Lab 4: Simple Firewall using OpenFlow

This lab is a continuation of Lab 1 where you were first introduced to the Mininet environment. In that lab you played around with some of the basic functionality of the mininet environment, in this lab we will take that one step further by introducing you to the OpenFlow Controller.

Software-Defined Networking & OpenFlow:

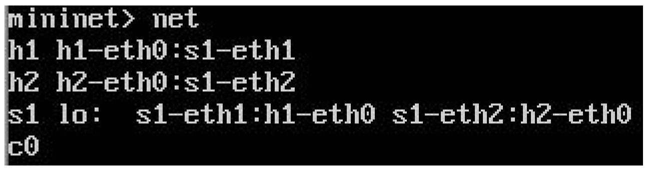
Software-Defined Networking (SDN) is the next big buzzword in computer networking. It started with Ethane[1], then SANE[2], and and OpenFlow[3]. SDN is the act of separating the control plane from the data plane. This has probably not been covered in class to this point, so we will define the control plane as the entity which manages the functionality of how information moves through the data plane. One can think of the control plane as the brains of a device and the data plane as the hands. What the brain thinks, the hands carry out.

OpenFlow is the first widely accepted implementation of SDN. OpenFlow is the protocol which manages the communication between the control and data planes. In the above analogy, OpenFlow is the nerves that connect the brain to the hands.

Continuing our analogy, the brain is the Controller. The controller uses the OpenFlow protocol in order to control how the switch operates. The first controller was NOX[4], developed by Nicira, which then became open-sourced.

Mininet and OpenFlow:

Mininet is a software emulated network which includes the OpenFlow protocol and switches with OpenFlow enabled. When starting the mininet environment you will see a device in the network called c0, this is the controller. It communicates with the OpenVSwitch (OVS)[5] switches, to modify the forwarding logic.



```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
```

c0: The OpenFlow controller

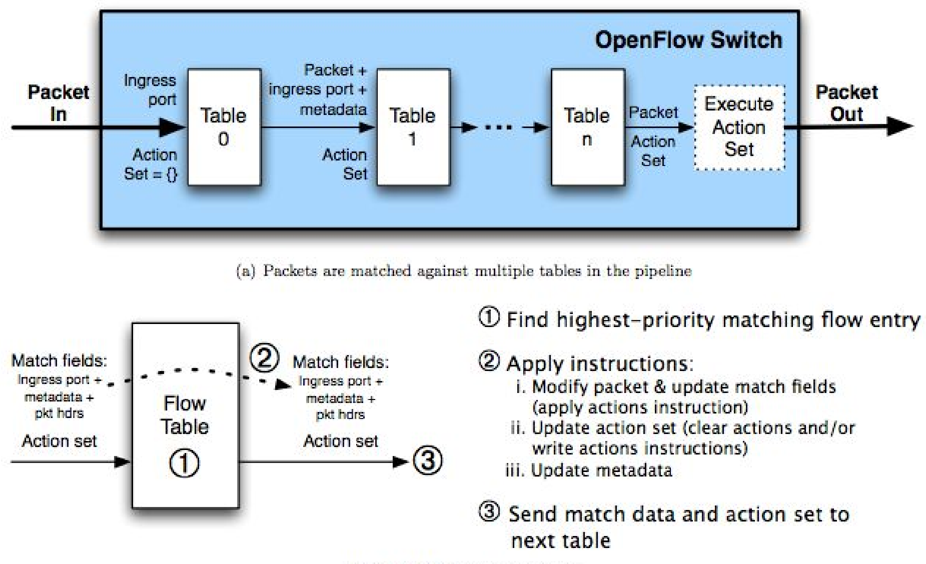
[1] http://klamath.stanford.edu/~nickm/papers/ethane-sigcomm07.pdf
[2] http://tiny-tera.stanford.edu/~nickm/papers/sane.pdf
[3] http://archive.openflow.org/documents/openflow-wp-030108.pdf
[4] http://www.noxrepo.org/
[5] http://openvswitch.org/

In this lab, you will use the POX controller, which uses the Python programming language. **A good tutorial for setting up POX can be found here**. It is **highly** recommended you use that resource to setup your controller and understand how to use the mininet environment with a remote controller.

OpenFlow 1.3:

Use the link provided in the header to view the complete specification for OpenFlow 1.3 if you need additional insight. OpenFlow 1.3 is the current OpenFlow protocol supported within the mininet environment.

OpenFlow Switch Specification Version 1.3.0

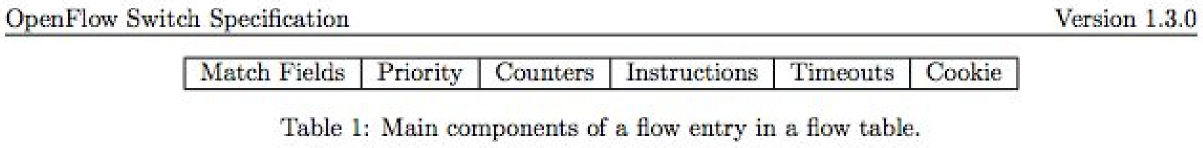

OpenFlow Switch
Packet In
Ingress port
Action Set = {}
Table 0
Packet + ingress port + metadata
Action Set
Table 1
Table n
Packet
Action Set
Execute Action Set
Packet Out
Match fields: Ingress port + metadata + pkt hdrs
Action set
Flow Table ①
② Match fields: Ingress port + metadata + pkt hdrs
Action set
③
① Find highest-priority matching flow entry
② Apply instructions:
i. Modify packet & update match fields (apply actions instruction)
ii. Update action set (clear actions and/or write actions instructions)
iii. Update metadata
③ Send match data and action set to next table

(a) Packets are matched against multiple tables in the pipeline

(b) Per-table packet processing

Figure 2: Packet flow through the processing pipeline

Each message that you send to the switch from the controller will modify the flow table, below is the flow table entry which a switch will act upon.



OpenFlow Switch Specification Version 1.3.0

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Table 1: Main components of a flow entry in a flow table.

You can follow the execution logic based on matching in the figure below. If an ofp_packet_in does not have a match header and it does not have a table-miss flow entry the packet will be dropped. If the packet has a table-miss flow entry it will be forwarded to the controller. If there is a match-entry for the packet, the switch will execute the action stored in the instruction field of the flow table.

5.3 Matching

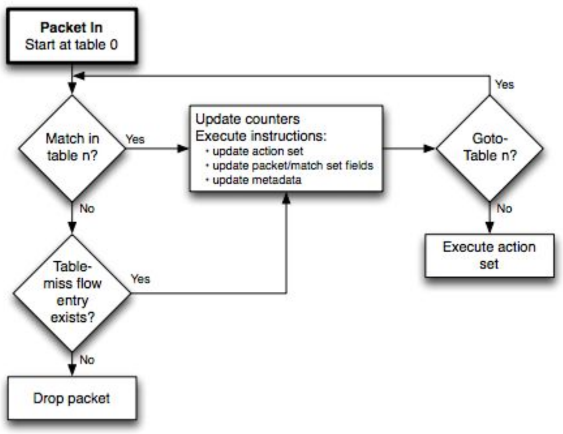
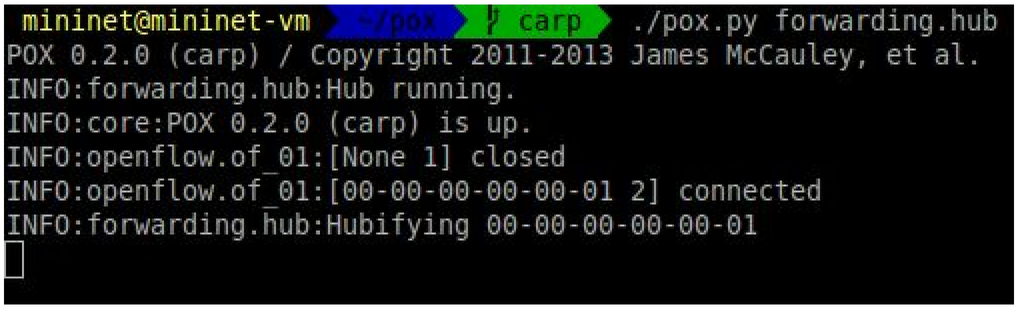


Figure 3: Flowchart detailing packet flow through an OpenFlow switch.

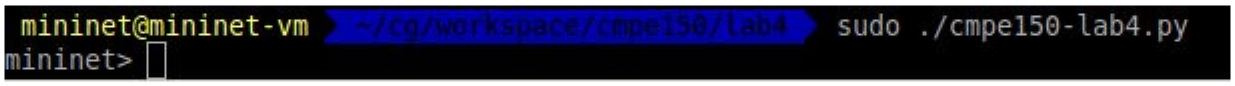
Assignment:

For this assignment you will be creating a simple firewall using OpenFlow and OVS enabled switches. The term Firewall is derived from construction, a firewall is a wall you place between rooms to stop a fire from spreading. In the case of networking it is the act of providing security by not letting specified packets or frames from passing over the firewall. This feature is good for minimizing attack vectors and limiting the surface which you expose to attackers.

You can use the following configuration script to set up mininet: lab4.py
This is assuming a remote controller is listening on the default IP and port (127.0.0.1,6633). Start your controller, and then run the configuration script, you should see:



```
 mininet@mininet-vm  ~/pox  ᚠ carp   ./pox.py forwarding.hub
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:forwarding.hub:Hub running.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[None 1] closed
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
```

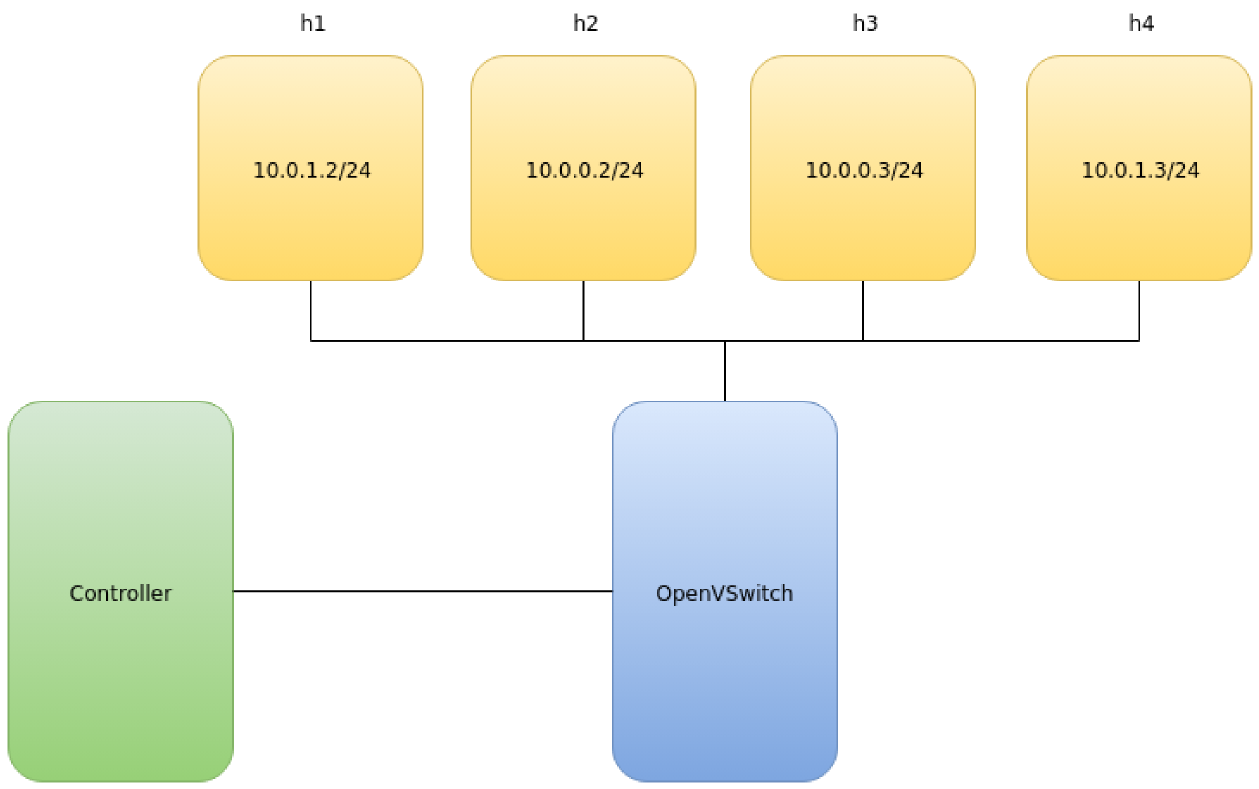


```
 mininet@mininet-vm  ~/cg/workspace/cmpe150/lab4   sudo ./cmpe150-lab4.py
mininet>
```

On the top picture, I started the controller, then on the bottom I ran the lab4 configuration script. The top photo shows that the switch has successfully connected with the controller.

The controller code is what you will need to modify in order to complete this lab.

The network configuration is shown below.



Rules:

The rules that you will need to implement in OpenFlow for this assignment are:

src ip	**dst ip**	**src port**	**dst port**	**action**
any ipv4	any ipv4	any port	any port	accept
any ipv4	any ipv4	-	-	drop

All traffic from an IPv4 host to another IPv4 host without a transport layer header should be dropped. All traffic with an IPv4 source,destination, and transport layer header should be accepted.

Be careful! The flow tables match the first rule with the high priority first, which order the rules are placed in the table matters based on your logic for what packets to accept or to drop.

Not only should the controller handle this logic, but it must install flow table entries on the switch, it is not acceptable to have the controller handle every ofp_packet_in event. To make sure the switch does not handle every packet you must install flow table entries using ofp_flow_mod.

You can test your rules using **ping**[6], **iperf**[7], or **nc**. To verify that the rules are installed you should use **dpctl dump-flows**.

To tell if you have configured it correctly, the following commands should have the following output from the mininet command line:

> pingall = fails completely
> iperf h1 h4 = works

[6] http://linux.die.net/man/8/ping
[7] https://iperf.fr/

Deliverables:

1. **README**: The readme should contain your name, and a description of your program.
2. **Your firewall code**: Should be included in the submission. It too should also contain your name in the comments and it needs to have the name **firewall.py**