Lab 1 Part 1

Mininet Python API

# Objective

Creating more complex topologies and run the mininet emulation from a script

# What you will learn

Use python script to create and modify complex Mininet topologies, run experiments and tests.

# Prerequisites

Mininet installation and basic networking knowledge

## Display Mininet CLI commands:

**$ 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.

**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.

*<Host h1: h1-eth0:10.0.0.1 pid=1792>*

*<Host h2: h2-eth0:10.0.0.2 pid=1794>*

*<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=1799>*

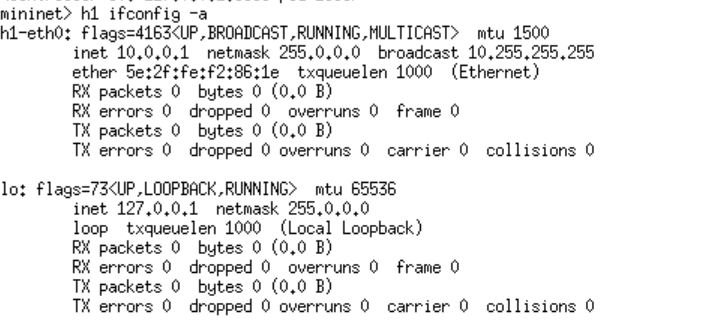
*<Controller c0: 127.0.0.1:6653 pid=1785>*

Information shown about device name, ethernet port and IP address, process id

* 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**

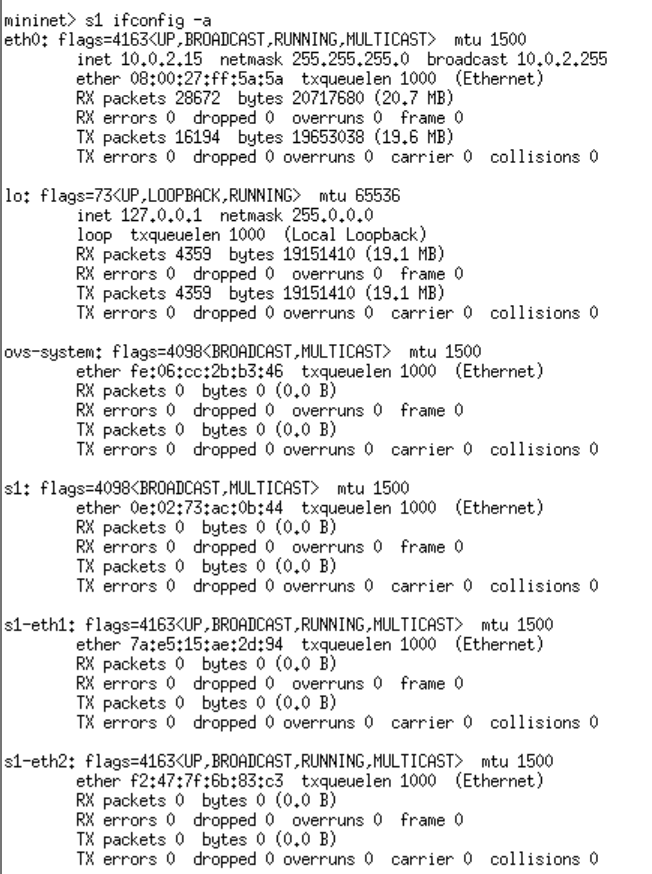
You should see the host’s h1-eth0 and loopback (lo) interfaces. Note that this interface (h1-eth0) is not seen by the primary Linux system when ifconfig is run, because it is specific to the network namespace of the host process.



* 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 (so commands to the switch (like ovs-ofctl) will be executed from a standard terminal (not within mininet)

**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 both s1 and h1.

**mininet> h1 arp -a**

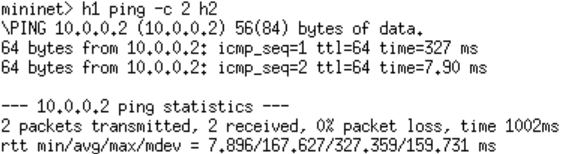
You will see an empty list

* Test connectivity between hosts to verify that you can ping from host 0 to host 1:

**mininet> h1 ping -c 2 h2**

*You will see that the first packet is slower (around 3-4 ms), while the second is much faster (around 0.1-0.2ms) why?*

Also 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.



* Try to run arp again and see what has changed for example for h1.

*Why is the table different now?*

**

* You can also check the performance (bandwidth) of the virtual connection

**mininet> iperf**

Depending on your machine you should see high capacity (orders of 10s of Gb/s)

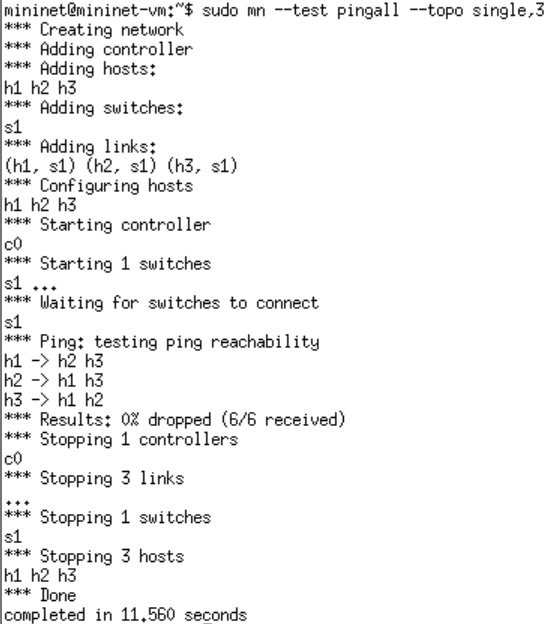
## Changing Topology Size and Type

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:

–topo enables to create a different topology. “Single” generates a star topology with all hosts connected to the same switch:

**$ 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

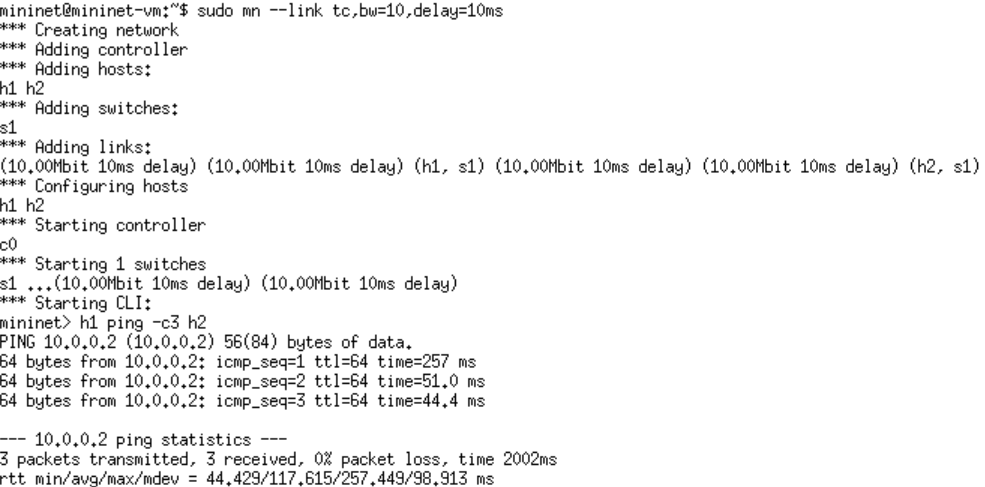
Mininet 2.0 allows you to set link parameters, and these can even be set automatically from the command line:

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

You can then test again the link delay

**mininet> h1 ping -c3 h2**

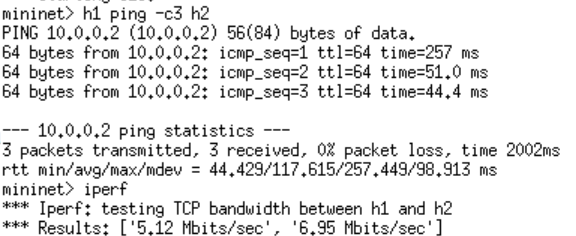
*Now you will see delays in the order of 40ms in the packets after the first. Why?*

**

You can also check the bandwidth between h1 and h2:

**mininet> iperf**

*Now you will see values of around 10Mb/s. Why?*



You can also run these from a separate terminal:

**mininet> xterm h1**

Then from the new terminal run the ping. However, now you need to ping the specific IP address

**root@mininet-vm:~# ping -c3 10.0.0.2**

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**

You can check the difference when you run again commands like ping or iperf. Lots of extra detail will print out.

Now instead try output, a setting that prints CLI output and little else:

**$ sudo mn -v output**

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:

You can see this file by typing:

**$ vi ~/mininet/custom/topo-2sw-2host.py**

Text

Description automatically generated

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.

Without the use of the –mac option you would get something like this:

**$ 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)*

If you use the –mac option instead you will get the following:

**$ 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)*

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.

# Running mininet from a script

* Create a Mininet tree topology in python.

Tree should have a depth of 2, with 2 nodes on each branch. Send a single ping from first host to last host.

h1

h2

h3

h4

s2

s3

s1

Create a new file called “tree2x2.py” using pico, nano or vi and add the following.

from mininet.net import Mininet →Import the Mininet library

from mininet.topolib import TreeTopo →Import the topology library

tree4 = TreeTopo(depth=2,fanout=2) →Create a tree topology with the TreeTopo API

net = Mininet(topo=tree4) →Create a handle to the mininet sim with the topology

above

net.start() →Start Mininet

h1, h4 = net.hosts[0], net.hosts[3] →Assign to variables h1 and h4 the hosts number 0 and 3

print(h1.cmd('ping -c1 %s' % h4.IP()))→Run the ping command from h1 to h4 and print the

results

net.stop() →Stop Mininet

Now that you have created your simulation script file, you can execute it to run your simulation.

**$ sudo python tree2x2.py**

*PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.*

*64 bytes from 10.0.0.4: icmp\_seq=1 ttl=64 time=1112 ms*

*--- 10.0.0.4 ping statistics ---*

*1 packets transmitted, 1 received, 0% packet loss, time 0ms*

*rtt min/avg/max/mdev = 1111.572/1111.572/1111.572/0.000 ms*

At times you might need to clean up mininet to delete any previous active configuration:

**$ sudo mn -c**

Now modify the code to add a web server

The Http server is on h4.=, the Http client is on h1.

from mininet.net import Mininet

from mininet.topolib import TreeTopo

tree4 = TreeTopo(depth=2,fanout=2)

net = Mininet(topo=tree4)

net.start()

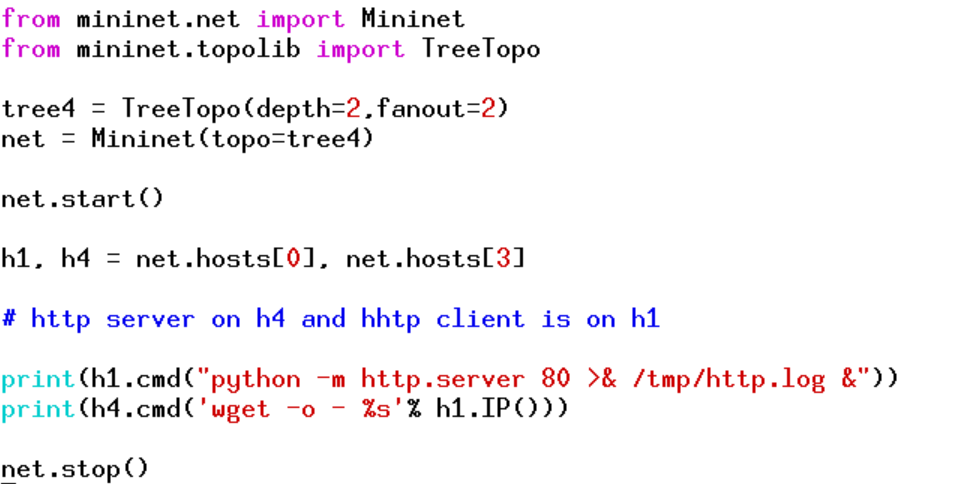
h1, h4 = net.hosts[0], net.hosts[3]

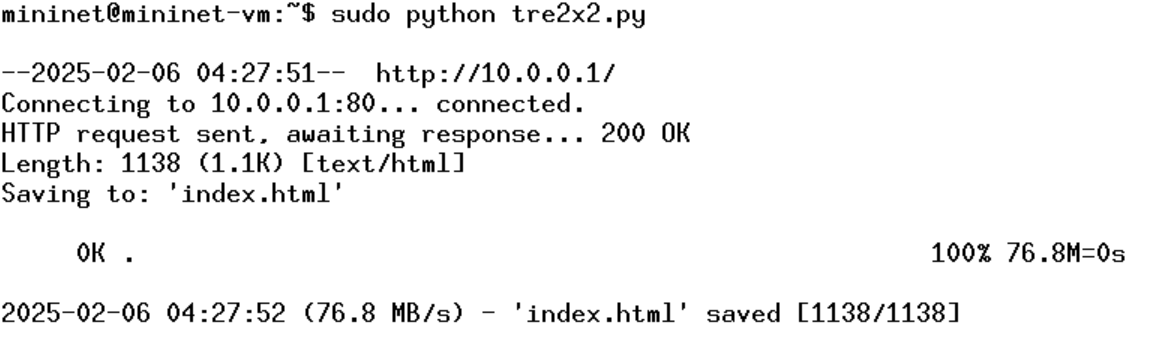
print(h1.cmd("python -m http.server 80 >& /tmp/http.log &"))

print(h4.cmd('wget -o - %s'% h1.IP()))

net.stop()

You should see the output showing the download speed.





Load test using iperf

*from mininet.net import Mininet*

*from mininet.topolib import TreeTopo*

*tree4 = TreeTopo(depth=2,fanout=2)*

*net = Mininet(topo=tree4)*

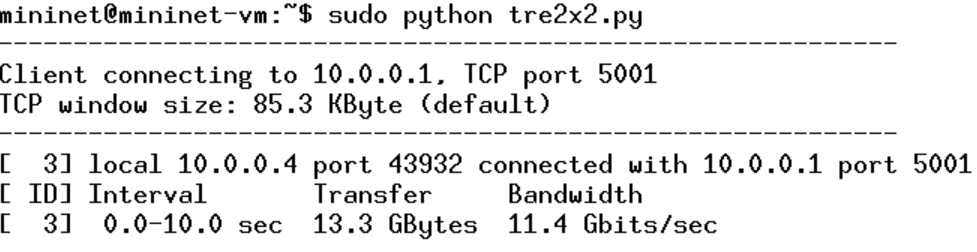
*net.start()*

*h1, h4 = net.hosts[0], net.hosts[3]*

*h1.cmd('iperf -s &')*

*print(h4.cmd('iperf -c %s' % h1.IP()))*

*net.stop()*



# Appendix

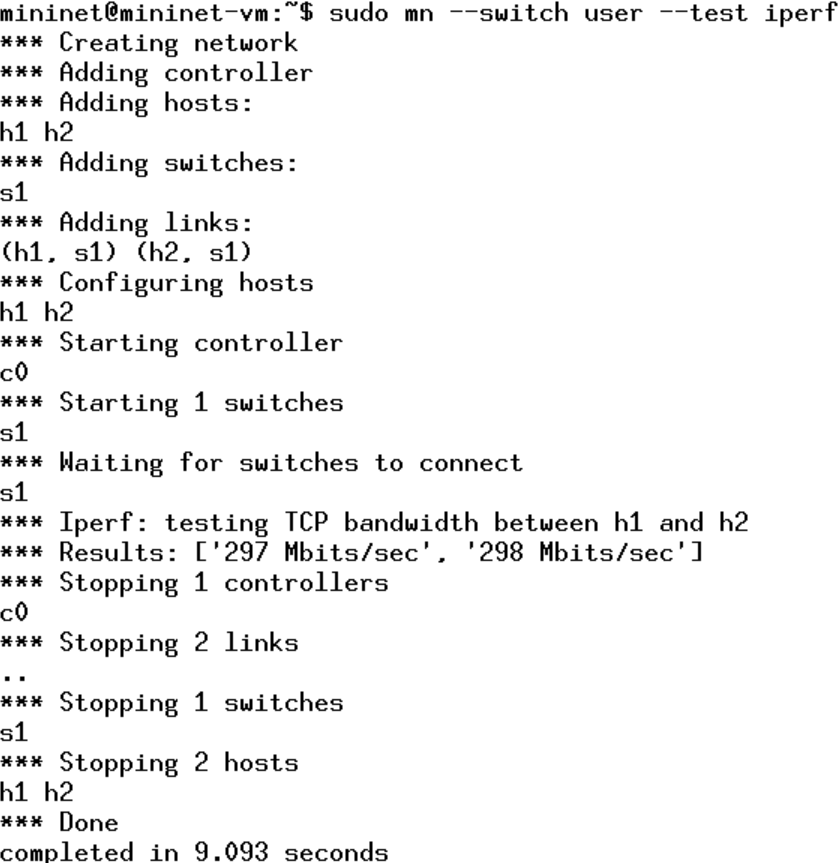
## Using Other Switch Types

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.

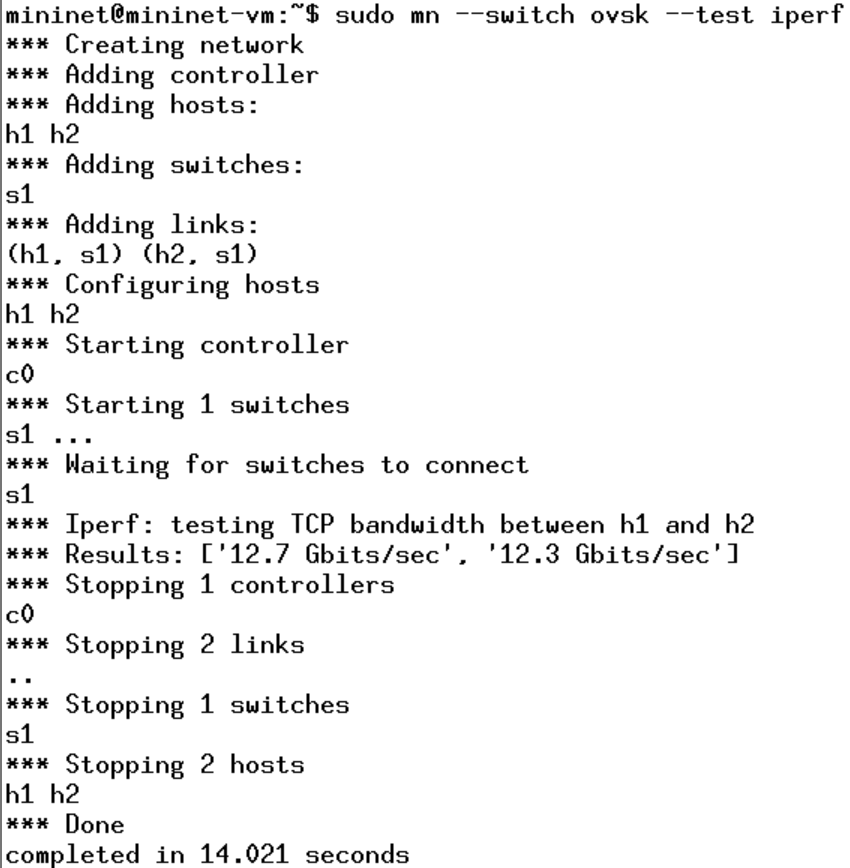
If you do the ping test shown earlier, you should notice a much higher delay, since now packets must endure additional kernel-to-user-space transitions. The ping time will be more variable, as the user-space process representing the host may be scheduled in and out by the OS.



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:

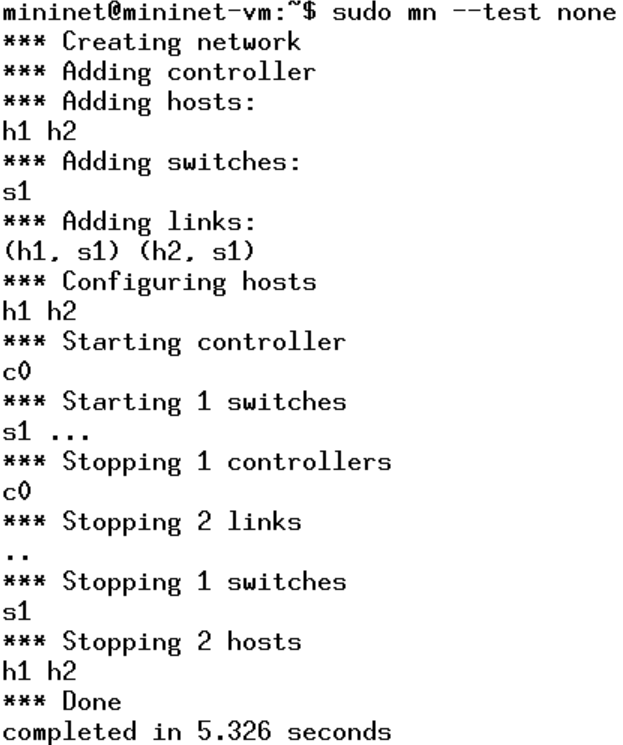
$ sudo mn --switch ovsk --test iperf



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:

mininet> exit

## 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 the setup

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 http.server 80 &

mininet> h2 wget -O - h1

...

mininet> h1 kill %python

NOTE: For Python 3, the HTTP server is called http.server; for Python 2, it is called SimpleHTTPServer. Make sure you are using the right one for the version of Mininet you are running. To find out which Python version Mininet is using, you can type

mininet> py sys.version

3.8.10 (default, Nov 14 2022, 12:59:47)

Exit the CLI:

mininet> exit

Cleanup

If Mininet crashes for some reason, clean it up:

$ sudo mn -c

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.