EE555 Project

“Open Flow Protocol “

Name – Rupasree Roy

Introduction:

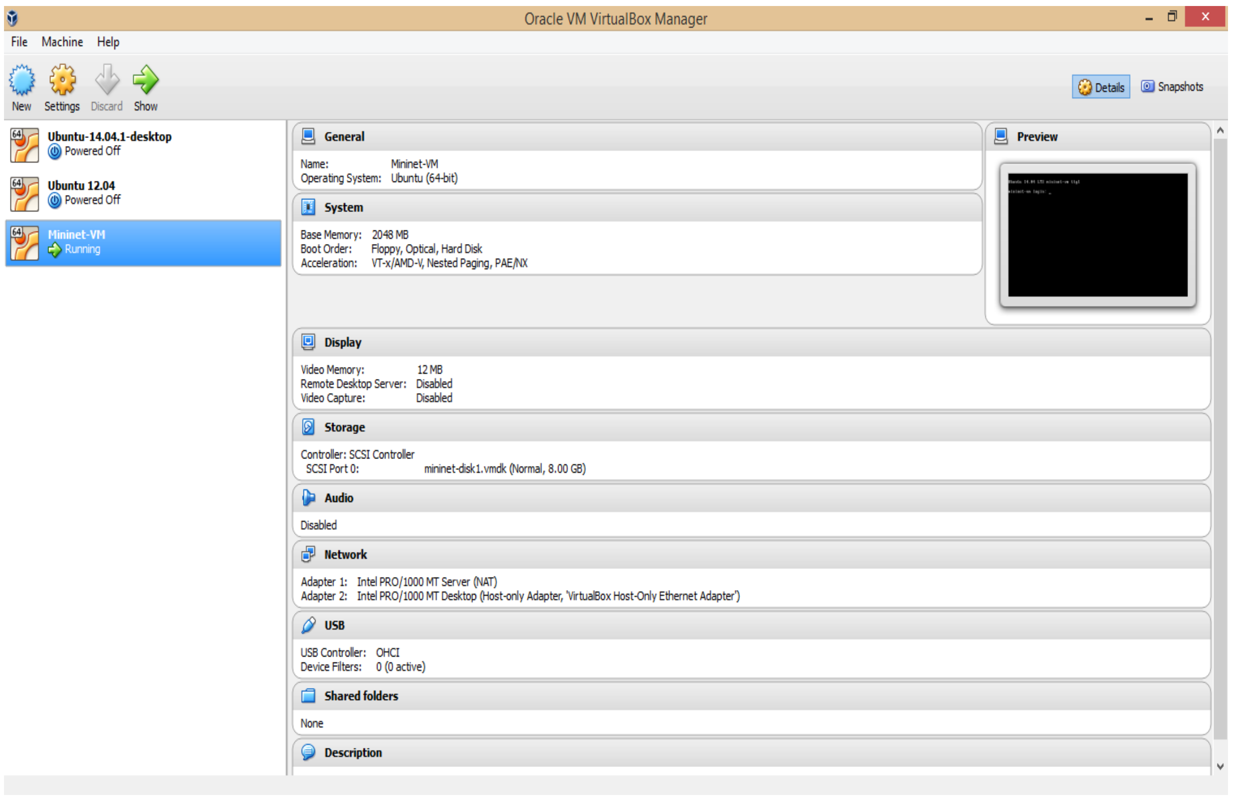
In this project we create a self-learning layer 3 switch. As cited in the tutorial OpenFlow is a communication protocol that gives access to the forwarding plane of a network switch or router over the network. It is used to govern the communication between a controller and the switch in a software defined network (SDN) environment. This project was divided into two parts where the topology for each part was different.

Steps taken to complete this project:

Download Files:

Step 1: I had Virtual box pre-installed in my computer.

Step 2: I installed Virtual Machine Image (OVF format) for Mininet 2.2.0 in my 64-bit system. In Ubuntu as mentioned in the tutorial X server, gnome Terminal and SSH are in built.



Ubuntu-14.04.1-desktop
Powered Off
Ubuntu 12.04
Powered Off
Mininet-VM
Running

General
Name: Mininet-VM
Operating System: Ubuntu (64-bit)

System
Base Memory: 2048 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX

Display
Video Memory: 12 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Storage
Controller: SCSI Controller
SCSI Port 0: mininet-disk1.vmdk (Normal, 8.00 GB)

Audio
Disabled

Network
Adapter 1: Intel PRO/1000 MT Server (NAT)
Adapter 2: Intel PRO/1000 MT Desktop (Host-only Adapter, 'VirtualBox Host-Only Ethernet Adapter')

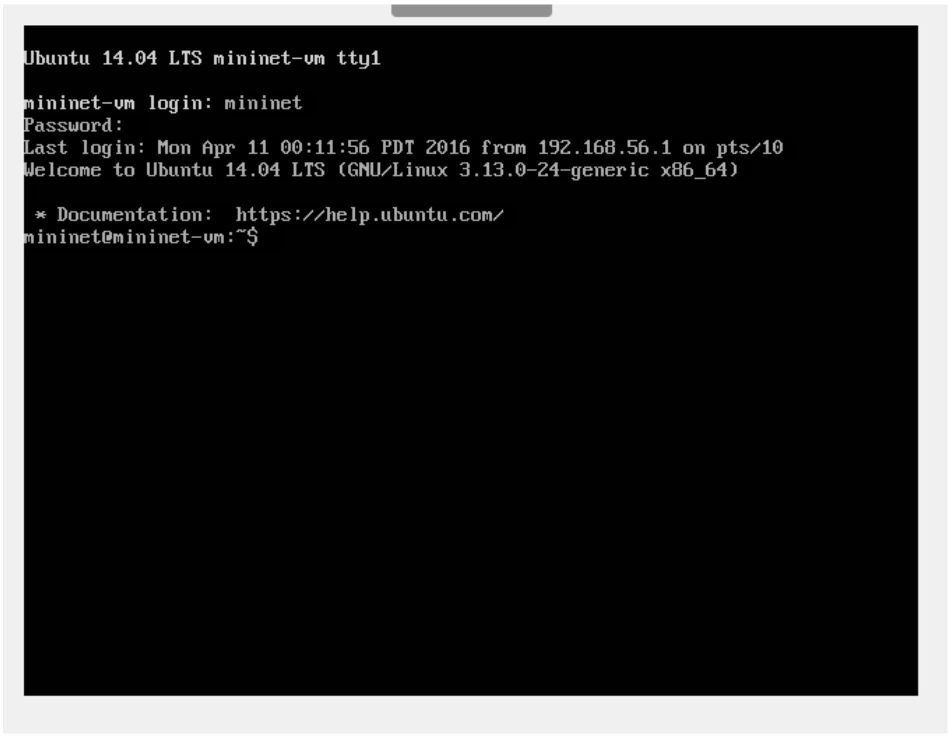
USB
USB Controller: OHCI
Device Filters: 0 (0 active)

Shared folders
None

Description

Finish VM Setup:

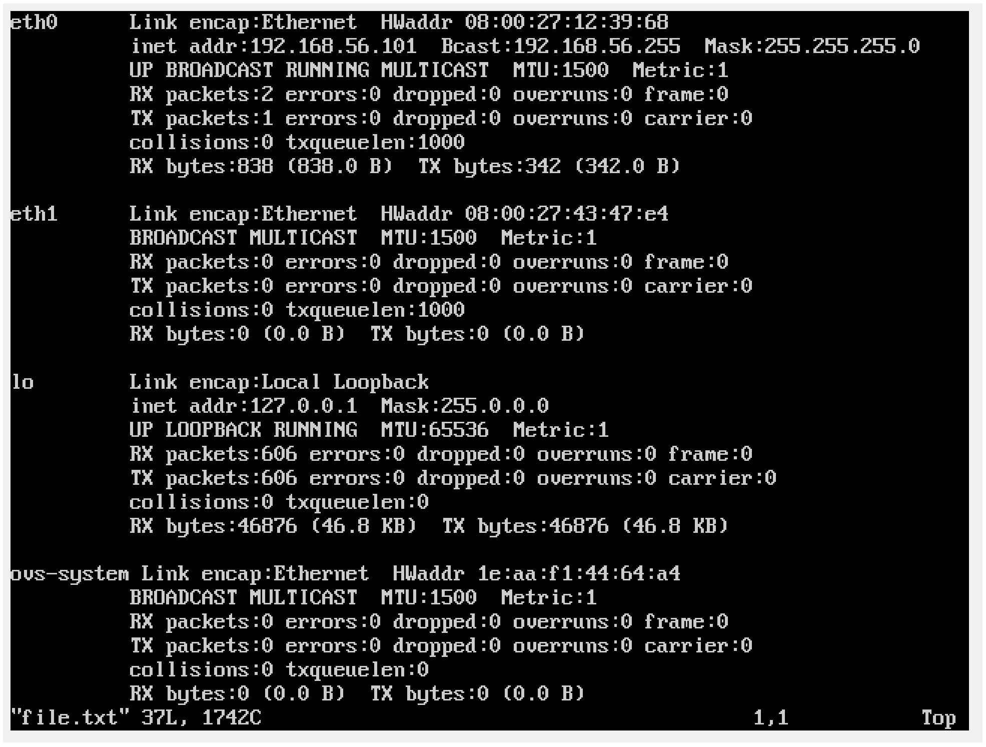
Step 3: I selected VM -> Settings tab -> Network -> Adapter 2. I “Enabled adapter” box and attached to it “host-only-network”. Started the VM and logged into the VM console window using ‘mininet’ as the username and password.



```
Ubuntu 14.04 LTS mininet-vm tty1

mininet-vm login: mininet
Password:
Last login: Mon Apr 11 00:11:56 PDT 2016 from 192.168.56.1 on pts/10
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
mininet@mininet-vm:~$
```

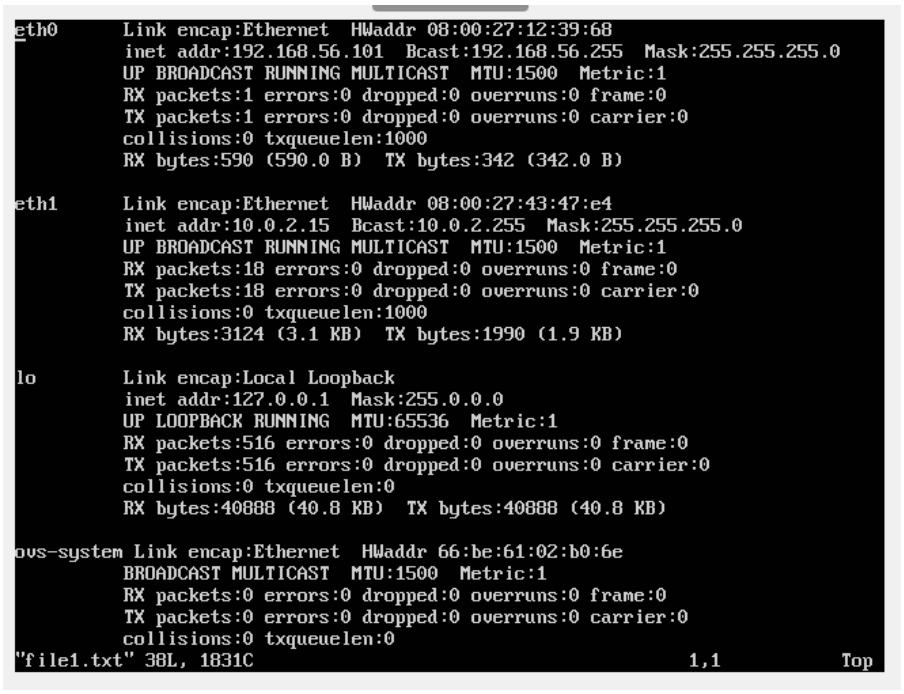
Access VM via SSH:

Step 4: From the VM console, typed ifcofig –a. The following three interface (eth0, eth1, lo) are shown:



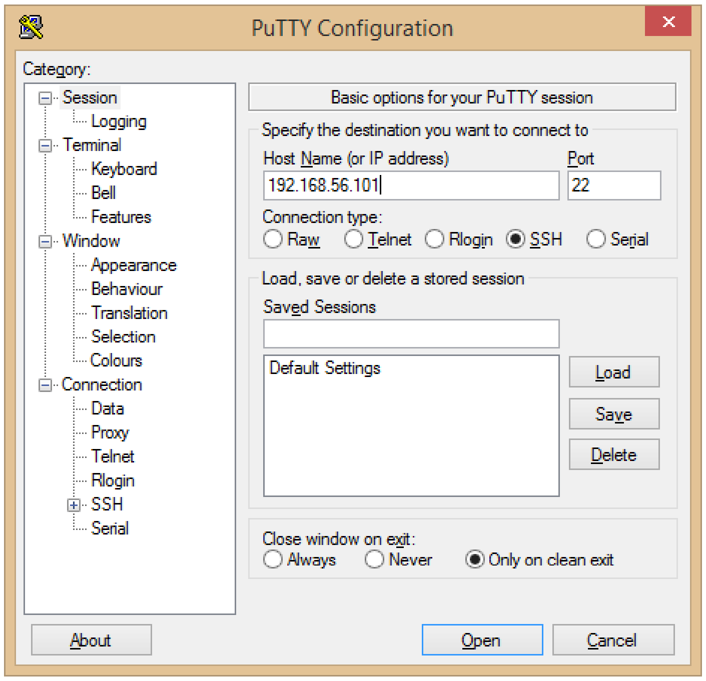
```
eth0      Link encap:Ethernet  HWaddr 08:00:27:12:39:68
          inet addr:192.168.56.101  Bcast:192.168.56.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:838 (838.0 B)  TX bytes:342 (342.0 B)

eth1      Link encap:Ethernet  HWaddr 08:00:27:43:47:e4
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:606 errors:0 dropped:0 overruns:0 frame:0
          TX packets:606 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:46876 (46.8 KB)  TX bytes:46876 (46.8 KB)

ovs-system Link encap:Ethernet  HWaddr 1e:aa:f1:44:64:a4
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
"file.txt" 37L, 1742C                                         1,1           Top
```

Step 5: As can be seen above eth1 doesn't have any IP address, It can be assigned using sudo dhclient eth1 and we get:



```
eth0      Link encap:Ethernet  HWaddr 08:00:27:12:39:68
          inet addr:192.168.56.101  Bcast:192.168.56.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:590 (590.0 B)  TX bytes:342 (342.0 B)

eth1      Link encap:Ethernet  HWaddr 08:00:27:43:47:e4
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:18 errors:0 dropped:0 overruns:0 frame:0
          TX packets:18 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3124 (3.1 KB)  TX bytes:1990 (1.9 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:516 errors:0 dropped:0 overruns:0 frame:0
          TX packets:516 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:40888 (40.8 KB)  TX bytes:40888 (40.8 KB)

ovs-system Link encap:Ethernet  HWaddr 66:be:61:02:b0:6e
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
"file1.txt" 38L, 1831C                                        1,1           Top
```

Step 6: I installed Xming and Putty and put IP address of the host-only network which is 192.168.56.101 in the Host Name in Putty shown below and SSH'd to these network.



Specify the destination you want to connect to

Host Name (or IP address): 192.168.56.101

Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

Close window on exit: Always Never [✓ Only on clean exit]

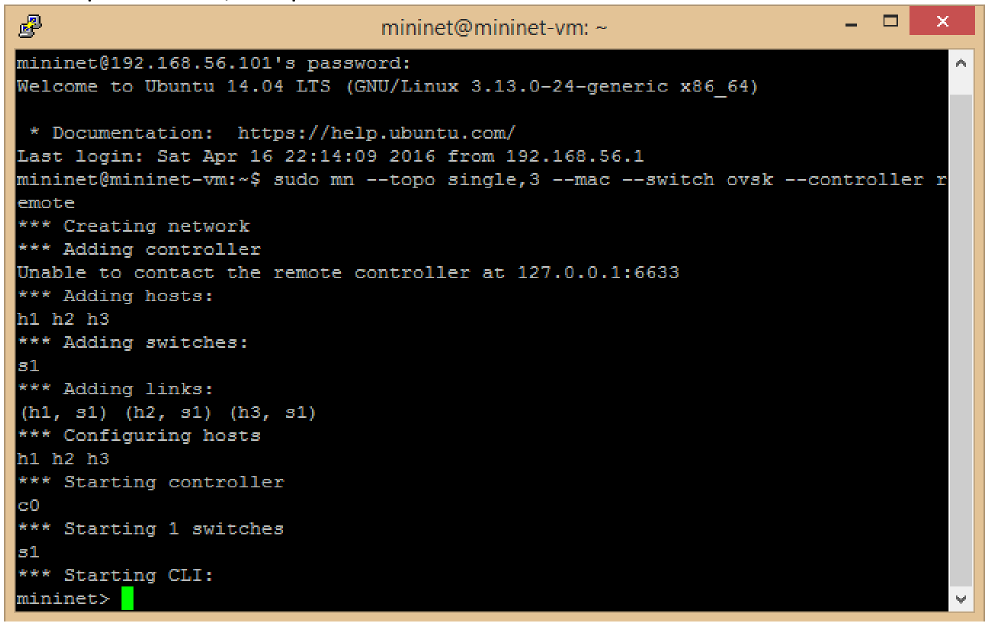
Step 7: Enabled the X11 forwarding by starting Xming first then clicking putty -> Connection -> SSH -> X11 then clicking on Forwarding -> Enable X11 Forwarding.

Start Network:

Step 8: We create the network given in the tutorial in the VM in an SSH terminal by entering,

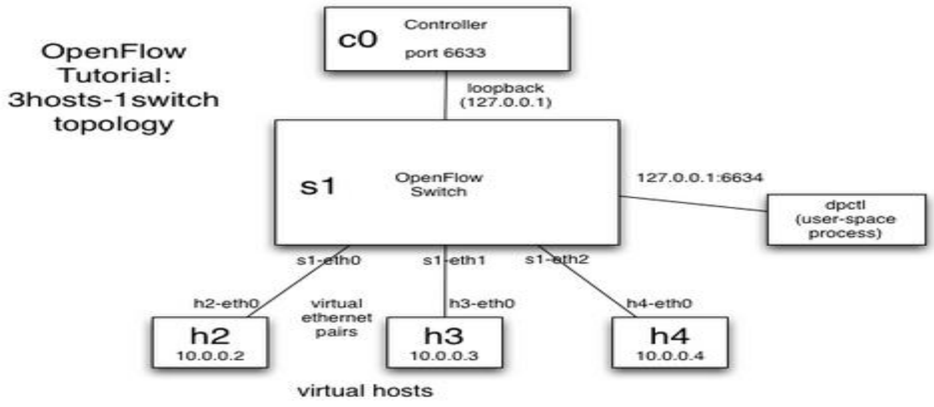
```
sudo mn --topo single,3 --mac --switch ovsk --controller remote
```

this creates 3 hosts and a switch and an OpenFlow controller. As given in the tutorial it tells the Mininet to start up a 3-host, single-(openvSwitch-based)switch topology, set the MAC address of each host equal to its IP, and point to a remote controller which defaults to the localhost.



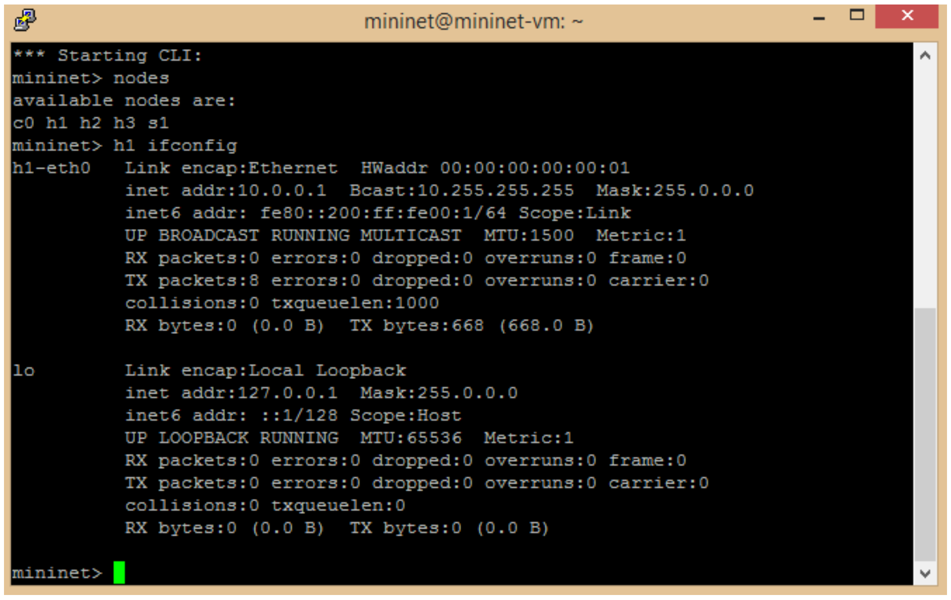
```
mininet@192.168.56.101's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Sat Apr 16 22:14:09 2016 from 192.168.56.1
mininet@mininet-vm:~$ sudo mn --topo single,3 --mac --switch ovsk --controller r
emote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

The topology it just created is:



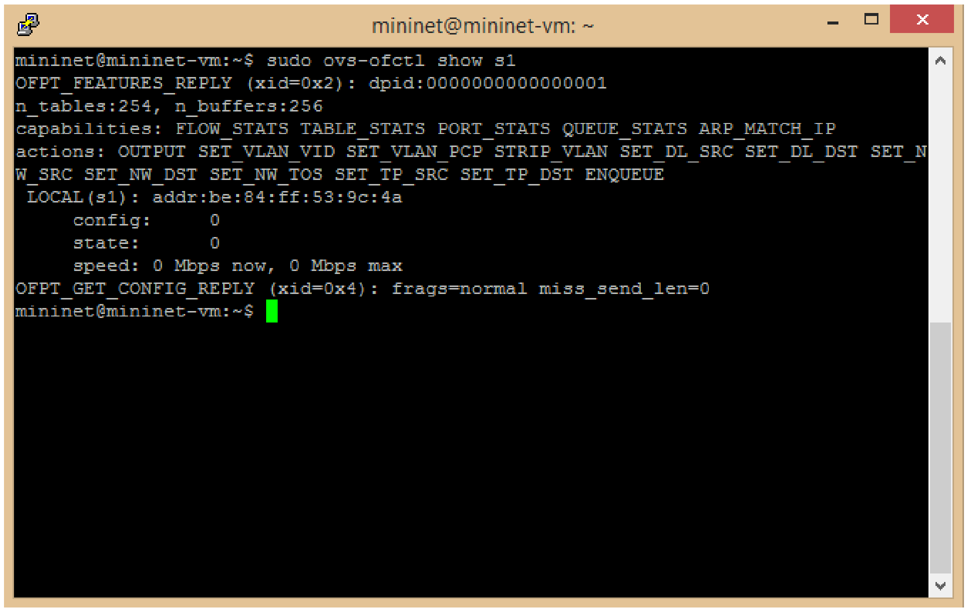
Mininet Brief Intro:

Step 9: Executed Mininet - specific commands like – nodes, h1 ifconfig (gives the IP of host 1), xterm h1 h2, sudo mn –c (clears residual state or processes)

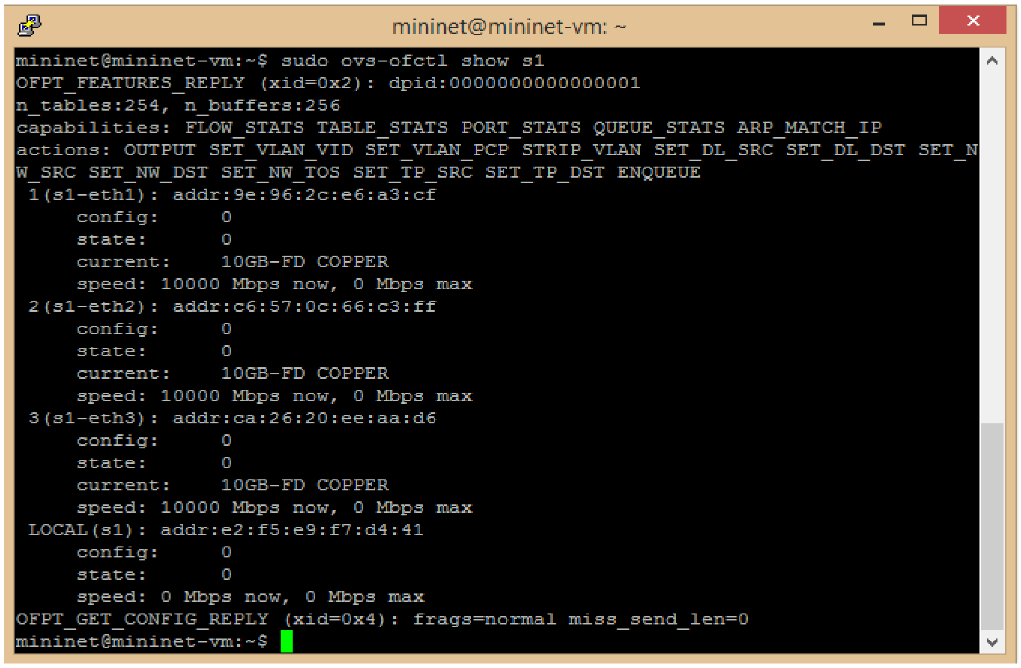

mininet@mininet-vm: ~

```
*** Starting CLI:
mininet> nodes
available nodes are:
c0 h1 h2 h3 s1
mininet> h1 ifconfig
h1-eth0   Link encap:Ethernet  HWaddr 00:00:00:00:00:01
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::200:ff:fe00:1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:668 (668.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet> 
```

Step 10: Next we enable visibility and control over a single switch's low table using the command `sudo ovs-ofctl show s1`. As given in the tutorial the 'show' command connects to the switch and dumps out its port state and capabilities.

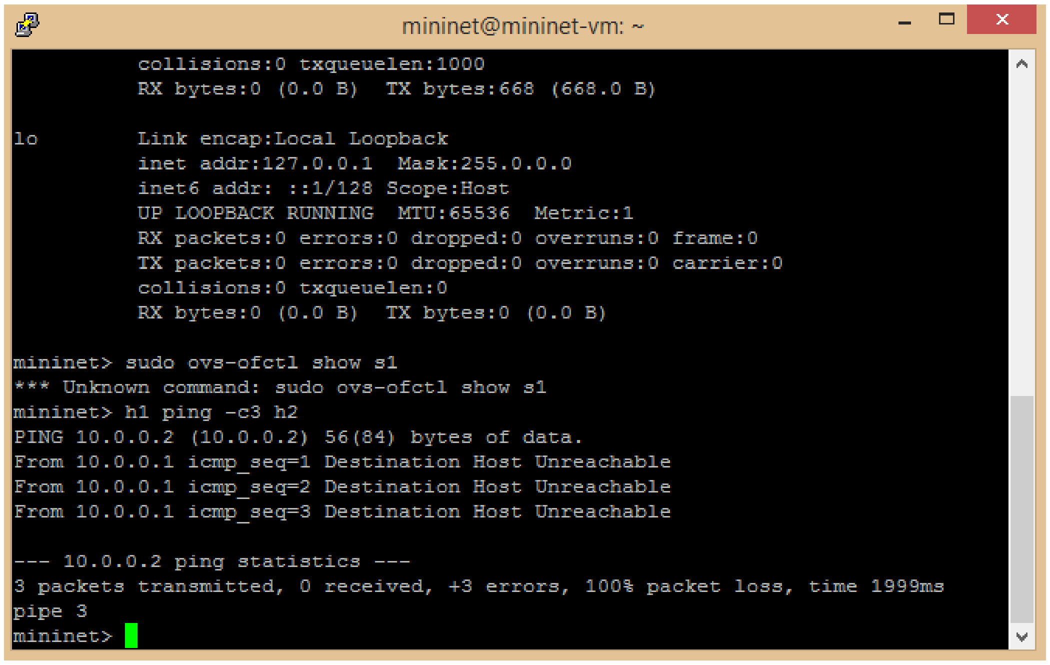

mininet@mininet-vm: ~

```
mininet@mininet-vm:~$ sudo ovs-ofctl show s1
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000000000001
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST SET_N
W_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST ENQUEUE
 LOCAL(s1): addr:be:84:ff:53:9c:4a
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
mininet@mininet-vm:~$ 
```



```
mininet@mininet-vm:~$ sudo ovs-ofctl show s1
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000000000001
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST SET_N
W_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST ENQUEUE
 1(s1-eth1): addr:9e:96:2c:e6:a3:cf
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s1-eth2): addr:c6:57:0c:66:c3:ff
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 3(s1-eth3): addr:ca:26:20:ee:aa:d6
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s1): addr:e2:f5:e9:f7:d4:41
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
mininet@mininet-vm:~$
```

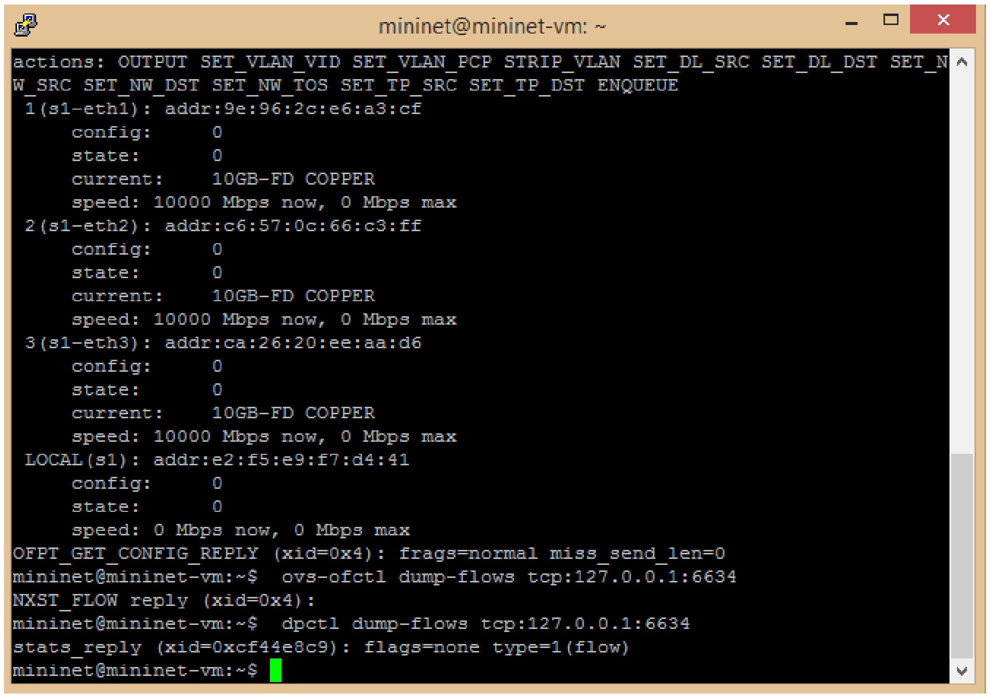
Step 11: Now we try to ping h2 from h1 using the command h1 `ping -c3 h2`. As given in the tutorial host h2 is automatically replaced with its IP address in the Mininet console. As seen from the console below 3 packets are transmitted in error, this is because the switch flow table is empty. Besides that, there is no controller connected to the switch and therefore the switch doesn't know what to do with incoming traffic, leading to ping failure.



```
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:668 (668.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet> sudo ovs-ofctl show s1
*** Unknown command: sudo ovs-ofctl show s1
mininet> h1 ping -c3 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 1999ms
pipe 3
mininet>
```

Accessing remote OVS instances or the Standard reference switch:

Step 12: Connected to a passive TCP port using the command `dpctl dump-flows tcp: 127.0.0.1:6634.`



```
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST SET_N
W_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST ENQUEUE
 1(s1-eth1): addr:9e:96:2c:e6:a3:cf
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s1-eth2): addr:c6:57:0c:66:c3:ff
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 3(s1-eth3): addr:ca:26:20:ee:aa:d6
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s1): addr:e2:f5:e9:f7:d4:41
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
mininet@mininet-vm:~$  ovs-ofctl dump-flows tcp:127.0.0.1:6634
NXST_FLOW reply (xid=0x4):
mininet@mininet-vm:~$  dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0xcf44e8c9): flags=none type=1(flow)
mininet@mininet-vm:~$
```

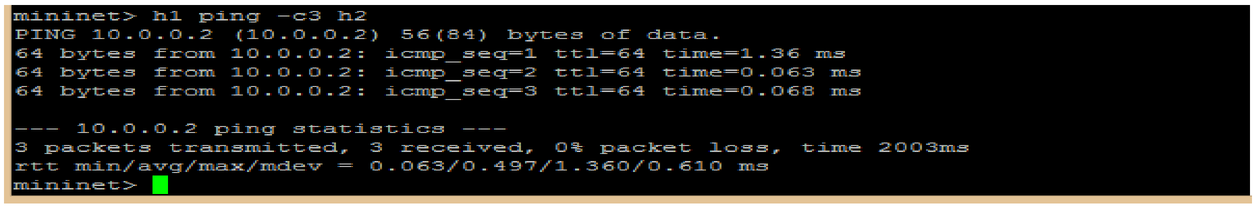
Ping Test:

Step 13: Next we do:

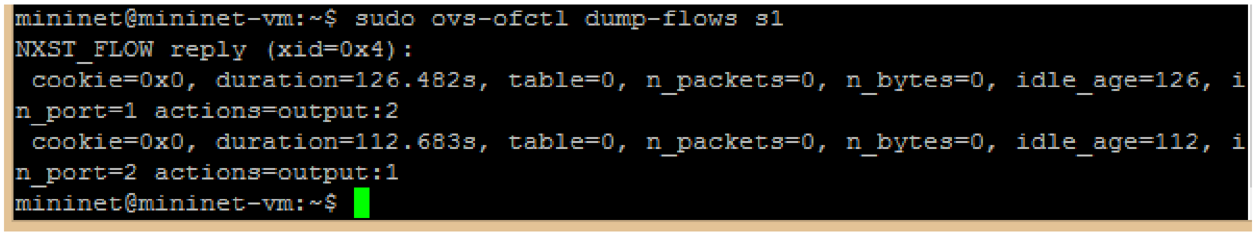
```
sudo ovs-ofctl add-flow s1 in_port=1,actions=output:2
sudo ovs-ofctl add-flow s1 in_port=2,actions=output:1
```

This will forward packets coming at port 1 to port 2 and vice-versa.

Step 14: The above step is verified by checking the flow table using sudo `ovs-ofctl dump-flows s1` and typing `h1 ping -c3 h2 in the Mininet console.`



```
mininet> h1 ping -c3 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.36 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.063 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.068 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.063/0.497/1.360/0.610 ms
mininet>
```



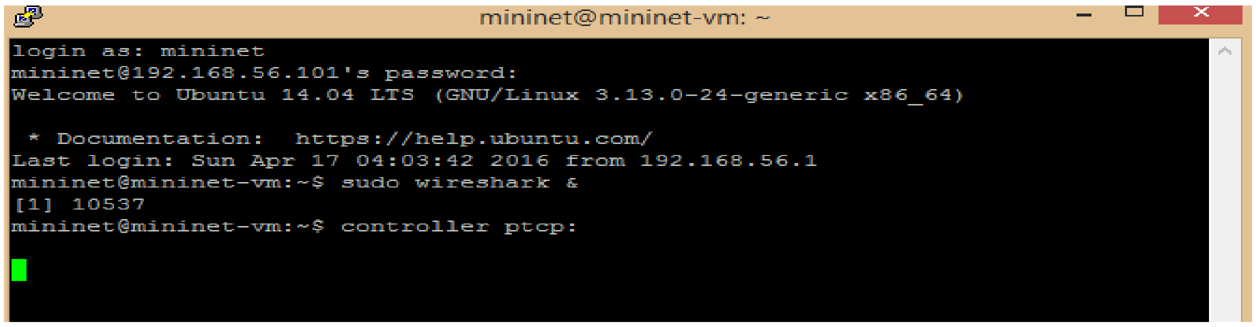
```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=126.482s, table=0, n_packets=0, n_bytes=0, idle_age=126, i
n_port=1 actions=output:2
 cookie=0x0, duration=112.683s, table=0, n_packets=0, n_bytes=0, idle_age=112, i
n_port=2 actions=output:1
mininet@mininet-vm:~$
```

Wireshark:

Step 15: Start the Wireshark using `sudo wireshark &`
In the filter we type ‘of’.

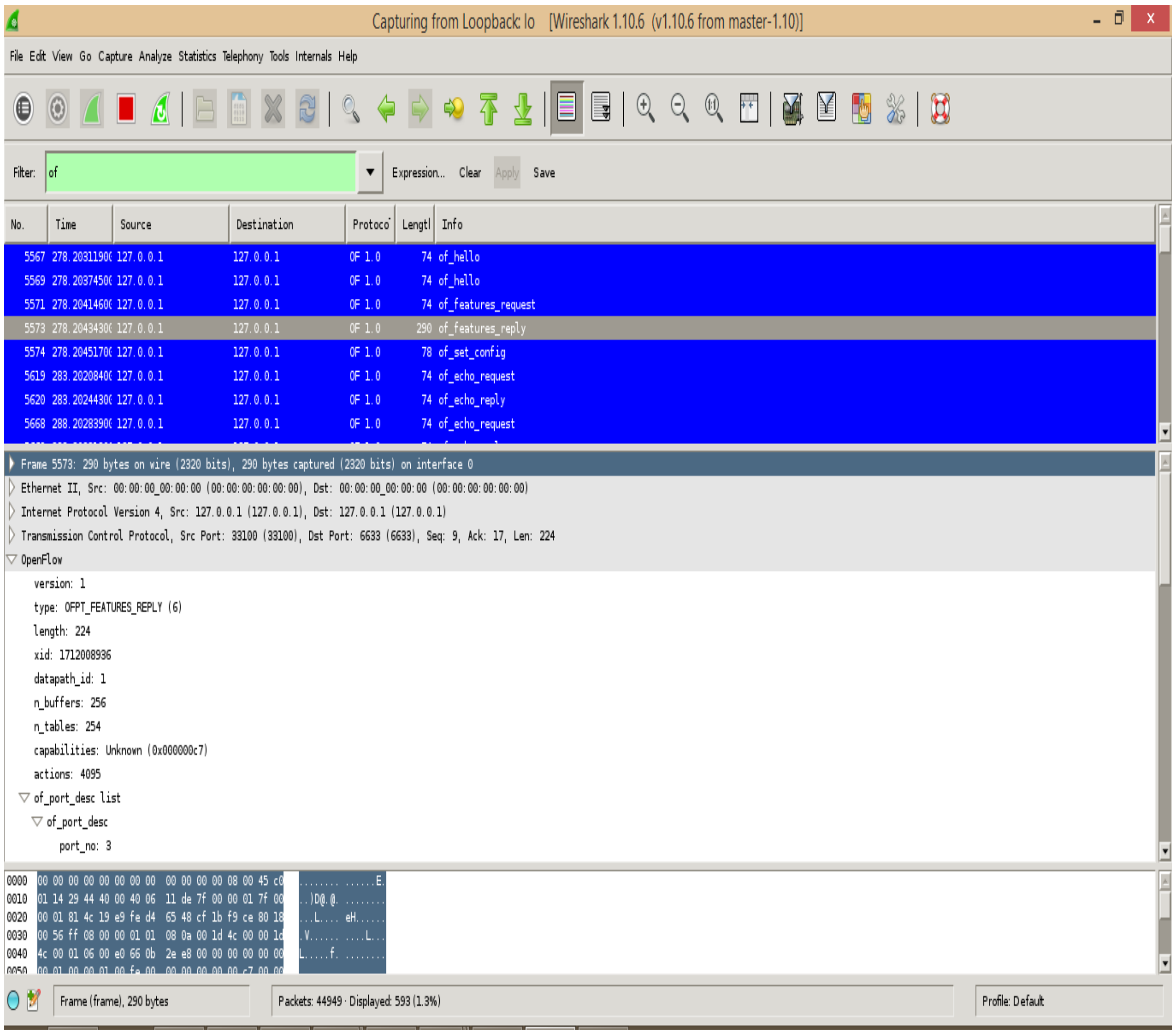
Start Controller and view Startup messages in Wireshark:

Step 16: Next we start the OpenFlow reference controller using: `controller ptcp`



```
login as: mininet
mininet@192.168.56.101's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Sun Apr 17 04:03:42 2016 from 192.168.56.1
mininet@mininet-vm:~$ sudo wireshark &
[1] 10537
mininet@mininet-vm:~$ controller ptcp:
```

The Wireshark shows messages as below. As described in the tutorial there are of_hello, of_features_request, of_features_reply and of_set_config.


Capturing from Loopback: lo [Wireshark 1.10.6 (v1.10.6 from master-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: of
Expression... Clear Apply Save

```
No.   Time        Source     Destination  Protocol  Length  Info
5567  278.2031190 127.0.0.1  127.0.0.1    OF 1.0    74      of_hello
5569  278.2037450 127.0.0.1  127.0.0.1    OF 1.0    74      of_hello
5571  278.2041460 127.0.0.1  127.0.0.1    OF 1.0    74      of_features_request
5573  278.2043430 127.0.0.1  127.0.0.1    OF 1.0    290     of_features_reply
5574  278.2045170 127.0.0.1  127.0.0.1    OF 1.0    78      of_set_config
5619  283.2020840 127.0.0.1  127.0.0.1    OF 1.0    74      of_echo_request
5620  283.2024430 127.0.0.1  127.0.0.1    OF 1.0    74      of_echo_reply
5668  288.2028390 127.0.0.1  127.0.0.1    OF 1.0    74      of_echo_request
```

```
Frame 5573: 290 bytes on wire (2320 bits), 290 bytes captured (2320 bits) on interface 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
Transmission Control Protocol, Src Port: 33100 (33100), Dst Port: 6633 (6633), Seq: 9, Ack: 17, Len: 224
OpenFlow
  version: 1
  type: OFPT_FEATURES_REPLY (6)
  length: 224
  xid: 1712008936
  datapath_id: 1
  n_buffers: 256
  n_tables: 254
  capabilities: Unknown (0x000000c7)
  actions: 4095
  of_port_desc list
    of_port_desc
      port_no: 3
```

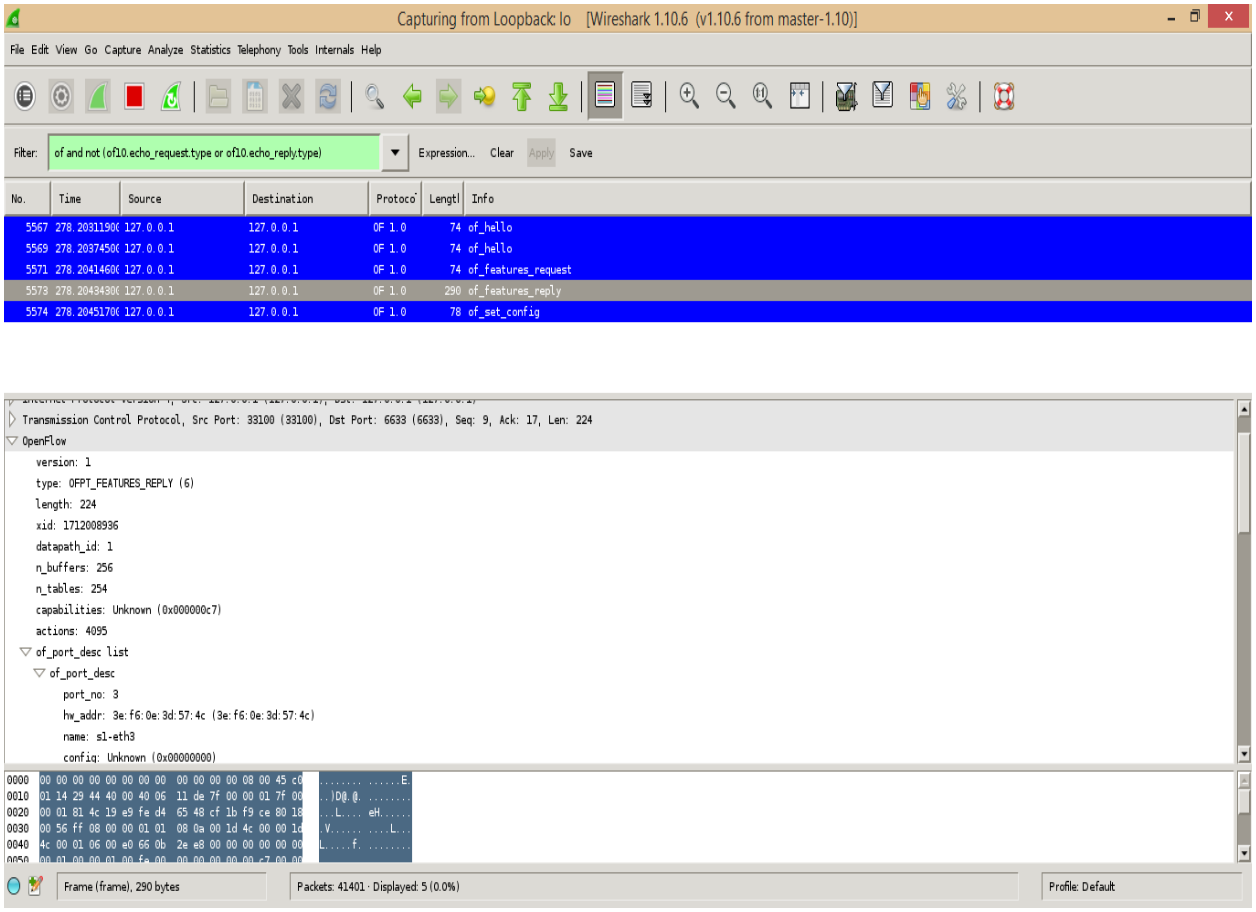
```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 c0   ........ ......E.
0010  01 14 29 44 40 00 40 06  11 de 7f 00 00 01 7f 00   ..)D@.@. ........
0020  00 01 81 4c 19 e9 fe d4  65 48 cf 1b f9 ce 80 18   ...L.... eH......
0030  00 56 ff 08 00 00 01 01  08 0a 00 1d 4c 00 00 1d   .V...... ....L...
0040  4c 00 01 06 00 e0 66 0b  2e e8 00 00 00 00 00 00   L.....f. ........
```

Frame (frame), 290 bytes | Packets: 44949 · Displayed: 593 (1.3%) | Profile: Default

<u>View OpenFlow Messages for Ping:</u>

Step 17: Next we type the following the Wireshark filter:

```
of and not (of10.echo_request.type or of10.echo_reply.type)
```


Capturing from Loopback: lo   [Wireshark 1.10.6 (v1.10.6 from master-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save

```
Filter: of and not (of10.echo_request.type or of10.echo_reply.type)

No.   Time         Source      Destination  Protoco  Lengtl  Info
 5567 278.2031190( 127.0.0.1   127.0.0.1    OF 1.0       74  of_hello
 5569 278.2037450( 127.0.0.1   127.0.0.1    OF 1.0       74  of_hello
 5571 278.2041460( 127.0.0.1   127.0.0.1    OF 1.0       74  of_features_request
 5573 278.2043430( 127.0.0.1   127.0.0.1    OF 1.0      290  of_features_reply
 5574 278.2045170( 127.0.0.1   127.0.0.1    OF 1.0       78  of_set_config
```

```
▷ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▷ Transmission Control Protocol, Src Port: 33100 (33100), Dst Port: 6633 (6633), Seq: 9, Ack: 17, Len: 224
▽ OpenFlow
    version: 1
    type: OFPT_FEATURES_REPLY (6)
    length: 224
    xid: 1712008936
    datapath_id: 1
    n_buffers: 256
    n_tables: 254
    capabilities: Unknown (0x000000c7)
    actions: 4095
  ▽ of_port_desc list
    ▽ of_port_desc
        port_no: 3
        hw_addr: 3e:f6:0e:3d:57:4c (3e:f6:0e:3d:57:4c)
        name: s1-eth3
        config: Unknown (0x00000000)
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 c0   ........ ......E.
0010  01 14 29 44 40 00 40 06  11 de 7f 00 00 01 7f 00   ..)D@.@. ........
0020  00 01 81 4c 19 e9 fe d4  65 48 cf 1b f9 ce 80 18   ...L.... eH......
0030  00 56 ff 08 00 00 01 01  08 0a 00 1d 4c 00 00 1d   .V...... ....L...
0040  4c 00 01 06 00 e0 66 0b  2e e8 00 00 00 00 00 00   L.....f. ........
0050  00 01 00 00 01 00 fe 00  00 00 00 00 00 c7 00 00
```

Frame (frame), 290 bytes | Packets: 41401 · Displayed: 5 (0.0%) | Profile: Default

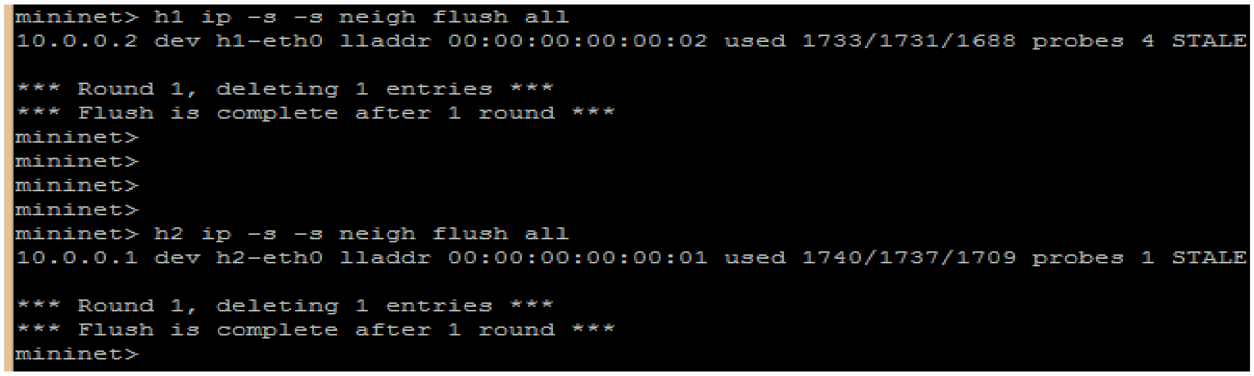
Step 18: View OpenFlow Messages for Ping:

I type the following:

```
sudo ovs-ofctl del-flows s1
```

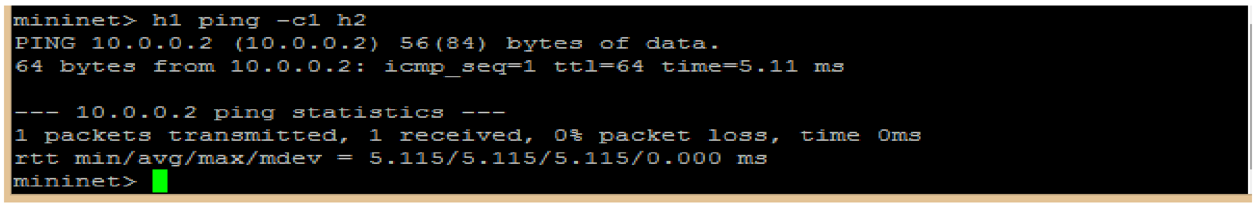
I also clean up the ARP cache on both hosts using:

```
h1 ip -s -s neigh flush all
h2 ip -s -s neigh flush all
```



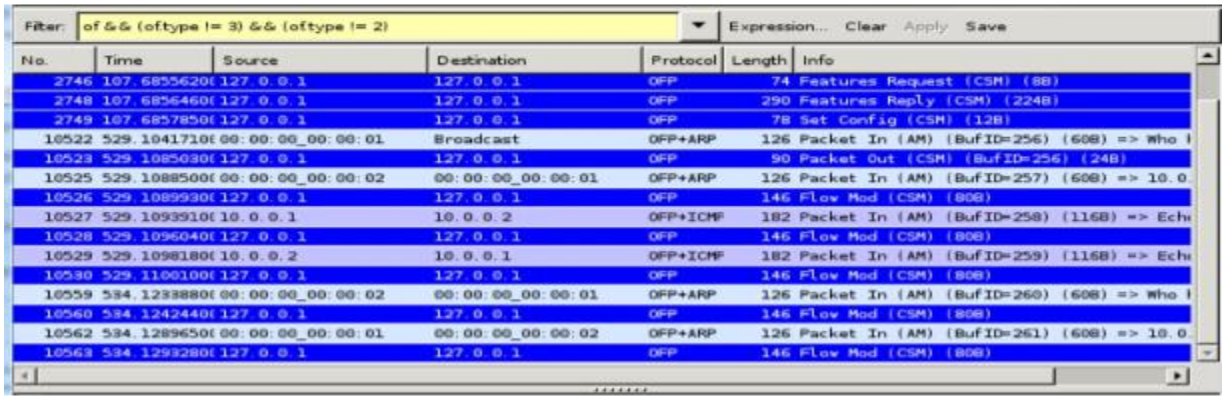
```
mininet> h1 ip -s -s neigh flush all
10.0.0.2 dev h1-eth0 lladdr 00:00:00:00:00:02 used 1733/1731/1688 probes 4 STALE

*** Round 1, deleting 1 entries ***
*** Flush is complete after 1 round ***
mininet>
mininet>
mininet>
mininet>
mininet> h2 ip -s -s neigh flush all
10.0.0.1 dev h2-eth0 lladdr 00:00:00:00:00:01 used 1740/1737/1709 probes 1 STALE

*** Round 1, deleting 1 entries ***
*** Flush is complete after 1 round ***
mininet>
```

Next ping `h1 ping -c1 h2` from the Mininet console

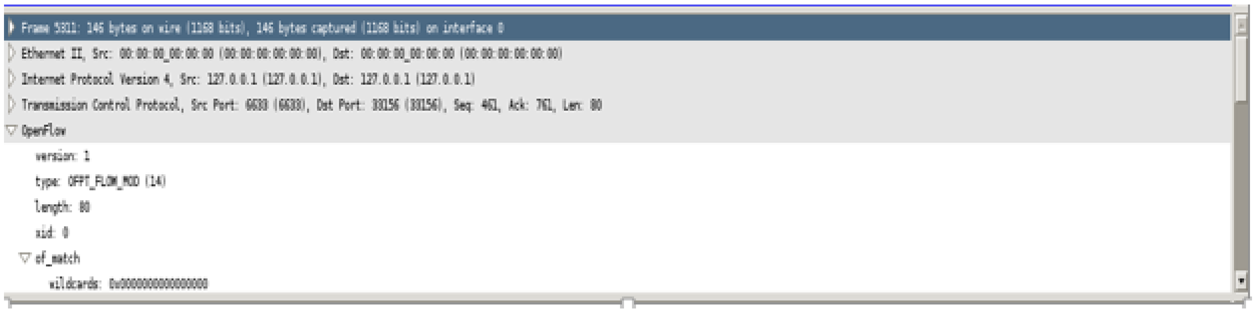


```
mininet> h1 ping -c1 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=5.11 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 5.115/5.115/5.115/0.000 ms
mininet>
```

With the above command the Wireshark window will see a number of new messages as shown below:

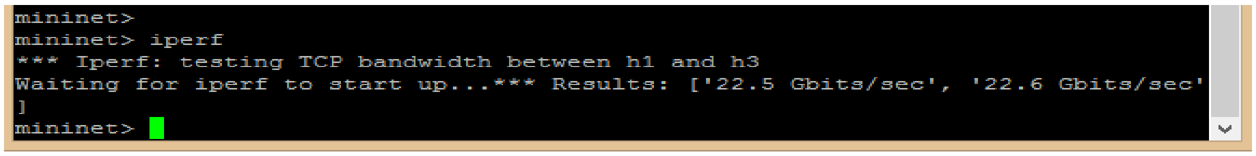


```
Filter: of && (oftype != 3) && (oftype != 2)                Expression...  Clear  Apply  Save

No.    Time             Source                Destination           Protocol  Length Info
  2746 107.6855620(127.0.0.1               127.0.0.1              OFP          74 Features Request (CSM) (8B)
  2748 107.6856460(127.0.0.1               127.0.0.1              OFP         290 Features Reply (CSM) (224B)
  2749 107.6857850(127.0.0.1               127.0.0.1              OFP          78 Set Config (CSM) (12B)
 10522 529.1041710(00:00:00_00:00:01       Broadcast              OFP+ARP     126 Packet In (AM) (BufID=256) (60B) => Who I
 10523 529.1085030(127.0.0.1               127.0.0.1              OFP          90 Packet Out (CSM) (BufID=256) (24B)
 10525 529.1088500(00:00:00_00:00:02       00:00:00_00:00:01      OFP+ARP     126 Packet In (AM) (BufID=257) (60B) => 10.0.
 10526 529.1089930(127.0.0.1               127.0.0.1              OFP         146 Flow Mod (CSM) (80B)
 10527 529.1093910(10.0.0.1                10.0.0.2               OFP+ICMP    182 Packet In (AM) (BufID=258) (116B) => Ech
 10528 529.1096040(127.0.0.1               127.0.0.1              OFP         146 Flow Mod (CSM) (80B)
 10529 529.1098180(10.0.0.2                10.0.0.1               OFP+ICMP    182 Packet In (AM) (BufID=259) (116B) => Ech
 10530 529.1100100(127.0.0.1               127.0.0.1              OFP         146 Flow Mod (CSM) (80B)
 10559 534.1233880(00:00:00_00:00:02       00:00:00_00:00:01      OFP+ARP     126 Packet In (AM) (BufID=260) (60B) => Who I
 10560 534.1242440(127.0.0.1               127.0.0.1              OFP         146 Flow Mod (CSM) (80B)
 10562 534.1289650(00:00:00_00:00:01       00:00:00_00:00:02      OFP+ARP     126 Packet In (AM) (BufID=261) (60B) => 10.0.
 10563 534.1293280(127.0.0.1               127.0.0.1              OFP         146 Flow Mod (CSM) (80B)
```

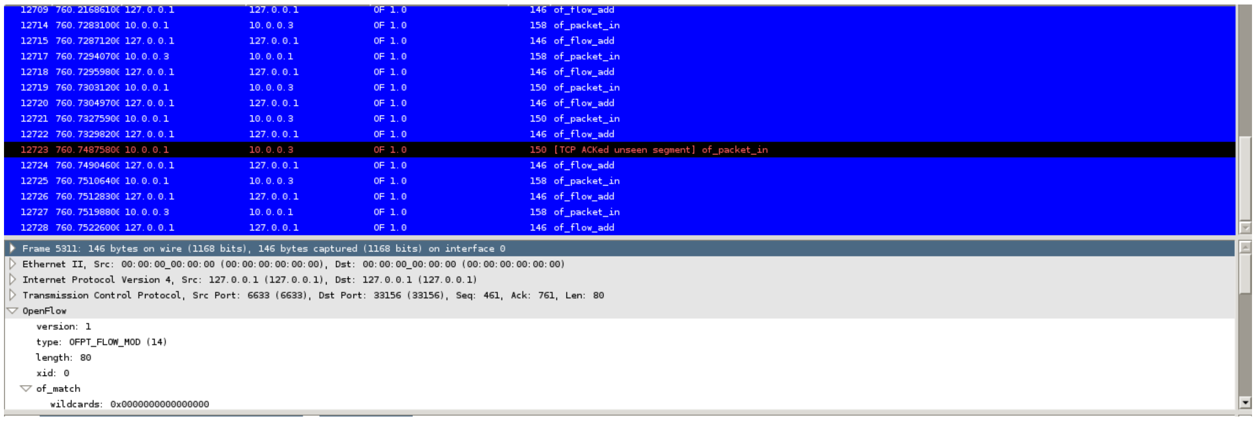


```
▷ Frame 5311: 146 bytes on wire (1168 bits), 146 bytes captured (1168 bits) on interface 0
▷ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▷ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▷ Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 33156 (33156), Seq: 461, Ack: 761, Len: 80
▽ OpenFlow
    version: 1
    type: OFPT_FLOW_MOD (14)
    length: 80
    xid: 0
  ▽ of_match
      wildcards: 0x0000000000000000
```

<u>Benchmark Controller w/iperf:</u> (Flow based switch)

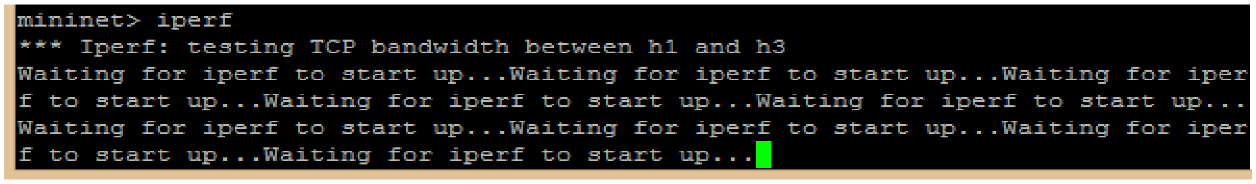


```
mininet>
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
Waiting for iperf to start up...*** Results: ['22.5 Gbits/sec', '22.6 Gbits/sec'
]
mininet>
```



```
12709 760.2168610( 127.0.0.1        127.0.0.1        OF 1.0        146 of_flow_add
12714 760.7283100( 10.0.0.1         10.0.0.3         OF 1.0        158 of_packet_in
12715 760.7287120( 127.0.0.1        127.0.0.1        OF 1.0        146 of_flow_add
12717 760.7294070( 10.0.0.3         10.0.0.1         OF 1.0        158 of_packet_in
12718 760.7295980( 127.0.0.1        127.0.0.1        OF 1.0        146 of_flow_add
12719 760.7303120( 10.0.0.1         10.0.0.3         OF 1.0        150 of_packet_in
12720 760.7304970( 127.0.0.1        127.0.0.1        OF 1.0        146 of_flow_add
12721 760.7327590( 10.0.0.1         10.0.0.3         OF 1.0        150 of_packet_in
12722 760.7329820( 127.0.0.1        127.0.0.1        OF 1.0        146 of_flow_add
12723 760.7487580( 10.0.0.1         10.0.0.3         OF 1.0        150 [TCP ACKed unseen segment] of_packet_in
12724 760.7490460( 127.0.0.1        127.0.0.1        OF 1.0        146 of_flow_add
12725 760.7510640( 10.0.0.1         10.0.0.3         OF 1.0        158 of_packet_in
12726 760.7512830( 127.0.0.1        127.0.0.1        OF 1.0        146 of_flow_add
12727 760.7519880( 10.0.0.3         10.0.0.1         OF 1.0        158 of_packet_in
12728 760.7522600( 127.0.0.1        127.0.0.1        OF 1.0        146 of_flow_add

Frame 5311: 146 bytes on wire (1168 bits), 146 bytes captured (1168 bits) on interface 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 33156 (33156), Seq: 461, Ack: 761, Len: 80
OpenFlow
   version: 1
   type: OFPT_FLOW_MOD (14)
   length: 80
   xid: 0
 of_match
    wildcards: 0x0000000000000000
```

After exiting the mininet and starting the same mininet with the user-space switch using `sudo mn --topo single,3 --mac --controller remote --switch user` and running `iperf` in the mininet console again we get :

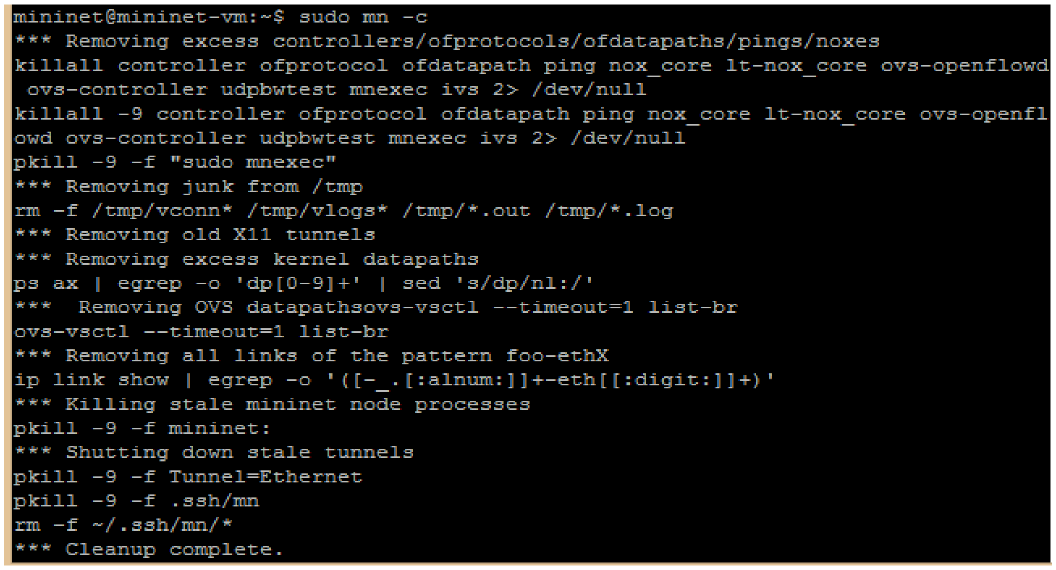


```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
Waiting for iperf to start up...Waiting for iperf to start up...Waiting for iper
f to start up...Waiting for iperf to start up...Waiting for iperf to start up...
Waiting for iperf to start up...Waiting for iperf to start up...Waiting for iper
f to start up...Waiting for iperf to start up...
```

Creating a Learning Switch:

Controller Choice: POX (Python)

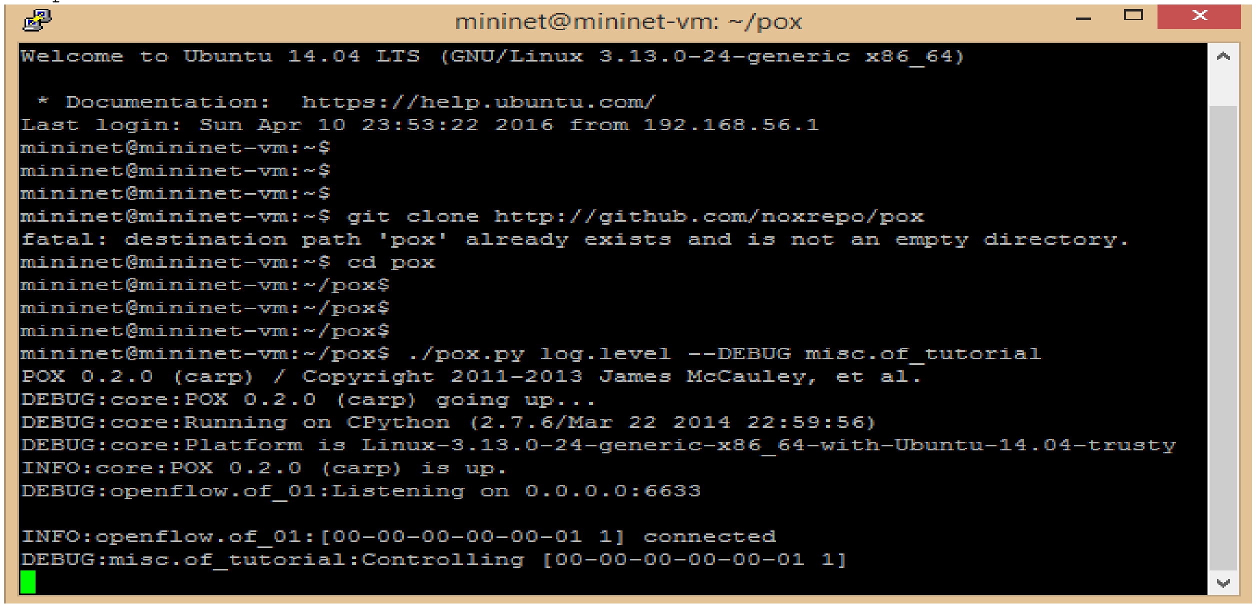
Here we kill the controller and also run sudo mn –c to make sure that everything is clean and using faster kernel switch.



```
mininet@mininet-vm:~$ sudo mn -c
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd
 ovs-controller udpbwtest mnexec ivs 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openfl
owd ovs-controller udpbwtest mnexec ivs 2> /dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
***  Removing OVS datapathsovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
```

Running the following commands which creates the topology and changes directory to pox to find that the controller is UP and running.

```
sudo mn --topo single,3 --mac --switch ovsk --controller remote
git clone http://github.com/noxrepo/pox
cd pox
```

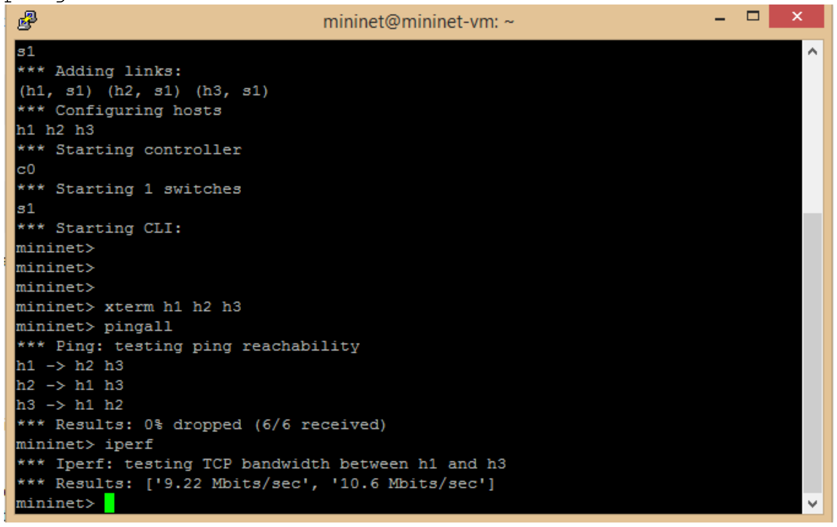


```
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Sun Apr 10 23:53:22 2016 from 192.168.56.1
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$ git clone http://github.com/noxrepo/pox
fatal: destination path 'pox' already exists and is not an empty directory.
mininet@mininet-vm:~$ cd pox
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$
mininet@mininet-vm:~/pox$ ./pox.py log.level --DEBUG misc.of_tutorial
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Mar 22 2014 22:59:56)
DEBUG:core:Platform is Linux-3.13.0-24-generic-x86_64-with-Ubuntu-14.04-trusty
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633

INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-01 1]
```

<u>Verify Hub Behavior with tcpdump:</u>

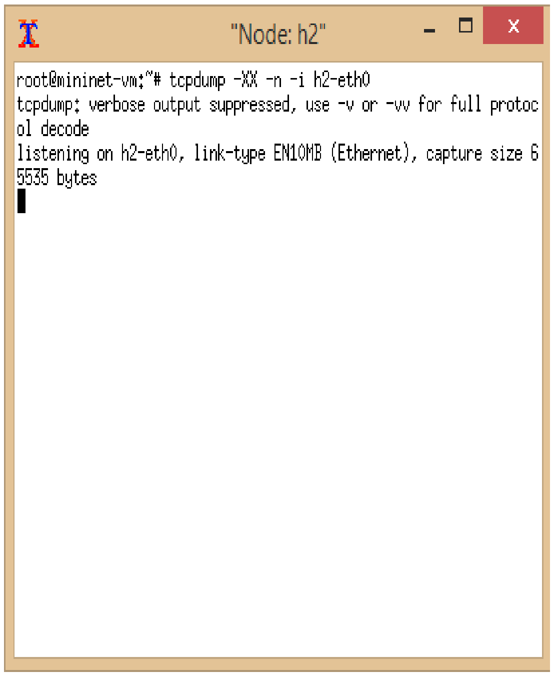
We start three xterm for three hosts h1, h2, h3 using command xterm h1, h2, h3. We then run the following tcpdump commands in the proper xterms:

```
tcpdump -XX -n -i h2-eth0
tcpdump -XX -n -i h3-eth0
ping -c1 10.0.0.2
```

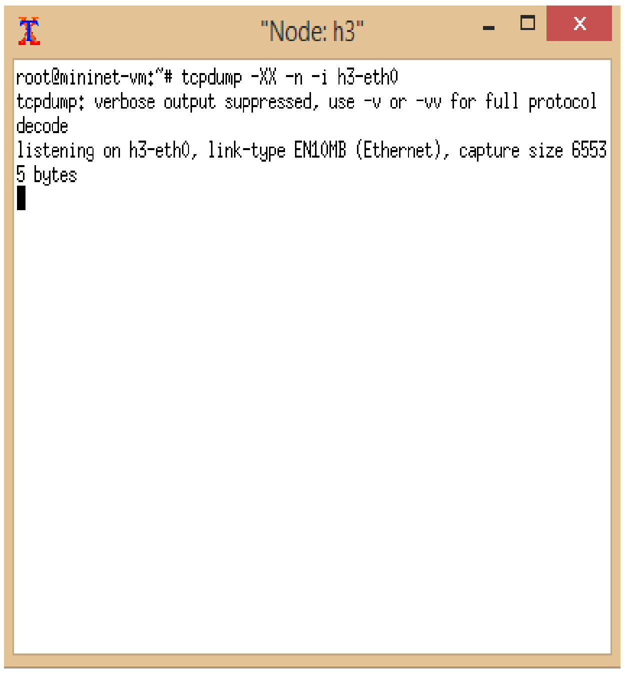


```
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
mininet>
mininet>
mininet> xterm h1 h2 h3
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['9.22 Mbits/sec', '10.6 Mbits/sec']
mininet>
```

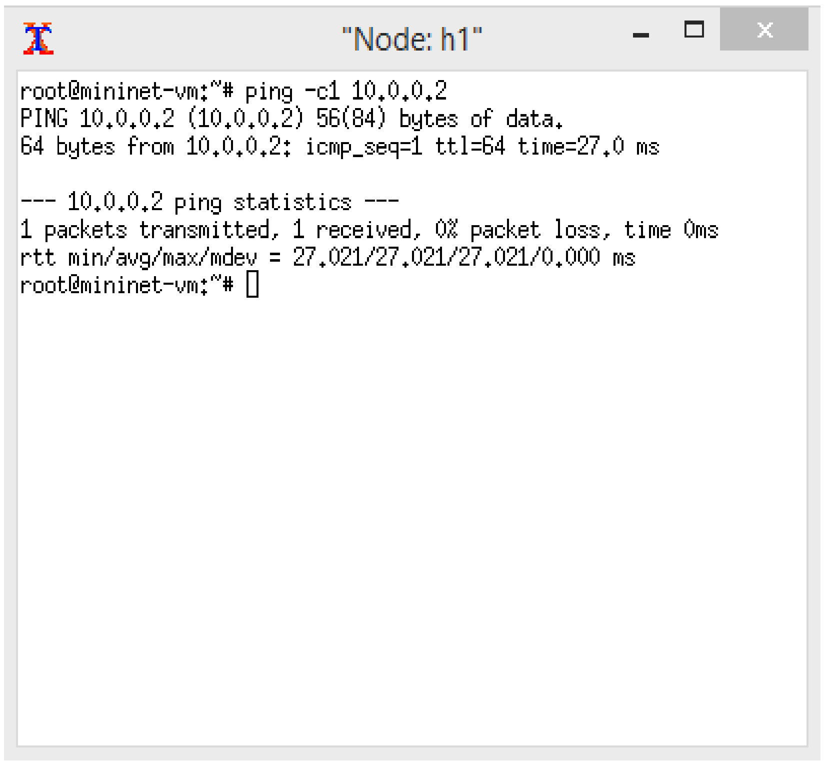
We get the following outputs:


"Node: h2"

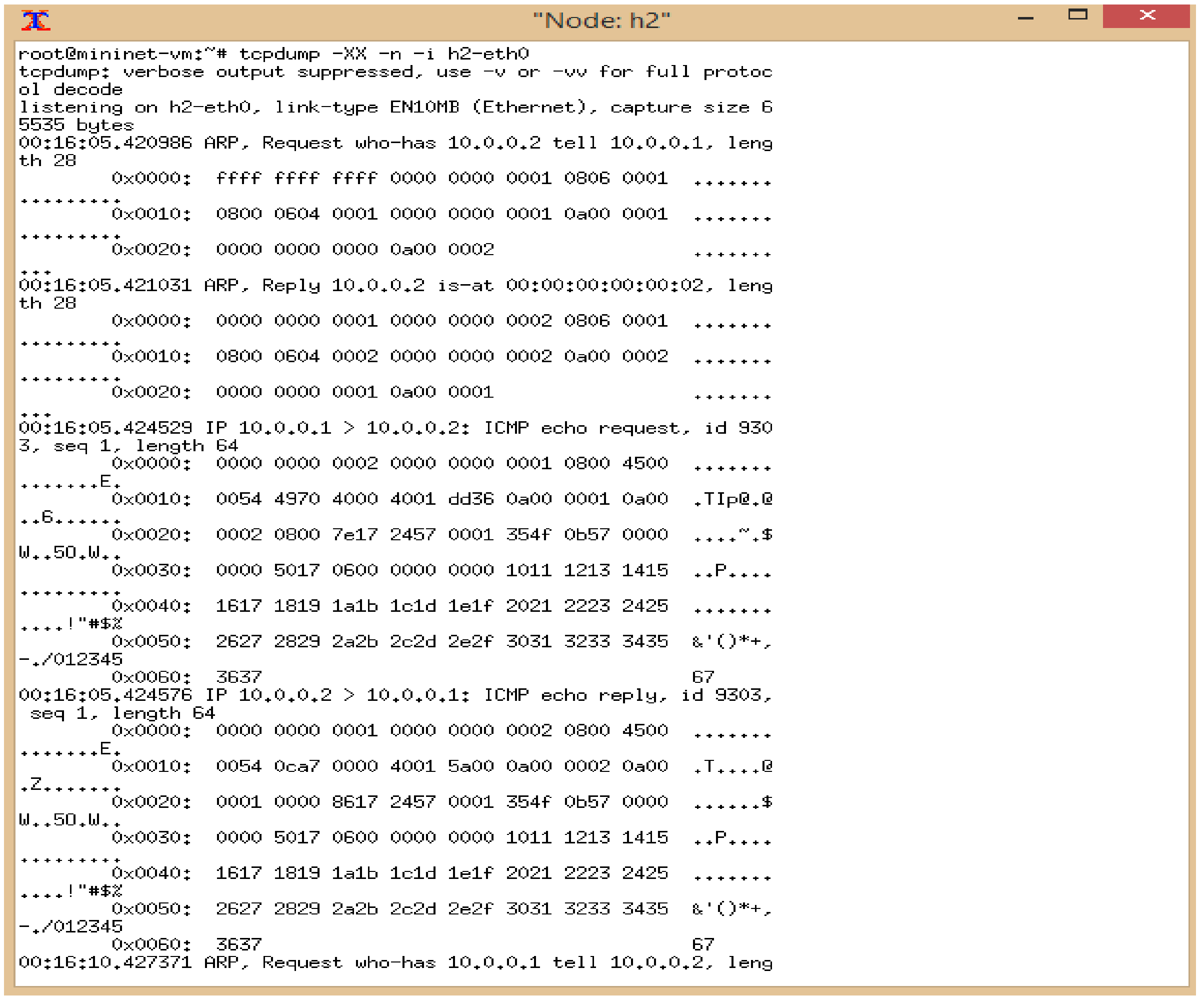
```
root@mininet-vm:~# tcpdump -XX -n -i h2-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protoc
ol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 6
5535 bytes
```


"Node: h3"

```
root@mininet-vm:~# tcpdump -XX -n -i h3-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol
decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 6553
5 bytes
```

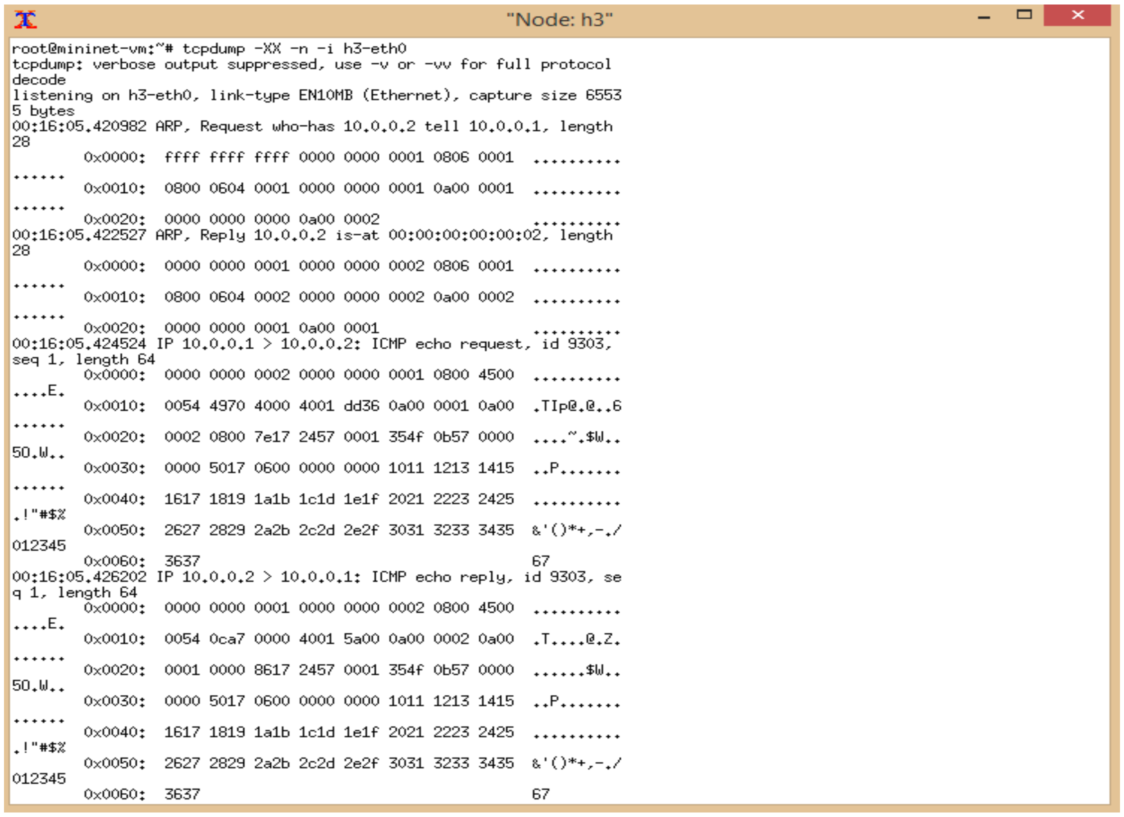

"Node: h1"

```
root@mininet-vm:~# ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=27.0 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 27.021/27.021/27.021/0.000 ms
root@mininet-vm:~# 
```

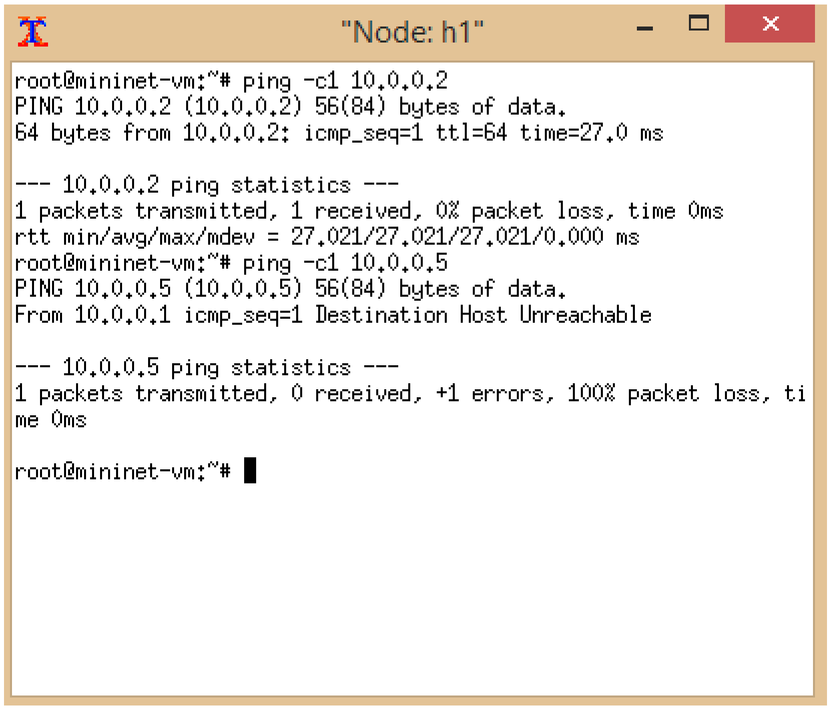


"Node: h2"

```
root@mininet-vm:~# tcpdump -XX -n -i h2-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protoc
ol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 6
5535 bytes
00:16:05.420986 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, leng
th 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  .......
.........
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  .......
.........
        0x0020:  0000 0000 0000 0a00 0002                 .......
...
00:16:05.421031 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, leng
th 28
        0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  .......
.........
        0x0010:  0800 0604 0002 0000 0000 0002 0a00 0002  .......
.........
        0x0020:  0000 0000 0001 0a00 0001                 .......
...
00:16:05.424529 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 930
3, seq 1, length 64
        0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  .......
.......E.
        0x0010:  0054 4970 4000 4001 dd36 0a00 0001 0a00  .TIp@.@
..6.......
        0x0020:  0002 0800 7e17 2457 0001 354f 0b57 0000  ....~.$
W..50.W..
        0x0030:  0000 5017 0600 0000 0000 1011 1213 1415  ..P....
.........
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  .......
.....!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,
-./012345
        0x0060:  3637                                     67
00:16:05.424576 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 9303,
 seq 1, length 64
        0x0000:  0000 0000 0001 0000 0000 0002 0800 4500  .......
.......E.
        0x0010:  0054 0ca7 0000 4001 5a00 0a00 0002 0a00  .T....@
.Z........
        0x0020:  0001 0000 8617 2457 0001 354f 0b57 0000  ......$
W..50.W..
        0x0030:  0000 5017 0600 0000 0000 1011 1213 1415  ..P....
.........
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  .......
.....!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,
-./012345
        0x0060:  3637                                     67
00:16:10.427371 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, leng
```

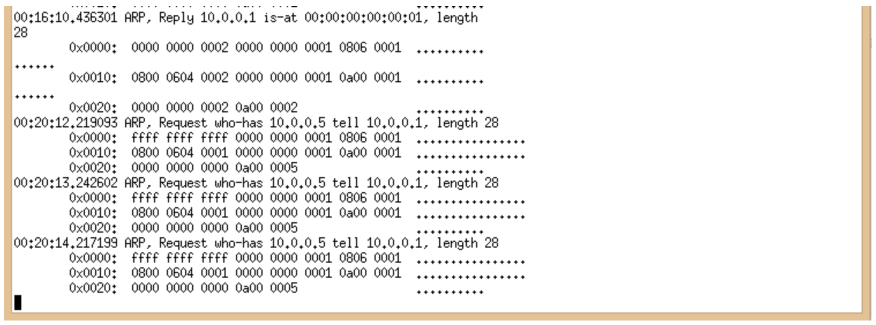


```
root@mininet-vm:~# tcpdump -XX -n -i h3-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol
decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 6553
5 bytes
00:16:05.420982 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length
28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  ..........
......
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ..........
......
        0x0020:  0000 0000 0000 0a00 0002                 ..........
00:16:05.422527 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length
28
        0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  ..........
......
        0x0010:  0800 0604 0002 0000 0000 0002 0a00 0002  ..........
......
        0x0020:  0000 0000 0001 0a00 0001                 ..........
00:16:05.424524 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 9303,
seq 1, length 64
        0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  ..........
....E.
        0x0010:  0054 4970 4000 4001 dd36 0a00 0001 0a00  .TIp@.@..6
......
        0x0020:  0002 0800 7e17 2457 0001 354f 0b57 0000  ....~.$W..
50.W..
        0x0030:  0000 5017 0600 0000 0000 1011 1213 1415  ..P.......
......
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ..........
.!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./
012345
        0x0060:  3637                                     67
00:16:05.426202 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 9303, se
q 1, length 64
        0x0000:  0000 0000 0001 0000 0000 0002 0800 4500  ..........
....E.
        0x0010:  0054 0ca7 0000 4001 5a00 0a00 0002 0a00  .T....@.Z.
......
        0x0020:  0001 0000 8617 2457 0001 354f 0b57 0000  ......$W..
50.W..
        0x0030:  0000 5017 0600 0000 0000 1011 1213 1415  ..P.......
......
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ..........
.!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./
012345
        0x0060:  3637                                     67
```

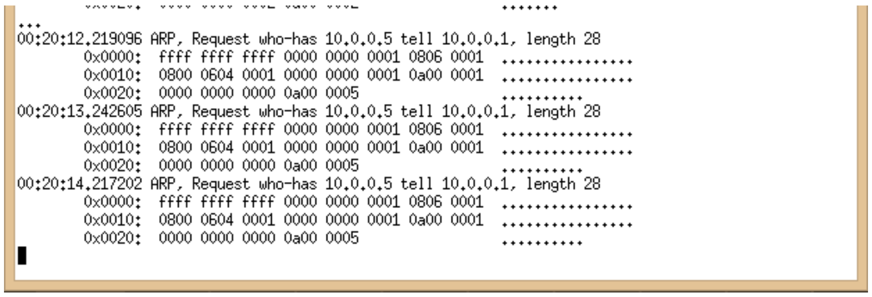


```
root@mininet-vm:~# ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=27.0 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 27.021/27.021/27.021/0.000 ms
root@mininet-vm:~# ping -c1 10.0.0.5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable

--- 10.0.0.5 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, ti
me 0ms

root@mininet-vm:~#
```

As seen above the controller floods the packet to host 2 and host 3. Thus here the controller acts like a hub.



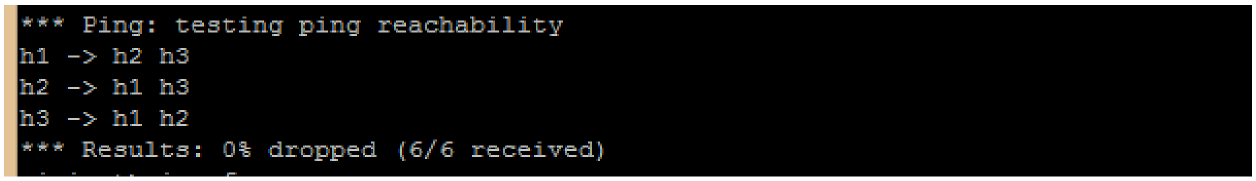
```
00:16:10.436301 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01, length
28
        0x0000:  0000 0000 0002 0000 0000 0001 0806 0001  ..........
......
        0x0010:  0800 0604 0002 0000 0000 0001 0a00 0001  ..........
......
        0x0020:  0000 0000 0002 0a00 0002                 ..........
00:20:12.219093 ARP, Request who-has 10.0.0.5 tell 10.0.0.1, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0000 0a00 0005                 ..........
00:20:13.242602 ARP, Request who-has 10.0.0.5 tell 10.0.0.1, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0000 0a00 0005                 ..........
00:20:14.217199 ARP, Request who-has 10.0.0.5 tell 10.0.0.1, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0000 0a00 0005                 ..........
```



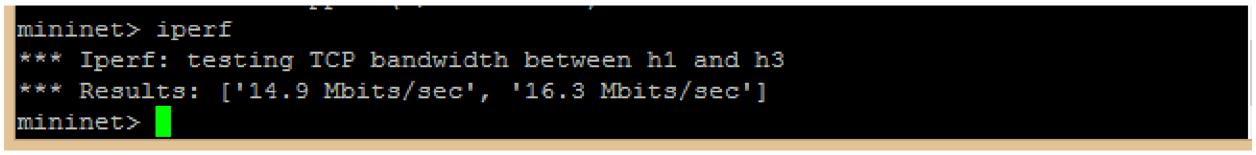
```
...
00:20:12.219096 ARP, Request who-has 10.0.0.5 tell 10.0.0.1, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0000 0a00 0005                 ..........
00:20:13.242605 ARP, Request who-has 10.0.0.5 tell 10.0.0.1, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0000 0a00 0005                 ..........
00:20:14.217202 ARP, Request who-has 10.0.0.5 tell 10.0.0.1, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0000 0a00 0005                 ..........
```

Benchmark Hub Controller w/iperf:

Verifying reachability:



```
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
```



```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['14.9 Mbits/sec', '16.3 Mbits/sec']
mininet>
```

As seen from above the bandwidth is 16.3 Mbps which is less compared to the open flow switch which we got earlier as 22.6 Gbps. This is because the reference controller is replaced with POX controller and all packets goes upto the controller.

Open Hub Code and Proceed:

Here we need to modify the **pox/misc/of_tutorial.py** file and verify the behavior of the combination of switch and controller as a controller-based Ethernet learning switch, and flow-accelerated learning switch.

Controller-based Ethernet learning switch: For this section I made the following changes to the Python Code:

def act_like_switch (self, packet, packet_in):

self.mac_to_port[packet.src] = packet_in.in_port;

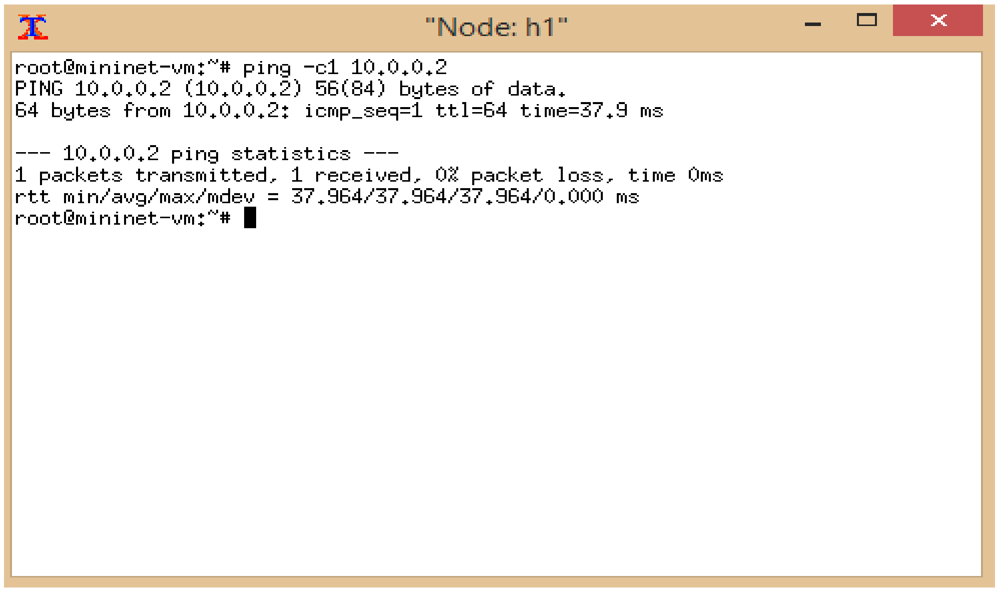
if self.mac_to_port.get(packet.dst) != None:

self.resend_packet (packet_in,self.mac_to_port[packet.dst])

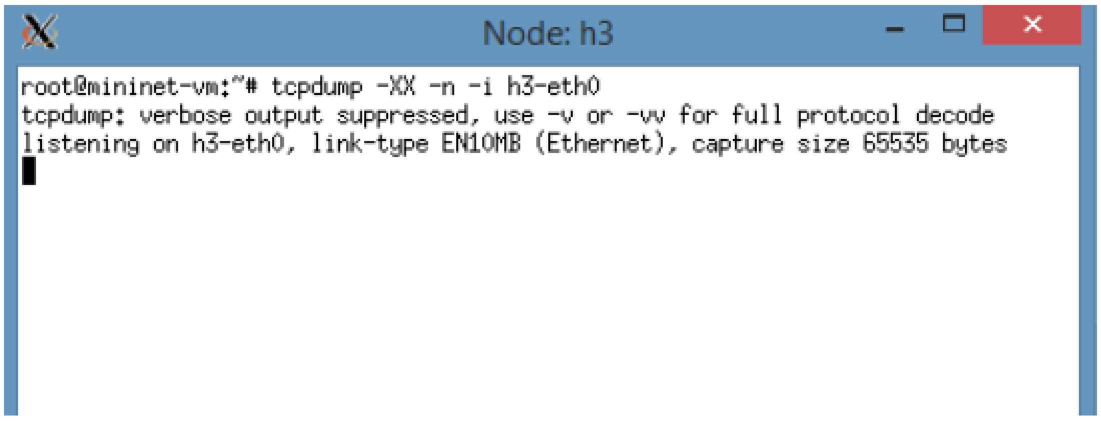
else:

self.resend_packet(packet_in, of.OFPP_ALL)

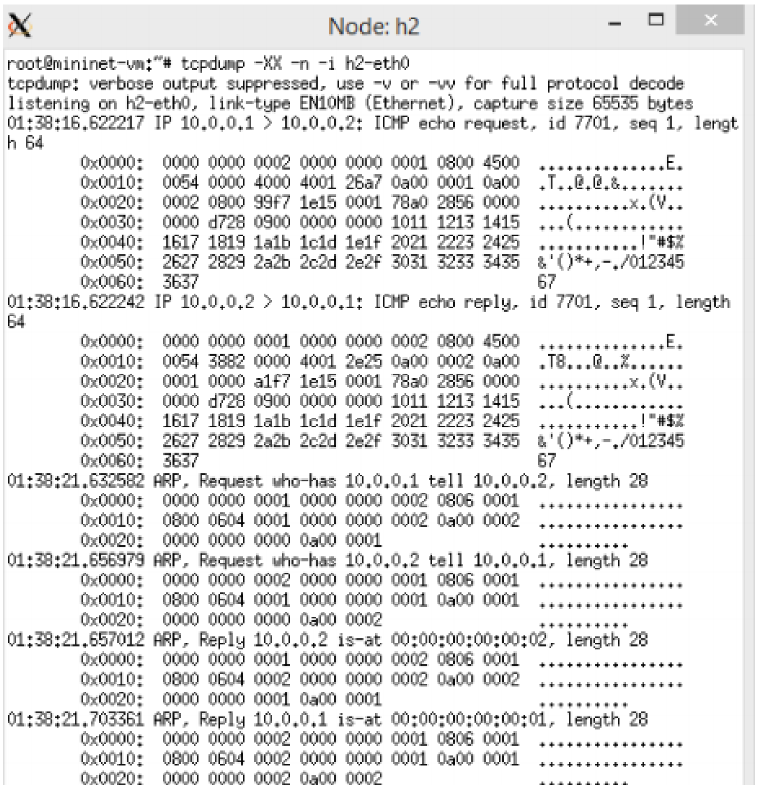
After making these changes ping h2 from h1 using: ping –c1 10.0.0.2 and traceroute in h2 and h3. In this section the ICMP ping packets won't reach h3 because the switch will not flood the packets to all the hosts.



```
root@mininet-vm:~# ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=37.9 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 37.964/37.964/37.964/0.000 ms
root@mininet-vm:~#
```



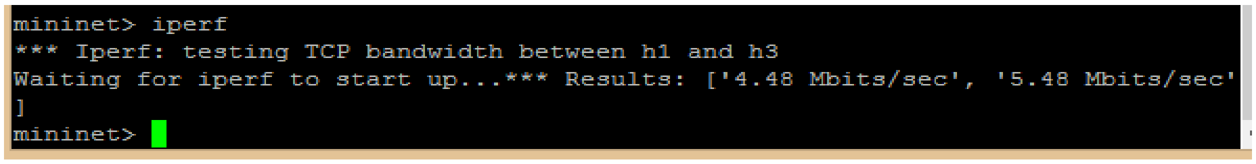
```
root@mininet-vm:~# tcpdump -XX -n -i h3-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
```



Node: h2

```
root@mininet-vm:~# tcpdump -XX -n -i h2-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
01:38:16.622217 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 7701, seq 1, lengt
h 64
        0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  ..............E.
        0x0010:  0054 0000 4000 4001 26a7 0a00 0001 0a00  .T..@.@.&.......
        0x0020:  0002 0800 99f7 1e15 0001 78a0 2856 0000  ..........x.(V..
        0x0030:  0000 d728 0900 0000 0000 1011 1213 1415  ...(............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
01:38:16.622242 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 7701, seq 1, length
64
        0x0000:  0000 0000 0001 0000 0000 0002 0800 4500  ..............E.
        0x0010:  0054 3882 0000 4001 2e25 0a00 0002 0a00  .T8...@..%......
        0x0020:  0001 0000 a1f7 1e15 0001 78a0 2856 0000  ..........x.(V..
        0x0030:  0000 d728 0900 0000 0000 1011 1213 1415  ...(............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
01:38:21.632582 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
        0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0002 0a00 0002  ................
        0x0020:  0000 0000 0000 0a00 0001                 ..........
01:38:21.656979 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
        0x0000:  0000 0000 0002 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0000 0a00 0002                 ..........
01:38:21.657012 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
        0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0002 0a00 0002  ................
        0x0020:  0000 0000 0001 0a00 0001                 ..........
01:38:21.703361 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01, length 28
        0x0000:  0000 0000 0002 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0002 0a00 0002                 ..........
```

Doing iperf we get:



```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
Waiting for iperf to start up...*** Results: ['4.48 Mbits/sec', '5.48 Mbits/sec'
]
mininet> 
```

Throughput reduces because every time a packet comes in it asks the controller. This is because there is no flow entry which is created.

Flow-accelerated learning switch:

In this part I made the following changes to the Python file:

def act_like_switch (self, packet, packet_in):

self.mac_to_port [packet.src] = packet_in.in_port

if packet.dst in self.mac_to_port:

msg = of.ofp_flow_mod()

msg.match = of.ofp_match.from_packet(packet)

msg.idle_timeout = 10

msg.hard_timeout = 20

msg.buffer_id = packet_in.buffer_id

action = of.ofp_action_output(port = self.mac_to_port[packet.dst])

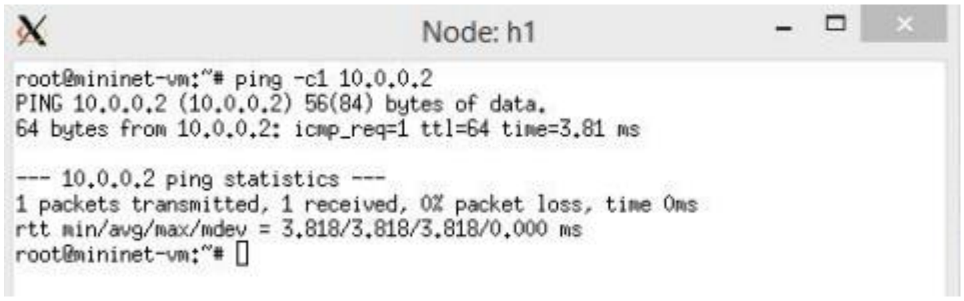
msg.actions.append(action)

self.connection.send(msg)

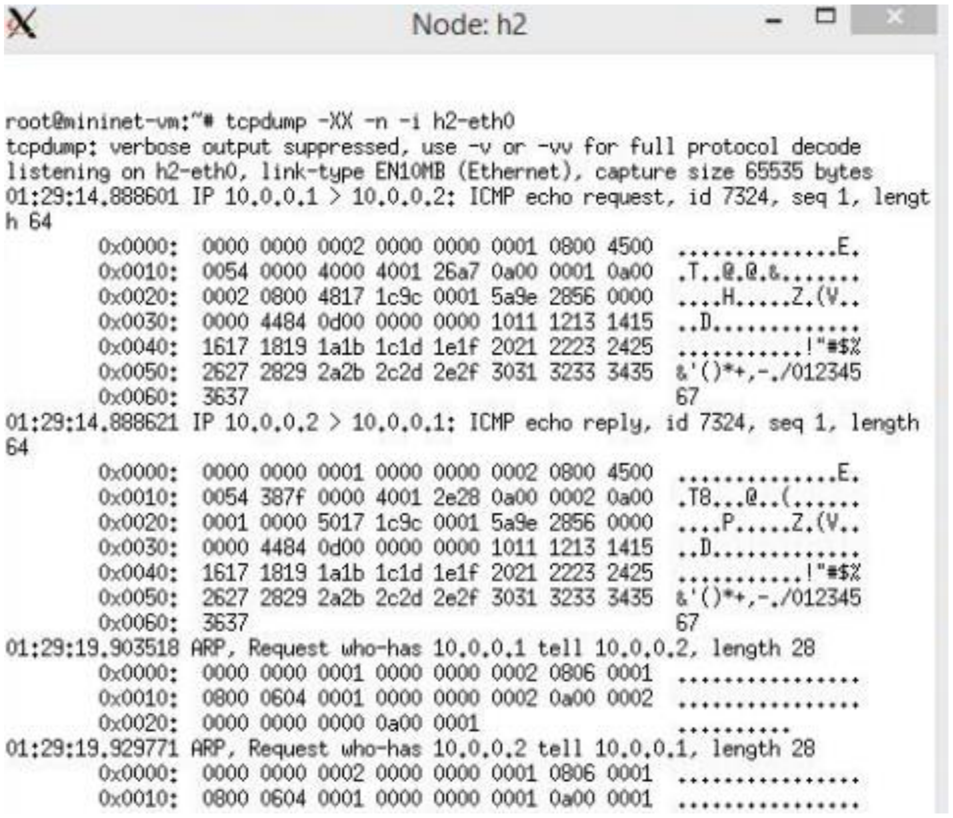
else:

self.resend_packet(packet_in, of.OFPP_ALL)

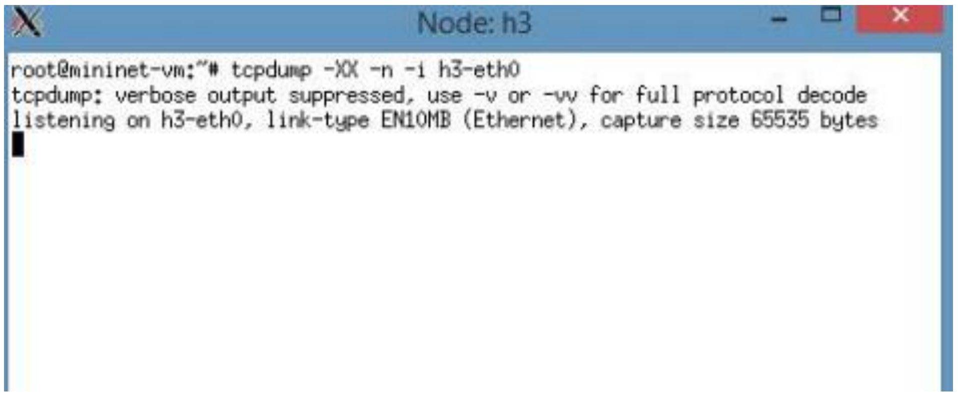
After making these changes ping h1 to h2. Traceroute in h2 shows packets but in h3 there are no ICMP packets.



```
root@mininet-vm:~# ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=3.81 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 3.818/3.818/3.818/0.000 ms
root@mininet-vm:~# 
```

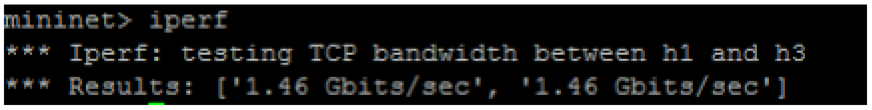


```
root@mininet-vm:~# tcpdump -XX -n -i h2-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
01:29:14.888601 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 7324, seq 1, lengt
h 64
        0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  ..............E.
        0x0010:  0054 0000 4000 4001 26a7 0a00 0001 0a00  .T..@.@.&.......
        0x0020:  0002 0800 4817 1c9c 0001 5a9e 2856 0000  ....H.....Z.(V..
        0x0030:  0000 4484 0d00 0000 0000 1011 1213 1415  ..D.............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
01:29:14.888621 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 7324, seq 1, length
64
        0x0000:  0000 0000 0001 0000 0000 0002 0800 4500  ..............E.
        0x0010:  0054 387f 0000 4001 2e28 0a00 0002 0a00  .T8...@..(......
        0x0020:  0001 0000 5017 1c9c 0001 5a9e 2856 0000  ....P.....Z.(V..
        0x0030:  0000 4484 0d00 0000 0000 1011 1213 1415  ..D.............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
01:29:19.903518 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
        0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0002 0a00 0002  ................
        0x0020:  0000 0000 0000 0a00 0001                 ..........
01:29:19.929771 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
        0x0000:  0000 0000 0002 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ................
```



```
root@mininet-vm:~# tcpdump -XX -n -i h3-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
```

Doing iperf we get:

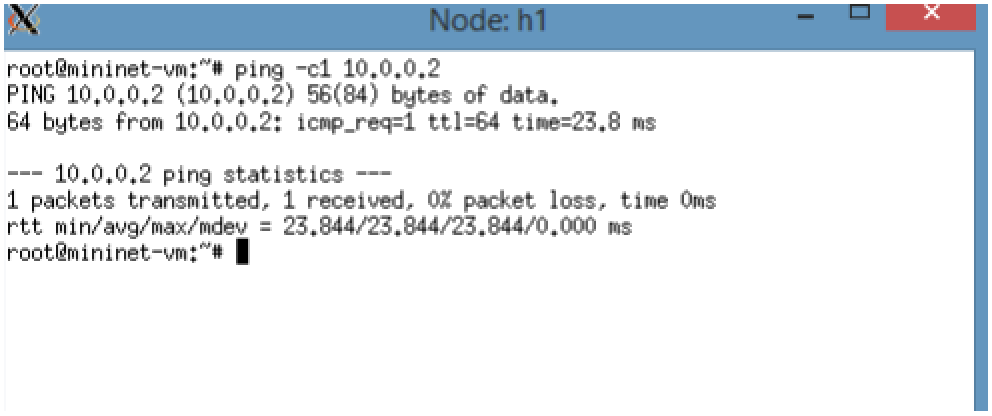


```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['1.46 Gbits/sec', '1.46 Gbits/sec']
```

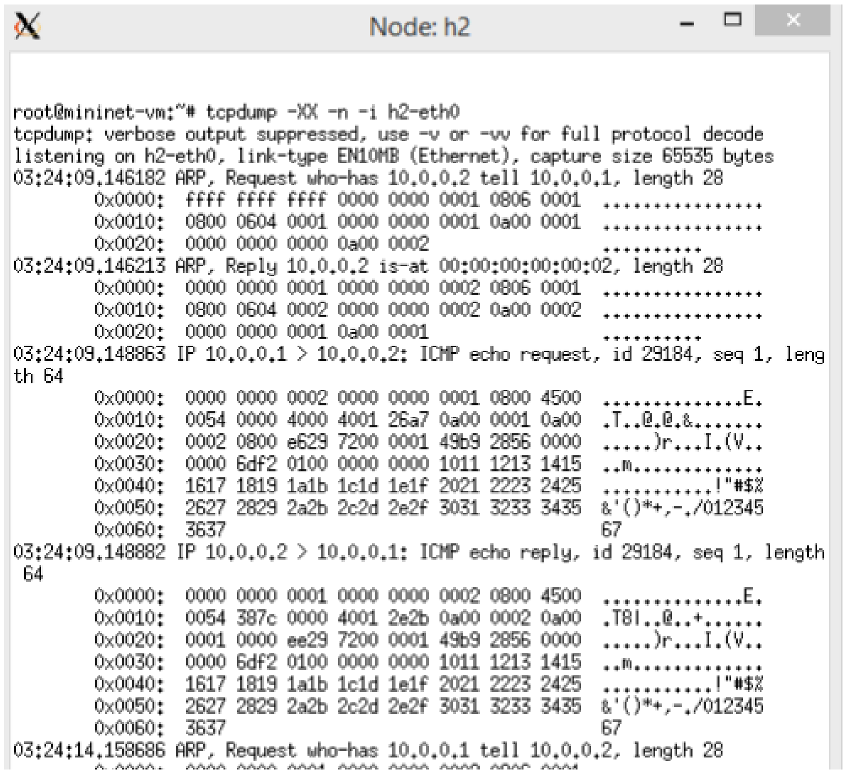
As seen above the throughput increases because the packets doesn't have to go to the controller every time and checks the flow table.

<u>Testing your controller:</u>

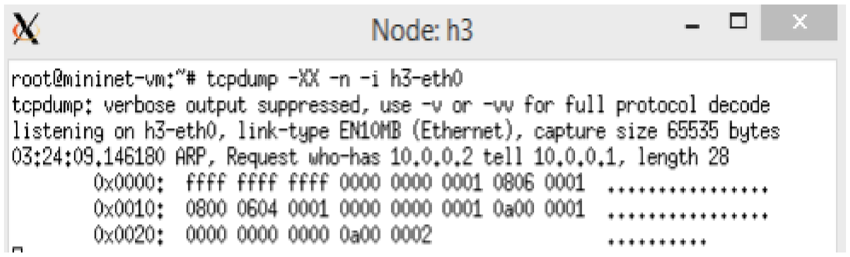
Here we need to first verify that when all packets arrive at the controller, only broadcast packets (like ARPs) and packets with unknown destination locations (like the first packet sent for a flow) go out all non-input ports. This is verified from the above screenshots as we can see that h3 doesn't receive any packets. However when I do ping first there is an ARP packet in h3. I do this with tcpdump on an xterm for each host.



```
root@mininet-vm:~# ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=23.8 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 23.844/23.844/23.844/0.000 ms
root@mininet-vm:~#
```

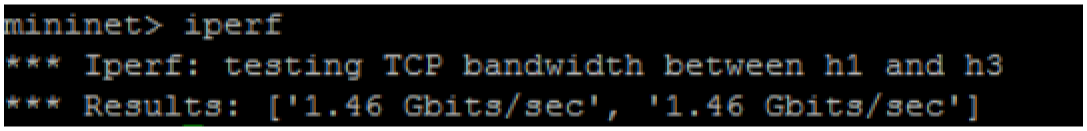


```
root@mininet-vm:~# tcpdump -XX -n -i h2-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
03:24:09.146182 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0000 0a00 0002                 ..........
03:24:09.146213 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
        0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0002 0a00 0002  ................
        0x0020:  0000 0000 0001 0a00 0001                 ..........
03:24:09.148863 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 29184, seq 1, leng
th 64
        0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  ..............E.
        0x0010:  0054 0000 4000 4001 26a7 0a00 0001 0a00  .T..@.@.&.......
        0x0020:  0002 0800 e629 7200 0001 49b9 2856 0000  .....)r...I.(V..
        0x0030:  0000 6df2 0100 0000 0000 1011 1213 1415  ..m.............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
03:24:09.148882 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 29184, seq 1, length
 64
        0x0000:  0000 0000 0001 0000 0000 0002 0800 4500  ..............E.
        0x0010:  0054 387c 0000 4001 2e2b 0a00 0002 0a00  .T8|..@..+......
        0x0020:  0001 0000 ee29 7200 0001 49b9 2856 0000  .....)r...I.(V..
        0x0030:  0000 6df2 0100 0000 0000 1011 1213 1415  ..m.............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
03:24:14.158686 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
```



```
root@mininet-vm:~# tcpdump -XX -n -i h3-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
03:24:09.146180 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0000 0a00 0002                 ..........
```

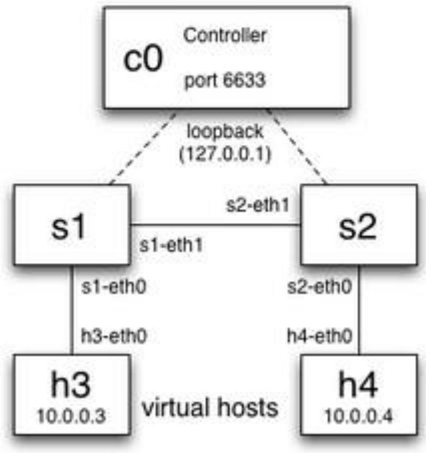
Doing iperf we get the same throughput as we got when using the reference learning switch controller earlier:

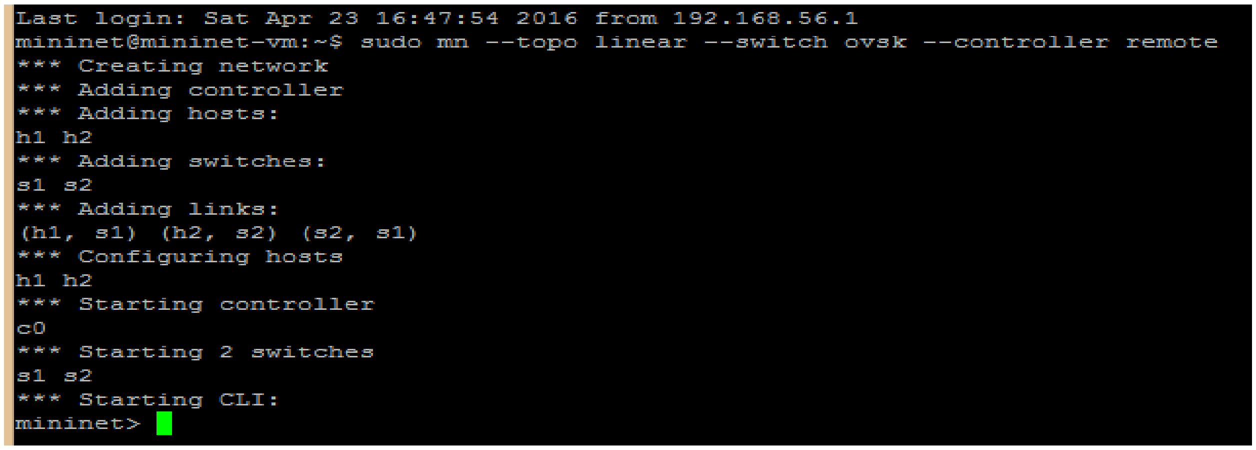


```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['1.46 Gbits/sec', '1.46 Gbits/sec']
```

<u>Support multiple switches:</u>

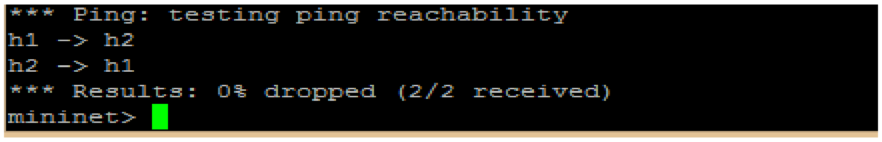
Now I create a 2-swicth topology where each switch has a single connected host.





```
Last login: Sat Apr 23 16:47:54 2016 from 192.168.56.1
mininet@mininet-vm:~$ sudo mn --topo linear --switch ovsk --controller remote
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s2) (s2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 2 switches
s1 s2
*** Starting CLI:
mininet>
```

After the mods, to verify that the controller is working we type pingall in the MIninet console and get:

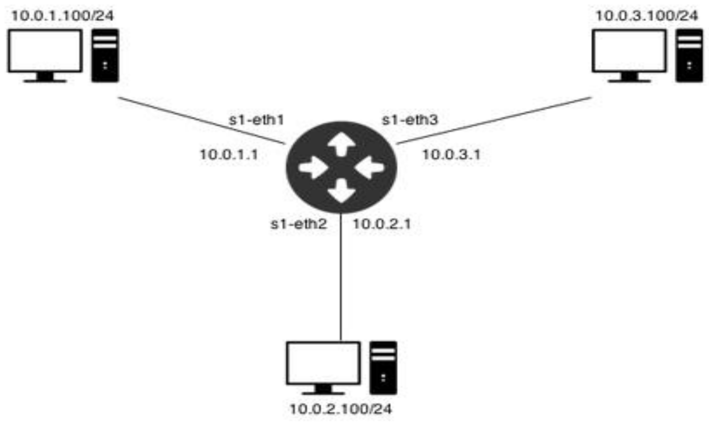


```
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

<u>Router exercise: (PART A)</u>

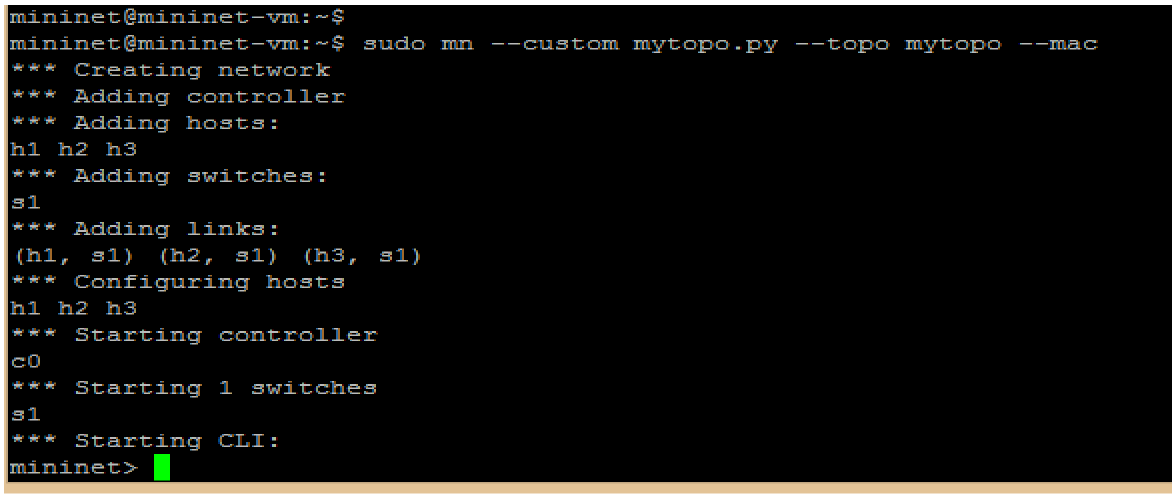
Here we will make a static layer-3 forwarder/ switch.

<u>Create topology:</u>



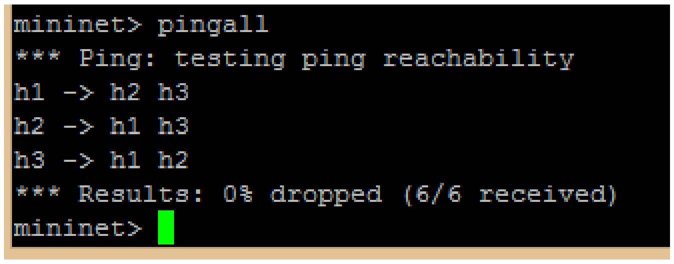
Set up hosts:

We make a topology as shown above by making the required changes to mytopo.py. As seen the controller is UP and running.



```
mininet@mininet-vm:~$
mininet@mininet-vm:~$ sudo mn --custom mytopo.py --topo mytopo --mac
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

Doing pingall we get:

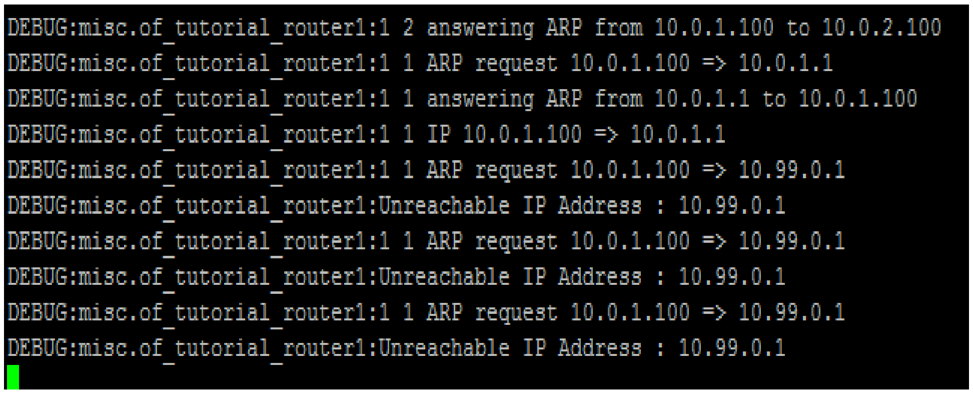


```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```

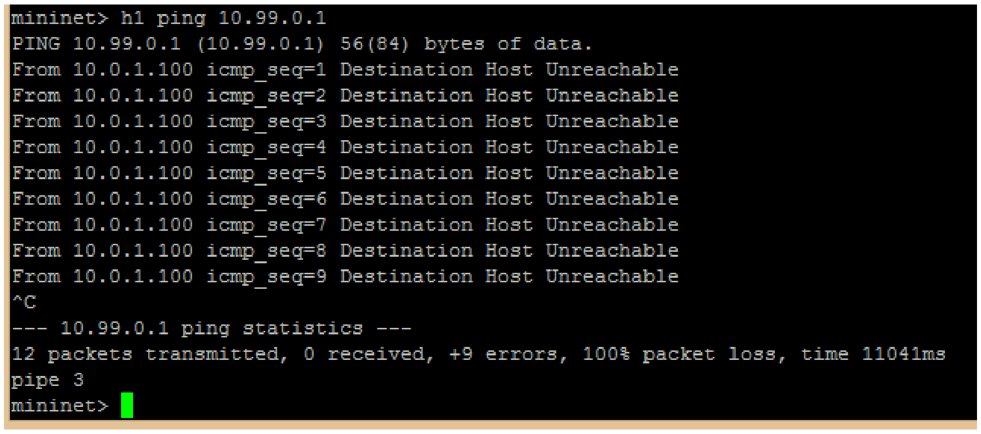
After making the necessary changes to of_tutorial_switch.py we test our topology with regards to the various guidelines given in the tutorial. I made two files, one of_tutorial_switch and the other of_tutorial_routerA.

Testing your router:

- Attempts to send from 10.0.1.2 to an unknown address range like 10.99.0.1 should yield an ICMP destination unreachable message.



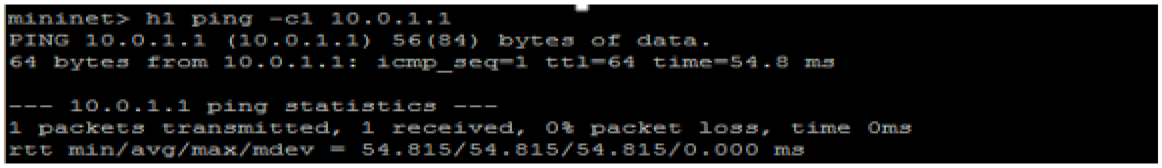
```
DEBUG:misc.of_tutorial_router1:1 2 answering ARP from 10.0.1.100 to 10.0.2.100
DEBUG:misc.of_tutorial_router1:1 1 ARP request 10.0.1.100 => 10.0.1.1
DEBUG:misc.of_tutorial_router1:1 1 answering ARP from 10.0.1.1 to 10.0.1.100
DEBUG:misc.of_tutorial_router1:1 1 IP 10.0.1.100 => 10.0.1.1
DEBUG:misc.of_tutorial_router1:1 1 ARP request 10.0.1.100 => 10.99.0.1
DEBUG:misc.of_tutorial_router1:Unreachable IP Address : 10.99.0.1
DEBUG:misc.of_tutorial_router1:1 1 ARP request 10.0.1.100 => 10.99.0.1
DEBUG:misc.of_tutorial_router1:Unreachable IP Address : 10.99.0.1
DEBUG:misc.of_tutorial_router1:1 1 ARP request 10.0.1.100 => 10.99.0.1
DEBUG:misc.of_tutorial_router1:Unreachable IP Address : 10.99.0.1
```



```
mininet> h1 ping 10.99.0.1
PING 10.99.0.1 (10.99.0.1) 56(84) bytes of data.
From 10.0.1.100 icmp_seq=1 Destination Host Unreachable
From 10.0.1.100 icmp_seq=2 Destination Host Unreachable
From 10.0.1.100 icmp_seq=3 Destination Host Unreachable
From 10.0.1.100 icmp_seq=4 Destination Host Unreachable
From 10.0.1.100 icmp_seq=5 Destination Host Unreachable
From 10.0.1.100 icmp_seq=6 Destination Host Unreachable
From 10.0.1.100 icmp_seq=7 Destination Host Unreachable
From 10.0.1.100 icmp_seq=8 Destination Host Unreachable
From 10.0.1.100 icmp_seq=9 Destination Host Unreachable
^C
--- 10.99.0.1 ping statistics ---
12 packets transmitted, 0 received, +9 errors, 100% packet loss, time 11041ms
pipe 3
mininet>
```

As seen a ping to 10.99.0.1 which is an unknown host gives Destination Host Unreachable message, this is because the controller doesn't have the above IP address and thus sends an ICMP message to the switch and that sends it to the host which pings (in this example it is h1)

- The router should be pingable, and should generate an ICMP echo reply in response to an ICMP echo request.

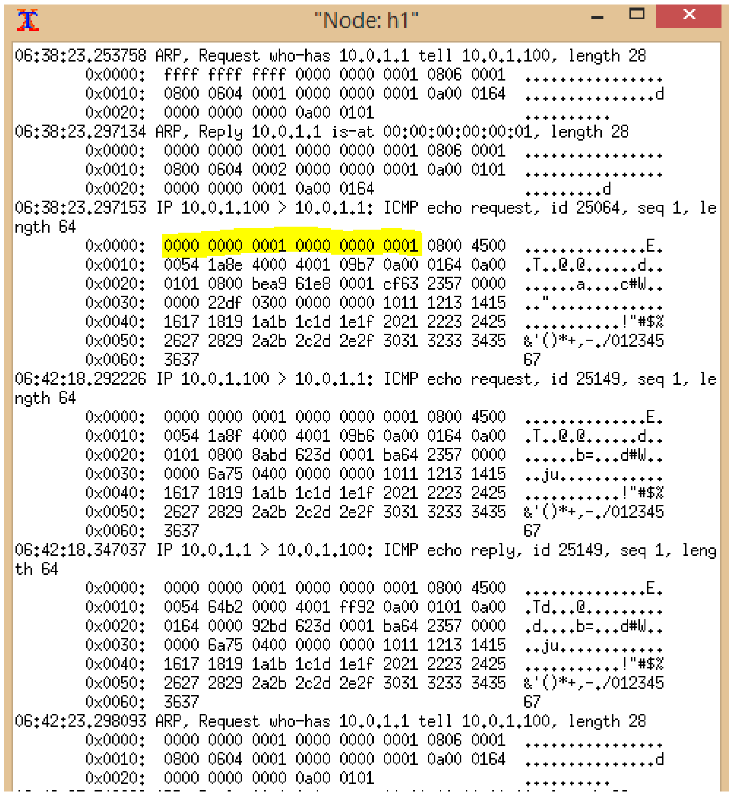


```
mininet> h1 ping -c1 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=54.8 ms

--- 10.0.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 54.815/54.815/54.815/0.000 ms
```

As seen from the above screenshot, the ping from host h1 to its own router 10.0.1.1 sends a ICMP ECHO reply message and the ping is successful.

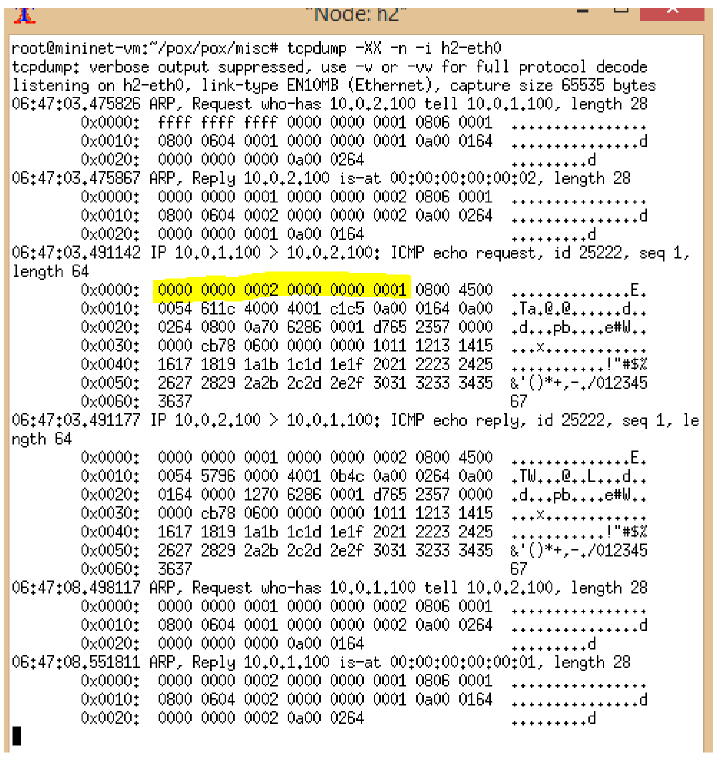
- Packets sent to hosts on a known address range should have their MAC destination field changed to that of the next-hop router.

In this case we do tcpdump on host h1 and h3 to find that their MAC destination field is that of the next hop router. As seen the MAC destination in this case is 00-00-00-00-00-01 which is that of the default router.


"Node: h1"

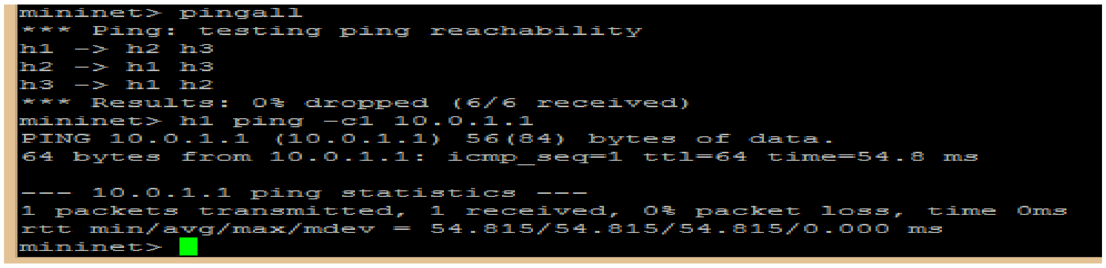
```
06:38:23.253758 ARP, Request who-has 10.0.1.1 tell 10.0.1.100, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0164  ...............d
        0x0020:  0000 0000 0000 0a00 0101                 ..........
06:38:23.297134 ARP, Reply 10.0.1.1 is-at 00:00:00:00:00:01, length 28
        0x0000:  0000 0000 0001 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0001 0a00 0101  ................
        0x0020:  0000 0000 0001 0a00 0164                 .........d
06:38:23.297153 IP 10.0.1.100 > 10.0.1.1: ICMP echo request, id 25064, seq 1, le
ngth 64
        0x0000:  0000 0000 0001 0000 0000 0001 0800 4500  ..............E.
        0x0010:  0054 1a8e 4000 4001 09b7 0a00 0164 0a00  .T..@.@......d..
        0x0020:  0101 0800 bea9 61e8 0001 cf63 2357 0000  ......a....c#W..
        0x0030:  0000 22df 0300 0000 0000 1011 1213 1415  ..".............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
06:42:18.292226 IP 10.0.1.100 > 10.0.1.1: ICMP echo request, id 25149, seq 1, le
ngth 64
        0x0000:  0000 0000 0001 0000 0000 0001 0800 4500  ..............E.
        0x0010:  0054 1a8f 4000 4001 09b6 0a00 0164 0a00  .T..@.@......d..
        0x0020:  0101 0800 8abd 623d 0001 ba64 2357 0000  ......b=...d#W..
        0x0030:  0000 6a75 0400 0000 0000 1011 1213 1415  ..ju............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
06:42:18.347037 IP 10.0.1.1 > 10.0.1.100: ICMP echo reply, id 25149, seq 1, leng
th 64
        0x0000:  0000 0000 0001 0000 0000 0001 0800 4500  ..............E.
        0x0010:  0054 64b2 0000 4001 ff92 0a00 0101 0a00  .Td...@.........
        0x0020:  0164 0000 92bd 623d 0001 ba64 2357 0000  .d....b=...d#W..
        0x0030:  0000 6a75 0400 0000 0000 1011 1213 1415  ..ju............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
06:42:23.298093 ARP, Request who-has 10.0.1.1 tell 10.0.1.100, length 28
        0x0000:  0000 0000 0001 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0164  ...............d
        0x0020:  0000 0000 0000 0a00 0101                 ..........
```

For h1 pinging h2 the next hop router is 00-00-00-00-00-02 which is as shown below:


"Node: h2"

```
root@mininet-vm:~/pox/pox/misc# tcpdump -XX -n -i h2-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
06:47:03.475826 ARP, Request who-has 10.0.2.100 tell 10.0.1.100, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0164  ...............d
        0x0020:  0000 0000 0000 0a00 0264                 .........d
06:47:03.475867 ARP, Reply 10.0.2.100 is-at 00:00:00:00:00:02, length 28
        0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0002 0a00 0264  ...............d
        0x0020:  0000 0000 0001 0a00 0164                 .........d
06:47:03.491142 IP 10.0.1.100 > 10.0.2.100: ICMP echo request, id 25222, seq 1,
length 64
        0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  ..............E.
        0x0010:  0054 611c 4000 4001 c1c5 0a00 0164 0a00  .Ta.@.@......d..
        0x0020:  0264 0800 0a70 6286 0001 d765 2357 0000  .d...pb....e#W..
        0x0030:  0000 cb78 0600 0000 0000 1011 1213 1415  ...x............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
06:47:03.491177 IP 10.0.2.100 > 10.0.1.100: ICMP echo reply, id 25222, seq 1, le
ngth 64
        0x0000:  0000 0000 0001 0000 0000 0002 0800 4500  ..............E.
        0x0010:  0054 5796 0000 4001 0b4c 0a00 0264 0a00  .TW...@..L...d..
        0x0020:  0164 0000 1270 6286 0001 d765 2357 0000  .d....pb....e#W..
        0x0030:  0000 cb78 0600 0000 0000 1011 1213 1415  ...x............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
06:47:08.498117 ARP, Request who-has 10.0.1.100 tell 10.0.2.100, length 28
        0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0002 0a00 0264  ...............d
        0x0020:  0000 0000 0000 0a00 0164                 .........d
06:47:08.551811 ARP, Reply 10.0.1.100 is-at 00:00:00:00:00:01, length 28
        0x0000:  0000 0000 0002 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0001 0a00 0164  ...............d
        0x0020:  0000 0000 0002 0a00 0264                 .........d
```

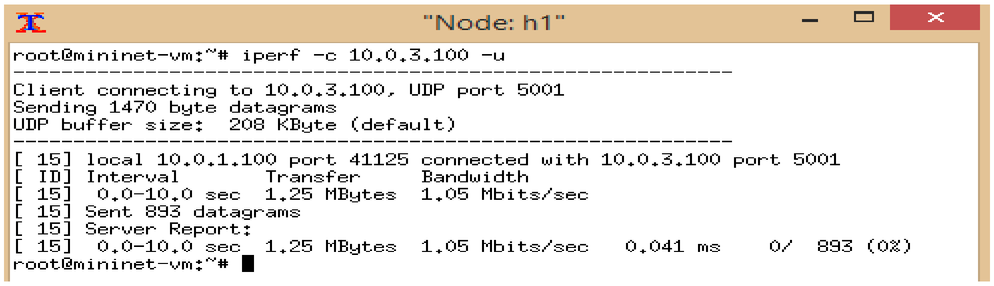
As shown below pingall also works properly:



```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> h1 ping -c1 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=54.8 ms

--- 10.0.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 54.815/54.815/54.815/0.000 ms
mininet>
```

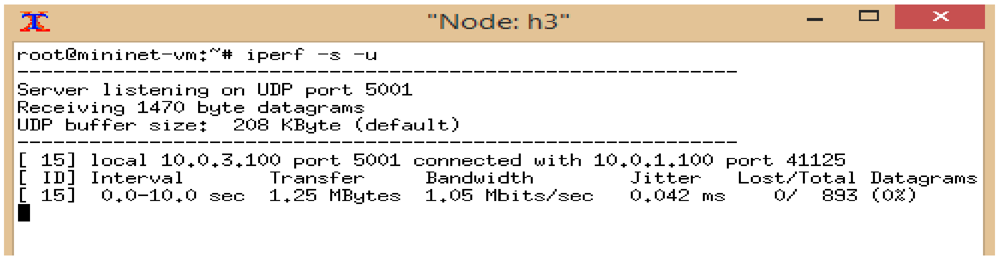
Now running iperf to test TCP and UDP traffic, I open xterm in host 1 and host 3 as shown below:

Here we are running iperf to test UDP connection. As seen below the bandwidth is 1.05 Mbits/sec



"Node: h1"

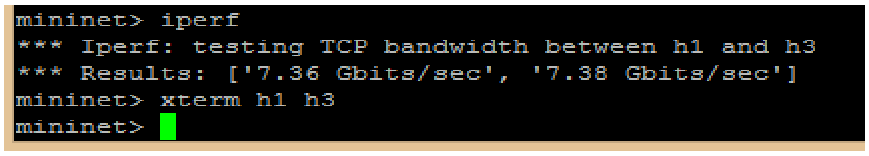
```
root@mininet-vm:~# iperf -c 10.0.3.100 -u
------------------------------------------------------------
Client connecting to 10.0.3.100, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 15] local 10.0.1.100 port 41125 connected with 10.0.3.100 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 15]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[ 15] Sent 893 datagrams
[ 15] Server Report:
[ 15]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.041 ms    0/  893 (0%)
root@mininet-vm:~#
```



"Node: h3"

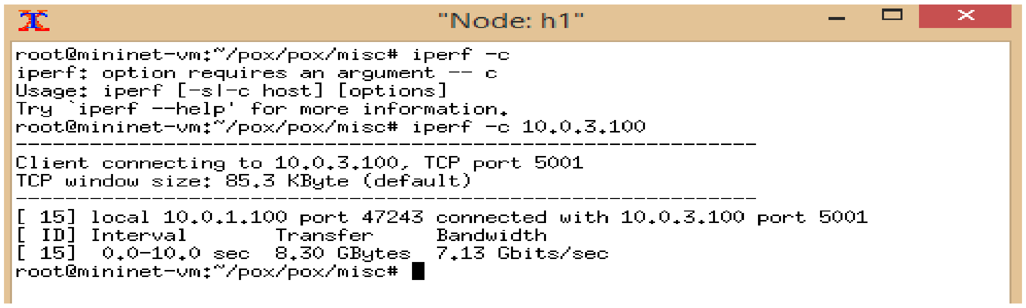
```
root@mininet-vm:~# iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 15] local 10.0.3.100 port 5001 connected with 10.0.1.100 port 41125
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[ 15]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec  0.042 ms    0/  893 (0%)
```

Testing TCP traffic:

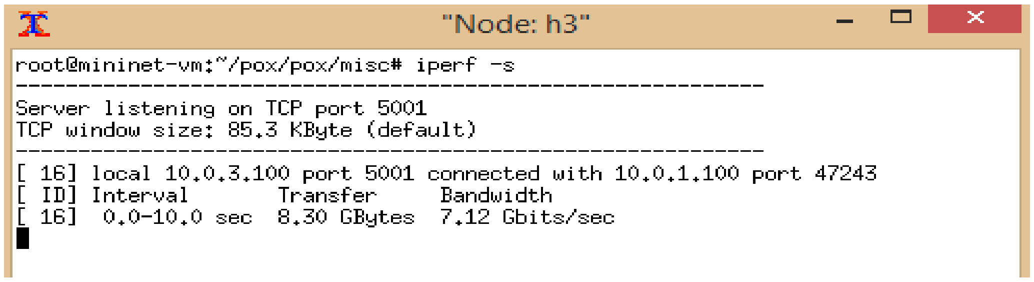


```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['7.36 Gbits/sec', '7.38 Gbits/sec']
mininet> xterm h1 h3
mininet>
```

From xming we get (for TCP):



```
root@mininet-vm:~/pox/pox/misc# iperf -c
iperf: option requires an argument -- c
Usage: iperf [-s|-c host] [options]
Try `iperf --help' for more information.
root@mininet-vm:~/pox/pox/misc# iperf -c 10.0.3.100
------------------------------------------------------------
Client connecting to 10.0.3.100, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 15] local 10.0.1.100 port 47243 connected with 10.0.3.100 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 15]  0.0-10.0 sec  8.30 GBytes  7.13 Gbits/sec
root@mininet-vm:~/pox/pox/misc#
```



```
root@mininet-vm:~/pox/pox/misc# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 16] local 10.0.3.100 port 5001 connected with 10.0.1.100 port 47243
[ ID] Interval       Transfer     Bandwidth
[ 16]  0.0-10.0 sec  8.30 GBytes  7.12 Gbits/sec
```

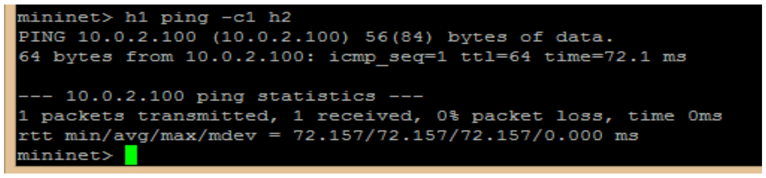
As seen from above the TCP bandwidth is 7.12 Gbit/sec which is way higher than the UDP bandwidth which is 1.05 Mbits/sec. This is because most of the data transfer takes place through TCP connection.

Flow Mods:

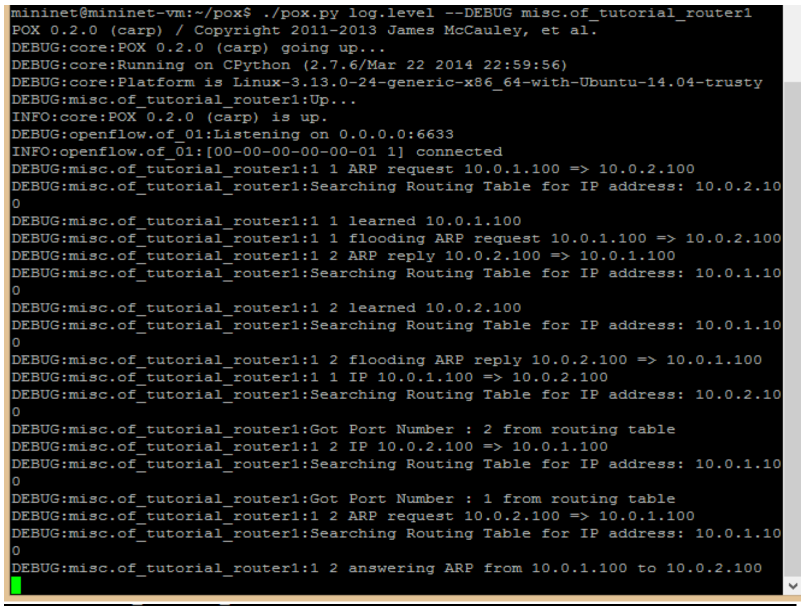
Here after installing flow mods, we iperf again to note the bandwidth.

{Add iperf for flow mods}

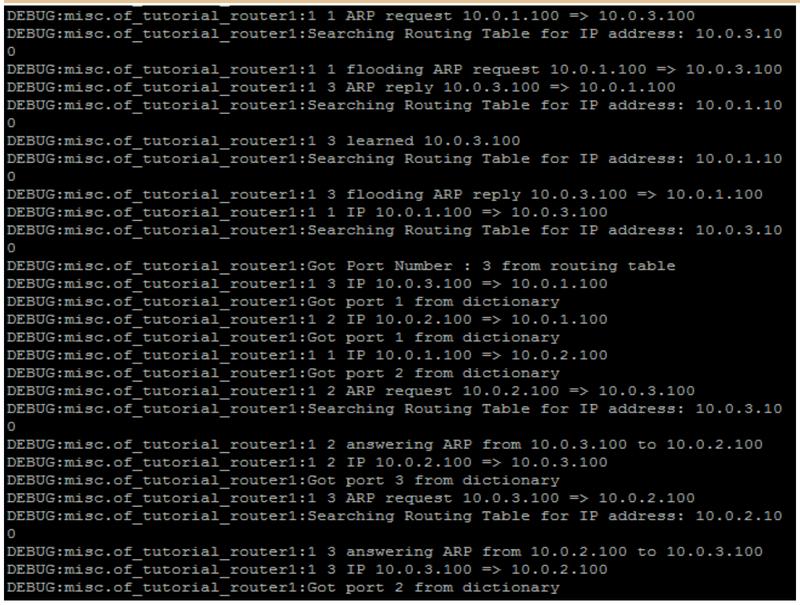
The below screenshots are some of the controller messages when we do h1 ping –c1 h2



```
mininet> h1 ping -c1 h2
PING 10.0.2.100 (10.0.2.100) 56(84) bytes of data.
64 bytes from 10.0.2.100: icmp_seq=1 ttl=64 time=72.1 ms

--- 10.0.2.100 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 72.157/72.157/72.157/0.000 ms
mininet>
```

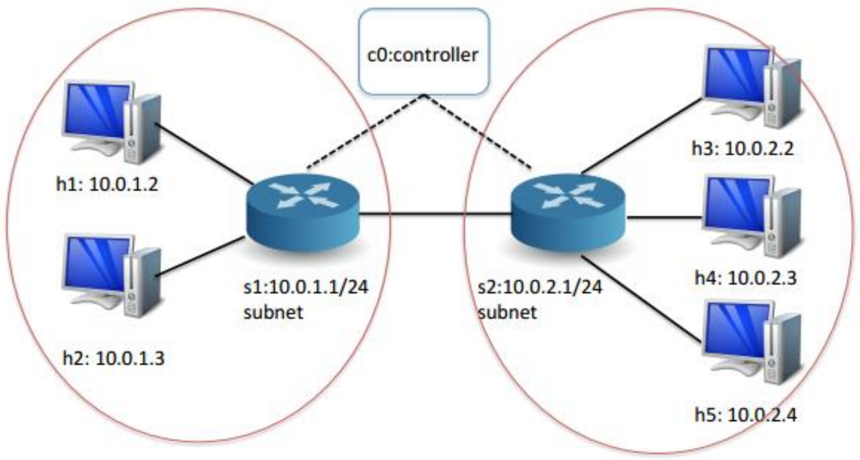


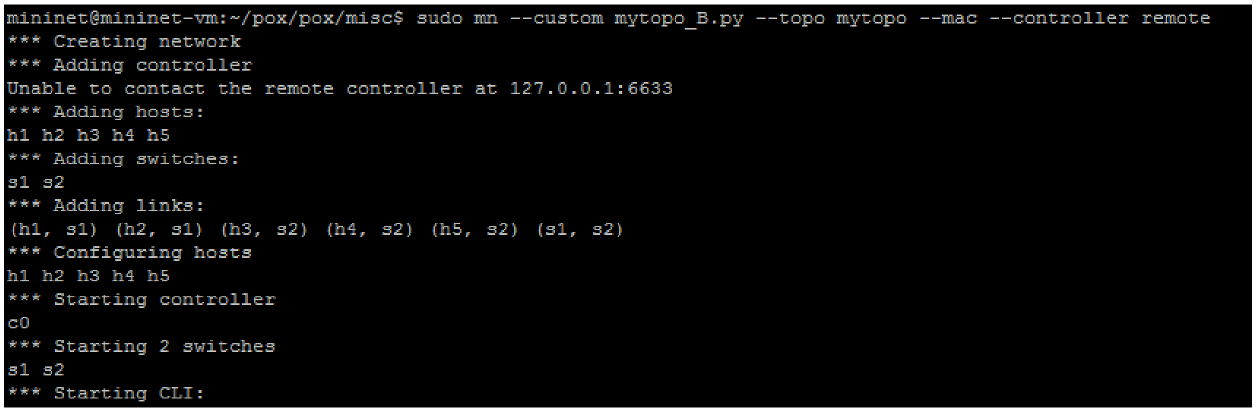
```
mininet@mininet-vm:~/pox$ ./pox.py log.level --DEBUG misc.of_tutorial_router1
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Mar 22 2014 22:59:56)
DEBUG:core:Platform is Linux-3.13.0-24-generic-x86_64-with-Ubuntu-14.04-trusty
DEBUG:misc.of_tutorial_router1:Up...
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
DEBUG:misc.of_tutorial_router1:1 1 ARP request 10.0.1.100 => 10.0.2.100
DEBUG:misc.of_tutorial_router1:Searching Routing Table for IP address: 10.0.2.10
0
DEBUG:misc.of_tutorial_router1:1 1 learned 10.0.1.100
DEBUG:misc.of_tutorial_router1:1 1 flooding ARP request 10.0.1.100 => 10.0.2.100
DEBUG:misc.of_tutorial_router1:1 2 ARP reply 10.0.2.100 => 10.0.1.100
DEBUG:misc.of_tutorial_router1:Searching Routing Table for IP address: 10.0.1.10
0
DEBUG:misc.of_tutorial_router1:1 2 learned 10.0.2.100
DEBUG:misc.of_tutorial_router1:Searching Routing Table for IP address: 10.0.1.10
0
DEBUG:misc.of_tutorial_router1:1 2 flooding ARP reply 10.0.2.100 => 10.0.1.100
DEBUG:misc.of_tutorial_router1:1 1 IP 10.0.1.100 => 10.0.2.100
DEBUG:misc.of_tutorial_router1:Searching Routing Table for IP address: 10.0.2.10
0
DEBUG:misc.of_tutorial_router1:Got Port Number : 2 from routing table
DEBUG:misc.of_tutorial_router1:1 2 IP 10.0.2.100 => 10.0.1.100
DEBUG:misc.of_tutorial_router1:Searching Routing Table for IP address: 10.0.1.10
0
DEBUG:misc.of_tutorial_router1:Got Port Number : 1 from routing table
DEBUG:misc.of_tutorial_router1:1 2 ARP request 10.0.2.100 => 10.0.1.100
DEBUG:misc.of_tutorial_router1:Searching Routing Table for IP address: 10.0.1.10
0
DEBUG:misc.of_tutorial_router1:1 2 answering ARP from 10.0.1.100 to 10.0.2.100
```



```
DEBUG:misc.of_tutorial_router1:1 1 ARP request 10.0.1.100 => 10.0.3.100
DEBUG:misc.of_tutorial_router1:Searching Routing Table for IP address: 10.0.3.10
0
DEBUG:misc.of_tutorial_router1:1 1 flooding ARP request 10.0.1.100 => 10.0.3.100
DEBUG:misc.of_tutorial_router1:1 3 ARP reply 10.0.3.100 => 10.0.1.100
DEBUG:misc.of_tutorial_router1:Searching Routing Table for IP address: 10.0.1.10
0
DEBUG:misc.of_tutorial_router1:1 3 learned 10.0.3.100
DEBUG:misc.of_tutorial_router1:Searching Routing Table for IP address: 10.0.1.10
0
DEBUG:misc.of_tutorial_router1:1 3 flooding ARP reply 10.0.3.100 => 10.0.1.100
DEBUG:misc.of_tutorial_router1:1 1 IP 10.0.1.100 => 10.0.3.100
DEBUG:misc.of_tutorial_router1:Searching Routing Table for IP address: 10.0.3.10
0
DEBUG:misc.of_tutorial_router1:Got Port Number : 3 from routing table
DEBUG:misc.of_tutorial_router1:1 3 IP 10.0.3.100 => 10.0.1.100
DEBUG:misc.of_tutorial_router1:Got port 1 from dictionary
DEBUG:misc.of_tutorial_router1:1 2 IP 10.0.2.100 => 10.0.1.100
DEBUG:misc.of_tutorial_router1:Got port 1 from dictionary
DEBUG:misc.of_tutorial_router1:1 1 IP 10.0.1.100 => 10.0.2.100
DEBUG:misc.of_tutorial_router1:Got port 2 from dictionary
DEBUG:misc.of_tutorial_router1:1 2 ARP request 10.0.2.100 => 10.0.3.100
DEBUG:misc.of_tutorial_router1:Searching Routing Table for IP address: 10.0.3.10
0
DEBUG:misc.of_tutorial_router1:1 2 answering ARP from 10.0.3.100 to 10.0.2.100
DEBUG:misc.of_tutorial_router1:1 2 IP 10.0.2.100 => 10.0.3.100
DEBUG:misc.of_tutorial_router1:Got port 3 from dictionary
DEBUG:misc.of_tutorial_router1:1 3 ARP request 10.0.3.100 => 10.0.2.100
DEBUG:misc.of_tutorial_router1:Searching Routing Table for IP address: 10.0.2.10
0
DEBUG:misc.of_tutorial_router1:1 3 answering ARP from 10.0.2.100 to 10.0.3.100
DEBUG:misc.of_tutorial_router1:1 3 IP 10.0.3.100 => 10.0.2.100
DEBUG:misc.of_tutorial_router1:Got port 2 from dictionary
```

Router Exercise (PART B) :

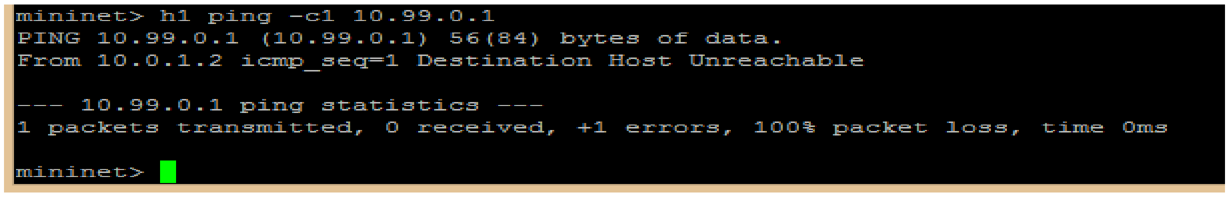




```
mininet@mininet-vm:~/pox/pox/misc$ sudo mn --custom mytopo_B.py --topo mytopo --mac --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4 h5
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (h5, s2) (s1, s2)
*** Configuring hosts
h1 h2 h3 h4 h5
*** Starting controller
c0
*** Starting 2 switches
s1 s2
*** Starting CLI:
```

Running the same test cases as for PART A, we get:

- Attempts to send from 10.0.1.2 to an unknown address range like 10.99.0.1 should yield an ICMP destination unreachable message.

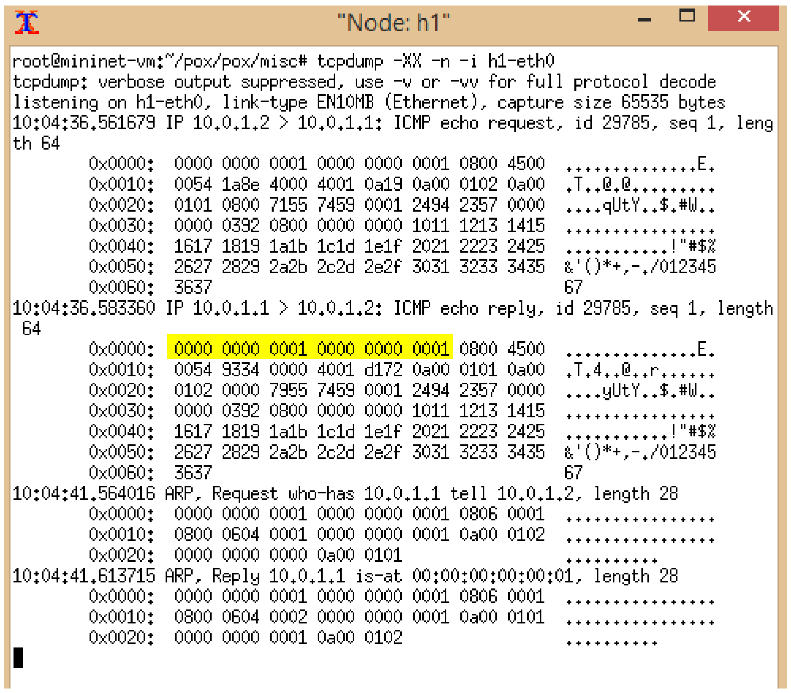


```
mininet> h1 ping -c1 10.99.0.1
PING 10.99.0.1 (10.99.0.1) 56(84) bytes of data.
From 10.0.1.2 icmp_seq=1 Destination Host Unreachable

--- 10.99.0.1 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

mininet>
```

As seen the packets are dropped as the host 10.99.0.1 is not present in the controller. Thus like before an ICMP message is sent to the switch which passes it to the host (h1)

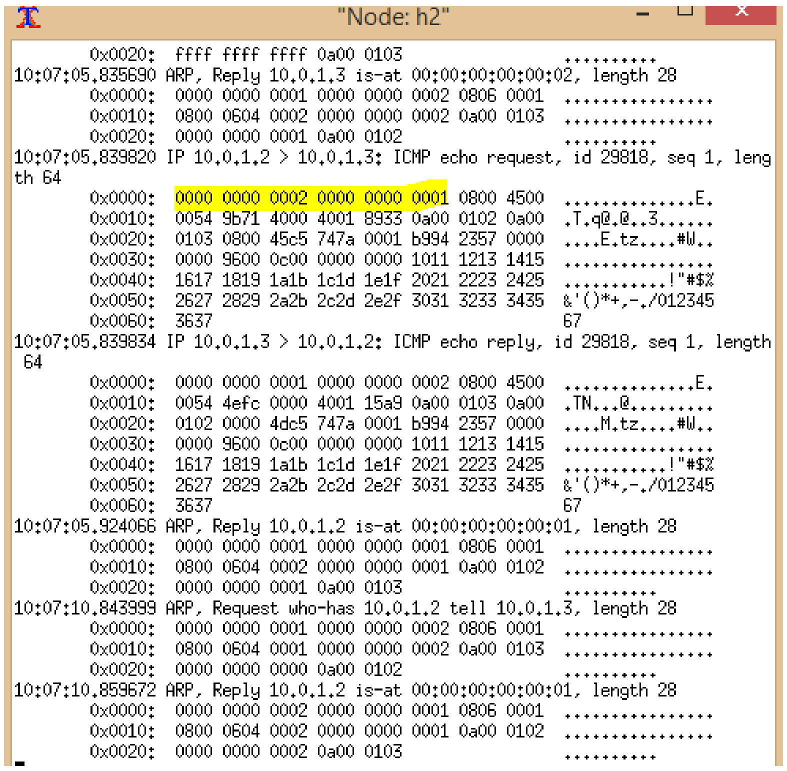
- Packets sent to hosts on a known address range should have their MAC destination field changed to that of the next-hop router.

Like for part A we get similar outputs in part B after doing tcpdump in host1, this is hosw by the screenhot below:



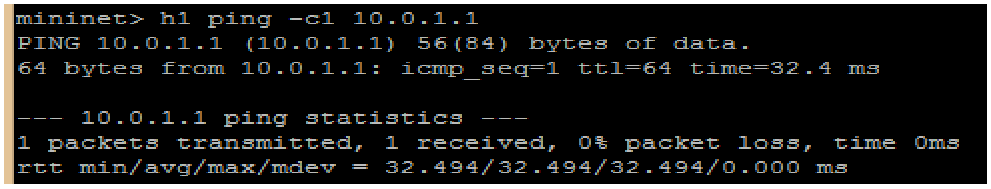
```
root@mininet-vm:~/pox/pox/misc# tcpdump -XX -n -i h1-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h1-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
10:04:36.561679 IP 10.0.1.2 > 10.0.1.1: ICMP echo request, id 29785, seq 1, leng
th 64
        0x0000:  0000 0000 0001 0000 0000 0001 0800 4500  ..............E.
        0x0010:  0054 1a8e 4000 4001 0a19 0a00 0102 0a00  .T..@.@.........
        0x0020:  0101 0800 7155 7459 0001 2494 2357 0000  ....qUtY..$.#W..
        0x0030:  0000 0392 0800 0000 0000 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
10:04:36.583360 IP 10.0.1.1 > 10.0.1.2: ICMP echo reply, id 29785, seq 1, length
 64
        0x0000:  0000 0000 0001 0000 0000 0001 0800 4500  ..............E.
        0x0010:  0054 9334 0000 4001 d172 0a00 0101 0a00  .T.4..@..r......
        0x0020:  0102 0000 7955 7459 0001 2494 2357 0000  ....yUtY..$.#W..
        0x0030:  0000 0392 0800 0000 0000 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
10:04:41.564016 ARP, Request who-has 10.0.1.1 tell 10.0.1.2, length 28
        0x0000:  0000 0000 0001 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0102  ................
        0x0020:  0000 0000 0000 0a00 0101                 ..........
10:04:41.613715 ARP, Reply 10.0.1.1 is-at 00:00:00:00:00:01, length 28
        0x0000:  0000 0000 0001 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0001 0a00 0101  ................
        0x0020:  0000 0000 0001 0a00 0102                 ..........
```

Likewise a tcpdump in h2 gives the following messages:



```
        0x0020:  ffff ffff ffff 0a00 0103                 ..........
10:07:05.835690 ARP, Reply 10.0.1.3 is-at 00:00:00:00:00:02, length 28
        0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0002 0a00 0103  ................
        0x0020:  0000 0000 0001 0a00 0102                 ..........
10:07:05.839820 IP 10.0.1.2 > 10.0.1.3: ICMP echo request, id 29818, seq 1, leng
th 64
        0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  ..............E.
        0x0010:  0054 9b71 4000 4001 8933 0a00 0102 0a00  .T.q@.@..3......
        0x0020:  0103 0800 45c5 747a 0001 b994 2357 0000  ....E.tz....#W..
        0x0030:  0000 9600 0c00 0000 0000 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
10:07:05.839834 IP 10.0.1.3 > 10.0.1.2: ICMP echo reply, id 29818, seq 1, length
 64
        0x0000:  0000 0000 0001 0000 0000 0002 0800 4500  ..............E.
        0x0010:  0054 4efc 0000 4001 15a9 0a00 0103 0a00  .TN...@.........
        0x0020:  0102 0000 4dc5 747a 0001 b994 2357 0000  ....M.tz....#W..
        0x0030:  0000 9600 0c00 0000 0000 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
10:07:05.924066 ARP, Reply 10.0.1.2 is-at 00:00:00:00:00:01, length 28
        0x0000:  0000 0000 0001 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0001 0a00 0102  ................
        0x0020:  0000 0000 0001 0a00 0103                 ..........
10:07:10.843999 ARP, Request who-has 10.0.1.2 tell 10.0.1.3, length 28
        0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0002 0a00 0103  ................
        0x0020:  0000 0000 0000 0a00 0102                 ..........
10:07:10.859672 ARP, Reply 10.0.1.2 is-at 00:00:00:00:00:01, length 28
        0x0000:  0000 0000 0002 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0001 0a00 0102  ................
        0x0020:  0000 0000 0002 0a00 0103                 ..........
```

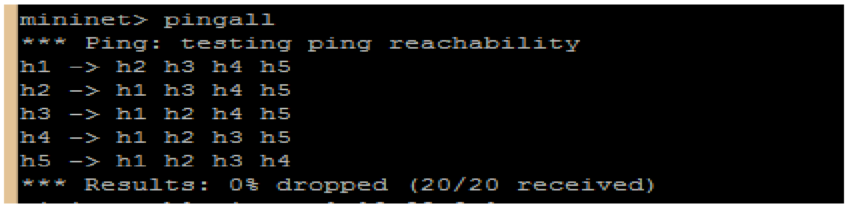
- The router should be pingable, and should generate an ICMP echo reply in response to an ICMP echo request.



```
mininet> h1 ping -c1 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=32.4 ms

--- 10.0.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 32.494/32.494/32.494/0.000 ms
```

As seen from the above screenshot there is an ICMP ECHO reply message in response to the ICMP echo request. The ping from host h1 to its own router 10.0.1.1 sends a ICMP ECHO reply message and the ping is successful.

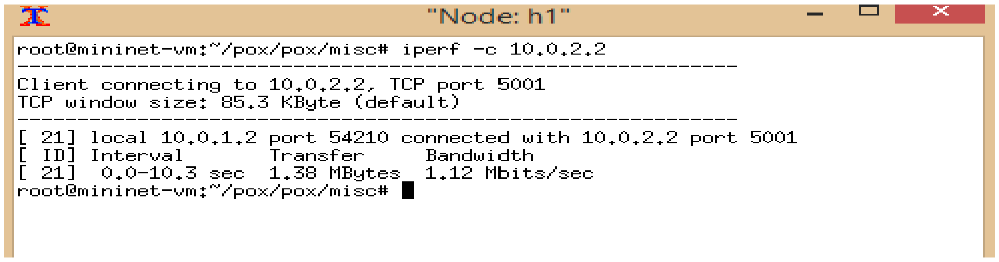
Pingall also works fine as shown below:



```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
```

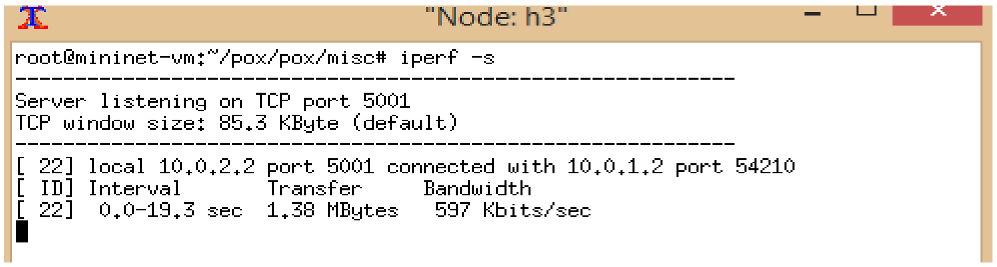
Now running iperf to test TCP and UDP traffic, I open xterm in host 1 and host 3 as shown below:

Here we are running iperf to test TCP connection. As seen below the bandwidth is 1.12 Mbits/sec



"Node: h1"

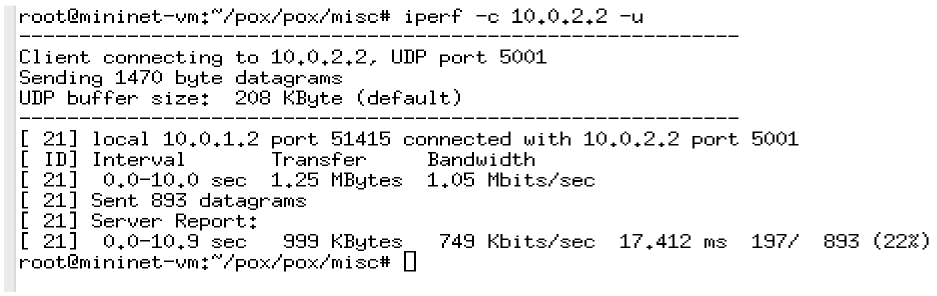
```
root@mininet-vm:~/pox/pox/misc# iperf -c 10.0.2.2
------------------------------------------------------------
Client connecting to 10.0.2.2, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 21] local 10.0.1.2 port 54210 connected with 10.0.2.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 21]  0.0-10.3 sec  1.38 MBytes  1.12 Mbits/sec
root@mininet-vm:~/pox/pox/misc#
```



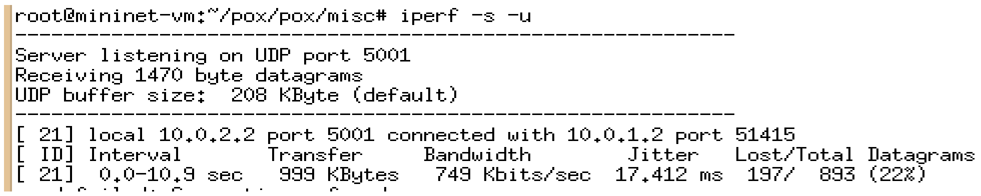
"Node: h3"

```
root@mininet-vm:~/pox/pox/misc# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 22] local 10.0.2.2 port 5001 connected with 10.0.1.2 port 54210
[ ID] Interval       Transfer     Bandwidth
[ 22]  0.0-19.3 sec  1.38 MBytes   597 Kbits/sec
```

Similarly checking the UDP traffic: Here the bandwidth is 1.05 Mbits/sec. The reason for which is the same.

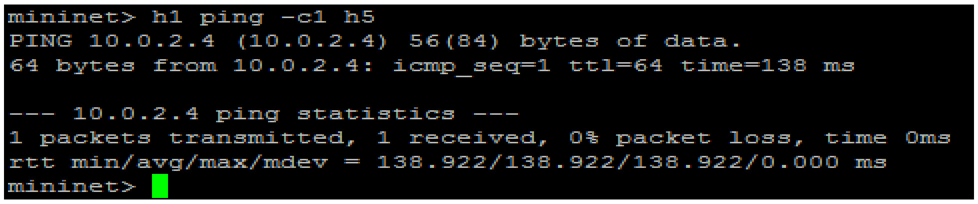


```
root@mininet-vm:~/pox/pox/misc# iperf -c 10.0.2.2 -u
------------------------------------------------------------
Client connecting to 10.0.2.2, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 21] local 10.0.1.2 port 51415 connected with 10.0.2.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 21]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[ 21] Sent 893 datagrams
[ 21] Server Report:
[ 21]  0.0-10.9 sec   999 KBytes   749 Kbits/sec  17.412 ms  197/  893 (22%)
root@mininet-vm:~/pox/pox/misc# 
```



```
root@mininet-vm:~/pox/pox/misc# iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 21] local 10.0.2.2 port 5001 connected with 10.0.1.2 port 51415
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[ 21]  0.0-10.9 sec   999 KBytes   749 Kbits/sec  17.412 ms  197/  893 (22%)
```

Pinging to other hosts:



```
mininet> h1 ping -c1 h5
PING 10.0.2.4 (10.0.2.4) 56(84) bytes of data.
64 bytes from 10.0.2.4: icmp_seq=1 ttl=64 time=138 ms

--- 10.0.2.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 138.922/138.922/138.922/0.000 ms
mininet> 
```

<u>Description about the flow of the project:</u>

- There can either be an ARP request or an IP packet which is received. Firstly the ARP cache is empty. As the hosts starts interacting the ARP cache gets filled, however there is timeout for the cache as well. When an ARP request is received, the cache is checked, if a match is found then an ARP reply is sent.
- If the packet is an ARP reply then it simply floods it to all the hosts.
- In case of an IP packet whose source IP address is not present in the cache, it is added to the cache and then forwarded to the destination. The destination IP address is then checked with the ip_to_port (for its corresponding port) and if a match is found then forward it else send an unreachable message.
- Here as mentioned in the tutorial, we also take care of a message queue. When there is new entry the packets waiting for this are all forwarded.
- In case there is no entry corresponding to a packet in the ARP table then a ARP request message is sent.

<u>Conclusion:</u> For the implementation of this project we use Python and Mininet version 2.2. This was the first project I did on SDN and got to use few new tools like Mininet, Putty. I had few initial problems with wireshark as I was not getting the proper packets. I had no problem installing Mininet, Putty or Xming, it was all very simple. With this tutorial I learnt a lot about how actually in practical switch, hub and routers behave and also got a very good understanding about Software Defined Network.