A short walk-through of Mininet and POX

This tutorial has three parts. The first part covers the basics of the Mininet network emulation environment under which your programming assignment will be carried out. The second part covers the basics of the POX controller that you need to program. The third part provides some guidelines for you to do your programming assignment.

Part 1: Project Environment Set Up

1. Setup Mininet

The easiest way to get started is to use **a pre-packaged Mininet/Ubuntu Virtual Machine (VM)**. This VM contains the Mininet itself, all OpenFlow binaries and controllers, and tweaks to the kernel configuration to support larger Mininet networks.

Download Virtualization System

You can use any virtualization system of your choice, but we recommend installing VirtualBox. It's free and runs on Windows, Linux and OS X.
VirtualBox (Recommended): https://www.virtualbox.org/wiki/Downloads
For other virtualizations systems, visit here.

Download Mininet VM

We recommend using the latest Mininet image, which is in the package file we provide. You can also go to www.mininet.org for more information. It comes with the latest version of Mininet and two OpenFlow Controllers (POX and Pyretic). The download will take some time. It's ~ 800MB, compressed.

Setup Virtual Machine

- **Start VirtualBox**
- **Select File>Import Appliance and select the .ova file**
- **Press the "Import" button.**
    - This will unpack and import the VM in your local machine. It will take a while, as the unpacked image is about 3 GB.

Boot VM

Now, you are ready to start your VM. Press the "Start" arrow icon or double-click your VM within the VirtualBox window.
In the VM console window, log in with the user name and password for your VM. The username and password for this VM are:

- User name - **mininet**
- Password - **mininet**

Note that this user is a sudoer, so you can execute commands with root permissions by typing *sudo command*, where *command* is the command you wish to execute with root permission.

2. Setup POX and POXDesk

POX should be installed and run on another Virtual Machine. You need to download and install an Ubuntu system (we recommend version 14.04) on that machine. POXDesk is an extensible web-based GUI for POX, which makes it more convenient to monitor the status of the network and switches. This is an optional add-on, you can decide whether to use it. It helps you quickly visualize the topology you create.

Download POX

Go to the root directory and pull the POX repository:

```
git clone https://github.com/noxrepo/pox
```

Download and Install POXDesk

Follow these commands to install POXDesk:

```
cd pox/ext
git clone https://github.com/MurphyMcAfee/poxdesk
cd poxdesk
wget http://downloads.sourceforge.net/qooxdoo/qooxdoo-2.0.2-sdk.zip
unzip qooxdoo-2.0.2-sdk.zip
mv qooxdoo-2.0.2-sdk qx
cd poxdesk
./generate.py
```

When mininet is started and switches are connected with the POX controller, you can access the manage page of POXDesk at:

```
127.0.0.1:8000/poxdesk
```

3. Mininet

Mininet is a powerful network emulation tool. It creates a realistic virtual network, running real kernel, switch and application code on a single machine. You can easily interact with your network using Command Line Interface(CLI). Notice that to test connectivity, you need to run a POX controller that has switch functionality.

Mininet Basics

Display Mininet Command Line Interface (CLI) commands:

```
mininet> help
```

Display nodes:

```
mininet> nodes
```

If the first string of the CLI command is a host, switch or controller name, the command is executed on that node. For instance, to show the interface of host h1:

```
mininet> h1 ifconfig
```

Test connectivity between hosts. For example, test the connectivity between h1 and h2:

```
mininet> h1 ping h2
```

Alternatively, you can test the connectivity between all hosts by typing:

```
mininet> pingall
```

Exit Mininet:

```
mininet> exit
```

Clean up:

```
$ sudo mn –c
```

Mininet Python API

A more powerful way to construct the network topology is to use Mininet Python APIs. ‘mininet.topo.Topo’ is the class that defines the network topology. We can define a child class of ‘mininet.topo.Topo’ and initialize the network topology with the help of the class methods.

addHost(name, opts): add a host to the network with the name being ‘name’ parameter. Four our case, hosts are named as h1, h2, h3, and h4. ‘opts’ is a dictionary that includes the parameters for the host.

addSwitch(name, opts): add an Openflow vSwitch to the network with the name being ‘name’ parameter. For our case, switches are named as s1, s2, s3, and s4. ‘opts’ is a dictionary that includes the parameters for the switch.

addLink(node1, node2, port1, port2, opts): add a bidirectional link between port1 of node1 and port2 of nodes. ‘opts’ is a dictionary that includes the parameters for the link. ‘bw’ is one of the parameters that stands for bandwidth, and unit is Mbps.

Mininet(topo, link, controller) is the initialization function of the Mininet network emulation environment. Using the defined topology, we can set the remote controller and the communication protocol between data plain and control plain.

More information can be found here.

Create QoS Queue

For Task 4, you need QoS queues with different bandwidths to implement the VPN service. **ovs-vsctl** command can be used create several queues at a certain (virtual or real) network interface. As an option, you can set the maximum and minimum speed for each queue. Here is an example:

sudo ovs-vsctl – set Port eth0 qos=@newqos -- --id=@newqos create QoS type=linux-htb other-config:max-rate=1000000 queues=0=@q0,1=@q1 -- --id=@q0 create Queue other-config:max-rate=600000 other-config:min-rate=600000 -- --id=@q1 create Queue other-config:max-rate=400000 other-config:min-rate=200000

It means setting up a QoS service **newqos** at interface **eth0** with a maximum rate of 1 Mb/s, and creating 2 queues **q0** and **q1** at **eth0** with corresponding maximum and minimum speed. Thus the controller can insert routing entries in the switch to send packets to queue **q0** or **q1** for different requirements. More information can be found here.

2. Connection between Mininet and POX

A requirement for this system to work is that the Mininet network can communicate with the POX controller. This is a brief introduction for the setup of VM.

Select your VM and go to the Settings Tab. Go to Network -> Adapter 1. Select the “Enable adapter” box, and attach it to “host-only Adapter”. Make sure that your VM is turned off. Otherwise VirtualBox will not allow you to make these changes. This will connect the guest VM that runs Mininet to the same internal network with your host machine. You can examine

whether the VM and your host machine are in the same network by checking the IP addresses with **ifconfig** command.

Part 2: Basics of the POX Controller

3. The POX Controller

After setting up the network environment, a controller is needed to install flow entries to the switch so that packets can be forwarded among hosts. Go to the directory of pox in the POX VM:

$cd ~/pox

Then, start a simple hub application:

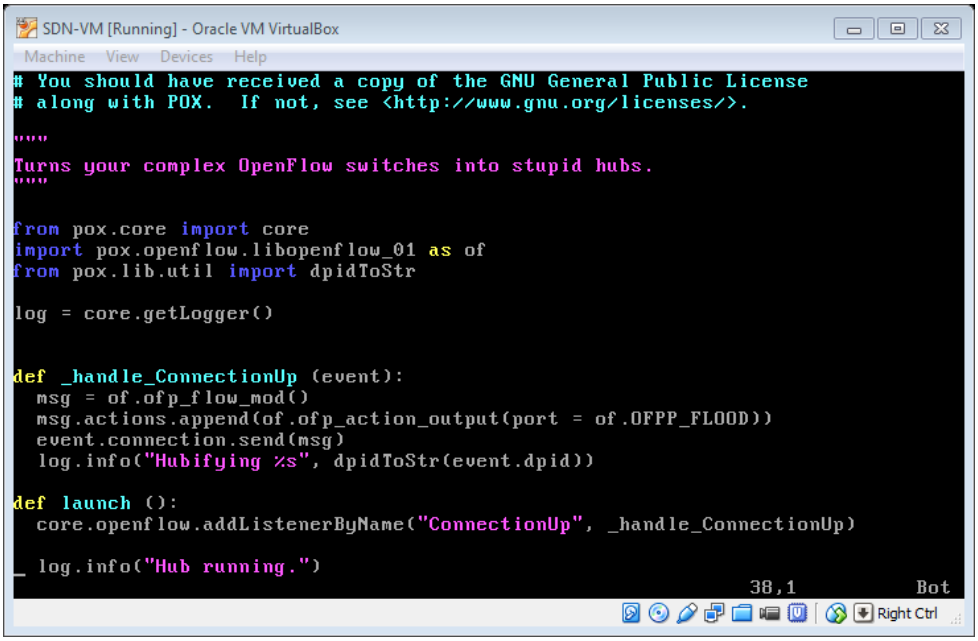
$ pox.py log.level –DEBUG pox.forwarding.hub

Notice that the source code of the hub example is located in the directory:

~/pox/pox/forwarding

Then you can verify the connectivity of each host by either using pingall command.

Now let's have a look at the hub code:



In general a POX controller consists of three parts:

1. Listener
2. Control logic
3. Messenger

First you need to figure out the type of the event you want the controller to listen to (e.g., ConnectionUp, PacketIn, etc).

Then using some logic you can distinguish between different flows and attach a proper action for a specific flow.

Finally you send the message to the switch to add the new rule in the Openflow table.

Hint: A sample level-2 learning switch 'switch_l2.py' might be helpful to grasp how POX works.

Part 3: Testing your code

Once you have your code ready, you can start testing the applications. Use Task 1~3 as an example. In the host machine that runs POX controller, copy **controller.py** (as you name) and **polocy.in** to ~/pox/pox/misc directory. In the Mininet VM, copy **mininetTopo.py** (as you name) and **topology.in** to some directory.

Run POX controller:

```
$ cd ~/pox

$ ./pox.py pox.misc.controller
```

In the Mininet VM, use the IP address of the host machine for the Mininet initialization function Mininet() in treeTopo.py. Run the network topology:

```
$ sudo python mininetTopo.py
```

First, you can check whether your self-learning switch is working. In mininet, try `pingall`, you should see all hosts can connect to each other.

Second, you can verify whether you have successfully installed the firewall application. For example, you can test whether the TCP traffic between `h2` and `h4` on port 4001 is correctly blocked:

```
mininet > h4 iperf –s –p 4001 &
```

```
mininet > h4 iperf –s –p 4000 &
mininet > h2 iperf –c h4 –p 4001 –t 5
```

The result should show that h2 cannot connect to the server.

```
mininet > h2 iperf –c h4 –p 4000 –t 5
```

The result should show the throughput as specified in “**topology.in**”.

Appendix A: Iperf

Iperf is a tool to measure the bandwidth and the quality of a network link.

Usage:

On the server side (h2 (10.0.0.2) for example):

```
# iperf –s -p 4001
```

will initiate a tcp server listening on port 4001

On the client side (h1 (10.0.0.1) for example):

```
# iperf –c 10.0.0.2 -p 4001
```

will initiate a tcp client that connects to host 10.0.0.2 on port 4001, and performs measurement of the link.

More examples are available on: http://openmaniak.com/iperf.php

More references to read:

POX references:

- https://openflow.stanford.edu/display/ONL/POX+Wiki (recommend you read this if you want to fully understand POX)
- http://archive.openflow.org/wk/index.php/OpenFlow_Tutorial#Sending_OpenFlow_messages_with_POX (very easy document to follow)
- http://sdnhub.org/tutorials/pox/

Mininet References:

- http://mininet.org/
- https://github.com/mininet/mininet/wiki/Introduction-to-Mininet