Mawlana Bhashani Science and Technology University

Santosh, Tangail-1902.

Lab Report

Department of Information and Communication Technology

Report No: 04

Report Name: SDN Controllers and Mininet.

Course Title: Network Planning and designing Lab.

Course Code: ICT-3208

Submitted By	Submitted To
Name: **Md Sohanur Islam** ID: **IT-18011** Session: 2017-18 3rd Year 2nd Semester Dept. of Information & Communication Technology, MBSTU.	**Nazrul Islam** **Assistant Professor,** Dept. of Information & Communication Technology, MBSTU.

Objectives : The main objectives of the lab how to 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:

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.

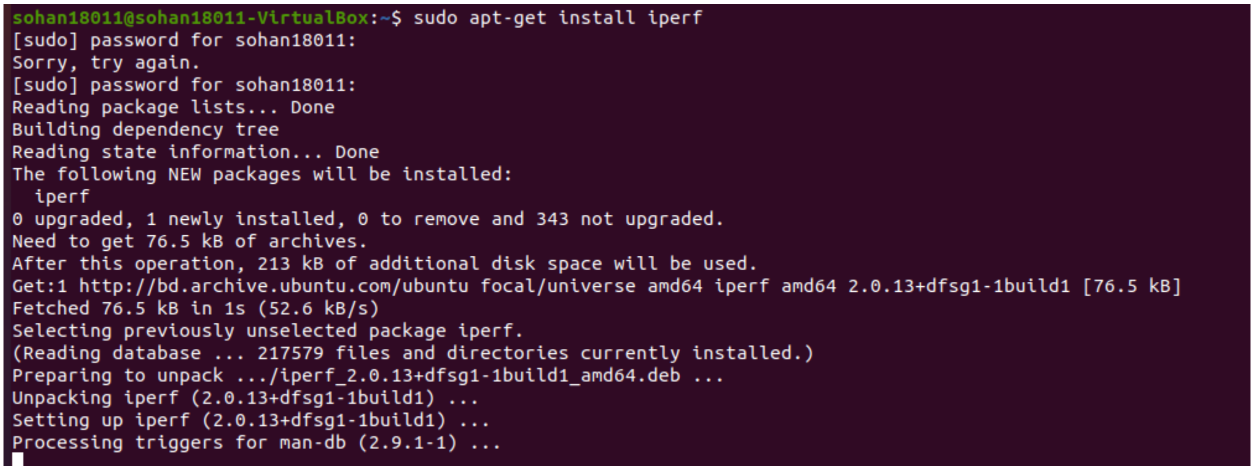
Software Defined Networking: Software Defined Networking that by separating control of network functions from hardware devices, administrators acquire more power to route and direct traffic in response to changing requirements.

Controller: Controller is suitable for initial testing of OpenFlow networks. 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.

Mininet : Mininet creates a realistic virtual network, running real kernel, switch and application code, on a single machine .

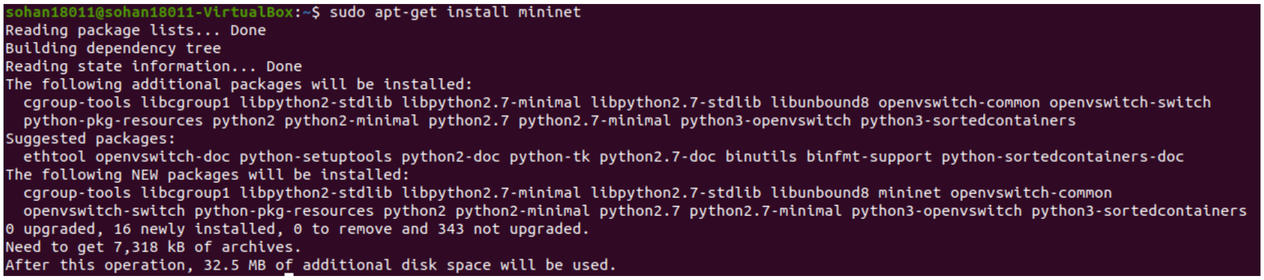
Methodology :

Install iperf :



```
sohan18011@sohan18011-VirtualBox:~$ sudo apt-get install iperf
[sudo] password for sohan18011:
Sorry, try again.
[sudo] password for sohan18011:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  iperf
0 upgraded, 1 newly installed, 0 to remove and 343 not upgraded.
Need to get 76.5 kB of archives.
After this operation, 213 kB of additional disk space will be used.
Get:1 http://bd.archive.ubuntu.com/ubuntu focal/universe amd64 iperf amd64 2.0.13+dfsg1-1build1 [76.5 kB]
Fetched 76.5 kB in 1s (52.6 kB/s)
Selecting previously unselected package iperf.
(Reading database ... 217579 files and directories currently installed.)
Preparing to unpack .../iperf_2.0.13+dfsg1-1build1_amd64.deb ...
Unpacking iperf (2.0.13+dfsg1-1build1) ...
Setting up iperf (2.0.13+dfsg1-1build1) ...
Processing triggers for man-db (2.9.1-1) ...
```

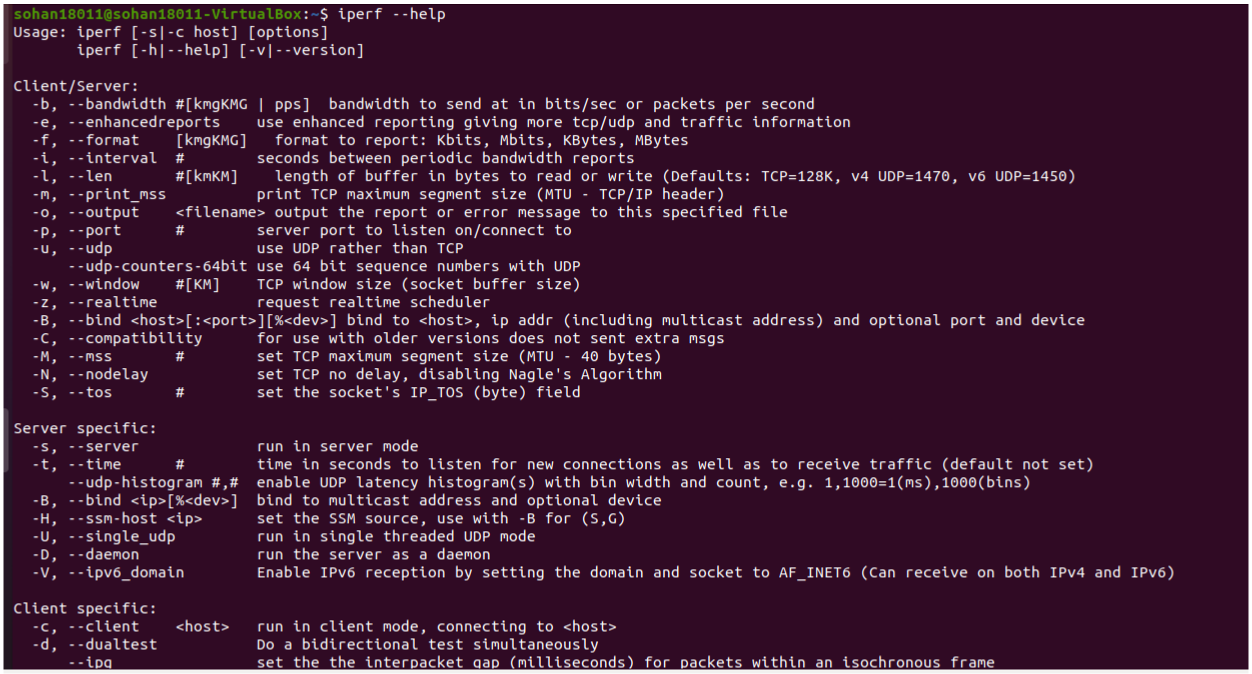
Install Mininet:



```
sohan18011@sohan18011-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 libcgroup1 libpython2-stdlib libpython2.7-minimal libpython2.7-stdlib libunbound8 openvswitch-common openvswitch-switch
  python-pkg-resources python2 python2-minimal python2.7 python2.7-minimal python3-openvswitch python3-sortedcontainers
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 libcgroup1 libpython2-stdlib libpython2.7-minimal libpython2.7-stdlib libunbound8 mininet openvswitch-common
  openvswitch-switch python-pkg-resources python2 python2-minimal python2.7 python2.7-minimal python3-openvswitch python3-sortedcontainers
0 upgraded, 16 newly installed, 0 to remove and 343 not upgraded.
Need to get 7,318 kB of archives.
After this operation, 32.5 MB of additional disk space will be used.
```

Exercises:

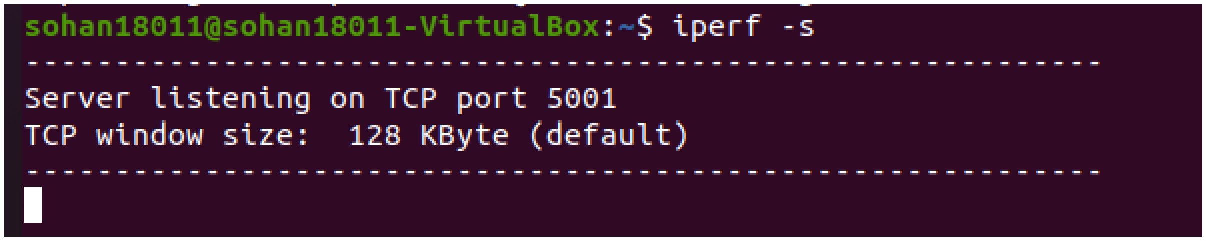
4.1.1: Open a Linux terminal, and execute the command line iperf --help. Provide four configuration options of iperf.



```
sohan18011@sohan18011-VirtualBox:~$ iperf --help
Usage: iperf [-s|-c host] [options]
       iperf [-h|--help] [-v|--version]

Client/Server:
  -b, --bandwidth #[kmgKMG | pps]  bandwidth to send at in bits/sec or packets per second
  -e, --enhancedreports    use enhanced reporting giving more tcp/udp and traffic information
  -f, --format    [kmgKMG]   format to report: Kbits, Mbits, KBytes, MBytes
  -i, --interval  #        seconds between periodic bandwidth reports
  -l, --len       #[kmKM]    length of buffer in bytes to read or write (Defaults: TCP=128K, v4 UDP=1470, v6 UDP=1450)
  -m, --print_mss          print TCP maximum segment size (MTU - TCP/IP header)
  -o, --output    <filename> output the report or error message to this specified file
  -p, --port      #        server port to listen on/connect to
  -u, --udp                use UDP rather than TCP
      --udp-counters-64bit use 64 bit sequence numbers with UDP
  -w, --window    #[KM]    TCP window size (socket buffer size)
  -z, --realtime           request realtime scheduler
  -B, --bind <host>[:<port>][%<dev>] bind to <host>, ip addr (including multicast address) and optional port and device
  -C, --compatibility      for use with older versions does not sent extra msgs
  -M, --mss       #        set TCP maximum segment size (MTU - 40 bytes)
  -N, --nodelay            set TCP no delay, disabling Nagle's Algorithm
  -S, --tos       #        set the socket's IP_TOS (byte) field

Server specific:
  -s, --server             run in server mode
  -t, --time      #        time in seconds to listen for new connections as well as to receive traffic (default not set)
      --udp-histogram #,#  enable UDP latency histogram(s) with bin width and count, e.g. 1,1000=1(ms),1000(bins)
  -B, --bind <ip>[%<dev>]  bind to multicast address and optional device
  -H, --ssm-host <ip>      set the SSM source, use with -B for (S,G)
  -U, --single_udp         run in single threaded UDP mode
  -D, --daemon             run the server as a daemon
  -V, --ipv6_domain        Enable IPv6 reception by setting the domain and socket to AF_INET6 (Can receive on both IPv4 and IPv6)

Client specific:
  -c, --client    <host>   run in client mode, connecting to <host>
  -d, --dualtest           Do a bidirectional test simultaneously
      --ipg                set the the interpacket gap (milliseconds) for packets within an isochronous frame
```

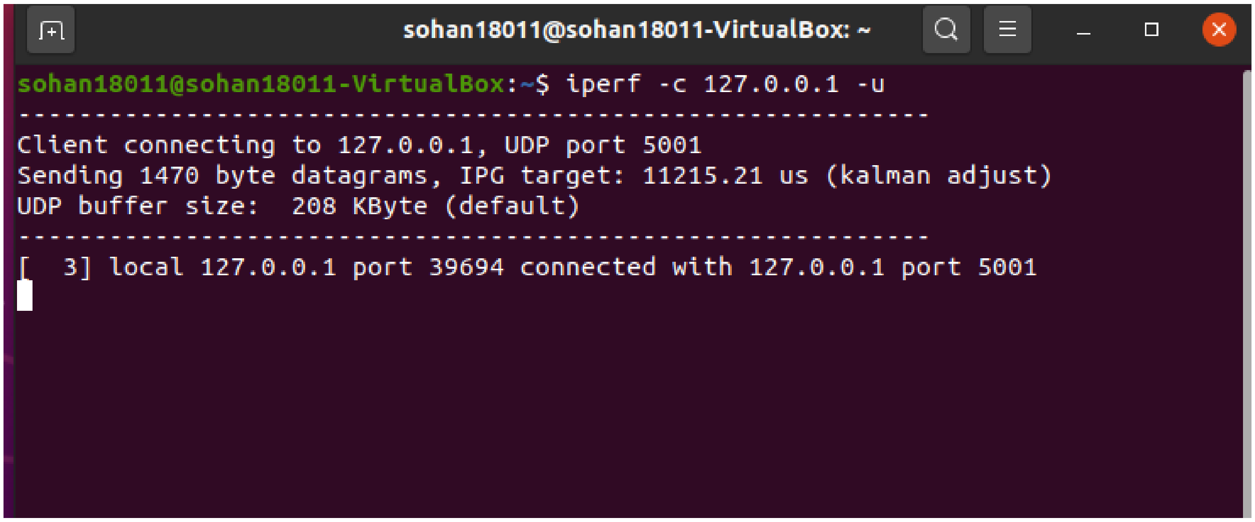
Exercise 4.1.2: Open two Linux terminals, and configure terminal-1 as client (iperf –c IPv4_server_address) and terminal-2 as server (iperf -s).

Terminal -1 :



```
sohan18011@sohan18011-VirtualBox:~$ iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size:  128 KByte (default)
------------------------------------------------------------
```

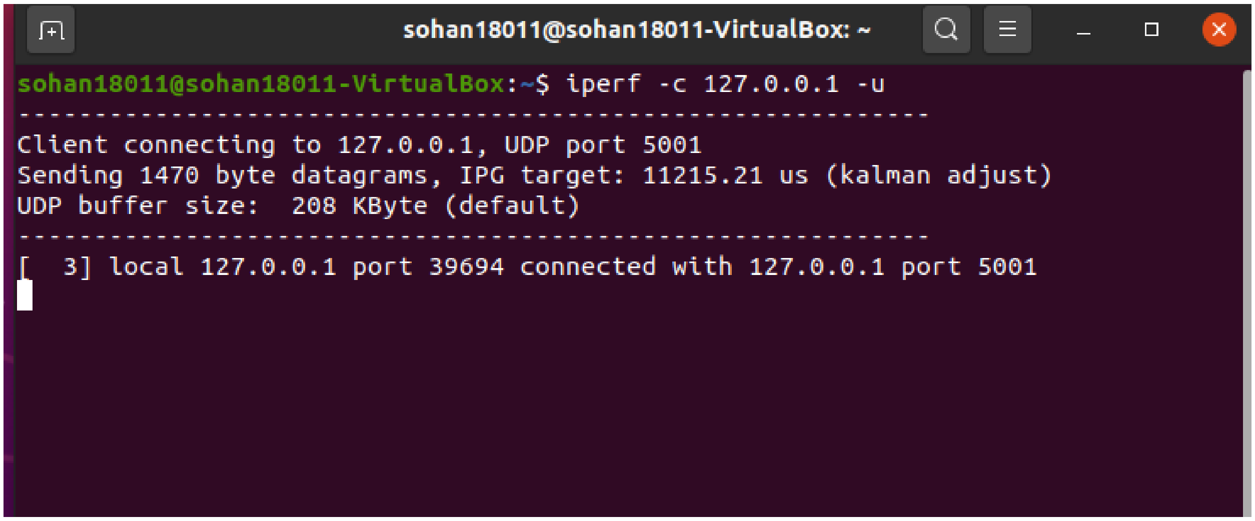
Terminal -2:



```
sohan18011@sohan18011-VirtualBox:~$ iperf -c 127.0.0.1 -u
------------------------------------------------------------
Client connecting to 127.0.0.1, UDP port 5001
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 127.0.0.1 port 39694 connected with 127.0.0.1 port 5001
```

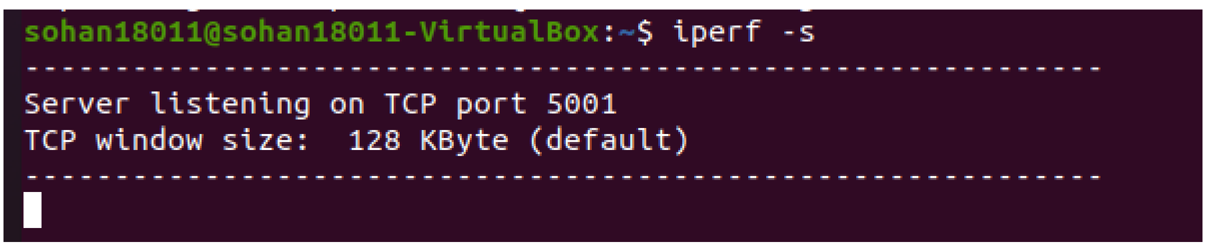
Exercise 4.1.3: Open two Linux terminals, and configure terminal-1 as client and terminal-2 as server for exchanging UDP traffic.

Terminal -1 as client :



```
sohan18011@sohan18011-VirtualBox:~$ iperf -c 127.0.0.1 -u
------------------------------------------------------------
Client connecting to 127.0.0.1, UDP port 5001
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 127.0.0.1 port 39694 connected with 127.0.0.1 port 5001
```

Terminal-2 as server :



```
sohan18011@sohan18011-VirtualBox:~$ iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size:  128 KByte (default)
------------------------------------------------------------
```

Exercise 4.1.4: Open two Linux terminals, and configure terminal-1 as client and terminal-2 as server for exchanging UDP traffic, with:

# Packet length = 1000bytes

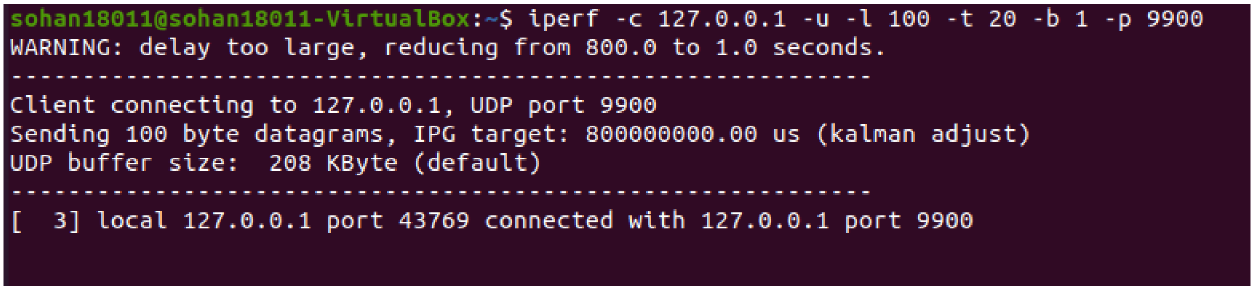
# Time = 20 seconds

# Bandwidth = 1Mbps #

# Port = 9900

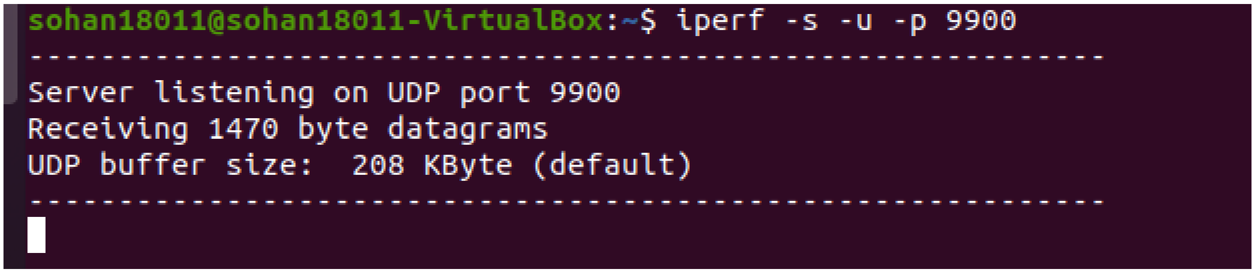
The Command lines are :

Terminal-1 :



```
sohan18011@sohan18011-VirtualBox:~$ iperf -c 127.0.0.1 -u -l 100 -t 20 -b 1 -p 9900
WARNING: delay too large, reducing from 800.0 to 1.0 seconds.
------------------------------------------------------------
Client connecting to 127.0.0.1, UDP port 9900
Sending 100 byte datagrams, IPG target: 800000000.00 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 127.0.0.1 port 43769 connected with 127.0.0.1 port 9900
```

Terminal-2 :

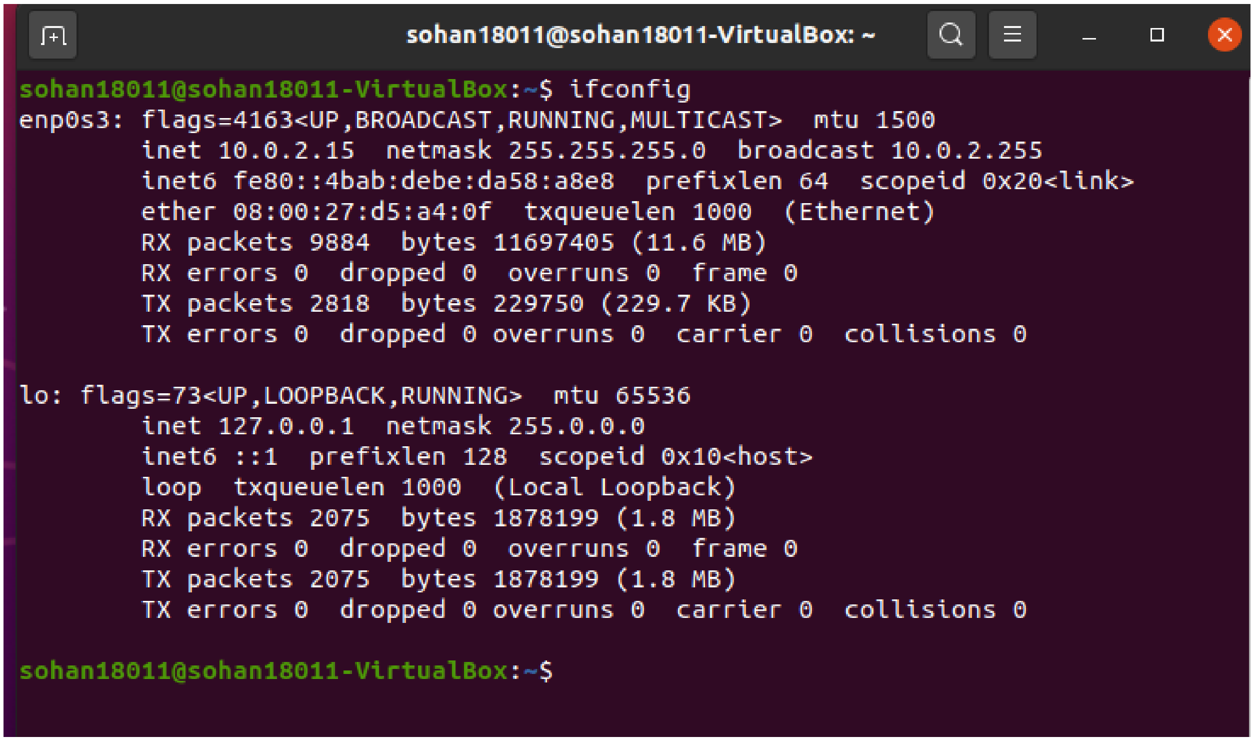


```
sohan18011@sohan18011-VirtualBox:~$ iperf -s -u -p 9900
------------------------------------------------------------
Server listening on UDP port 9900
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
```

Using Mininet:

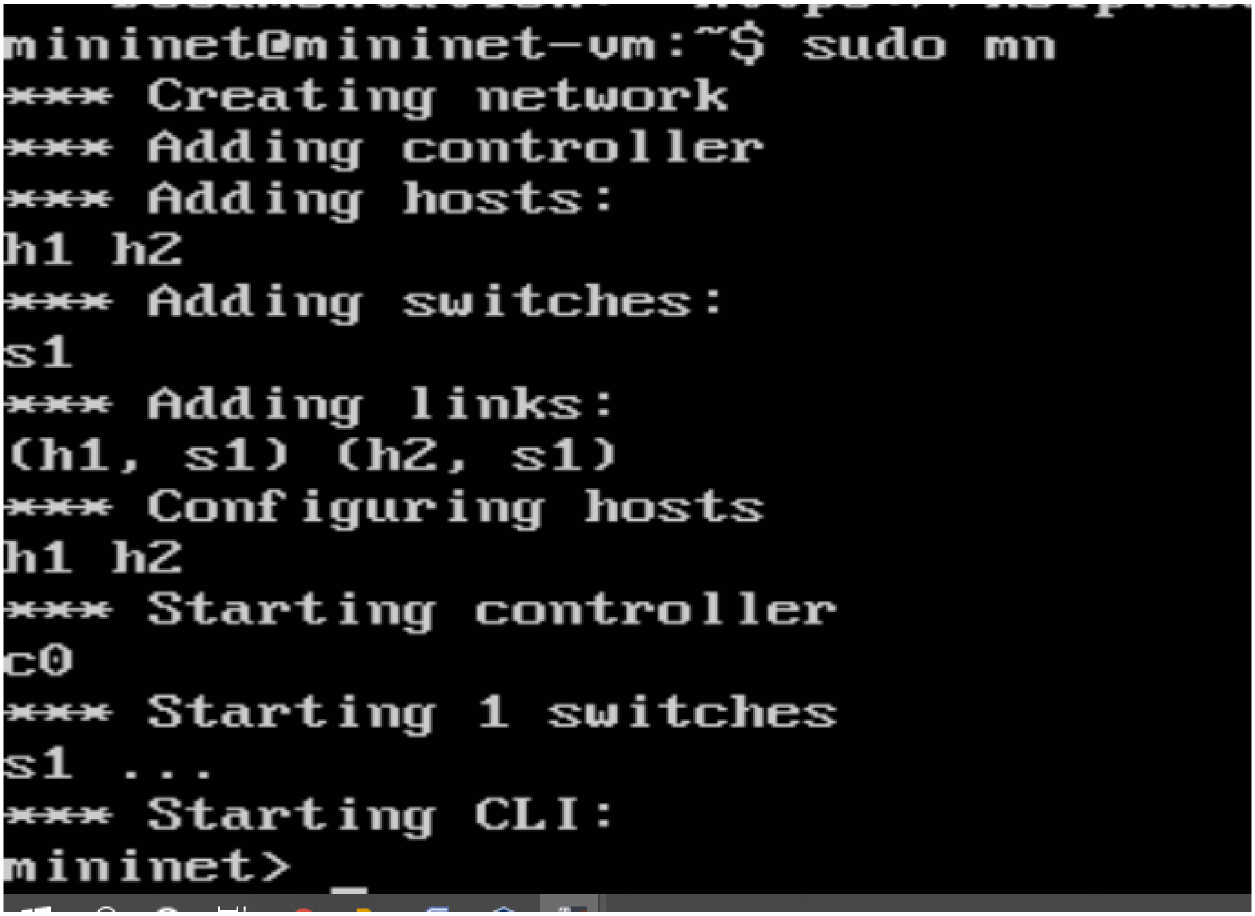
Exercise 4.2.1: Open two Linux terminals, and execute the command line ifconfig in terminal-1.

Interfaces are :



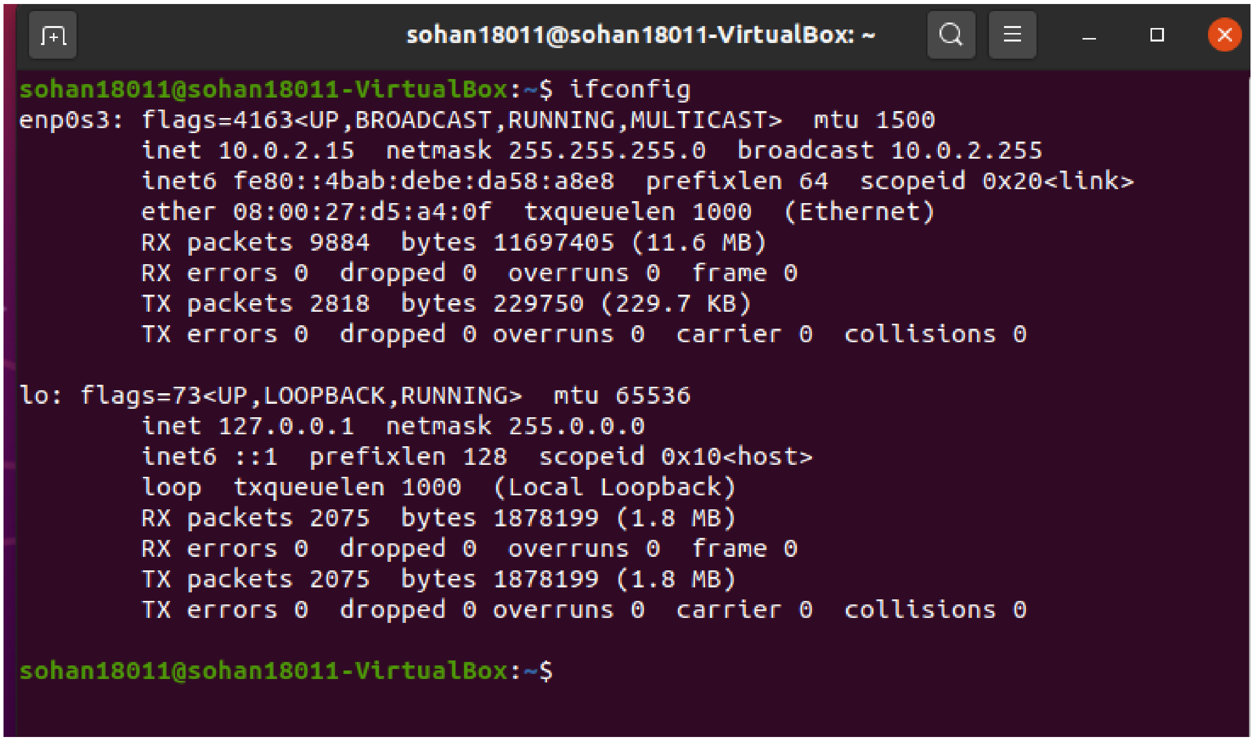
```
sohan18011@sohan18011-VirtualBox:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::4bab:debe:da58:a8e8  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:d5:a4:0f  txqueuelen 1000  (Ethernet)
        RX packets 9884  bytes 11697405 (11.6 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2818  bytes 229750 (229.7 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 2075  bytes 1878199 (1.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2075  bytes 1878199 (1.8 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

sohan18011@sohan18011-VirtualBox:~$
```

In terminal-2, execute the command line sudo mn:



```
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> _
```

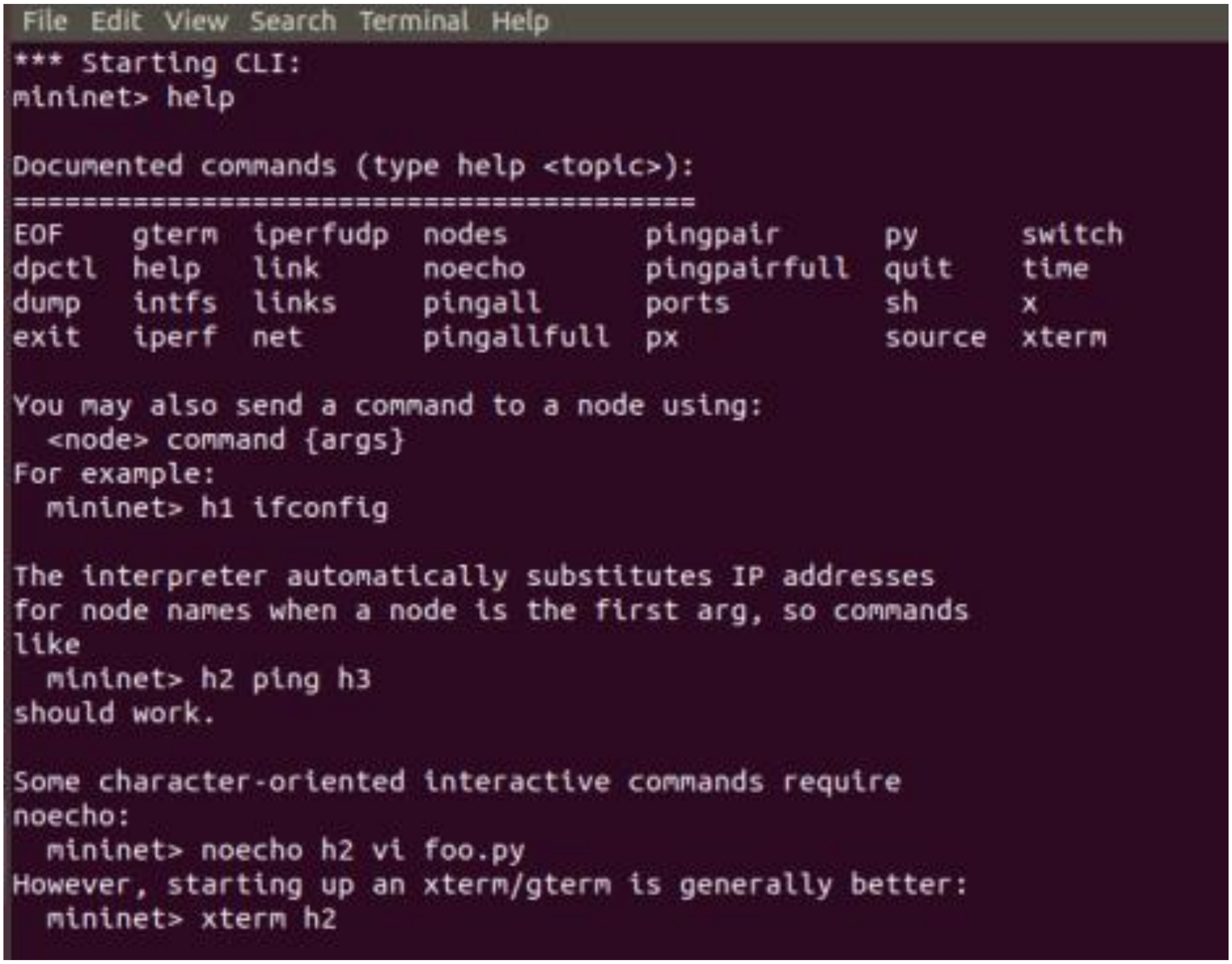
In terminal-1 execute the command line ifconfig , then real and virtual interfaces are :



```
sohan18011@sohan18011-VirtualBox:~$ ifconfig
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::4bab:debe:da58:a8e8  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:d5:a4:0f  txqueuelen 1000  (Ethernet)
        RX packets 9884  bytes 11697405 (11.6 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2818  bytes 229750 (229.7 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 2075  bytes 1878199 (1.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2075  bytes 1878199 (1.8 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

sohan18011@sohan18011-VirtualBox:~$
```

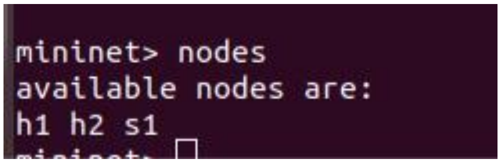
Exercise 4.2.2: Interacting with mininet; in terminal-2, display the following command lines and explain what it does:

#   mininet> help



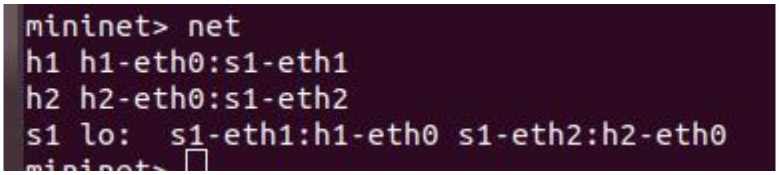
```
*** Starting CLI:
mininet> help

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
```

# mininet> nodes



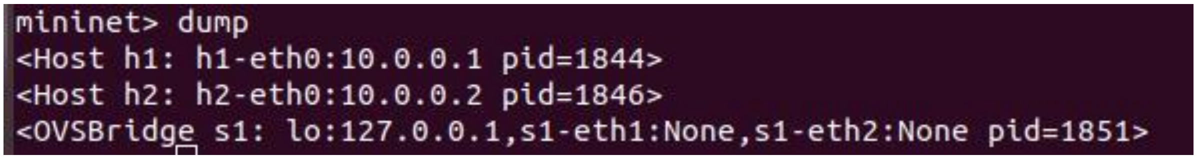
```
mininet> nodes
available nodes are:
h1 h2 s1
mininet> 
```

# mininet> net



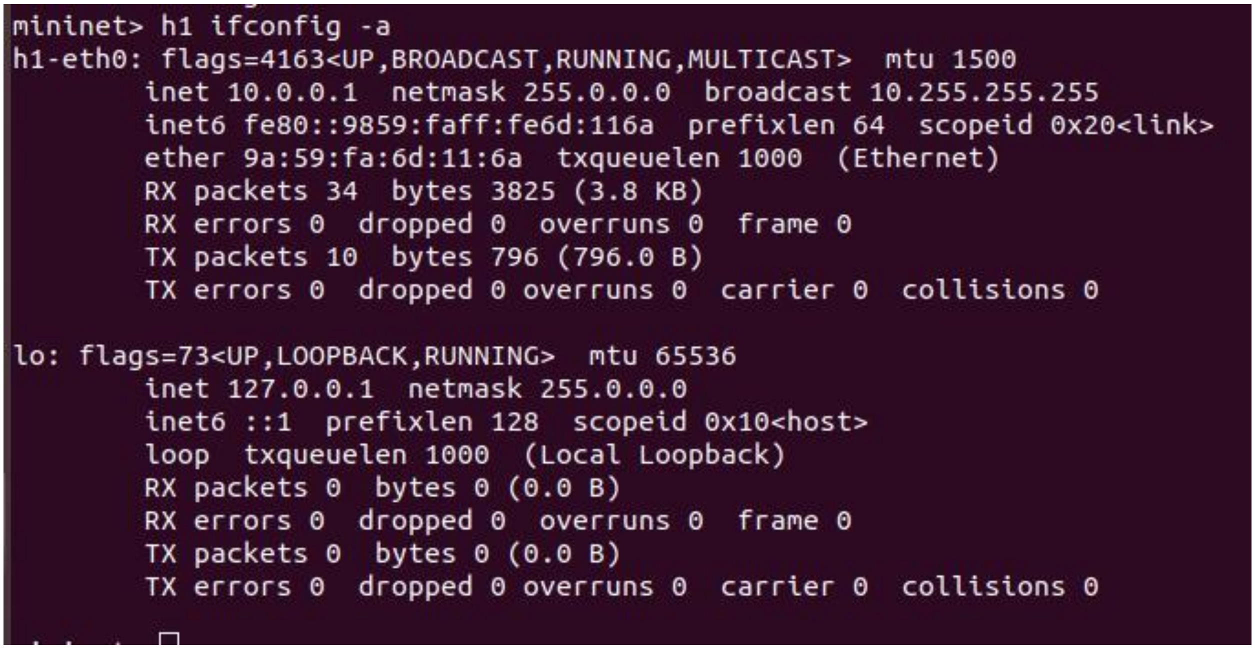
```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
mininet> 
```

# mininet> dump



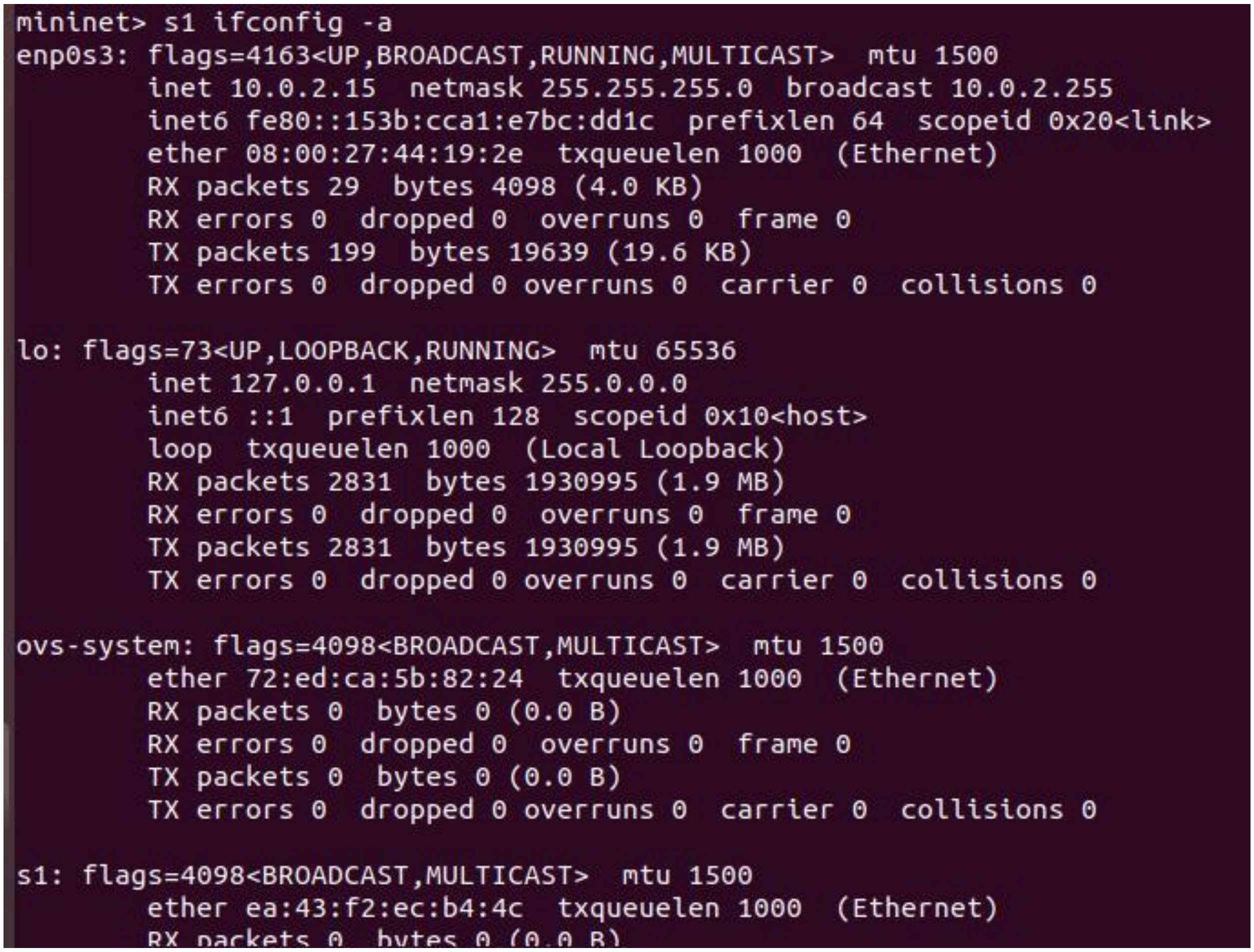
```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=1844>
<Host h2: h2-eth0:10.0.0.2 pid=1846>
<OVSBridge s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=1851>
```

# mininet> h1 ifconfig –a



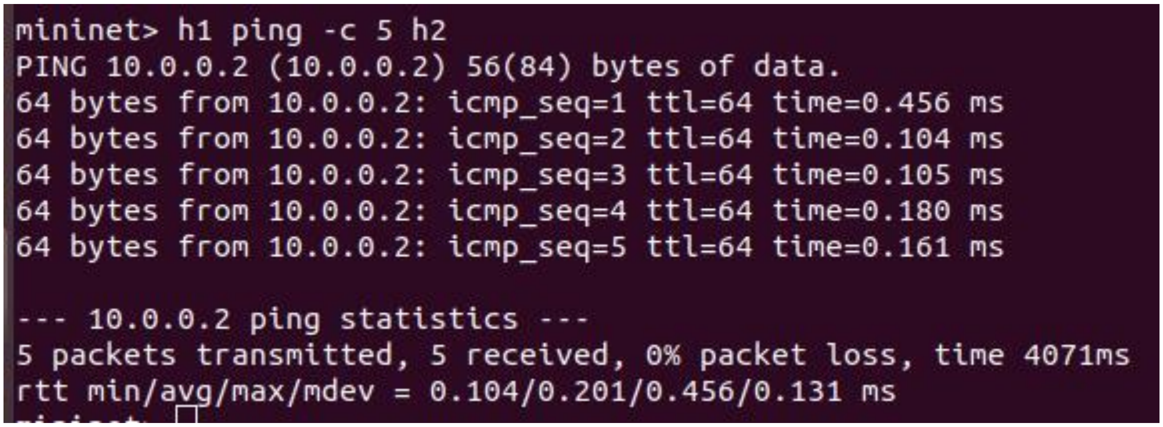
```
mininet> h1 ifconfig -a
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::9859:faff:fe6d:116a  prefixlen 64  scopeid 0x20<link>
        ether 9a:59:fa:6d:11:6a  txqueuelen 1000  (Ethernet)
        RX packets 34  bytes 3825 (3.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 796 (796.0 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 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> s1 ifconfig –a



```
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::153b:cca1:e7bc:dd1c  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:44:19:2e  txqueuelen 1000  (Ethernet)
        RX packets 29  bytes 4098 (4.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 199  bytes 19639 (19.6 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 2831  bytes 1930995 (1.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2831  bytes 1930995 (1.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ovs-system: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether 72:ed:ca:5b:82:24  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 ea:43:f2:ec:b4:4c  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
```

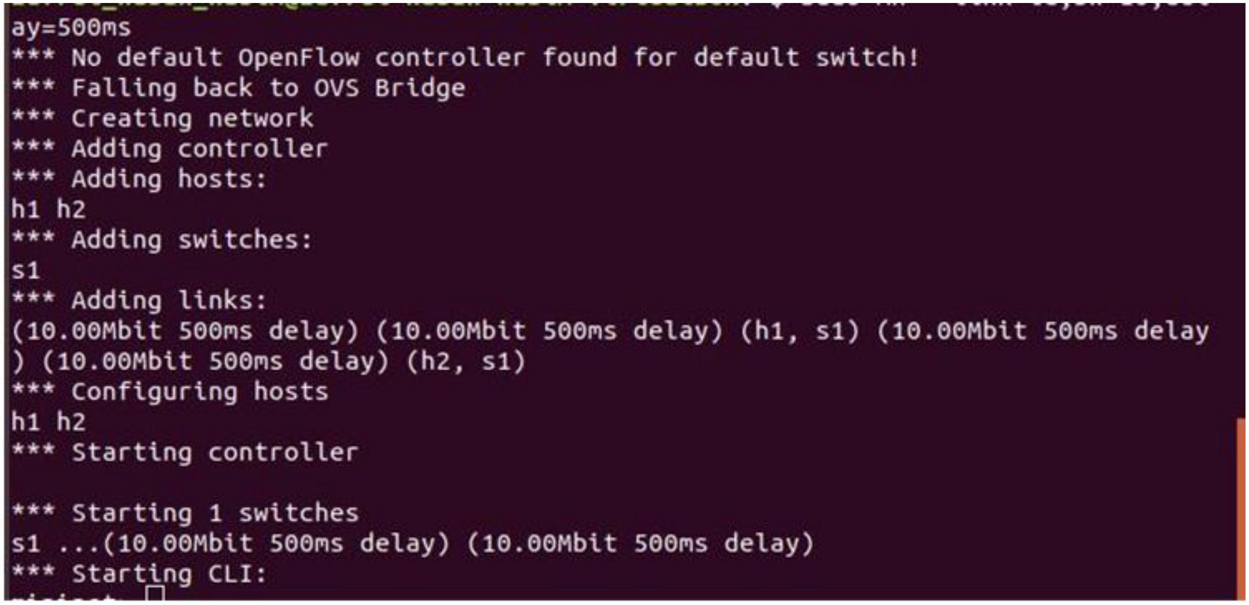
# mininet> h1 ping -c 5 h2



```
mininet> h1 ping -c 5 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=0.456 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.104 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.105 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.180 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.161 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4071ms
rtt min/avg/max/mdev = 0.104/0.201/0.456/0.131 ms
mininet>
```

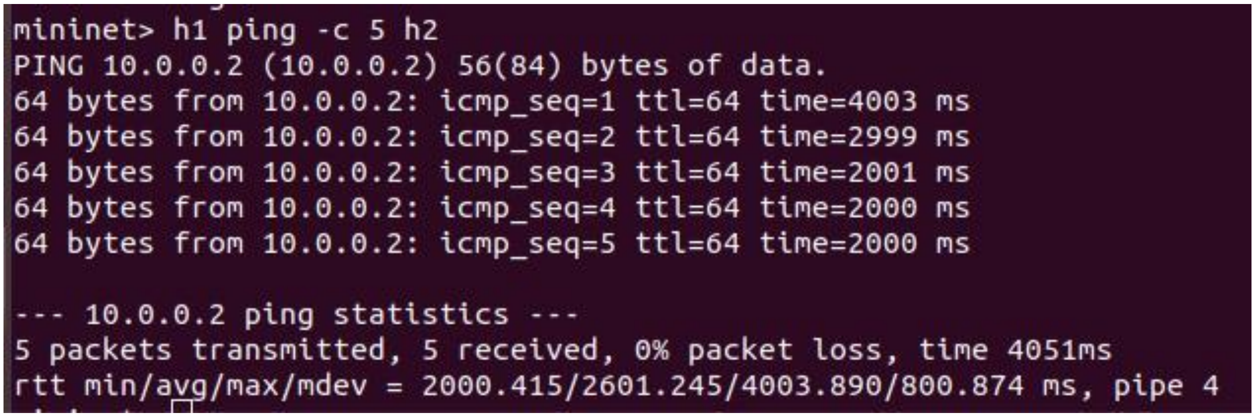
Exercise 4.2.3: In terminal - 2, display the following command line:

# sudo mn --link tc,bw=10,delay=500ms.



```
ay=500ms
*** 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:
(10.00Mbit 500ms delay) (10.00Mbit 500ms delay) (h1, s1) (10.00Mbit 500ms delay
) (10.00Mbit 500ms delay) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

*** Starting 1 switches
s1 ...(10.00Mbit 500ms delay) (10.00Mbit 500ms delay)
*** Starting CLI:
```

# mininet> h1 ping -c 5 h2



```
mininet> h1 ping -c 5 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=4003 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=2999 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=2001 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=2000 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=2000 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4051ms
rtt min/avg/max/mdev = 2000.415/2601.245/4003.890/800.874 ms, pipe 4
```

Conclusion :

This lab give me information about, install and configure SDN Controller , Install and understand how the mininet simulator works and also install and use traffic generators as powerful tools for testing network performance. I have understood that how to use mininet as teaching, development and research . . I also learnt how to Implement and to run basic examples for understanding the role of the controller and how it interact with mininet.