Mohammed VI Polytechnic University
School of Computer Science

Advanced Network Programming
Lab 2 – Practical
Load Balancer

Instructions:

For this lab you will create a simple load balancer architecture using an SDN controller and OpenFlow-enabled switches. When there are multiple paths in a network, using the load balancing concept, we can effectively utilize available paths, and ensure minimum congestion and packet loss.

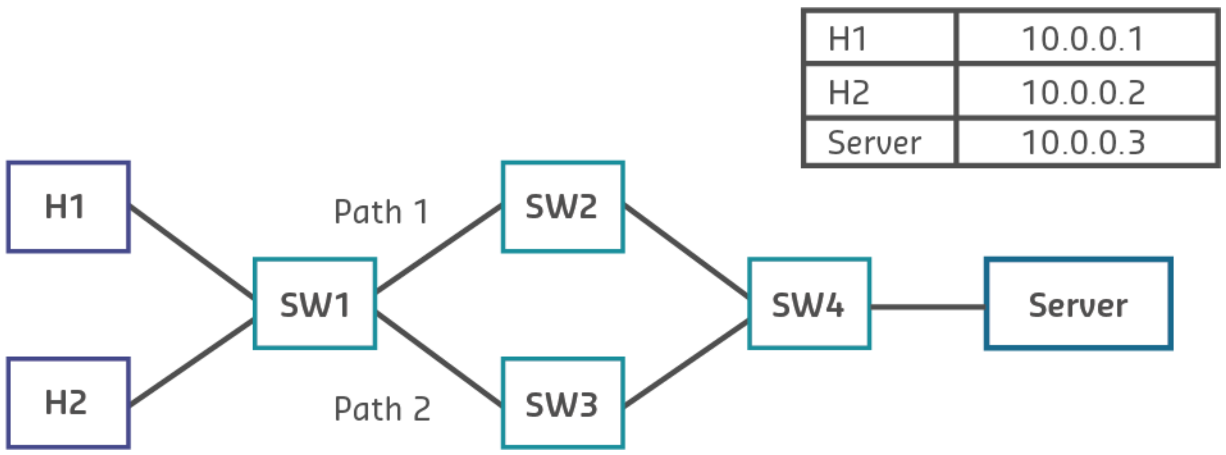
SDN Controller:
For this lab, you can use any type of SDN controller (POX, RYU, NOX etc.), to create the Load Balancer application. We have used POX controller in our examples and instructions.

More details of the POX controller can be found here: https://noxrepo.github.io/pox-doc/html/

OpenFlow:
In this lab, we will explore how OpenFlow rules can be installed in switches to control the network traffic.

More details of the OpenFlow protocol, can be found here:
https://opennetworking.org/software-defined-standards/specifications/

Topology:



H1	10.0.0.1
H2	10.0.0.2
Server	10.0.0.3

Task 1:
Create the Load Balancer Application with the Load Balancing Rules:

Following the given topology, we will assume that the load balancing happens at SW1 for forwarding traffic and at SW4 for returning traffic. The rules that you will need to implement in OpenFlow switches for this task are:

Source IP	Destination IP	Action (Path to be Used)
H1	H3	SW1 - SW2 - SW4
H2	H3	SW1 - SW3 - SW4
H3	H1	SW4 - SW2 - SW1
H3	H2	SW4 - SW3 - SW1

Basically, your load balancer application should allow H1 and H3 (i.e., the server) to communicate through Path 1 and H2 and H3 to communicate through Path 2.

Be careful! Flow tables match the rule with highest priority first, where priority is established based on the order rules are placed in the table.

In this lab, we will try the **Proactive Controller approach** – All OpenFlow rules should be installed at the beginning of network initialization. You can use any type of SDN controller (POX, RYU, NOX etc.), we have given instructions for POX controller.

Create "*LoadBalancer*" application with necessary codes to install OpenFlow rules and run with the selected controller to install all OpenFlow rules.

1. Setup the Mininet VM

This has been done in Lab 0.

2. Install POX controller in Mininet VM

The Mininet VM comes with POX installed.

3. Create LoadBalancer.py application to install OpenFlow rules inside the pox directory.

The file will be provided by the instructor.

More details of the OpenFlow rules, can be found at https://opennetworking.org/software-defined-standards/specifications/

OpenFlow rules for Switch 1:

Allow all ARP traffic

H1 traffic follows Path 1

```
if m.name == "s1-eth1":

      #flood ARP packets
      msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x806),action=of.ofp_action_output(port=of.OFPP_ALL))
      event.connection.send(msg)

      msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x800,nw_src="10.0.0.1",nw_dst="10.0.0.3"),action=of.ofp_action_output(port=3))
      event.connection.send(msg)

      msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x800,nw_src="10.0.0.2",nw_dst="10.0.0.3"),action=of.ofp_action_output(port=4))
      event.connection.send(msg)

      msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x800,nw_src="10.0.0.3",nw_dst="10.0.0.1"),action=of.ofp_action_output(port=1))
      event.connection.send(msg)

      msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x800,nw_src="10.0.0.3",nw_dst="10.0.0.2"),action=of.ofp_action_output(port=2))
      event.connection.send(msg)
```

H2 traffic follows Path 2

OpenFlow rules for Switch 2:

```
elif m.name == "s2-eth1":

      #flood ARP packets
      msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x806),action=of.ofp_action_output(port=of.OFPP_ALL))
      event.connection.send(msg)

      msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x800,nw_src="10.0.0.1",nw_dst="10.0.0.3"),action=of.ofp_action_output(port=2))
      event.connection.send(msg)

      msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x800,nw_src="10.0.0.3",nw_dst="10.0.0.1"),action=of.ofp_action_output(port=1))
      event.connection.send(msg)
```

OpenFlow rules for Switch 3:

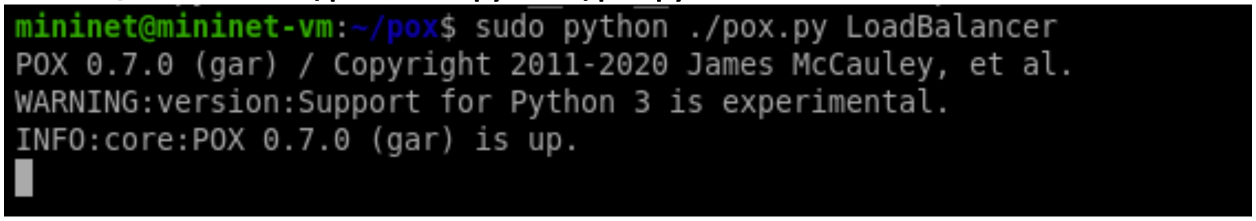
```
elif m.name == "s3-eth1":

      #flood ARP packets
      msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x806),action=of.ofp_action_output(port=of.OFPP_ALL))
      event.connection.send(msg)

      msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x800,nw_src="10.0.0.2",nw_dst="10.0.0.3"),action=of.ofp_action_output(port=2))
      event.connection.send(msg)

      msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x800,nw_src="10.0.0.3",nw_dst="10.0.0.2"),action=of.ofp_action_output(port=1))
      event.connection.send(msg)
```

OpenFlow rules for Switch 4:

```
elif m.name == "s4-eth1":

      #flood ARP packets
      msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x806),action=of.ofp_action_output(port=of.OFPP_ALL))
      event.connection.send(msg)

      msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x800,nw_src="10.0.0.1",nw_dst="10.0.0.3"),action=of.ofp_action_output(port=3))
      event.connection.send(msg)

      msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x800,nw_src="10.0.0.2",nw_dst="10.0.0.3"),action=of.ofp_action_output(port=3))
      event.connection.send(msg)

      msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x800,nw_src="10.0.0.3",nw_dst="10.0.0.1"),action=of.ofp_action_output(port=1))
      event.connection.send(msg)

      msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x800,nw_src="10.0.0.3",nw_dst="10.0.0.2"),action=of.ofp_action_output(port=2))
      event.connection.send(msg)
```

4. Run your LoadBalancer.py application with the POX controller.

mininet@mininet-vm:~/pox$ sudo python ./pox.py LoadBalancer



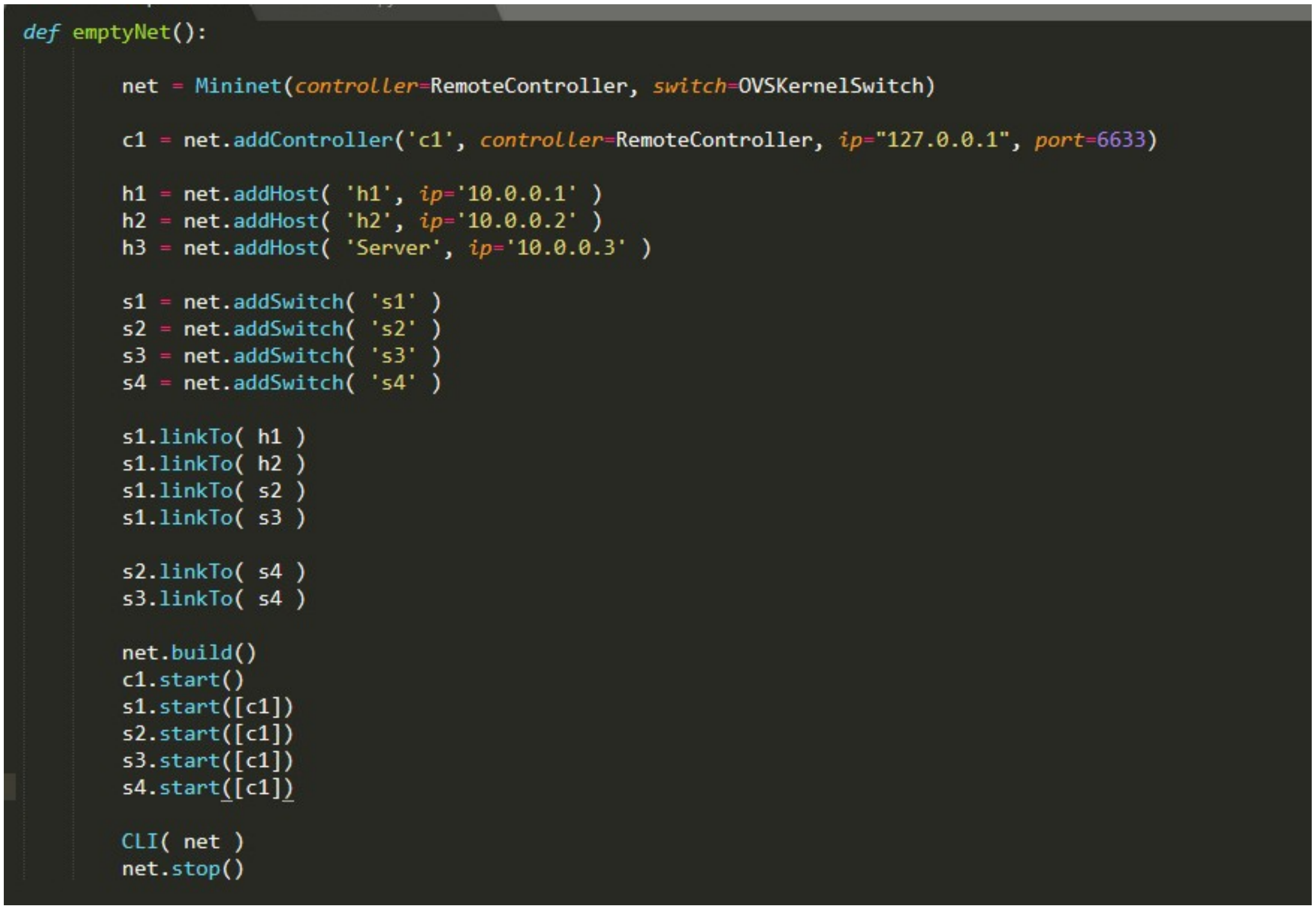
```
mininet@mininet-vm:~/pox$ sudo python ./pox.py LoadBalancer
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
WARNING:version:Support for Python 3 is experimental.
INFO:core:POX 0.7.0 (gar) is up.
```

Task 2: Create the given topology in Mininet:

1. Create a topology application (LoadBalancerTopo.py) according to the given topology diagram.

mininet@mininet-vm:~$ mkdir SDN mininet@mininet-vm:~$ cd SDN

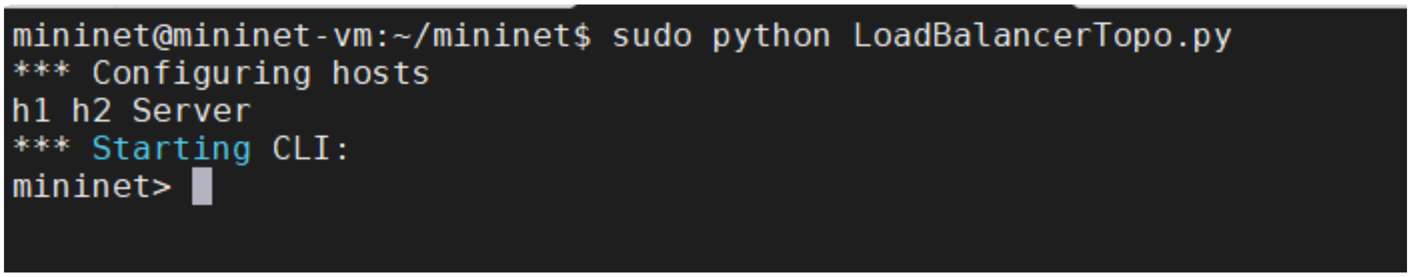
The file will be provided by the instructor.



```python
def emptyNet():

    net = Mininet(controller=RemoteController, switch=OVSKernelSwitch)

    c1 = net.addController('c1', controller=RemoteController, ip="127.0.0.1", port=6633)

    h1 = net.addHost( 'h1', ip='10.0.0.1' )
    h2 = net.addHost( 'h2', ip='10.0.0.2' )
    h3 = net.addHost( 'Server', ip='10.0.0.3' )

    s1 = net.addSwitch( 's1' )
    s2 = net.addSwitch( 's2' )
    s3 = net.addSwitch( 's3' )
    s4 = net.addSwitch( 's4' )

    s1.linkTo( h1 )
    s1.linkTo( h2 )
    s1.linkTo( s2 )
    s1.linkTo( s3 )

    s2.linkTo( s4 )
    s3.linkTo( s4 )

    net.build()
    c1.start()
    s1.start([c1])
    s2.start([c1])
    s3.start([c1])
    s4.start([c1])

    CLI( net )
    net.stop()
```

2. Run your topology by using the command below.

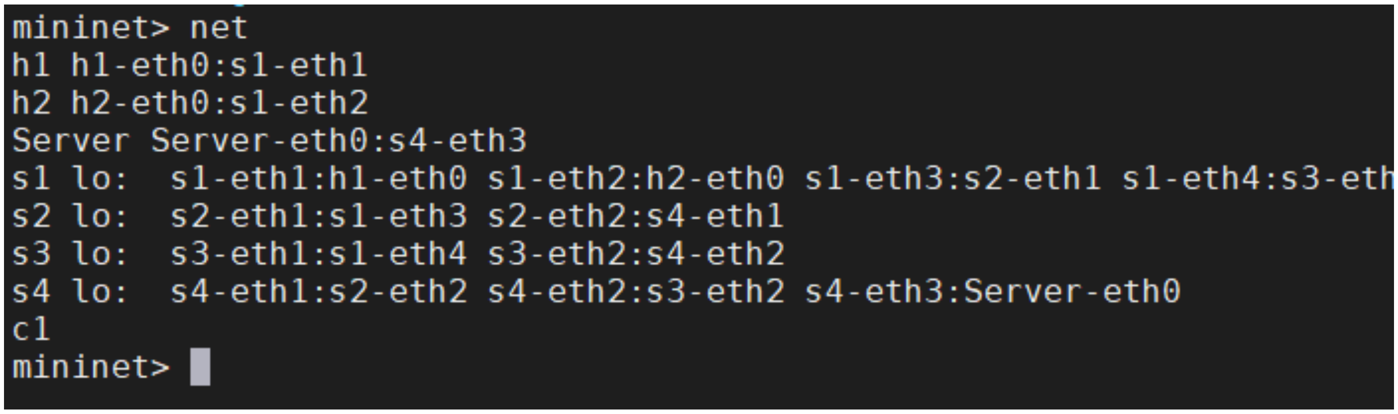
mininet@mininet-vm:~/SDN$ sudo python ./LoadBalancerTopo.py



```
mininet@mininet-vm:~/mininet$ sudo python LoadBalancerTopo.py
*** Configuring hosts
h1 h2 Server
*** Starting CLI:
mininet> 
```

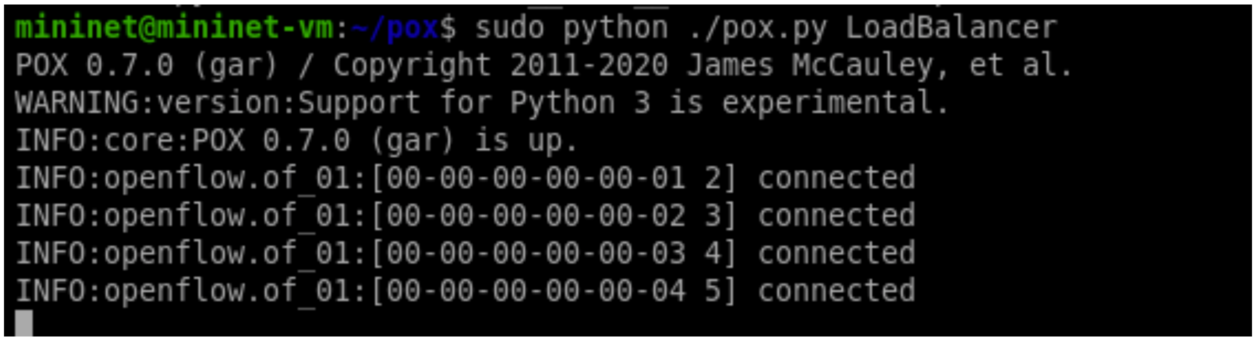
Task 3: Testing

1. Use "net" command in mininet to verify the topology.



```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
Server Server-eth0:s4-eth3
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:s2-eth1 s1-eth4:s3-eth
s2 lo:  s2-eth1:s1-eth3 s2-eth2:s4-eth1
s3 lo:  s3-eth1:s1-eth4 s3-eth2:s4-eth2
s4 lo:  s4-eth1:s2-eth2 s4-eth2:s3-eth2 s4-eth3:Server-eth0
c1
mininet> 
```

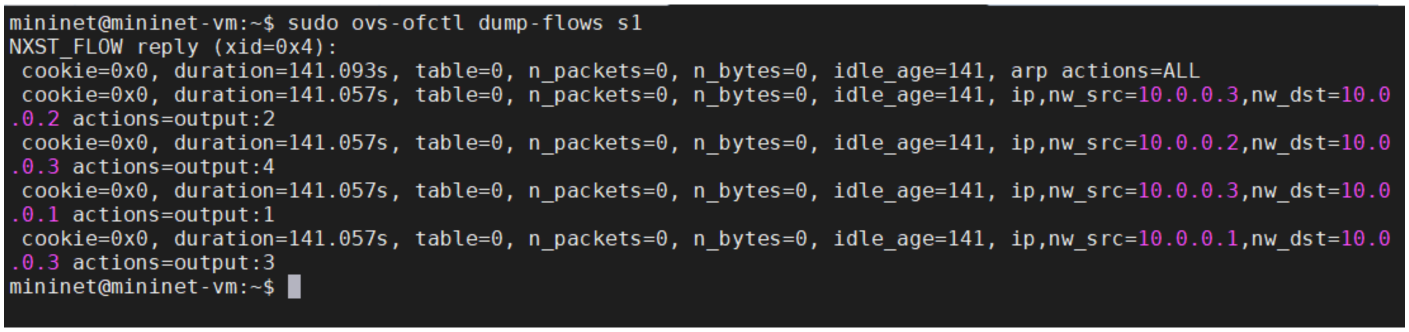
2. Check the controller prompt to verify switch connections to the controller.



```
mininet@mininet-vm:~/pox$ sudo python ./pox.py LoadBalancer
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
WARNING:version:Support for Python 3 is experimental.
INFO:core:POX 0.7.0 (gar) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:openflow.of_01:[00-00-00-00-00-02 3] connected
INFO:openflow.of_01:[00-00-00-00-00-03 4] connected
INFO:openflow.of_01:[00-00-00-00-00-04 5] connected
```

3. Use "sudo ovs-ofctl dump-flows s1" to verify the commands installed in the switch **s1** (Note: Since we are implementing a proactive controller, all rules should be installed in the switches before initiation of any communication).

More details of the OpenVSwitch based commands, can be found here:
http://www.openvswitch.org/support/dist-docs/ovs-ofctl.8.html

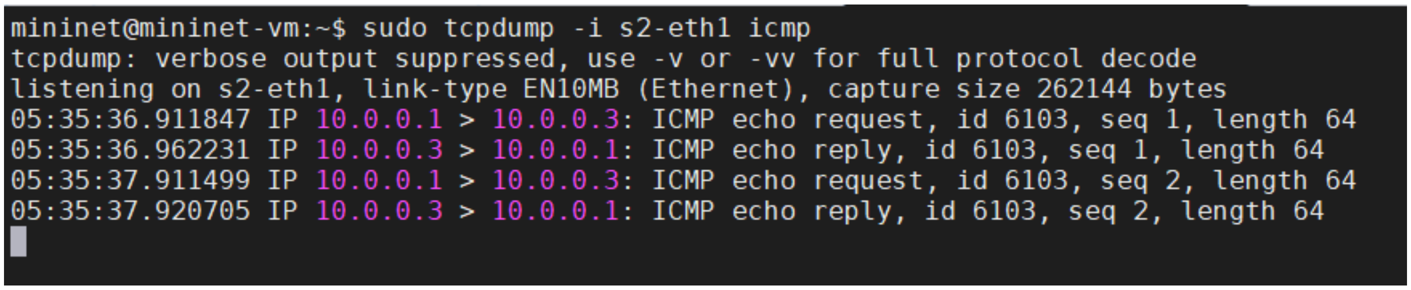


```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=141.093s, table=0, n_packets=0, n_bytes=0, idle_age=141, arp actions=ALL
 cookie=0x0, duration=141.057s, table=0, n_packets=0, n_bytes=0, idle_age=141, ip,nw_src=10.0.0.3,nw_dst=10.0
.0.2 actions=output:2
 cookie=0x0, duration=141.057s, table=0, n_packets=0, n_bytes=0, idle_age=141, ip,nw_src=10.0.0.2,nw_dst=10.0
.0.3 actions=output:4
 cookie=0x0, duration=141.057s, table=0, n_packets=0, n_bytes=0, idle_age=141, ip,nw_src=10.0.0.3,nw_dst=10.0
.0.1 actions=output:1
 cookie=0x0, duration=141.057s, table=0, n_packets=0, n_bytes=0, idle_age=141, ip,nw_src=10.0.0.1,nw_dst=10.0
.0.3 actions=output:3
mininet@mininet-vm:~$
```

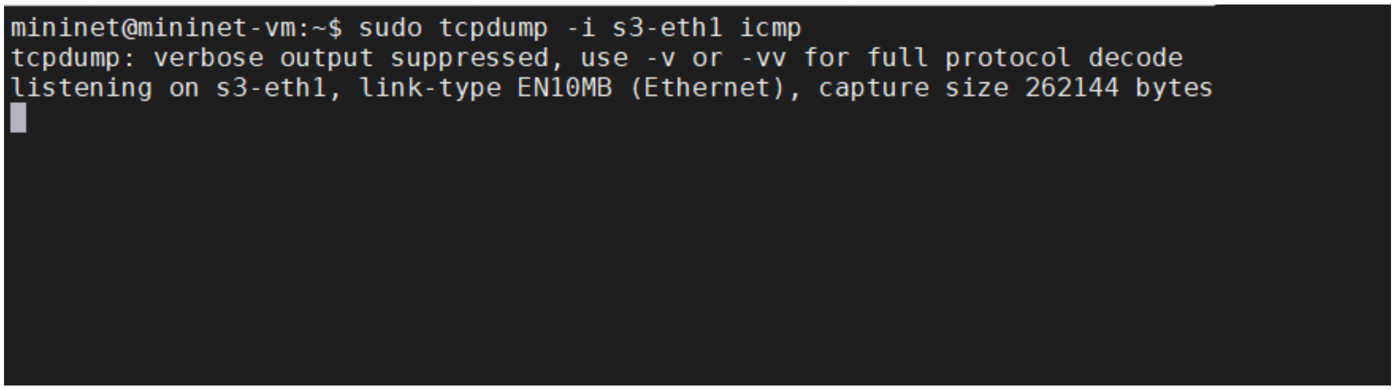
4. Try to ping from h1 to server: It should be successful, and packets should follow Path 1

```
mininet> h1 ping -c2 10.0.0.3
```

To verify h1 to server packets are following Path 1, monitor the traffic at Switch 2 and Switch 3, and you should be able to see packets going through Switch 2, which is on Path 1.



```
mininet@mininet-vm:~$ sudo tcpdump -i s2-eth1 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s2-eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
05:35:36.911847 IP 10.0.0.1 > 10.0.0.3: ICMP echo request, id 6103, seq 1, length 64
05:35:36.962231 IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 6103, seq 1, length 64
05:35:37.911499 IP 10.0.0.1 > 10.0.0.3: ICMP echo request, id 6103, seq 2, length 64
05:35:37.920705 IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 6103, seq 2, length 64
```



```
mininet@mininet-vm:~$ sudo tcpdump -i s3-eth1 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s3-eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
```