Laboratory 3 Mininet network emulator

Part 1: Introduction to Mininet

1.1 Mininet features

Mininet is a network emulator, or perhaps more precisely a network emulation orchestration system. It runs a collection of end-hosts, switches, routers, and links on a single Linux kernel. It uses lightweight virtualization to make a single system look like a complete network, running the same kernel, system, and user code. A Mininet host behaves just like a real machine; you can `ssh` into it (if you start up `sshd` and bridge the network to your host) and run arbitrary programs (including anything that is installed on the underlying Linux system.) The programs you run can send packets through what seems like a real Ethernet interface, with a given link speed and delay. Packets get processed by what looks like a real Ethernet switch, router, or middlebox, with a given amount of queueing. When two programs, like an `iperf` client and server, communicate through Mininet, the measured performance should match that of two (slower) native machines [https://github.com/mininet/mininet/wiki/Introduction-to-Mininet].

Mininet features:

- It's **fast** - starting up a simple network takes just a few seconds. This means that your run-edit-debug loop can be very quick.
- You can **create custom topologies**: a single switch, larger Internet-like topologies, the Stanford backbone, a data center, or anything else.
- You can **run real programs**: anything that runs on Linux is available for you to run, from web servers to TCP window monitoring tools to Wireshark.
- You can **customize packet forwarding**: Mininet's switches are programmable using the OpenFlow protocol. Custom Software-Defined Network designs that run in Mininet can easily be transferred to hardware OpenFlow switches for line-rate packet forwarding.
- You can **share and replicate results**: anyone with a computer can run your code once you've packaged it up.
- You can **use it easily**: you can create and run Mininet experiments by writing simple (or complex if necessary) Python scripts.
- Mininet is an **open source project**, so you are encouraged to examine its source code on https://github.com/mininet, modify it, fix bugs, file issues/feature requests, and submit patches/pull requests. You may also edit this documentation to fix any errors or add clarifications or additional information.

1.2 Mininet basic commands.

Display Mininet CLI commands:
```
mininet> help
```

Display nodes:
```
mininet> nodes
```

Display links:
```
mininet> net
```

Dump information about all nodes:
```
mininet> dump
```

Part 2. Practice

Step 1: Starting Mininet

1) Open VirtulaBox virtualization environment and start the Mininet virtual machine. Login to the Mininet machine using username **`mininet`** and password **`mininet`**.

2) To view control traffic using the OpenFlow Wireshark dissector, first open wireshark in the background:

```
$ sudo wireshark &
```

In the Wireshark filter box, enter the **`of`** value for the filter, then click `Apply`. In Wireshark, click `Capture`, then `Interfaces`, then select `Start` on the `loopback interface (lo)`.For now, there should be no OpenFlow packets displayed in the main window.

2) Start the Mininet program with the default topology with the command:

```
$ sudo mn -h
```

This command will start Mininet with a topology based on a switch with two hosts attached to it and a basic SDN controller – Figure 1. The Mininet CLI will come up:

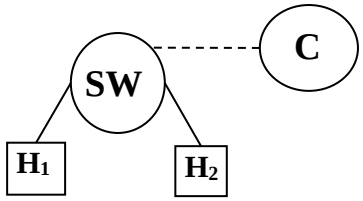

Figure 1

mininet>

Q1. Analyze the network topology using the basic Mininet commands.

Besides the basic Mininet commands system commands can be used on any of the virtual machines representing the nodes in the topology – `h1`, `h2` and `s1` in the basic topology. Commands like `ifconfig`, `ping`, `ps`, `iperf`, etc., can be used on the virtual nodes. Examples:

To check details about the interfaces on a host or switch:

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

```
mininet> s1 ifconfig -a
```

You should see the host's `h1-eth0` and `loopback (lo)` interfaces. Note that this interface (`h1-eth0`) is not seen by the primary Linux system when `ifconfig` is run, because it is specific to the network namespace of the host process.

In contrast, the switch by default runs in the root network namespace, so running a command on the "switch" is the same as running it from a regular terminal.

Q2. Write the IP addresses for the hosts `h1` and `h2` find out using the `ifconfig` command.

Note that **only the network** is virtualized; each host process sees the same set of processes and directories. For example, print the process list from a host process:

```
mininet> h1 ps -a
```

This should be the same as that seen by the root network namespace:

```
mininet> s1 ps -a
```

Step 2: Test connectivity between hosts

Now, verify that you can `ping` from host 0 to host 1:

```
mininet> h1 ping -c 1 h2
```

If a string appears later in the command with a node name, that node name is replaced by its IP address; this happened for `h2`.

You should see OpenFlow control traffic in Wireshark. The first host ARPs for the MAC address of the second, which causes a `packet_in` message to go to the controller. The controller then sends a `packet_out` message to flood the broadcast packet to other ports on the switch (in this example, the only other data port). The second host sees the ARP request and sends a reply. This reply goes to the controller, which sends it to the first host and pushes down a flow entry.

Now the first host knows the MAC address of the second, and can send its `ping` via an `ICMP Echo Request`. This request, along with its corresponding reply from the second host, both go the controller and result in a flow entry pushed down (along with the actual packets getting sent out).

Q3. Write the RTT values obtained with the two `ping` commands.

Repeat the last ping:

```
mininet> h1 ping -c 1 h2
```

You should see a much lower `ping` time for the second try. A flow entry covering ICMP `ping` traffic was previously installed in the switch, so no control traffic was generated, and the packets immediately pass through the switch.

Q4. Write the RTT values obtained with the two `ping` commands. Rerun the ping command, this time sending 40 packets. Notice what happens with the RTT value.

An easier way to run this test is to use the Mininet CLI built-in pingall command, which does an all-pairs ping:

```
mininet> pingall
```

Run a simple web server and client

Remember that `ping` isn't the only command you can run on a host! Mininet hosts can run any command or application that is available to the underlying Linux system (or VM) and its file system. You can also enter any bash command, including job control (&, jobs, kill, etc.)

Next, try starting a simple HTTP server on h1, making a request from h2, then shutting down the web server:

```
mininet> h1 python -m SimpleHTTPServer 80 &

mininet> h2 wget -O - h1s

...

mininet> h1 kill %python
```

To end the Mininet session exit the CLI:

```
mininet> exit
```

If Mininet crashes for some reason, clean it up:

```
$ sudo mn -c
```

Use `iperf` to exchange ip traffic between hosts

Repeat the experiment with the HTTP server using `iperf` server and client instead. Consult the web to find out how to configure `iperf` (E.g. https://openmaniak.com/iperf.php). Run the `iperf` server on h1 and the `iperf` client on h2.

Using `iperf` with TCP traffic measure the link capacity between host 1 and host 2.

Q5. Write the value for the link capacity between `h1` and `h2`.

Mininet 2.0 allows you to set link parameters, and these can be set automatically from the command line. Before launching the next command, you should stop the running Mininet first.

```
$ sudo mn --link tc,bw=10,delay=30ms
```

Measure the link capacity and link delay (RTT) between `h1` and `h2` using `iperf` and `ping` commands. Use ping with multiple packets to get the correct link delay. Remember that the delay for the first packet has a larger value due to interaction with the controller:

Q6. Write the value for the link capacity and RTT between `h1` and `h2`.

Changing the topology size and type. Custom topology.

The default topology is a single switch connected to two hosts. You could change this to a different topo with --topo, and pass parameters for that topology's creation. Start Mininet with the following command:

```
$ sudo mn --topo single,3
```

Q7. Inspect the topology. How many host there are in this topology?

Restart Mininet with the following command:

```
$ sudo mn --topo linear, 4
```

Q8. Inspect the topology with Mininet basic commands. Draw the new topology?

Run a regression test to directly test the connectivity for a topology:

```
$ sudo mn --test pingall --topo single,3
```

Custom topologies can be easily defined as well, using a simple Python API, and an example is provided in `custom/topo-2sw-2host.py`. This example connects two switches directly, with a single host connected on each switch (Figure 3).

Using the custom topology provided in the file `topo-2sw-2host.py` build the new topology given in Figure 2. Rename the file as `custom_topo.py`. Start Mininet with the new topology:

```
sudo mn --custom ~/mininet/custom/custom_topo.py --topo mytopo
```

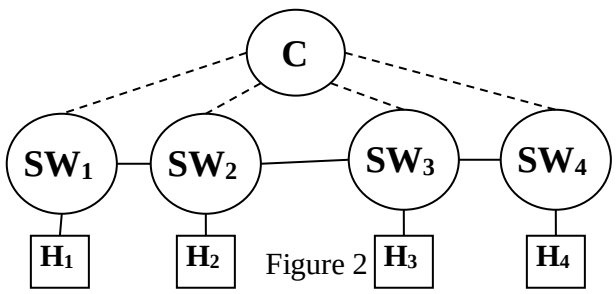

Figure 2

simple topology example (topo-2sw-2host.py) download

```
"""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 newl
topology enables one to pass in '--topo=mytopo' from the command l
"""

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() ) }
```

Figure 3 [http://mininet.org/walkthrough/]

References:

- http://mininet.org/walkthrough/
- https://github.com/mininet/mininet/wiki/Introduction-to-Mininet
- https://github.com/mininet/mininet/wiki/Documentation
- https://openmaniak.com/iperf.php
- https://github.com/mininet/openflow-tutorial/wiki