

Assignment 3

CS456/656 Computer Networks

Winter 2024

A Network on a Laptop

*Due Date: **April 1, 2024, 11:59 pm***

Work on this assignment is to be completed individually

1) Objective

The goal of this programming assignment is to get hands-on experience in computer networking by simulating a virtual network using Mininet. Mininet is a network emulator which creates a network of (virtual) hosts, switches, controllers, and links. This assignment will be an opportunity to comprehensively review the Internet Protocol (IP) in a virtual Software Defined Network (SDN). Specifically, we will experiment with routing configurations and the OpenFlow protocol, which are the building blocks of an SDN. Note that SDNs differ from traditional networks which use distributed protocols on routers and switches to find paths and control network traffic.

You will begin with installing Mininet and using Mininet's Python API, implement a virtual network topology. Next, the switches in the virtual SDN will be configured to forward packets according to given rules and specifications. If the configurations are correct, the virtual hosts inside the virtual network will be able to communicate with each other. This assignment is divided into four parts: Part A, B, C, and D.

2) Background and Setup

Background. You will find necessary background material in the following textbook chapter sections and articles:

- a. Chapter 4 Section 4.4 Generalized Forwarding and SDN
- b. Chapter 4 Section 4.5 Middleboxes
- c. Chapter 5 Section 5.5 The SDN Control Plane
- d. Mininet Documentation at <http://mininet.org/overview/>

Setup. For this assignment, you will have to download a virtual box and virtual machine to run on the virtual box. **You are encouraged to start early and install the VM as soon as possible, so that you have plenty of time to seek help on Piazza for setup and coding.**

1. To download a virtual machine, please download and install [VirtualBox](#). (If you own a MacBook with an M1 processor, please follow the instructions in Section 8)

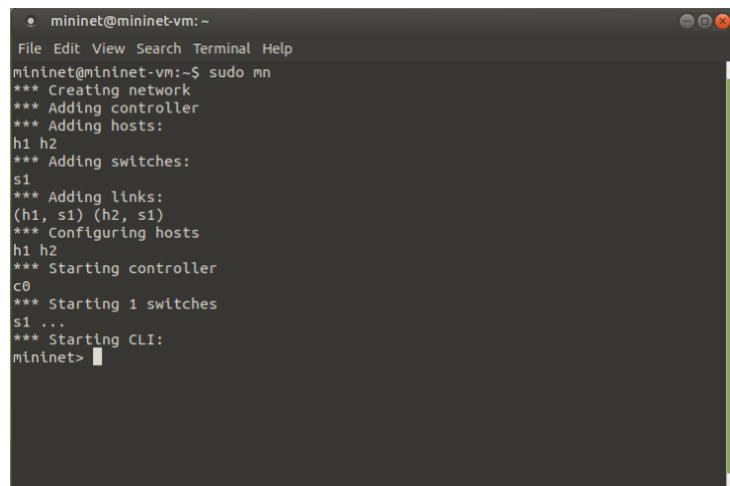
2. We have created a virtual machine for CS456 Computer Networking course, which includes the software needed to complete the assignment. Please download it from [here](#), it will download as an .ova file.

To run the virtual machine, double-click on the downloaded .ova file, which will open in VirtualBox. Complete importing the appliance, our virtual machine, without changing the default settings.

Getting Started

Quick setup guide assuming you have VirtualBox installed:

1. Start the provided VM in Virtualbox.
2. After the VM boots, you will see a screen with a password prompt for the user `mininet`. Enter `mininet` as the password.
3. In the CS456 VM provided for this course, you will find all CS456 related files in the directory `/home/mininet/cs456-a3`
4. Open a terminal by selecting Menu -> System Tools -> MATE Terminal.
5. Enter `sudo mn` in the terminal, you should see the following output:



```
mininet@mininet-vm: ~  
File Edit View Search Terminal Help  
mininet@mininet-vm:~$ sudo mn  
*** Creating network  
*** Adding controller  
*** Adding hosts:  
h1 h2  
*** Adding switches:  
s1  
*** Adding links:  
(h1, s1) (h2, s1)  
*** Configuring hosts  
h1 h2  
*** Starting controller  
c0  
*** Starting 1 switches  
s1 ...  
*** Starting CLI:  
mininet>
```

The above output shows that Mininet has created a simple network topology with one switch and two hosts connected to it. Type `exit`, to exit the Mininet shell, then type `sudo mn -c` to clear the created topology.

```

mininet@mininet-vm:~$ sudo mn -c
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflowd
ovs-controllerovs-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflo
wd ovs-controllerovs-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/nul
l
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
*** Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([_.:alnum:])+eth[[:digit:]]+'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.

```

Note that the creation of your next topology may fail, if you do not clear the previous one. Also, note that you will face problems if you have two instances of Mininet running at the same time (e.g., in two different terminals).

- The VM comes with Open vSwitch and a controller pre-installed. Open vSwitch (OVS) is used for creating a programmable switch, i.e., a switch that works with flow table rules. The controller installs the rules in the network switches based on the network topology to ensure connectivity between hosts. Mininet uses OVS and a controller in its default network. To test whether the controller works, type `sudo mn --test pingall` in the vm terminal (Ping is a network utility that sends packets from one host to another to check connectivity). You should see an output similar to the illustration below.

```

mininet@mininet-vm:~$ sudo mn --test pingall
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 5.408 seconds
mininet@mininet-vm:~$

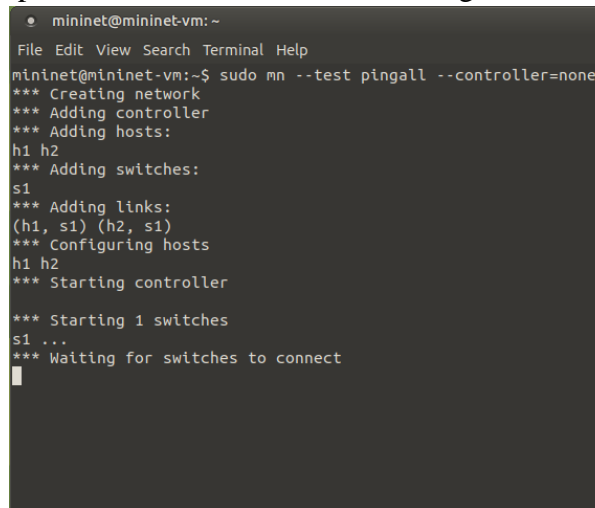
```

Note the logs that state that Mininet is starting the controller. Also note that both hosts are reachable from one another according to the logs. Again, type in `sudo mn -c` to clear the created topology.

7. To see the effect of the controller, we will repeat the above command but ask Mininet to ignore the controller. Type in the command:

```
sudo mn --test pingall --controller=None
```

You should see an output like the one illustrated in the figure below.

A screenshot of a terminal window titled 'mininet@mininet-vm:~'. The terminal shows the command 'sudo mn --test pingall --controller=None' being executed. The output includes: '*** Creating network', '*** Adding controller', '*** Adding hosts: h1 h2', '*** Adding switches: s1', '*** Adding links: (h1, s1) (h2, s1)', '*** Configuring hosts h1 h2', '*** Starting controller', '*** Starting 1 switches s1 ...', and '*** Waiting for switches to connect'. The terminal is currently stuck at the 'Waiting for switches to connect' message.

```
mininet@mininet-vm:~  
File Edit View Search Terminal Help  
mininet@mininet-vm:~$ sudo mn --test pingall --controller=None  
*** Creating network  
*** Adding controller  
*** Adding hosts:  
h1 h2  
*** Adding switches:  
s1  
*** Adding links:  
(h1, s1) (h2, s1)  
*** Configuring hosts  
h1 h2  
*** Starting controller  
  
*** Starting 1 switches  
s1 ...  
*** Waiting for switches to connect  
█
```

We see that Mininet gets stuck, while waiting for the switches to connect. This is because in the absence of a controller, the switches do not have forwarding rules and do not know how to forward any of the packets.

For the first two parts of this assignment, Part A and Part B, we will be working without a controller, meaning that we will be adding static flow table entries to the switches manually, rather than having a controller do it. The objective is to learn about flow tables and forwarding rules.

3) Part A: Hands on Mininet

In this part, you will learn how to write OVS rules for forwarding packets in a custom topology. Mininet allows the *simulation* of arbitrary network topologies using a Python API. You will learn how to create a custom topology to run in Mininet using the API. You will also learn how to add static OVS rules to switches in a topology.

In the CS456 VM provided for this course, you will find all CS456 related files for Part A in the directory `/home/mininet/cs456-a3/part-A`. For Part A, the files are, `topology.py`, `mininet_topology_full.pdf` and `ovs_connect_h0h1.sh`

Running a Custom Topology

The python file, `topology.py` uses the Mininet Python API to create a custom topology. A pictorial representation of the network created in `topology.py` is in `mininet_topology_full.pdf`. Run the custom topology file in Mininet using the command:

```
sudo python topology_filename.py
```

where `topology_filename.py` is the name of the topology file you need to run. You will see that running `topology.py`, will create 10 hosts and 10 switches in a Mininet shell.

Use the commands below in the Mininet shell to explore the topology.

```
mininet> nodes
mininet> net
mininet> dump
mininet> links
```

The Mininet shell also allows running commands from each node. Try the following commands for example:

```
mininet> h1 ifconfig
mininet> s1 ping s2
```

When running the above commands, the Mininet shell will infer that the first item (*h0*) is the name of the node on which to run the rest of the command (*ping h1*); it will, behind-the-scenes, enter that node's bash in your stead.

Furthermore, to access a specific node, you can use:

```
xterm node_name
```

where `node_name` is the name of the node you are trying to access, e.g., `h1`, `s1`, etc. This will open a new terminal window inside the VM. Now, you can run any command, such as, `ifconfig` inside that window to see the interfaces of the corresponding node.

Refer to the Mininet [walk through](#) for more Mininet commands.

Adding OVS rules

The python file we have created for you creates the topology but does not create the OVS rules for forwarding packets, therefore, there is no connection between the hosts. You can verify this by running the command `h0 ping h1` inside the Mininet shell and comparing the result with the previous ping. You should note that **h0 cannot ping h1**. You can press `Ctrl+C` to kill the ping command, without terminating the Mininet VM.

The shell script `ovs_connect_h0h1.sh` includes the necessary commands to enable OpenFlow version 13 and install the forwarding entries to Open vSwitches `s0` and `s1`.

The shell script has to be executed while Mininet is running the custom topology. Therefore, while Mininet VM is running the custom topology, open, **a new terminal window**, and change directory (`cd`) to the directory `/home/mininet/cs456-a3/part-A` that contains the file `ovs_connect_h0h1.sh`, and run:

```
sudo ./ovs_connect_h0h1.sh
```

Go back to the Mininet shell and try `h0 ping h1` again. You will see that the ping works. Try the ping in the opposite direction, does that work as well?

Read and understand the file `ovs_connect_h0h1.sh`. The first two commands set `s0` and `s1` to use OpenFlow version 1.3. The two commands add flow entries to `s0`, and the last two commands add flow entries to `s1`. Explain, in your own words, what each entry in each one of the `add-flow` commands means, in a file called, `partA.md`

Write a shell script, called `partA_connect.sh` to add the right flow entries in the appropriate switches to make sure that the following pairs of hosts, and only these hosts, can ping each other:

`h1 ↔ h4 h2 ↔ h0 h3 ↔ h6.`

4) Part B: Custom Topologies

In this part, you will create a custom topology using the python API for Mininet. The sample custom topology of part A, `topology.py`, is a good starting point for learning how to create your own topologies. For Part B, you must create the topology illustrated in Figure 1 using Mininet, such that the host pairs Alice and Bob can ping each other, the host pairs Bob and Carol can ping each other, while Alice and Carol remain disconnected, which means that the ping between the two pair gets stuck.

Note that while the OVS rules that you saw in part A had multiple matching fields, in general, not every one of those fields is required in an OVS rule, e.g., an OVS rule may route traffic based on link-

layer addresses and nothing else. Also, note that routing in every switch should be implemented by OVS rules, other methods of providing connectivity will not be accepted.

Restrictions on required OVS rules: the forwarding nodes in Figure 1 are Layer-3 switches (or routers). No switch should forward packets based on the incoming port number or L2 MAC addresses, i.e., you will lose points if you match flows by `in_port` or `MAC addr`. Choosing the unspecified MAC addresses in the topology file is up to the student.

Tips on implementing Part B:

- 1) Do NOT set IP addresses on *any switch* in Mininet. It will stop the OVS rules from being effective. The gateway addresses (shown as `gw` in Figure) are meant to be used in the same way as in part A, in the host's settings.
- 2) Mininet switches can be pinged from any node, regardless of the OVS rules you introduce. Therefore, to debug connectivity issues, run Wireshark on the virtual machine and choose an interface to monitor ICMP traffic while a ping is running. You can alternatively run `sudo ovs-ofctl -O OpenFlow13 dump-flows R1` (replace R1 with the appropriate router name) to see the number of packets matched with any rule in the switch, to make sure the rule is acting as it should.

You will need to submit two files for this part of the assignment:

- A python script: This will set up the topology depicted in the figure, such that all hosts are connected to the network. (No hosts get a network unreachable error when trying to ping)
- A shell script: This will set the correct OVS rules in the switches, such that the host pairs specified previously are able to ping one another.

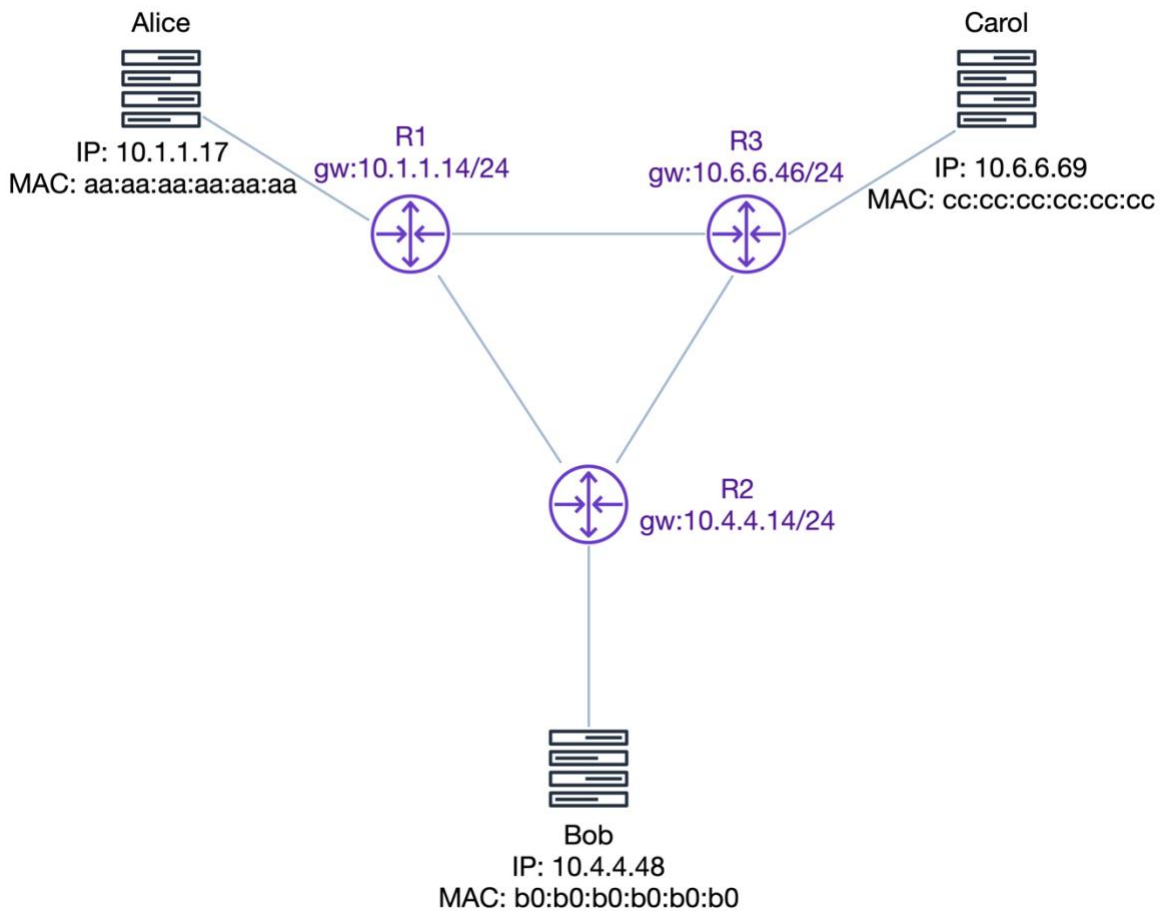


Figure 1. Topology for Part B: Custom Topologies. The squares represent hosts for Alice, Bob and Carol. R1, R2, and R3 are layer-3 switches, each with 3 ports.

5) Part C: Introducing the Controller

In this part, we explore the SDN controller and its value for configuring networks from a centralized component. In parts A and B, you have seen how configuring networks at a low level, even for achieving very simple requirements, can become complex and exhaustive.

However, in the real world, it is often not a human operator that comes up with those OpenFlow rules! The controller is a centralized software that acts as the brains of the SDN network. Using the SDN controller, you can instruct high-level policies, such as “host A should be able to ping host B”, and let the controller figure out how that translates to low-level OpenFlow commands and deploy it on the network switching devices. The controller is a server that will run in a separate terminal than the one you will use for Mininet commands. The CS456 VM you are provided with comes with a preinstalled POX controller.

Follow the instructions below to run a topology connected to the POX controller:

1. Clear any topology you may have running using the `sudo mn -c` command.
2. Open a terminal, `cd` to `~/pox`
3. Type and run the command below to start the POX controller and open its shell

```
./pox.py --verbose py openflow.of_01 --port=6633  
openflow.discovery forwarding.l2_learning host_tracker
```

Note that you will see the controller's logs in this window after you run the Mininet topology.

4. Open a different terminal window, and run the following command:

```
sudo mn --controller=remote,ip=127.0.0.1,port=6633  
--topo=tree,depth=3 --mac --switch ovs
```

This will create a tree topology as illustrated below and connect it to the POX controller on port 6633 that was started in Step 3.

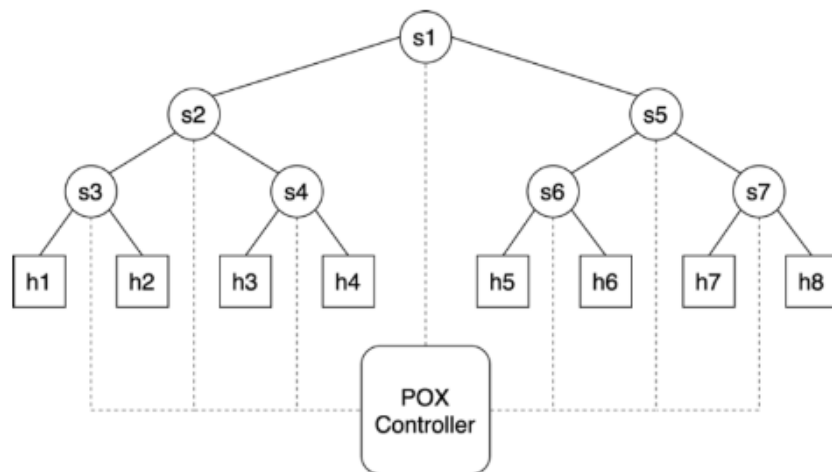


Figure 2. A sample tree topology available in Mininet.

5. In a new terminal window, you can explore the settings of a switch `s1` and the flow rules on a switch `s1` using the following commands:

```
sudo ovs-ofctl show s1  
sudo ovs-ofctl dump-flows s1
```

The objective of this part of the assignment, is to explore the effect of the POX controller on the network by answering the following questions.

- 1) Try pinging h5 from h1 (type `h1 ping h5` in Mininet). The ping should succeed, even though you have not installed any OVS rules on the switches yourself. Study the output in the POX controller terminal. What has the controller done? Include a screenshot of the output as well as your interpretation of it. Make sure that the screenshot is readable. Hint: To interpret the effect of the controller, consider the path ping packets must take from h1 to h5. Now consider the number of logs generated by the controller and explain how they relate to the topology.
- 2) Take a screenshot of the ping RTT times for the first 5 ping messages. Compare the RTT of the first ping message with the subsequent ones. Is there a difference? Why or why not?
- 3) Open a new terminal in the CS456 VM and dump the flow rules installed on switch `s1` using the command, `sudo ovs-ofctl dump-flows s1`. Dump the flow rules on all the switches both before and after the ping, make sure you change `s1` in the command above to the appropriate switch name. Compare the initial flow rules with the ones installed in the switches after initiating ping between hosts. What do you think the initial rules are for? Do all switches have newly installed flow rules after the ping? Explain why. Include a screenshot of the terminal showing all the dump commands and their outputs in your answer, use multiple screenshots, if necessary. The screenshots should be readable and clearly show which output pertains to which switch. Observe how the OVS rules are different from the rules you defined in part A. Explain what that indicates about the type of packet-forwarding that this controller implements? (Hint: Consider the fields that are or are not being used in the OVS rules compared to part A.)

6) Part D: Middlebox

Middlebox is a real or virtual device that is placed inside the network and is used to perform some network function. A middlebox network function might modify the network traffic passing through the middlebox or passively collect information about it. For instance, a firewall is a network function that matches the traffic passing through it against a set of configured rules and filters out parts of the traffic, e.g., dropping all traffic for a certain TCP port or allowing traffic only from certain IP addresses.

In this part of the assignment, you will write a program to install the necessary OpenFlow rules to deploy a simple middlebox. Rather than deploying a complicated middlebox like a firewall or a load-balancer that requires lots of configuration by systems administrators, we will deploy a very simple middlebox that will append the utf-8 encoded string “from the middlebox” to the payload of all the packets it receives.

In the CS456 VM provided for this course, you will find some of the CS456 related files for Part D in the directory `/home/mininet/cs456-a3/partD`. We will provide you with a program to send UDP packets in `udp_client.py`, the middlebox program `cs456_middlebox.py`, the custom topology `cs456_tree_topology.py`, and a UDP server program in `udp_server.py` so that you can test your code. The objective of this part of the assignment will be to use the POX SDN controller to install the OpenFlow rules necessary for your middlebox to work.

Your program should be able to work in any *loop-free* network topology. That is, in any loop free network topology, you should be able to deploy the middlebox program, and use the UDP client program to send a message to the UDP server program and the message should have “from the middlebox” appended to it by the time it reaches the UDP server.

We have provided you with a skeleton file at `/home/mininet/pox/pox/cs456/a3.py` that you should use as a starting point for writing your controller program.

Because your program needs to run within the POX controller framework, the steps for running your program will be different than that of a traditional python program. To run your program, you will first start POX and load the required modules.

1. Clear any topology you may have running using the `sudo mn -c` command.
2. Open a terminal, `cd` to `~/pox`
3. Run the following command:

```
./pox.py --verbose py openflow.of_01 --port=6633 \
    openflow.discovery forwarding.l2_learning host_tracker cs456.a3
```

This will start the POX command line where you can issue commands that run within the POX controller framework. Notice that we have loaded both the `forwarding.l2_learning`, the `openflow.discovery` and the `host_tracker` modules. The first, `forwarding.l2_learning` module turns every OpenFlow switch that connects to the controller into a MAC learning switch. This ensures that our ARP requests are switched through the network correctly. The `openflow.discovery` module enables the topology discovery features of the POX controller. In this way, an application can retrieve a graph of the network topology. A network graph reveals vital information about connectivity and reachability. We use it to compute the shortest path between the client node and the middlebox node and the middlebox node and the server node. The third module, the `host_tracker`, will keep track of the hosts in the network so that we can ask POX where the hosts are located and retrieve information about their MAC and IP addresses. For host tracking to work you always need to run `pingall` from

the Mininet console, *after you run the POX controller* **but** before you test your code.

Now that you are running the POX command line you can load all the symbols from the python module that you are working on by typing:

```
from cs456.a3 import *
```

To make the assignment simpler, your application will **not** need to set up the flow rules to implement the middlebox functionality in real time, instead you will use the command line interface to provide your program with the following parameters:

- DPID of the switch that the client host is attached to.
 - DPID of the switch that the server host is attached to.
 - DPID of the switch that the middlebox host is attached to.
 - Source UDP port of the traffic
 - Destination UDP port of the traffic
4. Your program will use this information, along with the POX APIs, to install the necessary flow rules to *steer* only the UDP traffic from the `client` host to the `middlebox` host and finally on to the `server` host. The middlebox host should only receive the UDP traffic that matches the 4-tuple that defines the packet from the client host to the server host, with source UDP port and destination UDP port. The middlebox program should append the message “from the middlebox” onto the payload of each of the UDP packets it receives and then transmit the augmented packet back out of its network interface.
 5. The POX controller must be programmed for the middlebox functionality we require. The file `/home/mininet/pox/pox/cs456/a3.py` provides the code necessary to program the POX controller. The `install_udp_middlebox_flow` function, which we have provided for you in the `a3.py` file, will print a message encouraging you to complete the assignment by adding code to this function. The skeleton file also contains some helper functions that would be useful in completing the assignment. The parameters and return values of each of the helper functions are documented within the source code. The `install_udp_middlebox_flow` function also contains some comments that will guide you in your completion of this part of the assignment.

To run the program in your controller, type and execute the command below in the POX command line:

```
install_udp_middlebox_flow(<client_dpid>,<server_dpid>,<middlebox_dpid>,source_port,destination_port)
```

6. Testing Your Program. You are expected to test your implementation of `install_udp_middlebox_flow` using Mininet. As was mentioned earlier, your program is expected to work in various loop-free network topologies so you should test with two or three different topologies. The Mininet command line program can be used to automatically create a few different network topologies of various sizes. For example, to create a *linear* topology with 5 nodes and one host attached to each node run:

```
sudo mn --mac --topo=linear,5 \  
--controller=remote,ip=127.0.0.1,port=6633
```

We will *only* use the built in linear topology, with differing number of hosts, to evaluate your submission. However, for testing we have provided an additional custom topology. The topology is a tree topology where the branching factor of the tree is always two and there is a host connected to every node in the tree. You can instantiate this custom topology using the command:

```
sudo mn --mac \  
--custom /home/mininet/cs456-utils/cs456_tree_topology.py \  
--topo=SimpleTreeTopo,3 \  
--controller=remote,ip=127.0.0.1,port=6633
```

Once you have started Mininet you need to start the middlebox program on the middlebox host that you have chosen and the server program on the server host that you have chosen.

Finally, you need to follow the steps described in step 5 to run `install_udp_middlebox_flow` with the DPID's that correspond to your client, server and middlebox hosts. All the programs that are to be run on the hosts in the network are in `/home/mininet/cs456-a3/part-D` directory and will tell you the parameters they need if you invoke them with the `--help` flag.

You can start an `xterm` session on the client, middlebox and server hosts from the Mininet terminal and run the appropriate programs from the `/home/mininet/cs456-a3/part-D` directory with the correct parameters to transmit a message from client host to server host. The controller program you wrote should install the appropriate flow rules

to enable the middlebox functionality that is required for Part D of this assignment. The expected result is that for every UDP packet sent from client host with source port, the server host should print a log line with the message that was sent appended with the phrase “ from the middlebox.”

7) Tips and Tricks

- Always assume that there is exactly one host connected to every switch. This assumption is easy to realize in practice by using either the Mininet linear topology or the custom tree topology that has been provided for you.
- Some documentation for POX is available at <https://noxrepo.github.io/pox-doc/html/>. If you're wondering how to do something with the POX API, you can usually find the answer on this page.
- When creating the assignment, the TA's found that the most efficient way to develop the controller program was to do the following:
 - Run the Mininet VM GUI and start an instance of Mininet using the desktop of the VM. This is important because there is no way to start shells on the Mininet hosts without an X server (i.e., if you are SSHing to the VM rather than using the GUI)
 - To write the necessary code to complete the assignment and to run POX, SSH to the Mininet VM and work in the terminal, this will let you avoid dealing with a possibly sluggish VM GUI.
 - We have already setup the CS456 VM to enable SSH, you can run `ssh` on your local machine, by opening a terminal or through putty, to remotely login to the CS456 VM by running the `ssh` command below

```
ssh -p40000 mininet@127.0.0.1
```

- If you prefer using a GUI text editor to write your code, you can set up a *shared folder* on the VM so that you will be able to access files on the VM from your host computer and edit them locally. For help setting up a shared folder, see [shared folder setup](#) (you can ignore the parts about installing the VirtualBox guest additions, we've done that for you already).
- For Part D of the assignment, you are not allowed to use OpenFlow to **modify** any of the traffic passing through the switches. Instead, your program should steer traffic destined for the server host to the middlebox host before the modified traffic is steered to the server host. The only modification of the traffic sent by the client with the same UDP source port will be done by the middlebox program.
- When you are debugging your implementation, remember that you can use `ovs-ofctl` to inspect the flow tables of the switches in the network.

8) VM on M1 Macs

The Apple M1 is an ARM-based CPU, so you cannot install VirtualBox and use Intel-based VMs such as the one provided for this assignment easily like the others. It is strongly suggested that you find another machine to complete this assignment, since there will be performance degradations with the solution below. If that is not possible for you, then follow these instructions.

1. Download and install **UTM** from [this link](#). Once downloaded, run the *.dmg* file.
2. Download [this .qcow2 file](#).
3. In the opened UTM window, click on the *Create a New Virtual Machine* button.
4. To set up the VM, follow this [video](#).

9) Deliverables

- 1) For Part A there are two deliverables:
 - 1) The file `partA.md`, which explains what each field name in the first add-flow command in the provided file means. Each field should be explained in no more than one sentence.
 - 2) Submit a file named `partA_connect.sh`, which includes the OVS commands that enable only the following pairs of hosts to ping each other and only each other:

`h1 ↔ h4 h2 ↔ h0 h3 ↔ h6`

Make sure that after you run the provided `topology.py` and `partA_connect.sh` in two separate terminal windows, the specified host pairs can ping each other with no other steps required. You will lose marks if any additional hosts can ping each other, or if any of the mentioned hosts cannot ping each other. Your script **MUST** contain OVS routing entries and comments similar to those provided to you in the walkthrough.

- 2) For Part B, there are two deliverables:
 - 1) Submit the Python code for implementing the topology for Part B, in a file named `partB_topology.py`.
 - 2) The shell script `partB_connect.sh` that connects the hosts through OVS rules as described in part B.
- 3) For Part C, there is one deliverable, a file named `partC.pdf` that clearly answers the three questions asked in part C, by including explanations that are justified by one or more of the required screenshots.
- 4) For Part D, there is one deliverable, your completed `/home/pox/pox/cs456/a3.py` file.

10) Instructions for Submission

Submit all your files in a single compressed file (.zip, .tar etc.) using Assignment 3 drop box on LEARN.

For the automatic grading scripts to run smoothly, make sure that all your files are next to each other in one directory, do not put files in subdirectories.

Remember, you cannot share your program or work with any other student or in any group on social media. This programming assignment is to be completed individually. You **will** lose marks if your answers are not legible.

You must hand in the following files / documents:

- Source code files: `partB_topology.py` and `a3.py`
- Shell script files: `PartA_connect.sh` and `partB_connect.sh`
- Written answers: in `partA.md` and `partC.pdf`

You are encouraged to start early and install the VM as soon as possible, so that you have plenty of time to seek help on Piazza for setup and coding.

11) Use of Generative AI

Use of generative AI is allowed if you are capable of fully understanding the generated code and what it is doing line by line, by yourself. You are also required to provide the entire chat log/prompts used to generate the code.

The AI tool used should be documented in your README file in the following format:

Filename: <name of file, e.g., server.py>

Lines: <X> to <Y>

Reference: <Bibliographic Reference as described here:
https://subjectguides.uwaterloo.ca/chatgpt_generative_ai/aigeneratedcontentcitation>

Chat logs/Prompts: < Chat logs/Prompts >
<extra notes>

X and Y are line numbers in the corresponding file. If the chat log or prompt is too long, provide a link to a document on OneDrive containing the full chat log/prompt(s). Make sure that the document is accessible to the TAs. The extra notes should contain a high-level description of how you tweaked and adapted the auto-generated code.

12) Additional Notes

- a. Be clear about all the questions that you are answering, keeping answers brief and concise while including all the required information.
- b. For answering the OpenFlow questions in Parts A and B, make sure that shell scripts containing the OVS commands work. The shell scripts will be tested and expected to provide the appropriate connectivity.

- c. The python and shell scripts are expected to run seamlessly in the VM provided for the course. You do not need any additional dependencies for parts A, B and D. In case you must, provide the installation guides in a file `readme_partX.md` where X is replaced by A, B or D. It is your responsibility to make sure the installation guide is clear and brief.
- d. Using the Markdown syntax in the answer files is not required but encouraged.

13) Grading Rubric

Rubric Item	Scenario Description	Failure Penalty
Part A: Hands on Mininet (15%)		
1	Question A-I: Explain the meaning of the fields in one rule, explain the meaning of the rule	5%
2	Question A-II: The connections between pairs being set up by running the shell script	6%
3	Question A-II: The remaining pairs remaining disconnected after running the shell script (should only be granted if the previous item works)	4%
Part B: Custom Topologies (40%)		
5	Question B-I: Correct specification of the network links, topology, IP, specified MAC addresses and gw	10%
6	Question B-I: Observation of the specified constraints on OVS rules. Only granted to those who have implemented 7 and 8.	10%
7	Question B-II: Working ping between Alice and Bob, Carol and Bob	10%
8	Question B-II: Alice and Carol should remain disconnected, only granted to those who have implemented item 7.	10%
Part C: Introducing the Controller (15%)		
10	Question C-I: Correct controller logs	2%

11	Question C-I: Explaining what the controller is doing	2%
12	Question C-II: Ping output and observation	2%
13	Question C-II: RTT difference justification	2%
14	Question C-III: Correct flows from all switches	2%
15	Question C-III: Explaining the collective effect of the installed flows, and comparing the flows before and after ping	5%
Part D: Middlebox (25%)		
16	D-I: Flow rules are correctly implemented for the linear topology	15%
17	D-II: Code Inspection	10%
General: Code readability, documentation and inspection 5%		