# Project 2 Practice Questions and Solutions

In this document, we first briefly describe Software Defined Networking (SDN), OpenFlow, and POX controller. We then go over practice questions (similar to PA2, but simpler) and the example code.

**SDN & OpenFlow**

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 HW 1, we experimented with Mininet using its internal controller. We can also use our own designed controller to send commands to the switches. We will be using the POX controller in this project, which is written in Python. Note that you might want to build **Mininet from source** for this project; otherwise, you may run into problems.

**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 diagram of a switch

AI-generated content may be incorrect.

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:

A screenshot of a computer

AI-generated content may be incorrect.

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.

A diagram of a software flowchart

AI-generated content may be incorrect.

All of the figures and information in this section are from the [OpenFlow 1.3 specification](https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf), which you can reference if you would like additional information

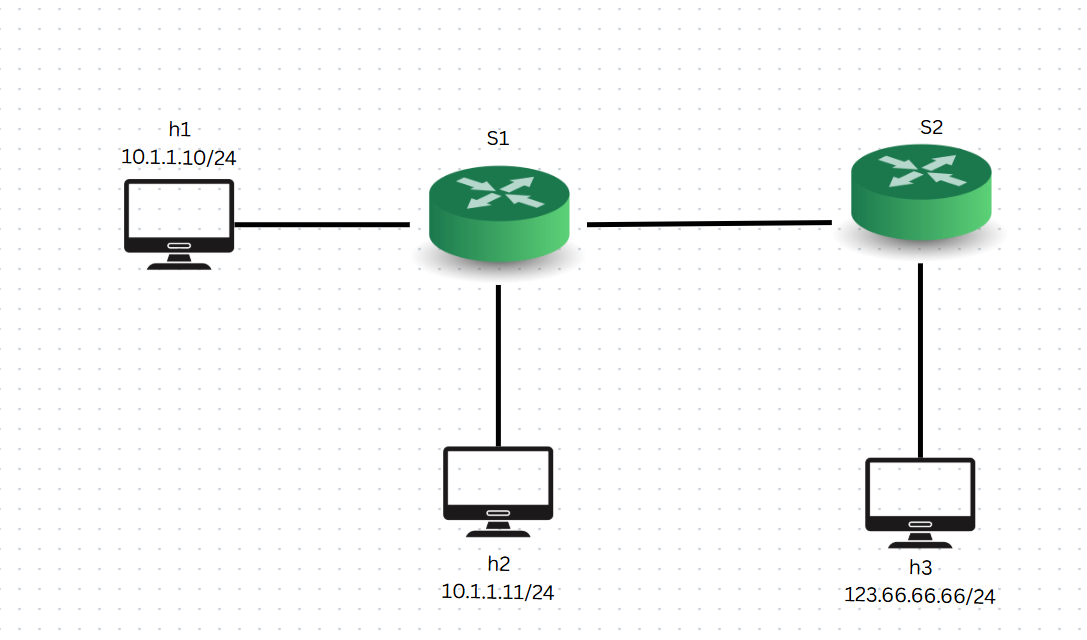
**Install POX**

Please check the following document to install and then use POX controller (for this project, you should first install POX): <http://intronetworks.cs.luc.edu/auxiliary_files/mininet/poxwiki.pdf>

* Double check your POX version
* We use POX 0.3.0 (dart) / Copyright 2011-2014 James McCauley, et al. You can get it by using

**git checkout dart**

**Practice Questions**



Objectives:

* Create a Mininet Topology to represent the above topology.
* Create a Pox controller with the following features:
  + All hosts are able to communicate, EXCEPT:
    - Untrusted Host (h3) cannot send ICMP traffic to Hosts h1 and h2.

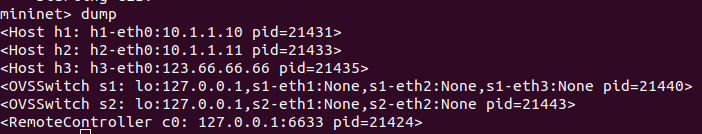
**Example Code**

* practice\_topo.py: code for constructing the above topology. In particular, pay attention to build() function. You’ll need to write your build() function for the assignment for PA2.
* practice\_controller.py: SDN controller for achieving the above requirements. In particular, pay attention to do\_final() function. You’ll need to write your do\_final () function for the assignment for PA2. Two helper functions, send\_out() and send\_drop(), are very important. They differ only in one line: send\_out() function sends an OpenFlow message to a switch to install a flow entry into the switch’s flow table and tell the switch if packets match this header pattern, then forward them out of this port. While send\_drop() function sends an OpenFlow message to install a rule that matches certain packets and performs no actions, meaning packets matching that rule will be dropped by the switch..
* Running the code: To run the controller, place practice\_controller.py in the ~/pox/pox.misc directory. You can then launch the controller with the command **sudo ~/pox/pox.py misc.practice\_controller.** To runthe Mininet file to build topology, plact it in ~ and run the command sudo python ~/practice\_topo.py. You need to run both files at the same time (in two different terminal windows).

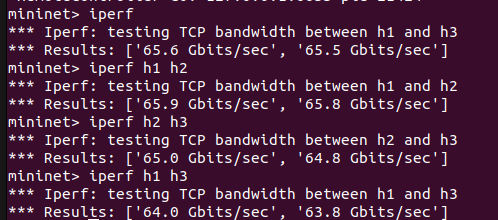
Make sure that the above code makes sense to you. See Jie’s video to get more details on how to run the code. Once you fully understand the above code and can run them successfully to get the following desired results, you can go ahead to work on PA2.

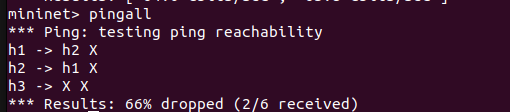
**Desired Results**

1. Use dump command to show that the topology has been correctly set up.



1. Use iperf command to show that all hosts can communicate with each other. Note that iperf by default uses TCP.



1. Use pingall to show that Host h3 cannot sent ICMP traffic to Hosts h1 and h2. (Note that Host h1 should be able to ping Host h2). 

**Hints and useful commands**

* You might want to build Mininet from source for this project
* Double check the POX version (see earlier).
* Double check your python version; python3 could work, but ideally you want to use python2 for POX controller instead. To install, simply run:

sudo apt install python

* To clean up excess controllers (this is explained in Jie’s video):

sudo mn -c