Mokhtari (9831143) – Computer Networks 2 – PRJ 03

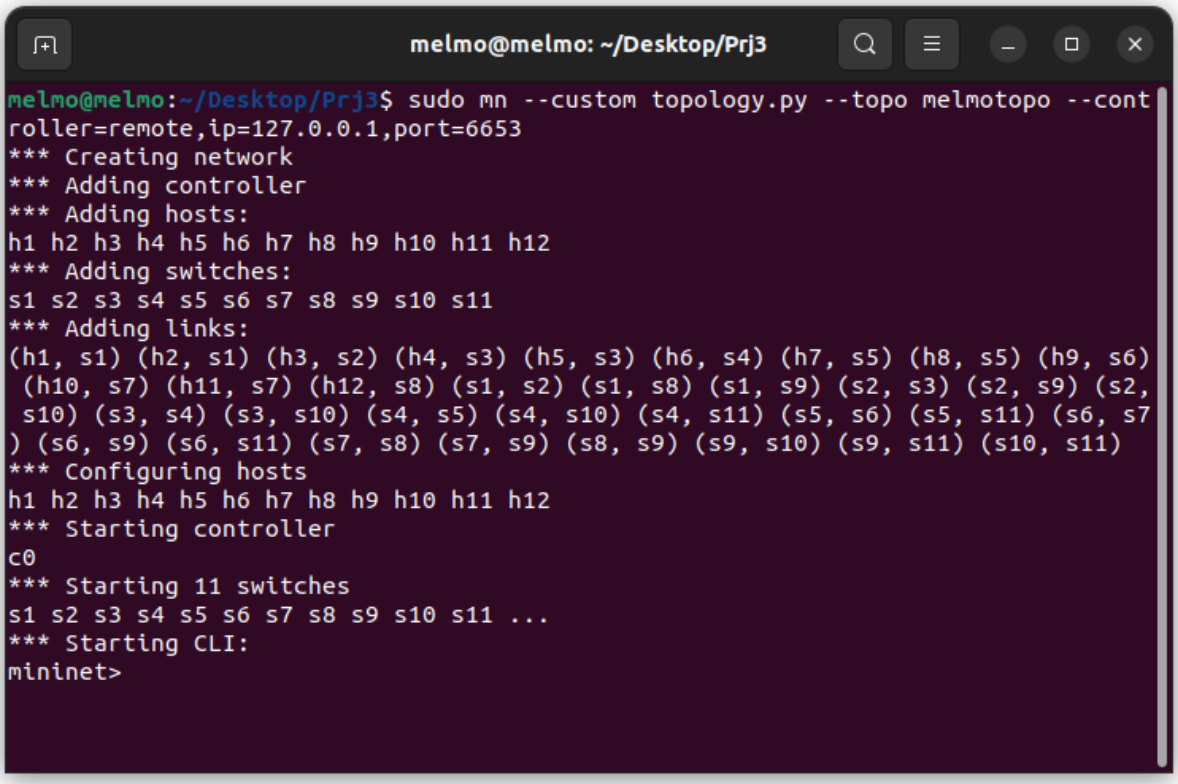
P1 & P2:

The topology on the following page is generated by the code below:

- (This code's .py file is also attached to this assignment.)

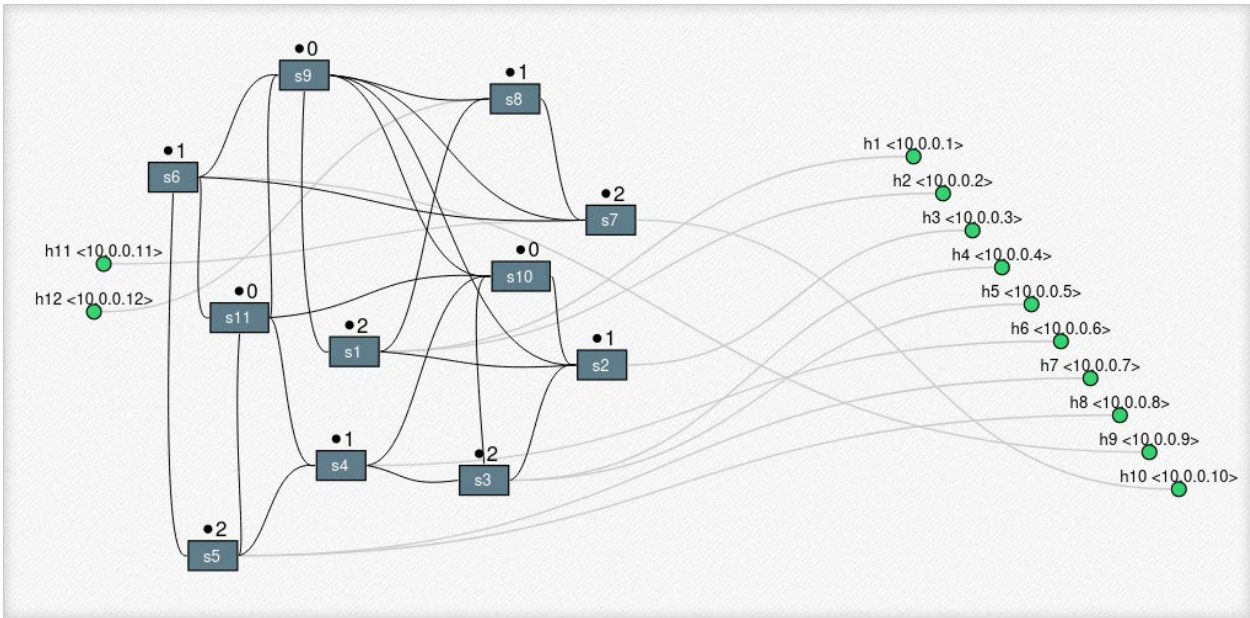
```
from mininet.topo import Topo

class Project(Topo):
    def __init__(self):
        # Initialize topology
        Topo.__init__(self)
        # Add hosts
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        h3 = self.addHost('h3')
        h4 = self.addHost('h4')
        h5 = self.addHost('h5')
        h6 = self.addHost('h6')
        h7 = self.addHost('h7')
        h8 = self.addHost('h8')
        h9 = self.addHost('h9')
        h10 = self.addHost('h10')
        h11 = self.addHost('h11')
        h12 = self.addHost('h12')
        # Add switches
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')
        s4 = self.addSwitch('s4')
        s5 = self.addSwitch('s5')
        s6 = self.addSwitch('s6')
        s7 = self.addSwitch('s7')
        s8 = self.addSwitch('s8')
        s9 = self.addSwitch('s9')
        s10 = self.addSwitch('s10')
        s11 = self.addSwitch('s11')
        # Add links
        self.addLink(h1, s1)
        self.addLink(h2, s1)
        self.addLink(h3, s2)
        self.addLink(h4, s3)
        self.addLink(h5, s3)
        self.addLink(h6, s4)
        self.addLink(h7, s5)
        self.addLink(h8, s5)
        self.addLink(h9, s6)
        self.addLink(h10, s7)
        self.addLink(h11, s7)
```

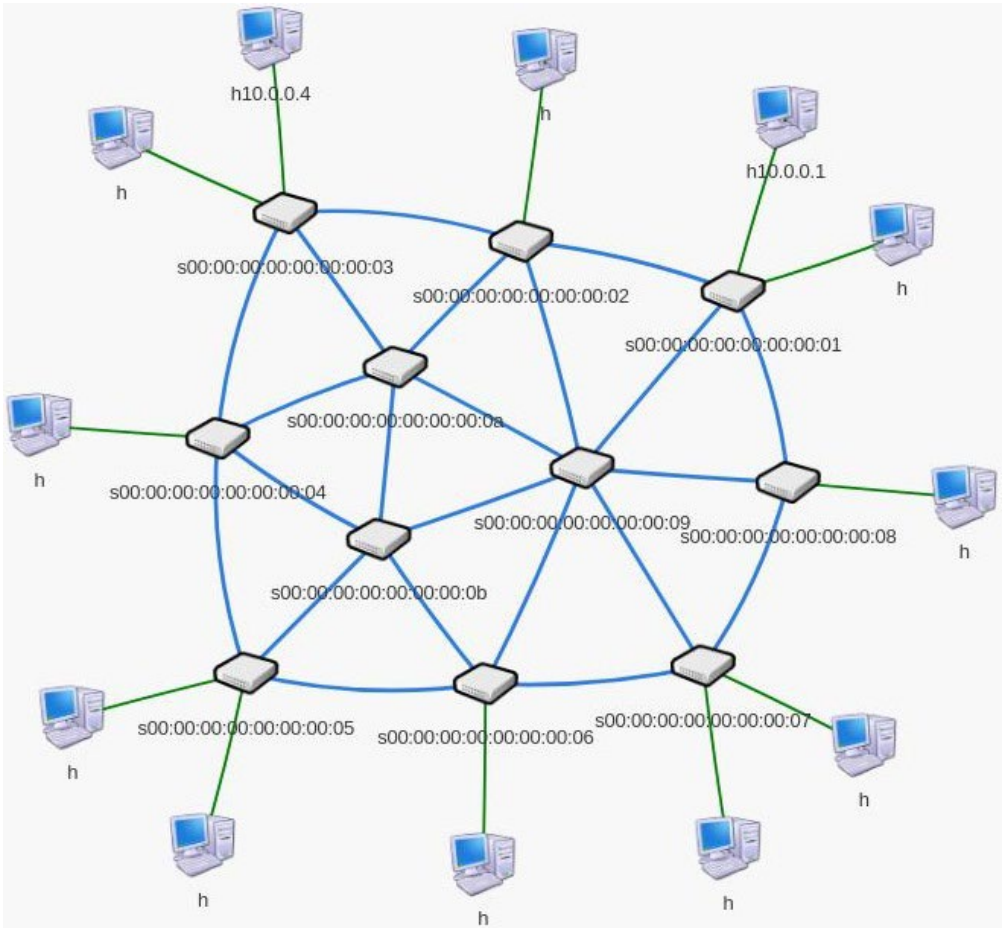
```
        self.addLink(h12, s8)
        self.addLink(s1, s2)
        self.addLink(s1, s8)
        self.addLink(s1, s9)
        self.addLink(s2, s3)
        self.addLink(s2, s9)
        self.addLink(s2, s10)
        self.addLink(s3, s10)
        self.addLink(s3, s4)
        self.addLink(s4, s5)
        self.addLink(s4, s10)
        self.addLink(s4, s11)
        self.addLink(s5, s11)
        self.addLink(s5, s6)
        self.addLink(s6, s7)
        self.addLink(s6, s9)
        self.addLink(s6, s11)
        self.addLink(s7, s8)
        self.addLink(s7, s9)
        self.addLink(s8, s9)
        self.addLink(s9, s10)
        self.addLink(s9, s11)
        self.addLink(s10, s11)
topos = {'melmotopo': (lambda: Project())}
```



Mokhtari (9831143) – Computer Networks 2 – PRJ 03

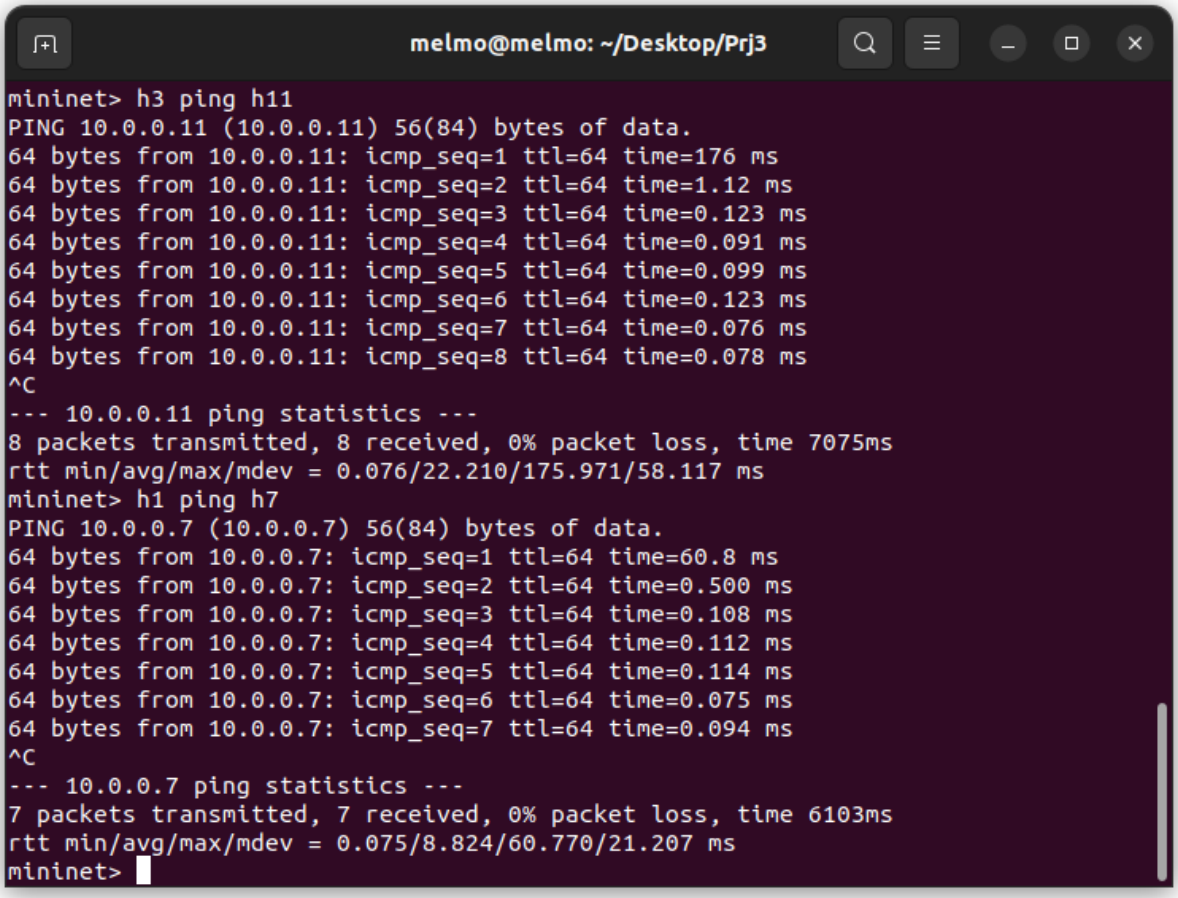
The topology is visualized using a floodlight and an online topology visualizer app. The website "http://demo.spear.narmox.com/app/?apiurl=demo#!/mininet" generated the image below. Floodlight automatically generates the image at the bottom of the page..

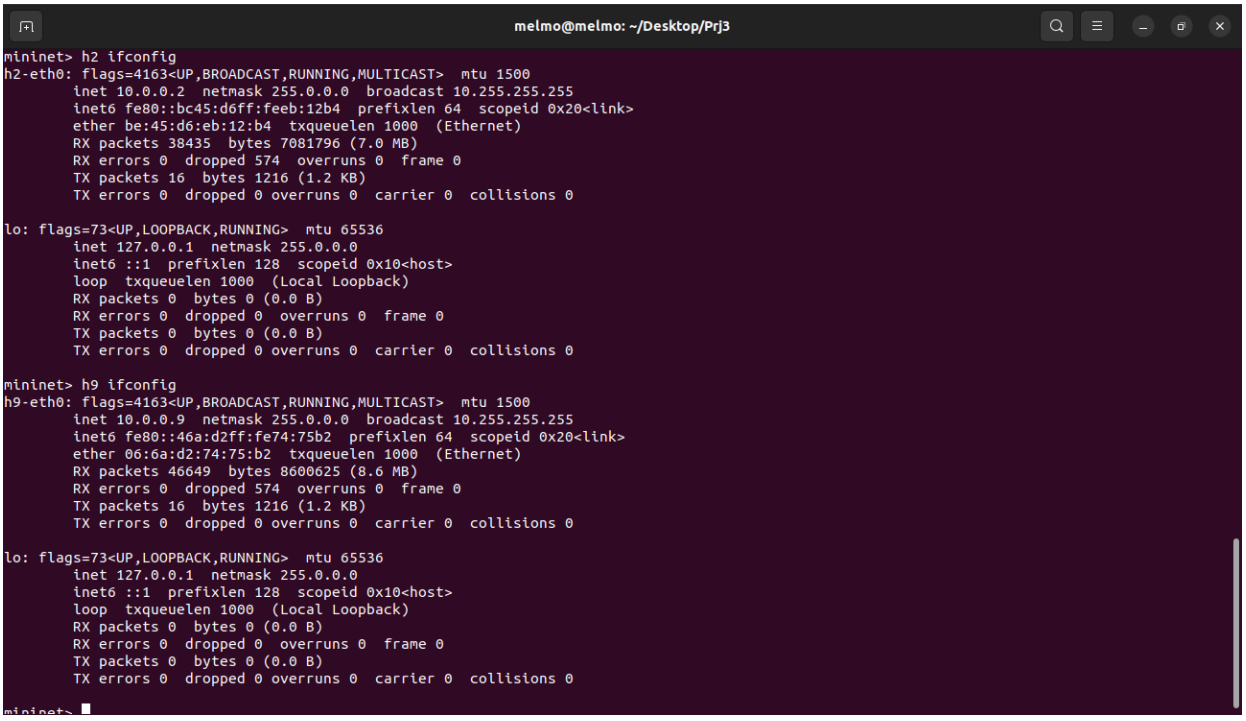




Mokhtari (9831143) – Computer Networks 2 – PRJ 03

P3:





Mokhtari (9831143) – Computer Networks 2 – PRJ 03

"Node: h2"

```
root@melmo:/home/melmo/Desktop/Prj3# ping 10.0.0.9
PING 10.0.0.9 (10.0.0.9) 56(84) bytes of data.
64 bytes from 10.0.0.9: icmp_seq=1 ttl=64 time=0.823 ms
64 bytes from 10.0.0.9: icmp_seq=2 ttl=64 time=0.070 ms
64 bytes from 10.0.0.9: icmp_seq=3 ttl=64 time=0.104 ms
64 bytes from 10.0.0.9: icmp_seq=4 ttl=64 time=0.117 ms
64 bytes from 10.0.0.9: icmp_seq=5 ttl=64 time=0.154 ms
64 bytes from 10.0.0.9: icmp_seq=6 ttl=64 time=0.118 ms
64 bytes from 10.0.0.9: icmp_seq=7 ttl=64 time=0.072 ms
^C
--- 10.0.0.9 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6130ms
rtt min/avg/max/mdev = 0.070/0.208/0.823/0.252 ms
root@melmo:/home/melmo/Desktop/Prj3#
```

"Node: h9"

```
root@melmo:/home/melmo/Desktop/Prj3# ping 10.0.0.2
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=39.9 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.080 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.099 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.102 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.069 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.103 ms
^C
--- 10.0.0.2 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5077ms
rtt min/avg/max/mdev = 0.069/6.721/39.873/14.826 ms
root@melmo:/home/melmo/Desktop/Prj3#
```

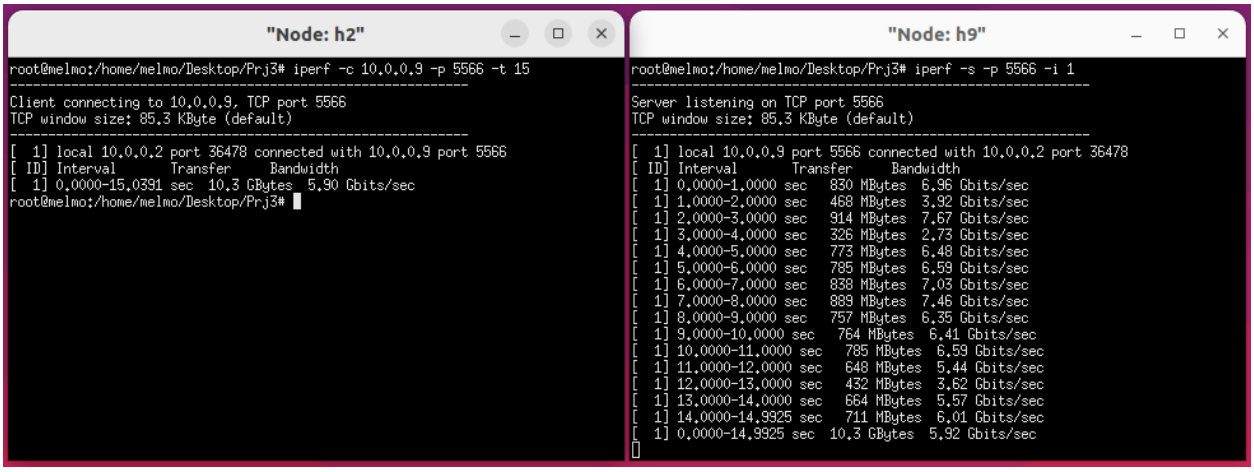
P4:

melmo@melmo: ~/Desktop/Prj3

```
mininet> nodes
available nodes are:
c0 h1 h10 h11 h12 h2 h3 h4 h5 h6 h7 h8 h9 s1 s10 s11 s2 s3 s4 s5 s6 s7 s8 s9
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s2-eth1
h4 h4-eth0:s3-eth1
h5 h5-eth0:s3-eth2
h6 h6-eth0:s4-eth1
h7 h7-eth0:s5-eth1
h8 h8-eth0:s5-eth2
h9 h9-eth0:s6-eth1
h10 h10-eth0:s7-eth1
h11 h11-eth0:s7-eth2
h12 h12-eth0:s8-eth1
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:s2-eth2 s1-eth4:s8-eth2 s1-eth5:s9-eth1
s2 lo:  s2-eth1:h3-eth0 s2-eth2:s1-eth3 s2-eth3:s3-eth3 s2-eth4:s9-eth2 s2-eth5:s10-eth1
s3 lo:  s3-eth1:h4-eth0 s3-eth2:h5-eth0 s3-eth3:s2-eth3 s3-eth4:s10-eth2 s3-eth5:s4-eth2
s4 lo:  s4-eth1:h6-eth0 s4-eth2:s3-eth5 s4-eth3:s5-eth3 s4-eth4:s10-eth3 s4-eth5:s11-eth1
s5 lo:  s5-eth1:h7-eth0 s5-eth2:h8-eth0 s5-eth3:s4-eth3 s5-eth4:s11-eth2 s5-eth5:s6-eth2
s6 lo:  s6-eth1:h9-eth0 s6-eth2:s5-eth5 s6-eth3:s7-eth3 s6-eth4:s9-eth3 s6-eth5:s11-eth3
s7 lo:  s7-eth1:h10-eth0 s7-eth2:h11-eth0 s7-eth3:s6-eth3 s7-eth4:s8-eth3 s7-eth5:s9-eth4
s8 lo:  s8-eth1:h12-eth0 s8-eth2:s1-eth4 s8-eth3:s7-eth4 s8-eth4:s9-eth5
s9 lo:  s9-eth1:s1-eth5 s9-eth2:s2-eth4 s9-eth3:s6-eth4 s9-eth4:s7-eth5 s9-eth5:s8-eth4 s9-eth6:s10-eth4 s9-eth7:s11-eth4
s10 lo:  s10-eth1:s2-eth5 s10-eth2:s3-eth4 s10-eth3:s4-eth4 s10-eth4:s9-eth6 s10-eth5:s11-eth5
s11 lo:  s11-eth1:s4-eth5 s11-eth2:s5-eth4 s11-eth3:s6-eth5 s11-eth4:s9-eth7 s11-eth5:s10-eth5
c0
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=19363>
<Host h2: h2-eth0:10.0.0.2 pid=19365>
<Host h3: h3-eth0:10.0.0.3 pid=19367>
<Host h4: h4-eth0:10.0.0.4 pid=19369>
<Host h5: h5-eth0:10.0.0.5 pid=19371>
<Host h6: h6-eth0:10.0.0.6 pid=19373>
<Host h7: h7-eth0:10.0.0.7 pid=19375>
<Host h8: h8-eth0:10.0.0.8 pid=19377>
<Host h9: h9-eth0:10.0.0.9 pid=19379>
<Host h10: h10-eth0:10.0.0.10 pid=19381>
<Host h11: h11-eth0:10.0.0.11 pid=19383>
<Host h12: h12-eth0:10.0.0.12 pid=19385>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None,s1-eth4:None,s1-eth5:None pid=19390>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None,s2-eth4:None,s2-eth5:None pid=19393>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None,s3-eth4:None,s3-eth5:None pid=19396>
<OVSSwitch s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None,s4-eth3:None,s4-eth4:None,s4-eth5:None pid=19399>
<OVSSwitch s5: lo:127.0.0.1,s5-eth1:None,s5-eth2:None,s5-eth3:None,s5-eth4:None,s5-eth5:None pid=19402>
<OVSSwitch s6: lo:127.0.0.1,s6-eth1:None,s6-eth2:None,s6-eth3:None,s6-eth4:None,s6-eth5:None pid=19405>
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
h3-eth0<->s2-eth1 (OK OK)
h4-eth0<->s3-eth1 (OK OK)
h5-eth0<->s3-eth2 (OK OK)
h6-eth0<->s4-eth1 (OK OK)
h7-eth0<->s5-eth1 (OK OK)
h8-eth0<->s5-eth2 (OK OK)
h9-eth0<->s6-eth1 (OK OK)
h10-eth0<->s7-eth1 (OK OK)
h11-eth0<->s7-eth2 (OK OK)
h12-eth0<->s8-eth1 (OK OK)
s1-eth3<->s2-eth2 (OK OK)
s1-eth4<->s8-eth2 (OK OK)
s1-eth5<->s9-eth1 (OK OK)
s2-eth3<->s3-eth3 (OK OK)
s2-eth4<->s9-eth2 (OK OK)
s2-eth5<->s10-eth1 (OK OK)
s3-eth5<->s4-eth2 (OK OK)
s3-eth4<->s10-eth2 (OK OK)
s4-eth3<->s5-eth3 (OK OK)
s4-eth4<->s10-eth3 (OK OK)
s4-eth5<->s11-eth1 (OK OK)
s5-eth5<->s6-eth2 (OK OK)
s5-eth4<->s11-eth2 (OK OK)
s6-eth3<->s7-eth3 (OK OK)
s6-eth4<->s9-eth3 (OK OK)
s6-eth5<->s11-eth3 (OK OK)
s7-eth4<->s8-eth3 (OK OK)
s7-eth5<->s9-eth4 (OK OK)
s8-eth4<->s9-eth5 (OK OK)
s9-eth6<->s10-eth4 (OK OK)
s9-eth7<->s11-eth4 (OK OK)
s10-eth5<->s11-eth5 (OK OK)
mininet>
```

Mokhtari (9831143) – Computer Networks 2 – PRJ 03

P5:




```
Iperf -s -p 5566 -i 1
```

Start the TCP server (-s) at h9 with port 5566 (-p). Monitor the results every one second (-i). The default setting is TCP.

```
Iperf -c 10.0.0.9 -p 5566 -t 15
```

Start the TCP client (-c) at h2. Also, set the transmission duration (-t) to 15 seconds. Note: After -c, you need to specify the server's IP address, which is 10.0.0.9.

In this case, H2 represents the client side, where throughput is described, and H9 represents the server side, where the average bandwidth in 15 seconds was 5.90 GBit/sec.

It's important to keep in mind that when using TCP, packages are sent window by window, with the window size in this instance being 85.3 KBytes, as shown in the screenshots.

P6:

The three paths I used were as follows:

1. Path 1 is defined as h12->s8->s1->s2->s3->s4->s5->h8.
2. Path 2 is defined as h5->s3->s10->s2->s9->s1->s8->s7->h10.
3. Path 3 is defined as h6->s4->s0b->s6->s9->s8->h12.

P7:

To accomplish this, we must run the flow files on the terminal. Path 1 is described in more detail below.

The same justifications apply to other paths and flows as well.

Mokhtari (9831143) – Computer Networks 2 – PRJ 03

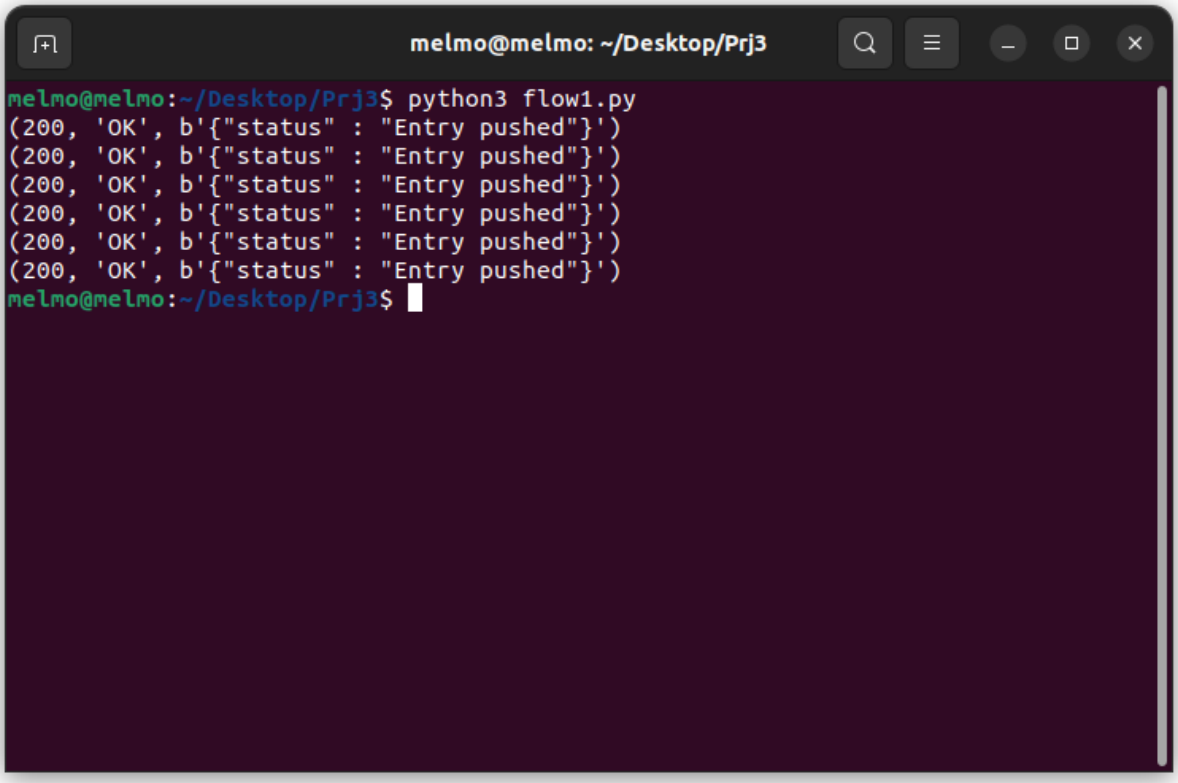
The code below creates flow 1:

- (The .py file for this code is also attached to this assignment.)

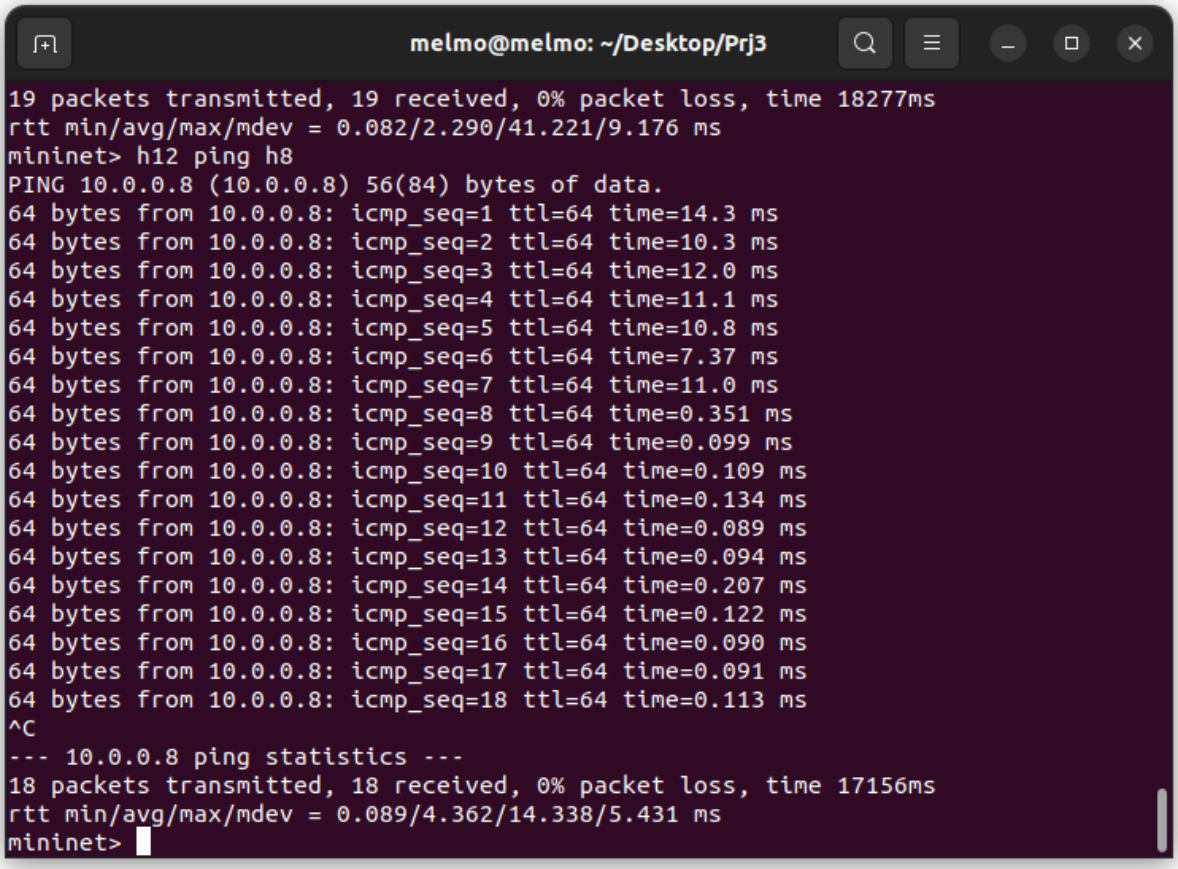
```
import http.client
import json

class StaticEntryPusher(object):
    def __init__(self,server):
        self.server = server
    def get(self,data):
        ret = self.rest_call({},'GET')
        return json.loads(ret[2])

    def Set(self,data):
        ret = self.rest_call(data,'POST')
        return ret[0] == 200

    def remove(self,objtype,data):
        ret = self.rest_call(data,'DELETE')
        return ret[0] == 200

    def rest_call(self,data,action):
        path ='/wm/staticentrypusher/json'
        header = {
            'Content-Type':'application/json',
            'Accept':'application/json'
        }
        body = json.dumps(data)
        Conn = http.client.HTTPConnection(self.server, 8080)
        Conn.request(action,path,body,header)
        response = Conn.getresponse()
        ret = (response.status,response.reason,response.read())
        print (ret)
        Conn.close()
        return ret
pusher = StaticEntryPusher('127.0.0.1')

entry1 ={
    "switch":"00:00:00:00:00:00:00:08",
    "name": "entry1",
    "eth_type":"0x0800",
    "ipv4_src":"10.0.0.12",
    "ipv4_dst": "10.0.0.8",
    "priority":"32768",
    "in_port":"1",
    "active":"true",
    "actions":"output=2"
```

```
}

entry2 = {
    "switch": "00:00:00:00:00:00:00:01",
    "name": "entry2",
    "eth_type": "0x0800",
    "ipv4_src": "10.0.0.12",
    "ipv4_dst": "10.0.0.8",
    "priority": "32768",
    "in_port": "4",
    "active": "true",
    "actions": "output=3"
}

entry3 = {
    "switch": "00:00:00:00:00:00:00:02",
    "name": "entry3",
    "eth_type": "0x0800",
    "ipv4_src": "10.0.0.12",
    "ipv4_dst": "10.0.0.8",
    "priority": "32768",
    "in_port": "2",
    "active": "true",
    "actions": "output=3"
}

entry4 = {
    "switch": "00:00:00:00:00:00:00:03",
    "name": "entry4",
    "eth_type": "0x0800",
    "ipv4_src": "10.0.0.12",
    "ipv4_dst": "10.0.0.8",
    "priority": "32768",
    "in_port": "3",
    "active": "true",
    "actions": "output=5"
}

entry5 = {
    "switch": "00:00:00:00:00:00:00:04",
    "name": "entry5",
    "eth_type": "0x0800",
    "ipv4_src": "10.0.0.12",
    "ipv4_dst": "10.0.0.8",
    "priority": "32768",
    "in_port": "2",
    "active": "true",
```

```
    "actions": "output=3"
}

entry6 = {
    "switch": "00:00:00:00:00:00:00:05",
    "name": "entry6",
    "eth_type": "0x0800",
    "ipv4_src": "10.0.0.12",
    "ipv4_dst": "10.0.0.8",
    "priority": "32768",
    "in_port": "3",
    "active": "true",
    "actions": "output=2"
}

pusher.Set(entry1)
pusher.Set(entry2)
pusher.Set(entry3)
pusher.Set(entry4)
pusher.Set(entry5)
pusher.Set(entry6)
```



Following the execution of the flow 1 code, h12 and h9 will be assigned IP addresses on Topolgy. Let's ping these two now:



As an example, let's check the status of switch 8 together:

Because we are pinging, the sender sends a request to the destination using the ICMP protocol, and the destination must respond, so two flows are created, the origin of which is 10.0.0.12 and the destination of which is 10.0.0.8, and this address is reversed in the second flow.

The input and output ports are also shown in the figure below, and because each flow table contains two parts, match and action, these two items are also shown in each flow.

For instance, in flow 0, the action is to exit from port 2. It is also determined how many packets were sent from the sender to the receiver, which is equal to the number of packets sent in flow 1 with the new source and destination.

The MAC addresses of the source and destination hosts are eth dest and eth src, respectively.

```
Flow 0:
  Packet count: "244"
  Matches: {"in_port":"1","eth_type":"0x800","ipv4_src":"10.0.0.12","ipv4_dst":"10.0.0.8"}
  Actions: "output=2"
Flow 1:
  Packet count: "244"
  Matches: {"in_port":"3","eth_dst":"ce:69:27:b7:11:5c","eth_src":"da:f1:2b:fe:9e:0b","eth_type":"0x800","ipv4_src":"10.0.0.8","ipv4_dst":"10.0.0.12"
  Actions: "output=1"
Flow 2:
  Packet count: "1"
  Matches: {"in_port":"4","eth_dst":"ce:69:27:b7:11:5c","eth_src":"da:f1:2b:fe:9e:0b","eth_type":"0x806"}
  Actions: "output=1"
Flow 3:
  Packet count: "0"
  Matches: {"in_port":"1","eth_dst":"da:f1:2b:fe:9e:0b","eth_src":"ce:69:27:b7:11:5c","eth_type":"0x806"}
  Actions: "output=4"
Flow 4:
  Packet count: "1528166"
  Matches: {}
  Actions: "output=controller"
```


Switch 5 is another example, the details of which are shown in the figure below:


```
Flow 0:
  Packet count: "233"
  Matches: {"in_port":"3","eth_type":"0x800","ipv4_src":"10.0.0.12","ipv4_dst":"10.0.0.8"}
  Actions: "output=2"
Flow 1:
  Packet count: "232"
  Matches: {"in_port":"2","eth_dst":"ce:69:27:b7:11:5c","eth_src":"da:f1:2b:fe:9e:0b","eth_type":"0x800","ipv4_src":"10.0.0.8","ipv4_dst":"10.0.0.12"}
  Actions: "output=5"
Flow 2:
  Packet count: "1848614"
  Matches: {}
  Actions: "output=controller"
```


The status of other switches can be checked in the same way.

We can also visit the flow table of each switch in the switches info section of the Floodlight UI, which allows us to ensure that the entry with the highest priority is placed in the first line, which is the one we defined.

It is also obvious that the number of packets sent in Switch 8 (the source host is connected to it) and the number of packets received in Switch 5 (the destination host is connected to it) will increase because we established a connection between the two hosts that were connected to these switches (h12 ping h8), and the last column of both tables shows the output port or the action to be performed.

The same trait can be continued for flows 2 and 3.

- The Python code for each of these two flows is provided as an attachment to this assignment and is shown below, respectively.

The code below creates flow 2:

```
import http.client
import json
```

```
class StaticEntryPusher(object):
    def __init__(self, server):
        self.server = server

    def get(self, data):
        ret = self.rest_call({}, 'GET')
        return json.loads(ret[2])

    def Set(self, data):
        ret = self.rest_call(data, 'POST')
        return ret[0] == 200

    def remove(self, objtype, data):
        ret = self.rest_call(data, 'DELETE')
        return ret[0] == 200

    def rest_call(self, data, action):
        path = '/wm/staticentrypusher/json'
        header = {
            'Content-Type': 'application/json',
            'Accept': 'application/json'
        }
        body = json.dumps(data)
        Conn = http.client.HTTPConnection(self.server, 8080)
        Conn.request(action, path, body, header)
        response = Conn.getresponse()
        ret = (response.status, response.reason,
response.read())
        print(ret)
        Conn.close()
        return ret

pusher = StaticEntryPusher('127.0.0.1')

entry1 = {
    "switch": "00:00:00:00:00:00:00:03",
    "name": "entry1",
    "eth_type": "0x0800",
    "ipv4_src": "10.0.0.05",
    "ipv4_dst": "10.0.0.10",
    "priority": "32768",
    "in_port": "2",
    "active": "true",
    "actions": "output=4"
}

entry2 = {
```

```
    "switch": "00:00:00:00:00:00:00:0a",
    "name": "entry2",
    "eth_type": "0x0800",
    "ipv4_src": "10.0.0.05",
    "ipv4_dst": "10.0.0.10",
    "priority": "32768",
    "in_port": "2",
    "active": "true",
    "actions": "output=1"
}

entry3 = {
    "switch": "00:00:00:00:00:00:00:09",
    "name": "entry3",
    "eth_type": "0x0800",
    "ipv4_src": "10.0.0.05",
    "ipv4_dst": "10.0.0.10",
    "priority": "32768",
    "in_port": "2",
    "active": "true",
    "actions": "output=1"
}

entry4 = {
    "switch": "00:00:00:00:00:00:00:01",
    "name": "entry4",
    "eth_type": "0x0800",
    "ipv4_src": "10.0.0.05",
    "ipv4_dst": "10.0.0.10",
    "priority": "32768",
    "in_port": "5",
    "active": "true",
    "actions": "output=4"
}

entry5 = {
    "switch": "00:00:00:00:00:00:00:08",
    "name": "entry5",
    "eth_type": "0x0800",
    "ipv4_src": "10.0.0.05",
    "ipv4_dst": "10.0.0.10",
    "priority": "32768",
    "in_port": "2",
    "active": "true",
    "actions": "output=3"
}
```

```
entry6 = {
    "switch": "00:00:00:00:00:00:00:07",
    "name": "entry6",
    "eth_type": "0x0800",
    "ipv4_src": "10.0.0.05",
    "ipv4_dst": "10.0.0.10",
    "priority": "32768",
    "in_port": "4",
    "active": "true",
    "actions": "output=2"
}

pusher.Set(entry1)
pusher.Set(entry2)
pusher.Set(entry3)
pusher.Set(entry4)
pusher.Set(entry5)
pusher.Set(entry6)
```

The code below creates flow 3:

```
import http.client
import json

class StaticEntryPusher(object):
    def __init__(self, server):
        self.server = server

    def get(self, data):
        ret = self.rest_call({}, 'GET')
        return json.loads(ret[2])

    def Set(self, data):
        ret = self.rest_call(data, 'POST')
        return ret[0] == 200

    def remove(self, objtype, data):
        ret = self.rest_call(data, 'DELETE')
        return ret[0] == 200

    def rest_call(self, data, action):
        path = '/wm/staticentrypusher/json'
        header = {
            'Content-Type': 'application/json',
            'Accept': 'application/json'
        }
        body = json.dumps(data)
```

```
        Conn = http.client.HTTPConnection(self.server, 8080)
        Conn.request(action, path, body, header)
        response = Conn.getresponse()
        ret = (response.status, response.reason,
response.read())
        print(ret)
        Conn.close()
        return ret

pusher = StaticEntryPusher('127.0.0.1')

entry1 = {
    "switch": "00:00:00:00:00:00:00:04",
    "name": "entry1",
    "eth_type": "0x0800",
    "ipv4_src": "10.0.0.06",
    "ipv4_dst": "10.0.0.12",
    "priority": "32768",
    "in_port": "1",
    "active": "true",
    "actions": "output=5"
}

entry2 = {
    "switch": "00:00:00:00:00:00:00:0b",
    "name": "entry2",
    "eth_type": "0x0800",
    "ipv4_src": "10.0.0.06",
    "ipv4_dst": "10.0.0.12",
    "priority": "32768",
    "in_port": "1",
    "active": "true",
    "actions": "output=3"
}

entry3 = {
    "switch": "00:00:00:00:00:00:00:06",
    "name": "entry3",
    "eth_type": "0x0800",
    "ipv4_src": "10.0.0.06",
    "ipv4_dst": "10.0.0.12",
    "priority": "32768",
    "in_port": "5",
    "active": "true",
    "actions": "output=4"
}
```

```
entry4 = {
    "switch": "00:00:00:00:00:00:00:09",
    "name": "entry4",
    "eth_type": "0x0800",
    "ipv4_src": "10.0.0.06",
    "ipv4_dst": "10.0.0.12",
    "priority": "32768",
    "in_port": "3",
    "active": "true",
    "actions": "output=5"
}

entry5 = {
    "switch": "00:00:00:00:00:00:00:08",
    "name": "entry5",
    "eth_type": "0x0800",
    "ipv4_src": "10.0.0.06",
    "ipv4_dst": "10.0.0.12",
    "priority": "32768",
    "in_port": "4",
    "active": "true",
    "actions": "output=1"
}

pusher.Set(entry1)
pusher.Set(entry2)
pusher.Set(entry3)
pusher.Set(entry4)
pusher.Set(entry5)
```

All three Python files for flows were successfully tested in the terminal and passed!