Exercise 6 – MN Python API & POX

1. The Python API and Topologies (50P)

Please explain – line by line – what happens in the following code:

```python
def runExp():
    topo = customTopo( n=2 )
    net = Mininet( topo=topo )
    net.start()
    CLI( net )
    net.stop()

class customTopo( Topo ):
    def __init__( self, n, **kwargs ):
        Topo.__init__( self, **kwargs )
        h1, h2 = self.addHost( 'h1' ), self.addHost( 'h2' )
        s1 = self.addSwitch( 's1' )
        for _ in range( n ):
            self.addLink( s1, h1 )
            self.addLink( s1, h2 )

if __name__ == '__main__':
    setLogLevel( 'info' )
    runExp()
```

2. The POX controller (100P)

One of the main features of Mininet is that you can easily port your emulated networks to real-world production networks. To do so, you will need to define a mapping of your production network to an emulated Mininet. This also includes the controllers you use.

In this exercise you will have a look at the POX controller. POX is included in the VM image we are using for this course. What we want to do first, is to bring up a very simple controller implementation that acts like a standard hub.

a. (Optional) You will read the solution on the next page, but think about which parameters you would have to call **mn** with to generate a single switch topology with three hosts that assigns nodes with simplified MAC addresses, runs the Open vSwitch implementation on the switch and will link with an external controller.

For that, we bring up a new mininet (don't for get to clean up before you do this):

```
$ sudo mn --topo single,3 --mac --switch ovsk --controller remote
```

Now, we will bring up the POX controller in a separate terminal:

```
$ ./pox/pox.py log.level --DEBUG misc.of_tutorial
```

b. (10P) What do you observe after connecting the controller?

We will now check if our hub is working correctly.

c. (20P) Open one xterm window for each host and run tcpdump on hosts 1 and 2:

```
# tcpdump -XX -n -i <interface>
```

Then, in the xterm of host 3, try to ping the IP address of h1 or h2. Also try to ping a device that is not reachable. Please explain what you observe (Hint: if you recall your basic lecture on computer networks, you should know that a hub is generally a dumb device.)

As one main part of this exercise, you will now change the POX controller so that it operates as a learning switch instead of a hub. Stop your current hub controller and your current emulated network. Clean up appropriately.

d. (Optional) Recap how a learning switch is operating.

e. (70P) Open up the file **`pox/pox/misc/of_tutorial.py`** in the editor of your choice. In the Python code you will see that we are currently operating our controller with the help of the **`act_like_hub()`** method. Your task in this exercise is to

i. (10P) modify the controller to use **`act_like_switch()`** instead.

ii. (40P) implement **`act_like_switch()`** so that your controller actually acts like a switch. There are some helpful hints in the code.

iii. (20P) It is somewhat inefficient to let the controller decide the fate of every single packet. Implement an **act_like_flow_switch()** method in which your controller instead installs flow-rules into the switch for all packets that are initiually flow-table misses. This will improve the performance of the forwarding on subsequent packets. You can use the POX API and the POX documentation regarding OpenFlow here (hint: have a very close look at **ofp_flow_mod**).

3. POX Extension (50P)

(45P) Extend your POX controller so that it also acts as a traffic monitor when acting as a flow learning switch (act_like_flow_switch()). The monitor should simply count all the traffic in bytes that is going to or coming from host h1.

(5P) Why is it not sufficient to count traffic in handle_packet_in()?

SUBMISSION: Submit your Python code alongside your text answers.