MAWLANA BHASHANI SCIENCE AND TECHNOLOGY UNIVERSITY

Santosh,Tangail-1902

LAB REPORT

Lab Report No	: 01
Lab Report name	: Mininet walkthrough
Course Title	: Network Planning and designing Lab.
Course Code	: ICT-3208
Date of Performance	: 04-11-2020
Date of Submission	: 04-11-2020

Submitted by,

Student Name : Tanvir Ahmed

Student ID : IT-18043

Session : 2017-18

3rd Year 2nd semester

Dept. of ICT

Submitted to,

Nazrul Islam

Assistant Professor

Dept. of ICT,

MBSTU.

Lab report – 01

Lab report Name : Mininet walkthrough .

Objectives : Setup mininet emulator and understand deeply how virtual hosts , switches , controllers and links work in associating with SDN and how these supports Openflow for highly flexible custom routing .

Explanation :

What is Mininet?

Mininet emulates a complete network of hosts, links, and switches on a single machine. To create a sample two-host, one-switch network, just run:

```
sudo mn
```

Mininet is useful for interactive development, testing, and demos, especially those using OpenFlow and SDN. OpenFlow-based network controllers prototyped in Mininet can usually be transferred to hardware with minimal changes for full line-rate execution.

How does it work?

Mininet creates virtual networks using process-based virtualization and network namespaces - features that are available in recent Linux kernels. In Mininet, hosts are emulated as bash processes running in a network namespace, so any code that would normally run on a Linux server (like a web server or client program) should run just fine within a Mininet "Host". The Mininet "Host" will have its own private network interface and can only see its own processes. Switches in Mininet are software-based switches like Open vSwitch or the OpenFlow reference switch. Links are virtual ethernet pairs, which live in the Linux kernel and connect our emulated switches to emulated hosts (processes).

How to install Mininet ?

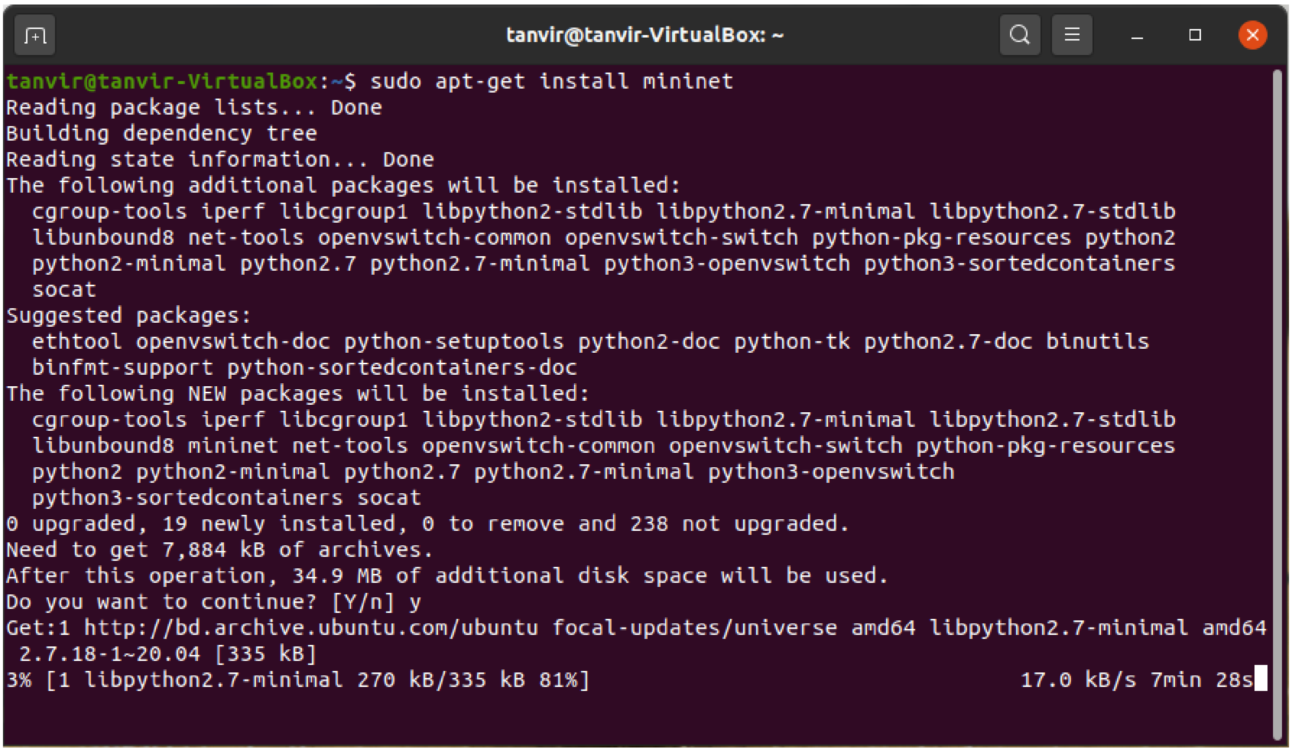
There are multiple choices to install mininet :

1. **Easiest "installation" - use pre-built VM image**

   https://github.com/mininet/mininet/releases

2. **Next-easiest option: use Ubuntu package**

   sudo apt-get install mininet



```
tanvir@tanvir-VirtualBox: ~
tanvir@tanvir-VirtualBox:~$ sudo apt-get install mininet
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  cgroup-tools iperf libcgroup1 libpython2-stdlib libpython2.7-minimal libpython2.7-stdlib
  libunbound8 net-tools openvswitch-common openvswitch-switch python-pkg-resources python2
  python2-minimal python2.7 python2.7-minimal python3-openvswitch python3-sortedcontainers
  socat
Suggested packages:
  ethtool openvswitch-doc python-setuptools python2-doc python-tk python2.7-doc binutils
  binfmt-support python-sortedcontainers-doc
The following NEW packages will be installed:
  cgroup-tools iperf libcgroup1 libpython2-stdlib libpython2.7-minimal libpython2.7-stdlib
  libunbound8 mininet net-tools openvswitch-common openvswitch-switch python-pkg-resources
  python2 python2-minimal python2.7 python2.7-minimal python3-openvswitch
  python3-sortedcontainers socat
0 upgraded, 19 newly installed, 0 to remove and 238 not upgraded.
Need to get 7,884 kB of archives.
After this operation, 34.9 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://bd.archive.ubuntu.com/ubuntu focal-updates/universe amd64 libpython2.7-minimal amd64
 2.7.18-1~20.04 [335 kB]
3% [1 libpython2.7-minimal 270 kB/335 kB 81%]                          17.0 kB/s 7min 28s
```

3. **Native installation from source.**

   git clone git://github.com/mininet/mininet.git

Everyday Mininet Usage :

Mininet help message :

sudo mn –h

```
tanvir@tanvir-VirtualBox: ~
tanvir@tanvir-VirtualBox:~$ mn -h
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
```

Interact with Hosts and Switches :

sudo mn

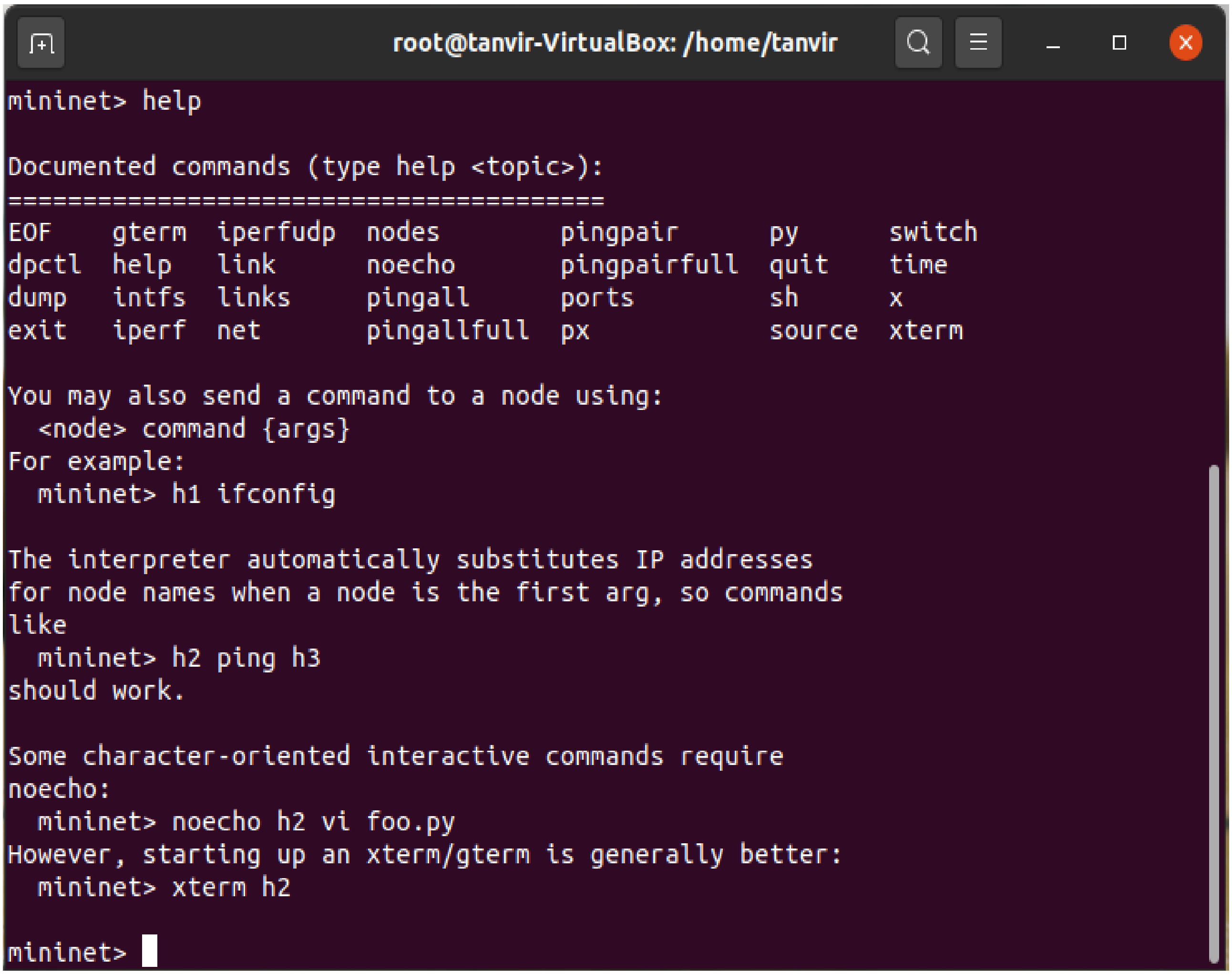
```
root@tanvir-VirtualBox: /home/tanvir
root@tanvir-VirtualBox:/home/tanvir# mn
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

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.

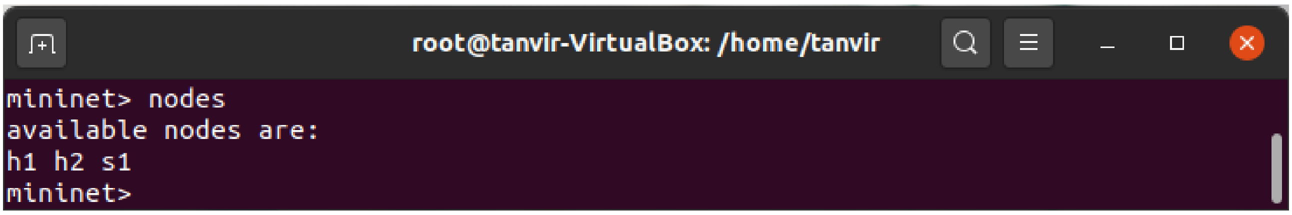
Display Mininet CLI commands:

mininet> help



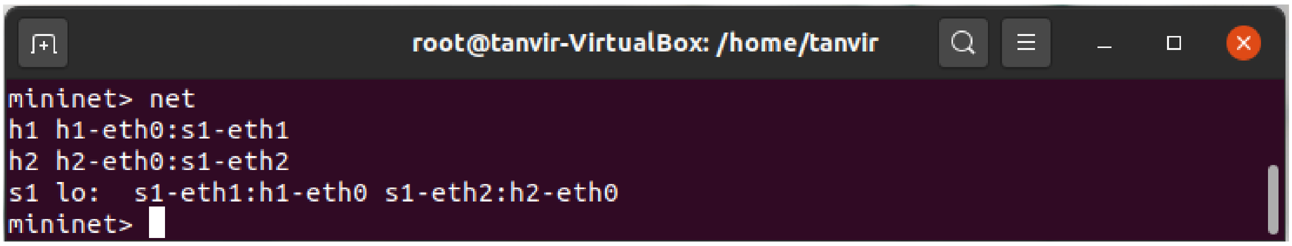
Display nodes:

mininet> nodes



Display links:

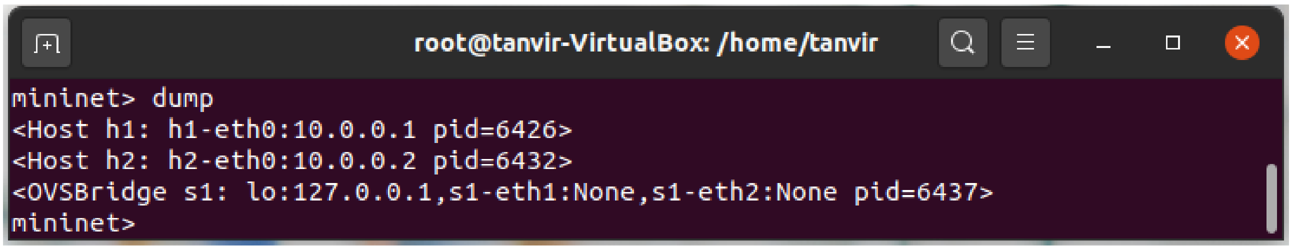
mininet> net



```
root@tanvir-VirtualBox: /home/tanvir
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
mininet>
```

Dump information about all nodes:

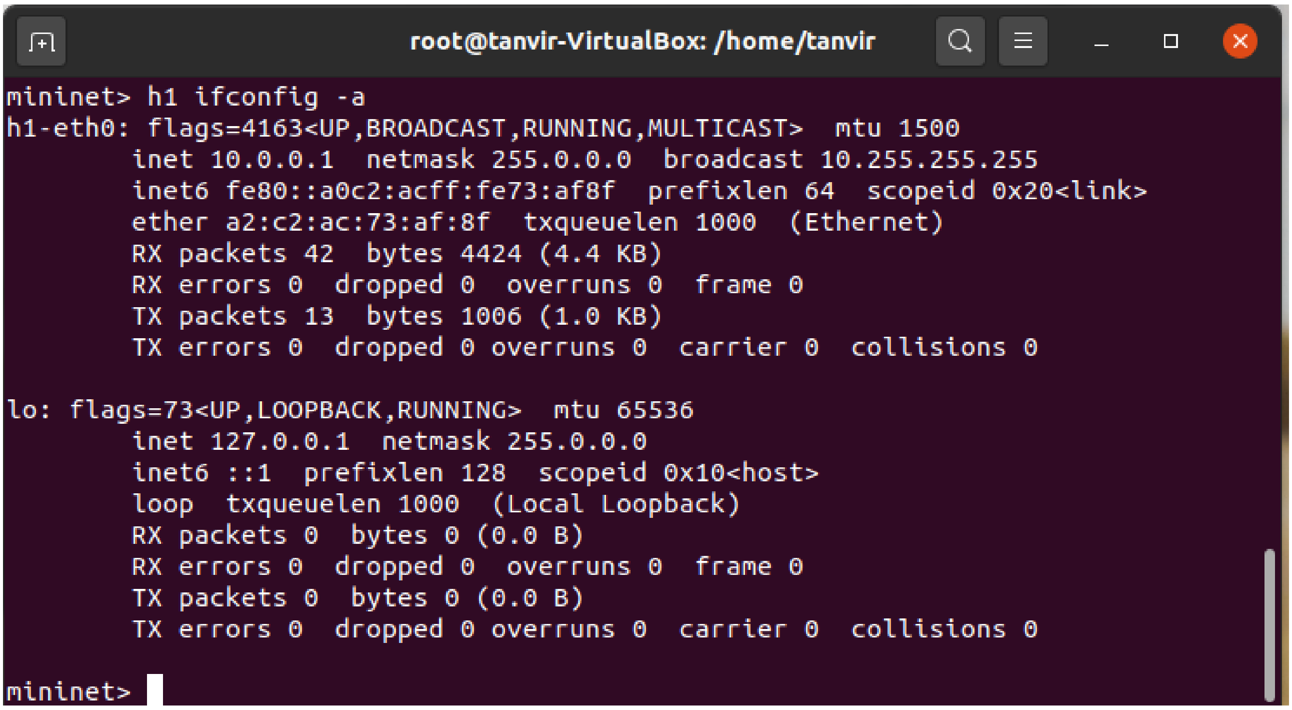
mininet> dump



```
root@tanvir-VirtualBox: /home/tanvir
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=6426>
<Host h2: h2-eth0:10.0.0.2 pid=6432>
<OVSBridge s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=6437>
mininet>
```

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



```
root@tanvir-VirtualBox: /home/tanvir
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::a0c2:acff:fe73:af8f  prefixlen 64  scopeid 0x20<link>
        ether a2:c2:ac:73:af:8f  txqueuelen 1000  (Ethernet)
        RX packets 42  bytes 4424 (4.4 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 13  bytes 1006 (1.0 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

mininet>
```

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

root@tanvir-VirtualBox: /home/tanvir

```
mininet> s1 ifconfig -a
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::903d:a4f4:4244:f7a6  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:62:ad:cb  txqueuelen 1000  (Ethernet)
        RX packets 14650  bytes 16474659 (16.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4775  bytes 350923 (350.9 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 272  bytes 24284 (24.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 272  bytes 24284 (24.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ovs-system: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether 22:7c:9e:43:6f:6b  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 6e:f4:42:59:af:41  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 24  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::bcf3:48ff:fe08:c7ef  prefixlen 64  scopeid 0x20<link>
        ether be:f3:48:08:c7:ef  txqueuelen 1000  (Ethernet)
        RX packets 13  bytes 1006 (1.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 42  bytes 4424 (4.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::f0dd:8cff:fe81:ee24  prefixlen 64  scopeid 0x20<link>
        ether f2:dd:8c:81:ee:24  txqueuelen 1000  (Ethernet)
        RX packets 13  bytes 1006 (1.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 43  bytes 4494 (4.4 KB)
        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.

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

```
root@tanvir-VirtualBox: /home/tanvir

mininet> h1 ps -a
    PID TTY          TIME CMD
   1011 tty2     00:00:15 Xorg
   1118 tty2     00:00:00 gnome-session-b
   6409 pts/0    00:00:00 sudo
   6410 pts/0    00:00:00 bash
   6418 pts/0    00:00:00 mn
   6749 pts/1    00:00:00 ps
mininet>
```

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

mininet> s1 ps –a

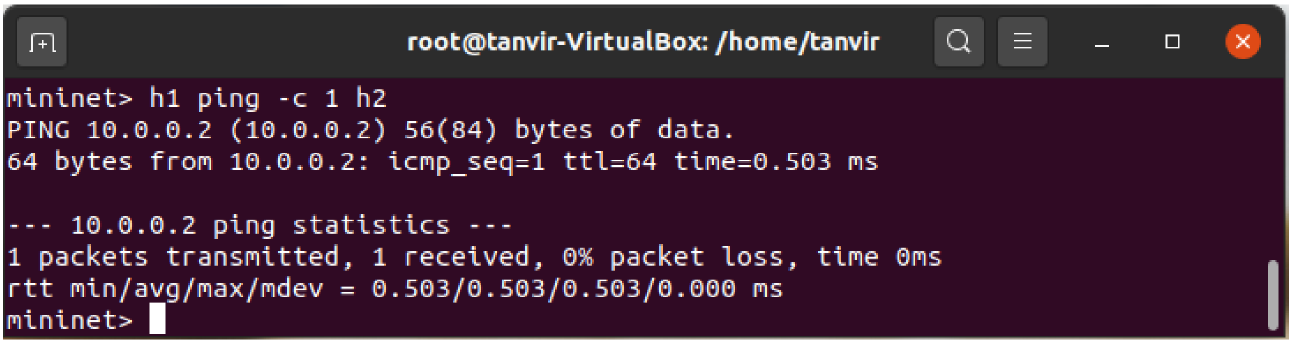
```
root@tanvir-VirtualBox: /home/tanvir

mininet> s1 ps -a
    PID TTY          TIME CMD
   1011 tty2     00:00:16 Xorg
   1118 tty2     00:00:00 gnome-session-b
   6409 pts/0    00:00:00 sudo
   6410 pts/0    00:00:00 bash
   6418 pts/0    00:00:00 mn
   6751 pts/3    00:00:00 ps
mininet>
```

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



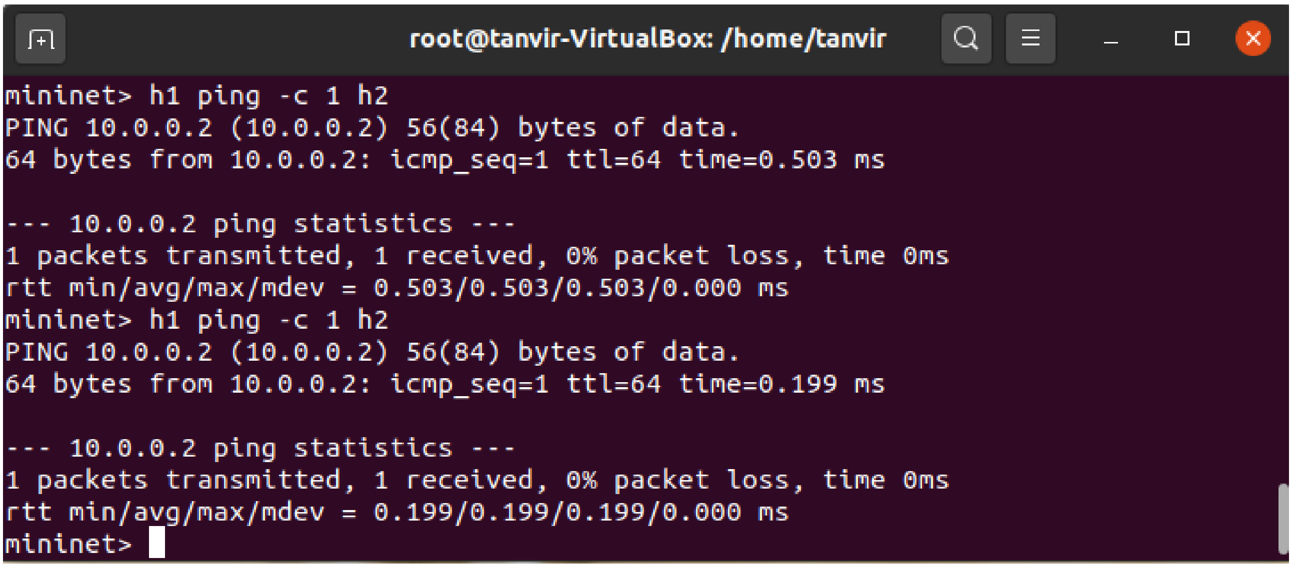
If a string appears later in the command with a node name, that node name is replaced by its IP address; this happened for h2.

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:

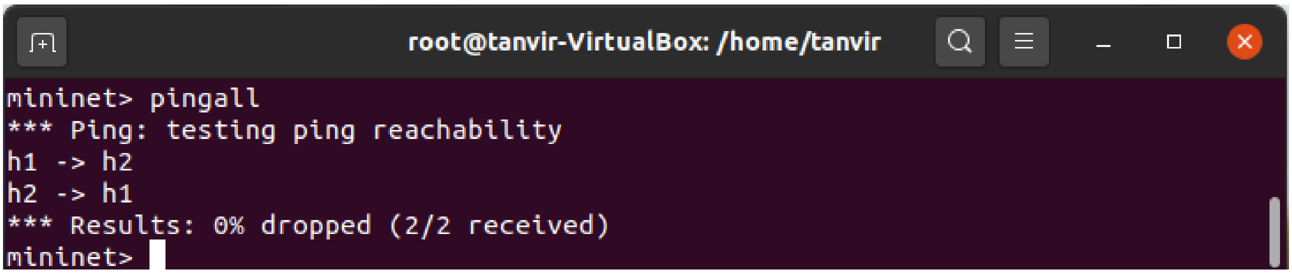
mininet> h1 ping -c 1 h2



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:

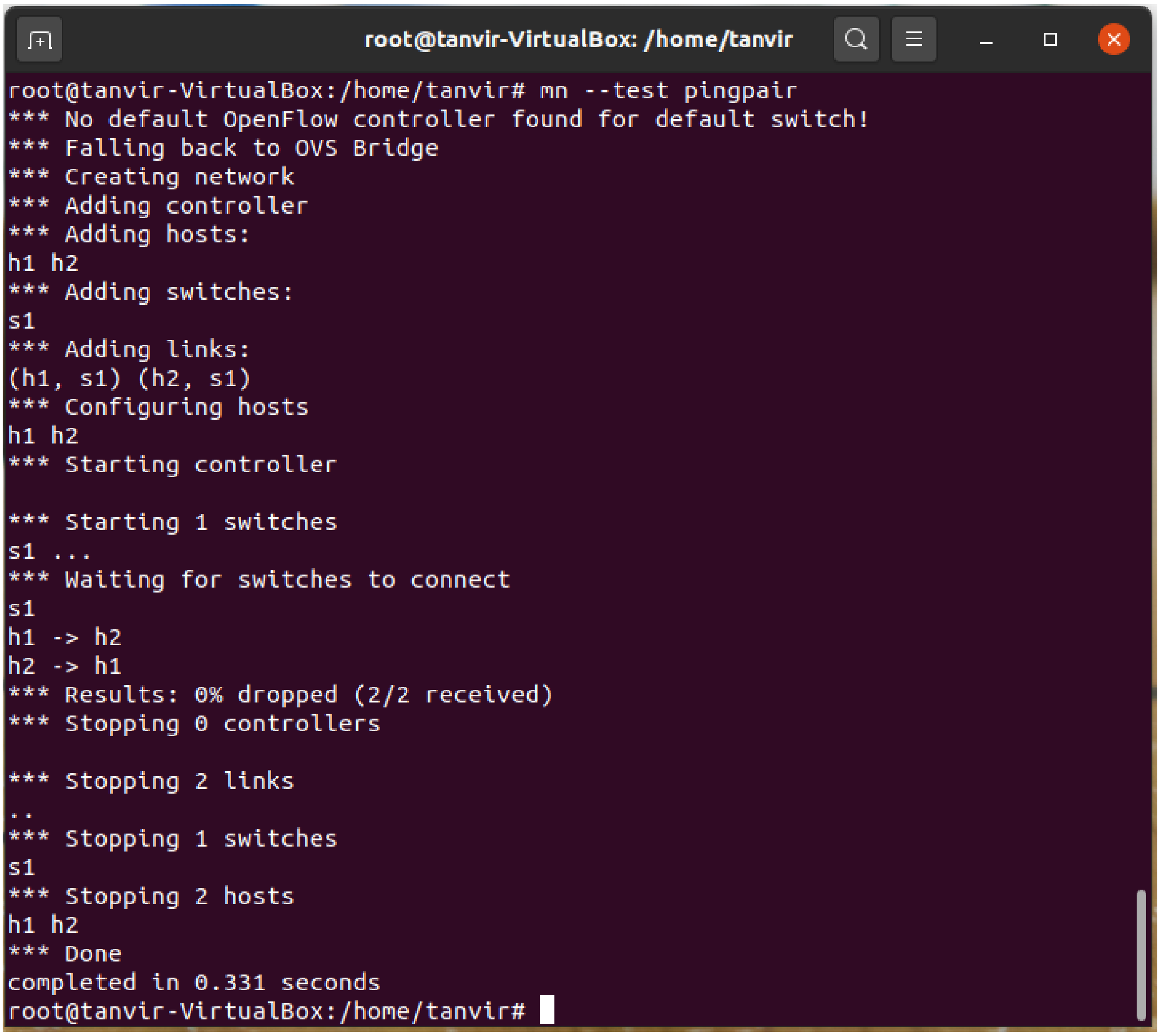
mininet> pingall



Advanced Startup Options

Run a Regression Test :

$ sudo mn --test pingpair

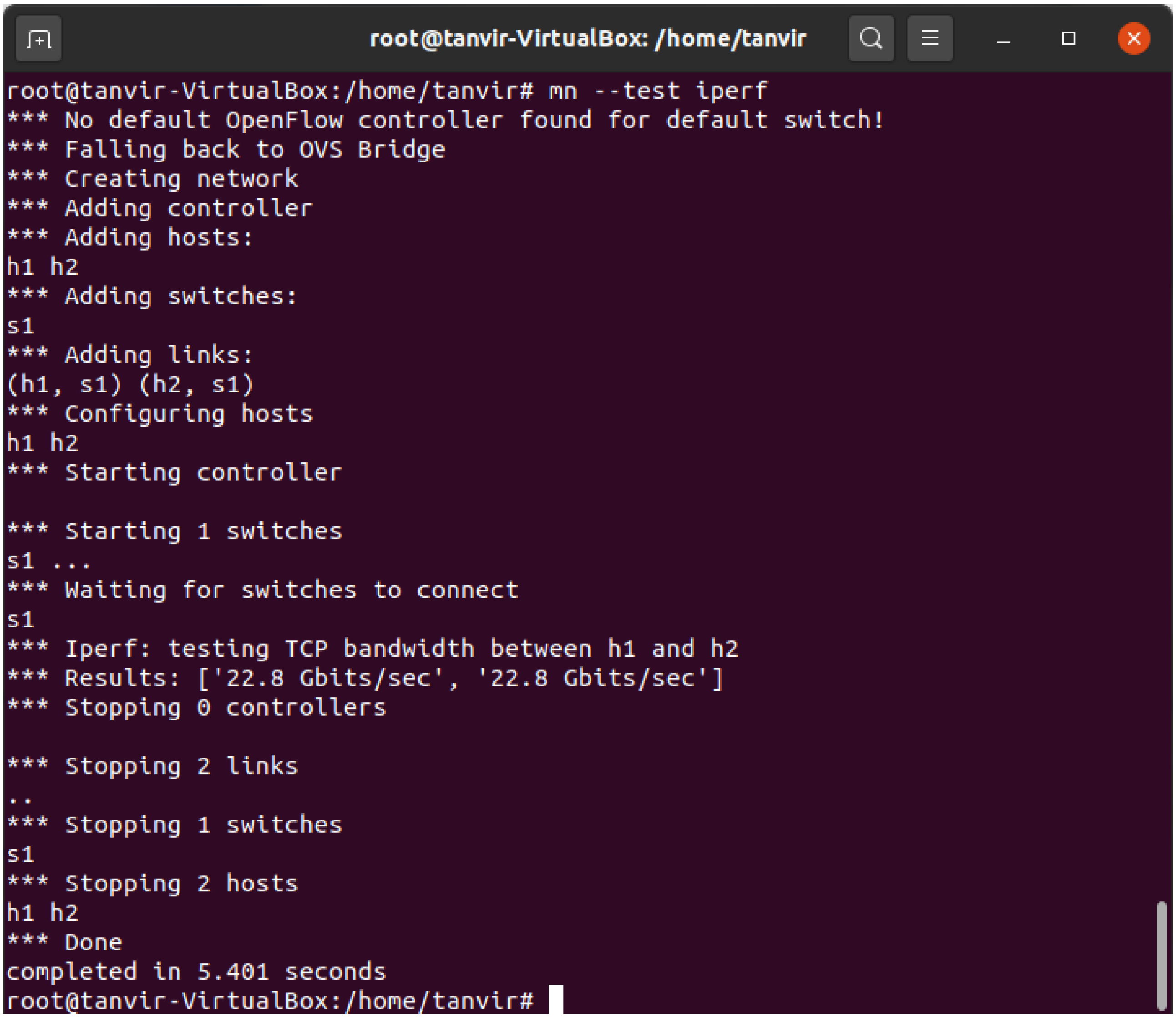


```
root@tanvir-VirtualBox:/home/tanvir# mn --test pingpair
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 0 controllers

*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 0.331 seconds
root@tanvir-VirtualBox:/home/tanvir#
```

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



```
root@tanvir-VirtualBox: /home/tanvir
root@tanvir-VirtualBox:/home/tanvir# mn --test iperf
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['22.8 Gbits/sec', '22.8 Gbits/sec']
*** Stopping 0 controllers

*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 5.401 seconds
root@tanvir-VirtualBox:/home/tanvir#
```

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.

Conclusion: This experiment was done in Ubuntu operating system . Here I learned how to setup mininet . It was easy. Then I used some basic commands in mininet terminal and have come to know how they are used in a correct way. I think mininet is a very helpful emulator where we have the limitation in hardware and time . It makes simple to plan , test and apply in real life networking . I really enjoyed this experiment and I will keep learning this emulator so that it can be used in a efficient way .