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

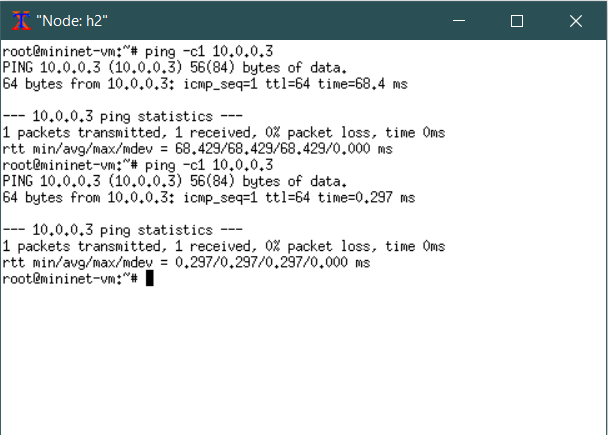
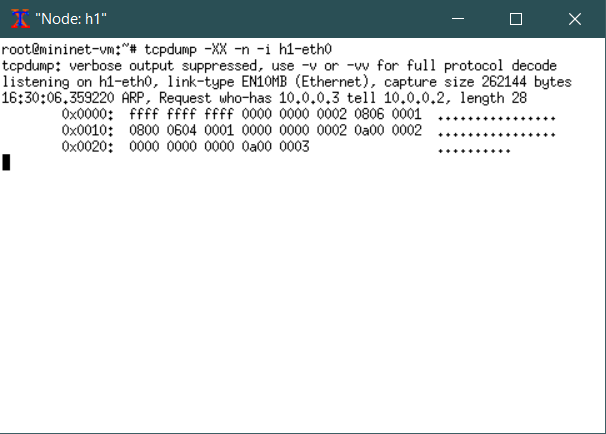
Then we open another SSH terminal and run the following command to start controller.

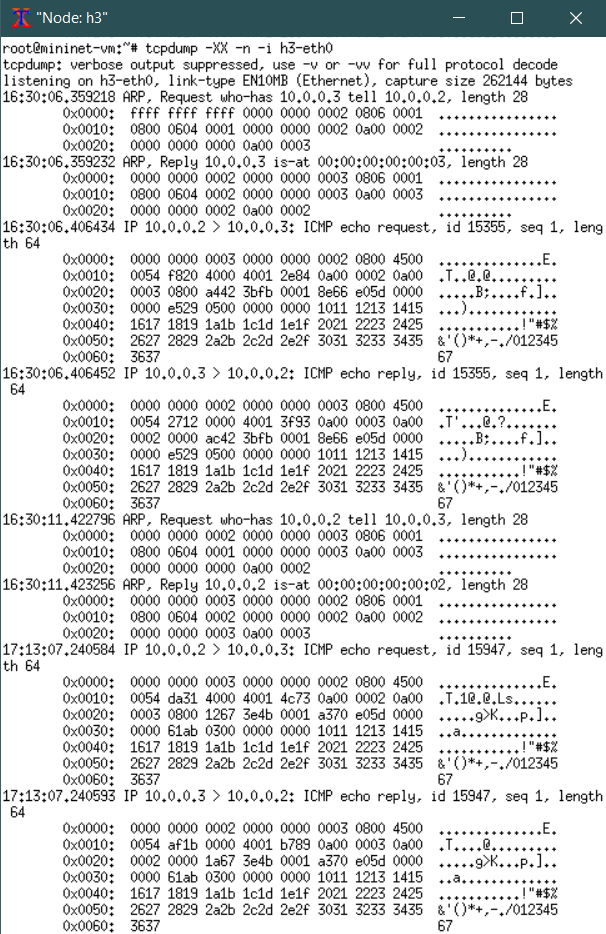
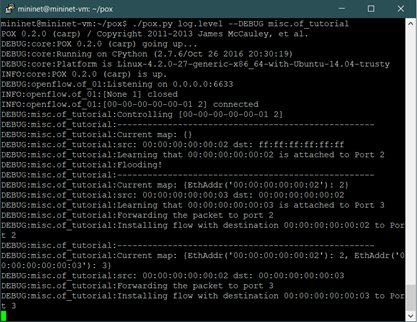
$ ./pox.py log.level --DEBUG misc.of\_tutorial

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

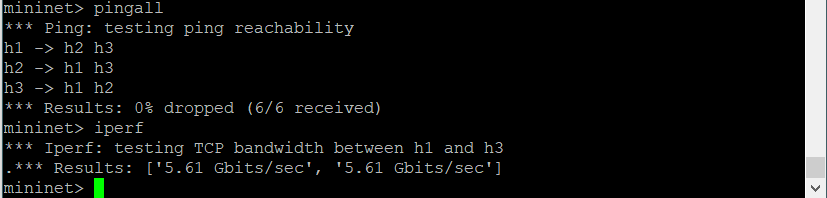
$ sudo mn --topo single,3 --mac --controller remote --switch ovsk

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.





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.



**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.

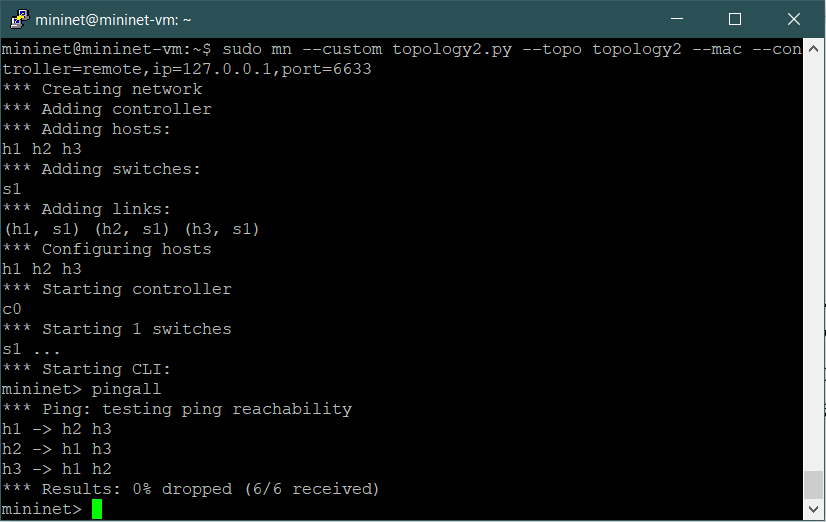
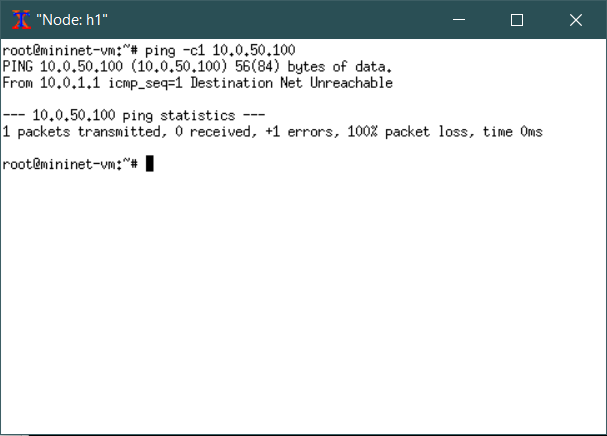
In this part, we open two terminals and type the following command in one terminal to run the controller.

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

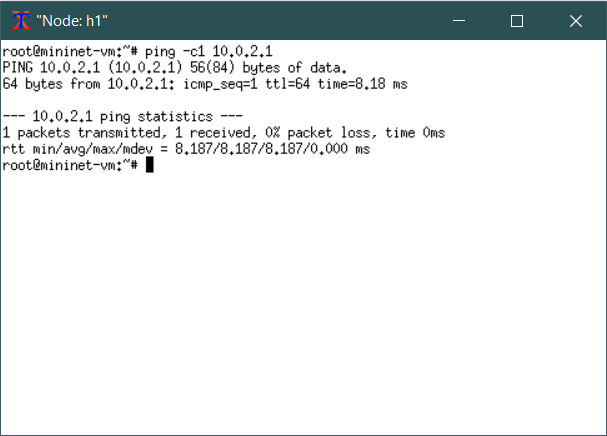
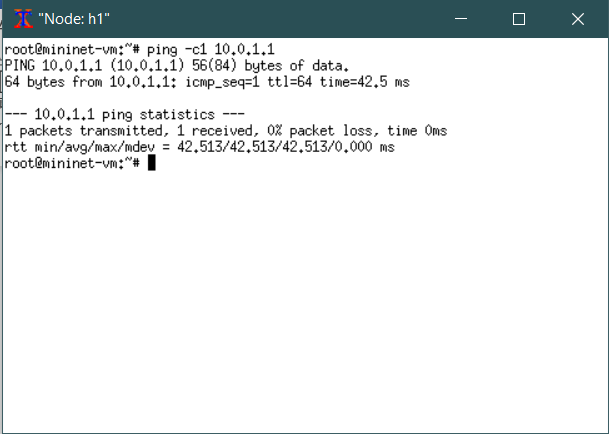
In the other terminal, run topology by entering the following command.

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

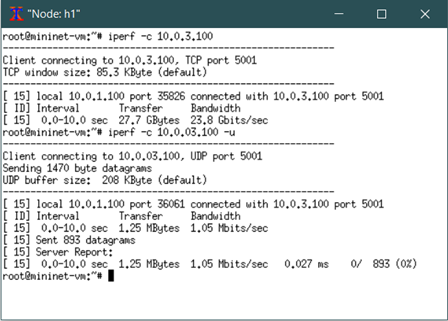
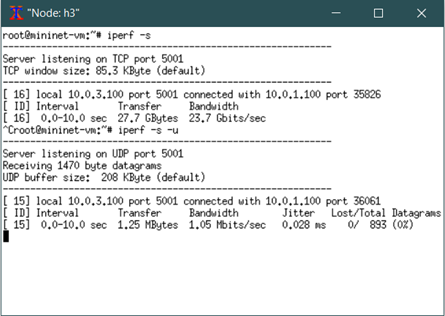
In the test phase, we use command “xterm h1 h2 h3” to run 3 hosts in terminal separately. First, we check the connectivity between every single host using the command “pingall”. We also test if router can yield ICMP destination unreachable message when we ping an unknown destination such as 10.0.50.100.



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.

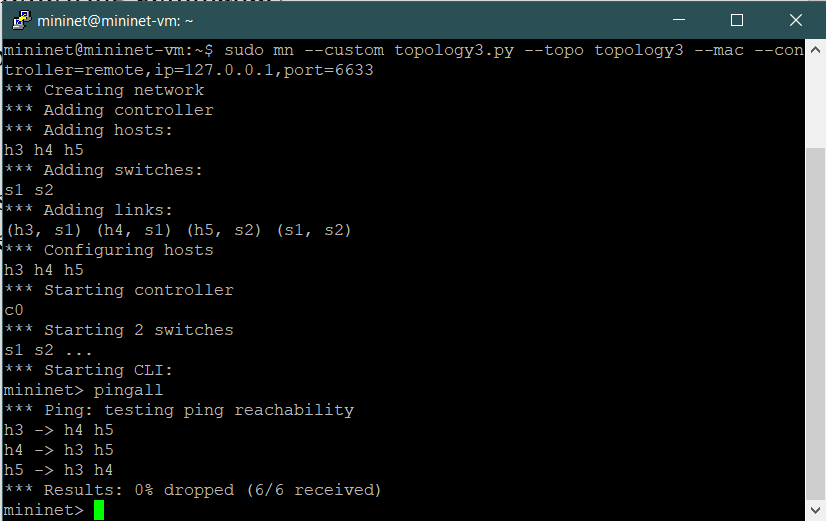
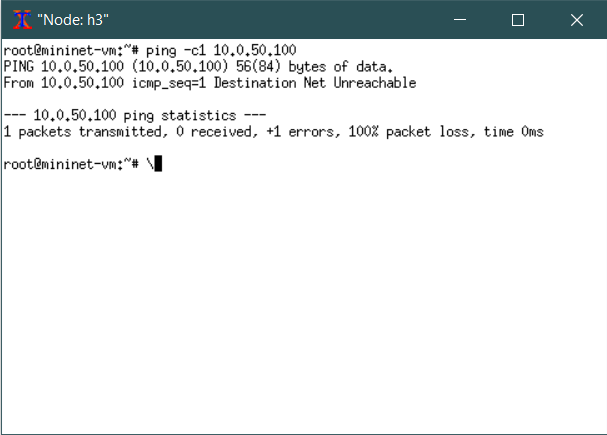
In this section, we open two terminals and type the following command in one terminal to run the controller.

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

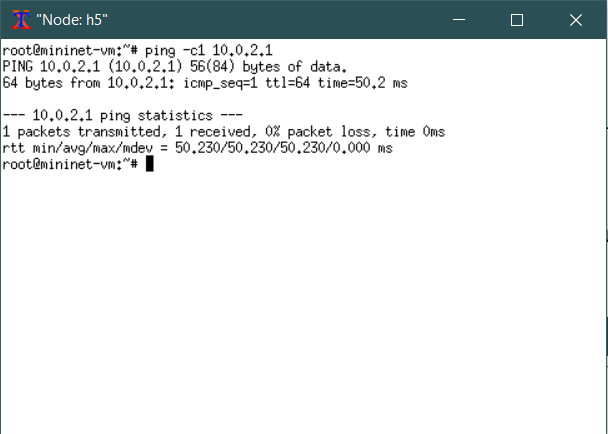
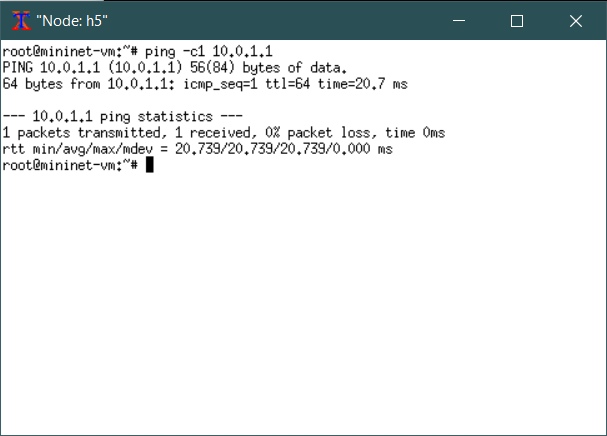
In the other terminal, run topology by entering the following command.

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

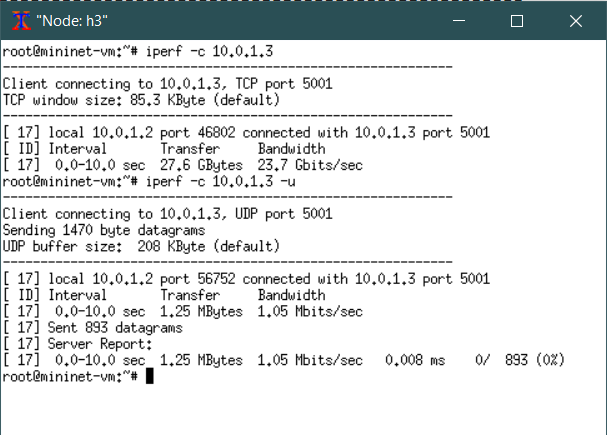
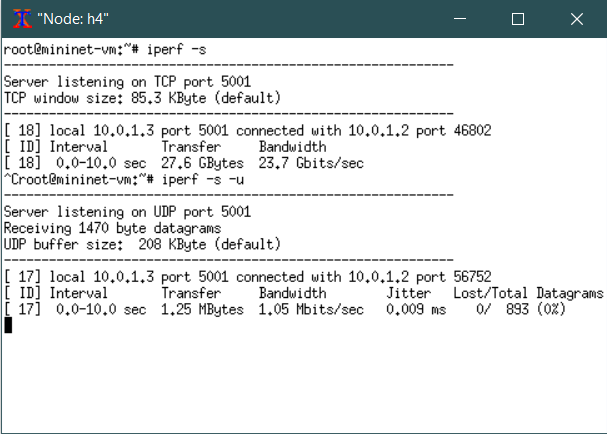
In the test phase, we use command “xterm h1 h2 h3” to run 3 hosts in terminal separately. First, we check the connectivity between every single host using the command “pingall”. We also test if router can yield ICMP destination unreachable message when we ping an unknown destination such as 10.0.50.100.

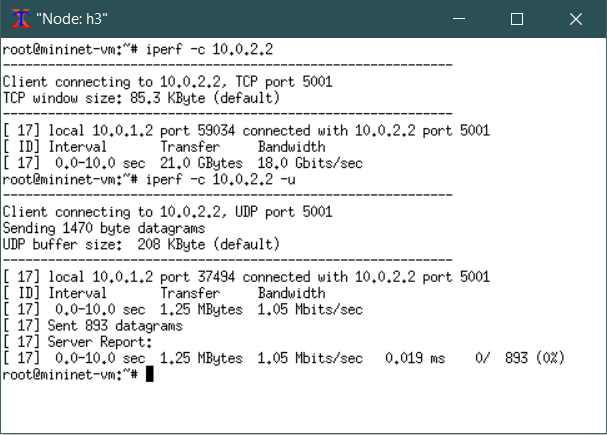
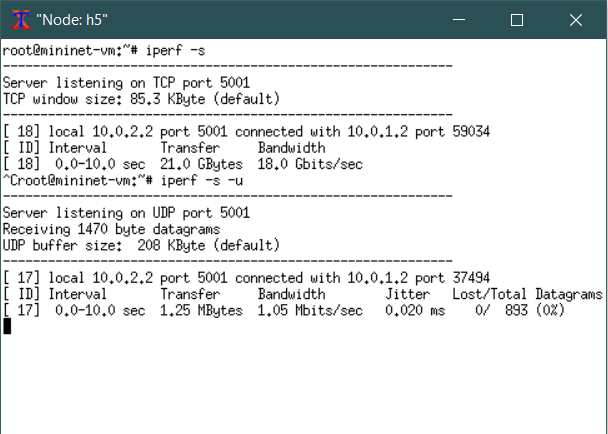


Then, we test whether host5 can communicate with it gateway and the interface of other router.



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.

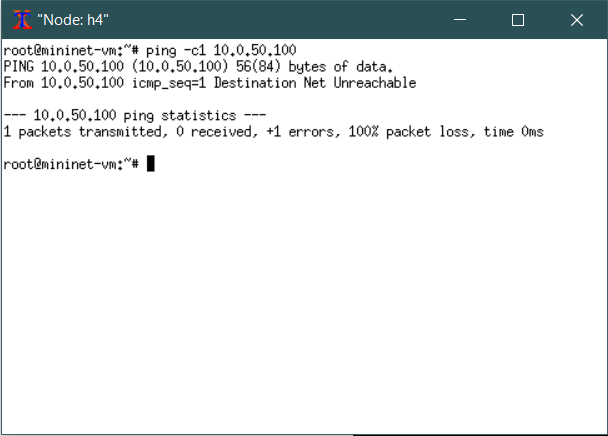
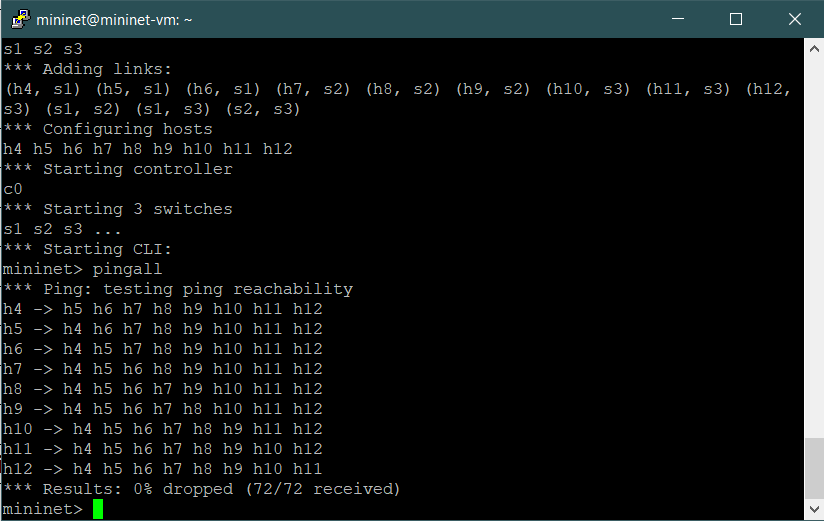
Now simply flooding traffic from router would not do the work anymore, since there is a loop in the topology. Our way is to create an extra condition for ARP packet practice: check whether this packet is destinated for “my” subnet. If it is not, discard it. Otherwise flood it. Proper host in subnet will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

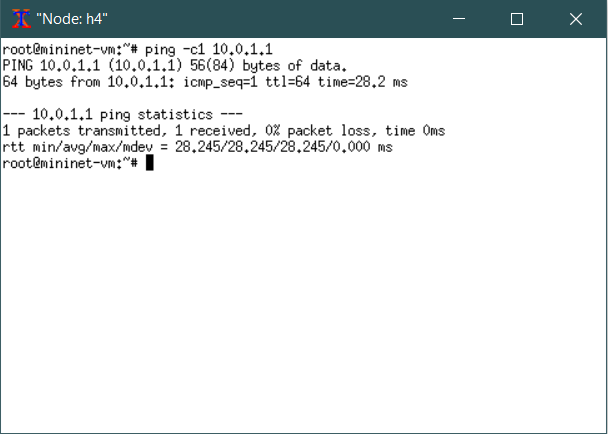
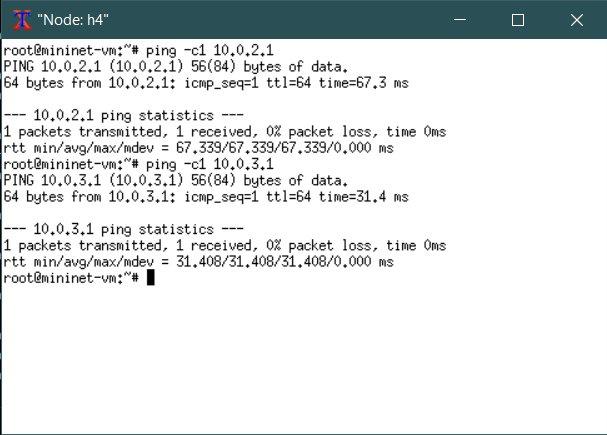
$ sudo ./pox.py log.level --DEBUG misc.controller4

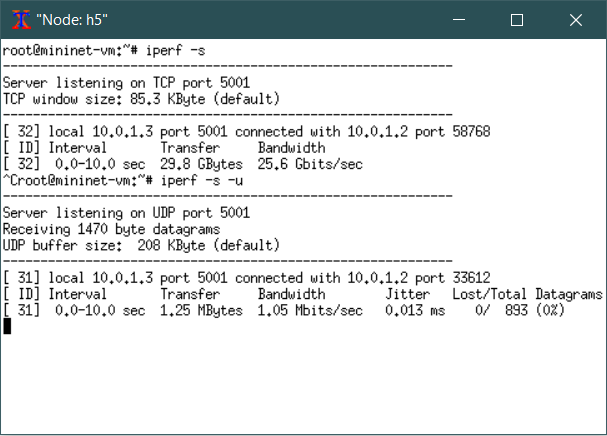
In the other terminal, run topology by entering the following command.

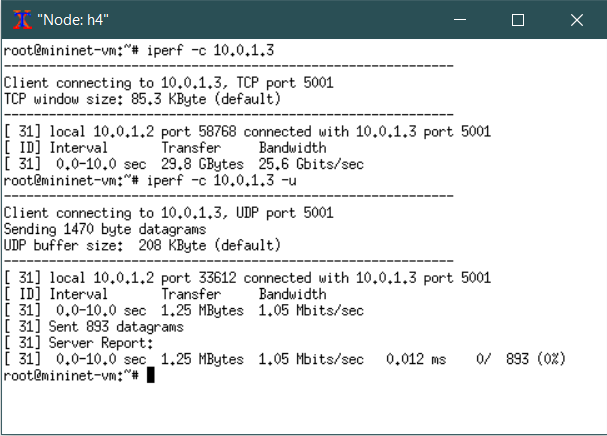
$ sudo mn --custom topology4.py --topo topology4 --mac --controller=remote,ip=127.0.0.1,port=6633

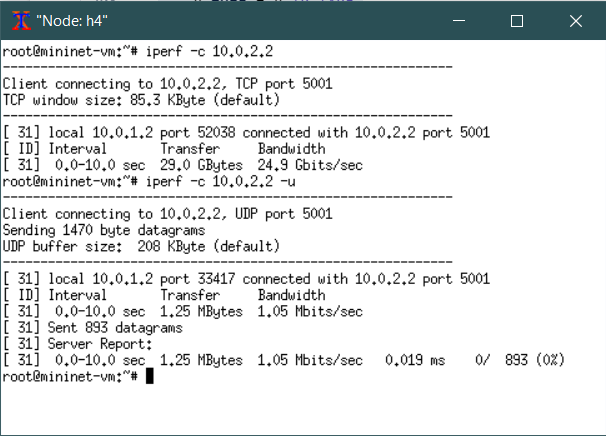
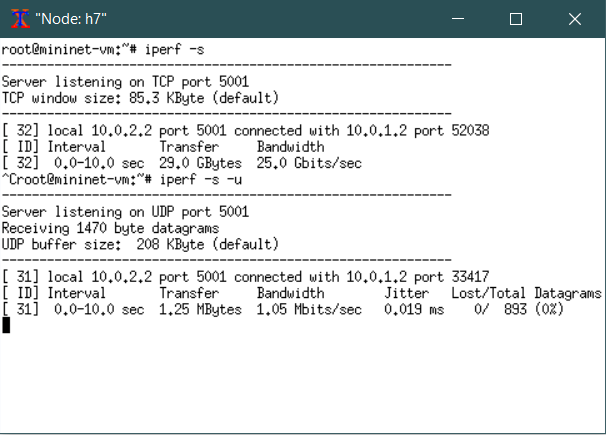
In the test phase, we use command “xterm h1 h2 h3” to run 3 hosts in terminal separately. First, we check the connectivity between every single host using the command “pingall”. We also test if router can yield ICMP destination unreachable message when we ping an unknown destination such as 10.0.50.100.

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.





**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.

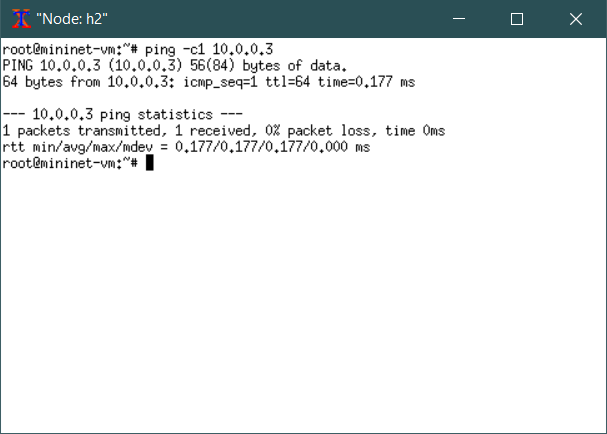
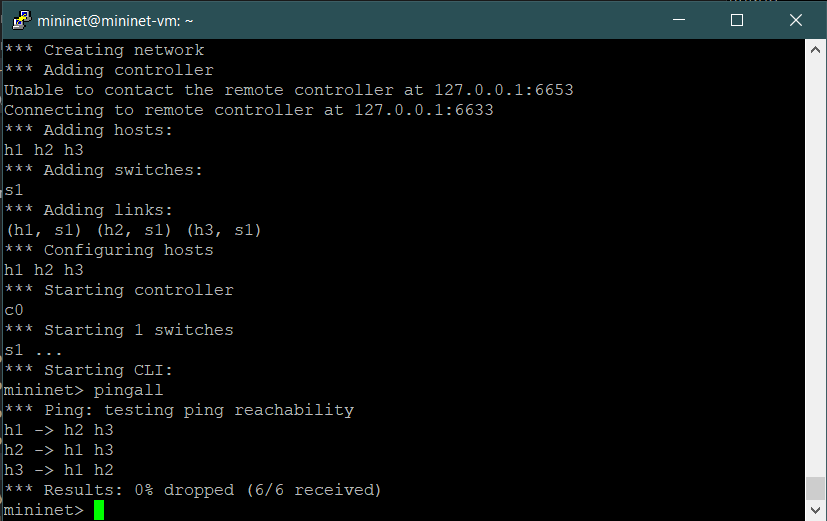
In this section, we open two terminals and type the following command in one terminal to run the controller.

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

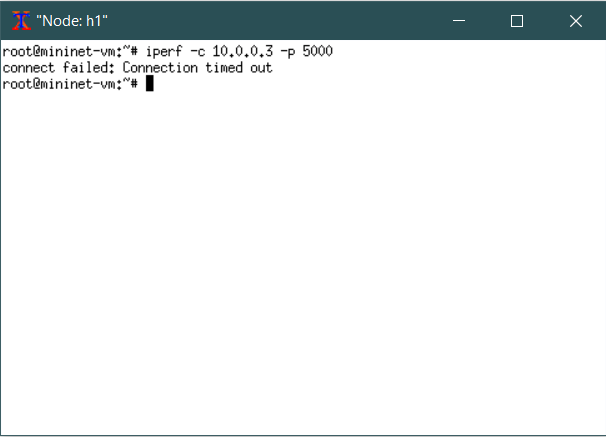
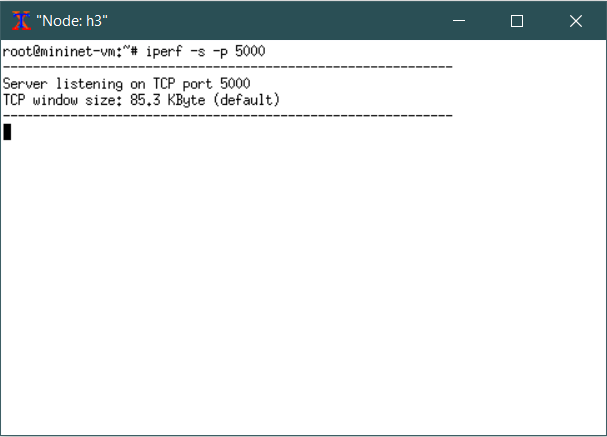
In the other terminal, run topology by entering the following command.

$ 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.

