Lab 3: Simple Firewall using OpenFlow

This lab builds on the knowledge acquired through Lab 1 where you were first introduced to the Mininet environment. It will also help you prepare for the class project. In Lab 1, you were introduced to some basic functionality of Mininet. In this lab, we will take that one step further by introducing you to Software-Defined Networking (SDN) and the OpenFlow protocol.

Software-Defined Networking & OpenFlow:

Software-Defined Networking (SDN) is a recently proposed networking paradigm in which the data and the control planes are decoupled from one another. One can think of the control plane as being the network's "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.

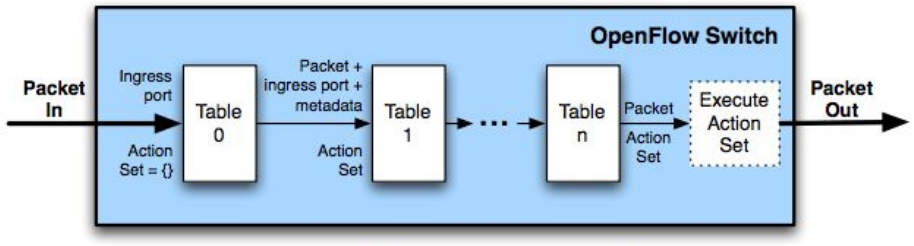
The SDN control plane is implemented by the "controller" and the data plane by "switches". The controller acts as the "brain" of the network, and sends commands (a.k.a. "rules") to the switches on how to handle traffic. OpenFlow has emerged as the de facto SDN standard and specifies how the controller and the switches communicate as well as the rules controllers install on switches.

Mininet and OpenFlow:

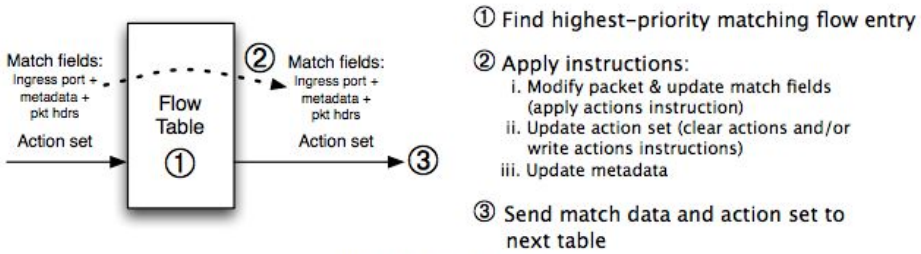
In Lab 1, we experimented with Mininet using its internal controller. In this lab (and the final project), we will instead be using our own controller to send commands to the switches. We will be using the POX controller, which is written in Python.

OpenFlow 1.3 Overview:

OpenFlow 1.3 is the version of the OpenFlow protocol supported within the Mininet environment. The following diagram explains the operation of OpenFlow switches.



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



(b) Per-table packet processing

Figure 2: Packet flow through the processing pipeline

Note that when the packet comes into an OpenFlow switch, the switch will reference a table containing “rules” and “actions”. This “flow” table contains the following fields:

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

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

The figure below shows the flow of execution that follows. If an ofp_packet_in does not match any of the flow entries and the flow table does not have a “table-miss” flow entry, the packet will be dropped. If the packet matches the “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 corresponding flow table.

5.3 Matching

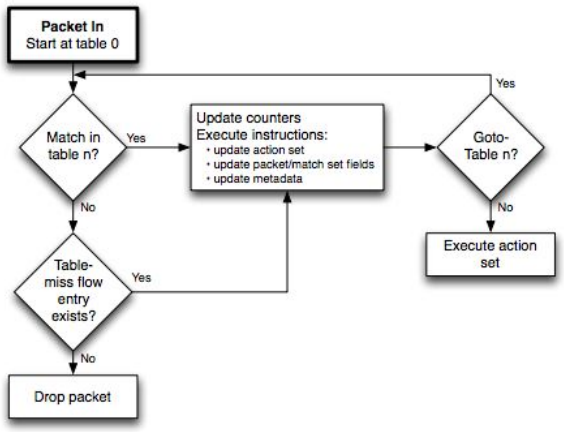


Figure 3: Flowchart detailing packet flow through an OpenFlow switch.

All of the figures and information in this section are from the OpenFlow 1.3 specification, which you can reference if you would like additional information

Assignment:

The python files are available in a zip file here.

For this assignment 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 Lab, we will provide you with the Mininet configuration, lab3.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.

In this Lab, we will also provide you with a skeleton POX controller: lab3controller.py. This file will be where you will make your modifications to create the firewall.

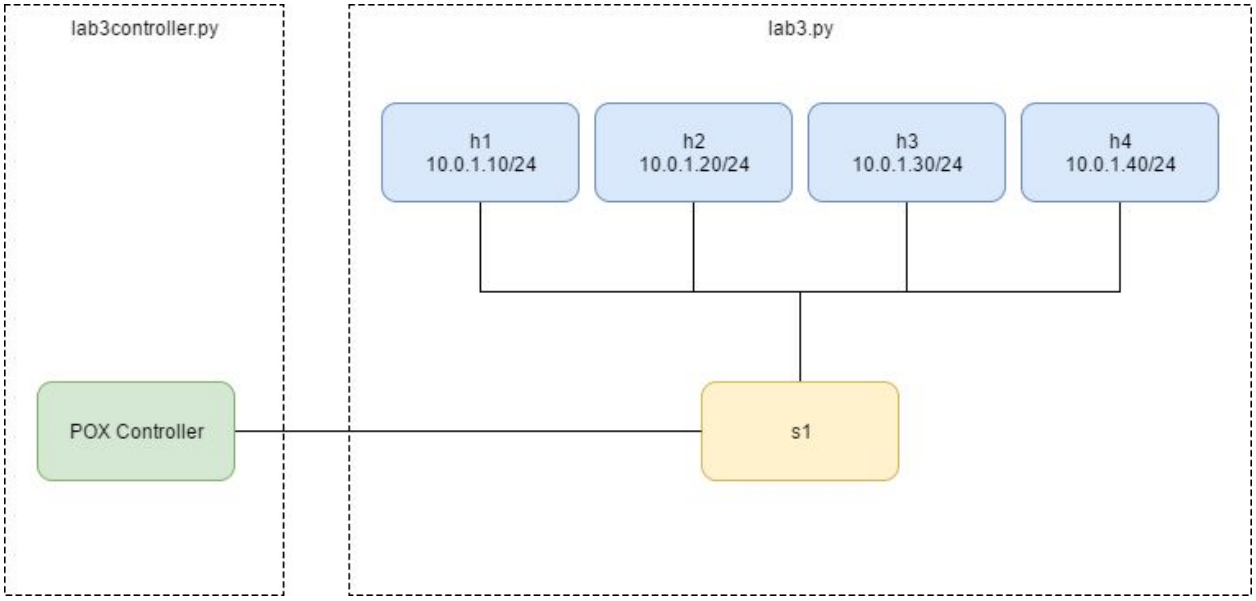
Running the Code:

To run the controller, place lab3controller.py in the ~/pox/pox/misc directory. You can then launch the controller with the command **sudo ~/pox/pox.py misc.lab3controller**

To run the mininet file, place it in ~ and run the command **sudo python ~/lab3.py**

To do this assignment, you will need to be running both files at the same time (in 2 different terminal windows).

The topology that will be created will look as follows:



Rules:

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

src ip	**dst ip**	**protocol**	**action**
any ipv4	any ipv4	tcp	accept
any	any	arp	accept
any ipv4	any ipv4	-	drop

Basically, your Firewall should allow all ARP and TCP traffic to pass. However, any other type of traffic should be dropped. It as 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. You will be downgraded if your switch simply asks the controller what to do for every packet it receives.

Hint: To do this, look up ofp_flow_mod.

Useful Resources:
https://openflow.stanford.edu/display/ONL/POX+Wiki
Inside your VM, the pox/forwarding/l2_learning.py example file.

Testing/Submission/Grading:

To test your controller, first start the controller, then start the mininet script. When you are prompted with the mininet CLI, run the following commands and take a screenshot of each:

[30 points] pingall : This should fail, since ICMP traffic should be blocked.
-20 points: ping succeeds
The remaining 10 points will be awarded depending on the quality of the explanation given.

[70 points] dpctl dump-flows : This should show a few entries. These are the entries that you installed into the switch with of_flow_mod. You'll need to do this within the timeout you specified in your of_flow_mod for the entries to show up!
-40 points: no flows shown
The remaining 30 points will be awarded depending on the quality of the explanation given.

[70 points] iperf : This should succeed.
-40 points: iperf fails
The remaining 30 points will be awarded depending on the quality of the explanation given.

Additionally, you must submit your firewall code. It should be named **lab3controller.py**.
Note: Your code will be tested. Ensure the screenshots are of your own code. Submitting screenshots that are not of your own code is considered **cheating** and a violation of the academic integrity agreement.

[30 points] Firewall code and screenshots submitted and named properly:
-10 points: lab3.pdf wrong format or name.
-10 points: lab3controller.py wrong format, wrong name, or missing.
-10 points: README not submitted

Deliverables:

1. **lab3.pdf**: Screenshots of the above commands. If you are not able to get the expected results, below your screenshot, explain what you think is going on (for partial credit).

2. **lab3controller.py**: Your firewall code.
3. **README (or README.txt)**: A readme file explaining your submission.

Note: You do not need to submit lab3.py. You SHOULD NOT modify lab3.py.