Homework 1 Report

Christina Pavlopoulou
cpavl001@ucr.edu

Andres Calderon
acald013@ucr.edu

May 13, 2016

Part A

For the part A we only changed the code of the file topo-2sw-2host.py (listing 1) to design the topology described in the assignment.

1. Running the command h1 ping h2, we observe that the time needed for the procedure to be completed is 32.901 ms. Running h1 ping h5 takes 33.270 ms. We observe that it takes approximately the same time. This happens because the controller is set to deliver packets in all the hosts, so it doesn't matter which is the host. The other noticeable thing is that in both cases (h1 ping h2 and h1 ping h5) all the hosts and switches observe the same traffic because of the aforementioned reason.
2. The throughput that we get between h1 and h2 is 12.1 Mbits/sec while the one between h1 and h5 is 5.28Mbits/sec. This happens because the hosts h1 and h2 are connected directly while h1 and h5 are not.
3. If we run the command pingall we see that all the hosts are reachable (See figure 1).

Part B

1. In this case, the output of command h1 ping h2 was 31.864 ms. Results for h1 ping h5 was 31.955 ms. For this case, we only observe traffic

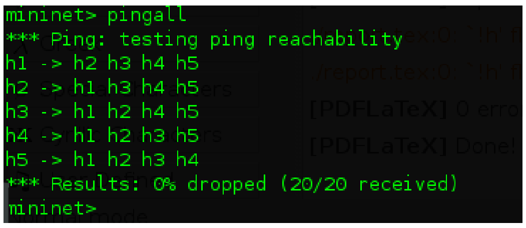
```python
from mininet.topo import Topo

class MyTopo2( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        leftUpHost = self.addHost( 'h1')
        leftDownHost = self.addHost( 'h2')
        leftSwitch = self.addSwitch( 's1' )
        middleSwitch = self.addSwitch( 's2' )
        rightSwitch = self.addSwitch( 's3' )
        rightUpHost = self.addHost( 'h3')
        rightMiddleHost = self.addHost( 'h4')
        rightDownHost = self.addHost( 'h5')

        # Add links
        self.addLink( leftUpHost, leftSwitch )
        self.addLink( leftDownHost, leftSwitch )
    self.addLink( leftSwitch, middleSwitch )
    self.addLink( middleSwitch, rightSwitch )
        self.addLink( rightSwitch, rightUpHost )
        self.addLink( rightSwitch, rightMiddleHost )
        self.addLink( rightSwitch, rightDownHost )


topos = { 'mytopo2': ( lambda: MyTopo2() ) }
```

Listing 1: Basic topology.

between the host involved in the ping procedure because the new controller map the mac address to the corresponding port and not to all the port as in the first part.

2. The iperf output between h1 and h2 was 27.9 Mbits/sec while between h1 and h5 was 8.84 Mbits/sec. We can see an improvement from part A because of the learning controller.

3. The pingall command retrieves the same results showing that all the hosts are reachable.



```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
mininet>
```

Figure 1: Output of pingall command.

```
def act_like_switch (self,event, packet, packet_in):
  """
  Implement switch-like behavior.
  """
  self.mac_to_port[packet.src] = event.port

  if packet.dst in self.mac_to_port:
    outport=self.mac_to_port[packet.dst]
    self.resend_packet(packet_in, outport)
  else:
    self.resend_packet(packet_in, of.OFPP_ALL)
```

Listing 2: Controller part B.

Part C

For the part C we changed the controller code in the file of_tutorial.py (listing 3) to make the switches even smarter than part B.

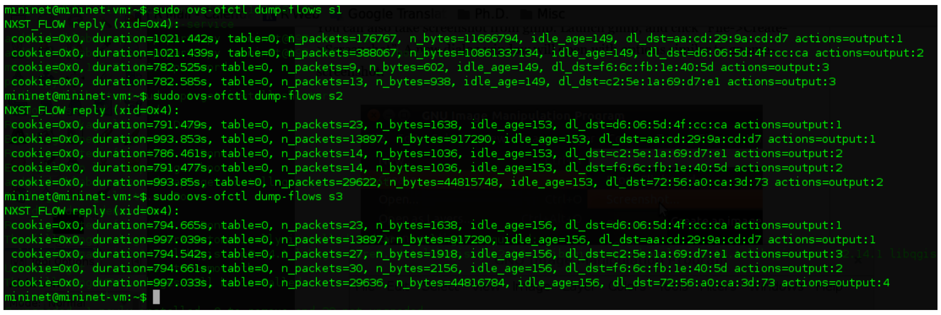
1. At this part it only takes 2.740 ms to ping between h1 and h2 and 23.840 ms between h1 and h5. The big difference is because of the larger distance between h1 and h5 than distance between h1 and h2. Comparing with the first part it takes less time in both cases since the controller accepts the packets from which the switch doesn't have an entry flow. Here only the hosts between which the ping is performed have traffic.

2. The throughput between h1 and h2 is 12.6 Gbits/sec and between h1 and h5 is 31.4 Mbis/sec. The reason for the difference is the same as in part A and B. However, the throughput is even more than in parts B and C because the switches are even smarter.

3. The pingall command verifies that all the hosts are reachable.

```python
def act_like_switch (self, event, packet, packet_in):
  """
  Implement switch-like behavior.
  """
  self.mac_to_port[packet.src]=event.port

  if packet.dst in self.mac_to_port:
    outport=self.mac_to_port[packet.dst]
    msg = of.ofp_flow_mod()
    msg.match.dl_dst = packet.dst
    msg.buffer_id = event.ofp.buffer_id
    msg.actions.append(of.ofp_action_output(port=outport))
    self.connection.send(msg)
  else:
   self.resend_packet(packet_in, of.OFPP_ALL)
```

Listing 3: Controller part C.



```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=1021.442s, table=0, n_packets=176767, n_bytes=11666794, idle_age=149, dl_dst=aa:cd:29:9a:cd:d7 actions=output:1
 cookie=0x0, duration=1021.439s, table=0, n_packets=388067, n_bytes=10861337134, idle_age=149, dl_dst=d6:06:5d:4f:cc:ca actions=output:2
 cookie=0x0, duration=782.525s, table=0, n_packets=9, n_bytes=602, idle_age=149, dl_dst=f6:6c:fb:1e:40:5d actions=output:3
 cookie=0x0, duration=782.585s, table=0, n_packets=13, n_bytes=938, idle_age=149, dl_dst=c2:5e:1a:69:d7:e1 actions=output:3
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s2
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=791.479s, table=0, n_packets=23, n_bytes=1638, idle_age=153, dl_dst=d6:06:5d:4f:cc:ca actions=output:1
 cookie=0x0, duration=993.853s, table=0, n_packets=13897, n_bytes=917290, idle_age=153, dl_dst=aa:cd:29:9a:cd:d7 actions=output:1
 cookie=0x0, duration=786.461s, table=0, n_packets=14, n_bytes=1036, idle_age=153, dl_dst=c2:5e:1a:69:d7:e1 actions=output:2
 cookie=0x0, duration=791.477s, table=0, n_packets=14, n_bytes=1036, idle_age=153, dl_dst=f6:6c:fb:1e:40:5d actions=output:2
 cookie=0x0, duration=993.85s, table=0, n_packets=29622, n_bytes=44815748, idle_age=153, dl_dst=72:56:a0:ca:3d:73 actions=output:2
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s3
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=794.665s, table=0, n_packets=23, n_bytes=1638, idle_age=156, dl_dst=d6:06:5d:4f:cc:ca actions=output:1
 cookie=0x0, duration=997.039s, table=0, n_packets=13897, n_bytes=917290, idle_age=156, dl_dst=aa:cd:29:9a:cd:d7 actions=output:1
 cookie=0x0, duration=794.542s, table=0, n_packets=27, n_bytes=1918, idle_age=156, dl_dst=c2:5e:1a:69:d7:e1 actions=output:3
 cookie=0x0, duration=794.661s, table=0, n_packets=30, n_bytes=2156, idle_age=156, dl_dst=f6:6c:fb:1e:40:5d actions=output:2
 cookie=0x0, duration=997.033s, table=0, n_packets=29636, n_bytes=44816784, idle_age=156, dl_dst=72:56:a0:ca:3d:73 actions=output:4
mininet@mininet-vm:~$ 
```

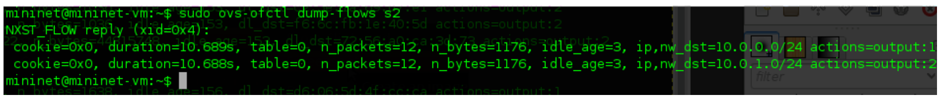
Figure 2: Output of ovs-ofctl dump-flow for the three switches.

4. The number of rules is 4 for switch 1 and 5 for switches 2 and 3 (figure 2). Switches 2 and 3 send the flow to each hosts according to the IP address they get.

Part D

We use the topology in listing 4 for this part of the assignment. To install IP matching rules in switch 2 we use the commands:

```
  sudo ovs-ofctl add-flow dl_type=0x0800,nw_dst=10.0.0.0/24,actions=output:1
sudo ovs-ofctl add-flow dl_type=0x0800,nw_dst=10.0.1.0/24,actions=output:2
```



```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s2
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=10.689s, table=0, n_packets=12, n_bytes=1176, idle_age=3, ip,nw_dst=10.0.0.0/24 actions=output:1
 cookie=0x0, duration=10.688s, table=0, n_packets=12, n_bytes=1176, idle_age=3, ip,nw_dst=10.0.1.0/24 actions=output:2
mininet@mininet-vm:~$
```

Figure 3: Output of ovs-ofctl dump-flow for switch 2.

Listing 4: New topology.

1. The result of ping between h1 and h2 in this case was 4.041 ms. Between h1 and h5 was 23.189 ms. We can see a big difference similar to what we saw in part C. We observe that when we ping between hosts in the same switch, the traffic is only between these two hosts. However, when we ping host in different switches there is traffic in all the hosts.

2. The throughput between h1 and h2 was 5.71 Gbits/sec while the throughput between h1 and h5 was 43.8 Mbits/sec. The behavior is similar to the one we observe in part C.

3. The pingall command verifies that all the hosts are reachable.

4. For switch 2 we see two rules now (figure 3). This happens because every flow which begins with 10.0.0 is sent to the left and every flow starting with 10.0.1 goes to the right. Switches 1 and 3 remain the same.

5. This controller is better than options in parts A and B. It is similar than controller in part C but we evaluate less rules in part D. However, in part C the traffic was observed in less hosts when we ping among them.