CSCI 5280 – Software-Defined Networking

Lab 1
SDN Mininet-MiniEdit

University of Colorado Boulder
Department of Computer Science
Network Engineering

Professor Levi Perigo, Ph.D.

Lab Summary

Mininet is the industry leading software network simulator used for SDN testing and troubleshooting. This lab will install Mininet (with a POX controller), and setup a basic learning switch application. This lab is intended to demonstrate basic functionality and understanding.

#NOTE: All commands that start with (#) or ($) must be entered in the Linux terminal. The commands that start with (mininet>) must be entered after starting Mininet from the Linux terminal. Review the following guides for additional information on Mininet and OpenFlow:

1. What is Mininet: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet
2. Here is additional information on Mininet: http://mininet.org/walkthrough/
3. Here is additional information on OpenFlow and Mininet: https://github.com/mininet/openflow-tutorial/wiki

In this lab, you will also learn the basics of MiniEdit, which is the GUI of Mininet. MiniEdit can be used to configure Mininet topology, specify controller locations, and even run Mininet simulations. The intention of this lab is to expose the student to various options in SDN, specifically Mininet. Students are encouraged to use this lab as a guide only, and to expand on the topics for additional learning and experiments.

Objective 1 – Connect to the Mininet VM and Install a POX SDN Controller

1. You will first be in the Linux shell of the VM.
2. Verify that the VM can ping your machine.
3. Verify that the VM has Internet connectivity by pinging 8.8.8.8.
    a. Note: If the previous steps do not work, you can't continue; you must have this working before continuing with this lab.
4. Issue the following command:
    a. $ sudo git clone http://github.com/noxrepo/pox

```
mininet@csci5280-vm2-kima4508:~$ sudo git clone http://github.com/noxrepo/pox
sudo: unable to resolve host csci5280-vm2-kima4508
Cloning into 'pox'...
remote: Enumerating objects: 13058, done.
remote: Counting objects: 100% (283/283), done.
remote: Compressing objects: 100% (113/113), done.
remote: Total 13058 (delta 174), reused 265 (delta 165), pack-reused 12775
Receiving objects: 100% (13058/13058), 5.02 MiB | 0 bytes/s, done.
Resolving deltas: 100% (8423/8423), done.
Checking connectivity... done.
```

2. Provide a screenshot of the POX controller installed successfully. **[5 points]**

```
mininet@csci5280-vm2-kima4508:~/pox$ ./pox.py --verbose forwarding.l2_learning
POX 0.0.0 / Copyright 2011 James McCauley
DEBUG:core:POX 0.0.0 going up...
DEBUG:core:Running on CPython (2.7.6/Nov 12 2018 20:00:40)
INFO:core:POX 0.0.0 is up.
This program comes with ABSOLUTELY NO WARRANTY.  This program is free software,
and you are welcome to redistribute it under certain conditions.
Type 'help(pox.license)' for details.
DEBUG:openflow.of_01:Listening for connections on 0.0.0.0:6633
Ready.
POX>
```

3. Instead of a POX controller, what are two other controllers you could have installed? **[5 points]**

OpenDayLight - popular in industry

Ryu - ease of deployment

4. If you had to choose any controller, which would it be and why? **[5 points]**

- **OpenDayLight -**

  1. It is popular in the industry and there exists a lot of community support which could help in easy troubleshooting
  2. Also, it is extremely modular and customizable allowing developers to add/remove components.
  3. It supports multiple southbound protocols and northbound APIs

Objective 2 - Run a Default Mininet Network

1. Initialize the default Mininet network topology. What command did you use? **[5 points]**

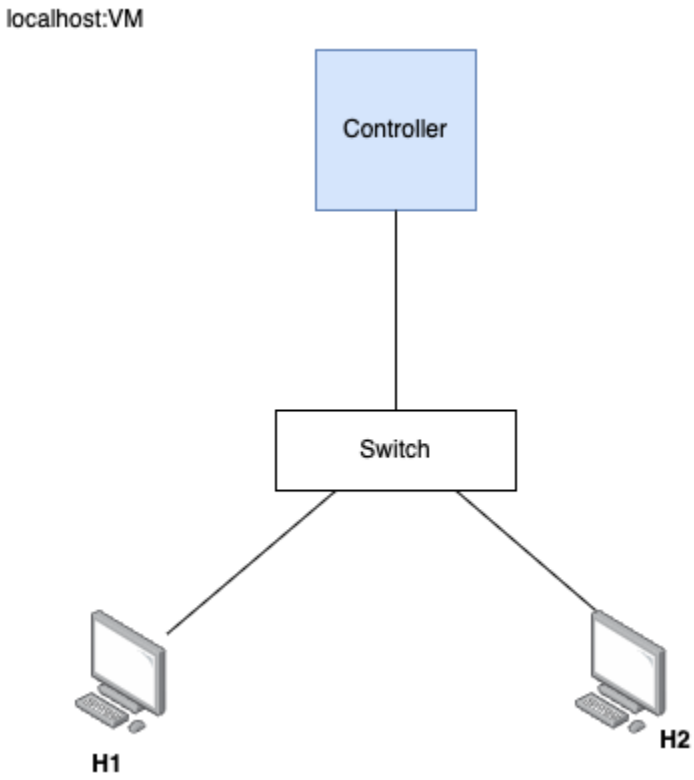
   - **sudo mn** → This command can be used to generate a basic topology, but the controller settings are pointed to localhost and we would be using controller from Mininet namespace.

```
mininet@csci5280-vm2-kima4508:~$ sudo mn
sudo: unable to resolve host csci5280-vm2-kima4508
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> pinall
*** Unknown command: pinall
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

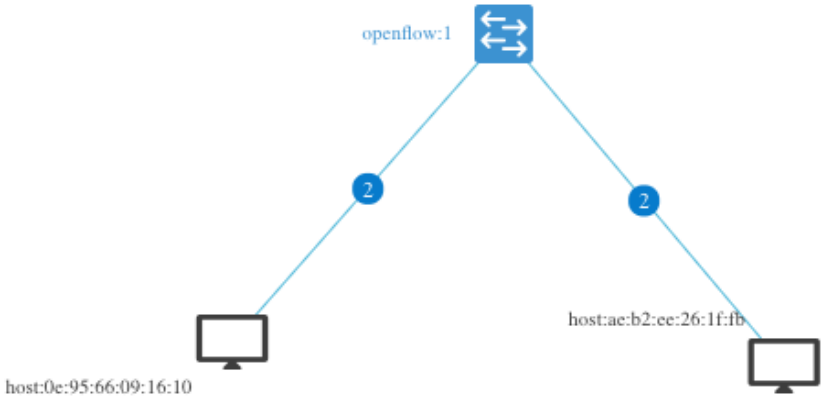
   - **sudo mn --controller=remote,ip=10.224.76.18** → This command with the additional controller option allows setting the remote VM as SDN controller.

   By using the second command, I was able to connect the mininet with the SDN controller to generate a view in the ODL Web UI.

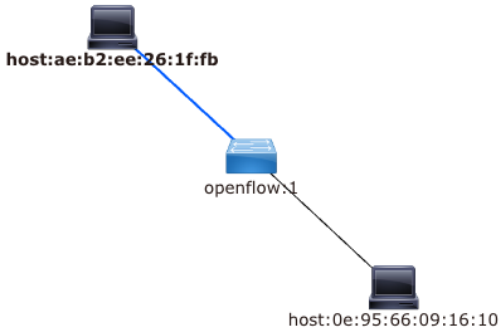
2. Provide/"draw" a rough diagram of the default network that was created. **[5 points]**



Viewed in OpenFlow Manager

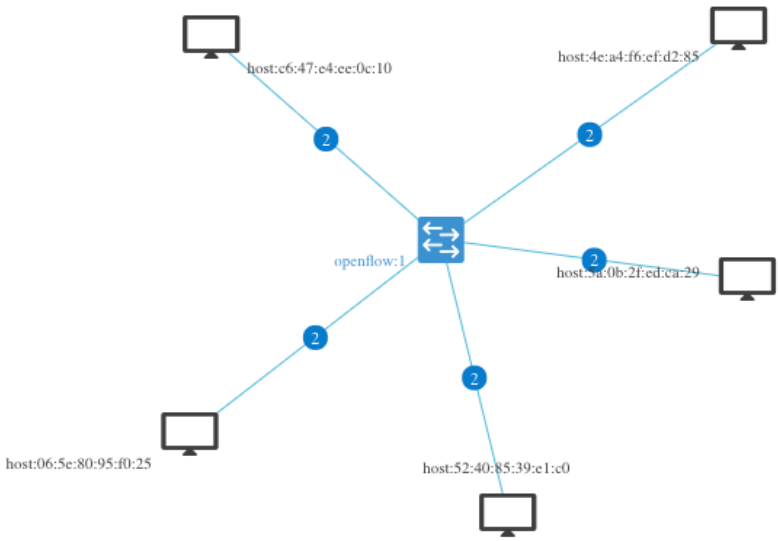


Viewed in OpenDayLight Controller WebUI -

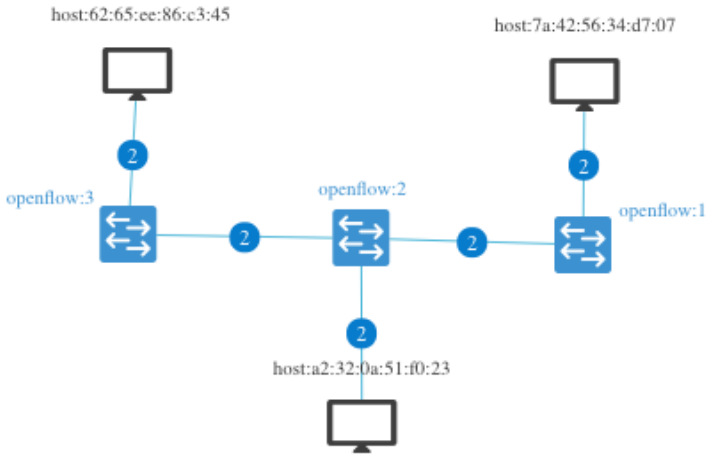


3. What are three network topologies that can be used (except sudo mn) by default in Mininet (provide the commands)? **[5 points]**

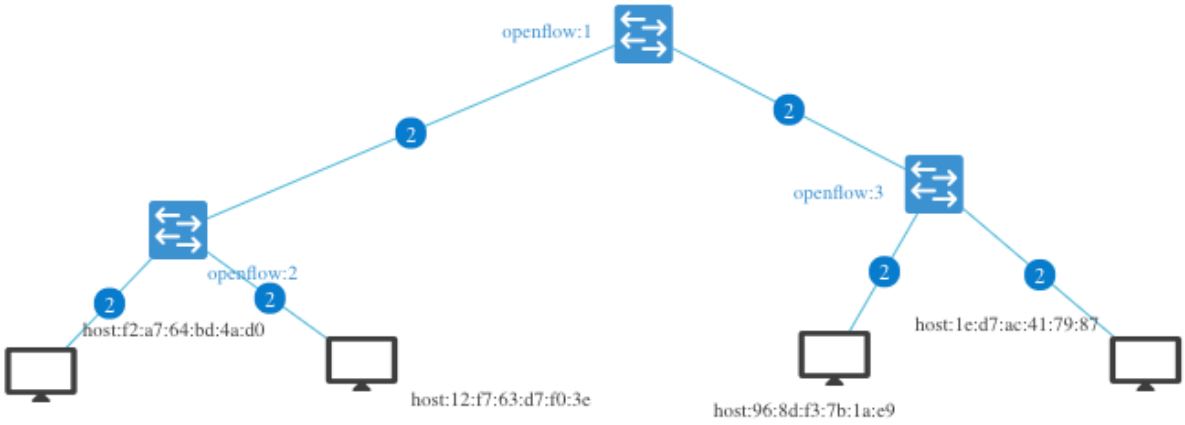
Single Switch Topology - sudo mn --controller=remote,ip=10.224.76.18 --topo=single,5
This topology consists of a single switch.



Linear Switch Topology - sudo mn --controller=remote,ip=10.224.76.18 --topo=linear,3
the switches in this are connected in linear fashion



Tree Topology - sudo mn --controller=remote,ip=10.224.76.18 --topo=tree,depth=2,fanout=2
Tree like topology can be generated by this.



Reversed Topology -
The topology remains similar to single switch topology just the connections are reversed as shown below.

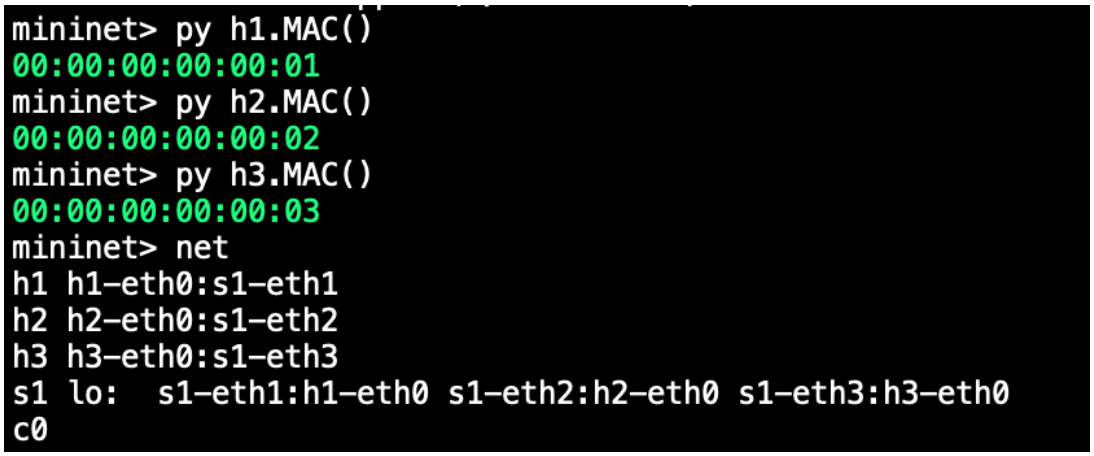
```
mininet> net
h1 h1-eth0:s1-eth2
h2 h2-eth0:s1-eth1
s1 lo:  s1-eth1:h2-eth0 s1-eth2:h1-eth0
c0
mininet>
```

Objective 3 – Implement a Programmable Layer 2 Switch and Remote Controller in Mininet

1. Open two terminal windows to the Mininet VM. (One will be used for Mininet and the other will be used for the POX controller)

2. On the first terminal, run the command:

   a. # sudo mn --topo=single,3 --mac --controller=remote --switch=ovsk



   b. Breakdown and explain what this command does in detail? **[5 points]**

   –topo=single,3 → This means to create a topology with a single switch, and 3 hosts connected to it.

   –mac → Assign mac address to host which is easier to read and simple to write.

   –controller=remote → The controller is located remotely and not located in the mininet's isolated namespace. It can be on local VM or remote VM.

   –switch=ovsk → use openflow vSwitch pre-installed with MiniNet

3. Try to ping the created nodes
    a. mininet> h1 ping h2

```
mininet> h1 ping 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
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable
From 10.0.0.1 icmp_seq=5 Destination Host Unreachable
From 10.0.0.1 icmp_seq=6 Destination Host Unreachable
From 10.0.0.1 icmp_seq=7 Destination Host Unreachable
From 10.0.0.1 icmp_seq=8 Destination Host Unreachable
From 10.0.0.1 icmp_seq=9 Destination Host Unreachable
From 10.0.0.1 icmp_seq=10 Destination Host Unreachable
```

4. Will it ping? Why or why not? **[5 points]**

```
mininet@csci5280-vm2-kima4508:~$ sudo mn --topo=single,3 --mac --controller=remote --switch=ovsk
sudo: unable to resolve host csci5280-vm2-kima4508
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** 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> pingall
*** Ping: testing ping reachability
h1 ->
```

Ping doesn't work because there is no controller attached to install the required flows.
C0 is a default controller but once we specify the option –controller=remote, Mininet tries to connect to the controller outside the Mininets default namespace.
Since we have not specified the remote controller ip, it tries to connect to OpenDaylight controller on 6653. Since there is no response, it tries to connect to the POX controller on 6633.

Since there is no controller, the switch operation cannot be performed, hence ping cannot work.
The log suggest that it has set the controller to openDayLight anyway but is not really connected.

5. On the second terminal, run the POX controller now. Enter the following command:

```
$ sudo ~/pox/pox.py forwarding.l2_learning \
  openflow.spanning_tree --hold-down \
  log.level --DEBUG samples.pretty_log \
```

openflow.discovery host_tracker \
info.packet_dump

6. Explain each component in the above POX command. **[10 points]**

./pox.py → Starts POX controller on the VM

forwarding.l2_learning → Implement the behavior of switching such as learning source MAC, in the POX Controller

openflow.spanning_tree --hold-down → This discovers the view of the network and constructs a spanning tree and sets no_flood bit on the. The hold-down flag suggests that the No_flood bit should not be unset until the complete discovery cycle has been completed.

log.level –DEBUG → Enable logging

samples.pretty_log → Format the logs to be better viewed in the console

openflow.discovery → discovers connectivity between Openflow switches via LLDP

host_tracker → keeps track of hosts in the network, where they are present, their MAC and IP address.

info.packet_dump → this component dumps packet_in info.

7. Now on the first terminal, try to ping the nodes again
   a. mininet> h1 ping h2

```
mininet@csci5280-vm2-kima4508:~$ sudo mn --topo=single,3 --mac --controller=remote --switch=ovsk
sudo: unable to resolve host csci5280-vm2-kima4508
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to 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> h1 ping 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=51.5 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.161 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.044 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 0.044/17.255/51.560/24.257 ms
mininet>
```

   b. mininet> pingall

```
mininet@csci5280-vm2-kima4508:~$ sudo mn --topo=single,3 --mac --controller=remote --switch=o
vsk
sudo: unable to resolve host csci5280-vm2-kima4508
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to 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> ping all
*** Unknown command: ping all
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```

8. Were the nodes able to talk to each other? Why? Why not? Provide a screenshot **[10 points]**

Short Answer ->

The nodes are able to talk to each other, when the switch receives a frame from the host, it doesn't have the capability to make the decision. Thus, the switch sends this to the controller for decision making, wherein the controller inturn makes the decision, and installs appropriate flows on the switch.

Long Answer ->

Spanning tree updated is a log indicating the controller has enabled no_flood bit on certain ports to build a loop free topology.

```
mininet@csci5280-vm2-kima4508:~$ sudo ~/pox/pox.py forwarding.l2_learning   openflow.spanning_t
.level --DEBUG samples.pretty_log   openflow.discovery host_tracker   info.packet_dump
sudo: unable to resolve host csci5280-vm2-kima4508
POX 0.3.0 (dart) / Copyright 2011-2014 James McCauley, et al.
[openflow.spanning_tree ] Spanning tree component ready
[host_tracker           ] host_tracker ready
[info.packet_dump       ] Packet dumper running
[core                   ] POX 0.3.0 (dart) going up...
[core                   ] Running on CPython (2.7.6/Nov 12 2018 20:00:40)
[core                   ] Platform is Linux-4.2.0-42-generic-i686-with-Ubuntu-14.04-trusty
[core                   ] POX 0.3.0 (dart) is up.
[openflow.of_01         ] Listening on 0.0.0.0:6633
[openflow.of_01         ] [00-00-00-00-00-01 2] connected
[openflow.discovery     ] Installing flow for 00-00-00-00-00-01
[forwarding.l2_learning ] Connection [00-00-00-00-00-01 2]
[openflow.spanning_tree ] Spanning tree updated
```

Below IP to MAC entries were discovered by host_tracker module of the controller

```
[host_tracker           ] Learned 1 1 00:00:00:00:00:01
[host_tracker           ] Learned 1 1 00:00:00:00:00:01 got IP 10.0.0.1
[dump:00-00-00-00-00-01 ] [ethernet][arp]
[host_tracker           ] Learned 1 2 00:00:00:00:00:02
[host_tracker           ] Learned 1 2 00:00:00:00:00:02 got IP 10.0.0.2
[dump:00-00-00-00-00-01 ] [ethernet][arp]
```

Installing flow for Layer2 reachability between H1 and H2 (Mapping MAC to Port Number) by forwarding.l2_learning

```
[forwarding.l2_learning ] installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
[dump:00-00-00-00-00-01 ] [ethernet][ipv4][icmp][echo][56 bytes]
[forwarding.l2_learning ] installing flow for 00:00:00:00:00:01.1 -> 00:00:00:00:00:02.2
[dump:00-00-00-00-00-01 ] [ethernet][ipv4][icmp][echo][56 bytes]
[forwarding.l2_learning ] installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
```

Installing ICMP flow between H1 to H2

```
[dump:00-00-00-00-00-01 ] [ethernet][ipv4][icmp][echo][56 bytes]
[forwarding.l2_learning ] installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:03.3
[dump:00-00-00-00-00-01 ] [ethernet][ipv4][icmp][echo][56 bytes]
[forwarding.l2_learning ] installing flow for 00:00:00:00:00:03.3 -> 00:00:00:00:00:02.2
[dump:00-00-00-00-00-01 ] [ethernet][ipv4][icmp][echo][56 bytes]
[forwarding.l2_learning ] installing flow for 00:00:00:00:00:03.3 -> 00:00:00:00:00:01.1
[dump:00-00-00-00-00-01 ] [ethernet][ipv4][icmp][echo][56 bytes]
[forwarding.l2_learning ] installing flow for 00:00:00:00:00:01.1 -> 00:00:00:00:00:03.3
```

Check the debugs on the second terminal (with the POX controller). You should see the ARP entries that the switch has learned. Provide a screenshot of the debug information **[10 points]**

```
mininet@csci5280-vm2-kima4508:~$ sudo ~/pox/pox.py forwarding.l2_learning   openflow.spanning_tree -
nfo.packet_dump
sudo: unable to resolve host csci5280-vm2-kima4508
POX 0.3.0 (dart) / Copyright 2011-2014 James McCauley, et al.
[openflow.spanning_tree ] Spanning tree component ready
[host_tracker           ] host_tracker ready
[info.packet_dump       ] Packet dumper running
[core                   ] POX 0.3.0 (dart) going up...
[core                   ] Running on CPython (2.7.6/Nov 12 2018 20:00:40)
[core                   ] Platform is Linux-4.2.0-42-generic-i686-with-Ubuntu-14.04-trusty
[core                   ] POX 0.3.0 (dart) is up.
[openflow.of_01         ] Listening on 0.0.0.0:6633
[openflow.of_01         ] [00-00-00-00-00-01 2] connected
[openflow.discovery     ] Installing flow for 00-00-00-00-00-01
[forwarding.l2_learning ] Connection [00-00-00-00-00-01 2]
[host_tracker           ] Learned 1 1 00:00:00:00:00:01
[host_tracker           ] Learned 1 1 00:00:00:00:00:01 got IP 10.0.0.1
[dump:00-00-00-00-00-01 ] [ethernet][arp]
[host_tracker           ] Learned 1 2 00:00:00:00:00:02
[host_tracker           ] Learned 1 2 00:00:00:00:00:02 got IP 10.0.0.2
[dump:00-00-00-00-00-01 ] [ethernet][arp]
[forwarding.l2_learning ] installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
[dump:00-00-00-00-00-01 ] [ethernet][ipv4][icmp][echo][56 bytes]
[forwarding.l2_learning ] installing flow for 00:00:00:00:00:01.1 -> 00:00:00:00:00:02.2
[dump:00-00-00-00-00-01 ] [ethernet][ipv4][icmp][echo][56 bytes]
[forwarding.l2_learning ] installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
[dump:00-00-00-00-00-01 ] [ethernet][arp]
[host_tracker           ] Learned 1 3 00:00:00:00:00:03
[host_tracker           ] Learned 1 3 00:00:00:00:00:03 got IP 10.0.0.3
[dump:00-00-00-00-00-01 ] [ethernet][arp]
[forwarding.l2_learning ] installing flow for 00:00:00:00:00:03.3 -> 00:00:00:00:00:01.1
[dump:00-00-00-00-00-01 ] [ethernet][ipv4][icmp][echo][56 bytes]
[forwarding.l2_learning ] installing flow for 00:00:00:00:00:01.1 -> 00:00:00:00:00:03.3
[dump:00-00-00-00-00-01 ] [ethernet][ipv4][icmp][echo][56 bytes]
[forwarding.l2_learning ] installing flow for 00:00:00:00:00:03.3 -> 00:00:00:00:00:01.1
[dump:00-00-00-00-00-01 ] [ethernet][ipv4][icmp][echo][56 bytes]
[forwarding.l2_learning ] installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
[dump:00-00-00-00-00-01 ] [ethernet][ipv4][icmp][echo][56 bytes]
[forwarding.l2_learning ] installing flow for 00:00:00:00:00:01.1 -> 00:00:00:00:00:02.2
[dump:00-00-00-00-00-01 ] [ethernet][arp]
[dump:00-00-00-00-00-01 ] [ethernet][arp]
[forwarding.l2_learning ] installing flow for 00:00:00:00:00:03.3 -> 00:00:00:00:00:02.2
[dump:00-00-00-00-00-01 ] [ethernet][ipv4][icmp][echo][56 bytes]
[forwarding.l2_learning ] installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:03.3
[dump:00-00-00-00-00-01 ] [ethernet][ipv4][icmp][echo][56 bytes]
[forwarding.l2_learning ] installing flow for 00:00:00:00:00:03.3 -> 00:00:00:00:00:02.2
[dump:00-00-00-00-00-01 ] [ethernet][ipv4][icmp][echo][56 bytes]
[forwarding.l2_learning ] installing flow for 00:00:00:00:00:03.3 -> 00:00:00:00:00:01.1
[dump:00-00-00-00-00-01 ] [ethernet][ipv4][icmp][echo][56 bytes]
[forwarding.l2_learning ] installing flow for 00:00:00:00:00:01.1 -> 00:00:00:00:00:03.3
[dump:00-00-00-00-00-01 ] [ethernet][ipv4][icmp][echo][56 bytes]
```

Objective 4 – Mininet Tasks and CLI Commands

Part (a): Basic Mininet commands

1. Provide a screenshot of only the hostnames in the Mininet created above (include the command used). **[5 points]**

```
mininet> nodes
available nodes are:
c0 h1 h2 h3 s1
mininet>
```

2. Provide a screenshot of all the hosts devices in the Mininet, which includes their hostnames and IP addresses (include the command used). **[5 points]**

```
mininet> py "h1 ->" + h1.IP()
h1 ->10.0.0.1
mininet> py "h2 ->" + h2.IP()
h2 ->10.0.0.2
mininet> py "h3 ->" + h3.IP()
h3 ->10.0.0.3
```

3. How would you see the route-table for the switch in the network? Provide a screenshot (include the command). **[5 points]**

```
mininet> s1 route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         10.224.76.1     0.0.0.0         UG    0      0        0 eth0
10.224.76.0     *               255.255.252.0   U     0      0        0 eth0
```

Part (b): Additional Mininet features

1. Initialize the default mininet topology. What is the average RTT when h1 pings h2? [**2 points**]

   Average RTT is => 0.911 ms

```
mininet> h1 ping 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=2.50 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.192 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.040 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.040/0.911/2.502/1.126 ms
mininet>
```

2. Now initiate a mininet topology using-

   $ sudo mn --link tc,bw=10,delay=10ms

```
mininet> h1 ping 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=81.9 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=40.2 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=40.1 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=40.1 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=40.1 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 40.103/48.496/81.923/16.715 ms
mininet>
```

   Explain this command. [**5 points**]

   If we observe closely the RTT is greater which is around **48.496 ms**

   This command shapes the traffic on all the link, with a bandwidth of 10 Mbps and propagation delay of 10ms
   tc -> indicates traffic control on all links
   bw → indicates bandwidth property on the link of 10Mbps
   delay → indicates propagation delay of 10ms

3. What is the average RTT when h1 pings h2 now? Explain the reason behind this. [**5 points**]

```
mininet> h1 ping 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=81.9 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=40.2 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=40.1 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=40.1 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=40.1 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 40.103/48.496/81.923/16.715 ms
mininet>
```

The average round trip time is because of the low bandwidth of the link and propagation delay, together which has increased the time required for Echo request to go from H1-> S1-> H2 and for Echo Reply H2 -> S1 -> H1

There are total 4 hops for **Round Trip Time** from H1 to H2, thus **10ms * 4 = ~40ms** of delay.

4. Mininet allows us to use either the kernel-space switch or the user-space switch. Explain the difference between the two. [**5 points**]

   Mininet provides flexibility to either user kernel-space switch or user-space switch:

   Kernel-space switch →

   Kernel-space switches are implemented as a part of the operating system's kernel and offer high performance and efficiency. More closer to the hardware and emulates a more realistic network. It offers features and compatibility with Openflow.

   User-space switch →

   User space switch is a software running in user-space. They are not more realistic since a user-defined switch has to make kernel calls to perform operation and this makes it slower than kernel-switches. However, they are highly customizable and providings extreme debugging capabilities.

5. Execute these commands one after the other and provide the snapshots of their results-

   sudo mn --switch ovsk --test iperf
   sudo mn --switch user --test iperf

   Also, explain the results. [**10 points**]

sudo mn --switch ovsk --test iperf

The bandwidth in the kernel-space switch is ~45Gb/sec. Mininet creates the topology and then performs iPerf analysis between each unique host pair.

–switch ovsk specifies OpenVswitch running in kernel space

–test iperf this option creates a iperf network analysis by measuring throughput from TCP and UDP streams

```
mininet@csci5280-vm2-kima4508:~$ sudo mn --switch ovsk --test iperf
sudo: unable to resolve host csci5280-vm2-kima4508
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['45.5 Gbits/sec', '45.5 Gbits/sec']
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 10.269 seconds
```

sudo mn --switch user --test iperf

The bandwidth in the user-space switch is ~570Mb/sec. This is drastically low compared to kernel-space switches. Mininet creates the topology and then performs iPerf analysis between each unique host pair.

```
mininet@csci5280-vm2-kima4508:~$ sudo mn --switch user --test iperf
sudo: unable to resolve host csci5280-vm2-kima4508
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1
*** Waiting for switches to connect
s1
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['570 Mbits/sec', '571 Mbits/sec']
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1 ..
*** Stopping 2 hosts
h1 h2
*** Done
completed in 5.688 seconds
mininet@csci5280-vm2-kima4508:~$
```

Objective 5 – Update Mininet to Latest Version

1. List the command you would use to check the current version of Mininet in the VM **[5 points]**

   sudo mn –version

```
mininet@csci5280-vm2-kima4508:~$ sudo mn --version
sudo: unable to resolve host csci5280-vm2-kima4508
2.2.2
mininet@csci5280-vm2-kima4508:~$
```

2. PuTTY/SSH in to the Mininet VM and execute the following commands:

   $ cd mininet
   $ git remote -v
   $ git remote set-url origin https://github.com/mininet/mininet
   $ git fetch
   $ git checkout master
   $ git pull
   $ sudo make install

3. What is the git command used for? **[5 points]**

   cd mininet → Move to mininet folder

   git remote -v → check the remote repository for fetch and pull commands

   git remote set-url origin https://github.com/mininet/mininet → set the default remote repository of the local Mininet repository

   git fetch → fetches latest changes from git repository but doesn't apply them to the local branch. It updates. It updates local remote repository but doesn't affect the current branch

   git checkout master → change current branch to master branch

   git pull → pulls master branch latest changes and merges it into local master branch, basically updating the Mininet version

   sudo make install → build and install latest version of Mininet

4. Provide a screenshot that shows that the installation script is successful as well as the current version of Mininet **[5 Points]**

```
mininet@csci5280-vm2-kima4508:~$ cd mininet
mininet@csci5280-vm2-kima4508:~/mininet$ git remote -v
origin  https://github.com/mininet/mininet (fetch)
origin  https://github.com/mininet/mininet (push)
mininet@csci5280-vm2-kima4508:~/mininet$ git remote set-url origin https://gi
thub.com/mininet/mininet
mininet@csci5280-vm2-kima4508:~/mininet$ git fetch
mininet@csci5280-vm2-kima4508:~/mininet$ git checkout master
Already on 'master'
Your branch is up-to-date with 'origin/master'.
mininet@csci5280-vm2-kima4508:~/mininet$ git pull
Already up-to-date.
mininet@csci5280-vm2-kima4508:~/mininet$ sudo make install
sudo: unable to resolve host csci5280-vm2-kima4508
install -D mnexec /usr/bin/mnexec
install -D -t /usr/share/man/man1 mn.1 mnexec.1
python -m pip uninstall -y mininet || true
Can't uninstall 'mininet'. No files were found to uninstall.
python -m pip install .
Unpacking /home/mininet/mininet
  Running setup.py (path:/tmp/pip-Ji4B7I-build/setup.py) egg_info for package
 from file:///home/mininet/mininet

Requirement already satisfied (use --upgrade to upgrade): setuptools in /usr/
lib/python2.7/dist-packages (from mininet==2.3.1b4)
Installing collected packages: mininet
  Found existing installation: mininet 2.2.2
    Can't uninstall 'mininet'. No files were found to uninstall.
  Running setup.py install for mininet

    changing mode of /usr/local/bin/mn to 755
Successfully installed mininet
Cleaning up...
mininet@csci5280-vm2-kima4508:~/mininet$
```

```
mininet@csci5280-vm2-kima4508:~/mininet$ sudo  mn --version
sudo: unable to resolve host csci5280-vm2-kima4508
2.3.1b4
mininet@csci5280-vm2-kima4508:~/mininet$
```

Objective 6 – X11 Forwarding: Setup the SDN (controller/switch)

Environment

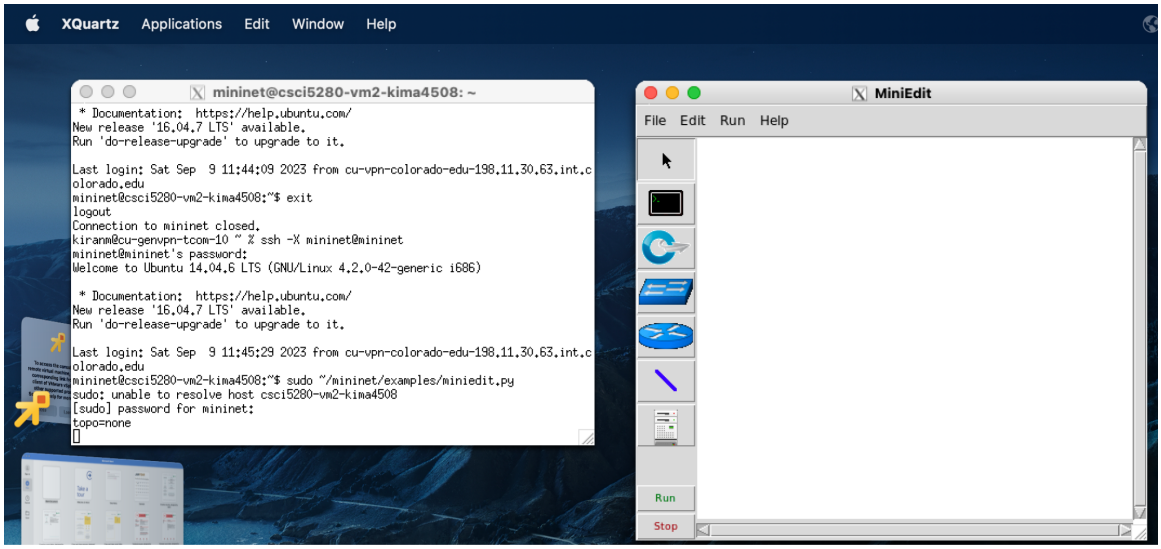
- Download Xming (Windows) or Xquartz (MAC).
- Run Xming software on local machine.
- Start a new PuTTy/SSH session to the Mininet VM.
- Before you connect, in PuTTy go to Connection > SSH > X11
- Check the “Enable X11 forwarding” box.

  [Note: If using MAC/Linux you can simply SSH with the “-X”.]
- Initiate the PuTTy connection.
- Verify that your local machine can talk to the Mininet VM.

Objective 7 – Use MiniEdit to Build a Simulation Topology

1. PuTTy/SSH in to Mininet VM with X11 forwarding enabled.
2. Execute the following command:

   sudo ~/mininet/examples/miniedit.py
3. Provide a screen shot of MiniEdit running in X11 **[5 points]**



4. Use the GUI to build a topology as is shown in the figure 1. below. All links have bandwidth of 20Mbps.

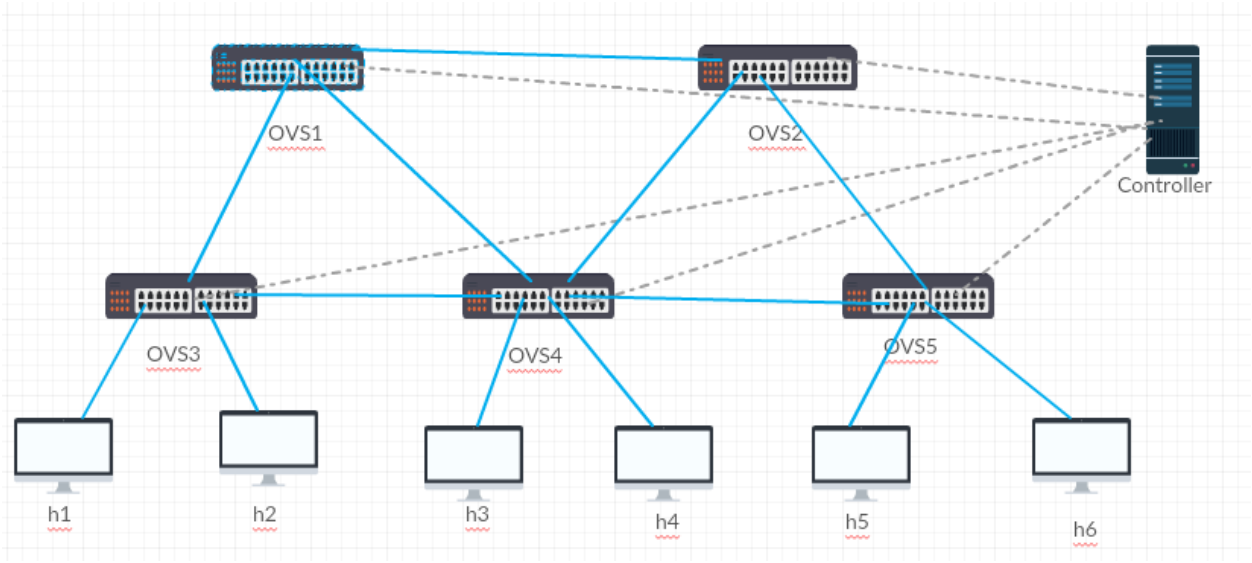


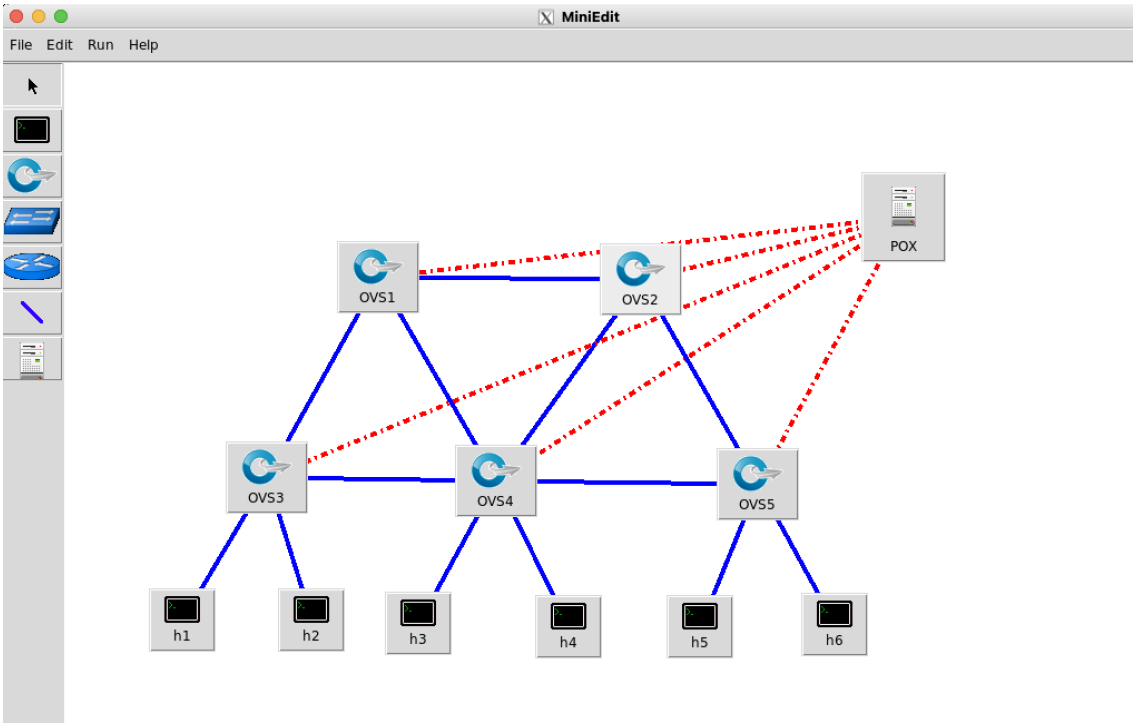
Figure 1. Mininet topology for Lab1

5. Start POX controller in a separate terminal of Mininet VM and clear any of the previously run Mininet topology. Provide screenshot of the controller running **[5 points]**

```
mininet@csci5280-vm2-kima4508:~/pox$ ./pox.py forwarding.l2_learning openf
low.spanning_tree --hold-down log.level --DEBUG samples.pretty_log openflo
w.discovery host_tracker info.packet_dump
POX 0.3.0 (dart) / Copyright 2011-2014 James McCauley, et al.
[openflow.spanning_tree ] Spanning tree component ready
[host_tracker           ] host_tracker ready
[info.packet_dump       ] Packet dumper running
[core                   ] POX 0.3.0 (dart) going up...
[core                   ] Running on CPython (2.7.6/Nov 12 2018 20:00:40)
[core                   ] Platform is Linux-4.2.0-42-generic-i686-with-Ubu
ntu-14.04-trusty
[core                   ] POX 0.3.0 (dart) is up.
[openflow.of_01         ] Listening on 0.0.0.0:6633
```

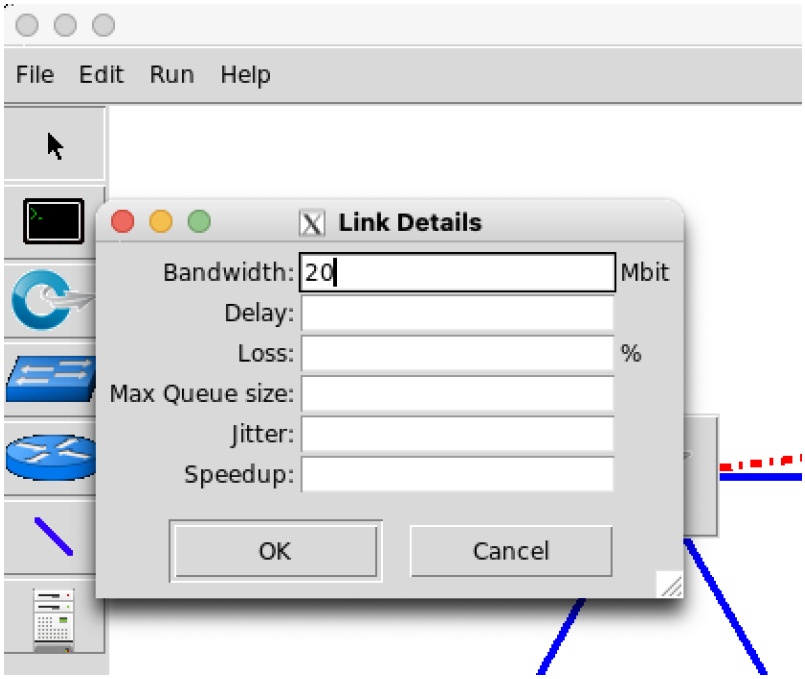
6. Set all the links to 20Mbps and point the controller to the correct IP and port number

a. Provide a screenshot of your topology inside MiniEdit **[5 points]**

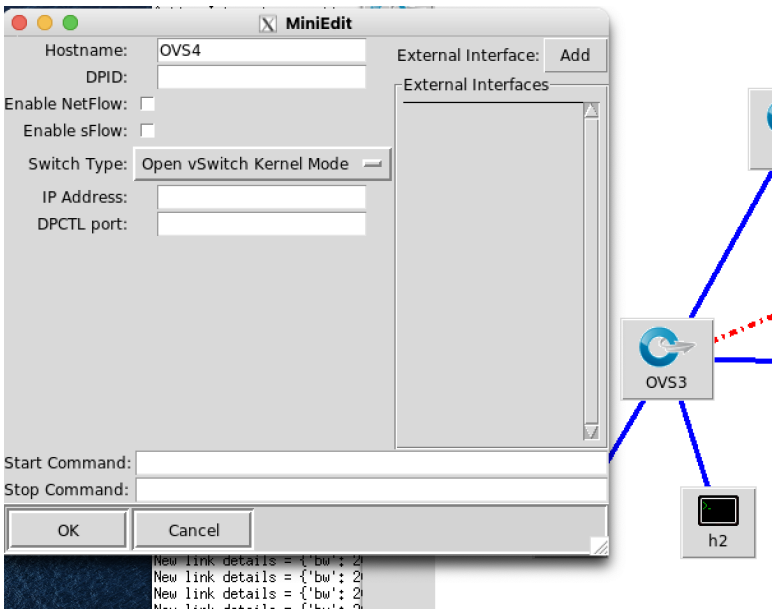


7. Provide screenshots of the different property windows of Controller, switch, host, and links **[20 points]**

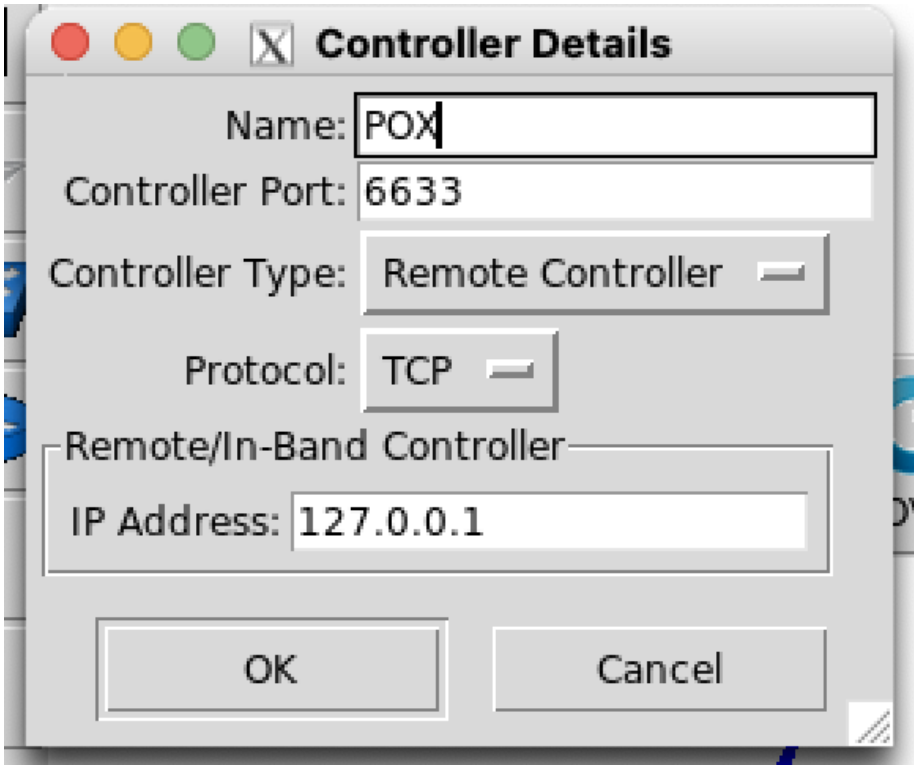
Link Config for 20Mbit

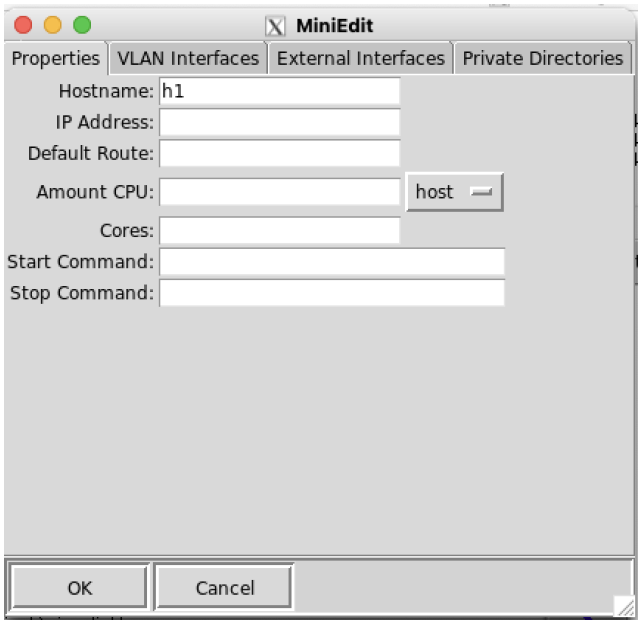


Switch config -> Open Virtual switch (Kernel mode)

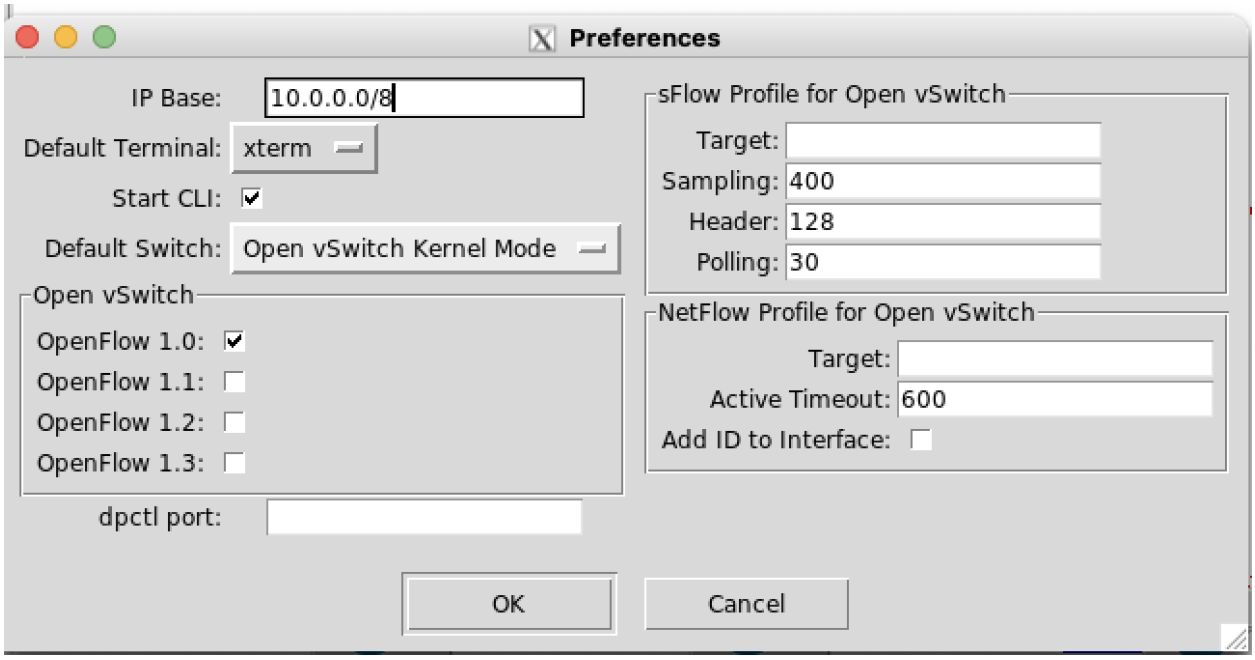


POX remote controller located on VM2. If POX is hosted on VM3 the IP should be changed





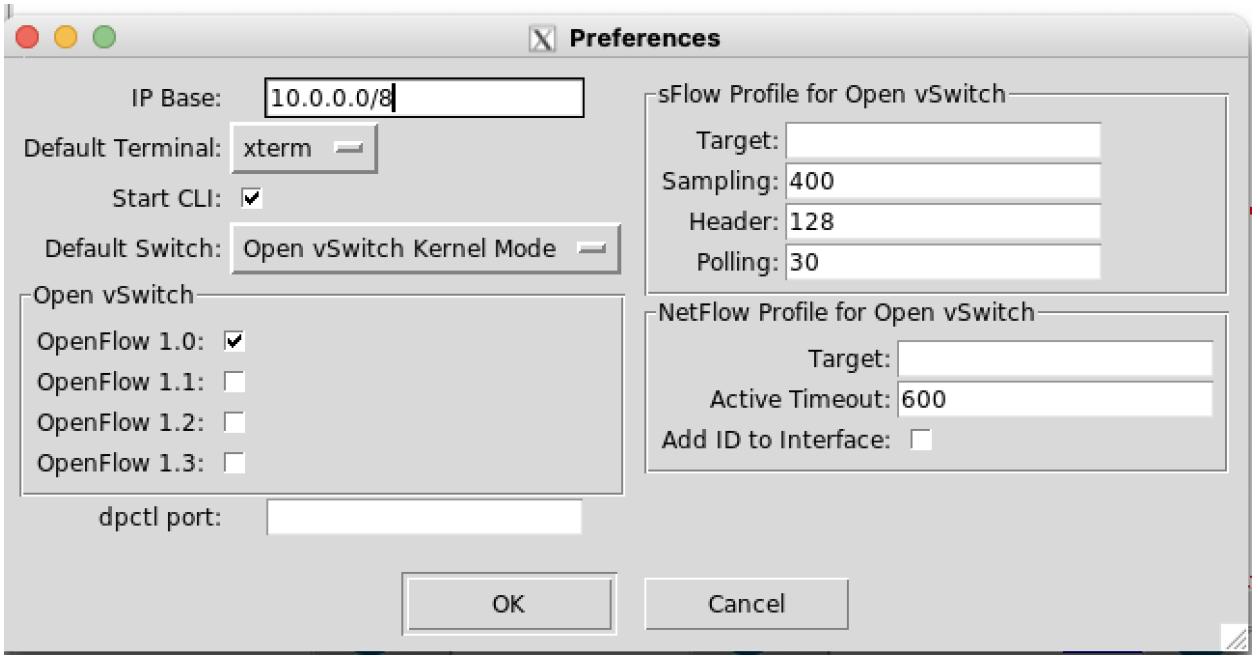
Host Ip assigned from Preferences IP Base



Logs for the configs changed in the console

```
Active Internet connections (servers and established)
tcp        0      0 0.0.0.0:6633            0.0.0.0:*               LISTEN
23950/python2.7
tcp        1      0 127.0.0.1:6633          127.0.0.1:60010         CLOSE_WAIT
-
tcp        0      0 127.0.0.1:60010         127.0.0.1:6633          FIN_WAIT2
-
New controller details for POX = {'remotePort': 6633, 'controllerProtocol': 'tcp
', 'hostname': 'POX', 'remoteIP': '10.224.76.18', 'controllerType': 'remote'}
Getting Hosts and Switches.
Getting controller selection:remote
Getting Links.
*** Configuring hosts
h5 h2 h6 h3 h4 h1
**** Starting 1 controllers
POX
**** Starting 5 switches
s4 s3 s1 s2 s5
No NetFlow targets specified.
No sFlow targets specified.
*** Stopping 1 controllers
POX
*** Stopping 13 links
.............
*** Stopping 5 switches
s4 s3 s1 s2 s5
*** Stopping 6 hosts
h5 h2 h6 h3 h4 h1
*** Done
New link details = {'bw': 20}
New link details = {'bw': 20}
New link details = {'bw': 20}
New link details = {'bw': 20}
New link details = {'bw': 20}
New link details = {'bw': 20}
New link details = {'bw': 20}
New link details = {'bw': 20}
New link details = {'bw': 20}
New link details = {'bw': 20}
New link details = {'bw': 20}
New link details = {'bw': 20}
New link details = {'bw': 20}
New link details = {'bw': 20}
New link details = {'bw': 20}
New link details = {'bw': 20}
New switch details for OVS1 = {'sflow': '0', 'switchIP': '', 'hostname': 'OVS1',
 'controllers': ['POX'], 'nodeNum': 1, 'switchType': 'ovs', 'netflow': '0'}
New switch details for OVS2 = {'sflow': '0', 'switchIP': '', 'hostname': 'OVS2',
 'controllers': ['POX'], 'nodeNum': 3, 'switchType': 'ovs', 'netflow': '0'}
New switch details for OVS3 = {'sflow': '0', 'switchIP': '', 'hostname': 'OVS3',
 'controllers': ['POX'], 'nodeNum': 4, 'switchType': 'ovs', 'netflow': '0'}
New switch details for OVS4 = {'sflow': '0', 'switchIP': '', 'hostname': 'OVS4',
 'controllers': ['POX'], 'nodeNum': 2, 'switchType': 'ovs', 'netflow': '0'}
New switch details for OVS5 = {'sflow': '0', 'switchIP': '', 'hostname': 'OVS5',
 'controllers': ['POX'], 'nodeNum': 5, 'switchType': 'ovs', 'netflow': '0'}
```

8. In the Preference of MiniEdit select “Start CLI” option



9. Save the topology as .py topology file using “Export Level2 script,” attach the .py file with your report **[10 points]**

   Attached in the document

Objective 8 – Run Simulation with MiniEdit

1. In MiniEdit click “RUN”

2. Provide a screenshot of the results **[10 points]**

```
Getting Hosts and Switches.
Getting controller selection:remote
Getting Links.
(20.00Mbit) (20.00Mbit) (20.00Mbit) (20.00Mbit) (20.00Mbit) (20.00Mbit) (20.00Mbit) (20.00Mbit) (20.00Mbit) (20.00Mbit)
(20.00Mbit) (20.00Mbit) (20.00Mbit) (20.00Mbit) (20.00Mbit) (20.00Mbit) (20.00Mbit) (20.00Mbit) (20.00Mbit) (20.00Mbit)
(20.00Mbit) (20.00Mbit) *** Configuring hosts
h2 h6 h5 h3 h4 h1
**** Starting 1 controllers
POX
**** Starting 5 switches
OVS1 (20.00Mbit) (20.00Mbit) (20.00Mbit) OVS4 (20.00Mbit) (20.00Mbit) (20.00Mbit) (20.00Mbit) (20.00Mbit) OVS3 (20.00Mbit) (20.00Mbi
t) (20.00Mbit) (20.00Mbit) OVS2 (20.00Mbit) (20.00Mbit) (20.00Mbit) OVS5 (20.00Mbit) (20.00Mbit) (20.00Mbit) (20.00Mbit)
No NetFlow targets specified.
No sFlow targets specified.


 NOTE: PLEASE REMEMBER TO EXIT THE CLI BEFORE YOU PRESS THE STOP BUTTON. Not exiting will prevent MiniEdit from quitting and will prevent you fr
om starting the network again during this session.

*** Starting CLI:
mininet>
mininet>
mininet>
mininet>
```

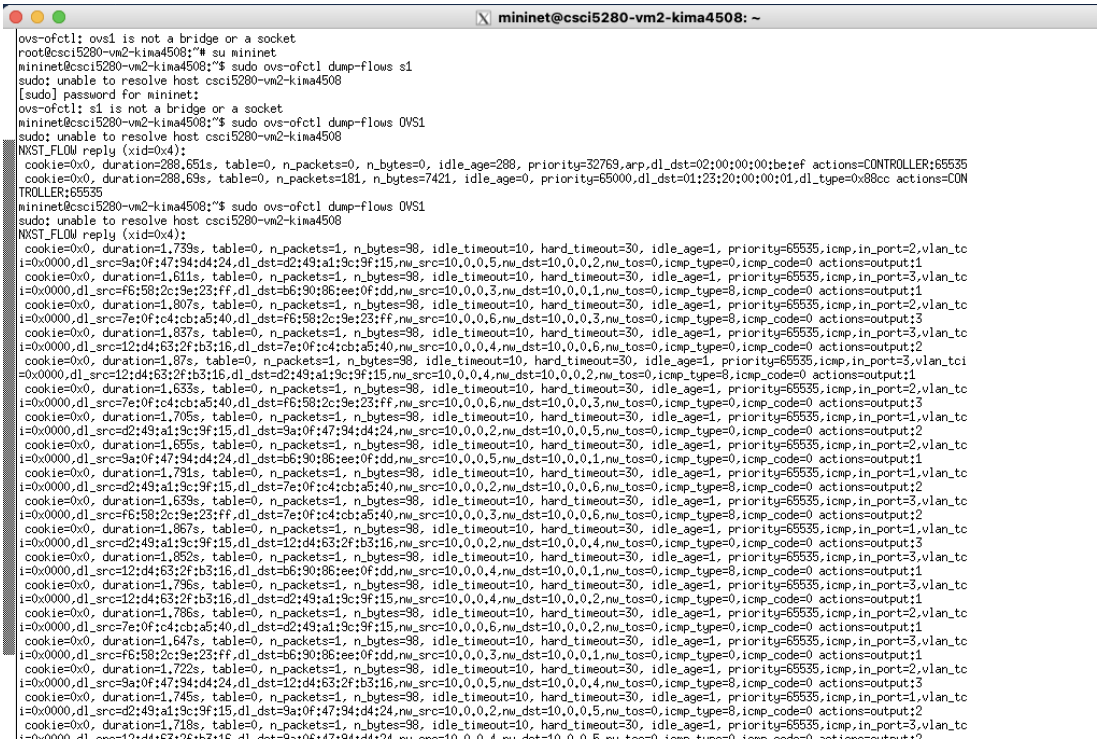
3. Run “pingall” in the CLI

```
mininet>
mininet>
mininet> pingall
*** Ping: testing ping reachability
h2 -> h6 h5 h3 h4 h1
h6 -> h2 h5 h3 h4 h1
h5 -> h2 h6 h3 h4 h1
h3 -> h2 h6 h5 h4 h1
h4 -> h2 h6 h5 h3 h1
h1 -> h2 h6 h5 h3 h4
*** Results: 0% dropped (30/30 received)
mininet>
```

4. Open a root terminal from MiniEdit or the switch CLI

5. Execute the command which shows all flows for switch 1

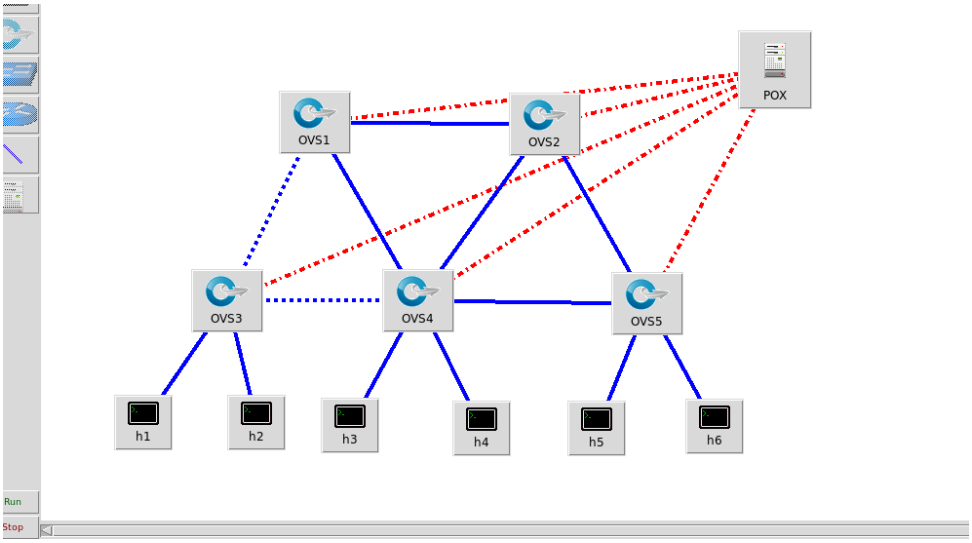
    a. Provide a screenshot of the command and output **[10 points]**



Objective 9 – Simulate a Broken Link

1. In the MiniEdit GUI, mark the link between “OVS3 and OVS1” as well as “OVS3 and OVS4” as “down” using the right click menu.

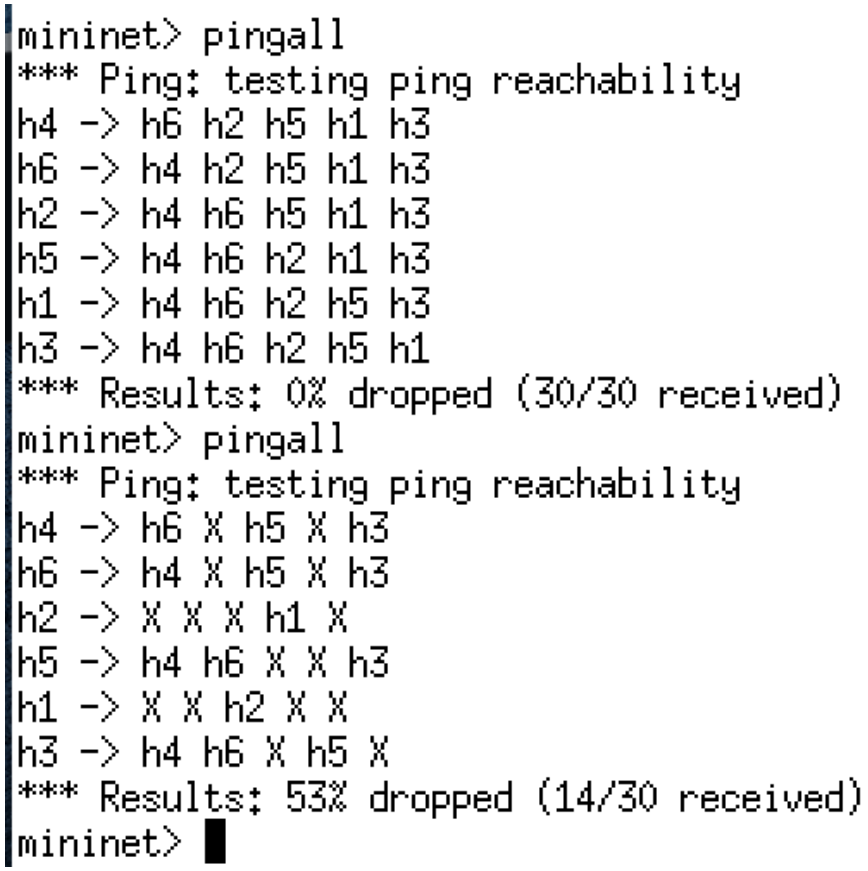
    a. Provide a screenshot of the updated topology **[10 points]**



Spanning Tree in Pox Controller updated twice

```
[openflow.discovery     ] link timeout: 00-00-00-00-00-01.1 -> 00-00-00-00-00-03.3
[openflow.discovery     ] link timeout: 00-00-00-00-00-03.3 -> 00-00-00-00-00-01.1
[openflow.discovery     ] link timeout: 00-00-00-00-00-04.6 -> 00-00-00-00-00-03.4
[openflow.discovery     ] link timeout: 00-00-00-00-00-03.4 -> 00-00-00-00-00-04.6
[openflow.spanning_tree ] Spanning tree updated
[openflow.spanning_tree ] Spanning tree updated
```

2. Run “pingall” in the Mininet CLI, provide a screenshot of the results **[5 points]**



Objective 10 – Report Questions

1. Explain five advantages of using Mininet **[5 points]**

    1. It is very fast and easy to use. All example topologies were created instantly on the fly
    2. Mininet integration with several controllers is easy and straightforward. Setting up POX and ODL as controllers was easy.
    3. It can save the cost of building an actual network to test, since the topology and real time bandwidth, delay and various granular factors can be configured, in the Mininet topology itself
    4. Mininet becomes a highly available emulator as we deploy it in VM.
    5. It is easy and possible to do customized packet forwarding in Mininet.

2. What are two limitations of Mininet **[2 points]**

    a. Network Topologies hosted in Mininet cannot exceed the CPU and bandwidth of the underlying single server
    b. Mininet cannot run switches which are not compatible with Linux Operating System

3. What exactly is POX and what do you think are its advantages? Do you think POX is actively used in the industry? Why/why not? [**5 points**]

    a. POX was originally developed as a SDN controller, but now also can be an Open-Flow Switch. It was primarily developed for education and research purpose and thus servers key advantages -
        i. easy to use
        ii. is lightweight and has low overhead
        iii. its open source allowing users to understand how controllers are built.

    b. Use case of POX depends on the purpose, but POX cannot be used in industry:
        i. It runs into scalability issues, when the topology scales
        ii. POX doesn't have critical security aspects which might be a consideration in large scale deployments.
        iii. POX is open source but it is not extremely stable and thus cannot have a wide community support since it doesn't have many features.

4. Like MiniEdit, what other tool can be used for creating customized Mininet topology scripts? **[3 points]**

    a. GNS3 can be integrated with Mininet and the topology can be further edited
    b. Python scripts can be used to build topologies via Topo.py
    c. ONOS and Faucet allow integration with Mininet to create custom topologies.

Total Score = _____/222