EE 555 Final Project

Haochen Xie 4759523759

Junquan Yu 3372029142

Scenario 1:

For scenario 1, we don't need to create any topology file. Scenario 1 uses the same topology as the default scenario for the hub which has a switch at center with three hosts connected to it. For the controller part, we modified the "of_tutorial.py" and filled out the missing code in it for the switch part

In our debugging phase, something goes wrong in the sense that the communication cannot be established between these attached hosts when we used the command like "pingall" etc. Using the Wireshark, we captured the packages in the network flow and checked the frame header of them. In combination with the analysis for the flow table using "ovs-ofctl dump-flows s1" command, we modified our program and got what we expected.

At the beginning, we open 2 SSH terminal through putty (windows) with X11 forwarding enabled and kill any running controller by the following command.

$ sudo killall controller

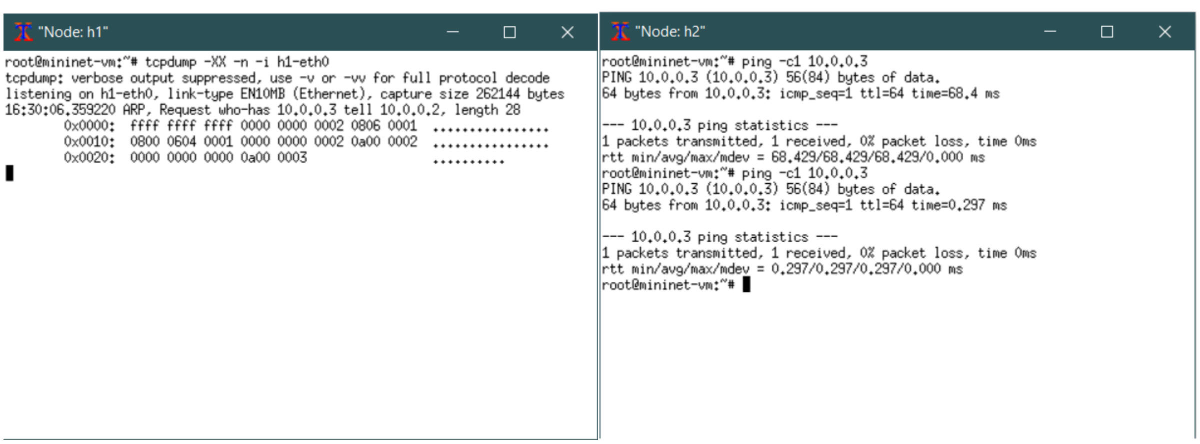
Then we open another SSH terminal and run the following command to start controller.

$ ./pox.py log.level --DEBUG misc.of_tutorial

Also, we run "sudo mn -c" to clean the potential remaining process. Run the following command to initiate the default topology.

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

In the test phase, we use command "xterm h1 h2 h3" to run 3 hosts in terminal separately. Holding the host1 and host 3 on listening status, we let h2 ping host3 for twice. From the output, we can see that host2 flood ARP request frame before sending the frame to host3 and the switch also flood it once receiving it. For the second ping, host2 can directly send the frame to the host3 without ARP requesting. Apart from this, we can see from the pox terminal that the flow table is successfully updated once receiving the ARP request and ARP reply packet.



"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 262144 bytes
16:30:06.359218 ARP, Request who-has 10.0.0.3 tell 10.0.0.2, length 28
        0x0000:  ffff ffff ffff 0000 0000 0002 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0002 0a00 0002  ................
        0x0020:  0000 0000 0000 0a00 0003                 ..........
16:30:06.359232 ARP, Reply 10.0.0.3 is-at 00:00:00:00:00:03, length 28
        0x0000:  0000 0000 0002 0000 0000 0003 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0003 0a00 0003  ................
        0x0020:  0000 0000 0002 0a00 0002                 ..........
16:30:06.406434 IP 10.0.0.2 > 10.0.0.3: ICMP echo request, id 15355, seq 1, leng
th 64
        0x0000:  0000 0000 0003 0000 0000 0002 0800 4500  ..............E.
        0x0010:  0054 f820 4000 4001 2e84 0a00 0002 0a00  .T..@.@.........
        0x0020:  0003 0800 a442 3bfb 0001 8e66 e05d 0000  .....B;....f.]..
        0x0030:  0000 e529 0500 0000 0000 1011 1213 1415  ...)............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
16:30:06.406452 IP 10.0.0.3 > 10.0.0.2: ICMP echo reply, id 15355, seq 1, length
 64
        0x0000:  0000 0000 0002 0000 0000 0003 0800 4500  ..............E.
        0x0010:  0054 2712 0000 4001 3f93 0a00 0003 0a00  .T'...@.?.......
        0x0020:  0002 0000 ac42 3bfb 0001 8e66 e05d 0000  .....B;....f.]..
        0x0030:  0000 e529 0500 0000 0000 1011 1213 1415  ...)............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
16:30:11.422796 ARP, Request who-has 10.0.0.2 tell 10.0.0.3, length 28
        0x0000:  0000 0000 0002 0000 0000 0003 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0003 0a00 0003  ................
        0x0020:  0000 0000 0000 0a00 0002                 ..........
16:30:11.423256 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
        0x0000:  0000 0000 0003 0000 0000 0002 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0002 0a00 0002  ................
        0x0020:  0000 0000 0003 0a00 0003                 ..........
17:13:07.240584 IP 10.0.0.2 > 10.0.0.3: ICMP echo request, id 15947, seq 1, leng
th 64
        0x0000:  0000 0000 0003 0000 0000 0002 0800 4500  ..............E.
        0x0010:  0054 da31 4000 4001 4c73 0a00 0002 0a00  .T.1@.@.Ls......
        0x0020:  0003 0800 1267 3e4b 0001 a370 e05d 0000  .....g>K...p.]..
        0x0030:  0000 61ab 0300 0000 0000 1011 1213 1415  ..a.............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
17:13:07.240593 IP 10.0.0.3 > 10.0.0.2: ICMP echo reply, id 15947, seq 1, length
 64
        0x0000:  0000 0000 0002 0000 0000 0003 0800 4500  ..............E.
        0x0010:  0054 af1b 0000 4001 b789 0a00 0003 0a00  .T....@.........
        0x0020:  0002 0000 1a67 3e4b 0001 a370 e05d 0000  .....g>K...p.]..
        0x0030:  0000 61ab 0300 0000 0000 1011 1213 1415  ..a.............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
```

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/Oct 26 2016 20:30:19)
DEBUG:core:Platform is Linux-4.2.0-27-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:[None 1] closed
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-01 2]
DEBUG:misc.of_tutorial:--------------------------------------------------
DEBUG:misc.of_tutorial:Current map: {}
DEBUG:misc.of_tutorial:src: 00:00:00:00:00:02 dst: ff:ff:ff:ff:ff:ff
DEBUG:misc.of_tutorial:Learning that 00:00:00:00:00:02 is attached to Port 2
DEBUG:misc.of_tutorial:Flooding!
DEBUG:misc.of_tutorial:--------------------------------------------------
DEBUG:misc.of_tutorial:Current map: {EthAddr('00:00:00:00:00:02'): 2}
DEBUG:misc.of_tutorial:src: 00:00:00:00:00:03 dst: 00:00:00:00:00:02
DEBUG:misc.of_tutorial:Learning that 00:00:00:00:00:03 is attached to Port 3
DEBUG:misc.of_tutorial:Forwarding the packet to port 2
DEBUG:misc.of_tutorial:Installing flow with destination 00:00:00:00:00:02 to Por
t 2
DEBUG:misc.of_tutorial:--------------------------------------------------
DEBUG:misc.of_tutorial:Current map: {EthAddr('00:00:00:00:00:02'): 2, EthAddr('0
0:00:00:00:00:03'): 3}
DEBUG:misc.of_tutorial:src: 00:00:00:00:00:02 dst: 00:00:00:00:00:03
DEBUG:misc.of_tutorial:Forwarding the packet to port 3
DEBUG:misc.of_tutorial:Installing flow with destination 00:00:00:00:00:03 to Por
t 3
```

After that, finally we check the connectivity between every single host using the command "pingall". And we also find transmission rate for switch is significantly faster than hub.

```
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: ['5.61 Gbits/sec', '5.61 Gbits/sec']
mininet>
```

Scenario 2

For scenario 2, we need to create two python files. One file that has the configuration of the topology and the other file that contains the code for the controller. The packet forwarding switch act as a router with three interfaces attached to it. All the three interfaces of the packet forwarding switch will be in different subnets.

In this case, the router needs to deal with ARP and IP packets. When ARP packets arrive, router would learn the IP, MAC and port mappings. It can just flood ARP request to find destination IP, since there is only one router. As for IP packets, simply forward them to proper port using mapping it learnt.

Actually, in this scenario controller is doing all the sending instead of using OpenFlow to control switch. This was used in scenario 3 and 4.

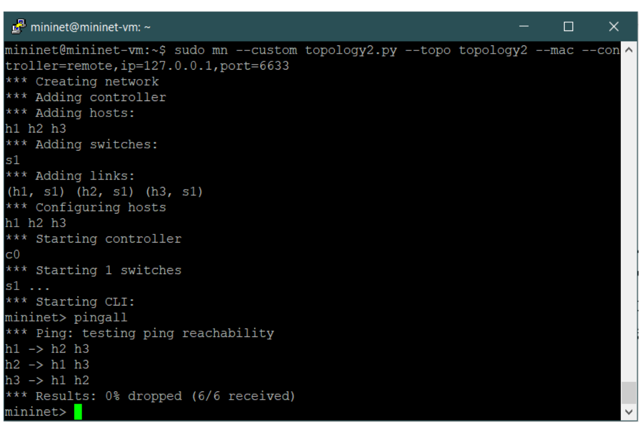
In this part, we open two terminals and type the following command in one terminal to run the controller.

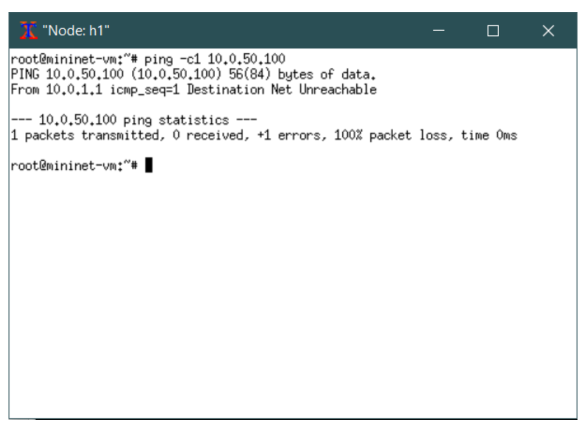
$ sudo ./pox.py log.level --DEBUG misc.controller2

In the other terminal, run topology by entering the following command.

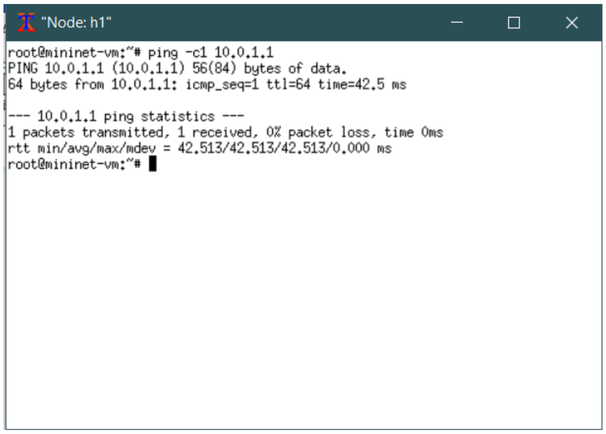
$ sudo mn --custom topology2.py --topo topology2 --mac --controller=remote,ip=127.0.0.1,port=6633

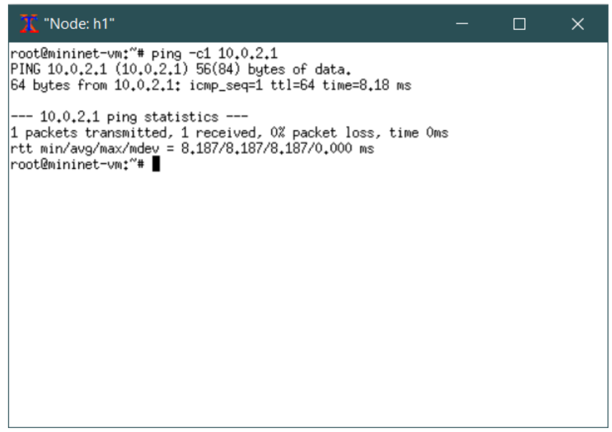
In the test phase, we use command "xterm h1 h2 h3" to run 3 hosts in terminal separately. First, we check the connectivity between every single host using the command "pingall". We also test if router can yield ICMP destination unreachable message when we ping an unknown destination such as 10.0.50.100.



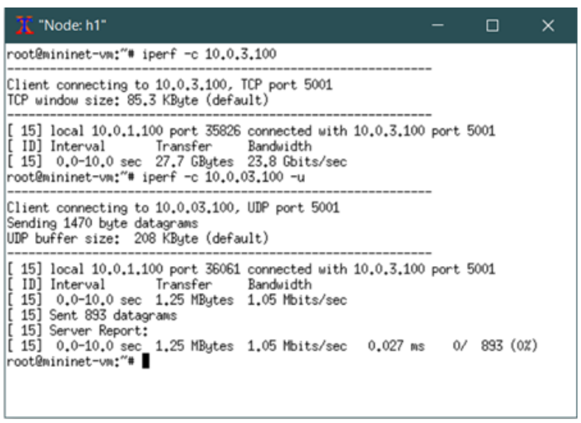


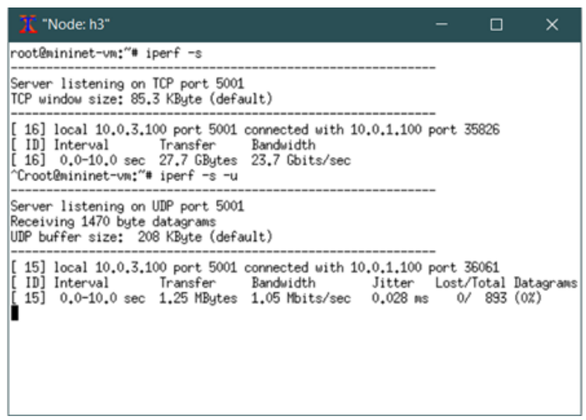
Then, we test whether host1 can communicate with it gateway and other interfaces of this router.





Finally, we use host1 as iperf client and host3 as server, run iperf command to test tcp and udp traffic.





Scenario 3

In this section we will have two packet forwarding switches connected by a single point to point link. One packet forwarding switch has two hosts attached to it with both the hosts in the same network. The second packet forwarding switch has one host attached to it. We need to create two python files. One file that has the configuration of the topology and the other file that contains the code for the controller in this scenario.

The key point here is the controller needs to send rules to more than one switch, which means it needs more connections. Here we use a dictionary to map those connections with DPID in POX. We also treat this number as the number of switches. Also, we use dictionaries to represent all tables. Still, the ARP request can be flooded, since there is no loop.

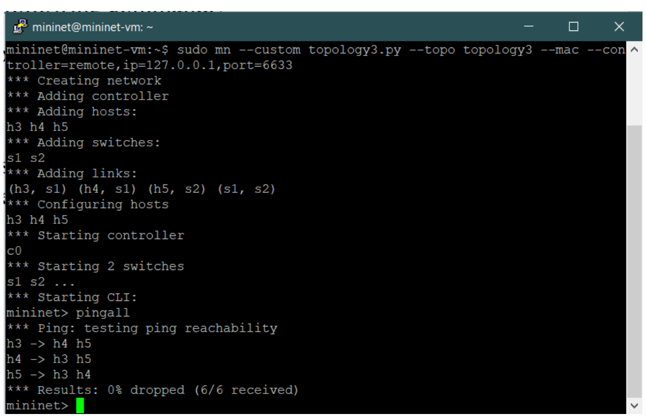
In this section, we open two terminals and type the following command in one terminal to run the controller.

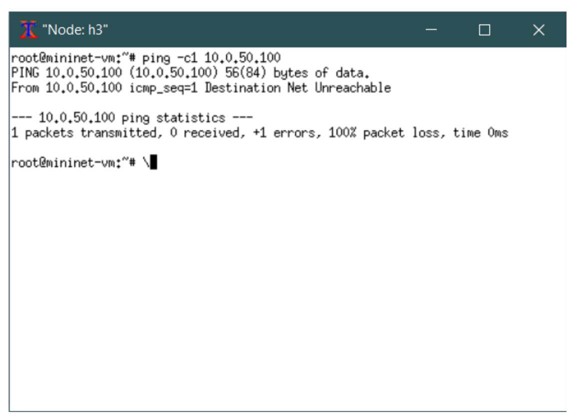
$ sudo ./pox.py log.level --DEBUG misc.controller3

In the other terminal, run topology by entering the following command.

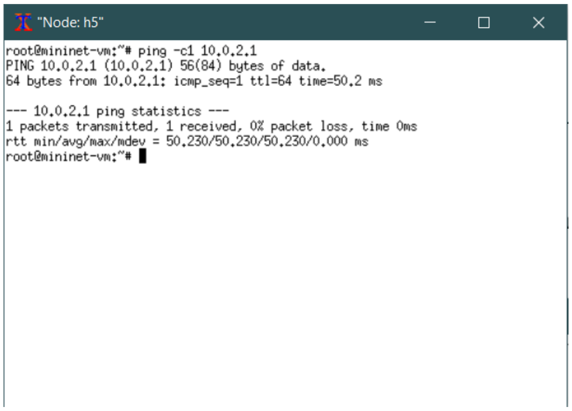
$ sudo mn --custom topology3.py --topo topology3 --mac --controller=remote,ip=127.0.0.1,port=6633

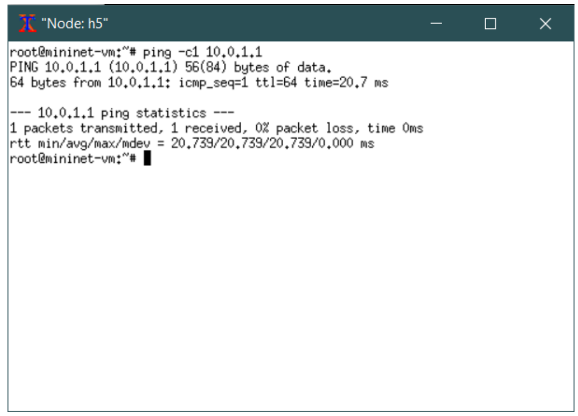
In the test phase, we use command “xterm h1 h2 h3” to run 3 hosts in terminal separately. First, we check the connectivity between every single host using the command “pingall”. We also test if router can yield ICMP destination unreachable message when we ping an unknown destination such as 10.0.50.100.



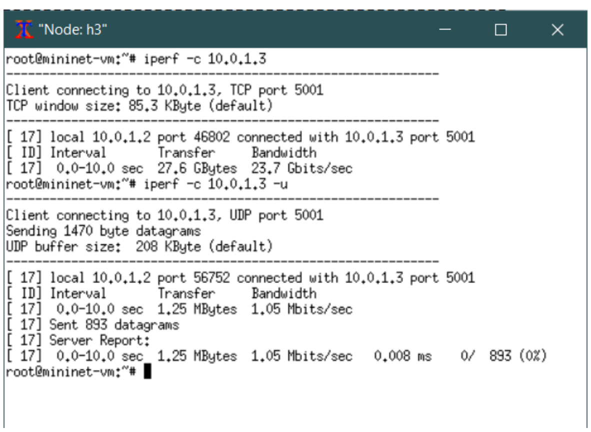


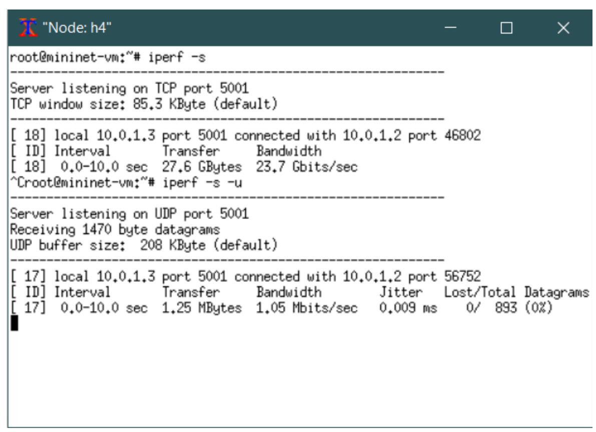
Then, we test whether host5 can communicate with it gateway and the interface of other router.

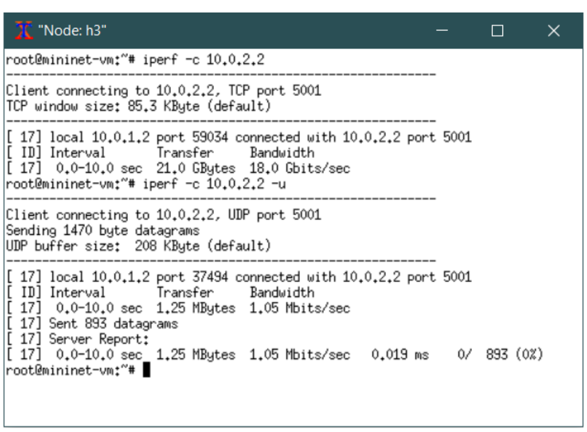


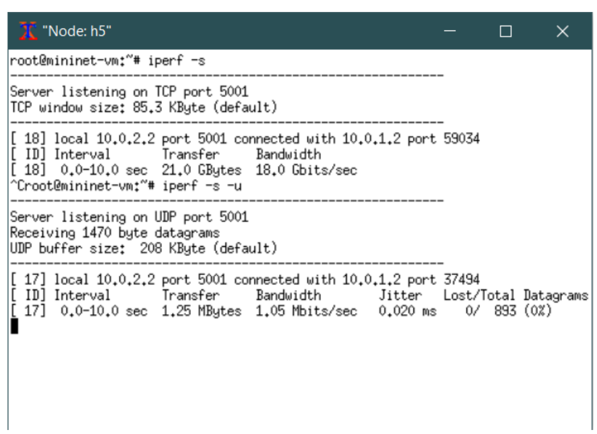


Finally, we use host3 as iperf client and host4 as server, run iperf command to test tcp and udp traffic between two hosts within the same subnet. And we also use host3 as iperf client and host5 as server, run iperf command to test tcp and udp traffic between two hosts within the different subnets.









Scenario 4

In this scenario we will have 3 routers connected to each other in a full mesh topology. Each router will have 3 hosts attached to it. we need to forward the packet internally if the destination host address belongs to the same LAN. If the destination is of not the same LAN, then you need to forward the packet to the nearest router.One file that has the configuration of the topology and the other file that contains the code for the controller in this scenario.

Now simply flooding traffic from router would not do the work anymore, since there is a loop in the topology. Our way is to create an extra condition for ARP packet practice: check whether this packet is destinated for “my” subnet. If it is not, discard it. Otherwise flood it. Proper host in subnet will respond and other routers, in this case, the 3rd router since of_ofp_flood will not forward the packet back to the inport, will discard this.

In this section, we open two terminals and type the following command in one terminal to run the controller.

$ sudo ./pox.py log.level --DEBUG misc.controller4

In the other terminal, run topology by entering the following command.

$ sudo mn --custom topology4.py --topo topology4 --mac --controller=remote,ip=127.0.0.1,port=6633

In the topology for this scenario:

H4 (10.0.1.2) links to the R1 port2 (10.0.1.1)

H5 (10.0.1.3) links to the R1 port3 (10.0.1.1)

H6 (10.0.1.4) links to the R1 port4 (10.0.1.1)

H7 (10.0.2.1) links to the R2 port2 (10.0.2.1)

H8 (10.0.2.2) links to the R2 port3 (10.0.2.1)

H9 (10.0.2.3) links to the R2 port4 (10.0.2.1)

H10 (10.0.3.2) links to the R3 port2 (10.0.3.1)

H11(10.0.3.3) links to the R3 port3 (10.0.3.1)

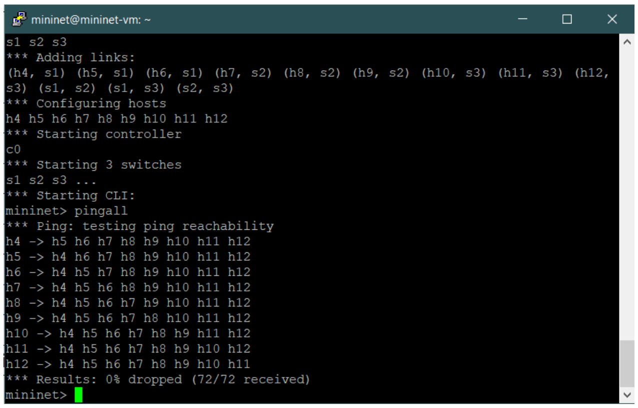
H12 (10.0.3.4) links to the R3 port4 (10.0.3.1)

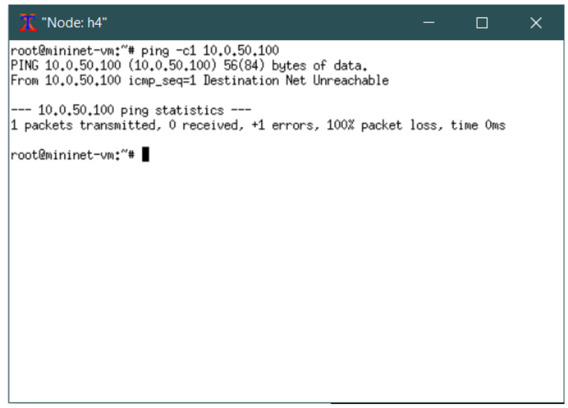
R1 port1 links to R2 port1

R2 port5 links to R3 port5

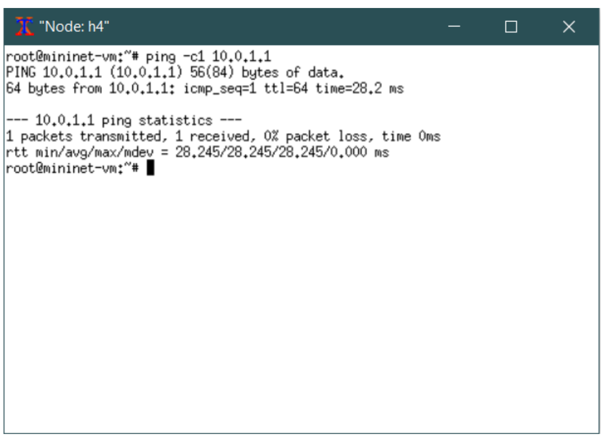
R3 port6 links to R1 port6

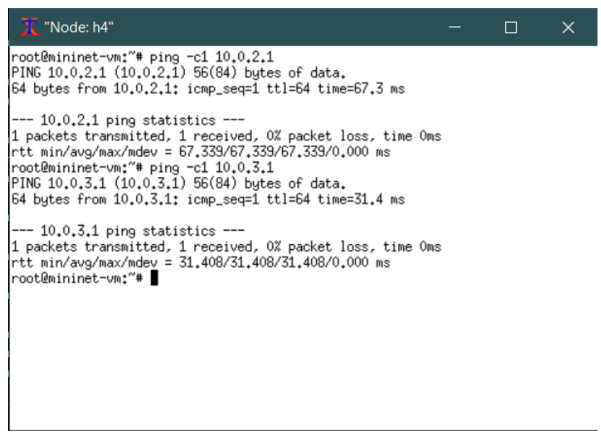
In the test phase, we use command “xterm h1 h2 h3” to run 3 hosts in terminal separately. First, we check the connectivity between every single host using the command “pingall”. We also test if router can yield ICMP destination unreachable message when we ping an unknown destination such as 10.0.50.100.



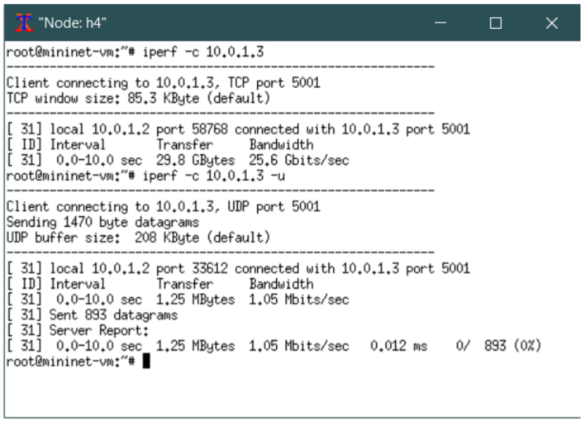


Then, we test whether host4 can communicate with it gateway and the interface of other routers.



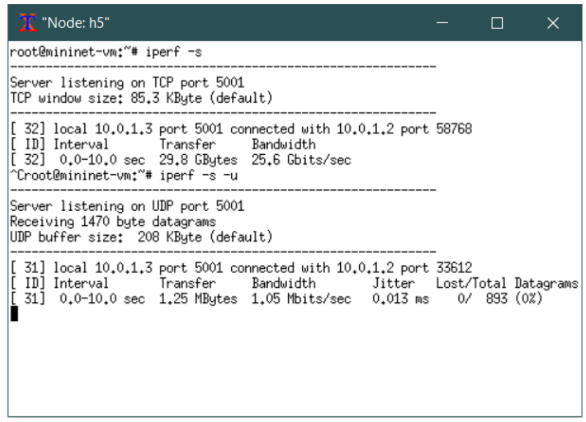


Finally, we use host4 as iperf client and host5 as server, run iperf command to test tcp and udp traffic between two hosts within the same subnet. And we also use host4 as iperf client and host7 as server, run iperf command to test tcp and udp traffic between two hosts within the different subnets.



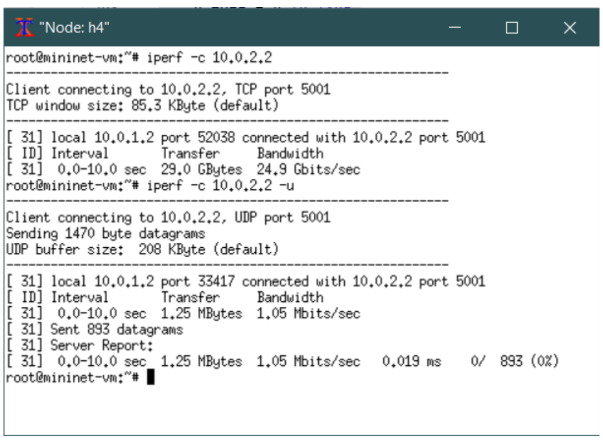
"Node: h4"

```
root@mininet-vm:~# iperf -c 10.0.1.3
------------------------------------------------------------
Client connecting to 10.0.1.3, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 31] local 10.0.1.2 port 58768 connected with 10.0.1.3 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 31]  0.0-10.0 sec  29.8 GBytes  25.6 Gbits/sec
root@mininet-vm:~# iperf -c 10.0.1.3 -u
------------------------------------------------------------
Client connecting to 10.0.1.3, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 31] local 10.0.1.2 port 33612 connected with 10.0.1.3 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 31]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[ 31] Sent 893 datagrams
[ 31] Server Report:
[ 31]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.012 ms    0/  893 (0%)
root@mininet-vm:~#
```



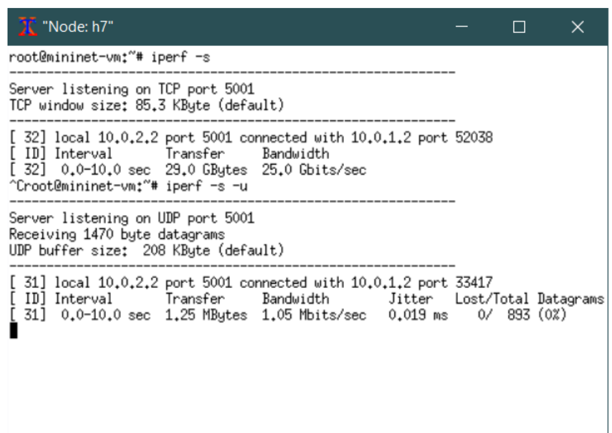
"Node: h5"

```
root@mininet-vm:~# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 32] local 10.0.1.3 port 5001 connected with 10.0.1.2 port 58768
[ ID] Interval       Transfer     Bandwidth
[ 32]  0.0-10.0 sec  29.8 GBytes  25.6 Gbits/sec
^Croot@mininet-vm:~# iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 31] local 10.0.1.3 port 5001 connected with 10.0.1.2 port 33612
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[ 31]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.013 ms    0/  893 (0%)
```



"Node: h4"

```
root@mininet-vm:~# iperf -c 10.0.2.2
------------------------------------------------------------
Client connecting to 10.0.2.2, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 31] local 10.0.1.2 port 52038 connected with 10.0.2.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 31]  0.0-10.0 sec  29.0 GBytes  24.9 Gbits/sec
root@mininet-vm:~# 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)
------------------------------------------------------------
[ 31] local 10.0.1.2 port 33417 connected with 10.0.2.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 31]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[ 31] Sent 893 datagrams
[ 31] Server Report:
[ 31]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.019 ms    0/  893 (0%)
root@mininet-vm:~#
```



"Node: h7"

```
root@mininet-vm:~# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 32] local 10.0.2.2 port 5001 connected with 10.0.1.2 port 52038
[ ID] Interval       Transfer     Bandwidth
[ 32]  0.0-10.0 sec  29.0 GBytes  25.0 Gbits/sec
^Croot@mininet-vm:~# iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 31] local 10.0.2.2 port 5001 connected with 10.0.1.2 port 33417
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[ 31]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.019 ms    0/  893 (0%)
```

Bonus Scenario:

In this scenario, in addition to having the basic Layer 2 functionality, the packet forwarding switch should also act as a Firewall in this case. It should block TCP iperf packets and allow any other kind of traffic between the hosts.

It is easy to achieve this. What you need to do is add rules when connecting to the switch. Ask switch to match packet with source or destination TCP port of a given number, and in our case, 5000, and to drop them.

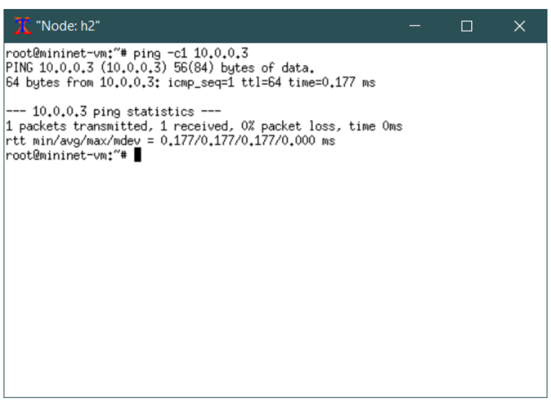
In this section, we open two terminals and type the following command in one terminal to run the controller.

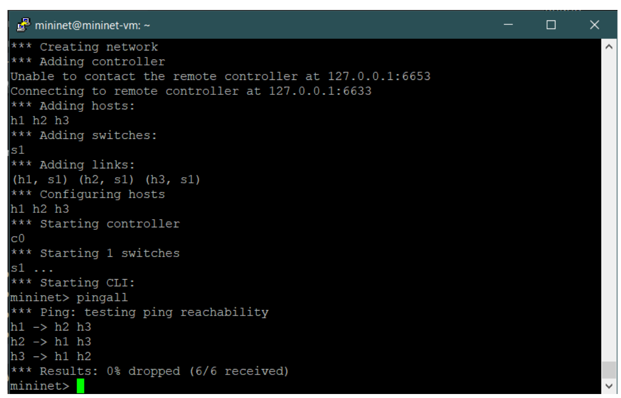
$ sudo ./pox.py log.level --DEBUG misc.controller5

In the other terminal, run topology by entering the following command.

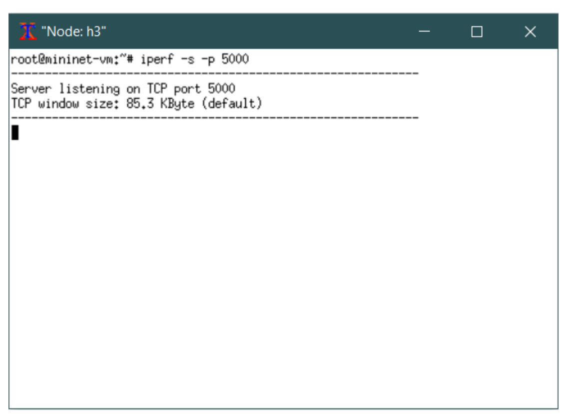
$ sudo mn --topo single,3 --mac --controller remote --switch ovsk

In the test phase, we use command “xterm h1 h2 h3” to run 3 hosts in terminal separately. First, as required by the document, we let the host2 ping the host3 and check the connectivity between every single host using the command “pingall”.





Then we test whether the iperf TCP block takes effect between the host1 and host 3. (we specify the port 5000)



Finally, we test whether the iperf TCP block takes effect between the host1 and host 3.