Assignment 3 – Mininet (Control plane)

Instructions:

Please complete this assignment by individuals. Both graduate and undergraduate students are expected to complete this programming assignment. This assignment is designed to help you learn how to program the OpenFlow controller POX to implement several network applications. Through this project, you will learn to implement applications such as firewall and premium service class for this network.

Before starting this assignment, please make sure that you have successfully set up the Mininet VM. POX is pre-installed in the VM.

For M1 users:

At the end, we were able to get Google Cloud Platform for our course. Those of you who have M1 chip Mac, should email me to get the credits and also send me your Google account so that I can share with the image. I refer you to the file I have uploaded under Modules => Week 5. Please, do not hesitate to contact me if you have any questions.

Task 1 - Building a virtual network (20 points):

In this section, your task is to build a virtual network in the Mininet network emulation environment. The network topology is given in an input file. Below is an example of a network topology, consisting of 7 **hosts**, 4 **switches** and 11 **links** that connect them together, as shown in Figure 1. In this network, the 7 hosts (with IP addresses from 10.0.0.1 to 10.0.0.7) are connected with 4 switches (with dpid from 1 to 4) by 7 links. The 4 switches are then connected together by 4 links. The capacity of each link is also shown in the figure.

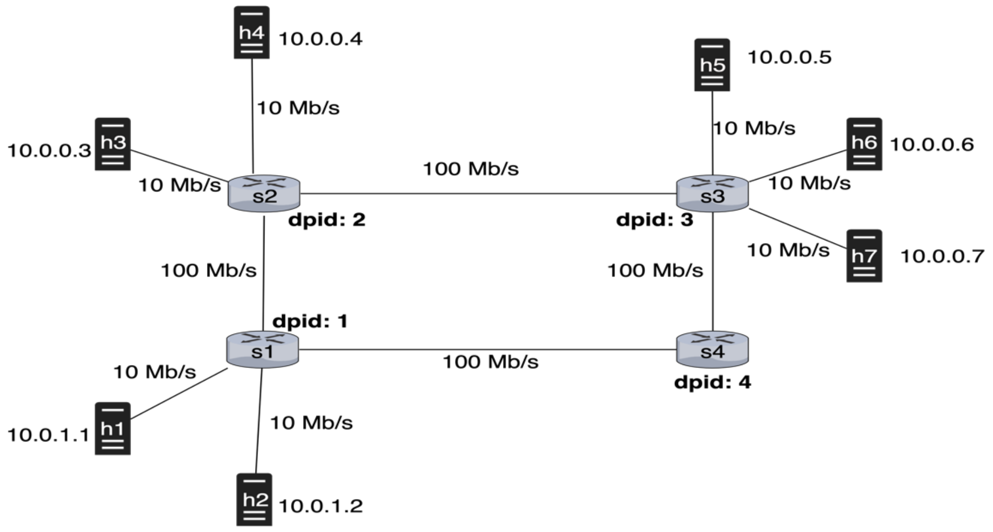

Figure 1. A sample network topology

You need to build a network with the topology and specifications described in the input file **topology.in**. The first line of the file has 3 integers N and M, L indicating that there are N hosts (h1,h2,...,hN), M switches ($s1$,$s2$,...,sM) and L links in the network. The following are L lines of tuples in the form of $< dev1, dev2, bw >$ that describe the links, meaning that $dev1$ and $dev2$ are connected via a bi-directional link of capacity bw Mb/s at both directions. A partial input file for the network in Figure 1 looks like follows:

```
7 4 11

h1,s1,10

h2,s1,10

h3,s2,10

h4,s2,10

...
```

The sample input file **topology.in** for the network in Figure 1 will be provided for your reference. You will also get a template of script written in Python, based on which you can add your own code to create the hosts, switches and links in Mininet.

Task 2 – Firewall (25 points):

In this section, you will implement a layer-3 firewall application using the POX controller. The application needs to block 2 types of traffic flows: 1) the **TCP** traffic sent to a certain **host** on a certain **port**, and 2) the **TCP** traffic originated from a certain **host** to another **host** on a certain **port**. You will be provided with an input file including a number of lines in the form of:

```
10.0.0.4,4001

10.0.0.2,10.0.0.5,1000
```

the **TCP** traffic sent to host h4 (10.0.0.4) on port 4001 and the **TCP** traffic from host h2 (10.0.0.2) to host h5 (10.0.0.5) on port 1000 should be blocked.

The basic idea is that when a connection between the switch and the controller is up, the application installs flow entries that **drop** all the **TCP** packets satisfying at least one of two types of rules described in the input file.

Task 3 - Premium traffic (35 points):

In this part, your task is to implement a premium class of traffic for certain hosts to receive higher bandwidth. Sometimes, hosts may desire higher received bandwidth in order to guarantee the quality of its tasks. Such hosts can pay extra for the premium class of traffic which guarantees at least X Mb/s of bandwidth, while the others stay in the normal class, for which the received bandwidth is limited to at most Y Mb/s. In our topology, assume a link between a host and a switch has the capacity of bw Mb/s, then X is set to be $0.8 \times bw$, while Y is $0.5 \times bw$. For simplicity, we also assume that links between switches will never become the bottleneck.

For the two tasks above, you will be provided with a file **policy.in** that describes the policies for the firewall and premium traffic. The file starts with a line of two integers N and M, indicating that there are N different firewall rules and M hosts that have paid for the premium class of traffic. Each firewall policy is expressed in either: 1) $<$ dst_ip, $p >$ or, 2) $< src\_ip, dst\_ip, p >$, followed by M lines listing the hosts. An example of the file is shown as follows:

```
2 3
10.0.0.4,4001

10.0.0.2,10.0.0.5,1000

10.0.0.1
10.0.0.3
10.0.0.7
```

The above example shows that given the topology shown in Figure 1, the first line indicates there are 2 firewall rules and 3 hosts have paid for the premium class of traffic. h1 (10.0.0.1), h3 (10.0.0.3) and h7 (10.0.0.7) are guaranteed to receive at least 8 Mb/s bandwidth, while the received bandwidth of other hosts is upper-bounded by 5 Mb/s.

The key idea of this application is to set up queues with different bandwidths at the interfaces of the switches using the **ovs-vsctl** tool that configures the Open vSwitch. Basically, **ovs-vsctl** helps to set up QoS queues in the Mininet virtual networks. When a packet arrives, the controller checks whether it belongs to premium or normal traffic, and then sends the packet to the corresponding queues.

Note that when you want to set queues to maintain a specific bandwidth, you should avoid adding the bandwidth value while initializing the link. (e.g. self.addLink(host, switch))

Handling conflicts:

The POX controller supports multiple concurrent applications. However, it is possible that one application will be in conflict with another application. In this assignment, the premium traffic application should not allow communication which is blocked by the firewall application. You are to make sure that no conflicting rules are installed by two different applications.

OpenFlow message structure ofp_flow_mod has an attribute called priority. Conflicts can be handled by setting different priority values to different rules.

Problem in POX's Default Configuration:

When you need to **flood** packets in POX, remember to modify the attribute port to ofp.OFPP_ALL (which is by default ofp.OFPP_FLOOD) to avoid the situation that the packet is not actually flooded.

Note: If you get the error below which is complaining that another program is listening to the Pox port, it means that probably you have not stopped the pox program correctly.

```
ERROR:openflow.of_01:Error 98 while binding socket: Address already in use
```

You need to run the following commands to terminate the other program:

```
sudo mn -c
sudo fuser -k 6633/tcp
```

Understanding Provided Files:

mininetTopo.py	The script template for creating the required network topology.
controller.py	A skeleton for the controller.
topology.in	A sample input file for building the virtual network. Your program should be able to read another given input file.
policy.in	A sample input file for firewall policies and premium traffic. Your program should be able to read another given input file.

Questions:

Task 1 (5 points):

Your program should be able to run by the command, "sudo python mininetTopo.py". And we expect to see that the right number of devices are launched. (Please print the nodes at the end of your program.)

Task 2 (11 points):

Please run the following commands in shell and add a screenshot of the results.

```
h4 iperf -s -p 4001
h5 iperf -c h4 -p 4001
h1 iperf -c h4 -p 4001
h4 iperf -s -p 8080
h5 iperf -c h4 -p 8080
h5 iperf -s -p 1000
h2 iperf -c h5 -p 1000
h1 iperf -c h5 -p 1000
h7 iperf -c h5 -p 1000
h5 iperf -s -p 8080
h2 iperf -c h5 -p 8080
```

Task 3 (4 points):

Please run the following commands in shell and add a screenshot of the results.

```
iperf h1 h3
iperf h1 h2
iperf h2 h1
iperf h2 h5
```

What to turn in:

You need to submit two Python program files, named **mininetTopo.py** and **controller.py**. The first program should take the topology file **topology.in** as input, and create the required virtual network. The second program should take the policy file **policy.in** as input, and implement the firewall and premium traffic applications. In addition, please submit a document with your answers to the questions. Turn them all into a .zip file.

You can submit the assignment through Canvas.

Bonus task – Dijkstra shortest path:

You need to implement the Dijkstra's algorithm using POX controller. Your program should be able to find the shortest paths, and install the required forwarding rules in the switches. You can use the Fattree topology that you created in the previous assignment to test.