MAWLANA BHASHANI SCIENCE AND TECHNOLOGY UNIVERSITY,

Santosh, Tangail -1902

Lab Report No	**:**	07
Lab Report Name	**:**	SDN Controllers and Mininet
Course Name	**:**	Computer Networks Lab

Submitted by,

Name : Sabrin Afroz

ID : IT-17007

Session : 2016-17

Dept. of ICT, MBSTU.

Submitted to,

Nazrul Islam

Assistant Professor

Dept. of ICT, MBSTU.

Objective :

The objective of the lab is 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 :

Iperf : Iperf is a widely used tool for network performance measurement and tuning. It is significant as a cross-platform tool that can produce standardized performance measurements for any network. Iperf has client and server functionality, and can create data streams to measure the throughput between the two ends in one or both directions. Typical iperf output contains a time-stamped report of the amount of data transferred and the throughput measured.

Software-defined networking (SDN) is an approach to networking that uses software-based controllers or application programming interfaces (APIs) to direct traffic on the network and communicate with the underlying hardware infrastructure.

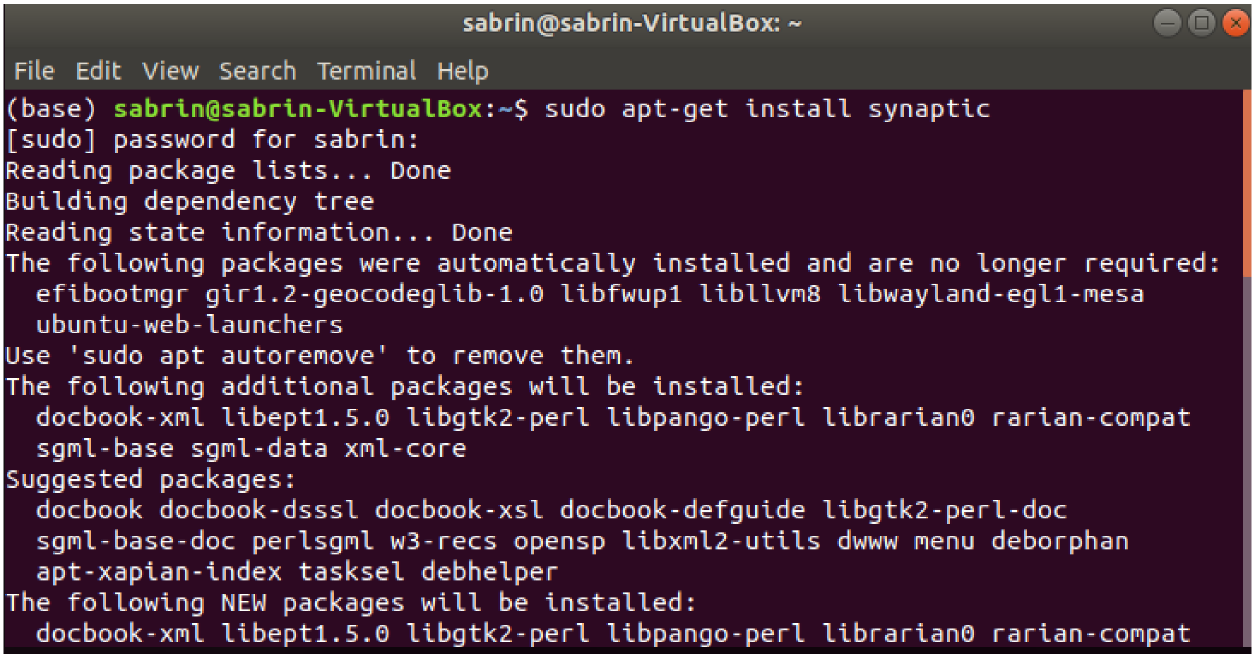
This is different from traditional networks, which use dedicated hardware devices (routers and switches) to control network traffic. SDN can create and control a virtual network or control a traditional hardware network with software.

While network virtualization enables the ability to segment different virtual networks within one physical network or connect devices on different physical networks into one virtual network, software-defined networking enables a new way of controlling the routing of data packets through a centralized server.

Mininet: Mininet creates a realistic virtual network, running real kernel, switch and application code, on a single machine (VM, cloud or native).

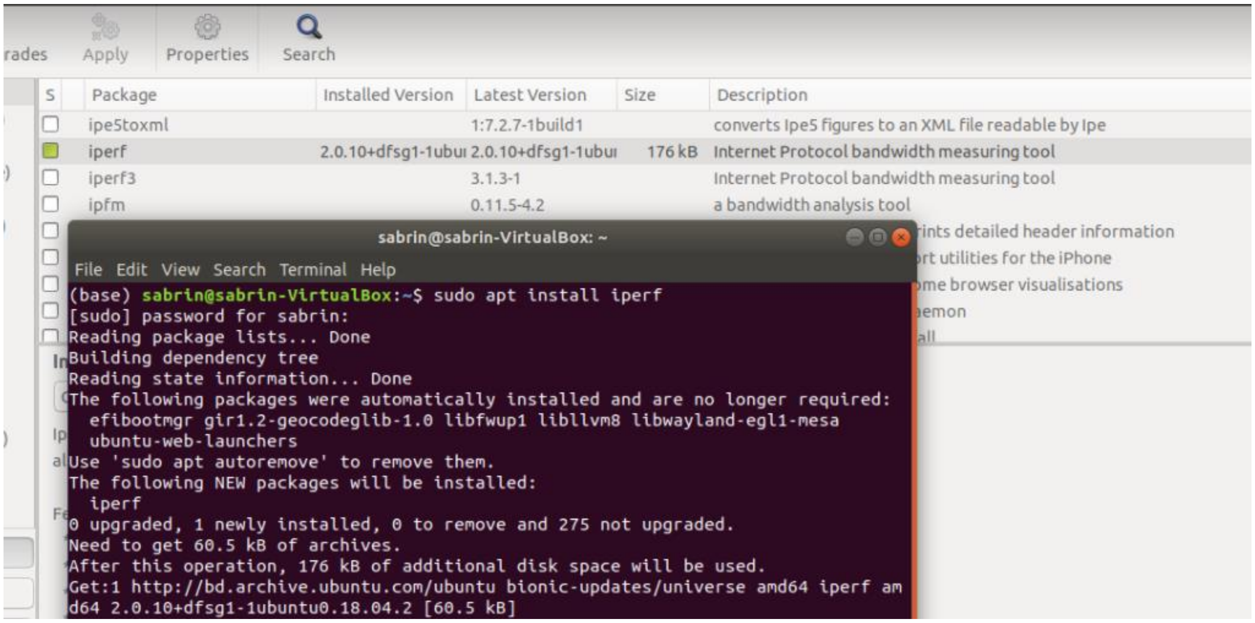
Methodology:

Install Synaptic:


sabrin@sabrin-VirtualBox: ~
File Edit View Search Terminal Help

```
(base) sabrin@sabrin-VirtualBox:~$ sudo apt-get install synaptic
[sudo] password for sabrin:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  efibootmgr gir1.2-geocodeglib-1.0 libfwup1 libllvm8 libwayland-egl1-mesa
  ubuntu-web-launchers
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  docbook-xml libept1.5.0 libgtk2-perl libpango-perl librarian0 rarian-compat
  sgml-base sgml-data xml-core
Suggested packages:
  docbook docbook-dsssl docbook-xsl docbook-defguide libgtk2-perl-doc
  sgml-base-doc perlsgml w3-recs opensp libxml2-utils dwww menu deborphan
  apt-xapian-index tasksel debhelper
The following NEW packages will be installed:
  docbook-xml libept1.5.0 libgtk2-perl libpango-perl librarian0 rarian-compat
```

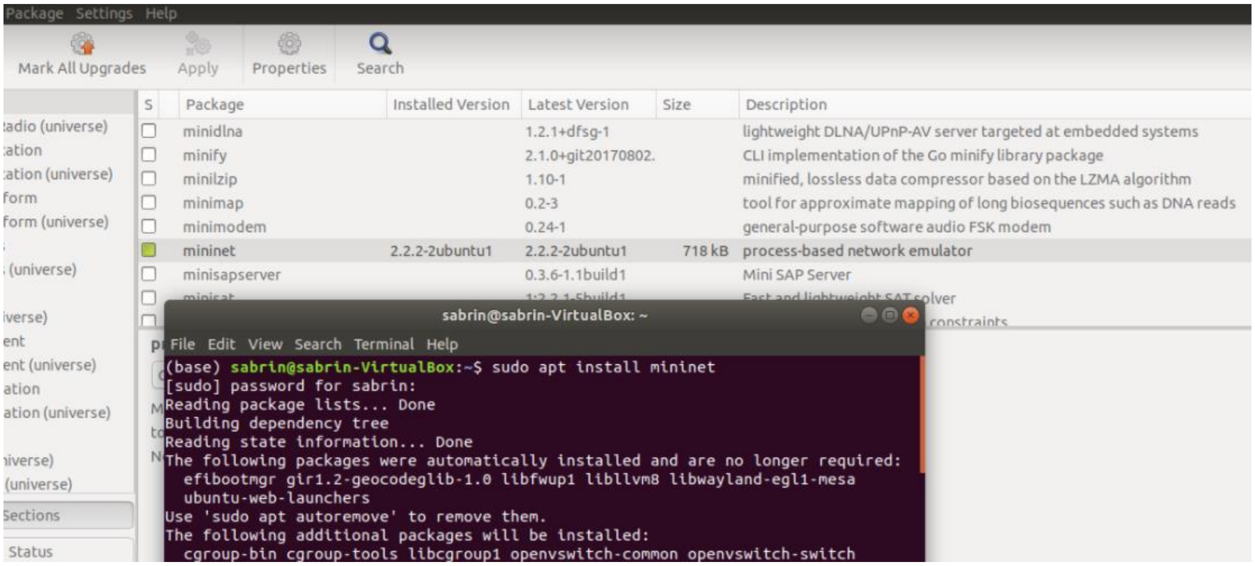
Install iperf :


rades
Apply
Properties
Search
sabrin@sabrin-VirtualBox: ~
File Edit View Search Terminal Help

S	Package	Installed Version	Latest Version	Size	Description
□	ipe5toxml		1:7.2.7-1build1		converts Ipe5 figures to an XML file readable by Ipe
🟩	iperf	2.0.10+dfsg1-1ubu	2.0.10+dfsg1-1ubu	176 kB	Internet Protocol bandwidth measuring tool
□	iperf3		3.1.3-1		Internet Protocol bandwidth measuring tool
□	ipfm		0.11.5-4.2		a bandwidth analysis tool
□					ints detailed header information
□					rt utilities for the iPhone
□					me browser visualisations
□					emon
□					all

```
(base) sabrin@sabrin-VirtualBox:~$ sudo apt install iperf
[sudo] password for sabrin:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  efibootmgr gir1.2-geocodeglib-1.0 libfwup1 libllvm8 libwayland-egl1-mesa
  ubuntu-web-launchers
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  iperf
0 upgraded, 1 newly installed, 0 to remove and 275 not upgraded.
Need to get 60.5 kB of archives.
After this operation, 176 kB of additional disk space will be used.
Get:1 http://bd.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 iperf am
d64 2.0.10+dfsg1-1ubuntu0.18.04.2 [60.5 kB]
```

Install mininet :



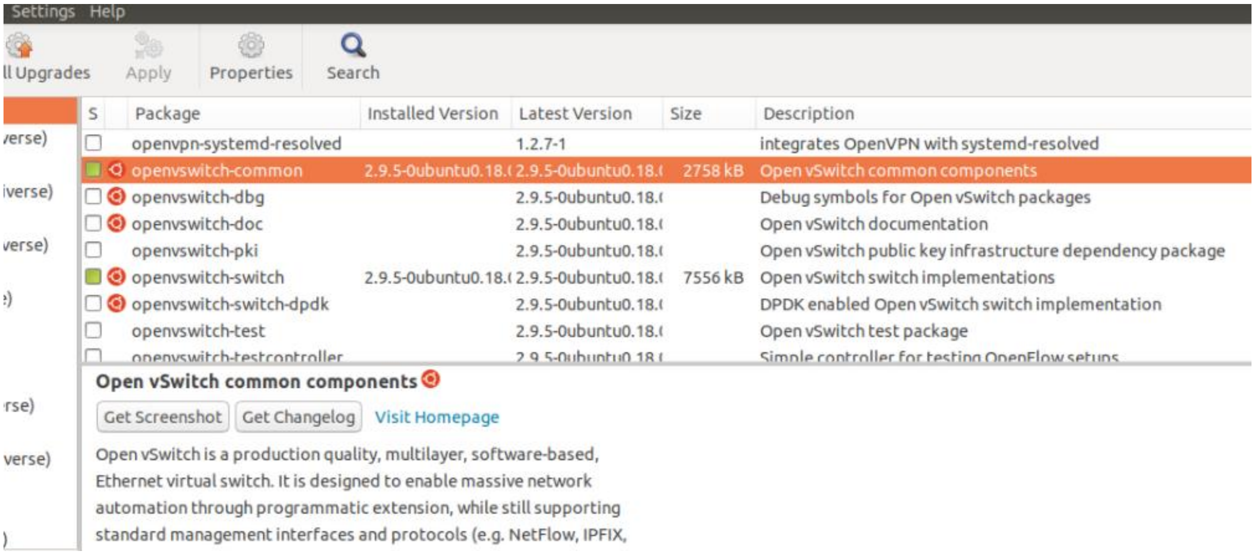
S	Package	Installed Version	Latest Version	Size	Description
□	minidlna		1.2.1+dfsg-1		lightweight DLNA/UPnP-AV server targeted at embedded systems
□	minify		2.1.0+git20170802.		CLI implementation of the Go minify library package
□	minilzip		1.10-1		minified, lossless data compressor based on the LZMA algorithm
□	minimap		0.2-3		tool for approximate mapping of long biosequences such as DNA reads
□	minimodem		0.24-1		general-purpose software audio FSK modem
🟩	mininet	2.2.2-2ubuntu1	2.2.2-2ubuntu1	718 kB	process-based network emulator
□	minisapserver		0.3.6-1.1build1		Mini SAP Server
□	minisat		1:2.2.1-5build1		Fast and lightweight SAT solver
□					constraints

sabrin@sabrin-VirtualBox: ~

File Edit View Search Terminal Help

```
(base) sabrin@sabrin-VirtualBox:~$ sudo apt install mininet
[sudo] password for sabrin:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  efibootmgr gir1.2-geocodeglib-1.0 libfwup1 libllvm8 libwayland-egl1-mesa
  ubuntu-web-launchers
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  cgroup-bin cgroup-tools libcgroup1 openvswitch-common openvswitch-switch
```

Install openvswitch-controller:



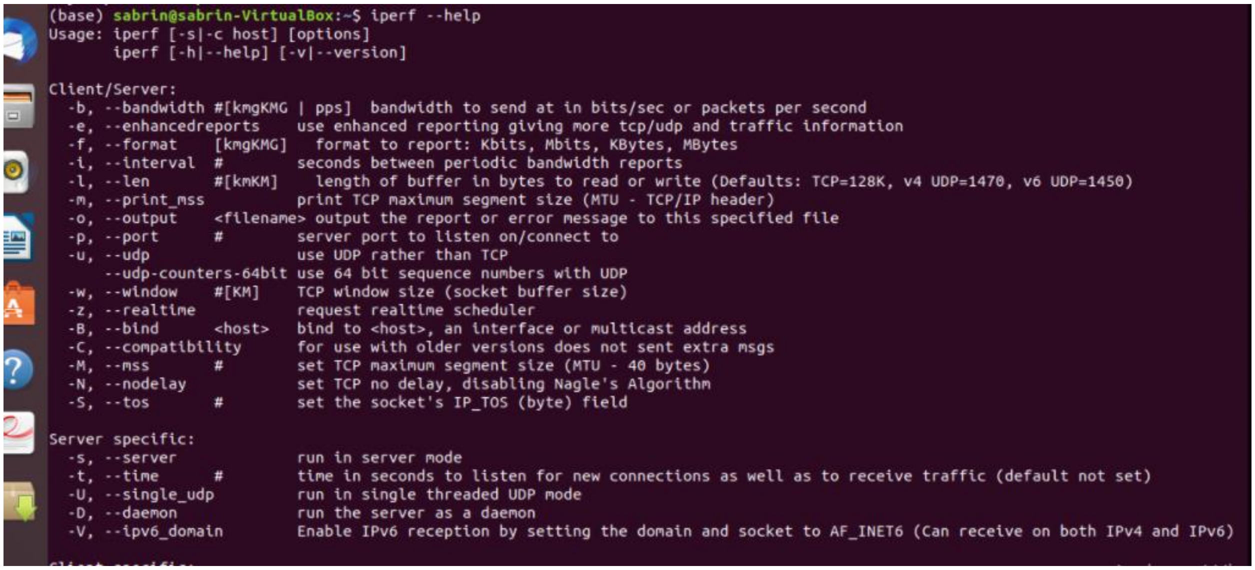
S	Package	Installed Version	Latest Version	Size	Description
□	openvpn-systemd-resolved		1.2.7-1		integrates OpenVPN with systemd-resolved
🟩	openvswitch-common	2.9.5-0ubuntu0.18.(	2.9.5-0ubuntu0.18.(	2758 kB	Open vSwitch common components
□	openvswitch-dbg		2.9.5-0ubuntu0.18.(		Debug symbols for Open vSwitch packages
□	openvswitch-doc		2.9.5-0ubuntu0.18.(		Open vSwitch documentation
□	openvswitch-pki		2.9.5-0ubuntu0.18.(		Open vSwitch public key infrastructure dependency package
🟩	openvswitch-switch	2.9.5-0ubuntu0.18.(	2.9.5-0ubuntu0.18.(	7556 kB	Open vSwitch switch implementations
□	openvswitch-switch-dpdk		2.9.5-0ubuntu0.18.(		DPDK enabled Open vSwitch switch implementation
□	openvswitch-test		2.9.5-0ubuntu0.18.(		Open vSwitch test package
□	openvswitch-testcontroller		2.9.5-0ubuntu0.18.(		Simple controller for testing OpenFlow setups

Open vSwitch common components

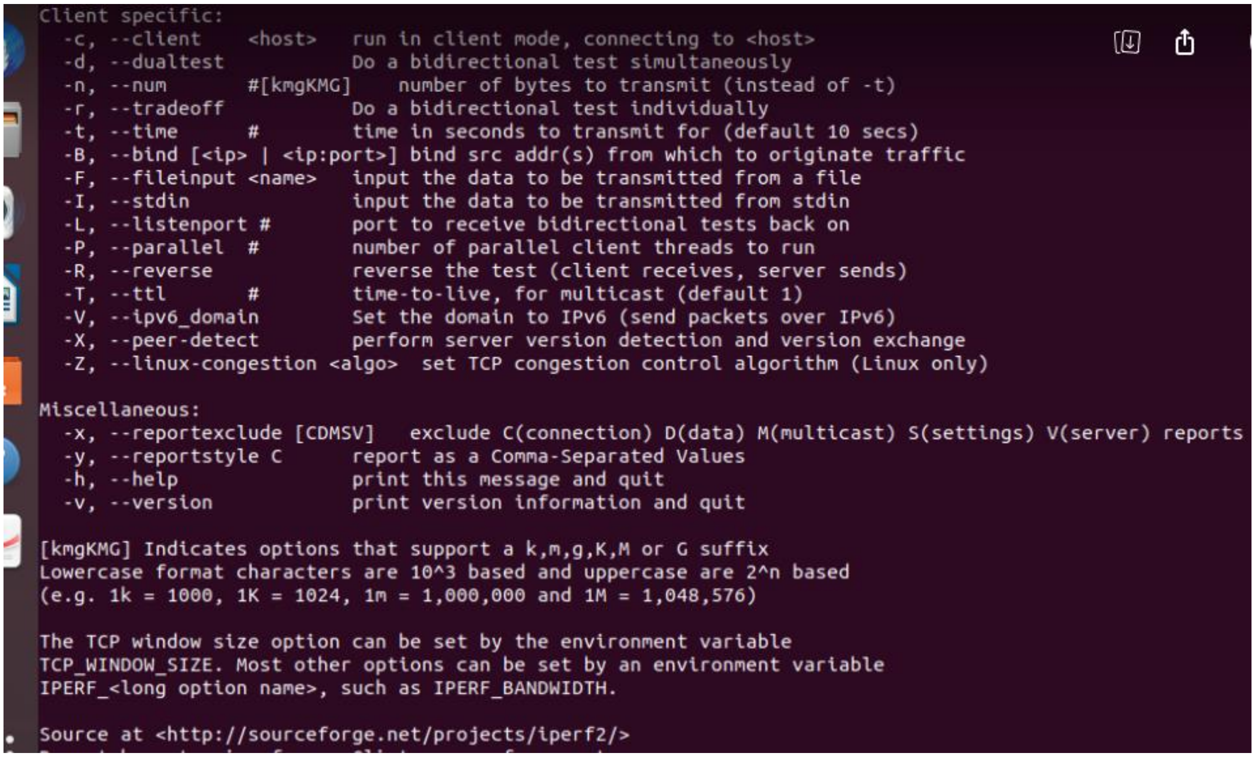
Get Screenshot   Get Changelog   Visit Homepage

Open vSwitch is a production quality, multilayer, software-based, Ethernet virtual switch. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols (e.g. NetFlow, IPFIX,

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

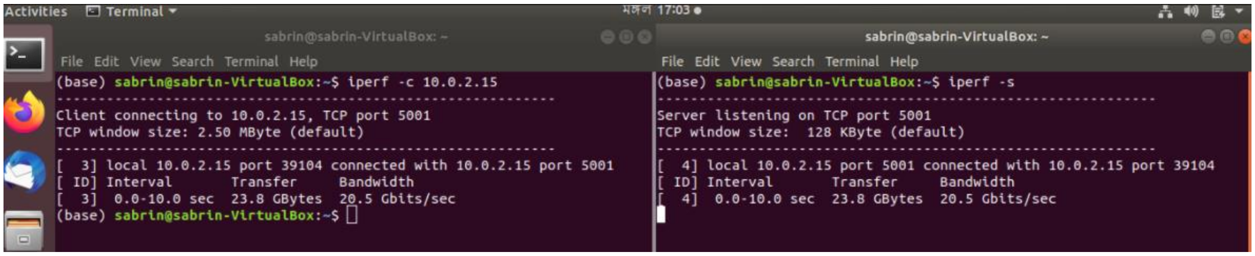


```
(base) sabrin@sabrin-VirtualBox:~$ iperf --help
Usage: iperf [-s|-c host] [options]
       iperf [-h|--help] [-v|--version]

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>   bind to <host>, an interface or multicast address
  -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)
  -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
  -n, --num       #[kmgKMG]    number of bytes to transmit (instead of -t)
  -r, --tradeoff           Do a bidirectional test individually
  -t, --time      #        time in seconds to transmit for (default 10 secs)
  -B, --bind [<ip> | <ip:port>] bind src addr(s) from which to originate traffic
  -F, --fileinput <name>   input the data to be transmitted from a file
  -I, --stdin              input the data to be transmitted from stdin
  -L, --listenport #       port to receive bidirectional tests back on
  -P, --parallel  #        number of parallel client threads to run
  -R, --reverse            reverse the test (client receives, server sends)
  -T, --ttl       #        time-to-live, for multicast (default 1)
  -V, --ipv6_domain        Set the domain to IPv6 (send packets over IPv6)
  -X, --peer-detect        perform server version detection and version exchange
  -Z, --linux-congestion <algo>  set TCP congestion control algorithm (Linux only)

Miscellaneous:
  -x, --reportexclude [CDMSV]   exclude C(connection) D(data) M(multicast) S(settings) V(server) reports
  -y, --reportstyle C      report as a Comma-Separated Values
  -h, --help               print this message and quit
  -v, --version            print version information and quit

[kmgKMG] Indicates options that support a k,m,g,K,M or G suffix
Lowercase format characters are 10^3 based and uppercase are 2^n based
(e.g. 1k = 1000, 1K = 1024, 1m = 1,000,000 and 1M = 1,048,576)

The TCP window size option can be set by the environment variable
TCP_WINDOW_SIZE. Most other options can be set by an environment variable
IPERF_<long option name>, such as IPERF_BANDWIDTH.

Source at <http://sourceforge.net/projects/iperf2/>
```

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).



Activities
Terminal
मंगळ 17:03
sabrin@sabrin-VirtualBox: ~
File Edit View Search Terminal Help

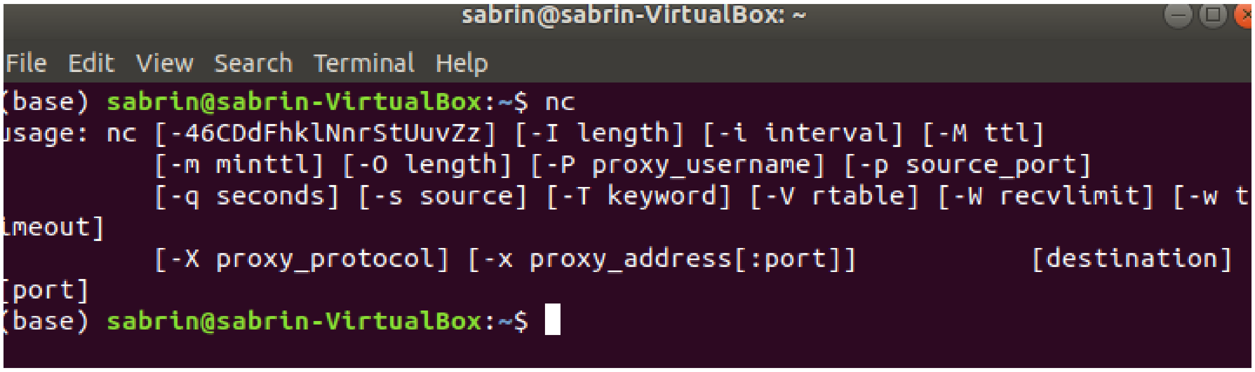
```
(base) sabrin@sabrin-VirtualBox:~$ iperf -c 10.0.2.15
------------------------------------------------------------
Client connecting to 10.0.2.15, TCP port 5001
TCP window size: 2.50 MByte (default)
------------------------------------------------------------
[  3] local 10.0.2.15 port 39104 connected with 10.0.2.15 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  23.8 GBytes  20.5 Gbits/sec
(base) sabrin@sabrin-VirtualBox:~$
```

```
(base) sabrin@sabrin-VirtualBox:~$ iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size:  128 KByte (default)
------------------------------------------------------------
[  4] local 10.0.2.15 port 5001 connected with 10.0.2.15 port 39104
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  23.8 GBytes  20.5 Gbits/sec
```

Exercise 4.1.3: Open two Linux terminals, and configure terminal-1 as client and terminal-2 as server for exchanging UDP traffic, which are the command lines?

Netcat Command:

Netcat(nc) command is installed by default in Linux OS.



sabrin@sabrin-VirtualBox: ~
File Edit View Search Terminal Help

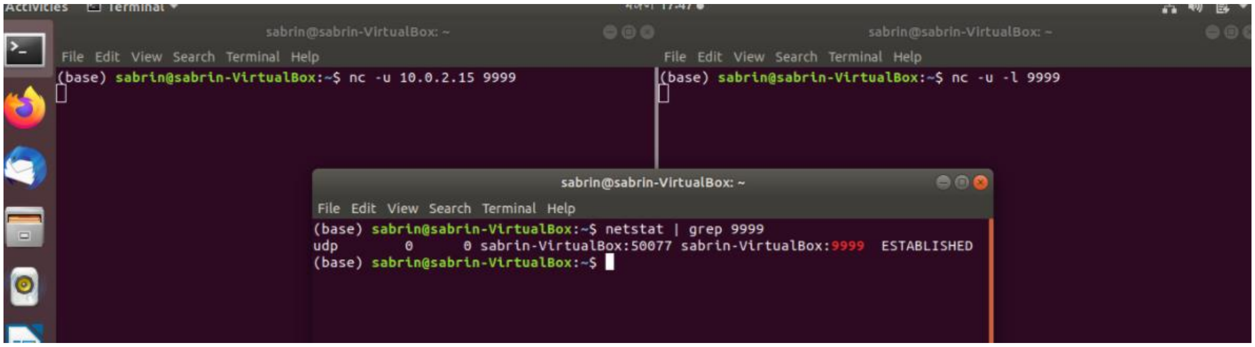
```
(base) sabrin@sabrin-VirtualBox:~$ nc
usage: nc [-46CDdFhklNnrStUuvZz] [-I length] [-i interval] [-M ttl]
          [-m minttl] [-O length] [-P proxy_username] [-p source_port]
          [-q seconds] [-s source] [-T keyword] [-V rtable] [-W recvlimit] [-w t
imeout]
          [-X proxy_protocol] [-x proxy_address[:port]]          [destination]
[port]
(base) sabrin@sabrin-VirtualBox:~$
```

This means nc command is already exist in Linux.

Start Server : $ nc –u –l 9999

Start client : $ nc -u 10.0.2.15 9999

Check connection : $ netstat | grep 9999



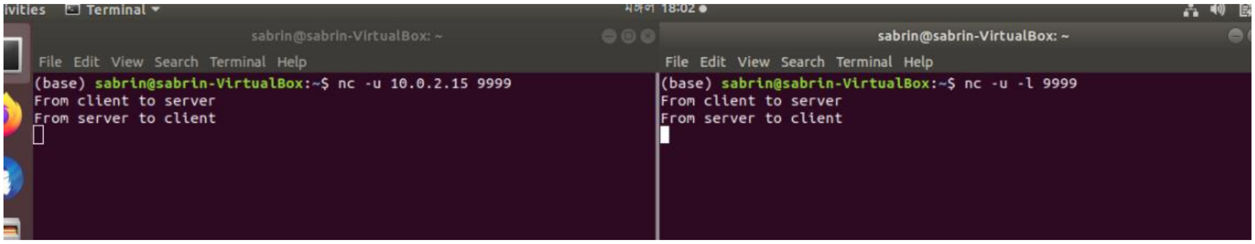
Activities
Terminal
sabrin@sabrin-VirtualBox: ~
File Edit View Search Terminal Help

```
(base) sabrin@sabrin-VirtualBox:~$ nc -u 10.0.2.15 9999
```

```
(base) sabrin@sabrin-VirtualBox:~$ nc -u -l 9999
```

```
(base) sabrin@sabrin-VirtualBox:~$ netstat | grep 9999
udp        0      0 sabrin-VirtualBox:50077 sabrin-VirtualBox:9999  ESTABLISHED
(base) sabrin@sabrin-VirtualBox:~$
```

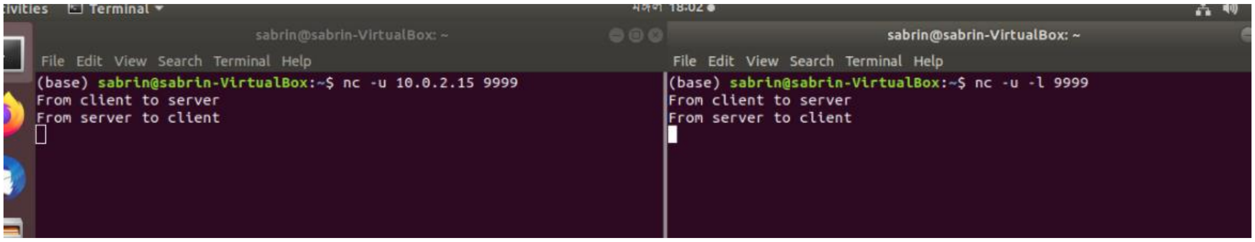
Send UDP packets from client to server:



```
(base) sabrin@sabrin-VirtualBox:~$ nc -u 10.0.2.15 9999
From client to server
From server to client
```

```
(base) sabrin@sabrin-VirtualBox:~$ nc -u -l 9999
From client to server
From server to client
```

Send UDP packets from server to client:

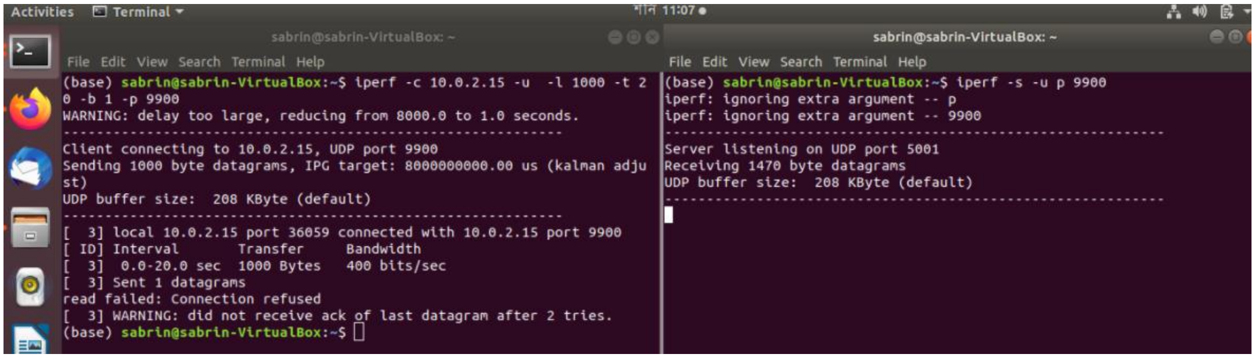


```
(base) sabrin@sabrin-VirtualBox:~$ nc -u 10.0.2.15 9999
From client to server
From server to client
```

```
(base) sabrin@sabrin-VirtualBox:~$ nc -u -l 9999
From client to server
From server to client
```

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

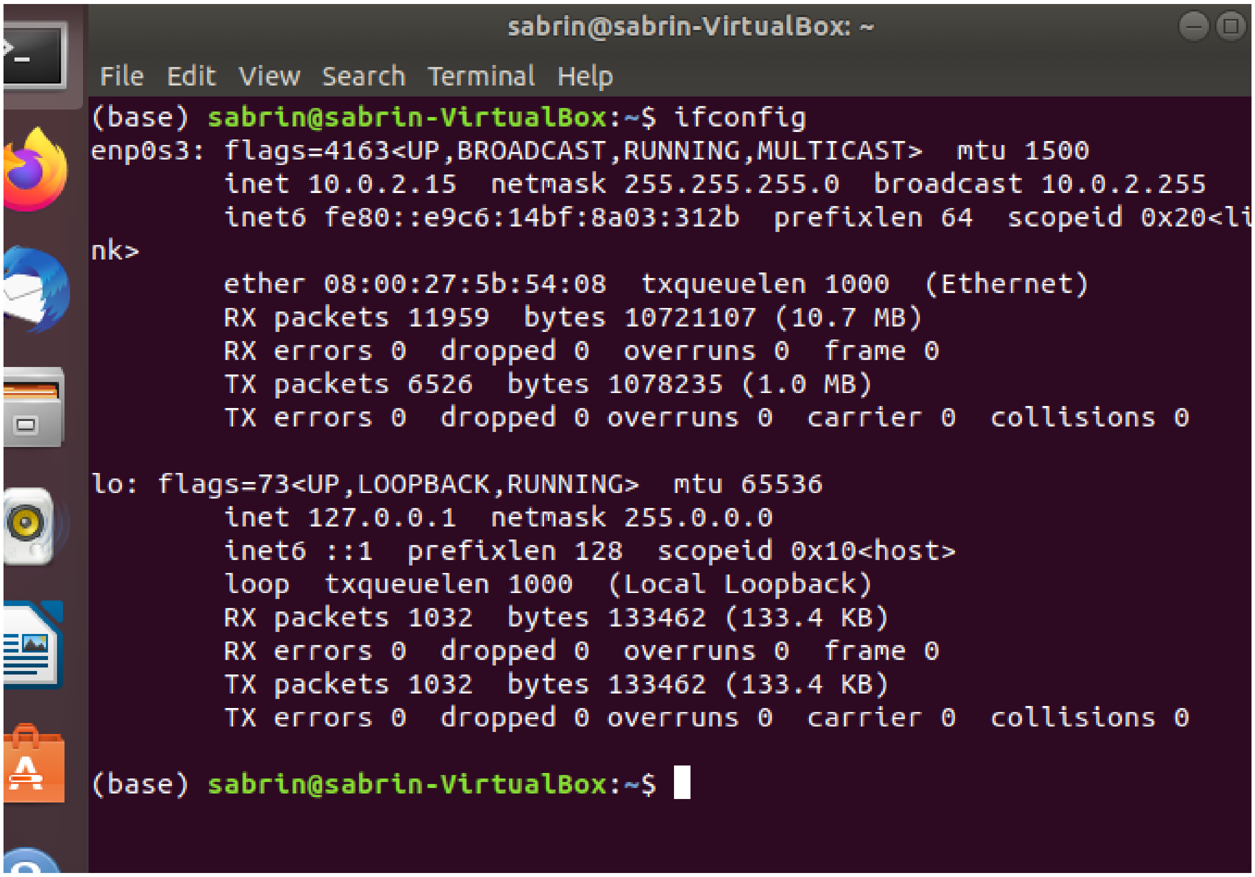
- Packet length = 1000bytes
- Time = 20 seconds
- Bandwidth = 1Mbps
- Port = 9900



```
(base) sabrin@sabrin-VirtualBox:~$ iperf -c 10.0.2.15 -u  -l 1000 -t 2
0 -b 1 -p 9900
WARNING: delay too large, reducing from 8000.0 to 1.0 seconds.
------------------------------------------------------------
Client connecting to 10.0.2.15, UDP port 9900
Sending 1000 byte datagrams, IPG target: 8000000000.00 us (kalman adju
st)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.0.2.15 port 36059 connected with 10.0.2.15 port 9900
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-20.0 sec  1000 Bytes   400 bits/sec
[  3] Sent 1 datagrams
read failed: Connection refused
[  3] WARNING: did not receive ack of last datagram after 2 tries.
(base) sabrin@sabrin-VirtualBox:~$
```

```
(base) sabrin@sabrin-VirtualBox:~$ iperf -s -u p 9900
iperf: ignoring extra argument -- p
iperf: ignoring extra argument -- 9900
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
```

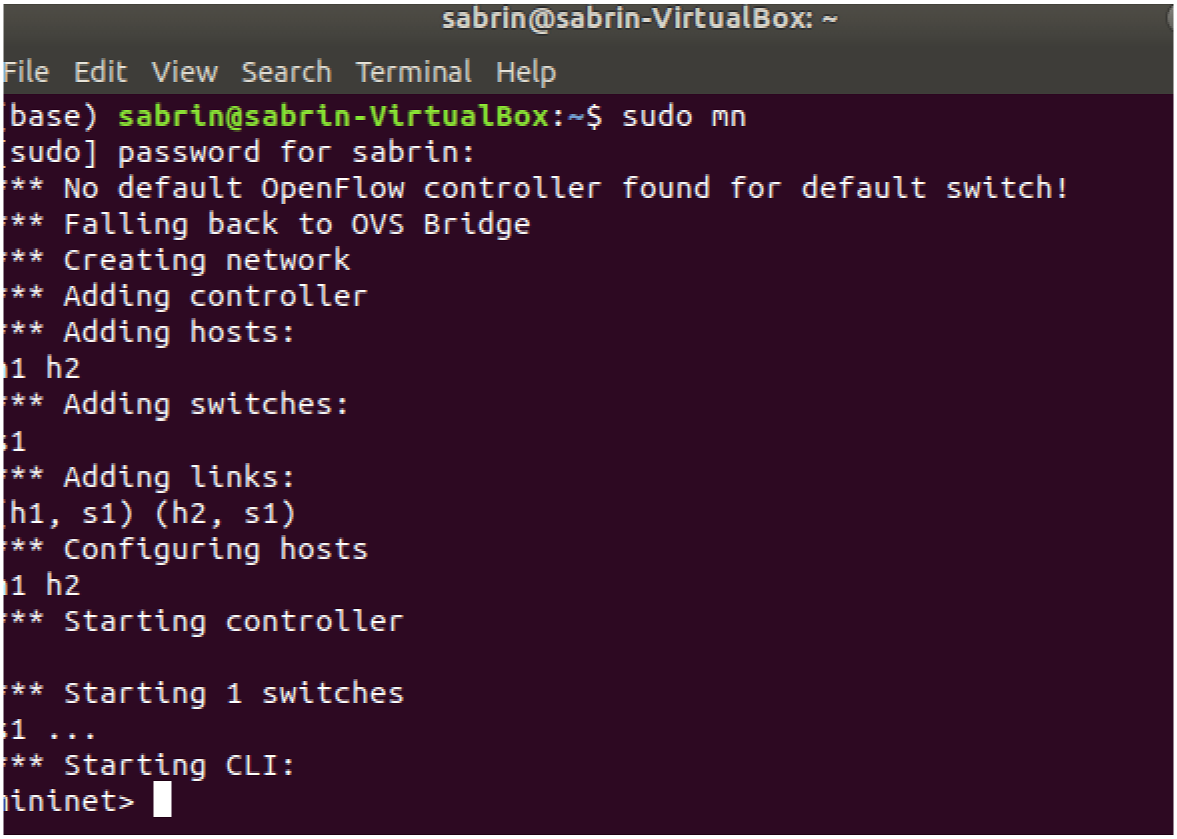
Exercise 4.2.1: Open two Linux terminals, and execute the command line ifconfig in terminal-1. How many interfaces are present?



sabrin@sabrin-VirtualBox: ~
File Edit View Search Terminal Help

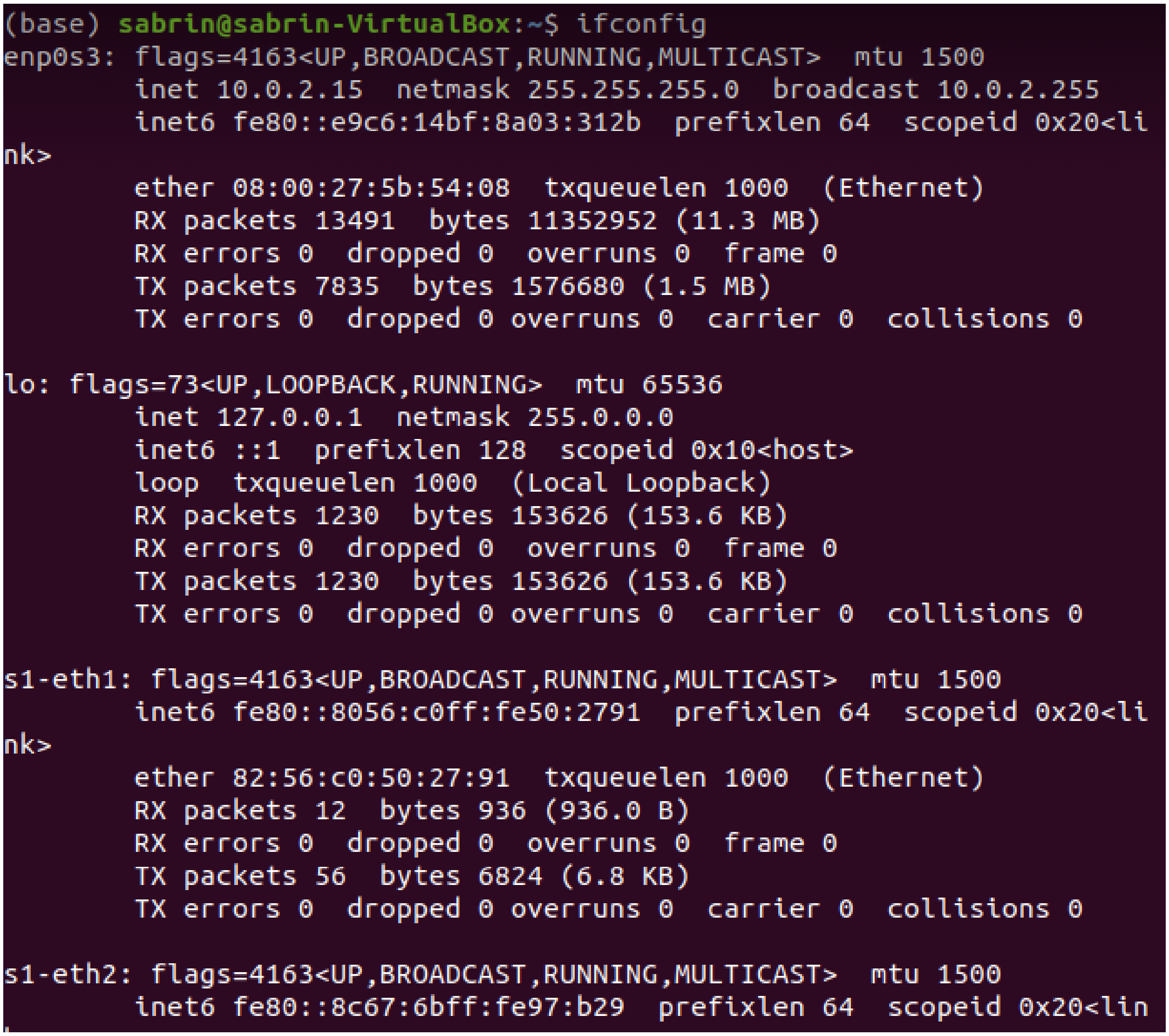
```
(base) sabrin@sabrin-VirtualBox:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::e9c6:14bf:8a03:312b  prefixlen 64  scopeid 0x20<li
nk>
        ether 08:00:27:5b:54:08  txqueuelen 1000  (Ethernet)
        RX packets 11959  bytes 10721107 (10.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6526  bytes 1078235 (1.0 MB)
        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 1032  bytes 133462 (133.4 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1032  bytes 133462 (133.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

(base) sabrin@sabrin-VirtualBox:~$
```

In terminal-2, execute the command line sudo mn, which is the output?

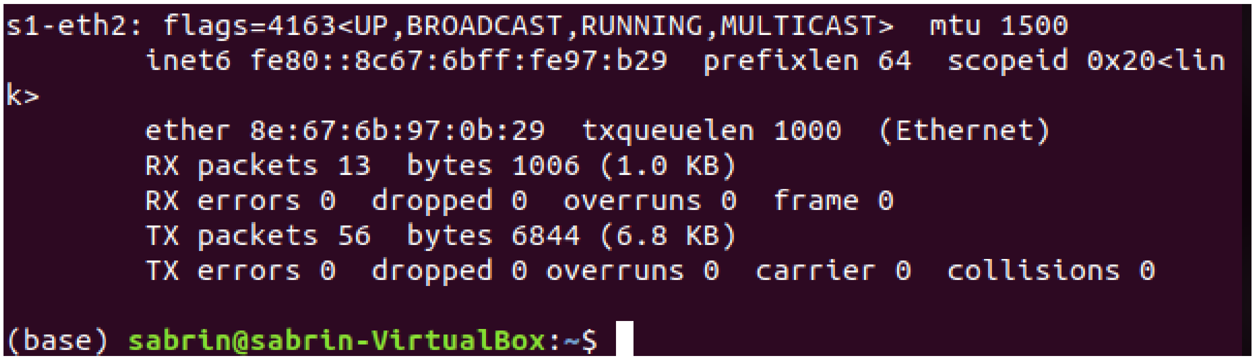

sabrin@sabrin-VirtualBox: ~
File Edit View Search Terminal Help

```
(base) sabrin@sabrin-VirtualBox:~$ sudo mn
[sudo] password for sabrin:
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
1 h2
*** Adding switches:
1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
1 h2
*** Starting controller

*** Starting 1 switches
1 ...
*** Starting CLI:
ininet>
```

In terminal-1 execute the command line ifconfig. How many real and virtual interfaces are present now?



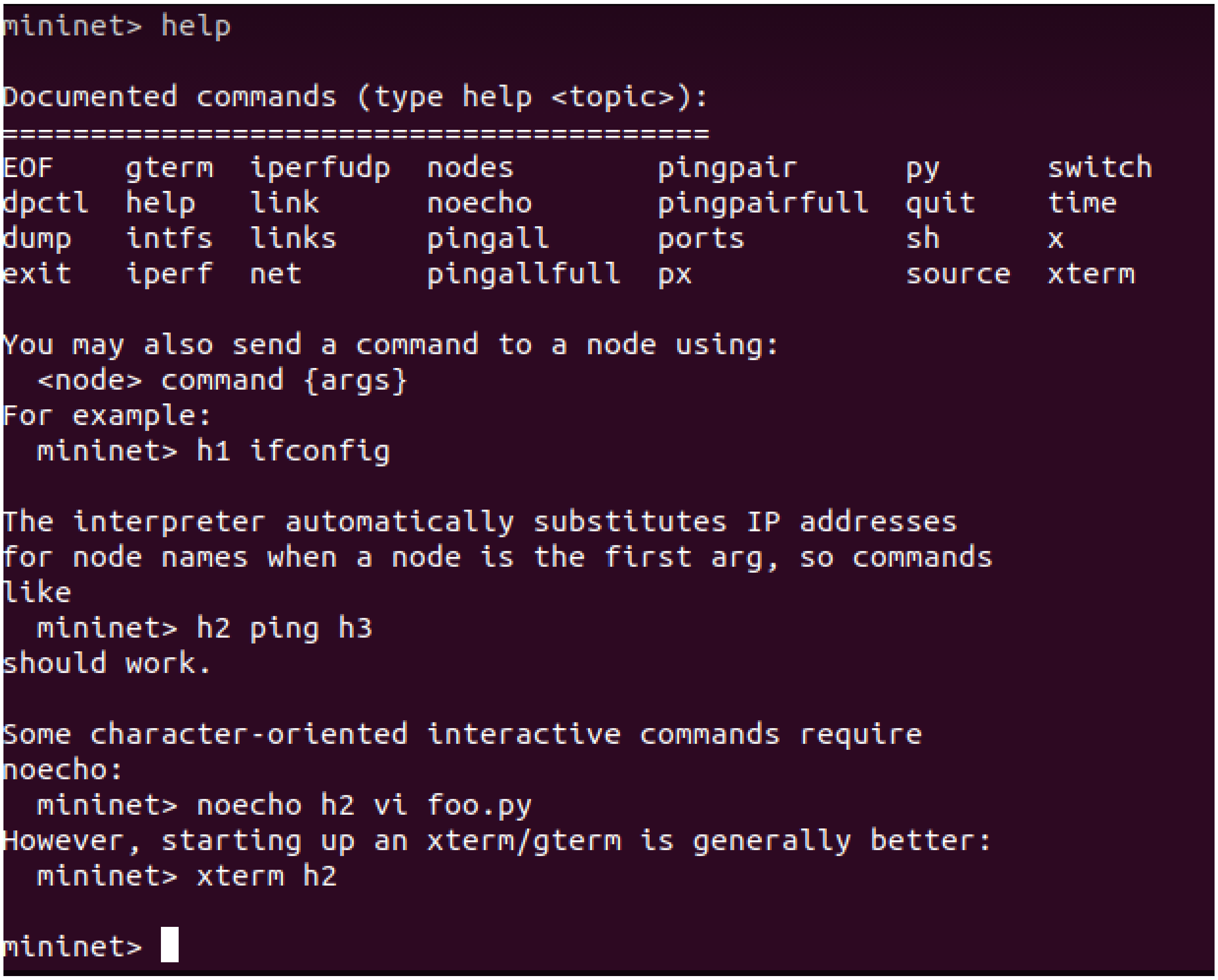
```
(base) sabrin@sabrin-VirtualBox:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::e9c6:14bf:8a03:312b  prefixlen 64  scopeid 0x20<li
nk>
        ether 08:00:27:5b:54:08  txqueuelen 1000  (Ethernet)
        RX packets 13491  bytes 11352952 (11.3 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 7835  bytes 1576680 (1.5 MB)
        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 1230  bytes 153626 (153.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1230  bytes 153626 (153.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::8056:c0ff:fe50:2791  prefixlen 64  scopeid 0x20<li
nk>
        ether 82:56:c0:50:27:91  txqueuelen 1000  (Ethernet)
        RX packets 12  bytes 936 (936.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 56  bytes 6824 (6.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::8c67:6bff:fe97:b29  prefixlen 64  scopeid 0x20<lin
```



```
s1-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::8c67:6bff:fe97:b29  prefixlen 64  scopeid 0x20<lin
k>
        ether 8e:67:6b:97:0b:29  txqueuelen 1000  (Ethernet)
        RX packets 13  bytes 1006 (1.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 56  bytes 6844 (6.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

(base) sabrin@sabrin-VirtualBox:~$ 
```

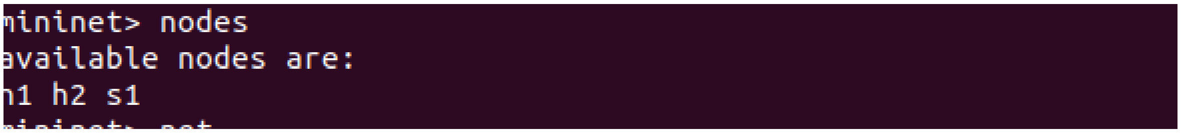
Exercise 4.2.2: Interacting with mininet; in terminal-2, display the following command lines and explain what it does:

mininet> help



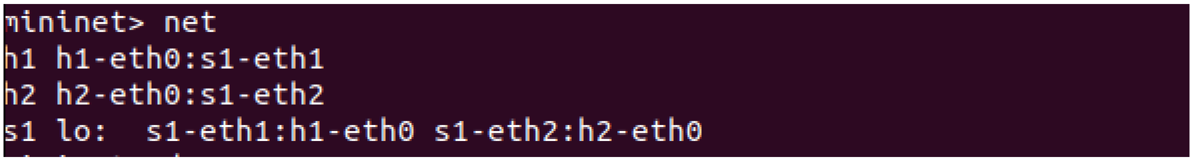
```
mininet> help

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

mininet> 
```

mininet> nodes



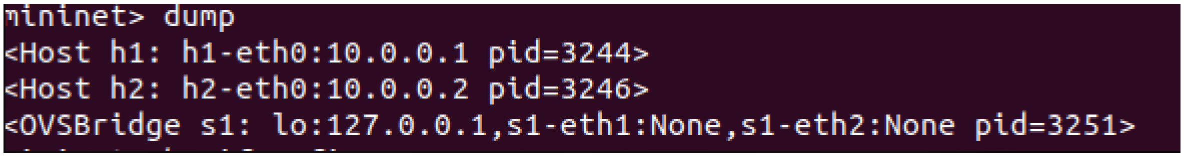
```
mininet> nodes
available nodes are:
h1 h2 s1
mininet> net
```

mininet> net



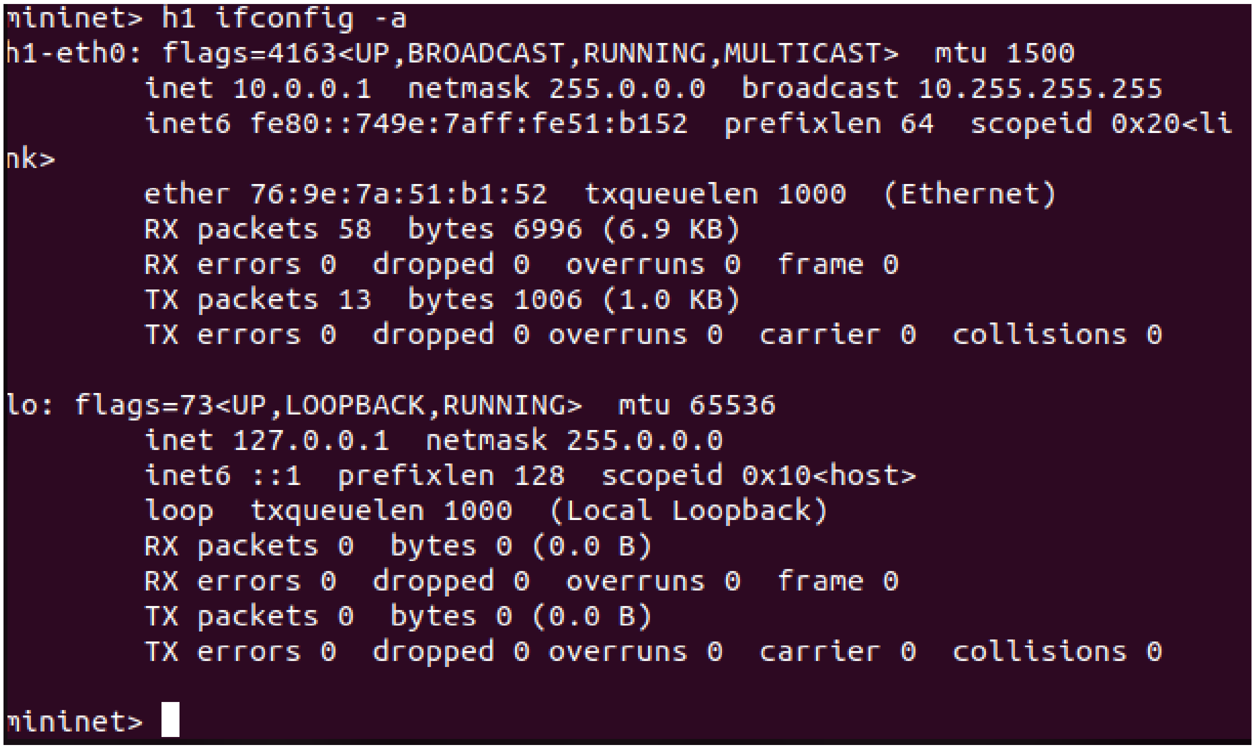
```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
```

mininet> dump



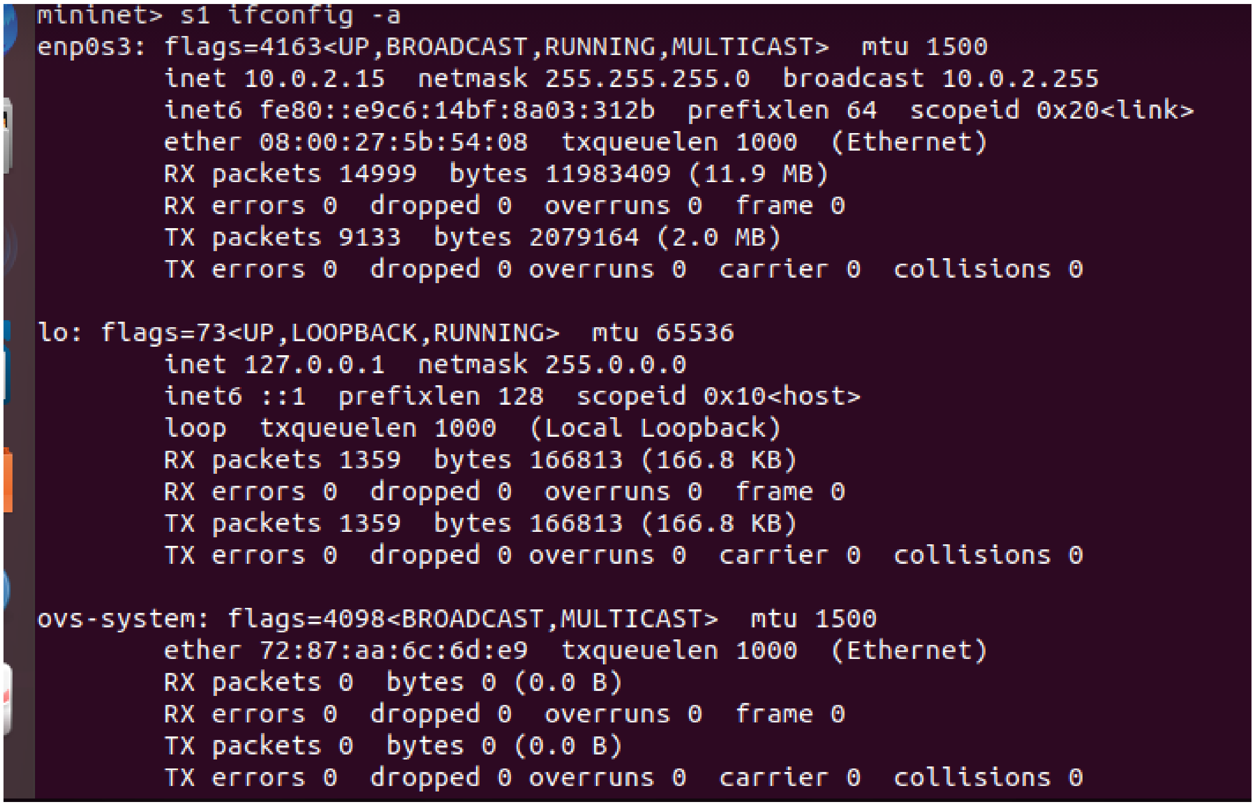
```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=3244>
<Host h2: h2-eth0:10.0.0.2 pid=3246>
<OVSBridge s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=3251>
```

mininet> h1 ifconfig –a

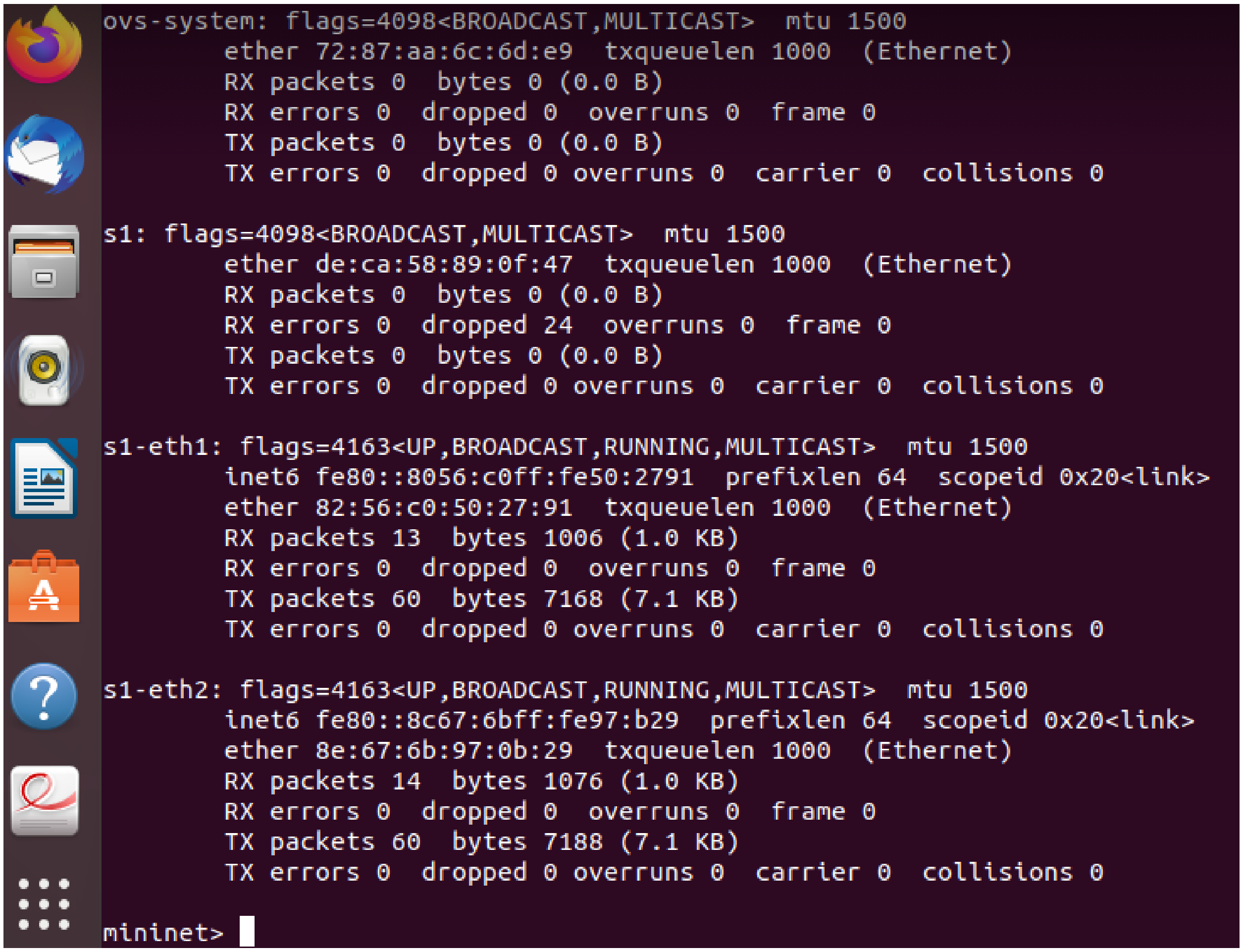


```
mininet> h1 ifconfig -a
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::749e:7aff:fe51:b152  prefixlen 64  scopeid 0x20<li
nk>
        ether 76:9e:7a:51:b1:52  txqueuelen 1000  (Ethernet)
        RX packets 58  bytes 6996 (6.9 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> █
```

mininet> s1 ifconfig –a

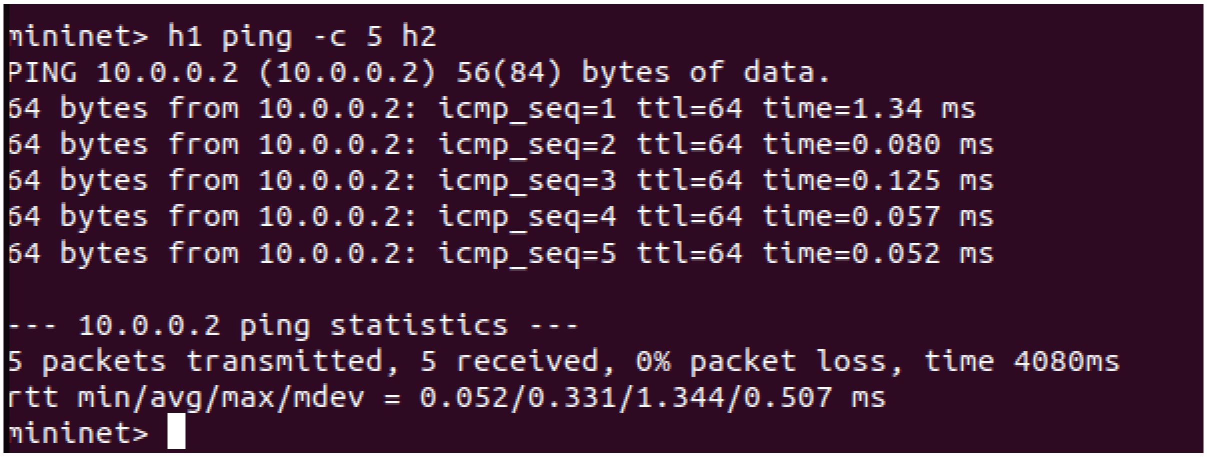


```
mininet> s1 ifconfig -a
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::e9c6:14bf:8a03:312b  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:5b:54:08  txqueuelen 1000  (Ethernet)
        RX packets 14999  bytes 11983409 (11.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9133  bytes 2079164 (2.0 MB)
        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 1359  bytes 166813 (166.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1359  bytes 166813 (166.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ovs-system: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether 72:87:aa:6c:6d:e9  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
```



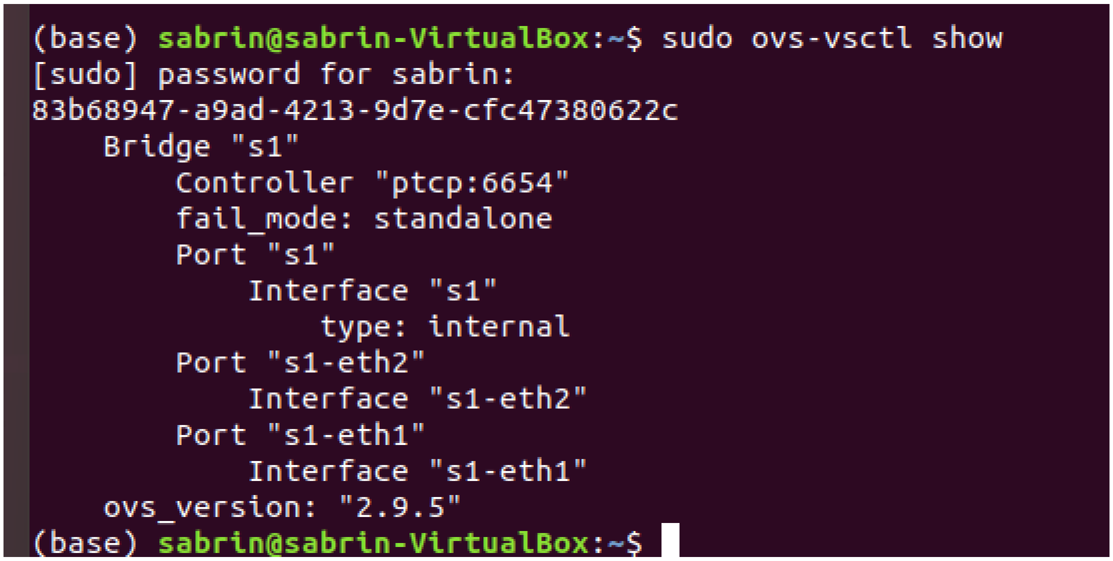
```
ovs-system: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether 72:87:aa:6c:6d:e9  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether de:ca:58:89:0f:47  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::8056:c0ff:fe50:2791  prefixlen 64  scopeid 0x20<link>
        ether 82:56:c0:50:27:91  txqueuelen 1000  (Ethernet)
        RX packets 13  bytes 1006 (1.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 60  bytes 7168 (7.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::8c67:6bff:fe97:b29  prefixlen 64  scopeid 0x20<link>
        ether 8e:67:6b:97:0b:29  txqueuelen 1000  (Ethernet)
        RX packets 14  bytes 1076 (1.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 60  bytes 7188 (7.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

mininet> 
```

mininet> h1 ping -c 5 h2



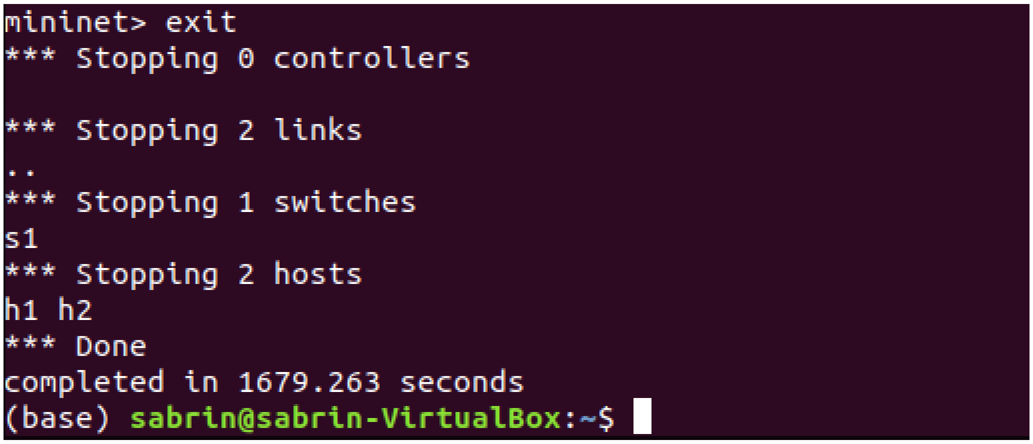
```
mininet> h1 ping -c 5 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.34 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.080 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.125 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.057 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.052 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4080ms
rtt min/avg/max/mdev = 0.052/0.331/1.344/0.507 ms
mininet> 
```

In terminal-1, display the following command line: sudo ovs-vsctl show, what is displayed?



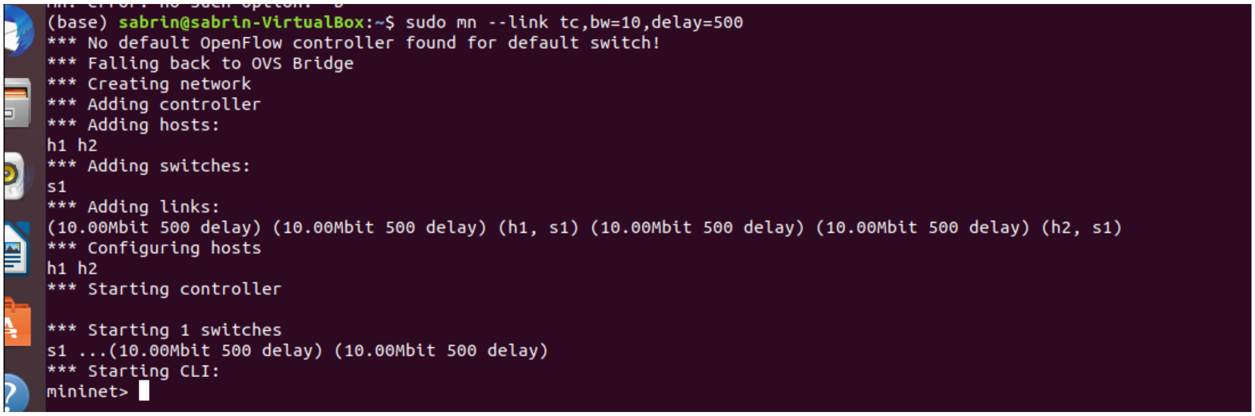
```
(base) sabrin@sabrin-VirtualBox:~$ sudo ovs-vsctl show
[sudo] password for sabrin:
83b68947-a9ad-4213-9d7e-cfc47380622c
    Bridge "s1"
        Controller "ptcp:6654"
        fail_mode: standalone
        Port "s1"
            Interface "s1"
                type: internal
        Port "s1-eth2"
            Interface "s1-eth2"
        Port "s1-eth1"
            Interface "s1-eth1"
    ovs_version: "2.9.5"
(base) sabrin@sabrin-VirtualBox:~$
```

mininet>exit



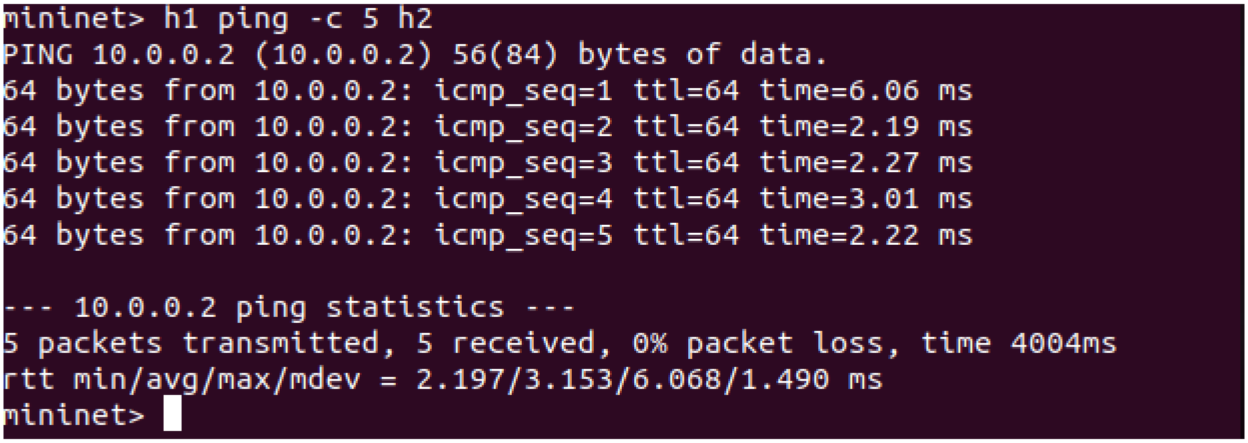
```
mininet> exit
*** Stopping 0 controllers

*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 1679.263 seconds
(base) sabrin@sabrin-VirtualBox:~$
```

Exercise 4.2.3: In terminal-2, display the following command line: sudo mn --link tc,bw=10,delay=500ms



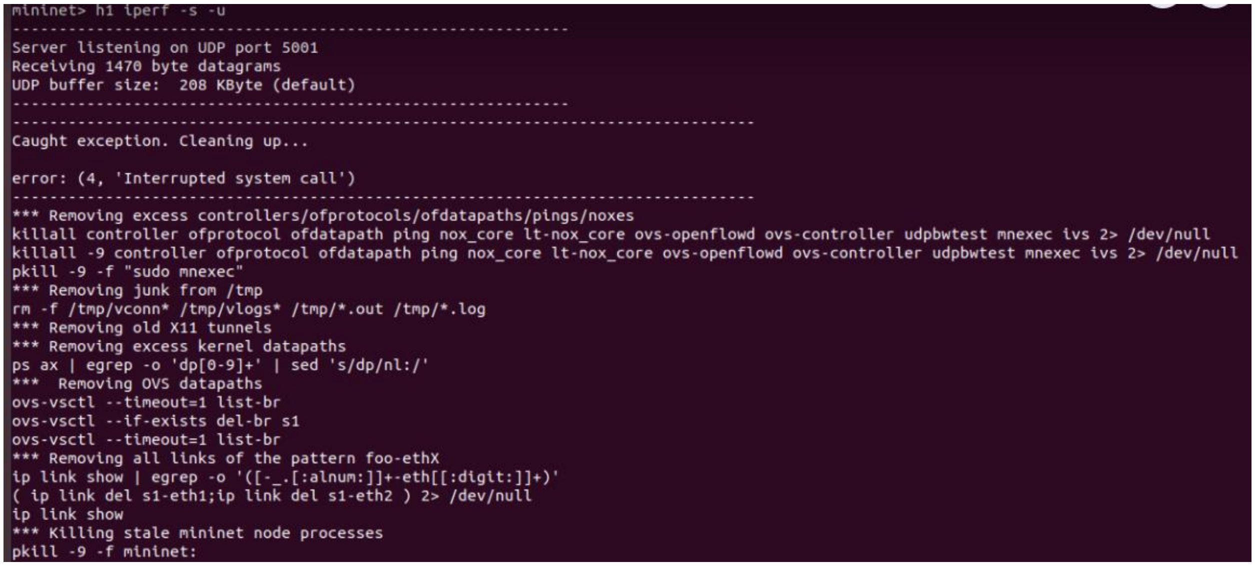
```
(base) sabrin@sabrin-VirtualBox:~$ sudo mn --link tc,bw=10,delay=500
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(10.00Mbit 500 delay) (10.00Mbit 500 delay) (h1, s1) (10.00Mbit 500 delay) (10.00Mbit 500 delay) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

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

mininet> h1 ping -c 5 h2

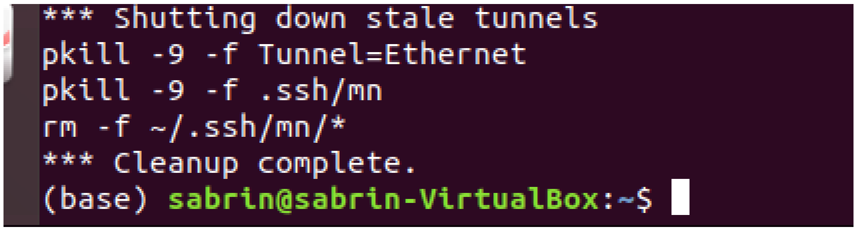


```
mininet> h1 ping -c 5 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=6.06 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=2.19 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=2.27 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=3.01 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=2.22 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4004ms
rtt min/avg/max/mdev = 2.197/3.153/6.068/1.490 ms
mininet>
```

mininet> h1 iperf –s –u

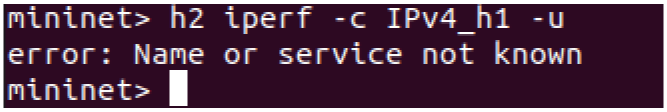


```
mininet> h1 iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
--------------------------------------------------------------------------------
Caught exception. Cleaning up...

error: (4, 'Interrupted system call')
--------------------------------------------------------------------------------
*** 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 --if-exists del-br s1
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
( ip link del s1-eth1;ip link del s1-eth2 ) 2> /dev/null
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.
(base) sabrin@sabrin-VirtualBox:~$ 
```

mininet> h2 iperf –c IPv4_h1 -u



```
mininet> h2 iperf -c IPv4_h1 -u
error: Name or service not known
mininet> 
```

Conclusion :

Software-defined networking (SDN) is an architecture that aims to make networks agile and flexible. The goal of SDN is to improve network control by enabling enterprises and service providers to respond quickly to changing business requirements.

In a software-defined network, a network engineer or administrator can shape traffic from a centralized control console without having to touch individual switches in the network. The centralized SDN controller directs the switches to deliver network services wherever they're needed, regardless of the specific connections between a server and devices.

Mininet is a network emulator which creates a network of virtual hosts, switches, controllers, and links. Mininet hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking.

Mininet:

- Provides a simple and inexpensive network testbed for developing OpenFlow applications
- Enables multiple concurrent developers to work independently on the same topology
- Supports system-level regression tests, which are repeatable and easily packaged
- Enables complex topology testing, without the need to wire up a physical network
- Includes a CLI that is topology-aware and OpenFlow-aware, for debugging or running network-wide tests
- Supports arbitrary custom topologies, and includes a basic set of parametrized topologies
- is usable out of the box without programming, but
- also Provides a straightforward and extensible Python API for network creation and experimentation.

Mininet provides an easy way to get correct system behavior and to experiment with topologies.

Mininet networks run real code including standard Unix/Linux network applications as well as the real Linux kernel and network stack.