Cpr E 489 Lab Experiment #7: Introduction to Mininet
(Total Points: 100)

Objective

To become familiar with Mininet, network emulation using Mininet, and its support for OpenFlow

Pre-Lab

1) Read the Wikipedia pages on Software Defined Networks and OpenFlow:
   https://en.wikipedia.org/wiki/Software-defined_networking
   https://en.wikipedia.org/wiki/OpenFlow
2) Download and install a virtualization system on your computer. We recommend VirtualBox (free, GPL) because it is **free** and works on OS X, Windows, and Linux (though it's slightly slower than VMware in our tests). You can also use Qemu for any platform, VMware Workstation for Windows or Linux, VMware Fusion for Mac, or KVM (free, GPL) for Linux.

Lab Expectations

Please carefully read the following important information, regarding the new format of Lab 7 in Week 13. A similar format will be used for Lab 8 in Week 15 as well.

1) All students will **work on Lab 7 on their own** instead of as a team.
2) You may work on the lab **ANY time at your convenience.**
3) **The TAs will be online during the regular lab hours** to help you and for Q&A:
   a. TA – Ethan Shoemaker will be available on Zoom (meeting ID: 978 354 397) on 4/7 and 4/14 from 2:10 PM to 6:00 PM.
   b. TA – James Bonner will be available on Zoom (meeting ID: 127 304 414) on 4/8 and 4/15 from 1:10 to 3:00 PM, and 4/10 and 4/17 from 1:10 to 3:00 PM.
4) Outside the regular lab hours, if you have any questions, you may contact the TAs via email or schedule an individual online meeting with the TAs.
5) After the lab, write up a lab report. Be sure to:
   - Summarize what you learned in a few paragraphs **(20 points)**
   - Perform all 4 parts (Part 1 through Part 4). After every part, summarize what you have learned, and substantiate it with screenshots from the virtual machine. Please label each screenshot with the corresponding screenshot number **(6 points for the summary of each part, and 2 points for each of the 18 screenshots, totaling 60 points)**.
   - Complete the two exercise problems at the end of this lab. **(10 points each, totaling 20 points)**
6) **Lab 7 report is due on 4/21 Tuesday 11 AM for all sections. Each student please submit a copy of your own lab report on Canvas.**

Problem Description

Mininet (http://www.mininet.org) is an emulation platform for networks, including hosts, switches, controllers, and network applications. It emulates a network by running the network code on virtual hardware. That is, it creates a realistic network by running real kernel, switch and application code, e.g., a real HTTP server.

In this lab, you will run several applications and use Wireshark to look at packet flows with OpenFlow packets. You will do the following:

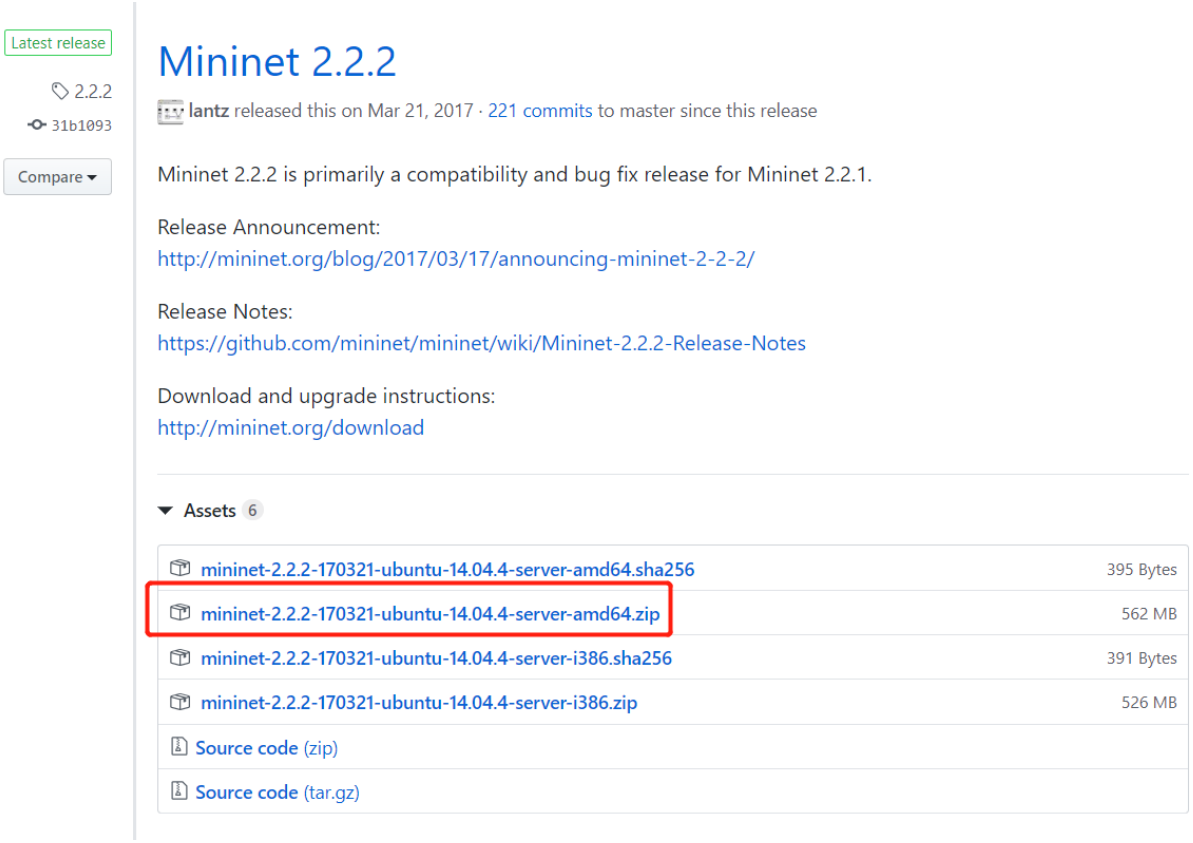
1) Run the Mininet network emulator, using various topologies.
2) Run several commands and applications, and observe the data and OpenFlow packets using Wireshark.
3) Use Mininet to emulate other built-in topologies and standard applications.

Procedure

This lab will use the tutorials and documentation available on the Mininet official web site:

1) First, you need to download Mininet from the following website:
https://github.com/mininet/mininet/wiki/Mininet-VM-Images
and then click on the address below

2) Download the following ZIP folder:
Mininet 2.2.2 on Ubuntu 14.04 LTS - 64 bit



3) After downloading the file, unzip it and open **mininet-2.2.2-170321-ubuntu-14.04.4-server-amd64.** This will open oracle VM VirtualBox manager. In Oracle VM, click on **import.**

4) After importing Mininet, a new panel will show Mininet-VM to the left. Double click on it to open Mininet.

5) After installing and starting the Mininet VM, login using

   username: mininet
   password: mininet

6) The command to start Mininet is mn, and it has to be run as a super user (ie 'sudo mn'). Before doing this, you should read the manual page of Mininet in order to learn about mn and its options:

   man mn

Note: The following walkthrough is available in its complete form online at: http://mininet.org/walkthrough/

Part 1: Everyday Mininet Usage

First, a (perhaps obvious) note on command syntax for this walkthrough:

- The "$" character precedes Linux commands that should be typed at the shell prompt
- "mininet>" precedes Mininet commands that should be typed at Mininet's Command-Line Interface (CLI),
- The "#" character precedes Linux commands that are typed at a root shell prompt

In each case, you should only type the command to the right of the prompt (and then press return/enter, of course!)

Display Startup Options

Let's get started with Mininet's startup options.

Type the following command to display a help message describing Mininet's startup options:

$ sudo mn -h

This walkthrough will cover typical usage of the majority of options listed.

Start Wireshark

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

$ sudo wireshark &

If Wireshark is installed but you cannot run it (e.g. you get an error like $DISPLAY not set), you can follow these steps to access a GUI within your Mininet environment (these steps are from the FAQ, which can be found at https://github.com/mininet/mininet/wiki/FAQ#wiki-x11-forwarding):

- sudo apt-get update
- sudo apt-get install xinit lwde (other GUI environments are available in the FAQ)
- sudo apt-get install virtualbox-guest-dkms
- You should be able to start X11 by simply calling startx.

Setting X11 up correctly will enable you to run other GUI programs and the xterm terminal emulator (used later in this lab).

After Wireshark has been started, setup Wireshark to filter for OpenFlow packets by typing of in the Wireshark filter box and clicking Apply:

In Wireshark, click Capture, then Interfaces, then select Start after checking the loopback interface (lo).

For now, there should be no OpenFlow packets displayed in the main window.

Note: Wireshark is installed by default in the Mininet VM image. If the system you are using does not have Wireshark and the OpenFlow plugin installed, you may be able to install both of them using Mininet's install.sh script as follows:

```
$ cd ~
$ git clone https://github.com/mininet/mininet  # if it's not already there
$ mininet/util/install.sh -w
```

Interact with Hosts and Switches

Open a terminal (press Ctrl+Alt+T), 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
```

(take a screenshot of this: #1)

Display links:

```
mininet> net
```

(take a screenshot of this: #2)

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 **(take a screenshot of this: #3)**

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 through 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 **(take a screenshot of this: #4)**

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 **(take a screenshot of this: #5)**

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:
(take a screenshot of this: #6)

```
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 **(take a screenshot of this: #7)**

This command creates a minimal topology, starts up the OpenFlow reference controller, runs an all-pairs-ping test, and finally, tears down both the topology and the controller.

Another useful test is iperf (give it about 10 seconds to complete):

$ sudo mn --test iperf **(take a screenshot of this: #8)**

This command creates the same Mininet, runs an iperf server on one host, runs an iperf client on the second host, and parses 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 **(take a screenshot of this: #9)**

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 **(take a screenshot of this: #10)**

Parameterized 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
```
(take a screenshot of this: #11)

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.

You can customize each link using Mininet's Python API, but for now you will probably want to continue with the walkthrough.

Adjustable Verbosity

The default verbosity level is info, which prints what Mininet is doing during startup and teardown. Compare this with the full debug output with the -v param:

```
$ sudo mn -v debug
...
mininet> exit
```

Lots of extra detail will print out. Now try output, a setting that prints CLI output and little else:

```
$ sudo mn -v output
mininet> exit
```

Outside the CLI, other verbosity levels can be used, such as warning, which is used with the regression tests to hide unneeded function output.

Custom Topologies

Custom topologies can be easily defined as well, using a simple Python API, and an example is provided in mininet/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)

Note: this file (topo-2sw-2host.py) already exists in the mininet-VM in this path: mininet/custom, you can use command 'ls' and 'cd' to access it.

```
"""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 )

```

```
33
34 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 ~/mininet/custom/topo-2sw-2host.py --topo mytopo --test pingall
```

(take a screenshot of this: #12)

ID = MAC

By default, hosts start with randomly assigned MAC addresses. This can make debugging tough, because every time the Mininet is created, the MACs change, so correlating control traffic with specific hosts is tough.

The --mac option is super-useful, and sets the host MAC and IP addrs to small, unique, easy-to-read IDs.

Before:

```
$ sudo mn
...
mininet> h1 ifconfig
h1-eth0  Link encap:Ethernet  HWaddr f6:9d:5a:7f:41:42
        inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:6 errors:0 dropped:0 overruns:0 frame:0
        TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:392 (392.0 B)  TX bytes:392 (392.0 B)
mininet> exit
```

After:

```
$ sudo mn --mac
...
mininet> h1 ifconfig
```

(take a screenshot of this: #13)

```
h1-eth0  Link encap:Ethernet  HWaddr 00:00:00:00:00:01
        inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
mininet> exit
```

In contrast, the MACs for switch data ports reported by Linux will remain random. This is because you can 'assign' a MAC to a data port using OpenFlow, as noted in the FAQ. This is a somewhat subtle point which you can probably ignore for now.

Other Switch Types

Other switch types can be used. For example, to run the user-space switch:

$ sudo mn --switch user --test iperf **(take a screenshot of this: #14)**

Note the much lower TCP iperf-reported bandwidth compared to that seen earlier with the kernel switch.

If you do the ping test shown earlier, you should notice a much higher delay, since now packets must endure additional kernel-to-user-space transitions. The ping time will be more variable, as the user-space process representing the host may be scheduled in and out by the OS.

On the other hand, the user-space switch can be a great starting point for implementing new functionality, especially where software performance is not critical.

Another example switch type is Open vSwitch (OVS), which comes preinstalled on the Mininet VM. The iperf-reported TCP bandwidth should be similar to the OpenFlow kernel module, and possibly faster:

$ sudo mn --switch ovsk --test iperf **(take a screenshot of this: #15)**

Mininet Benchmark

To record the time to set up and tear down a topology, use test 'none':

$ sudo mn --test none **(take a screenshot of this: #16)**

Everything in its own Namespace (user switch only)

By default, the hosts are put in their own namespace, while switches and the controller are in the root namespace. To put switches in their own namespace, pass the --innamespace option:

$ sudo mn --innamespace --switch user **(take a screenshot of this: #17)**

Instead of using loopback, the switches will talk to the controller through a separately bridged control connection. By itself, this option is not terribly useful, but it does provide an example of how to isolate different switches.

Note that this option does not (as of 11/19/12) work with Open vSwitch.

mininet> exit

Part 3: Mininet CLI Commands

Display Options

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

$ sudo mn

Find the list of CLI commands:

mininet> help

Python Interpreter

If the first phrase on the Mininet command line is py, then that command is executed with Python. This might be useful for extending Mininet, as well as probing its inner workings. Each host, switch, and controller has an associated Node object.

At the Mininet CLI, run:

mininet> py 'hello ' + 'world'

Print the accessible local variables:

mininet> py locals()

Next, see the methods and properties available for a node, using the dir() function:

mininet> py dir(s1)

You can read the on-line documentation for methods available on a node by using the help() function:

mininet> py help(h1) (Press "q" to quit reading the documentation.)

You can also evaluate methods of variables:

mininet> py h1.IP() **(take a screenshot of this: #18)**

Part 4: Python API Examples

The examples directory in the Mininet source tree includes examples of how to use Mininet's Python API, as well as potentially useful code that has not been integrated into the main code base.

Note: As noted at the beginning, this Walkthrough assumes that you are either using a Mininet VM, which includes everything you need, or a native installation with all of the associated tools, including the reference controller controller, which is part of the OpenFlow reference implementation and may be installed using install.sh -f if it has not been installed.

SSH daemon per host

One example that may be particularly useful runs an SSH daemon on every host:

$ sudo ~/mininet/examples/sshd.py

From another terminal, you can ssh into any host and run interactive commands:

$ ssh 10.0.0.1
$ ping 10.0.0.2
...
$ exit

Finally, exit the SSH example in Mininet:

$ exit

Remember to clean the network every time after you exit mn by executing:

sudo mn –c

Exercises

1) When you ran mn with the default topology, and then at h1 you executed
ping h2
it was observed that the first ping took much longer than subsequent pings.

   Explain why this has happened.

2) In Part 2 of the Walkthrough, an example of a custom topology was constructed using Python, in which 2 switches were connected, and one host was connected to each of the switches.

   Modify this custom topology to have 3 switches connected in a linear topology, and one host is connected to each of the two switches at the end. Run your code, and test the topology, e.g., using the command you learned from previous section (with your python file). Show the results of your tests (include a screenshot).

   Note: you can do the exercise by following these steps below:
   Step 1: cd into the folder where topo-2sw-2host.py is located. (path: ~/mininet/custom)
   Step 2: use command cp to copy the file:
   $ cp topo-2sw-2host.py new.py
   Step 3: use your favorite text editor (vim, emacs, pico) to edit your code:
   $ vim new.py
   Note: press Insert if you can't edit the text file.
   Step 4: After editing in vim (instructions vary depending on your text editor), press Esc to switch back to the command mode, and then enter :wq , then press Enter to save your file.
   Step 5: Run your code and test your topo:
   $ sudo mn --custom ~/mininet/custom/new.py --topo mytopo --test pingall.