Mawlana Bhashani Science & Technology University

Lab Report

Report No: 03
Course code: ICT-3208
Course title: Network Planning and Designing Lab
Date of Performance:
Date of Submission:

Submitted by

Name: Md. Rafatul Haque
ID:IT-17037
3rd year 2nd semester
Session: 2016-2017
Dept. of ICT
MBSTU.

Submitted To

Nazrul Islam
Assistant Professor
Dept. of ICT
MBSTU.

Lab Report Name: SDN Controllers and Mininet.

Objective:

- Install and use traffic generators as powerful tools for testing network performance.
- Install and configure SDN Controller.
- Install and understand how the mininet simulator works.
- Implement and run basic examples for understanding the role of the controller and how it interact with mininet.

Theory:

Traffic Generator:

What is iPerf?: iPerf is a tool for active measurements of the maximum achievable bandwidth
on IP networks. It supports tuning of various parameters related to timing, buffers and protocols
(TCP, UDP, SCTP with IPv4 and IPv6). For each test it reports the bandwidth, loss, and other
parameters.

Controller:

OVS-testcontroller is a simple OpenFlow controller that manages any number of switches over
the OpenFlow protocol, causing them to function as L2 MAC-learning switches or hubs. It is
suitable for initial testing of OpenFlow networks.

Ryu is a component-based software defined networking framework. Ryu provides software
components with well-defined API that make it easy for developers to create new network
management and control applications. Ryu supports various protocols for managing network
devices, such as OpenFlow, Netconf, OF-config, etc. About OpenFlow, Ryu supports fully 1.0,
1.2, 1.3, 1.4, 1.5 and Nicira Extensions. All of the code is freely available under the Apache 2.0
license.

Mininet: Mininet creates a realistic virtual network, running real kernel, switch and application code, on a single machine (VM, cloud or native) Because you can easily interact
with your network using the Mininet CLI (and API), customize it, share it with others, or deploy it
on real hardware, Mininet is useful for development, teaching, and research. Mininet is also a
great way to develop, share, and experiment with OpenFlow and Software-Defined Networking
systems.

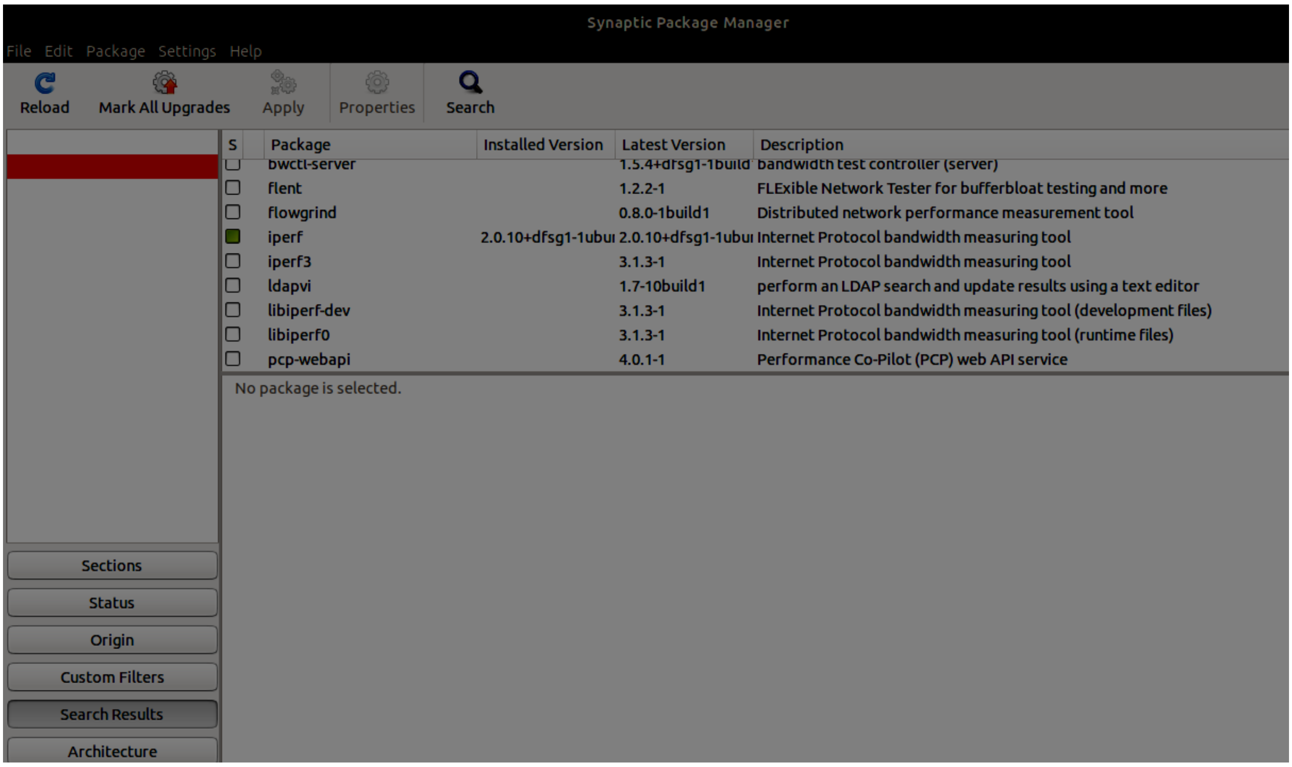
Methodology

TIP: For getting extra space in your USB-please use the following tips:

- Empty the trash
- Delete the Android related staff
- Delete the extras for other courses
- Delete the already installed package sources

Install iperf

1. Open the Synaptic Package Manager (Navigator ->System-> Synaptic Package Manager)
2. Setup the proxy:
    - Click on settings-> Preference -> Network
    - Click on manual proxy configuration
    - HTT and FTP Proxy: proxy.rmit.edu.au Port: 8080
3. Search for Quick filter `iperf`
4. Click on Mark for installation
5. Then click on Apply and wait until the package is installed



Install mininet

1. In Synaptic Package
2. Search for Quick filter `mininet`
3. Click on Mark for installation
4. Then click on Apply and wait until the package is installed

Install Controller

OVS controller:

1. In Synaptic Package
2. Search for Quick filter `openvswitch-controller`
3. Click on Mark for installation
4. Then click on Apply and wait until the package is installed

Part 1: Everyday Mininet Usage

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

```
rafatul@rafatul-HP-Notebook:~$ sudo mn -h
[sudo] password for rafatul:
Usage: mn [options]
(type mn -h for details)

The mn utility creates Mininet network from the command line. It can create
parametrized topologies, invoke the Mininet CLI, and run tests.

Options:
  -h, --help            show this help message and exit
  --switch=SWITCH       default|ivs|lxbr|ovs|ovsbr|ovsk|user[,param=value...]
                        ovs=OVSSwitch default=OVSSwitch ovsk=OVSSwitch
                        lxbr=LinuxBridge user=UserSwitch ivs=IVSSwitch
                        ovsbr=OVSBridge
  --host=HOST           cfs|proc|rt[,param=value...]
                        rt=CPULimitedHost{'sched': 'rt'} proc=Host
                        cfs=CPULimitedHost{'sched': 'cfs'}
  --controller=CONTROLLER
                        default|none|nox|ovsc|ref|remote|ryu[,param=value...]
                        ovsc=OVSController none=NullController
                        remote=RemoteController default=DefaultController
                        nox=NOX ryu=Ryu ref=Controller
  --link=LINK           default|ovs|tc|tcu[,param=value...] default=Link
                        ovs=OVSLink tcu=TCULink tc=TCLink
  --topo=TOPO           linear|minimal|reversed|single|torus|tree[,param=value
                        ...] linear=LinearTopo torus=TorusTopo tree=TreeTopo
                        single=SingleSwitchTopo
                        reversed=SingleSwitchReversedTopo minimal=MinimalTopo
  -c, --clean           clean and exit
  --custom=CUSTOM       read custom classes or params from .py file(s)
  --test=TESTS          cli|build|pingall|pingpair|iperf|all|iperfudp|none|pin
                        gpair|iperfudp|pingall|iperfUDP
  -x, --xterms          spawn xterms for each node
  -i IPBASE, --ipbase=IPBASE
                        base IP address for hosts
  --mac                 automatically set host MACs
  --arp                 set all-pairs ARP entries
  -v VERBOSITY, --verbosity=VERBOSITY
```

```
                        info|warning|critical|error|debug|output
  --innamespace         sw and ctrl in namespace?
  --listenport=LISTENPORT
                        base port for passive switch listening
  --nolistenport        don't use passive listening port
  --pre=PRE             CLI script to run before tests
  --post=POST           CLI script to run after tests
  --pin                 pin hosts to CPU cores (requires --host cfs or --host
                        rt)
  --nat                 [option=val...] adds a NAT to the topology that
                        connects Mininet hosts to the physical network.
                        Warning: This may route any traffic on the machine
                        that uses Mininet's IP subnet into the Mininet
                        network. If you need to change Mininet's IP subnet,
                        see the --ipbase option.
  --version             prints the version and exits
  --cluster=server1,server2...
                        run on multiple servers (experimental!)
  --placement=block|random
                        node placement for --cluster (experimental!)
```

Start Wireshark

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

$ sudo wireshark &

```
rafatul@rafatul-HP-Notebook:~$ sudo wireshark &
[1] 10191
rafatul@rafatul-HP-Notebook:~$ QStandardPaths: XDG_RUNTIME_DIR not set, defaulti
ng to '/tmp/runtime-root'
```

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.

Interact with Hosts and Switches

Start a minimal topology and enter the CLI:
$ sudo mn

```
rafatul@rafatul-HP-Notebook:~$ 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>
```

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

```
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
```

Display nodes:

mininet>nodes

```
mininet> nodes
available nodes are:
c0 h1 h2 s1
```

Display links:

mininet>net

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
```

Dump information about all nodes:

mininet>dump

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=12056>
<Host h2: h2-eth0:10.0.0.2 pid=12058>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=12063>
<OVSController c0: 127.0.0.1:6653 pid=12049>
```

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

```
mininet> h1 ifconfig -a
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::84df:20ff:fe69:2d9a  prefixlen 64  scopeid 0x20<link>
        ether 86:df:20:69:2d:9a  txqueuelen 1000  (Ethernet)
        RX packets 47  bytes 6049 (6.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 16  bytes 1216 (1.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

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
enp1s0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 30:e1:71:92:19:b9  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 9062  bytes 700457 (700.4 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9062  bytes 700457 (700.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ovs-system: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether 1a:ef:5a:36:dd:ca  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether 7a:77:14:a8:9c:4a  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 20  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::5085:e0ff:fe29:44a8  prefixlen 64  scopeid 0x20<link>
        ether 52:85:e0:29:44:a8  txqueuelen 1000  (Ethernet)
        RX packets 16  bytes 1216 (1.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
```

mininet>s1 ifconfig -a

```
s1: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether 7a:77:14:a8:9c:4a  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 20  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::5085:e0ff:fe29:44a8  prefixlen 64  scopeid 0x20<link>
        ether 52:85:e0:29:44:a8  txqueuelen 1000  (Ethernet)
        RX packets 16  bytes 1216 (1.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 47  bytes 6049 (6.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::fc0f:67ff:feb1:12d6  prefixlen 64  scopeid 0x20<link>
        ether fe:0f:67:b1:12:d6  txqueuelen 1000  (Ethernet)
        RX packets 16  bytes 1216 (1.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 47  bytes 6069 (6.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::a5ea:3d39:2263:3213  prefixlen 64  scopeid 0x20<link>
        ether 30:e3:7a:4e:64:70  txqueuelen 1000  (Ethernet)
        RX packets 209382  bytes 262490176 (262.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 157891  bytes 21959411 (21.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

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

```
mininet> h1 ps -a
  PID TTY          TIME CMD
 1190 tty1     00:00:00 gnome-session-b
 1256 tty1     00:00:23 gnome-shell
 1289 tty1     00:00:00 Xwayland
 1331 tty1     00:00:00 ibus-daemon
 1334 tty1     00:00:00 ibus-dconf
 1336 tty1     00:00:00 ibus-x11
 1396 tty1     00:00:00 gsd-xsettings
 1399 tty1     00:00:00 gsd-a11y-settin
 1400 tty1     00:00:00 gsd-clipboard
 1401 tty1     00:00:01 gsd-color
 1404 tty1     00:00:00 gsd-datetime
 1405 tty1     00:00:00 gsd-housekeepin
 1406 tty1     00:00:00 gsd-keyboard
 1410 tty1     00:00:00 gsd-media-keys
 1411 tty1     00:00:00 gsd-mouse
 1412 tty1     00:00:00 gsd-power
 1413 tty1     00:00:00 gsd-print-notif
 1416 tty1     00:00:00 gsd-rfkill
 1421 tty1     00:00:00 gsd-screensaver
 1424 tty1     00:00:00 gsd-sharing
 1428 tty1     00:00:00 gsd-smartcard
 1431 tty1     00:00:00 gsd-sound
 1433 tty1     00:00:00 gsd-wacom
 1447 tty1     00:00:00 ibus-engine-sim
 1628 tty2     00:06:43 Xorg
 1645 tty2     00:00:00 gnome-session-b
 1784 tty2     00:08:49 gnome-shell
 1830 tty2     00:00:04 ibus-daemon
 1834 tty2     00:00:00 ibus-dconf
 1838 tty2     00:00:00 ibus-x11
 1911 tty2     00:00:00 gsd-power
 1912 tty2     00:00:00 gsd-print-notif
 1915 tty2     00:00:00 gsd-rfkill
 1919 tty2     00:00:00 gsd-screensaver
 1922 tty2     00:00:01 gsd-sharing
```

```
1922 tty2     00:00:01 gsd-sharing
1928 tty2     00:00:00 gsd-smartcard
1930 tty2     00:00:00 gsd-sound
1936 tty2     00:00:00 gsd-xsettings
1937 tty2     00:00:00 gsd-wacom
1947 tty2     00:00:00 gsd-a11y-settin
1948 tty2     00:00:00 gsd-clipboard
1953 tty2     00:00:01 gsd-color
1954 tty2     00:00:00 gsd-datetime
1955 tty2     00:00:00 gsd-housekeepin
1956 tty2     00:00:00 gsd-keyboard
1960 tty2     00:00:00 gsd-media-keys
1965 tty2     00:00:00 gsd-mouse
1985 tty2     00:00:00 gsd-printer
2032 tty2     00:00:00 gsd-disk-utilit
2042 tty2     00:00:02 nautilus-deskto
2085 tty2     00:00:01 ibus-engine-sim
2502 tty2     00:00:06 gnome-software
2508 tty2     00:00:00 update-notifier
4084 tty2     00:00:00 deja-dup-monito
5161 tty2     00:04:11 MainThread
5555 tty2     00:00:13 Privileged Cont
5685 tty2     00:00:08 WebExtensions
5743 tty2     00:05:28 Web Content
6570 tty2     00:04:19 chrome
6577 tty2     00:00:00 cat
6578 tty2     00:00:00 cat
6591 tty2     00:00:00 chrome
6592 tty2     00:00:00 chrome
6602 tty2     00:00:00 nacl_helper
6603 tty2     00:00:00 nacl_helper
6606 tty2     00:00:00 chrome
6634 tty2     00:05:14 chrome
6641 tty2     00:01:24 chrome
6700 tty2     00:00:21 chrome
6717 tty2     00:00:00 chrome
6728 tty2     00:00:00 chrome
```

```
 6591 tty2     00:00:00 chrome
 6592 tty2     00:00:00 chrome
 6602 tty2     00:00:00 nacl_helper
 6603 tty2     00:00:00 nacl_helper
 6606 tty2     00:00:00 chrome
 6634 tty2     00:05:14 chrome
 6641 tty2     00:01:24 chrome
 6700 tty2     00:00:21 chrome
 6717 tty2     00:00:00 chrome
 6728 tty2     00:00:00 chrome
 7402 tty2     00:00:00 RDD Process
 9530 tty2     00:01:14 chrome
 9554 tty2     00:00:02 chrome
 9922 tty2     00:00:04 chrome
10191 pts/0    00:00:00 sudo
10192 pts/0    00:01:14 wireshark
10200 pts/0    00:00:00 dbus-launch
10379 pts/0    00:00:02 dumpcap
10570 tty2     00:03:26 chrome
11866 tty2     00:00:37 Web Content
12037 pts/0    00:00:00 sudo
12038 pts/0    00:00:00 mn
12081 pts/1    00:00:01 ovs-testcontrol
13461 tty2     00:00:12 chrome
13555 tty2     00:00:24 Web Content
14085 tty2     00:00:01 chrome
14124 tty2     00:00:00 chrome
14139 tty2     00:00:00 Web Content
```

This should be the exact same as that seen by the root network namespace:

mininet> s1 ps -a

```
mininet> s1 ps -a
  PID TTY          TIME CMD
 1190 tty1     00:00:00 gnome-session-b
 1256 tty1     00:00:24 gnome-shell
 1289 tty1     00:00:00 Xwayland
 1331 tty1     00:00:00 ibus-daemon
 1334 tty1     00:00:00 ibus-dconf
 1336 tty1     00:00:00 ibus-x11
 1396 tty1     00:00:00 gsd-xsettings
 1399 tty1     00:00:00 gsd-a11y-settin
 1400 tty1     00:00:00 gsd-clipboard
 1401 tty1     00:00:01 gsd-color
 1404 tty1     00:00:00 gsd-datetime
 1405 tty1     00:00:00 gsd-housekeepin
 1406 tty1     00:00:00 gsd-keyboard
 1410 tty1     00:00:00 gsd-media-keys
 1411 tty1     00:00:00 gsd-mouse
 1412 tty1     00:00:00 gsd-power
 1413 tty1     00:00:00 gsd-print-notif
 1416 tty1     00:00:00 gsd-rfkill
 1421 tty1     00:00:00 gsd-screensaver
 1424 tty1     00:00:00 gsd-sharing
 1428 tty1     00:00:00 gsd-smartcard
 1431 tty1     00:00:00 gsd-sound
 1433 tty1     00:00:00 gsd-wacom
 1447 tty1     00:00:00 ibus-engine-sim
 1628 tty2     00:06:55 Xorg
 1645 tty2     00:00:00 gnome-session-b
 1784 tty2     00:09:10 gnome-shell
 1830 tty2     00:00:05 ibus-daemon
 1834 tty2     00:00:00 ibus-dconf
 1838 tty2     00:00:00 ibus-x11
 1911 tty2     00:00:00 gsd-power
 1912 tty2     00:00:00 gsd-print-notif
 1915 tty2     00:00:00 gsd-rfkill
 1919 tty2     00:00:00 gsd-screensaver
 1922 tty2     00:00:01 gsd-sharing
```

```
1922 tty2     00:00:01 gsd-sharing
1928 tty2     00:00:00 gsd-smartcard
1930 tty2     00:00:00 gsd-sound
1936 tty2     00:00:00 gsd-xsettings
1937 tty2     00:00:00 gsd-wacom
1947 tty2     00:00:00 gsd-a11y-settin
1948 tty2     00:00:00 gsd-clipboard
1953 tty2     00:00:01 gsd-color
1954 tty2     00:00:00 gsd-datetime
1955 tty2     00:00:00 gsd-housekeepin
1956 tty2     00:00:00 gsd-keyboard
1960 tty2     00:00:00 gsd-media-keys
1965 tty2     00:00:00 gsd-mouse
1985 tty2     00:00:00 gsd-printer
2032 tty2     00:00:00 gsd-disk-utilit
2042 tty2     00:00:02 nautilus-deskto
2085 tty2     00:00:01 ibus-engine-sim
2502 tty2     00:00:06 gnome-software
2508 tty2     00:00:00 update-notifier
4084 tty2     00:00:00 deja-dup-monito
5161 tty2     00:04:12 MainThread
5555 tty2     00:00:13 Privileged Cont
5685 tty2     00:00:09 WebExtensions
5743 tty2     00:05:42 Web Content
6570 tty2     00:04:21 chrome
6577 tty2     00:00:00 cat
6578 tty2     00:00:00 cat
6591 tty2     00:00:00 chrome
6592 tty2     00:00:00 chrome
6602 tty2     00:00:00 nacl_helper
6603 tty2     00:00:00 nacl_helper
6606 tty2     00:00:00 chrome
6634 tty2     00:05:14 chrome
6641 tty2     00:01:25 chrome
6700 tty2     00:00:21 chrome
6717 tty2     00:00:00 chrome
6728 tty2     00:00:00 chrome
```

```
 6592 tty2     00:00:00 chrome
 6602 tty2     00:00:00 nacl_helper
 6603 tty2     00:00:00 nacl_helper
 6606 tty2     00:00:00 chrome
 6634 tty2     00:05:14 chrome
 6641 tty2     00:01:25 chrome
 6700 tty2     00:00:21 chrome
 6717 tty2     00:00:00 chrome
 6728 tty2     00:00:00 chrome
 7402 tty2     00:00:00 RDD Process
 9530 tty2     00:01:15 chrome
 9554 tty2     00:00:02 chrome
 9922 tty2     00:00:04 chrome
10191 pts/0    00:00:00 sudo
10192 pts/0    00:01:16 wireshark
10200 pts/0    00:00:00 dbus-launch
10379 pts/0    00:00:02 dumpcap
10570 tty2     00:03:26 chrome
11866 tty2     00:00:38 Web Content
12037 pts/0    00:00:00 sudo
12038 pts/0    00:00:00 mn
12081 pts/1    00:00:01 ovs-testcontrol
13461 tty2     00:00:12 chrome
13555 tty2     00:00:25 Web Content
14085 tty2     00:00:01 chrome
14124 tty2     00:00:00 chrome
14139 tty2     00:00:00 Web Content
15087 pts/4    00:00:00 ps
```

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

```
mininet> h1 ping -c 1 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=9.21 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 9.219/9.219/9.219/0.000 ms
```

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

```
mininet> h1 ping -c 1 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.32 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.323/1.323/1.323/0.000 ms
```

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

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
```

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..)

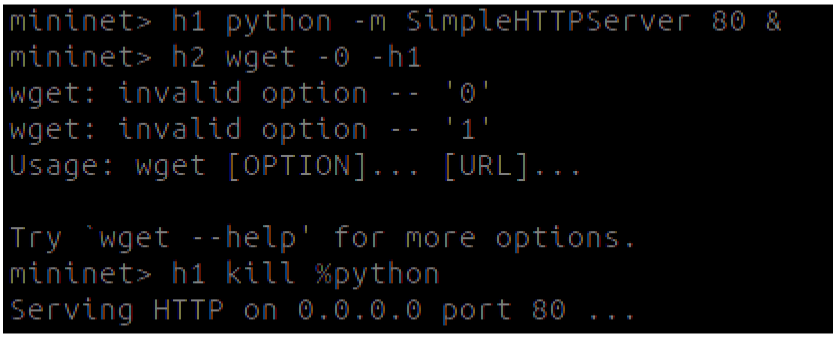
Next, try starting a simple HTTP server on **h1**, making a request from **h2**, then shutting down the web server:

mininet> h1 python -m SimpleHTTPServer 80 &

mininet> h2 wget -O – h1

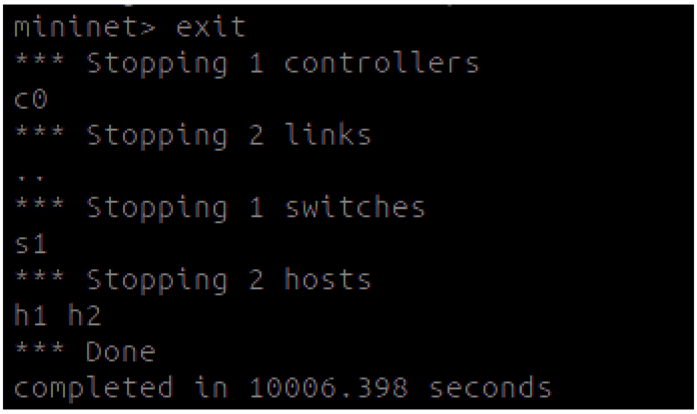
...

mininet> h1 kill %python



Exit the CLI:

mininet> exit



Cleanup

If Mininet crashes for some reason, clean it up:

$ sudo mn -c

```
rafatul@rafatul-HP-Notebook:~$ sudo mn -c
[sudo] password for rafatul:
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd ovs-controller udpbwtest mnexec ivs 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd ovs-controller udpbwtest mnexec ivs 2> /dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
***  Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
rafatul@rafatul-HP-Notebook:~$
```