CS5313: Computer Networks **Instructor: Dr. Deepak Tosh**
Fall 2020 **Due Date:** 11/15/2020

Assignment # 4: Software Defined Networking

GOAL:

Software-Defined Networking (SDN) is a recently proposed networking paradigm in which the data and control planes are decoupled from one another. One can think of the control plane as being the networks "brain", i.e., it is responsible for making all decisions, for example, how to forward data, while the data plane is what actually moves the data. In traditional networks, both the control- and data planes are tightly integrated and implemented in the forwarding devices that comprise a network. The SDN control plane is implemented by the "controller" and the data plane by "switches". The controller acts as the "brain" of the network and sends commands ("rules") to the switches on how to handle traffic. OpenFlow has emerged as the de facto SDN standard and specifies how the controller and the switches communicate as well as the rules controllers install on switches.

Disclaimer: This assignment can be done in groups of 2 students. You may choose the same partner with whom you did the assignment#2.

Environment Setup:

You need to use the Ubuntu VM that was set in assignment-3. Then, you can install the mininet and pox controller as per the instructions shown in the class.

Task-1: Programming Mininet Topologies (30 pts)

Mininet is also programmable using the python programming language. A set of sample topologies are provide in the "assign4starter.zip". Download and unzip this file in the Ubuntu VM and you'll find two different directories: **topo** and **pox**. Ignore the pox directory for now (it's used in Task-2). In the topo folder there are a variety of python files. Each defines a topology, run the part1 file with command "sudo python assign4starter/topos/part1.py". It will drop you into the CLI with the network topology defined in the python script.

1) Your task here is to modify part1.py to represent the following network topology

```
[h1]-----{s1}------[h2]
[h3]----/    \-----[h4]
```

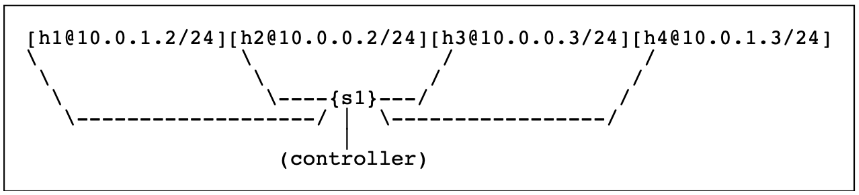
Where [x] means you create a host named x, {y} means a switch named y, and --- means there is a link between the node and the switch.

2) After creating the above architecture, provide the two items in a part1 folder in a compressed file: a) Your modified part1.py file; (b) Screenshots of the iperf, dump, and pingall commands (from mininet) in pdf format.

Task-2: SDN Controller using POX (70 pts)

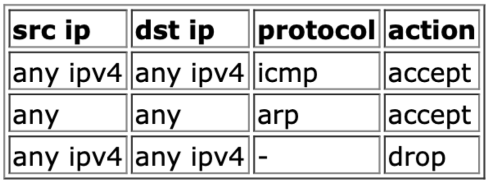
In task 1, you experimented with Mininet using its internal controller. In this (and future) task, you will instead be using a custom controller to send commands to the switches. Here, we will be using the POX controller, which is written in Python. ***For this part, you will create a simple firewall using OpenFlow-enabled switches***. The term "firewall" is derived from building construction: a firewall is a wall you place in buildings to stop a fire from spreading. In the case of networking, it is the act of providing security by not letting specified traffic pass through the firewall. This feature is good for minimizing attack vectors and limiting the network "surface" exposed to attackers.

In this part, you are provided with the Mininet topology, part2.py, to setup your network which assumes a remote controller listening on the default IP address and port number 127.0.0.1:6633. **You do not need to (and should not) modify this file.** The topology that this script will setup is as follows. Note that h1 and h4 are on the same subnet and a different one from h2 and h3.



```
[h1@10.0.1.2/24][h2@10.0.0.2/24][h3@10.0.0.3/24][h4@10.0.1.3/24]
 \                \              /                /
  \                \            /                /
   \                \----{s1}---/                /
    \--------------------/  |  \-----------------/
                            |
                       (controller)
```

For part-2, you are also provided with a skeleton POX controller (under pox folder): “part2controller.py”. **This file is required to be modified to create the firewall** that implements the following rules.



src ip	dst ip	protocol	action
any ipv4	any ipv4	icmp	accept
any	any	arp	accept
any ipv4	any ipv4	-	drop

Then, you need to deploy this new controller and run mininet under the new topology provided in “part2.py”.

Note: Basically, your Firewall should allow all ARP and ICMP traffic to pass. However, any other type of traffic should be dropped. It is acceptable to flood the allowable traffic out all ports. Be careful! Flow tables match the rule with highest priority first, where priority is established based on the order rules are placed in the table. When you create a rule in the POX controller, you need to also have POX "install" the rule in the switch. This makes it so the switch "remembers" what to do for a few seconds. **Do not handle each packet individually inside of the controller! Hint: To do this, look up ofp_flow_mod.** The OpenFlow tutorial (specifically " Sending OpenFlow messages with POX") and the POX Wiki are both useful resources for understanding how to use POX.

Deliverables:

1. Your part2controller.py file
2. Documentation on how you deployed the controller and how you run mininet with new topology.

3. A screenshot of the ***pingall*** command. Note that h1 and h4 should be able to ping each other (h2 and h3 as well), but not across subnets. Also, the ***iperf*** command should fail (as you're blocking IP traffic). This is realized as the command hanging.
4. A screenshot of the output of the ***dpctl dump-flows*** command. This should contain all of the rules you've inserted into your switch.

Task-3: Firewall in real network (Bonus 25 pts)

In task 2 you have implemented a simple firewall that allowed ICMP and ARP packets but blocked all other packets. For your part 3, you will be expanding on this to implement routing between subnets and implementing firewalls for certain subnets. The idea is to simulate an actual production network.

We will be simulating a network for a small company (topology is shown in the picture below). Assume that the company has a 3-floor building, with each floor having its own switch and subnet. Additionally, we have a switch and subnet for all the servers in the data center, and a core switch connecting everything together. Note that the names and IPs are not to be changed. You are provided the topology "./topos/part3.py" as well as a skeleton controller (./pox/part3controller.py). As with part 2, you need only modify the controller only.

```
[h10@10.0.1.10/24]--{s1}--\
[h20@10.0.2.20/24]--{s2}--{cores21}--{dcs31}--[serv1@10.0.4.10/24]
[h30@10.0.3.30/24]--{s3}--/     |
                                |
                    [hnotrust1@172.16.10.100/24
```

Your goal will be to allow traffic to be transmitted between all the hosts. In this part, you will be allowed to flood traffic on the secondary routers (s1, s2, s3, dcs31) in the same method that you did in part-2 (using a destination port of ***of.OFPP_FLOOD***). However, for the core router (cores21) you will need to specify specific ports for all IP traffic. You may find it easiest to determine the correct destination port by using the destination IP address and source IP address, as well as the source port on the switch that the packet originated from. Additionally, to protect our servers from the untrusted Internet, we will be blocking all IP traffic from the Untrusted Host to Server 1. To block the Internet from discovering our internal IP addresses, we will also block all ICMP traffic from the Untrusted Host. In summary of your goals:

- Create a Pox controller (as per task-2) with the following features: All nodes able to communicate **EXCEPT**
  - hnotrust1 cannot send ICMP traffic to h10, h20, h30, or serv1.
  - hnotrust1 cannot send any IP traffic to serv1.

Deliverables:

1. Your part3controller.py file.
2. A screenshot of the ***pingall*** command. All nodes but hnotrust should be able to send and respond to pings.
3. A screenshot of the ***iperf hnotrust1 h10*** and ***iperf h10 serv1*** commands. Though not shown in these commands, hnotrust should not be able to transfer to serv1. It should be able to transfer to other hosts.

A screenshot of the output of the ***dpctl dump-flows*** command. This should contain all of the rules you've inserted into your switches.

What to be submitted:

1. A **PDF formatted report** with all required answers, and necessary evidences as asked above including codes, execution samples, instructions for running the programs, and references used. The filename should be formatted as "*Lastname_firstname_Assign4.pdf*".
2. Well-documented source codes and *README.txt* in the form of **one single Zip** file.
3. Contribution details of both members of the team, if this assignment is done is in teams.

Submission:

Submit your **assignment** on **Blackboard only.**