1024 bytes to 192.168.4.2 port 9

At time 0s client sent 1024 bytes to 192.168.4.2 port 9

At node0 the received packet is classified and based source and destination, LSR1 (node0) finds a FEC for it. Then it will search FTN to map the FEC to a NHLFE entry.

0.00108432 [node 0] Classification of the received packet (idev 1 tos 0x0 DSCP Default ECN Not-ECT ttl 64 id 0 protocol 17 offset (bytes) 0 flags [none] length: 1052 192.168.1.1 > 192.168.4.2)

At node0 the received packet is classified and based source and destination, LSR1 (node0) finds a FEC for it. Then it will search FTN to map the FEC to a NHLFE entry

0.00108432 [node 0] Found suitable entry -- ftn0 (inet src:192.168.1.1/32 & inet dst:192.168.4.2/32), nhlfe(s): [default policy] (swap,100 nexthop 10.1.1.2)

0.00108432 [node 0] Search of the suitable nhlfe for ftn0 (inet src:192.168.1.1/32 & inet dst:192.168.4.2/32), nhlfe(s): [default policy] (swap,100 nexthop 10.1.1.2)

Based on the NHLFE information Node0 (LSR1) will assign label 100 to the packet and send it over 10.1.1.2 with MAC address 00:00:00:00:00:06

0.00108432 [node 0] nhlfe 0 swap,100 nexthop 10.1.1.2 -- selected (\*)

0.00108432 [node 0] Sending labeled packet via if1 dev2 hwaddr 00:00:00:00:00:06

The packet arrived at Node1 (LSR2) will also be routed the same way

0.00108432 [node 1] Classification of the received packet (idev 1 tos 0x0 DSCP Default ECN Not-ECT ttl 64 id 0 protocol 17 offset (bytes) 0 flags [none] length: 1052 192.168.2.1 > 192.168.4.2)

0.00108432 [node 1] Found suitable entry -- ftn0 (inet src:192.168.2.1/32 & inet dst:192.168.4.2/32), nhlfe(s): [default policy] (swap,200 nexthop 10.1.3.2)

0.00108432 [node 1] Search of the suitable nhlfe for ftn0 (inet src:192.168.2.1/32 & inet dst:192.168.4.2/32), nhlfe(s): [default policy] (swap,200 nexthop 10.1.3.2)

0.00108432 [node 1] nhlfe 0 swap,200 nexthop 10.1.3.2 -- selected (\*)

NHLFE information shows that the label 200 assign to the packet and it should be send to next hop with IP address 10.1.3.2 via MAC address 00:00:00:00:00:08

0.00108432 [node 1] Sending labeled packet via if1 dev2 hwaddr 00:00:00:00:00:08

For core LSR3 and LSR4

Node 2 is a core router so all it has to do is look up in its MPLS active sector of the forwarding table and swap the label accordingly. After the label is swapped it will send the label out of the port once the entry has been found in the forwarding table. Thus node2 uses an ILM map and FTN map to find out the label to be swapped with the current label.

0.00216896 [node 2] Packet from 03-06-00:00:00:00:00:05 received on node 2

0.00216896 [node 2] Stack top label:100 ttl:63

0.00216896 [node 2] Searching of label mapping for label 100 if0 dev1

0.00216896 [node 2] Found suitable entry -- ilm0 label 100, nhlfe(s): [default policy] (swap,300 nexthop 10.1.4.2)

0.00216896 [node 2] Search of the suitable nhlfe for ilm0 label 100, nhlfe(s): [default policy] (swap,300 nexthop 10.1.4.2)

Associated NHLFE shows that label 100 should be swapped with 300 and the next hop is 10.1.4.2. Node2 will swap the label and send the packet over MAC address 00:00:00:00:00:0a

0.00216896 [node 2] nhlfe 0 swap,300 nexthop 10.1.4.2 -- selected (\*)

0.00216896 [node 2] Sending labeled packet via if2 dev3 hwaddr 00:00:00:00:00:0a

0.00216896 [node 2] Packet from 03-06-00:00:00:00:00:07 received on node 2

0.00216896 [node 2] Stack top label:200 ttl:63

0.00216896 [node 2] Searching of label mapping for label 200 if1 dev2

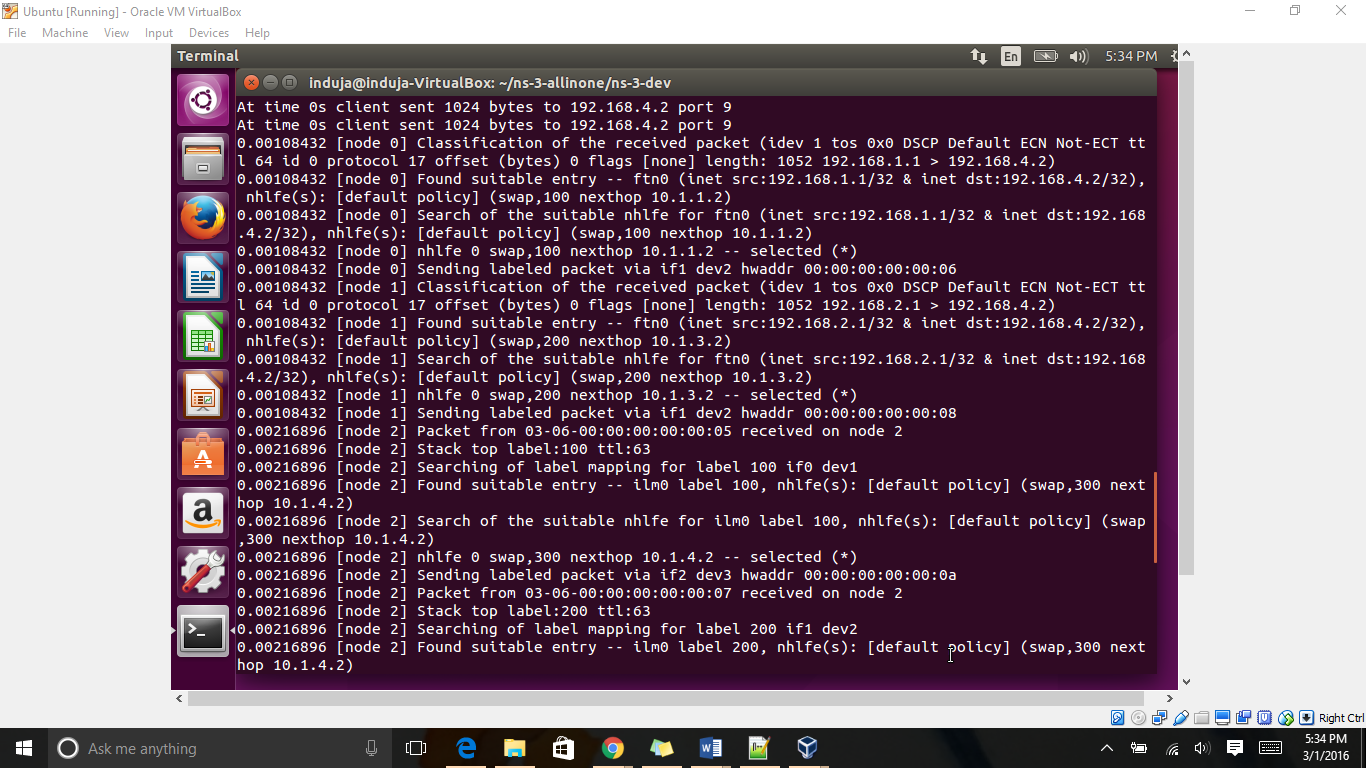
0.00216896 [node 2] Found suitable entry -- ilm0 label 200, nhlfe(s): [default policy] (swap,300 nexthop 10.1.4.2)

For LSR3, the incoming label 200 from LSR2 is swapped with label 300 and the next hop is 10.1.4.2. hence, node2 will also swap label 200 with 300 and will send to node3.

0.00216896 [node 2] Search of the suitable nhlfe for ilm0 label 200, nhlfe(s): [default policy] (swap,300 nexthop 10.1.4.2)

0.00216896 [node 2] nhlfe 0 swap,300 nexthop 10.1.4.2 -- selected (\*)

0.00216896 [node 2] Sending labeled packet via if2 dev3 hwaddr 00:00:00:00:00:0a

**

*Figure 15*

Associated NHLFE shows that label 300 should be swapped with 400 and the next hop is 10.1.5.2. Node3 will swap the label and send the packet over MAC address 00:00:00:00:00:0c

0.0032536 [node 3] Packet from 03-06-00:00:00:00:00:09 received on node 3

0.0032536 [node 3] Stack top label:300 ttl:62

0.0032536 [node 3] Searching of label mapping for label 300 if0 dev1

0.0032536 [node 3] Found suitable entry -- ilm0 label 300, nhlfe(s): [default policy] (swap,400 nexthop 10.1.5.2)

0.0032536 [node 3] Search of the suitable nhlfe for ilm0 label 300, nhlfe(s): [default policy] (swap,400 nexthop 10.1.5.2)

0.0032536 [node 3] nhlfe 0 swap,400 nexthop 10.1.5.2 -- selected (\*)

0.0032536 [node 3] Sending labeled packet via if1 dev2 hwaddr 00:00:00:00:00:0c

0.00333824 [node 3] Packet from 03-06-00:00:00:00:00:09 received on node 3

0.00333824 [node 3] Stack top label:300 ttl:62

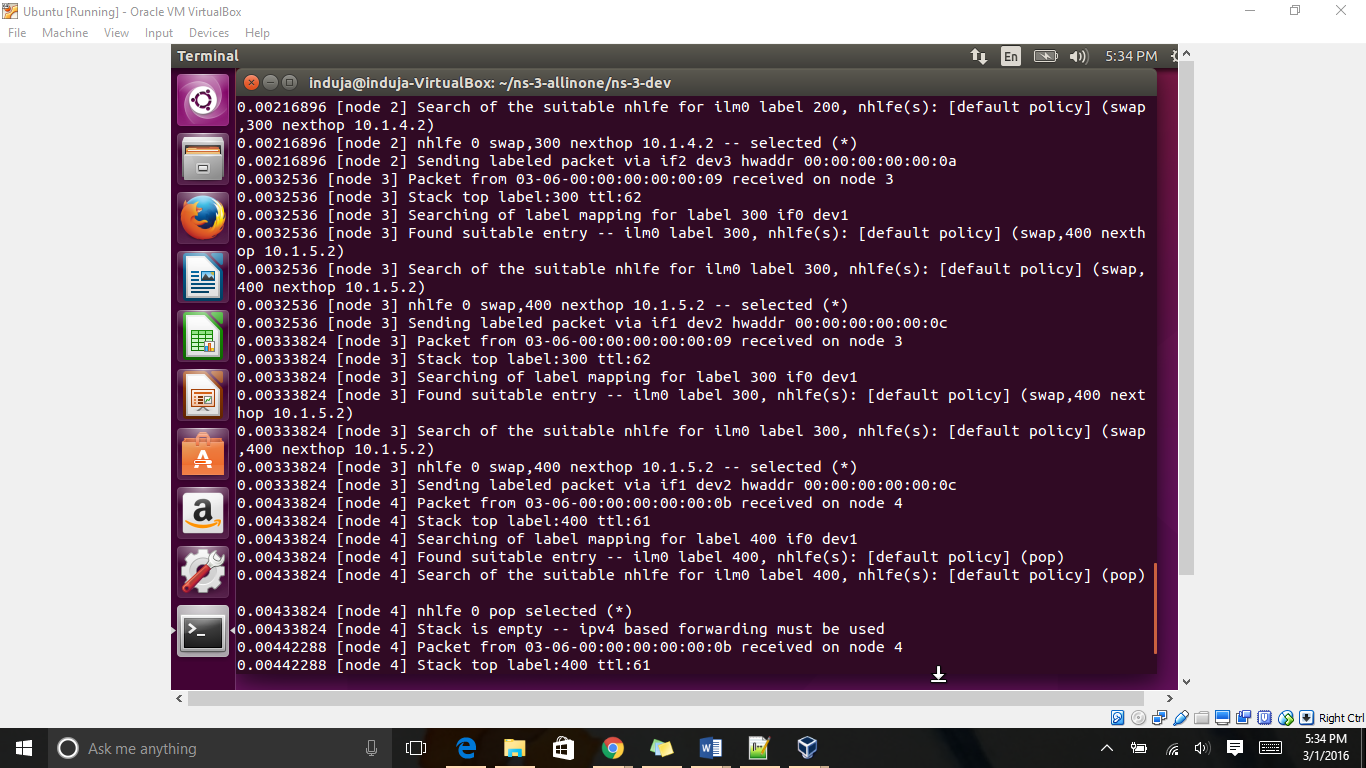
0.00333824 [node 3] Searching of label mapping for label 300 if0 dev1

0.00333824 [node 3] Found suitable entry -- ilm0 label 300, nhlfe(s): [default policy] (swap,400 nexthop 10.1.5.2)

0.00333824 [node 3] Search of the suitable nhlfe for ilm0 label 300, nhlfe(s): [default policy] (swap,400 nexthop 10.1.5.2)

0.00333824 [node 3] nhlfe 0 swap,400 nexthop 10.1.5.2 -- selected (\*)

0.00333824 [node 3] Sending labeled packet via if1 dev2 hwaddr 00:00:00:00:00:0c



*Figure 16*

For egress LSR5:

0.00433824 [node 4] Packet from 03-06-00:00:00:00:00:0b received on node 4

0.00433824 [node 4] Stack top label:400 ttl:61

0.00433824 [node 4] Searching of label mapping for label 400 if0 dev1

0.00433824 [node 4] Found suitable entry -- ilm0 label 400, nhlfe(s): [default policy] (pop)

0.00433824 [node 4] Search of the suitable nhlfe for ilm0 label 400, nhlfe(s): [default policy] (pop)

0.00433824 [node 4] nhlfe 0 pop selected (\*)

0.00433824 [node 4] Stack is empty -- ipv4 based forwarding must be used

Node 4 (LSR5) is an egress LSR and it will pop out the the label received that is 400 in out network.

0.00442288 [node 4] Packet from 03-06-00:00:00:00:00:0b received on node 4

0.00442288 [node 4] Stack top label:400 ttl:61

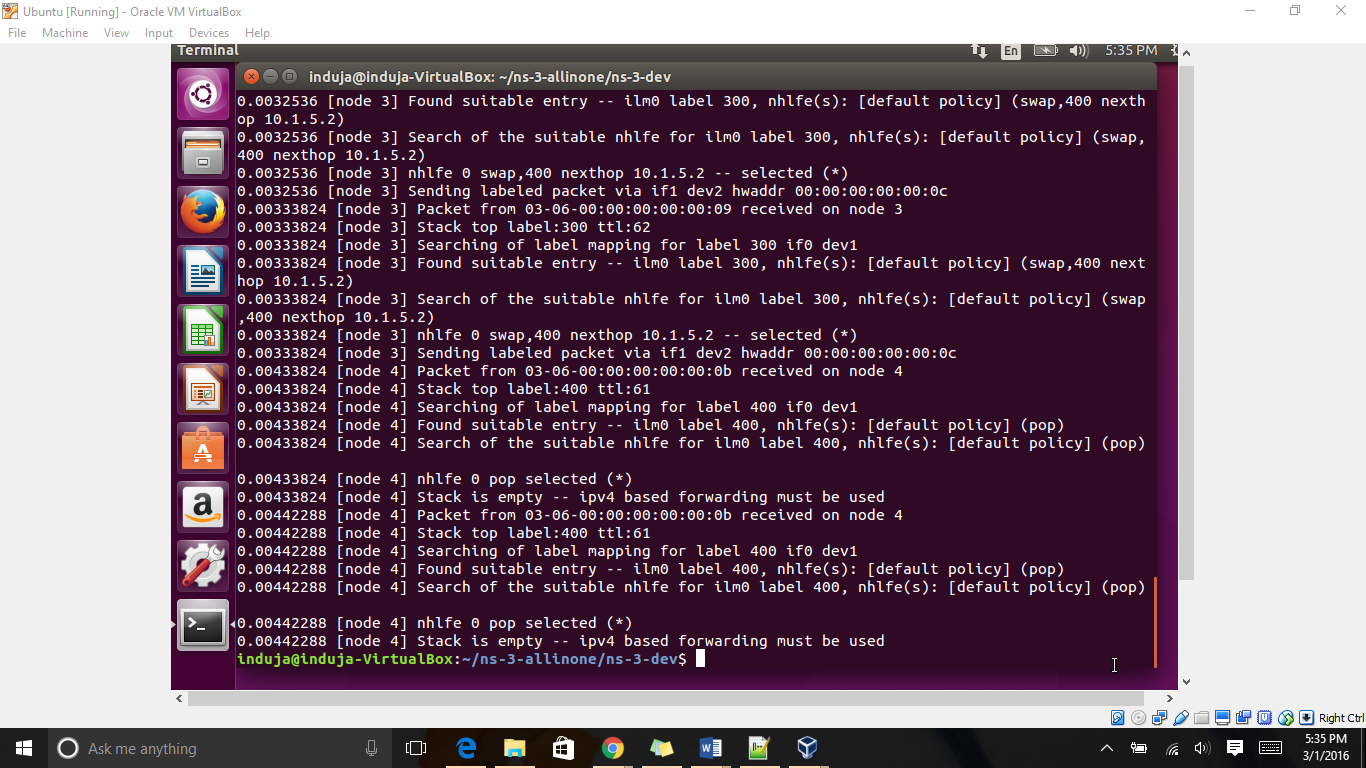
0.00442288 [node 4] Searching of label mapping for label 400 if0 dev1

0.00442288 [node 4] Found suitable entry -- ilm0 label 400, nhlfe(s): [default policy] (pop)

0.00442288 [node 4] Search of the suitable nhlfe for ilm0 label 400, nhlfe(s): [default policy] (pop)

0.00442288 [node 4] nhlfe 0 pop selected (\*)

0.00442288 [node 4] Stack is empty -- ipv4 based forwarding must be used



*Figure 17*

The last line of code shows that the label stack is empty and normal IPv4 routing should be used from here on to forward the packets.

Lowest Time to Live(TTL):

The lowest Time to Live was observed on the packets which were received on node 4. The TTL was observed to be 61.

**CONCLUSION**

In this project the concept of MPLS, a Label – Switched network and the simulation of computer networks in NS3 tool were studied. A detail analysis was done on each and every step of the C++ code given and understood the functioning of FIB such as NHFLE, ILM and FTN.