COMMUNICATIONS NETWORKS II (521377S) – Mininet

EXERCISE 2023

Mininet Overview

Mininet is one of the most widely used research tools for SDN and OpenFlow, it allows you to simulate a virtual network with switches, hosts, and an SDN controller with a single command. Hence, mininet is a great way to learn about SDN and OpenFlow with different network configurations. For more about mininet, please go to http://mininet.org/

Download and Installation:

Linux is the recommended platform for this exercise. For instructions on how to download and install mininet on Linux, check this link: http://mininet.org/download/ . Instructions are available for the whole process of downloading packages and their installation. There are also some experimental examples for the verification of the installation. If you follow the installation instructions correctly, you will have all the required packages installed including Mininet VM, OpenFlow tutorials with examples, as well as Wireshark. Wireshark is a free and open-source packet analyzer used for network troubleshooting, analysis, software and communications protocol development, and education. Please follow the link http://mininet.org/walkthrough/ to learn the basic mininet commands.

You will also learn some simple examples for setting different OpenFlow topologies, check connectivity between switches and hosts and do more.

For those who may want to perform these tasks from a Windows computer, you can download VirtualBox from this link: https://www.virtualbox.org/wiki/Downloads and configure it with Linux. You can also download the All-in-one SDN App Development Starter VM from this link http://sdnhub.org/tutorials/sdn-tutorial-vm/

NB: This package comes with some bug that needs to be fixed based on your system specifications and your OS configurations, it is the responsibility of students to figure out the fix for the bug if they choose to use this package.

Reporting

The exercise work is designed to be carried out by **groups of two or three students**. If you do not find a partner, you can do the laboratory work alone.

A written report should be submitted using the thesis format by **15th of March 2023 (23:59:59)** in pdf format through Moodle. Thesis report format can be found in the links below:

http://www.oulu.fi/sites/default/files/content/masters_guide_0.pdf

www.oulu.fi/sites/default/files/content/WordTemplate_2016.docx.

In the past, it was sometimes difficult to identify what questions are being answered with each response and paragraphs in the report. For more clarity, ***please include the numbered questions and their corresponding answers beneath in the report. Also label each figure according to the figure numbers provided in the instructions.***

No Email or paper form reports are accepted. Your written report, as minimum, should contain the answers to all questions and the appropriate figures as a result from the simulations. Reports will be graded as fail or pass. It is required that students pass the lab work before they can get the credit for the course.

Your submissions will be reviewed as soon as possible, and a feedback e-mail will be sent to group members. If your work is not of acceptable quality, a list of things to be fixed will be included. You may fix and resubmit your work multiple times before the deadline, but only once after the deadline, and this has to be within 7 days after the deadline.

For further information/assistance please contact:

Assistant:
Tharaka Hewa
e-mail: Tharaka.hewa@oulu.fi

EXERCISE OBJECTIVES

In this exercise, you'll be introduced to some tools that will be useful in understanding the basic operating principles of Software Defined Networking (SDN) and OpenFlow as well as their characteristic elements and development environment. Most of the tools and concepts used in this exercise were discussed in lectures 3 and 4 of the course.

It is advisable to review the lecture materials for a better understanding of the theoretical concepts behind the various tasks covered in this exercise. This exercise will also introduce both general and OpenFlow-specific debugging tools and commands.

In addition, we will introduce MinEdit; a simple graphical user interface for simulating SDN. Is purely based on Mininet, most people refer to it as Mininet GUI. This would come in handy for those who may not want to deal much with a coding environment. However, it is important to note that not all SDN features demonstrable in Mininet can be replicated in MiniEdit. It consists mainly of some basic elements that gets you started with your network design, giving you a pictorial view of what your emulated network would look like in reality and helping you position the elements more accurately. These tools will come in handy for anyone who would like to continue to work in various SDN related projects moving forward.

RELEVANT TERMINOLOGIES

OpenFlow: is a communication protocol allows access to the forwarding plane of a network switch or router over the network. In other words, it enables the server to inform the network switches where packets should be sent on the network. The OpenFlow protocol is the basis for SDN implementation. This is the protocol that allows for the separation of the control and forwarding functions of the SDN architecture.

OpenFlow Controller: It manages flow control in Software Defined Network (SDN) environments. It resides above the OpenFlow interface in the SDN reference model and handles the control plane functionalities. The OpenFlow reference distribution includes a controller that acts as an Ethernet learning switch in combination with an OpenFlow switch. In this exercise, you'll run it and look at messages being sent. You'll also learn to write our own controller on top of NOX or Beacon.

POX/NOX/Beacon: These are various platforms for running SDN controller applications. They are also referred to as controllers in various contexts. NOX is the original OpenFlow controller developed by Nicira Networks and now owned by VMware, along with OpenFlow back in 2009. It later developed into three versions; NOX classic which was the original version, New NOX which was based on C++, and then POX which is a python-based version of the controller. You can find more information about their peculiarities in this post: https://thenewstack.io/sdn-series-part-iii-nox-theoriginal-openflow-controller/. Beacon is a Java-based open source OpenFlow controller application.

OpenFlow Switch: This is generally a software program or a hardware device that forwards packets in an SDN environment. The OpenFlow switch sits below the OpenFlow interface on the SDN reference model. The OpenFlow reference distribution includes a user-space software switch. Open vSwitch is another software switch based on specific kernel. There are several hardware switches available from big players like Broadcom, HP, NEC, and others. One of the goals of SDN is to eliminate the platform dependence of such switches by making them act as mere packet forwarding devices.
ovs-ofctl: Is a command-line utility that sends quick OpenFlow messages to each individual switch's flow table. Be default, ovs-ofctl comes with Open vSwitch. It is useful for viewing switch port and flow stats or manually inserting flow entries. It also serves as a good debugging tool since its able to view flow state and flow counters. With this, any unexpected flow behavior can be check and rectified.

xterm terminal: This are command line terminals for individual hosts on an SDN environment. These terminals connect to the hosts in the virtual network. They are automatically created once you start up a the virtual network, we will see how to access and run commands on this terminals bother form the host command line and from the MinEdit environment. They are generally labeled at the top with the name of the associated host.

Wireshark: This is one of the most popular tools for analyzing network packets in general. It is not peculiar to SDN or OpenFlow. It comes very suitable for various SDN use cases for a number of reasons. First is a graphical utility hence making it more user friendly for both techies and the not so tech savvy. Second, it is integrated to the OpenFlow reference distribution by default, hence there is no need to make any special installations of the application. Third, the Wireshark dissector automatically parses OpenFlow messages sent to the OpenFlow default port (6633) in a conveniently readable way.

iperf: This is another generic tool in networking, again this is not peculiar to SDN or OpenFlow. It is a command-line utility generally used for testing the speed of a single TCP connection. **cbench:** This is a utility for testing the flow setup rate of OpenFlow controllers.

EXERCISE TASKS

In performing these tasks, is assumed that you have a running error-free Mininet environment already setup according to the instructions in the Download and Installation section. If you're having any problems installing and running Mininet on your system, please check the instructions and troubleshooting tips from mininet.org: http://mininet.org/download/.

Exercise 1: Basics of Mininet

For the first task, we will be making a walkthrough of the Mininet environment by performing some basic tasks on the platform.

Default Minimal Topology

By default, by running the command

```
$ sudo mn
```

you start the default minimal topology which consists of one OpenFlow kernel switch connected to two hosts, plus the OpenFlow reference controller. Remember that the controller can be initialized or configured to be outside the VM. This default command completes the following operations:

- Created 2 virtual hosts, each with a separate IP address.
- Created a single OpenFlow software switch in the kernel with 2 ports.
- Connected each virtual host to the switch with a virtual ethernet cable.

- Set the MAC address of each host equal to its IP.
- Configure the OpenFlow switch to connect to the local controller.

The default minimal topology is graphically represented below.

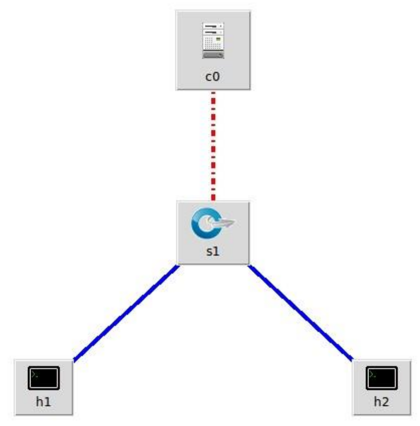

Figure 1: Default minimal topology

This command also opens the Mininet command line prompt CLI.

```
Mininet>
```

From here you can run various Mininet commands on the already established network.

To view the various commands that you can run on the Mininet CLI, run the *help* command from the CLI

```
mininet> help
```

Please check the Mininet walkthrough here http://mininet.org/walkthrough/ for more details on most of these commands and their usage scenarios.

One limitation you might notice rather quickly, is that once you exit the Mininet CLI, the initialized network with all its elements and configurations will disappear. This can be disheartening if is your first time especially if you have setup a complex topology with advanced configurations. I've come across several discussions online on how to save your instantiated network topology from the CLI, however, is hard to pin down a particular CLI based solution for this.

A more conventional approach to it would be to create a simple Python script with your custom topology and run from the host terminal or VM. You will find more information about creating such Python scripts from this same Mininet walkthrough page http://mininet.org/walkthrough/ under the section Custom Topologies. Further information can be found here also https://github.com/mininet/mininet/wiki/Introduction-to-Mininet. You will also find some custom topology Python scripts in the example folder in your Mininet directory /mininet/mininet/custom.

NB. Another approach for saving your custom topology would be to use the MiniEdit GUI, we will get to that later in another exercise. For this exercise, you will use your understanding from the Mininet walkthrough and other recommended materials to design your own custom topology using a simple Python script

Your custom topology should consist of 2 switches, 4 hosts, and 1 controller connected in the order depicted below:

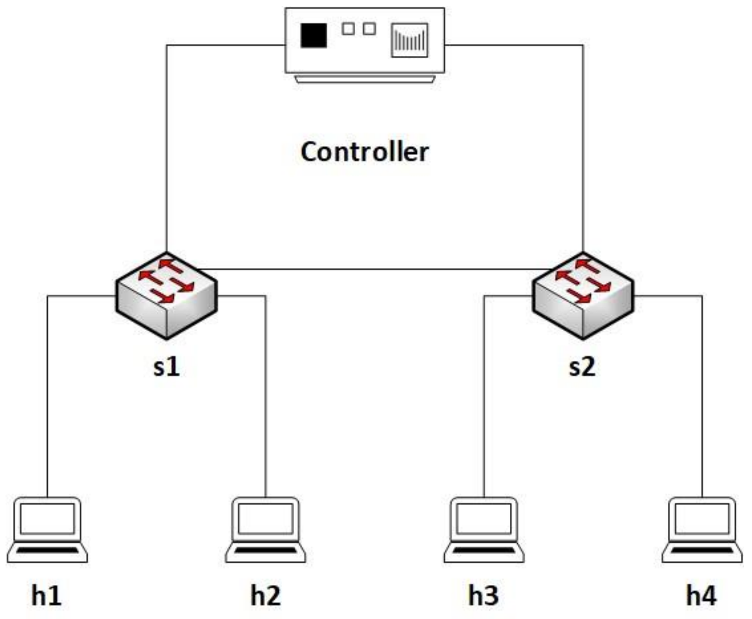

Figure 2: Custom topology for exercise 1

Your script should include functions for the following operations:

1) Dump host connections for all connected hosts
2) Test reachability of all connected hosts
3) Terminate the network

Now run the Python script and observe the outcome.

Note: If you're having any issues executing your Python script, check that you have the compatible version of Python installed. Also verify that all required dependents are properly installed.

Report:

- Include the Python script you created in the report (Fig1).
- Also include the screenshot of the script output in the report (Fig2).

Q1. What would be the outcome of the reachability test if there was no link between **s1** and **s2**? **Q2.** Would there be any difference in the outcome if you replaced the OpenFlow switches with legacy switches? Why? Why not?

The next task would be to modify the script you created in the task above by setting the following performance parameters for **h2** and **h3:**

a) Each of the two hosts gets 50% of the system CPU power

b) The links connecting both hosts to their respective switches are limited by the following parameters:

i) Bandwidth = 10Mbps ii) Delay = 5ms iii) Loss = 2%

iv) Maximum queue size = 1000

c) Test the connection speeds between **h1** and **h2**, **h1** and **h3, h2** and **h3, h1** and **h4.** d) Terminate the network.

Report:

- Include the modified Python script in the report (Fig3).
- Include the screenshot of the script output in the report (Fig4).

Q3. Explain briefly why you got the outcome you got from the speed tests by comparing the corresponding throughputs.

Q4. What difference would it make in the speed test if the links between all 4 hosts and their corresponding switches were defined using the same parameters?

Exercise 2: Integrating a remote controller

By default, Mininet comes with a custom controller which can easily be integrated to any topology by using the custom *controller()* subclasses. By using simple start() and stop() methods, Mininet will automatically start and stop your controller accordingly.

However, there are times when you may want to integrate an external controller to your Mininet topology. This external controller may reside on your VM, LAN or another machine entirely. For such scenarios, Mininet comes with a *RemoteController* class that allows for such integration. This *RemoteController* class acts as proxy for such remote controllers.

You will find more information on how to implement and use remote controllers on Mininet here https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

For this exercise, we will be using POX as our remote controller. The tasks are as follows:

1. Run the POX controller as a hub on your host machine or VM.
2. Modify the script in Exercise 1 to remotely connect to the POX controller. (Ensure the POX controller is fully running before running the Python script with your topology, otherwise you will encounter some error due to inability to detect the reference controller).
3. Also include a line that prints the flow tables using dpctl.
4. Run the modified Python script.

Report:

- Include a snapshot of the Python code showing the integration of your POX controller (Fig 5).
- Include a snapshot of the code outcome (Fig 6).

5. Now make the POX controller to act as a learning switch and run the code again. **Q5.** Compare the results of dpctl before and after converting the POX controller to a learning switch. What is the difference between the two and why?

Exercise 3: Creating custom topology using MiniEdit

For this exercise, we will be implementing same layer 2 learning function using POX controller. However, this time we will be doing that from the MiniEdit environment. As earlier introduced, MiniEdit the graphical representation of Mininet. It does not provide all features of Mininet directly from the GUI environment, however is a very effective and fast way to create a visual network topology that can be run by simple click of a run button and stopped in a similar way. It allows you to save your custom topologies as Python files that can later be run from the terminal. You can find more information on how to program with POX here http://www.brianlinkletter.com/using-the-pox-sdncontroller/. In this exercise we will also make a basic packet analysis using Wireshark. The tasks are as follows:

1) Launch the **miniedit** application by running the file **miniedit.py** from the folder **/mininet/examples**.
2) Create the topology below by dragging and dropping the required components to your workspace and linking them together.

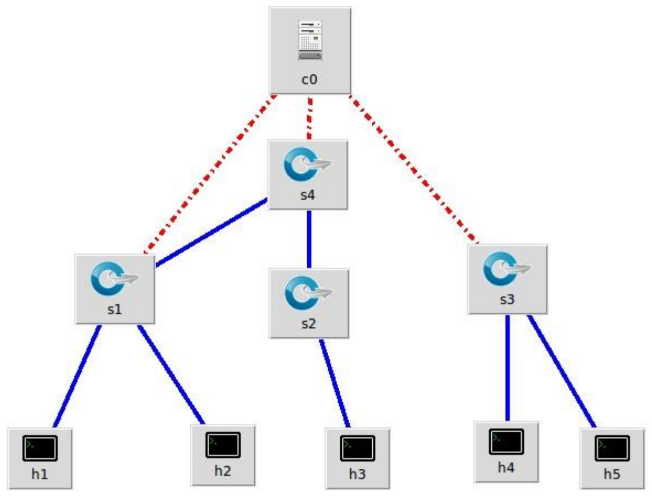

Figure 3: Custom topology for exercise 3

3) Rename the c0 to POX and change the controller type to remote controller. Leave the IP address as default (127.0.0.1) and the port number as 6633.
4) Save the MiniEdit file as PoxL2LearningBridge.mn to your mininet/examples directory. ***Report:*** Include this file to your submission (Fig 7).
5) Open a new terminal and search for the POX controller folder from your home or root directory.
6) From the POX directory, run the pox controller with the following command:

```
./pox.py forwarding.l2_learning
```

7) You should have an outcome that indicates that your POX controller is up and running.
8) On the MiniEdit application, go to **Edit>>Preferences**, and select the Start CLI option, this opens up a command line interface when you run your network, there you can issue other commands like you do on typical MiniEdit terminal. Also ensure that your OpenFlow version is set to 1.0, POX controller may not run with later versions.

Q6. Now run your network and observe the POX terminal. Briefly explain what you observed on the terminal, what elements can you identify and what do they imply?

9) From the MiniEdit command line, run the command ***pingall*** to test the reachability of the different hosts.
   Q7. Are you able to reach all nodes on the network? If not why? What modifications do you think are necessary to be able to reach all the hosts? If yes, what modifications did you have to make to reach all the hosts?
10) Check flow table on ***s4*** and dump flows.

    Report:

    - Include a screenshot of the outcome to your report (Fig 8).
    - Include a screenshot of the ping test before making any modifications (Fig 9).
11) Next open the xterm window on hosts ***h1*** and ***h3***
12) In the ***h1*** xterm window, start a Wireshark with the command, wireshark &.
13) In the ***h3*** xterm window, start a packet trace with the command tcpdump. These are two different methods of monitoring traffic on the virtual Ethernet ports of each host.
14) Then, run a ping command to send traffic between host ***h1*** and ***h3***. On the MiniEdit console window, enter the following command:

```
mininet> h1 ping h3
```

Q8. Describe what you see in the MiniEdit console, in the Wireshark window and in the host ***h3*** xterm window.

Report:

- Include screenshots of these outcomes to your report (Fig 9).

Exercise 4: Adding flow entries to the flow table

The purpose of this exercise is to introduce in a more practical way, the idea of adding flow entries to individual switch's flow table in an SDN network environment. Flow entries or flow rules as we discussed in lectures 3 and 4 are sets of predefined sets of matching parameters and actions to be taken when such parameters are matched by an arriving packet. Here we will be using the **ovs-ofctl** utility.

ovs-ofctl comes with Open vSwitch. It enables visibility and control over a single switch's flow table. By viewing flow state and flow counters it serves as a preliminary debugging tool in an SDN environment. You can read more about adding flow entries in OpenFlow tutorial on Github. The following are the tasks for this exercise:

1) Create a custom topology using the following command:

```
sudo mn --topo single,3 --mac --switch ovsk --controller remote
```

Q9. What actions will Mininet take based on this command? (List in similar order as in Exercise 1 default topology example)

2) Next open the switch terminal and run a simple command to reveal the current flow entries. **Q10.** What do you see in the flow entries? Why are they the way they currently are?
3) Next run the following command

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

Q11. What outcome do you see? What is the expected outcome? Why is it the way it currently is?

4) Now, go back to the Mininet console and try to ping h2 from h1.

Q12. Do you get any replies? Why? Why not?

5) Next task is to manually install the necessary flows to enable packet exchange between all nodes on the network.

Q13. What commands did you use? (Please include all the commands you used in a bullet list) 6) Run the command to reveal flow entries again.

Q14. What outcome do you see? Compare this outcome to the initial outcome in **Q11,** Why is it the way it currently is?

7) Now, go back to the Mininet console and try to ping h2 from h1. **Q12.**
Do you get any replies? Why? Why not?

Report:

- Include a screenshot of the flow entries to the report (Fig 10).
- Include a screenshot of ping test to the report (Fig 11).