# Networked Systems Coursework 1: Mininet and SDN

Viyan Raj

## 1 Deliverable 1: Programming Mininet Topologies

The file topo1.py was modified to contain 4 hosts (in the same subnet) connected to one switch. The results of running the requested commands in the VM are below.

```
mininet> h1 ping h4
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=11.0 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=0.375 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=0.066 ms
64 bytes from 10.0.0.5: icmp_seq=4 ttl=64 time=0.081 ms
64 bytes from 10.0.0.5: icmp_seq=5 ttl=64 time=0.079 ms
64 bytes from 10.0.0.5: icmp_seq=6 ttl=64 time=0.078 ms
64 bytes from 10.0.0.5: icmp_seq=7 ttl=64 time=0.076 ms
64 bytes from 10.0.0.5: icmp_seq=8 ttl=64 time=0.087 ms
64 bytes from 10.0.0.5: icmp_seq=9 ttl=64 time=0.079 ms
64 bytes from 10.0.0.5: icmp_seq=10 ttl=64 time=0.078 ms
^C
--- 10.0.0.5 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9200ms
rtt min/avg/max/mdev = 0.066/1.200/11.004/3.269 ms
```

Figure 1: Running 'h1 ping h4'

This shows 10 ICMP packets being transmitted between h1 and h4. It also shows that no packets were lost, as well as the minimum, average, maximum and maximum deviation of RTT (return trip time - time taken for the request to go from source to destination then back to source).

```
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['16.4 Gbits/sec', '22.9 Gbits/sec']
```

Figure 2: Running 'iperf h1 h4'

This shows the bandwidth of the TCP connection between h1 and h4 (which goes through the switch s1).

```
mininet> dump
<Host h1: h1-eth0:10.0.0.2 pid=1835>
<Host h2: h2-eth0:10.0.0.3 pid=1837>
<Host h3: h3-eth0:10.0.0.4 pid=1839>
<Host h4: h4-eth0:10.0.0.5 pid=1841>
<OVSSwitch switch1: lo:127.0.0.1,switch1-eth1:None,switch1-eth2:None,switch1-eth3:None,switch1-eth4:
None pid=1846>
<Controller c0: 127.0.0.1:6653 pid=1828>
```

Figure 3: Running 'dump'

This shows each host, along with it's MAC address and pid. It also shows the OVSSwitch (switch1) with it's ports and pid. It shows the default mininet controller as well.

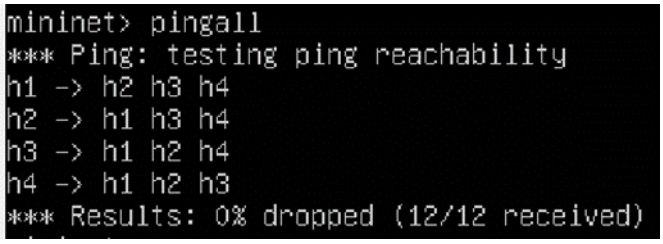


Figure 4: Running 'pingall'

This shows that all four hosts can reach each-other.

# 2 Deliverable 2: Using an OpenFlow Controller

## 2.1 Inspecting controller rules

The file topo2.py was run using the l2_learning.py controller. The results of running the requested commands in the VM are below.

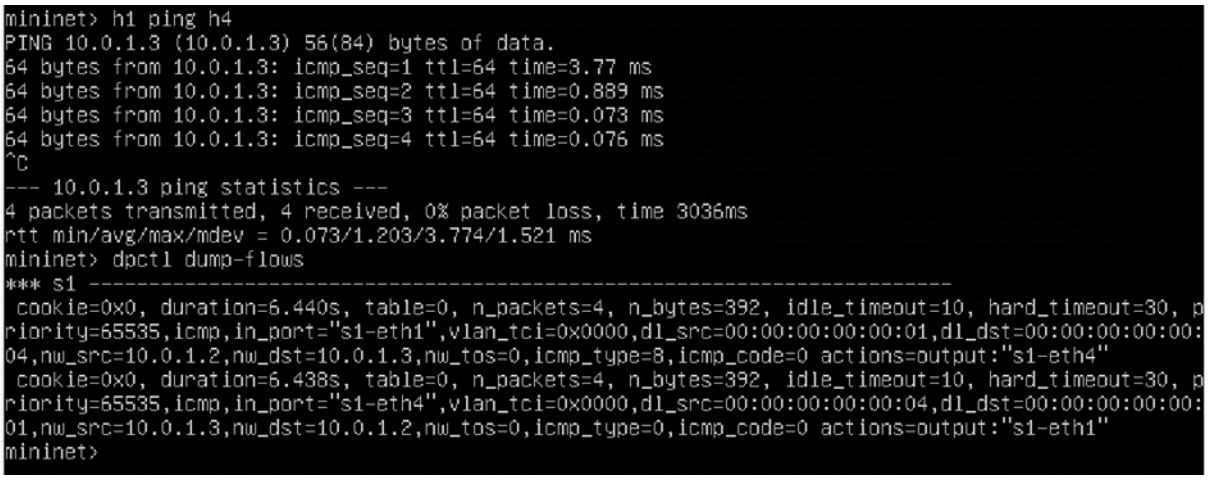


Figure 5: Running 'h1 ping h4' then 'dpctl dump-flows'

There are two rules installed by the controller.
The first matches ICMP packets coming from MAC address 00:00:00:00:00:01 and IP address 10.0.1.2 (which is host h1), with destination MAC address 00:00:00:00:00:04 and IP address 10.0.1.3 (which is host h4). These packets arrive on port "s1-eth1". The rule for these packets is to forward them to port "s1-eth4".
The second matches ICMP packets coming from MAC address 00:00:00:00:00:04 and IP address 10.0.1.3 (which is host h4), with destination MAC address 00:00:00:00:00:01 and IP address 10.0.1.2 (which is host h1). These packets arrive on port "s1-eth4". The rule for these packets is to forward them to port "s1-eth1".

## 2.2 l2_learning.py explanation

l2_learning.py is a layer 2 ethernet switch controller that learns and maintains MAC address to port mappings in the network. It uses these mappings to make forwarding decisions.

When the controller starts, it initialises an empty dictionary (self.macToPort) to store these mappings. When the controller receives a Packet-In message from the switch, it means that the switch has received a packet which it doesn't know the output port for. The controller then extracts the source and destination MAC addresses from the packet, updating its macToPort dictionary with the source

MAC address and the input port on which it received the packet. It then checks if it already knows the output port for the destination MAC address.

If the destination MAC address is in the macToPort dictionary, the controller sends a FlowMod message to the switch, instructing it to forward future packets with the destination MAC to the correct output port. If the destination MAC address is not known, the controller floods the packet to all ports except the incoming port, allowing the switch to learn the location of the destination host (once the destination host sends a reply).

When the controller determines the output port for a MAC address, it sends a FlowMod message to the switch to install a flow entry. This entry tells the switch how to forward packets with the specified source and destination MAC addresses. If the controller determines that a packet should be sent to a specific port, it sends a Packet-Out message to the switch, indicating the output port for the packet.

The flows/rules the controller installs are exact-matches on as many fields as possible. For example, there are separate flows installed for ARP queries and ICMP messages.

When 'h1 ping h4' is run, there is a chance of receiving an ICMP packet in h2, since the first time the switch receives a packet it has no mappings (hasn't learnt anything). So it will broadcast the packet on all ports (including port "s1-eth2"). This means that h2 could receive an ICMP packet.

After this first ping, h2 won't receive ICMP packets since the switch will now know which port to forward the packet on.

# 3 Deliverable 3: Implementing a Simple Firewall

The file exe1-controller.py was modified to install rules onto switch s1, in accordance with the specification. The 'ofp_flow_mod' module was used to define match-action rules which were then sent to the connection. The results of running the requested commands in the VM are below.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X h4
h2 -> X h3 X
h3 -> X h2 X
h4 -> h1 X X
*** Results: 66% dropped (4/12 received)
```

Figure 6: Running 'pingall'

This shows that h1 and h4 are able to ping each-other, and h2 and h3 are able to ping each-other. These two pairs of hosts can't ping the other pair since they are in different subnets.

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=12.670s, table=0, n_packets=8, n_bytes=784, priority=4,icmp actions=FLOOD
 cookie=0x0, duration=12.667s, table=0, n_packets=8, n_bytes=336, priority=3,arp actions=FLOOD
 cookie=0x0, duration=12.667s, table=0, n_packets=23, n_bytes=1978, priority=2,ipv6 actions=IN_PORT
 cookie=0x0, duration=12.667s, table=0, n_packets=0, n_bytes=0, priority=1,ip actions=IN_PORT
```

Figure 7: Running 'dpctl dump-flows'

This shows the four rules which the controller installed onto the switch. The first rule (priority 4) is to accept/flood ICMP packets (with source and destination IP being IPv4). The next rule (priority

3) is to accept/flood any ARP packets. The next rule (priority 2) is to drop/send via in-port any ipv6 packets. The final rule (priority 1) is to drop/send via in-port any ipv4 packets.