Assignment 2: Building your first network application

In the second assignment, we will start using POX, a controller that is compatible with the OpenFlow protocol version 1.0. You will learn basic concepts to develop and implement network applications. Again, we suggest you test your implementation in your Mininet installation before submitting it.

POX is included in the Mininet virtual machine provided to you. With the following command you can verify the presence of POX:

```
mininet@mininet-vm:~$ ls -al
drwxrwxr-x 17 mininet mininet 4096 Mar 21  2017 loxigen
drwxrwxr-x 13 mininet mininet 4096 Mar 21  2017 mininet
drwxrwxr-x 14 mininet mininet 4096 Mar 21  2017 oflops
drwxrwxr-x 11 mininet mininet 4096 Mar 21  2017 oftest
drwxrwxr-x 19 mininet mininet 4096 Mar 21  2017 openflow
drwxrwxr-x  7 mininet mininet 4096 Mar 21  2017 pox
```

In this exercise, we will use Mininet to emulate the devices that keep an OpenFlow interface with POX. For this reason, it is recommended to kill all processes associated with other controllers and Mininet:

```
mininet@mininet-vm:~$ sudo killall controller
mininet@mininet-vm:~$ sudo mn -c
```

Now we can start POX in another window, this application is in the folder with the same name:

```
mininet@mininet-vm~/pox$./pox/pox.py       log.level       --DEBUG
forwarding.hub

POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:forwarding.hub:Hub running.
```

With the above command, we start POX and enable notifications. In addition, we run a network application located in the folder */pox/forwarding*.

```
mininet@mininet-vm:~$ pwd
/home/mininet
```

The POX application is in the virtual machine folder located at the path */home/mininet/pox* and inside this are several pre-installed applications:

```
mininet@mininet-vm:~$ ls -al pox
drwxrwxr-x 15 mininet mininet  4096 May 17 11:05 pox
-rwxrwxr-x  1 mininet mininet  1287 Mar 21  2017 pox.py

mininet@mininet-vm:~$ ls -al pox/pox/forwarding/
-rw-rw-r--  1 mininet mininet  1092 Mar 21  2017 hub.py
```

To begin to familiarize with POX, we will use a simple topology, consisting of a controller, an OpenFlow switch, and three hosts:

```
mininet@mininet-vm:~$ sudo mn --topo single,3 --mac --switch ovsk
--controller remote
```

The above command uses the **--mac** option so that the MAC addresses assigned to the hosts are assigned sequentially; host h1 will have the MAC 00:00:00:00:00:01 and host h2 will have the MAC 00:00:00:00:00:02. The **--switch ovsk** option will cause the topology network devices to run the OpenvSwitch operating system and, finally, the **--controller remote** option forces to create a connection with an external controller.

After starting POX, together with Mininet, we can see how the switches are connected:

```
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Oct 26 2016 20:30:19)
DEBUG:core:Platform                                              is
Linux-4.2.0-27-generic-x86_64-with-Ubuntu-14.04-trusty
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-02 3] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-02
INFO:openflow.of_01:[00-00-00-00-00-01 4] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
```

```
INFO:openflow.of_01:[00-00-00-00-00-03 1] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-03
INFO:openflow.of_01:[00-00-00-00-00-04 2] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-04
```

In case we need Mininet devices to start connections to a particular port on which the POX application is receiving connections, we can add the parameter:

```
mininet@mininet-vm:~$ sudo mn --topo single,3 --mac --switch ovsk
--controller remote,port=6633
```

We can verify the hub behavior by opening three consoles, where each one corresponds to a host in the topology:

```
mininet> xterm h1 h2 h3
```

Using the tcpdump command on two of the consoles, we can check the messages arriving at the hosts.

To perform a capture, in the h1 console we use the tcpdump command on the host interface:

```
root@mininet-vm: tcpdump -xx -n -i h1-eth0
```

In h2 we execute the same:

```
root@mininet-vm: tcpdump -xx -n -i h2-eth0
```

Finally, in h3 we start a ping to h2 and to a host that is not created

```
root@mininet-vm: ping 10.0.0.2
root@mininet-vm: ping 10.0.0.5
```

It is important that you analyze the messages that h1 receives to understand the hub behavior. Does it receive messages that are not addressed to it?

POX API handling

The POX API (POX Wiki - Open Networking Lab) exposes some methods and classes that are useful to design different network applications:

- The **connection.send(msg)** method sends an OpenFlow message to a switch.
- The **ofp_action_output** class allows you to specify the switch port through which you want to send a packet. This also accepts special parameters such as **OFPP_FLOOD**, which tells a switch that it must send a message on all its ports except the port on which it was received. For example, the instruction **acc_output = of.ofp_action_output(port = of.OFPP_FLOOD)** creates an action that sends packets to all ports.
- The **ofp_match** class allows finding matches with the fields in the header of a packet and/or input ports of a packet. Unspecified aspects will accept any value. For example, an instantiated object like this: **coinc = of.ofp_match()**, can be modified to match with the packets which MAC source is 00:00:00:00:00:01 like this: **coinc.dl_src = 00:00:00:00:00:01**.
- The **ofp_packet_out** class tells a switch how to send a packet. This is done by the controller through an OpenFlow message.
- **ofp_flow_mod** Instructs a switch to install an entry in the flow table. For example, a rule that forwards packets received on port 1 through port 2 can be created as follows:

  nf = of.ofp_flow_mod()
  nf.coinc.in_port = 1
  nf.actions.append(of.ofp_action_output(port = 2))

We suggest you review in detail the POX API and experiment with the script of_sw_tutorial.py, where you can find different implementations of hubs and switches in POX. You can perform traces on the hosts and check the entries installed in the switch tables to understand the operation of each application. In the launch() method you must choose one of the functions defined in the script, commenting out the rest:

```
def launch ():
    core.openflow.addListenerByName("PacketIn",_handle_dumbhub_p
    acketin)
```

```
#core.openflow.addListenerByName("PacketIn",_handle_pairhub_
packetin)
#core.openflow.addListenerByName("PacketIn",_handle_lazyhub_
packetin)
#core.openflow.addListenerByName("PacketIn",_handle_badswitc
h_packetin)
#core.openflow.addListenerByName("PacketIn",_handle_pairswit
ch_packetin)
#core.openflow.addListenerByName("PacketIn"_handle_idealpair
switch_packetin)
```

Activity

To test your knowledge, you must implement a Firewall. A firewall is a network security equipment that allows or denies outgoing or incoming traffic to a network segment. It does this by analyzing different parameters of the link, network, or transport layer headers of a packet. In this case, you must develop a firewall that filters packets according to their source or destination MAC address. For this purpose, we will provide you with a list of MAC pairs. When a connection between a switch and the controller is established, the application must install the necessary rules to disable the communication between each MAC pair in the file.

Your Firewall should work in any topology. To make it simpler, you can implement rules on all switches in the network.

POX allows several applications to run in parallel, for example, in a switch you can run MAC learning together with the firewall, but you must be careful when installing rules that can generate conflicts between them (for example, two rules that have the same source and destination MAC address that perform different actions), in this case, an alternative is to assign priority levels to each application.

To test your application, you can copy your Python script and MAC list to the directory with the path ~/pox/pox/misc. After that run the driver with the l2_learning application (conventional switch) at the same time with your Firewall application as follows:

```
mininet@mininet-vm~/pox$ ./pox.py forwarding.l2_learning
misc.firewall
```

Thus, the controller will install the rules on the switches for traditional frame switching and the rules associated with your application.

Then, you can run any topology in Mininet to test your code. If the topology that you are using has closed loops, you must use an application that can eliminate them and build a spanning tree, POX includes an application that builds a spanning tree and you can use it, we invite you to consult on this topic. Another option is to use the **--arp** option when you run the topology in Mininet, this makes the hosts start with an ARP table with the entries corresponding to the other hosts in the topology. Your application must work in any topology, independently of the network devices and their connections, you must install the necessary rules to block the MACs included in the file.

To complete the practice, take the skeleton provided on the platform and include the portion of code that you think is appropriate to complete the activity and after you've checked it works, submit it.