Software-Defined Networks Lab

Experiment 1 – Creating SDN network topologies in Mininet

Release Date: March 13, 2018

1. Objective:

In this experiment, you will learn:

- How to get started with OpenFlow using Mininet
- How to create OpenFlow topologies in Mininet
- How to write test scripts in Mininet

2. Tutorial:

Note:

- **It's strongly recommended to run the experiments in virtual machine, i.e. Oracle VM VirtualBox[1], or VMware Workstation[2].**
- **Make sure you have Internet connection.**
- **Make sure you have super user permission.**

In the example, we will use Ubuntu 16.04[3] as our Operating System, but you can also choose other Linux OS.

(A) Setting up the Floodlight controller

In this experiment, we will use Floodlight as our SDN controller, you can see more information from their website: http://www.projectfloodlight.org/

[1] https://www.virtualbox.org/
[2] https://www.vmware.com/
[3] http://releases.ubuntu.com/16.04/

1. Open up the terminal window, type the commands below to prepare your building environment and install some dependencies (Note: the dollar sign ($) in front of the command is just the prompt of your command line, you don't need to type that symbol)

```
$ sudo apt-get update

$ sudo apt-get install build-essential default-jdk ant
python-dev eclipse git
```

2. Download Floodlight project from GitHub[4]

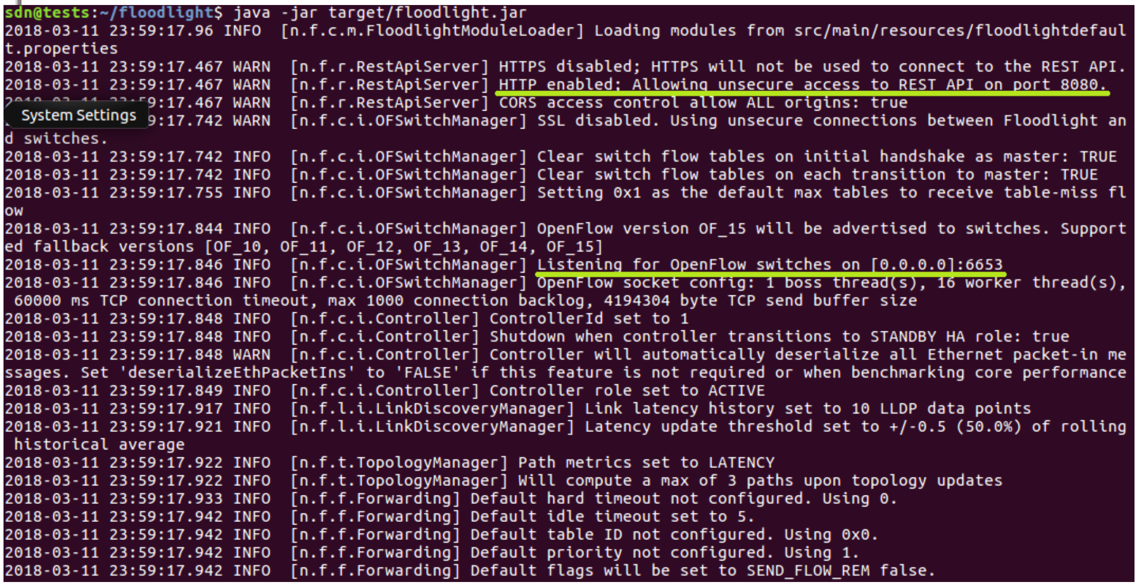
```
$ git clone https://github.com/floodlight/floodlight.git
```

3. Build Floodlight

```
$ cd floodlight

$ ant

$ sudo mkdir -p /var/lib/floodlight

$ sudo chmod 777 /var/lib/floodlight
```

4. Use the following command to run Floodlight (Note: press Ctrl-C to stop the program)

```
$ java -jar target/floodlight.jar
```



```
sdn@tests:~/floodlight$ java -jar target/floodlight.jar
2018-03-11 23:59:17.96 INFO  [n.f.c.m.FloodlightModuleLoader] Loading modules from src/main/resources/floodlightdefaul
t.properties
2018-03-11 23:59:17.467 WARN  [n.f.r.RestApiServer] HTTPS disabled; HTTPS will not be used to connect to the REST API.
2018-03-11 23:59:17.467 WARN  [n.f.r.RestApiServer] HTTP enabled; Allowing unsecure access to REST API on port 8080.
2018-03-11 23:59:17.467 WARN  [n.f.r.RestApiServer] CORS access control allow ALL origins: true
[illegible]9:17.742 WARN  [n.f.c.i.OFSwitchManager] SSL disabled. Using unsecure connections between Floodlight an
d switches.
2018-03-11 23:59:17.742 INFO  [n.f.c.i.OFSwitchManager] Clear switch flow tables on initial handshake as master: TRUE
2018-03-11 23:59:17.742 INFO  [n.f.c.i.OFSwitchManager] Clear switch flow tables on each transition to master: TRUE
2018-03-11 23:59:17.755 INFO  [n.f.c.i.OFSwitchManager] Setting 0x1 as the default max tables to receive table-miss fl
ow
2018-03-11 23:59:17.844 INFO  [n.f.c.i.OFSwitchManager] OpenFlow version OF_15 will be advertised to switches. Support
ed fallback versions [OF_10, OF_11, OF_12, OF_13, OF_14, OF_15]
2018-03-11 23:59:17.846 INFO  [n.f.c.i.OFSwitchManager] Listening for OpenFlow switches on [0.0.0.0]:6653
2018-03-11 23:59:17.846 INFO  [n.f.c.i.OFSwitchManager] OpenFlow socket config: 1 boss thread(s), 16 worker thread(s),
 60000 ms TCP connection timeout, max 1000 connection backlog, 4194304 byte TCP send buffer size
2018-03-11 23:59:17.848 INFO  [n.f.c.i.Controller] ControllerId set to 1
2018-03-11 23:59:17.848 INFO  [n.f.c.i.Controller] Shutdown when controller transitions to STANDBY HA role: true
2018-03-11 23:59:17.848 WARN  [n.f.c.i.Controller] Controller will automatically deserialize all Ethernet packet-in me
ssages. Set 'deserializeEthPacketIns' to 'FALSE' if this feature is not required or when benchmarking core performance
2018-03-11 23:59:17.849 INFO  [n.f.c.i.Controller] Controller role set to ACTIVE
2018-03-11 23:59:17.917 INFO  [n.f.l.i.LinkDiscoveryManager] Link latency history set to 10 LLDP data points
2018-03-11 23:59:17.921 INFO  [n.f.l.i.LinkDiscoveryManager] Latency update threshold set to +/-0.5 (50.0%) of rolling
 historical average
2018-03-11 23:59:17.922 INFO  [n.f.t.TopologyManager] Path metrics set to LATENCY
2018-03-11 23:59:17.922 INFO  [n.f.t.TopologyManager] Will compute a max of 3 paths upon topology updates
2018-03-11 23:59:17.933 INFO  [n.f.f.Forwarding] Default hard timeout not configured. Using 0.
2018-03-11 23:59:17.942 INFO  [n.f.f.Forwarding] Default idle timeout set to 5.
2018-03-11 23:59:17.942 INFO  [n.f.f.Forwarding] Default table ID not configured. Using 0x0.
2018-03-11 23:59:17.942 INFO  [n.f.f.Forwarding] Default priority not configured. Using 1.
2018-03-11 23:59:17.942 INFO  [n.f.f.Forwarding] Default flags will be set to SEND_FLOW_REM false.
```

(B) Setting up the Mininet environment

Mininet creates a realistic virtual network on a single machine, which is useful for development or research. You can visit their website, and follow their tutorial to learn

[4] https://github.com/

more about Mininet: http://mininet.org/

1. Download Mininet project from GitHub

```
$ git clone https://github.com/mininet/mininet.git
```

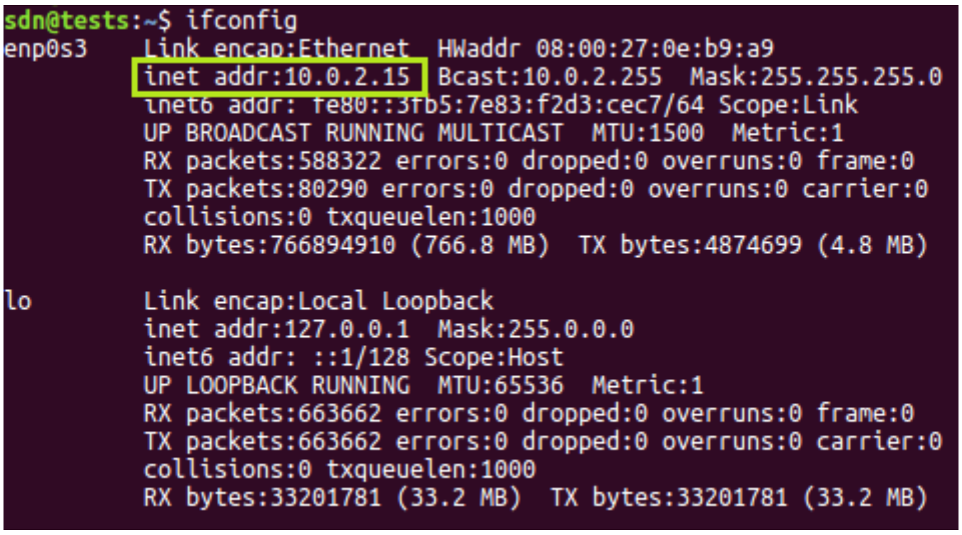
2. Install Mininet

```
$ ./mininet/util/install.sh -a
```

3. Once completed, you can issue the following command to see if it's working

```
$ sudo mn
```

4. If you type the command above, Mininet will create a very simple topology which contains one switch and two hosts connecting to that switch, and then you will be brought into the Mininet command prompt, you can type <Tab> twice to see available commands, you can use `pingall` command see if the hosts can ping each other. You can find out more on Mininet walkthrough: http://mininet.org/walkthrough/
5. We want Mininet to use our Floodlight controller. First, use `ifconfig` command in the floodlight VM to see what IP address the floodlight controller is using. In this example, it uses 10.0.2.15. (Which means the controller can be run on different host from Mininet, you can install in other machine, as long as they can reach each other.)



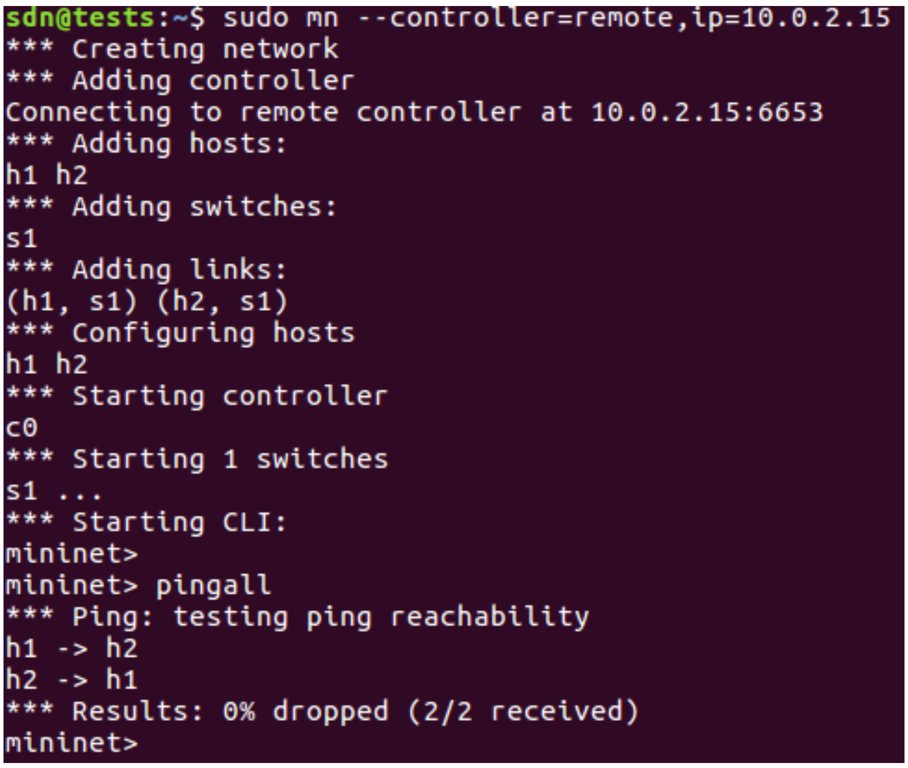
```
sdn@tests:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:0e:b9:a9
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::3fb5:7e83:f2d3:cec7/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:588322 errors:0 dropped:0 overruns:0 frame:0
          TX packets:80290 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:766894910 (766.8 MB)  TX bytes:4874699 (4.8 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:663662 errors:0 dropped:0 overruns:0 frame:0
          TX packets:663662 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:33201781 (33.2 MB)  TX bytes:33201781 (33.2 MB)
```

6. If you cannot use `ifconfig` command, you can try using the following command instead.

```
$ ip address show
```

7. After starting the floodlight controller, you can then use the following command to create the default topology and connect to floodlight controller.

```
$ sudo mn --controller=remote,ip=10.0.2.15
```



```
sdn@tests:~$ sudo mn --controller=remote,ip=10.0.2.15
*** Creating network
*** Adding controller
Connecting to remote controller at 10.0.2.15:6653
*** 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>
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

8. Sometimes you might have some problem setting up the Mininet topology, you can try cleaning up:

```
$ sudo mn -c
```

(C) Create custom topologies

Mininet provides python API (Python2) to setup topologies or create testing scripts. There's one simple custom topology example, which is placed at `` `mininet/custom/topo-2sw-2host.py` ``. If you follow this doc and use `` `git clone` `` method to download Mininet, you should see this file.

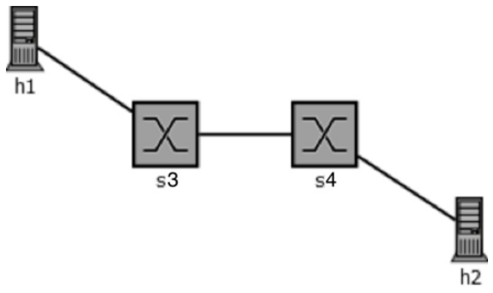
The content of the script is shown below:

```python
"""Custom topology example

Two directly connected switches plus a host for each switch:

   host --- switch --- switch --- host

Adding the 'topos' dict with a key/value pair to generate our newly defined
topology enables one to pass in '--topo=mytopo' from the command line.
"""

from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        leftSwitch = self.addSwitch( 's3' )
        rightSwitch = self.addSwitch( 's4' )

        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( leftSwitch, rightSwitch )
        self.addLink( rightSwitch, rightHost )


topos = { 'mytopo': ( lambda: MyTopo() ) }
```

We can find three useful commands in this simple script: `addHost`, `addSwitch`, `addLink`.
As their names indicate, these commands are used to add host, switch, and link to the topology, respectively.
We can use the following command to create this topology in Mininet.

```
$ sudo mn --custom mininet/custom/topo-2sw-2host.py --topo mytopo
```

The topology looks like:



Each link has unlimited bandwidth in this example.

The `--custom` argument means "use custom topology given in the following argument", and the `--topo mytopo` command indicates "use topology `mytopo` from the directory `topos` in the script". That means we can define many different topologies in one Python script, and use `--topo` to choose which one to create.

(D) Setting up properties of links

We can set several link properties to emulate different network conditions.
We can add many optional parameters to the `addLink` command, such as `bw`, `delay`, etc.
For example, the following line will add a link between the two switches with a bandwidth of 10Mb/s, link delay of 5ms, and 10% packet loss rate:

```
self.addLink(leftSw, centerSw, bw=10, delay='5ms', loss=10)
```

The table below lists the available parameters:

bw	delay	jitter	loss
disable_gro	speedup	use_hfsc	use_tbf
latency_ms	enable_ecn	enable_red	max_queue_size

Or you can see the API documentation for more detailed info:
http://mininet.org/api/classmininet_1_1link_1_1TCIntf.html

To enable this features, you will need to add `--link=tc` in the command line, which means "use Traffic Control link".

```
$ sudo mn --custom path/to/topo.py --link=tc --topo mytopo
```

(E) Write a Mininet testing script

We've learned how to create a custom topology and start the CLI of Mininet. But if we have many things to test, we can use the Mininet Python API to write automatic testing scripts.

The following is an example testing script:

```python
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.link import TCLink
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel

class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        leftSwitch = self.addSwitch( 's3' )
        rightSwitch = self.addSwitch( 's4' )

        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( leftSwitch, rightSwitch )
        self.addLink( rightSwitch, rightHost )

def perfTest():
    "Create network and run simple performance test"
    topo = MyTopo()
    net = Mininet(topo=topo, link=TCLink)
    net.start() # Start Mininet
    print('Dumping host connections')
    dumpNodeConnections(net.hosts)
    print('Testing network connectivity')
    net.pingAll()   # Do pingall test
    print('Testing bandwidth between h1 and h2')
    h1, h2 = net.get('h1', 'h2')     # Get nodes by name
    net.iperf((h1, h2))     # Do iperf test
    net.stop()  # Stop Mininet

if __name__ == '__main__':
    # Set log level
    # you can use 'info', 'warning', 'critical', 'error', 'debug', 'output'
    setLogLevel('info')
    perfTest()
```

Use `sudo python <testing_file>` to execute the testing script. This script will set up the Mininet topology, and use `pingall` and `iperf` to test the network, and then stop the network.

- Mininet commands:

Use the following line to use a remote controller. Remember to import `` `RemoteController` `` from `` `mininet.node` ``

```
from mininet.node import RemoteController

net.addController( 'myController',
controller=RemoteController, ip=<controller_IP> )
```

Line 32:

```
net = Mininet( topo=topo, link=TCLink )
```

creates a Mininet object using the topology "`topo`" with "`tclink`" and assign to the variable "`net`".

You can use the following line to let Mininet not run the built-in controller, and then you can add your controller.

```
net = Mininet( topo=topo, link=TCLink, controller=None )
```

Besides `pingall` and `iperf` test, you can view more useful tools from:
http://mininet.org/api/classmininet_1_1net_1_1Mininet.html

- Single host commands:

Use `` `cmdPrint()` `` to execute a command and display output on the specified node. For example, the following line will let h1 ping h2 10 times, and display results on the screen.

```
h1.cmdPrint( 'ping -c 10 ' + h2.IP() )
```

Use `` `popen( '<command> > <filename>', shell=True )` `` to redirect background output to a file. For example, the following line will let h2 ping h1 10 times, and redirect the results to file "`opfile`".

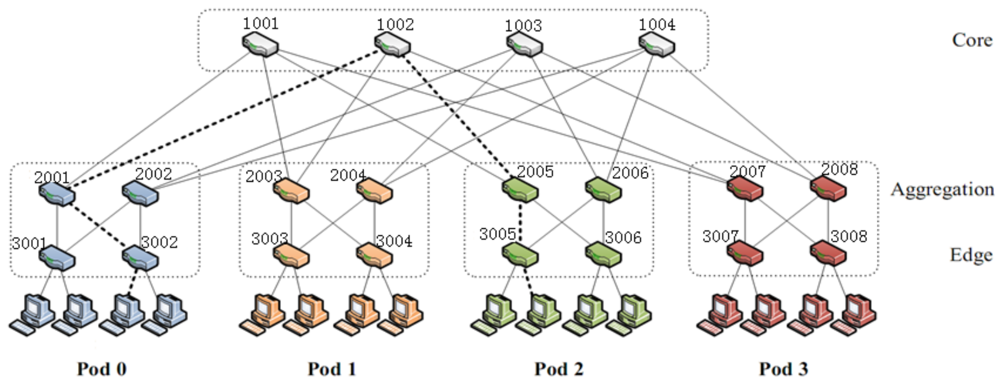
```
h2.popen( 'ping -c 10 ' + h1.IP() + '>opfile', shell=True )
```

- Useful tip:

You can use `` `xterm <host>` `` in Mininet command line to open an xterm window for the specified node. Then you can use the `` `ping` `` or `` `iperf` `` command directly and not need to use the cmdPrint().

3. Experiment:

In this experiment, we will use Fat Tree[5] topology, as an example shown below.



In the picture above, there are 4 pods and 4 core switches, each pod has 4 hosts, 2 edge switches and 2 aggregation switches. Each aggregation switch has 2 paths to other pods, so this topology has good fault tolerance capabilities.

Your job is to:

1. Create a Fat Tree topology as shown above in Mininet, with
   a. Each link in each pod has a 100 Mbps bandwidth.
   b. Each link between core switches and aggregation switches has a 1000 Mbps bandwidth and 2% packet loss rate.
2. Write a testing script to test the topology you created and do the following tests:
   a. Connect to the floodlight remote controller
   b. `dumpNodeConnection` function call
   c. `pingFull` function call
   d. Instead of doing iperf test with the `iperf` command provided directly by Mininet, choose one host in both pod 0, and pod 3 as the servers, and use `popen` to execute `iperf -s -u -i 1` on each of them.
   e. Choose one host in pod 0 as the client and use `cmdPrint` to execute `iperf -c <server IP> -u -t 10 -i 1 -b 100m` to connect to each server. Measure and report their achieved throughputs and experienced packet loss rates, respectively.
   f. Observe the reports on the client and servers.

[5] See page 8 of https://www.slideshare.net/AnkitaMahajan2/fattree-a-scalable-fauult-tolerant

4. Submission:

1. Name your testing script with <studentID>.py, e.g. 0556519.py
2. Upload the script onto e3.
3. Due date: 2018/03/27

5. Notes:

- Depending on your CPU and memory constrain, your Mininet topology might not be available immediately after you set up, it might take some times for the links to be ready.

6. Useful Links:

Mininet tutorials and API references
[1] http://mininet.org/walkthrough/
[2] https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#wiki-working
[3] http://mininet.org/api/classmininet_1_1link_1_1TCIntf.html
[4] http://mininet.org/api/classmininet_1_1net_1_1Mininet.html

Iperf
[5] http://openmaniak.com/iperf.php
[6] http://benjr.tw/3030