Assignment 3 Part 2: Mininet Walkthrough

In this assignment you will get introduced to the Mininet network emulator. Mininet allows you to create virtual networks on a single computer and test its operation just like you do in real networks. Later on we will do some network topologies and experiments using Mininet. To work with Mininet follow the subsequent steps:

1. Download the floodlight-vm.zip Linux Mininet VM from the following url: https://drive.google.com/file/d/1p0wCND8eOHAHtWQ1N7h1mCyJL4_DqrFn/view?usp=sharing
   (The username and password for the Linux VM are: floodlight)
2. Watch the following Youtube videos to better understand how to install Mininet and create simple networks on it:
   https://youtu.be/Y-bNHXyozgA
   https://youtu.be/4hIQ8h9aOf0

3. submit a report containing screen shots of the output resulting from the execution of the following Mininet commands and submit to Blackboard by the deadline above.

Interact with Hosts and Switches

Start a minimal topology and enter the CLI:

```
$ sudo mn
```

The default topology is the `minimal` topology, which includes one OpenFlow kernel switch connected to two hosts, plus the OpenFlow reference controller. This topology could also be specified on the command line with `--topo=minimal`. Other topologies are also available out of the box; see the `--topo` section in the output of `mn -h`.
All four entities (2 host processes, 1 switch process, 1 basic controller) are now running in the VM. The controller can be outside the VM, and instructions for that are at the bottom.

If no specific test is passed as a parameter, the Mininet CLI comes up.

In the Wireshark window, you should see the kernel switch connect to the reference controller.

Display Mininet CLI commands:

```
mininet> help
```

Display nodes:

```
mininet> nodes
```

Display links:

```
mininet> net
```

Dump information about all nodes:

```
mininet> dump
```

You should see the switch and two hosts listed.

If the first string typed into the Mininet CLI is a host, switch or controller name, the command is executed on that node. Run a command on a host process:

```
mininet> h1 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:

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

This will show the switch interfaces, plus the VM's connection out (`eth0`).
For other examples highlighting that the hosts have isolated network state, run `arp` and `route` on both `s1` and `h1`.
It would be possible to place every host, switch and controller in its own isolated network namespace, but there's no real advantage to doing so, unless you want to replicate a complex multiple-controller network. Mininet does support this; see the `--innamespace` option.
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 exact same as that seen by the root network namespace:

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

It would be possible to use separate process spaces with Linux containers, but currently Mininet doesn't do that. Having everything run in the "root" process namespace is convenient for debugging, because it allows you to see all of the processes from the console using `ps`, `kill`, etc.

Test connectivity between hosts

Now, verify that you can ping from host 1 to host 2:

```
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. 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).

Repeat the last `ping`:

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

You should see a much lower `ping` time for the second try (< 100us). 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.
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 - h1
...
mininet> h1 kill %python
```

Exit the CLI:

```
mininet> exit
```

Cleanup

If Mininet crashes for some reason, clean it up:

```
$ sudo mn -c
```

Part 2: Advanced Startup Options

Run a Regression Test

You don't need to drop into the CLI; Mininet can also be used to run self-contained regression tests.

Run a regression test:

```
$ sudo mn --test pingpair
```

This command created a minimal topology, started up the OpenFlow reference controller, ran an all-pairs-`ping` test, and tore down both the topology and the controller.
Another useful test is `iperf` (give it about 10 seconds to complete):

```
$ sudo mn --test iperf
```

This command created the same Mininet, ran an iperf server on one host, ran an iperf client on the second host, and parsed the bandwidth achieved.

Changing Topology Size and Type

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. For example, to verify all-pairs ping connectivity with one switch and three hosts:
Run a regression test:

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

Another example, with a linear topology (where each switch has one host, and all switches connect in a line):

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

Parametrized topologies are one of Mininet's most useful and powerful features.

Link variations

Mininet 2.0 allows you to set link parameters, and these can even be set automatically from the command line:

```
$ sudo mn --link tc,bw=10,delay=10ms
mininet> iperf
...
mininet> h1 ping -c10 h2
```

If the delay for each link is 10 ms, the round trip time (RTT) should be about 40 ms, since the ICMP request traverses two links (one to the switch, one to the destination) and the ICMP reply traverses two links coming back.

Custom Topologies

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 off each switch:

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

```
from mininet.topo import Topo
class MyTopo( Topo ):
  def __init__( self ):
    "Create custom topo."
    # Initialize topology
    Topo.__init__( self )
    # Add hosts and switches
    host1 = self.addHost( 'h1' )
    host2 = self.addHost( 'h2' )
    switch3 = self.addSwitch( 's3' )
    switch4 = self.addSwitch( 's4' )
    # Add links
    self.addLink( host1, switch3 )
    self.addLink( switch3, switch4 )
    self.addLink( switch4, host2 )
topos = { 'mytopo': ( lambda: MyTopo() ) }
```

When a custom mininet file is provided, it can add new topologies, switch types, and tests to the command-line. For example:

```
$ sudo mn --custom ~/floodlight/custom/topo-2sw-2host.py --topo mytopo --test
pingall
```

XTerm Display

For more complex debugging, you can start Mininet so that it spawns one or more xterms.

To start an `xterm` for every host and switch, pass the `-x` option:

```
$ sudo mn -x
```

After a second, the xterms will pop up, with automatically set window names.

Alternately, you can bring up additional xterms as shown below.

By default, only the hosts are put in a separate namespace; the window for each switch is unnecessary (that is, equivalent to a regular terminal), but can be a convenient place to run and leave up switch debug commands, such as flow counter dumps.

```
mininet> exit
```

The xterms should automatically close.

Part 3: Mininet Command-Line Interface (CLI) Commands

Display Options

To see the list of Command-Line Interface (CLI) options, start up a minimal topology and leave it running. Build the Mininet:

```
$ sudo mn
```

Display the options:

```
mininet> help
```

Link Up/Down

For fault tolerance testing, it can be helpful to bring links up and down.

To disable both halves of a virtual ethernet pair:

```
mininet> link s1 h1 down
```

You should see an OpenFlow Port Status Change notification get generated. To bring the link back up:

```
mininet> link s1 h1 up
```

XTerm Display

To display an xterm for h1 and h2:

```
mininet> xterm h1 h2
```