Lab 1: Installing and Using Mininet

Part 1: Installing Virtualbox

The first thing required to use Mininet is a Virtual Machine (VM) manager to run our Mininet VM. VirtualBox is free and open source and can be downloaded here. You are free to use kvm, vmware, or xen if you are familiar with those hypervisors - however, the TAs will not support them, so you are on your own.

If you are having trouble installing VirtualBox, make sure to consult the VirtualBox manual, Piazza, or the TAs. It is recommended that you **start the lab early** in case you encounter any problems setting up either VirtualBox or Mininet. If you do not have a computer, please contact the TAs and we can work something out.

Common Problems Installing VirtualBox:

- Did you download the correct version? (32-bit vs 64-bit)
- Is your computer really old? It might not be able to be virtualized -- talk to the TA.
- Make sure that virtualization is enabled in your BIOS.

Part 2: Installing Mininet

Once VirtualBox is installed, we can install the mininet VM. For this class, please use the VM available at this link. (You should use your UCSC google account to get access)

Using a GUI:

- Once the download is complete, open VirtualBox and select File>Import Appliance… Navigate to the OVA file you downloaded.
- When the mininet VM has been successfully imported, start the VM.
- You should be presented with a GUI. Helpful tips:
    a. Chromium is a web browser. You can use this to go to the class webpage, copy/paste example code from the PDFs.
    b. You can use your Google Drive to copy files to/from the VM.

Part 3: Using Mininet

A walkthrough can be found on the mininet page. First and foremost, here's some background and information on what is Mininet and why we are using it (we used to have the labs using actual routes and switches; since then we have migrated to virtualized labs).

What is Mininet? Mininet is a "network in a box" tool developed at Stanford in 2010. It is designed to allow large scale networks to be emulated in software on a laptop. Its rise has also been dictated by the use of OpenFlow (which will be the subject of Lab 3 and the Final Project). If you are interested in reading more, here is the original Mininet paper.

Why Mininet? Our previous "physical" network lab had been aging from the wear and tear of 11 years of usage (it was donated by Cisco in 2004). Mininet is probably one of the simplest forms of network emulators, is *free*, is open source, and is widely used by the research community, as well as by universities for teaching computer networks. It allows for more interesting topologies than what can be achieved in the physical lab. Most importantly, it has enabled us to accommodate CE 150/L's enrollment growth (only five years ago enrollment was around 40 students, while today it is peaking at 100). The physical lab only had 10 workstations, but over the years only fewer were operational. So in light of the growing student base we started looking towards a solution which scales with the number of students. Almost all students have access to a computer, so instead of buying hundreds of thousands of dollars worth of specialized equipment (e.g., routers, switches), we use general-purpose computers.

How does Mininet Work? Mininet works simply by creating a virtual network on your computer/laptop. It accomplishes this task by creating host namespaces (h1, h2, etc) and connecting them through virtual interfaces. So when we run the command *ping* between the linux namespaces h1 and h2, the ping will run from h1's namespace through a virtual interface pair created for h1 and h2, before it reaches h2. If h1 and h2 are connected through a switch as shown in the python code in the Mininet walkthrough, the ping will transit multiple virtual interface pairs. The switches that we will be using are running OpenVSwitch (OVS). Mininet will connect additional virtual interfaces between each virtual port on the switch with each connected host. The host name space allows each host to see the same file system, but operates as its own process that will run separately from each of the other host processes. The OVS version running on the Ubuntu image supports OpenFlow.

Understanding some mininet commands:

1. *sudo mn*: will start mininet

```
mininet@mininet-vm:~$ sudo mn
*** Creating network
*** Adding controller
*** 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>
```

2. You can use *mn -h* or type *help* after you have run mininet net.

```
mininet> help

Documented commands (type help <topic>):
========================================
EOF    gterm  iperfudp  nodes        pingpair      py      switch
dpctl  help   link      noecho       pingpairfull  quit    time
dump   intfs  links     pingall      ports         sh      x
exit   iperf  net       pingallfull  px            source  xterm

You may also send a command to a node using:
  <node> command {args}
For example:
  mininet> h1 ifconfig

The interpreter automatically substitutes IP addresses
for node names when a node is the first arg, so commands
like
  mininet> h2 ping h3
should work.

Some character-oriented interactive commands require
noecho:
  mininet> noecho h2 vi foo.py
However, starting up an xterm/gterm is generally better:
  mininet> xterm h2
```

3. We can see that when we launched mininet, it created a mini-network from the output. When we use the *net* command we can see *h1* indicated host 1, it has one network interface *eth0* which is connected to the switch on interface *eth1*. This is shown on the output line: *h1 h1-eth0:s1-eth1*. There is also host 2 (*h2*), switch 1 (*s1*), and controller 0 (*c0*).

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
mininet> nodes
available nodes are:
c0 h1 h2 s1
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=1049>
<Host h2: h2-eth0:10.0.0.2 pid=1052>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=1057>
<Controller c0: 127.0.0.1:6633 pid=1042>
```

   a. From dump we can also see what IP address have been assigned to h1 and h2. We also are given the pids of each process. Processes in mininet are used for hosts, switches, and controllers. Mininet is composed of processes using Interprocess Communication (IPC) to emulate a network environment. You can review the prelab's resources and reading to understand more.

4. *sudo mn -c*: cleans the mininet system. Use this if you are having errors to start mininet, in case some problem from a previous execution has persisted.

Running Mininet as a Python script:

To make custom topologies, it is useful to be able to refine a topology in a script. The following is an example of using a Python script to launch Mininet with a custom topology:

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.cli import CLI

class MyTopology(Topo):
    """
    A basic topology
    """
    def __init__(self):
```

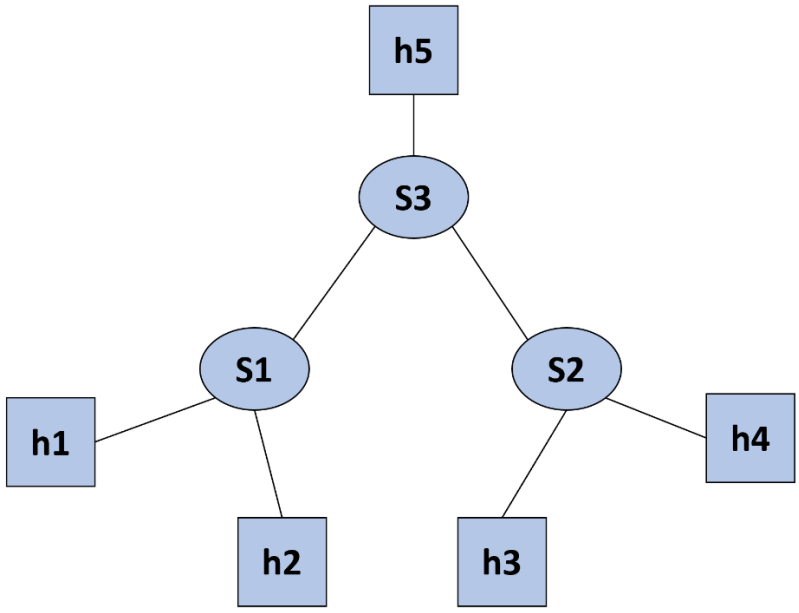
```
        Topo.__init__(self)

        # Set Up Topology Here
        switch = self.addSwitch('s1')     ## Adds a Switch

        host1 = self.addHost('h1')        ## Adds a Host

        self.addLink(host1, switch)       ## Add a link

if __name__ == '__main__':
    """
    If this script is run as an executable (by chmod +x), this is
    what it will do
    """

    topo = MyTopology()            ## Creates the topology
    net = Mininet( topo=topo )          ## Loads the topology
    net.start()                       ## Starts Mininet

    # Commands here will run on the simulated topology
    CLI(net)

    net.stop()                        ## Stops Mininet
```

This file is available here.

You should use this as a skeleton for getting started on the lab. On the Mininet site, The API Reference will be an excellent resource for figuring out how to run pings or open the command prompt in between the net.start() and net.stop() lines.

The Lab [100 pts]:

1. In Mininet change the default configuration through a python script to have 5 hosts connected through three switches as shown in the figure:



2. [30 pts] Save a screenshot of *dump* and *pingall* output. Explain what is being shown in the screenshot.
3. [10 pts] Run the *iperf* command as well, and screenshot the output, how fast is the connect?
4. Run wireshark, and using the display filter, filter for “of”. Note: When you run wireshark you should do so as “sudo wireshark”. When you choose an interface to capture on, you should select “any”.
    a. [20 pts] Run ping from a host to any other host using *hX ping -c 5 hY*. How many *of_packet_in* messages show up? Take a screenshot of your results.
    b. [20 pts] What is the source and destination IP addresses for these entries? Find another packet that matches the “of” filter with the OpenFlow typefield set to *OFPT_PACKET_OUT*. What is the source and destination IP address for this entry? Take screenshots showing your results.
    c. [20 pts] Replace the display filter for “of” to “icmp && not of”. Run *pingall* again, how many entries are generated in wireshark? What types of icmp entries show up? Take a screenshot of your results.

Submission:

You will submit 3 files for this assignment (student id is the part before ‘@’ in id@ucsc.edu):

1. <your student id>-lab1.pdf
    . The PDF with all of your solutions to the Lab.
2. <your student id>-topo.py
    . The mininet topology you created in Lab 1.
3. README.txt
    . A README file. This should describe the contents of each file you submit, and contain your name, e-mail, CruzID, and student ID number.