Mininet Walkthrough

This walkthrough demonstrates most Mininet commands, as well as its typical usage in concert with the Wireshark dissector.

The walkthrough assumes that your base system is the Mininet VM, or a native Ubuntu installation with all OpenFlow tools and Mininet installed (this is usually done using Mininet's `install.sh`).

The entire walkthrough should take under an hour.

Note: If you are using the Ubuntu Mininet 2.0.0d4 package, it uses a slightly different syntax for `Topo()` - e.g. `add_switch` vs. `addSwitch`, etc.. If you check out Mininet from source, you may wish to check out the `2.0.0d4` tag to see code (including code in `examples`) which is consistent with the 2.0.04 package.

Part 1: Everyday Mininet Usage

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

- `$` preceeds Linux commands that should be typed at the shell prompt
- `mininet>` preceeds Mininet commands that should be typed at Mininet's CLI,
- `#` preceeds 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`, 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 &
```

In the Wireshark filter box, enter this filter, then click `Apply`:

```
of
```

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.

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
```

If Wireshark is installed but you cannot run it (e.g. you get an error like `$DISPLAY not set`, please consult the FAQ: https://github.com/mininet/mininet/wiki/FAQ#wiki-x11-forwarding.)

Setting X11 up correctly will enable you to run other GUI programs and the `xterm` terminal emulator, used later in this walkthrough.

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 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. 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 automatially 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.

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 `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) 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 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() ) }
```

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
```

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
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.

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.

Xterms are also useful for running interactive commands that you may need to cancel, for which you'd like to see the output.

For example:

In the xterm labeled "switch: s1 (root)", run:

```
# dpctl dump-flows tcp:127.0.0.1:6634
```

Nothing will print out; the switch has no flows added. To use `dpctl` with other switches, start up mininet in verbose mode and look at the passive listening ports for the switches when they're created.

Now, in the xterm labeled "host: h1", run:

```
# ping 10.0.0.2
```

Go back to `s1` and dump the flows: # dpctl dump-flows tcp:127.0.0.1:6634

You should see multiple flow entries now. Alternately (and generally more convenient), you could use the `dpctl` command built into the Mininet CLI without needing any xterms or manually specifying the IP and port of the switch.

You can tell whether an xterm is in the root namespace by checking ifconfig; if all interfaces are shown (including `eth0`), it's in the root namespace. Additionally, its title should contain "(root)".

Close the setup, from the Mininet CLI:

```
mininet> exit
```

The xterms should automatically close.

Other Switch Types

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

```
$ sudo mn --switch user --test iperf
```

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
```

Mininet Benchmark

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

```
$ sudo mn --test none
```

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
```

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 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
```

Python Interpreter

If the first phrase on the Mininiet 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()
```

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
```

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
```

Exit SSH example mininet:

```
$ exit
```

You will wish to revisit the examples after you've read the Introduction to Mininet, which introduces the Python API.

Part 5: Walkthrough Complete!

Congrats! You've completed the Mininet Walkthrough. Feel free to try out new topologies and controllers or check out the source code.

Next Steps to mastering Mininet

If you haven't done so yet, you should definitely go through the OpenFlow tutorial.

Although you can get reasonably far using Mininet's CLI, Mininet becomes much more useful and powerful when you master its Python API. The Introduction to Mininet provides an introduction to Mininet and its Python API.

If you are wondering how to use a *remote controller* (e.g. one running outside Mininet's control), this is explained below.

Appendix: Supplementary Information

These are not required, but you might find them useful to skim.

Using a Remote Controller

Note: this step is not part of the default walkthrough; it is primarily useful if you have a controller running outside of the VM, such as on the VM host, or a different physical PC. The OpenFlow tutorial uses `controller --remote` for starting up a simple learning switch that you create using a controller framework like POX, NOX, Beacon or Floodlight.

When you start a Mininet network, each switch can be connected to a remote controller - which could be in the VM, outside the VM and on your local machine, or anywhere in the world.

This setup may be convenient if you already have a custom version of a controller framework and development tools (such as Eclipse) installed on the local machine, or you want to test a controller running on a different physical machine (maybe even in the cloud).

If you want to try this, fill in the host IP and/or listening port:

```
$ sudo mn --controller=remote,ip=[controller IP],port=[controller listening port]
```

For example, to run POX's sample learning switch, you could do something like

```
$ cd ~/pox
$ ./pox.py forwarding.l2_learning
```

in one window, and in another window, start up Mininet to connect to the "remote" controller (which is actually running locally, but outside of Mininet's control):

```
$ sudo mn --controller=remote,ip=127.0.0.1,port=6633
```

Note these are actually the default IP address and port values.

If you generate some traffic (e.g. `h1 ping h2`) you should be able to observe some output in the POX window showing that the switch has connected and that some flow table entries have been installed.

A number of OpenFlow controller frameworks are readily available and should work readily with Mininet as long as you start them up and specify the `remote` controller option with the correct IP address of the machine where your controller is runing and the correct port that it is listening on.

NOX Classic

The Mininet default install (using `util/install.sh -a`) does not install NOX Classic. If you would like to install it, run `sudo ~/mininet/util/install.sh -x`.

Note that NOX Classic is deprecated and may not be supported in the future.

To run a regression test with NOX running the NOX app `pyswitch`, the `NOX_CORE_DIR` env var must be set to the directory containing the NOX executable.

First verify that NOX runs:

```
$ cd $NOX_CORE_DIR
$ ./nox_core -v -i ptcp:
```

Ctrl-C to kill NOX, then run a test with NOX pyswitch:

```
$ cd
$ sudo -E mn --controller=nox,pyswitch --test pingpair
```

Note that the `--controller` option has a convenient syntax for specifying options to the controller type (in this case, `nox` running `pyswitch`.)

There's a hesitation of a few seconds while NOX loads and the switch connects, but then it should complete the `ping`.

Note that this time, `mn` was called via `sudo -E`, to keep the `NOX_CORE_DIR` environment variable. If you're running nox remotely, using `--controller remote`, then the `-E` isn't necessary. Alternately, you can change the first line of `/etc/sudoers`, via `sudo visudo`, to change the `env_reset` line to:

```
Defaults !env_reset
```

… so that when running `sudo` the env var setup isn't changed.

News

- Congrats to Mininet SIGCOMM 2018 Hackathon winners!
- new-post
- Open Networking Foundation promotes Mininet

- Mininet wins ACM SIGCOMM SOSR Software Systems Award !
- Mininet VM Security Advisory