Computer Network

Lab Assignment 1

Roll no-IIT2018142
Name-Amritansh Mishra

Q1) a) The given question asked us to create a topology in which we have four brackets, each with four forces and one top-of-rack (ToR) switch. These ToR switches are connected to a central root switch.

The steps of Algorithm
1)I made two functions, one for building the rack and connecting the four racks with the main switch.
2) The second function creates four racks and in each rack we connect a switch to the main switch.

Functions

```
def build( self ):
       self.racks = []
       rootSwitch = self.addSwitch( 's1' )
       for i in irange( 1, 4 ):
           rack = self.buildRack( i )
           self.racks.append( rack )
           for switch in rack:
               self.addLink( rootSwitch, switch )
```

This function creates the main switch s1 and four racks which are linked to s1.

1)self.racks[]-Creates an array of racks.
2)rootswitch=self.addSwitch(‘s1’)-Creates main Switch
3)The loop builds the racks and also links the racks to the main switch.

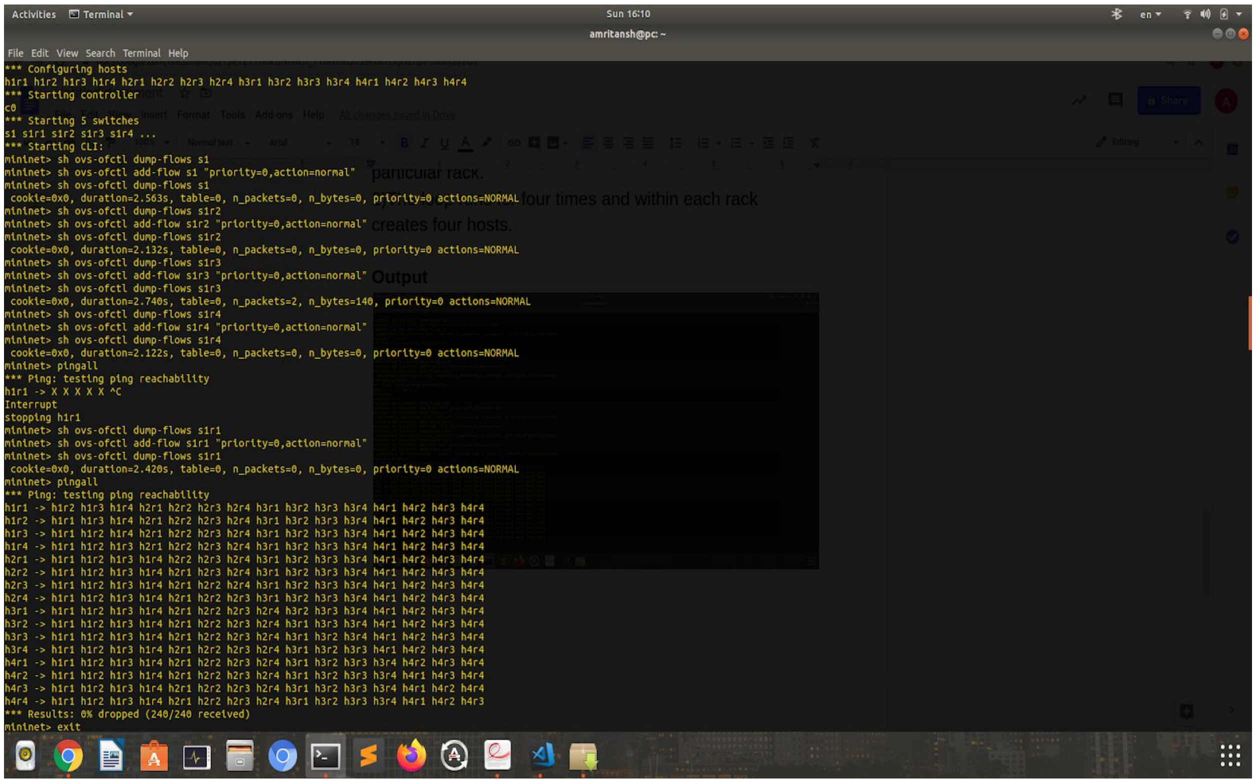
```
def buildRack( self, loc ):
       "Build a rack of hosts with a top-of-rack switch"

       dpid = ( loc * 16 ) + 1
       switch = self.addSwitch( 's1r%s' % loc, dpid='%x' % dpid )

       for n in irange( 1, 4 ):
           host = self.addHost( 'h%sr%s' % ( n, loc ) )
           self.addLink( switch, host )
```

This function creates the host in each of the four racks and then links to the switch it has.

1)switch=self.addSwitch()-creates the switch for the particular rack.

2)The loop runs for four times and within each rack creates four hosts.

Output



```
*** Configuring hosts
h1r1 h1r2 h1r3 h1r4 h2r1 h2r2 h2r3 h2r4 h3r1 h3r2 h3r3 h3r4 h4r1 h4r2 h4r3 h4r4
*** Starting controller
c0
*** Starting 5 switches
s1 s1r1 s1r2 s1r3 s1r4 ...
*** Starting CLI:
mininet> sh ovs-ofctl dump-flows s1
mininet> sh ovs-ofctl add-flow s1 "priority=0,action=normal"
mininet> sh ovs-ofctl dump-flows s1
 cookie=0x0, duration=2.563s, table=0, n_packets=0, n_bytes=0, priority=0 actions=NORMAL
mininet> sh ovs-ofctl dump-flows s1r2
mininet> sh ovs-ofctl add-flow s1r2 "priority=0,action=normal"
mininet> sh ovs-ofctl dump-flows s1r2
 cookie=0x0, duration=2.132s, table=0, n_packets=0, n_bytes=0, priority=0 actions=NORMAL
mininet> sh ovs-ofctl dump-flows s1r3
mininet> sh ovs-ofctl add-flow s1r3 "priority=0,action=normal"
mininet> sh ovs-ofctl dump-flows s1r3
 cookie=0x0, duration=2.740s, table=0, n_packets=2, n_bytes=140, priority=0 actions=NORMAL
mininet> sh ovs-ofctl dump-flows s1r4
mininet> sh ovs-ofctl add-flow s1r4 "priority=0,action=normal"
mininet> sh ovs-ofctl dump-flows s1r4
 cookie=0x0, duration=2.122s, table=0, n_packets=0, n_bytes=0, priority=0 actions=NORMAL
mininet> pingall
*** Ping: testing ping reachability
h1r1 -> X X X X X ^C
Interrupt
stopping h1r1
mininet> sh ovs-ofctl dump-flows s1r1
mininet> sh ovs-ofctl add-flow s1r1 "priority=0,action=normal"
mininet> sh ovs-ofctl dump-flows s1r1
 cookie=0x0, duration=2.420s, table=0, n_packets=0, n_bytes=0, priority=0 actions=NORMAL
mininet> pingall
*** Ping: testing ping reachability
h1r1 -> h1r2 h1r3 h1r4 h2r1 h2r2 h2r3 h2r4 h3r1 h3r2 h3r3 h3r4 h4r1 h4r2 h4r3 h4r4
h1r2 -> h1r1 h1r3 h1r4 h2r1 h2r2 h2r3 h2r4 h3r1 h3r2 h3r3 h3r4 h4r1 h4r2 h4r3 h4r4
h1r3 -> h1r1 h1r2 h1r4 h2r1 h2r2 h2r3 h2r4 h3r1 h3r2 h3r3 h3r4 h4r1 h4r2 h4r3 h4r4
h1r4 -> h1r1 h1r2 h1r3 h2r1 h2r2 h2r3 h2r4 h3r1 h3r2 h3r3 h3r4 h4r1 h4r2 h4r3 h4r4
h2r1 -> h1r1 h1r2 h1r3 h1r4 h2r2 h2r3 h2r4 h3r1 h3r2 h3r3 h3r4 h4r1 h4r2 h4r3 h4r4
h2r2 -> h1r1 h1r2 h1r3 h1r4 h2r1 h2r3 h2r4 h3r1 h3r2 h3r3 h3r4 h4r1 h4r2 h4r3 h4r4
h2r3 -> h1r1 h1r2 h1r3 h1r4 h2r1 h2r2 h2r4 h3r1 h3r2 h3r3 h3r4 h4r1 h4r2 h4r3 h4r4
h2r4 -> h1r1 h1r2 h1r3 h1r4 h2r1 h2r2 h2r3 h3r1 h3r2 h3r3 h3r4 h4r1 h4r2 h4r3 h4r4
h3r1 -> h1r1 h1r2 h1r3 h1r4 h2r1 h2r2 h2r3 h2r4 h3r2 h3r3 h3r4 h4r1 h4r2 h4r3 h4r4
h3r2 -> h1r1 h1r2 h1r3 h1r4 h2r1 h2r2 h2r3 h2r4 h3r1 h3r3 h3r4 h4r1 h4r2 h4r3 h4r4
h3r3 -> h1r1 h1r2 h1r3 h1r4 h2r1 h2r2 h2r3 h2r4 h3r1 h3r2 h3r4 h4r1 h4r2 h4r3 h4r4
h3r4 -> h1r1 h1r2 h1r3 h1r4 h2r1 h2r2 h2r3 h2r4 h3r1 h3r2 h3r3 h4r1 h4r2 h4r3 h4r4
h4r1 -> h1r1 h1r2 h1r3 h1r4 h2r1 h2r2 h2r3 h2r4 h3r1 h3r2 h3r3 h3r4 h4r2 h4r3 h4r4
h4r2 -> h1r1 h1r2 h1r3 h1r4 h2r1 h2r2 h2r3 h2r4 h3r1 h3r2 h3r3 h3r4 h4r1 h4r3 h4r4
h4r3 -> h1r1 h1r2 h1r3 h1r4 h2r1 h2r2 h2r3 h2r4 h3r1 h3r2 h3r3 h3r4 h4r1 h4r2 h4r4
h4r4 -> h1r1 h1r2 h1r3 h1r4 h2r1 h2r2 h2r3 h2r4 h3r1 h3r2 h3r3 h3r4 h4r1 h4r2 h4r3
*** Results: 0% dropped (240/240 received)
mininet> exit
```

How to run the code in Terminal-

1)sudo mn --custom datacenterBasic.py --topo dcbasic --mac --switch ovs --controller remote

2)Now Test pingall

3)If you get the output as

h1r1->X X X X X X X….

h2r2)->X X X X X X X

….

Then You need to type in mininet for five switches-:

1)sh ovs-ofctl dump-flows s1

2)sh ovs-ofctl add-flow s1 "priority=0,action=normal"

3)sh ovs-ofctl dump-flows s1

Repeat 1 2 3 for s1r1,s2r2,s2r3,s2r4.

Then Your Output will be correct.
Libraries used are-
1)Topo-It contains all the functions required for Creating Topology
2)-Irange-It contains some useful additional functions while using a loop.

b) In this question we needed to block some of the links mutually connected.
For that I have a pox controller rather than a remote controller.

Algorithm-(How does code work)

```
Topo.__init__( self )



       h11 = self.addHost( 'h1r1' ,ip='10.0.1.1', mac='00:00:00:00:00:11')
       h21 = self.addHost( 'h2r1',ip='10.0.2.1', mac='00:00:00:00:00:21' )
       h31 = self.addHost( 'h3r1' ,ip='10.0.3.1', mac='00:00:00:00:00:31')
       h41 = self.addHost( 'h4r1' ,ip='10.0.4.1', mac='00:00:00:00:00:41')

       h12 = self.addHost( 'h1r2',ip='10.0.1.2', mac='00:00:00:00:00:12' )
       h22 = self.addHost( 'h2r2',ip='10.0.2.2', mac='00:00:00:00:00:22' )
       h32 = self.addHost( 'h3r2' ,ip='10.0.3.2', mac='00:00:00:00:00:32')
       h42 = self.addHost( 'h4r2' ,ip='10.0.4.2', mac='00:00:00:00:00:42')
       h13 = self.addHost( 'h1r3' ,ip='10.0.1.3', mac='00:00:00:00:00:13')
       h23 = self.addHost( 'h2r3' ,ip='10.0.2.3', mac='00:00:00:00:00:23')
       h33 = self.addHost( 'h3r3' ,ip='10.0.3.3', mac='00:00:00:00:00:33')
       h43 = self.addHost( 'h4r3' ,ip='10.0.4.3', mac='00:00:00:00:00:43')

       h14 = self.addHost( 'h1r4',ip='10.0.1.4', mac='00:00:00:00:00:14' )
       h24 = self.addHost( 'h2r4',ip='10.0.2.4', mac='00:00:00:00:00:24' )
       h34 = self.addHost( 'h3r4' ,ip='10.0.3.4', mac='00:00:00:00:00:34')
       h44 = self.addHost( 'h4r4',ip='10.0.4.4', mac='00:00:00:00:00:44' )
```

In this part of code I have created the 16 hosts required and assigned an ip for each host and also assigned a mac address for each code.Mac will Play a significant role in assigning the rules of mutually blocking.

```
self.addLink( h11, s1 )
       self.addLink( h21, s1 )
       self.addLink( h31, s1 )
       self.addLink( h41, s1 )
```

This code connects each host to its switch

```
net.addLink( h11, s1 )
   net.addLink( h21, s1 )
   net.addLink( h31, s1 )
   net.addLink( h41, s1 )
```

This segment of code adds the link between host and its switch.

Next up we have firewall.py code which use to block connections.

In starting I define some rules which are useful

```
rules = [['00:00:00:00:00:11','00:00:00:00:00:21'],
['00:00:00:00:00:11', '00:00:00:00:00:31'],
['00:00:00:00:00:11','00:00:00:00:00:41'],
['00:00:00:00:00:21','00:00:00:00:00:31'],
['00:00:00:00:00:21','00:00:00:00:00:41'],
['00:00:00:00:00:31','00:00:00:00:00:41'],
```

This is an example where I defined the rules for execution and this is where mac addresses came into play as we define which two to block using their mac addresses.

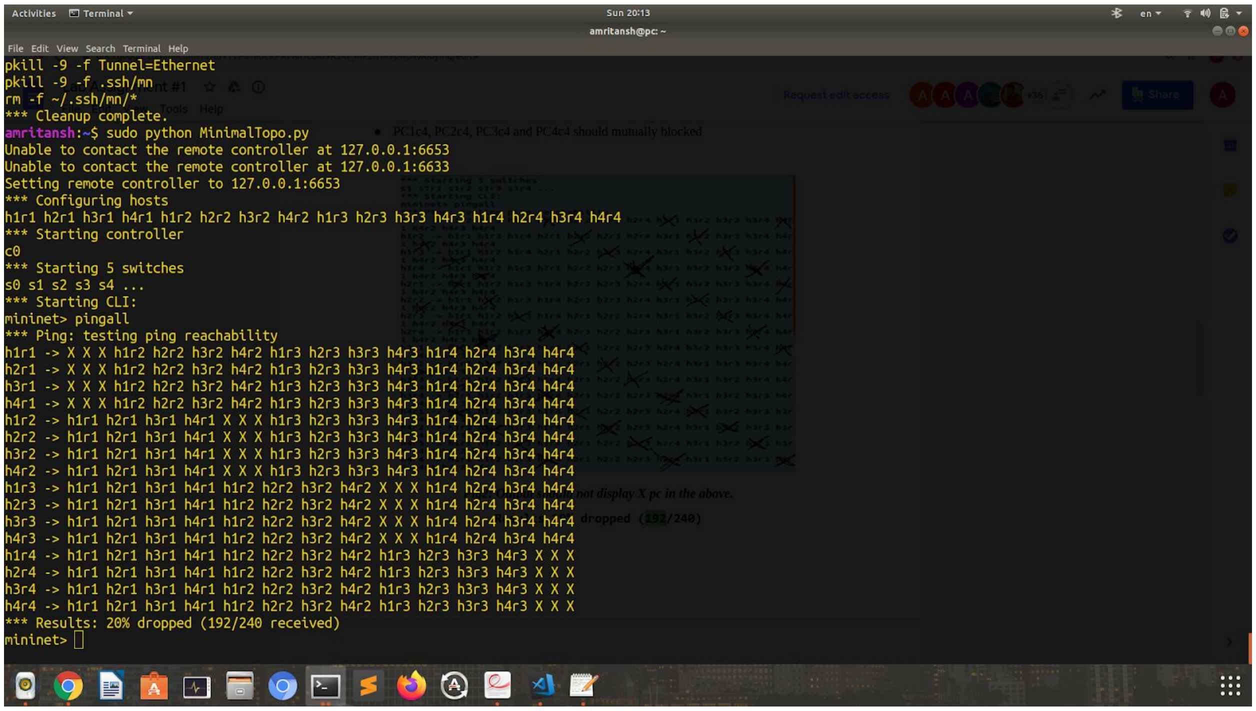
```
class SDNFirewall (EventMixin):

   def _init_ (self):
       self.listenTo(core.openflow)

   def _handle_ConnectionUp (self, event):
       for rule in rules:
           block = of.ofp_match()
           block.dl_src = EthAddr(rule[0])
           block.dl_dst = EthAddr(rule[1])
           flow_mod = of.ofp_flow_mod()
           flow_mod.match = block
           event.connection.send(flow_mod)
```

Next we create our SDNFirewall class in which the actual controller is going to be accessing and checking flows and modifying flow tables accordingly. The first function `__init__` is just a constructor. The second and more interesting function, `_handle_ConnectionUp` will fire up each time a host tries to reach another through the switches. When this happens, we will first iterate over each rule in our list of rules. Here we create match fields by providing the two hosts in the rule, specified by `rule[0]` and `rule[1]` to `block`. We then create an OpenFlow flow_mod message using `of.ofp_flow_mod()` and set its match field to our blocking rule. At the end, we use `event.connection.send(flow_mod)` to send our blocking rule to the switch so that it can be enforced. Lastly, we create the `launch` function that POX requires and pass our `SDNFirewall` class to it.Save the file firewall.py in pox/pox.misc.

How to run the code
1)Open the terminal and run-sudo python MinimalTopo.py(This should open your mininet and you should get the screen like this
2)Now open another terminal and type
$ cd pox
3)Now type
./pox.py log.level --DEBUG openflow.of_01 --port=6653 forwarding.l2_learning pox.misc.firewall
(remember you have to right --port=port_no which your remote controller has got connected to)

4)If no error comes run pingall in first terminal

Output-



```
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
amritansh:~$ sudo python MinimalTopo.py
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
*** Configuring hosts
h1r1 h2r1 h3r1 h4r1 h1r2 h2r2 h3r2 h4r2 h1r3 h2r3 h3r3 h4r3 h1r4 h2r4 h3r4 h4r4
*** Starting controller
c0
*** Starting 5 switches
s0 s1 s2 s3 s4 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1r1 -> X X X h1r2 h2r2 h3r2 h4r2 h1r3 h2r3 h3r3 h4r3 h1r4 h2r4 h3r4 h4r4
h2r1 -> X X X h1r2 h2r2 h3r2 h4r2 h1r3 h2r3 h3r3 h4r3 h1r4 h2r4 h3r4 h4r4
h3r1 -> X X X h1r2 h2r2 h3r2 h4r2 h1r3 h2r3 h3r3 h4r3 h1r4 h2r4 h3r4 h4r4
h4r1 -> X X X h1r2 h2r2 h3r2 h4r2 h1r3 h2r3 h3r3 h4r3 h1r4 h2r4 h3r4 h4r4
h1r2 -> h1r1 h2r1 h3r1 h4r1 X X X h1r3 h2r3 h3r3 h4r3 h1r4 h2r4 h3r4 h4r4
h2r2 -> h1r1 h2r1 h3r1 h4r1 X X X h1r3 h2r3 h3r3 h4r3 h1r4 h2r4 h3r4 h4r4
h3r2 -> h1r1 h2r1 h3r1 h4r1 X X X h1r3 h2r3 h3r3 h4r3 h1r4 h2r4 h3r4 h4r4
h4r2 -> h1r1 h2r1 h3r1 h4r1 X X X h1r3 h2r3 h3r3 h4r3 h1r4 h2r4 h3r4 h4r4
h1r3 -> h1r1 h2r1 h3r1 h4r1 h1r2 h2r2 h3r2 h4r2 X X X h1r4 h2r4 h3r4 h4r4
h2r3 -> h1r1 h2r1 h3r1 h4r1 h1r2 h2r2 h3r2 h4r2 X X X h1r4 h2r4 h3r4 h4r4
h3r3 -> h1r1 h2r1 h3r1 h4r1 h1r2 h2r2 h3r2 h4r2 X X X h1r4 h2r4 h3r4 h4r4
h4r3 -> h1r1 h2r1 h3r1 h4r1 h1r2 h2r2 h3r2 h4r2 X X X h1r4 h2r4 h3r4 h4r4
h1r4 -> h1r1 h2r1 h3r1 h4r1 h1r2 h2r2 h3r2 h4r2 h1r3 h2r3 h3r3 h4r3 X X X
h2r4 -> h1r1 h2r1 h3r1 h4r1 h1r2 h2r2 h3r2 h4r2 h1r3 h2r3 h3r3 h4r3 X X X
h3r4 -> h1r1 h2r1 h3r1 h4r1 h1r2 h2r2 h3r2 h4r2 h1r3 h2r3 h3r3 h4r3 X X X
h4r4 -> h1r1 h2r1 h3r1 h4r1 h1r2 h2r2 h3r2 h4r2 h1r3 h2r3 h3r3 h4r3 X X X
*** Results: 20% dropped (192/240 received)
mininet> 
```