Lab Exercise 3 (part A) Introduction to Mininet

Objectives:

- Learn the basic commands in Mininet
- Learn how to create basic network topologies in Mininet
- Learn Mininet API

Marks:

This exercise forms the first part of lab exercise 3. The second part (part B) will be part of next week's lab. Both parts together will comprise 15 marks. Only selected exercises will be marked. However, students must submit answers for all exercises.

Deadline:

Before your scheduled lab next week. So you get one week to work on this lab. For example, if you go to the Monday 12 noon lab, then your submission is due before 12 noon on the following Monday. You can submit as many times as you wish before the deadline. A later submission will override the earlier submission, so make sure you submit the correct file. Do not leave until the last moment to submit, as there may be technical or communications error and you will not have time to rectify it.

Late Penalty:

Late penalty will be applied as follows:

- 1 day after deadline: 20% reduction
- 2 days after deadline: 40% reduction
- 3 days after deadline: 60% reduction
- 4 or more days late: NOT accepted

Note that the above penalty is applied to your final mark. For example, if you submit your lab work 2 days late and your score on the lab is 8, then your final mark will be 8-3.2 (40% penalty) = 4.8.

Submission Instructions:

Submit a PDF document **lab3a.pdf** with answers to all questions for all exercises. Include all supporting documents such as topology files for Questions 5 and 6(*.py). Create a tar archive of all files called **lab3a.tar**. Submit the archive using give. Click on the submission link at the top of the page. Max file size for submission is 3MB.

Original Work Only:

You are strongly encouraged to discuss the questions with other students in your lab. However, each student must submit his or her own work. You may refer to the reference material and also conduct your own research to answer the questions.

Notation: In the examples below, we have used the `$` sign to represent Linux commands that should be typed at the shell prompt, `mininet>` to show Mininet commands that should be typed at Mininet's CLI (command line interface), and `#` to show Linux commands that are typed at a root shell prompt. In each case, you should only type the name of the command and then press return. The actual prompt may look very different on your computer (e.g. it may contain the computer's hostname, or your username, or the current directory name). The commands that **you** are supposed to type are in **this bold font**.

Part 1: Basic Commands in Mininet

In the first step, we start a simple network topology by running the following command:

```
$ sudo mn
```

The above command creates the default topology in Mininet (known as the *minimal* topology), which includes one OpenFlow[1] kernel switch connected to two hosts, plus the OpenFlow reference. This topology could also be mentioned on the command line using option *--topo=minimal* with the *mn* command. Once you run the above command, all four entities (two hosts, one switch, and one controller) are running in the VM and the Mininet CLI comes up. You can run the *help* command to see the list of commands in the CLI. In the rest of this section, you will practice the basic commands in Mininet.

Display nodes:

```
mininet> nodes
```

For more detailed information about the nodes in the network, use *dump*.

Display topology:

```
mininet> net
```

Simple commands within a node:

If the first string typed into the Mininet CLI is a host, switch or controller name, the corresponding command is executed on that node. For example, run the following command on host *h1* to show its network interfaces.

```
mininet> h1 ifconfig -a
```

[1] OpenFlow is communication protocol between switches and controllers. We do not need to know the details of the OpenFlow protocol in this course.

In the above command, you can replace *h1* with *s1* to see the details of the switch's interfaces. Also, for a list of processes in a host you can run *h1 ps –a*.

Note that a switch is a layer 2 packet switch (also known as a bridge). We will examine switches in details when we study about the link layer.

Question 1: Draw the Mininet minimal topology and include the IP and MAC addresses for hosts and the switch. To which switch ports are the hosts and controller connected?

Test connectivity between hosts:

In order to ping host 1 from host 2, run the following command:

```
mininet> h1 ping –c5 h2
```

Question 2: Examine the ping time for all five tries in the above command. What is the ping time for each try?

Question 3: You should see that the ping time for the first try is larger than the next tries. How do you explain this observation?

A simple way to test the connectivity of all nodes in the Mininet CLI is to run the *pingall* command, which does an all-pairs *ping*.

XTerm display

To display an xterm for host h1, run the following command. This will be useful to run commands within a host or switch.

```
mininet> xterm h1
```

Python Interpreter

If the first part of a command in Mininet is *py*, then the CLI interprets the command as a Python command. This helps you to manipulate the network objects, such as hosts, switches, and controllers, using the CLI. Here is an example which displays the IP address of host *h1*.

```
mininet> py h1.IP()
```

Exit and clean up

For exiting from the Mininet CLI, run command *exit*. It is strongly recommended to clean up after exiting by running command *sudo mn –c*.

NOTE: PLEASE REMEMBER TO MOUNT YOUR CSE HOME DIRECTORY IN THE VM AND TO SAVE YOUR WORK IN YOUR HOME DIRECTORY BEFORE QUITING.

Part 2: Custom Topologies

Mininet supports a simple Python API to create custom network topologies. You can create your custom topology by writing a few lines of Python code. For example, Figure 1 shows a topology with two switches and four hosts. Figure 2 implements the topology using the Mininet Python API. The code consists of a class, named MyFirstTopo which extends the Topo class provided by the API. We suggest that you SSH into your mininet VM so that you can use a GUI text editor such as gedit or emacs (see instructions in the Mininet Introduction document). Save the source code in a file, named *myfirsttopo.py* in your (mounted) CSE home directory. You can download the source code from here: myfirsttopo.py. Change the working directory to your CSE home directory and type:

```
mininet> sudo mn --custom myfirsttopo.py --topo myfirsttopo
```

Question 4: Draw Figure 1 and label the IP and MAC address for the hosts. Also label the interface names and MAC address of the switch ports that are connected to the hosts.

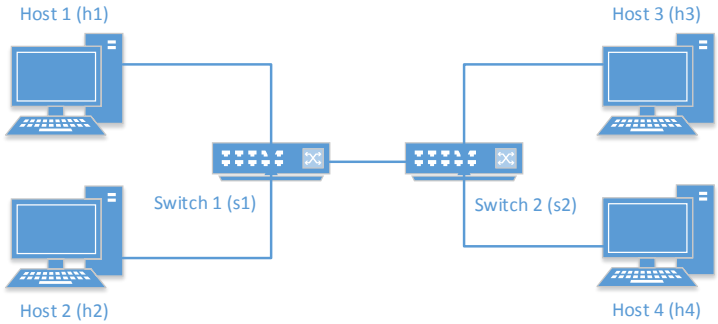

Figure 1 A sample network topology (myfirsttopo.py)

```
1.  from mininet.topo import Topo
2.
3.  class MyFirstTopo( Topo ):
4.      "Simple topology example."
5.      def __init__( self ):
6.          "Create custom topo."
7.          # Initialize topology
8.          Topo.__init__( self )
9.          # Add hosts and switches
10.         h1 = self.addHost( 'h1' )
11.         h2 = self.addHost( 'h2' )
12.         h3 = self.addHost( 'h3' )
13.         h4 = self.addHost( 'h4' )
14.         leftSwitch = self.addSwitch( 's1' )
15.         rightSwitch = self.addSwitch( 's2' )
16.         # Add links
17.         self.addLink( h1, leftSwitch )
18.         self.addLink( h2, leftSwitch )
19.         self.addLink( leftSwitch, rightSwitch )
20.         self.addLink( rightSwitch, h3 )
21.         self.addLink( rightSwitch, h4 )
22.
23. topos = { 'myfirsttopo': ( lambda: MyFirstTopo() ) }
```

Figure 2 Source code for the custom topology presented in Figure 1.

If you want to run a quick test without entering the CLI, you can use option *--test* in Mininet, as follow:

```
mininet> sudo mn --custom myfirsttopo.py --topo myfirsttopo --test pingall
```

Now assume that we want to accomplish the following:

1. Create the topology shown in Figure 1
2. Start the network
3. Show the detailed connectivity of all nodes in the network
4. Test the connectivity by pinging all nodes

We can automate this sequence of operations by writing a Python script. This helps us to automate network experiments. To this end, we extend the previous source code, as shown in Figure 3. Download the source code from here: myfirstexpr.py.

```
1.  #!/usr/bin/python
2.
3.  from mininet.topo import Topo
4.  from mininet.net import Mininet
5.  from mininet.util import dumpNodeConnections
6.  from mininet.log import setLogLevel
7.
8.  class MyFirstTopo( Topo ):
9.      "Simple topology example."
10.     def __init__( self ):
11.         "Create custom topo."
12.         # Initialize topology
13.         Topo.__init__( self )
14.         # Add hosts and switches
15.         h1 = self.addHost( 'h1' )
16.         h2 = self.addHost( 'h2' )
17.         h3 = self.addHost( 'h3' )
18.         h4 = self.addHost( 'h4' )
19.         leftSwitch = self.addSwitch( 's1' )
20.         rightSwitch = self.addSwitch( 's2' )
21.         # Add links
22.         self.addLink( h1, leftSwitch )
23.         self.addLink( h2, leftSwitch )
24.         self.addLink( leftSwitch, rightSwitch )
25.         self.addLink( rightSwitch, h3 )
26.         self.addLink( rightSwitch, h4 )
27.
28. def runExperiment():
29.     "Create and test a simple experiment"
30.     topo = MyFirstTopo( )
31.     net = Mininet(topo)
32.     net.start()
33.     print "Dumping host connections"
34.     dumpNodeConnections(net.hosts)
35.     print "Testing network connectivity"
36.     net.pingAll()
37.     net.stop()
38.
39. if __name__ == '__main__':
40.     # Tell mininet to print useful information
41.     setLogLevel('info')
42.     runExperiment()
```

Figure 3 Source code for an experiment on the custom topology.

Now download the source code and save it in a file named *myfirstexpr.py* and run the following command:

```
mininet> sudo python myfirstexpr.py
```

Remember to clean up (sudo mn –c) and also to save your work in your CSE home directory before moving forward.

Part 3: Mininet Python Classes

The Mininet Python API consists of a number of Python classes, such as Topo, Mininet, Host, Switch, Link and their subclasses. The API is built at three primary levels which provide different level of abstractions for customising the network topologies. Here is a brief description of the levels:

- Low-level API: This API includes the base node and link classes such as Host, Switch and Link. You can directly instantiate some objects from these classes to create a network.
- Mid-level API: This API consists of the Mininet class which provides a number of methods, such as addHost(), addSwitch(), and addLink(), for creating a network.
- High-level API: This API consists of the Topo class, which provides the ability to create reusable, parameterised topology templates. These templates can be passed to the *mn* command (via the *--custom* option as in Part 2 above) and used from the command line.

Figure 2 and Figure 3 are two examples that show how to use the high-level API. As you can see, the high-level API provides an object-oriented framework by presenting the *Topo* class. Figure 4 shows how to use the low-level API for creating the custom topology presented in Figure 1. Download the source code from here: myfirsttopo_lowlevel.py. Observe that, with the low-level API we create all the objects and connect the topology manually. You can test this code by typing the following:

```
mininet> sudo python myfirsttopo_lowlevel.py
```

You can find the detailed documentation of all classes provided in the Mininet Python API in http://mininet.org/api/annotated.html.

```
#!/usr/bin/python

from mininet.node import Host,OVSSwitch,Controller
from mininet.link import Link

h1 = Host( 'h1' )
h2 = Host( 'h2' )
h3 = Host( 'h3' )
h4 = Host( 'h4' )
s1 = OVSSwitch( 's1', inNamespace=False )
s2 = OVSSwitch( 's2', inNamespace=False )
c0 = Controller( 'c0', inNamespace=False )
Link( h1, s1 )
Link( h2, s1 )
Link( h3, s2 )
Link( h4, s2 )
Link( s1, s2 )
h1.setIP( '10.0.0.1/24' )
h2.setIP( '10.0.0.2/24' )
h3.setIP( '10.0.0.3/24' )
h4.setIP( '10.0.0.4/24' )
c0.start()
s1.start( [ c0 ] )
s2.start( [ c0 ] )
print h1.IP
print h2.IP
print h3.IP
print h4.IP
print 'Pinging ...'
print h1.cmd( 'ping -c3 ', h2.IP() )
print h1.cmd( 'ping -c3 ', h3.IP() )
s1.stop()
s2.stop()
c0.stop()
```

Figure 4 Source code for the topology presented in Figure 1 using low-level API.

Remember to clean up (sudo mn –c) and also to save your work in your CSE home directory before moving forward.

Part 4: Simple Tree Topology

Many data centres use a tree-like network topology. End-hosts (i.e. servers) connect to top-of-rack switches, called *edge switches* and form the leaves of the tree; one or more *core* switches form the root of the tree; and one or more *aggregation* switches form the intermediate nodes of the tree. In a *simple tree* topology, there is only one core switch connected to n aggregation switches; each aggregation switch is connected to n edge switches; each edges switches is connected to n hosts (servers). Figure 5 shows a simple tree topology where n $=$2.

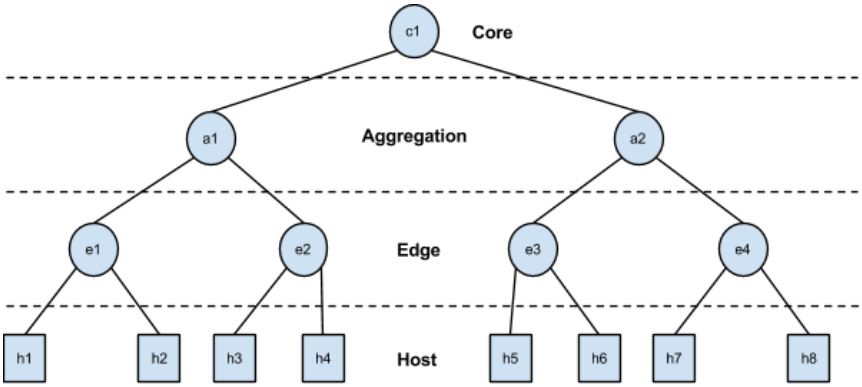

Figure 5 Simple-tree topology with $n = 2$.

Question 5: Write a Python program to use the high-level API and create a simple-tree data centre topology. You must follow the naming system as shown in Figure 6. The program gets the value of n as an input. You must organise all the source code in a single source file, name it *simpletree_highlevel.py*, and include this file in the submission archive.

Question 6: Repeat the previous question but now use the low-level API. You must organise all the source code in a single source file, name it *simpletree_lowlevel.py*, and include this file in the submission archive.

Question 7: Use your program (either high-level or low-level) to create a simple-tree with $n = 2$, as shown in Figure 5. Check the connectivity of all nodes in the network and write your observations. Now stop the core switch using command *switch c1 stop*, check the connectivity of all hosts and report your observations. How do you explain your findings? Write up your answer in the report.

Part 5: Useful Resources

- Introduction to Mininet: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet
- Mininet Walkthrough: http://mininet.org/walkthrough/
- Mininet Python API Reference Manual: http://mininet.org/api/annotated.html