Mawlana Bhashani Science and Technology University

Lab-Report

Lab Report No : 04

Lab Report Name : SDN Controllers and Mininet

Course Title : Computer Networks Lab

Submitted by

Name: Moniruzzaman & Mst.Zakia Sultana

ID: IT-18007 & IT-18027

3rd year 2nd semester

Session: 2017-2018

Dept. of ICT

MBSTU.

Submitted To

Nazrul Islam

Assistant Professor

Dept. of ICT

MBSTU.

Theory:

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.

Software-Defined vs. Traditional Networking:

The key difference between traditional and software-defined networking is how SDNs handle data packets. In a traditional network, the way a switch handles an incoming data packet is written into its firmware. Most switches — particularly those used in commercial data centers rather than enterprise environments — respond to and route all packets the same way. SDN provides admins with granular control over the way switches handle data, giving them the ability to automatically prioritize or block certain types of packets. This, in turn, allows for greater efficiency without the need to invest in expensive, application-specific network switches.

Benefits of Software-Defined Networking: There are several benefits to the more advanced level of control afforded by implementing SND in a multi-tenant network environment:

- **Automation:** SND allows for automation of complex operational tasks that make networks faster, more efficient and easier to manage.
- **Increased uptime:** SDN has proven effective in reducing deployment and configuration errors that can lead to service disruptions.
- **Less drain on resources:** SDN gives administrators control over how their routers and switches operate from a single, virtual workflow. This frees up key staff to focus on more important tasks.
- **Better visibility:** With SDN, system administrator’s gain improved visibility into overall network function, allowing them to allocate resources more effectively.
- **Cost savings:** SND can lead to significant overall costs savings. It also reduces the amount of spending required on infrastructure by allowing data centers to get the most use of their existing devices.

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.

Install iperf:

```
zakia@zakia-Inspiron-15-3567: ~
zakia@zakia-Inspiron-15-3567:~$ sudo apt-get install iperf
[sudo] password for zakia:
Reading package lists... Done
Building dependency tree
Reading state information... Done
iperf is already the newest version (2.0.13+dfsg1-1build1).
The following package was automatically installed and is no longer required:
  libfprint-2-tod1
Use 'sudo apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 113 not upgraded.
1 not fully installed or removed.
After this operation, 0 B of additional disk space will be used.
Do you want to continue? [Y/n] y
Setting up openvswitch-testcontroller (2.13.1-0ubuntu0.20.04.1) ...
ln: failed to create symbolic link 'cacert.pem': File exists
dpkg: error processing package openvswitch-testcontroller (--configure):
 installed openvswitch-testcontroller package post-installation script subprocess returned error exit status 1
Errors were encountered while processing:
 openvswitch-testcontroller
E: Sub-process /usr/bin/dpkg returned an error code (1)
zakia@zakia-Inspiron-15-3567:~$
```

Install Mininet:

```
zakia@zakia-Inspiron-15-3567:~$ sudo apt-get install mininet
Reading package lists... Done
Building dependency tree
Reading state information... Done
mininet is already the newest version (2.2.2-5ubuntu1).
The following package was automatically installed and is no longer required:
  libfprint-2-tod1
Use 'sudo apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 113 not upgraded.
1 not fully installed or removed.
After this operation, 0 B of additional disk space will be used.
Do you want to continue? [Y/n] y
Setting up openvswitch-testcontroller (2.13.1-0ubuntu0.20.04.1) ...
ln: failed to create symbolic link 'cacert.pem': File exists
dpkg: error processing package openvswitch-testcontroller (--configure):
 installed openvswitch-testcontroller package post-installation script subprocess returned error exit status 1
Errors were encountered while processing:
 openvswitch-testcontroller
E: Sub-process /usr/bin/dpkg returned an error code (1)
zakia@zakia-Inspiron-15-3567:~$
```

4. Exercises

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

```
zakia@zakia-Inspiron-15-3567:~$ 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 U
DP=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 a
nd 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 (defaul
t not set)
      --udp-histogram #,#  enable UDP latency histogram(s) with bin width and count, e.g. 1,1000=1(ms),1000(bi
ns)
  -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
      --isochronous <frames-per-second>:<mean>,<stddev> send traffic in bursts (frames - emulate video traffi
c)
  -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 ip (and optional port) from which to source 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/>
Report bugs to <iperf-users@lists.sourceforge.net>
zakia@zakia-Inspiron-15-3567:~$
```

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).

For terminal -1:

```
zakia@zakia-Inspiron-15-3567: ~
zakia@zakia-Inspiron-15-3567:~$ iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size:  128 KByte (default)
------------------------------------------------------------
```

For terminal -2:

```
zakia@zakia-Inspiron-15-3567: ~
zakia@zakia-Inspiron-15-3567:~$ 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 58362 connected with 127.0.0.1 port 5001
read failed: Connection refused
[  3] WARNING: did not receive ack of last datagram after 1 tries.
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  3] Sent 891 datagrams
zakia@zakia-Inspiron-15-3567:~$
```

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? Which are the statistics are provided at the end of transmission?

```
zakia@zakia-Inspiron-15-3567: ~
zakia@zakia-Inspiron-15-3567:~$ 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 42980 connected with 127.0.0.1 port 5001
read failed: Connection refused
[  3] WARNING: did not receive ack of last datagram after 1 tries.
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  3] Sent 891 datagrams
zakia@zakia-Inspiron-15-3567:~$
```

```
zakia@zakia-Inspiron-15-3567: ~
zakia@zakia-Inspiron-15-3567:~$ iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 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:

o Packet length = 1000bytes

o Time = 20 seconds

o Bandwidth = 1Mbps

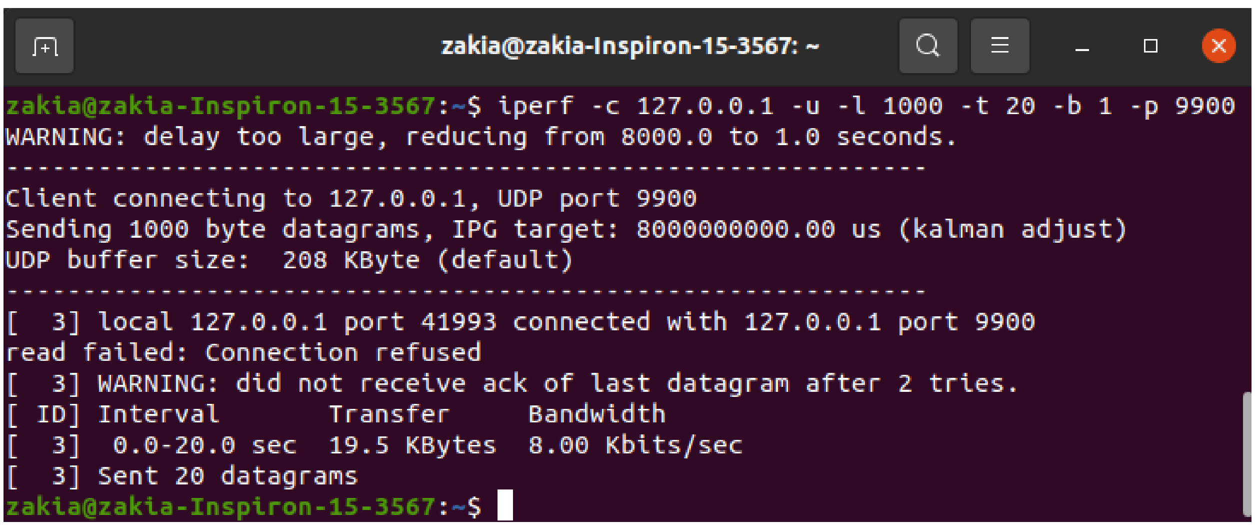
o Port = 9900

Which are the command lines?

The command lines are:

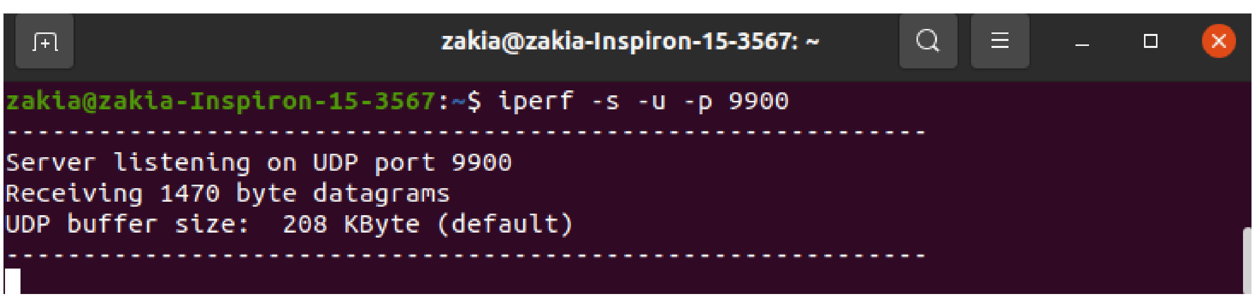
For terminal 1:

Iperf –c 127.0.0.1 –u –l 1000 –t 20 –b 1 –p 9900



For terminal 2:

Iperf –s –u –p 9900

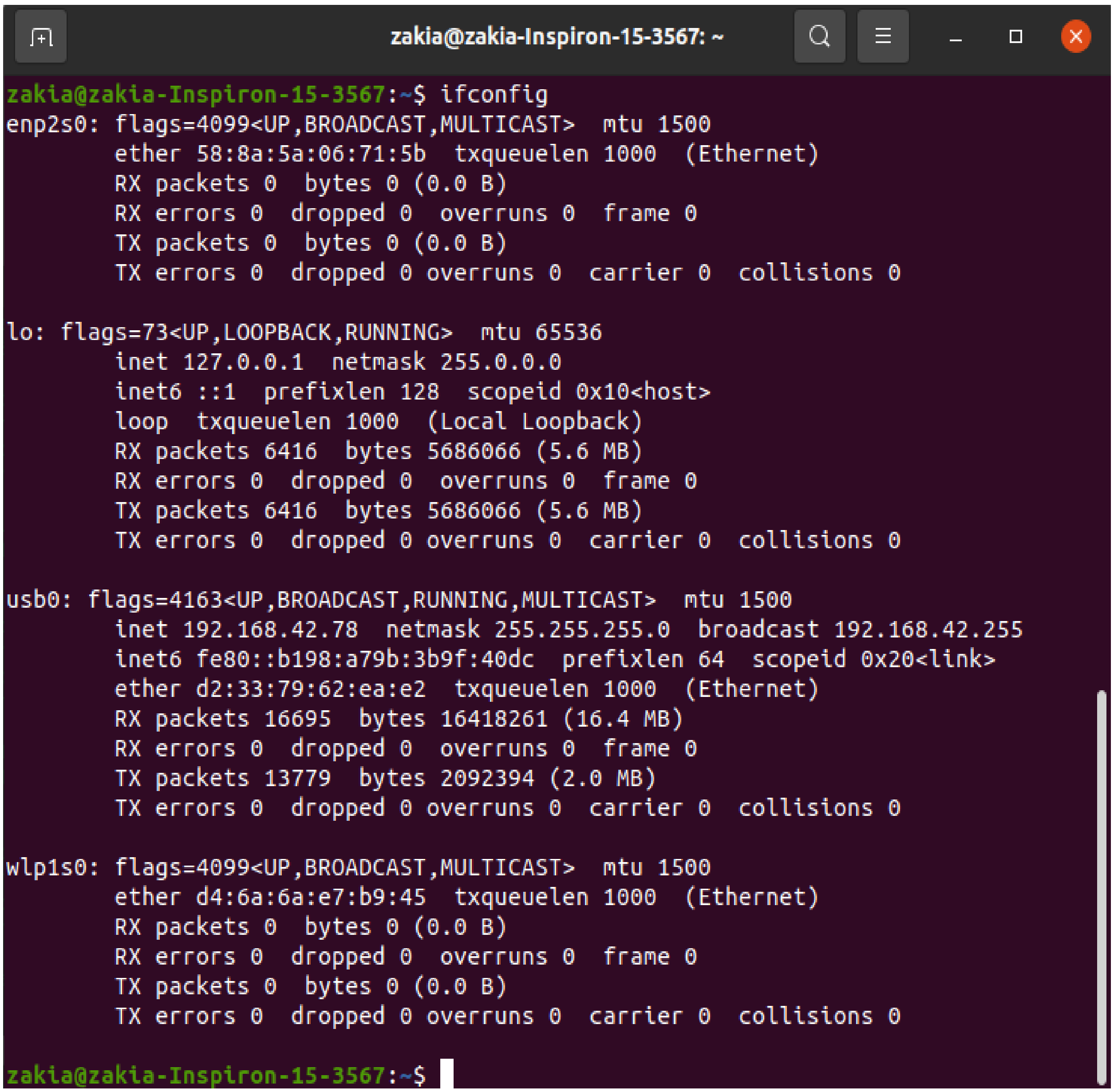


Using Mininet

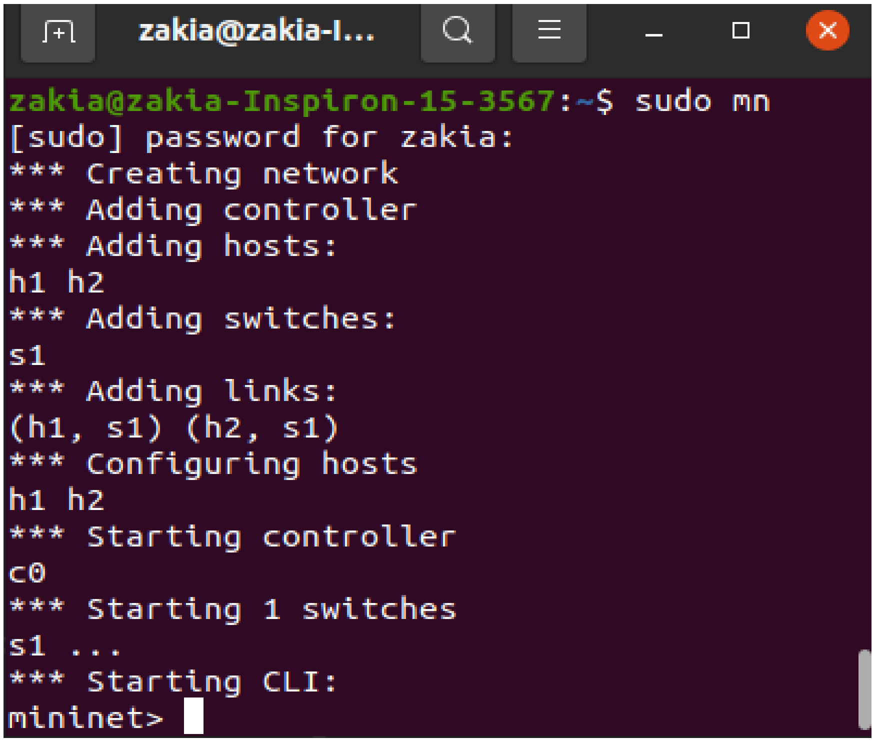
Exercise 4.2.1: Open two Linux terminals, and execute the command line ifconfig in terminal1. How many interfaces are present?

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

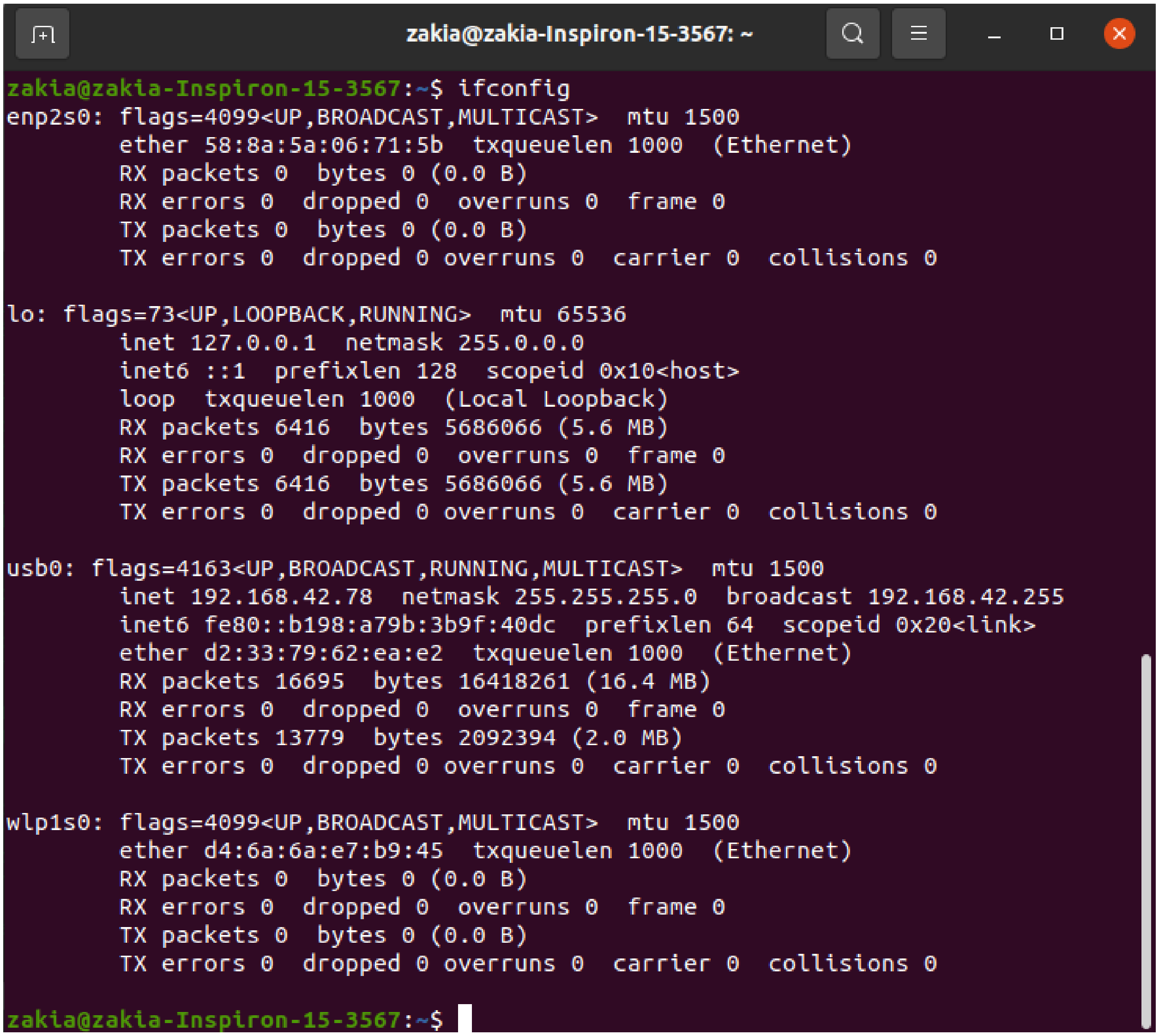
In terminal-1 execute the command line ifconfig. How many real and virtual interfaces are present now?



```
zakia@zakia-Inspiron-15-3567: ~
zakia@zakia-Inspiron-15-3567:~$ ifconfig
enp2s0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 58:8a:5a:06:71:5b  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 6416  bytes 5686066 (5.6 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6416  bytes 5686066 (5.6 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

usb0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.42.78  netmask 255.255.255.0  broadcast 192.168.42.255
        inet6 fe80::b198:a79b:3b9f:40dc  prefixlen 64  scopeid 0x20<link>
        ether d2:33:79:62:ea:e2  txqueuelen 1000  (Ethernet)
        RX packets 16695  bytes 16418261 (16.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 13779  bytes 2092394 (2.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp1s0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether d4:6a:6a:e7:b9:45  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

zakia@zakia-Inspiron-15-3567:~$
```



```
zakia@zakia-Inspiron-15-3567:~$ sudo mn
[sudo] password for zakia:
*** 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>
```

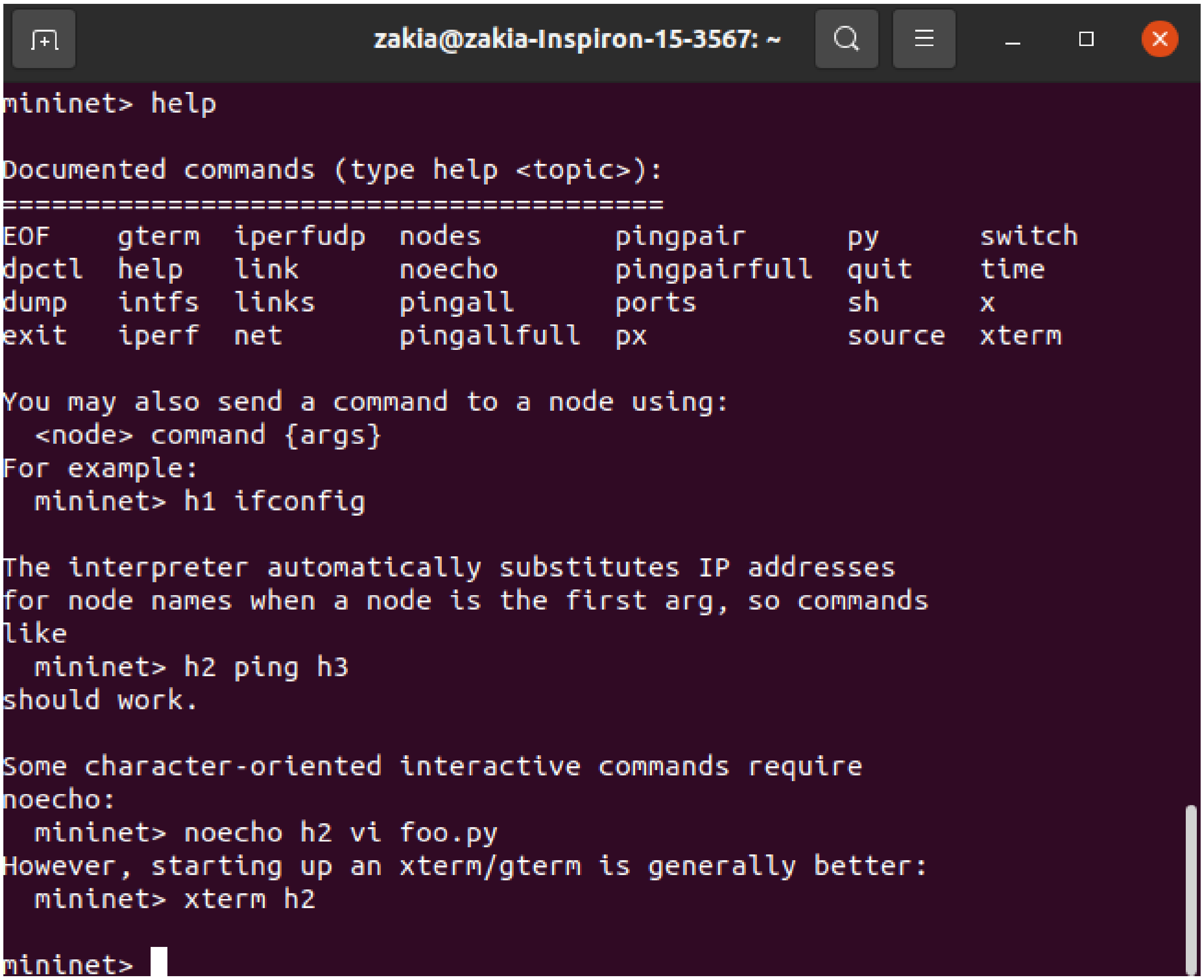


zakia@zakia-Inspiron-15-3567: ~

```
zakia@zakia-Inspiron-15-3567:~$ ifconfig
enp2s0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 58:8a:5a:06:71:5b  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 6416  bytes 5686066 (5.6 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6416  bytes 5686066 (5.6 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

usb0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.42.78  netmask 255.255.255.0  broadcast 192.168.42.255
        inet6 fe80::b198:a79b:3b9f:40dc  prefixlen 64  scopeid 0x20<link>
        ether d2:33:79:62:ea:e2  txqueuelen 1000  (Ethernet)
        RX packets 16695  bytes 16418261 (16.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 13779  bytes 2092394 (2.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp1s0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether d4:6a:6a:e7:b9:45  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

zakia@zakia-Inspiron-15-3567:~$
```

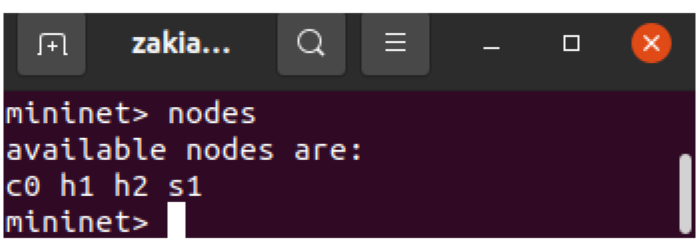
Exercise 4.2.2: Interacting with mininet; in terminal-2, display the following command lines and explain what it does:

mininet> help



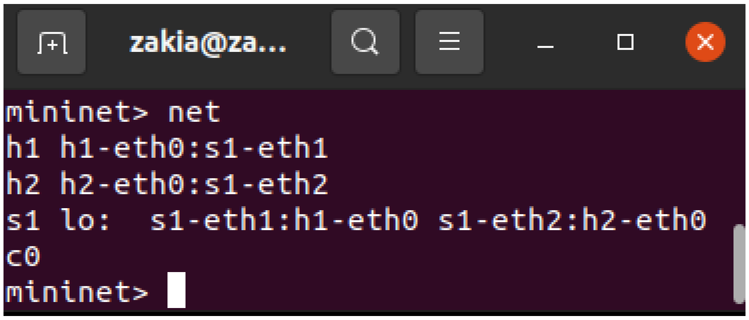
```
zakia@zakia-Inspiron-15-3567: ~
mininet> help

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



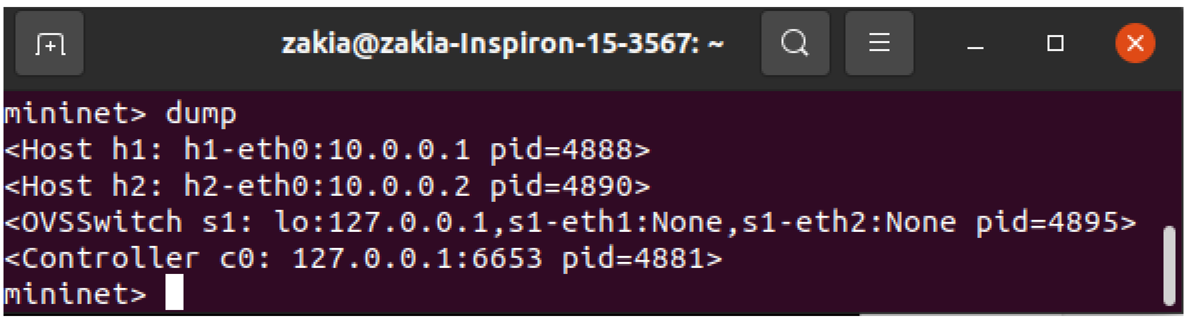
```
zakia...
mininet> nodes
available nodes are:
c0 h1 h2 s1
mininet>
```

mininet> net



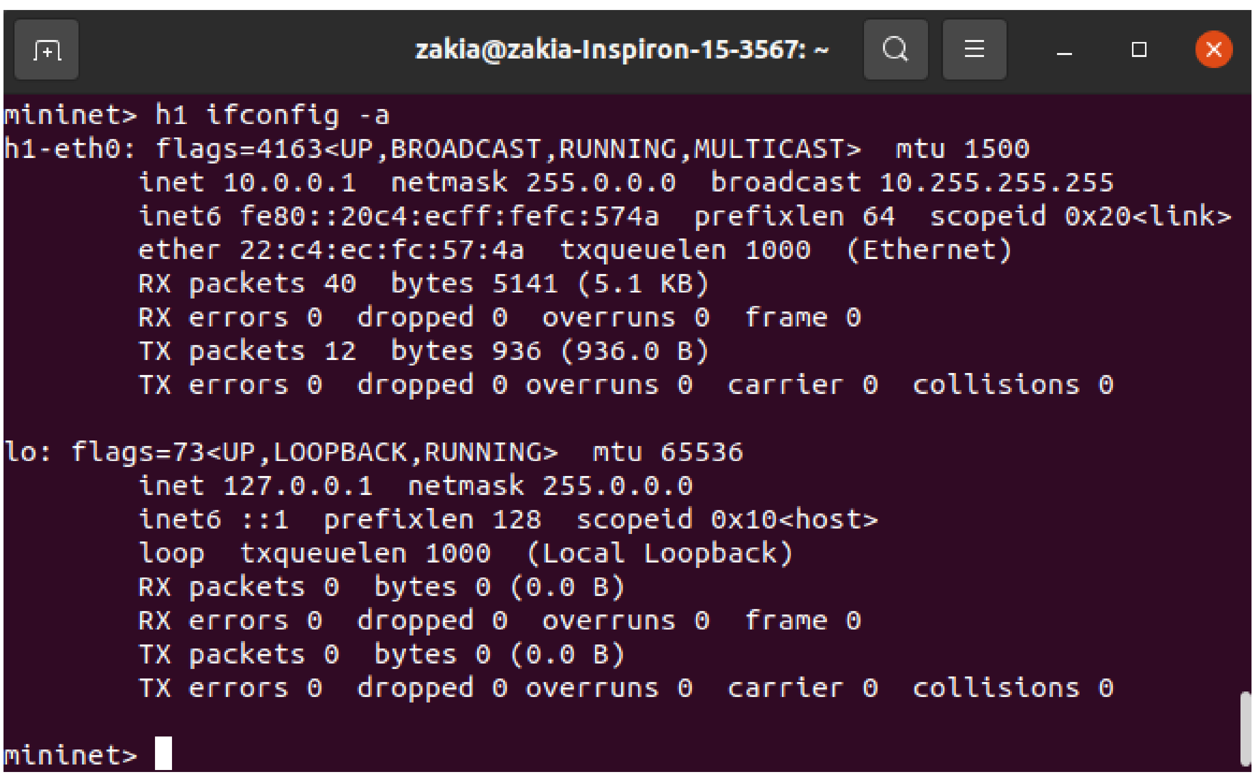

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
mininet> 
```

mininet> dump




```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=4888>
<Host h2: h2-eth0:10.0.0.2 pid=4890>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=4895>
<Controller c0: 127.0.0.1:6653 pid=4881>
mininet> 
```

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::20c4:ecff:fefc:574a  prefixlen 64  scopeid 0x20<link>
        ether 22:c4:ec:fc:57:4a  txqueuelen 1000  (Ethernet)
        RX packets 40  bytes 5141 (5.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 12  bytes 936 (936.0 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> 
```

mininet> s1 ifconfig –a

zakia@zakia-Inspiron-15-3567: ~

```
mininet> s1 ifconfig -a
enp2s0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 58:8a:5a:06:71:5b  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 6946  bytes 5726953 (5.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6946  bytes 5726953 (5.7 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ovs-system: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether 06:83:66:fd:de:4a  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 ce:95:30:70:74:41  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 22  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::8843:52ff:fe99:304a  prefixlen 64  scopeid 0x20<link>
        ether 8a:43:52:99:30:4a  txqueuelen 1000  (Ethernet)
        RX packets 12  bytes 936 (936.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
```

```
s1-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::942d:3ff:fe52:b6f8  prefixlen 64  scopeid 0x20<link>
        ether 96:2d:03:52:b6:f8  txqueuelen 1000  (Ethernet)
        RX packets 12  bytes 936 (936.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 40  bytes 5141 (5.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

usb0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.42.78  netmask 255.255.255.0  broadcast 192.168.42.255
        inet6 fe80::b198:a79b:3b9f:40dc  prefixlen 64  scopeid 0x20<link>
        ether d2:33:79:62:ea:e2  txqueuelen 1000  (Ethernet)
        RX packets 17865  bytes 16613917 (16.6 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 15085  bytes 2276072 (2.2 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp1s0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether d4:6a:6a:e7:b9:45  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

mininet>
```

mininet> h1 ping -c 5 h2

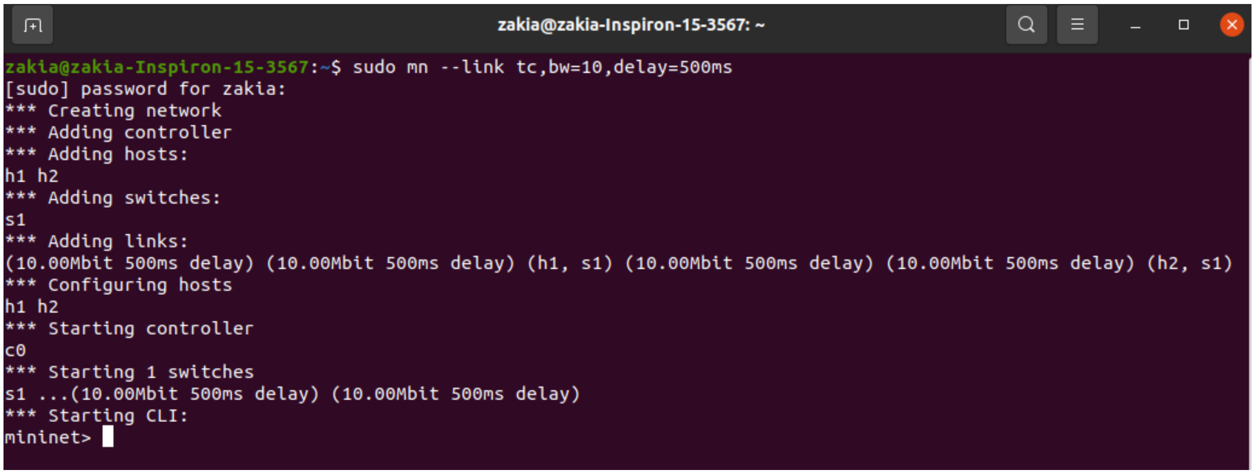
```
zakia@zakia-Inspiron-15-3567: ~

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=39.3 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.885 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.139 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.097 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.117 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4054ms
rtt min/avg/max/mdev = 0.097/8.111/39.317/15.605 ms
mininet>
```

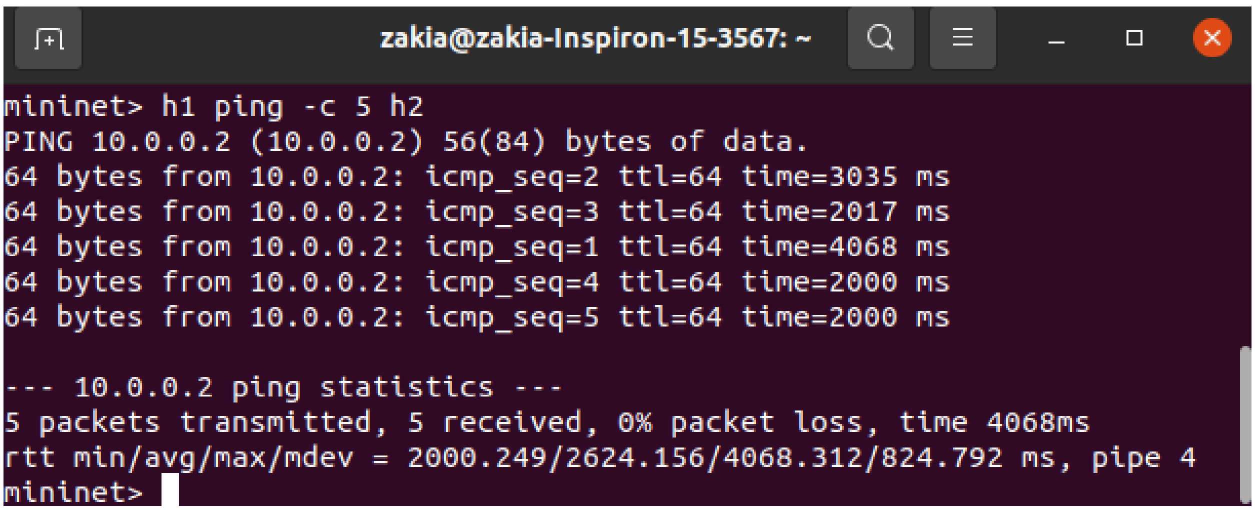
Exercise 4.2.3: In terminal-2, display the following command line: sudo mn --link tc,bw=10,delay=500ms

o mininet> h1 ping -c 5 h2, What happen with the link?

o mininet> h1 iperf –s –u &

o mininet> h2 iperf –c IPv4_h1 -u, Is there any packet loss?





Exercise 4.3.1: Remote controller: The following script reproduce same architecture as before using an external controller: (save as remote_controller.py)

Code:

*remote_controller

```python
from mininet.cli import CLI
from mininet.net import Mininet
from mininet.node import RemoteController


if'__main__' == __name__:
    net = Mininet(controller=RemoteController)
    c0 = net.addController('c0', port=6633)
    s1 = net.addSwitch('s1')

    h1 = net.addHost('h1')
    h2 = net.addHost('h2')

    net.addLink(s1, h1)
    net.addLink(s1, h2)

    net.build()
    c0.start()
    s1.start([c0])
    net.startTerms()
    CLI(net)
    net.stop()
```

The following scrip defines the behavior of the controller: (save as simple_switch_13.py, it is also available in blackboard)

*simple_switch_13

```python
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet


class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # install table-miss flow entry
        #
        # We specify NO BUFFER to max_len of the output action due to
        # OVS bug. At this moment, if we specify a lesser number, e.g.,
        # 128, OVS will send Packet-In with invalid buffer_id and
        # truncated packet data. In that case, we cannot output packets
        # correctly.          match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                          ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)

    def add_flow(self, datapath, priority, match, actions):
```

```python
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]

        mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                match=match, instructions=inst)
        datapath.send_msg(mod)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']
        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocols(ethernet.ethernet)[0]

        dst = eth.dst
        src = eth.src

        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})

        self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

        # learn a mac address to avoid FLOOD next time.
        self.mac_to_port[dpid][src] = in_port

        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        else:
            out_port = ofproto.OFPP_FLOOD

        actions = [parser.OFPActionOutput(out_port)]

        # install a flow to avoid packet_in next time
        if out_port != ofproto.OFPP_FLOOD:
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
            self.add_flow(datapath, 1, match, actions)

        data = None
        if msg.buffer_id == ofproto.OFP_NO_BUFFER:
            data = msg.data

        out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                                  in_port=in_port, actions=actions, data=data)
        datapath.send_msg(out)
```

Conclusion:

Mininet is a useful tool for teaching, development and research. With it, a realistic virtual network, running a real kernel switch and application code, can be set up in a few seconds on a single machine, either virtual or native. It is actively developed and supported. Emulation refers to the running of unchanged code on virtual hardware on the top of the physical host, interactively. It is handy, practical and low cost. It comes with certain restrictions, though, like slower speeds compared to running the same code on a hardware test-bed which is fast and accurate, but expensive. While a simulator requires code modifications and is slow as well. Mininet is a network emulator that enables the creation of a network of virtual hosts, switches, controllers, and links. Mininet hosts standard Linux network software, and its switches support OpenFlow, a software defined network (SDN) for highly flexible custom routing. It constructs a virtual network that appears to be a real physical network. You can create a network topology, simulate it and implement the various network performance parameters such as bandwidth, latency, packet loss, etc, with Mininet, using simple code. You can create the virtual network on a single machine (a VM, the cloud or a native machine). Mininet permits the creation of multiple nodes (hosts, switches or controllers), enabling a big network to be simulated on a single PC. This is very useful in experimenting with various topologies and different controllers, for different network scenarios. The programs that you run can send packets through virtual switches that seem like real Ethernet interfaces, with a given link speed and delay. Packets get processed by what looks like a real Ethernet switch, router, or middle-box, with a given amount of queuing. The Mininet CLI and API facilitate easy interaction with our network. Virtual hosts, switches, links and controllers created through Mininet are the real thing. They are just created using the Mininet emulator rather than hardware and for the most part, their behaviour is similar to discrete hardware elements**.**

Questions:

Question 5.1: Explain how the traffic generators work?

Answer:

There are a number of ways that traffic generator programs work. Some of them are purely robotic, while others are more detailed SEO-task programs that, while they do what they do very effectively, they don’t play by the rules and can earn your site several search penalties. In every case, you should avoid using such programs.

- Robotic refreshing
- Traffic routed through proxy servers
- Spoofed user agents
- Clickfarm traffic
- Hijacked traffic from hacked sites
- Automatic article submission.

Question 5.3: Which is the main difference between configuring UDP and TCP traffic?

Answer:

Transmission control protocol (TCP)	**User datagram protocol (UDP)**
TCP is a connection-oriented protocol. Connection-orientation means that the communicating devices should establish a connection before transmitting data and should close the connection after transmitting the data.	UDP is the Datagram oriented protocol. This is because there is no overhead for opening a connection, maintaining a connection, and terminating a connection. UDP is efficient for broadcast and multicast type of network transmission.
TCP is reliable as it guarantees delivery of data to the destination router.	The delivery of data to the destination cannot be guaranteed in UDP.
TCP provides extensive error checking mechanisms. It is because it provides flow control and acknowledgment of data.	UDP has only the basic error checking mechanism using checksums.
Sequencing of data is a feature of Transmission Control Protocol (TCP). this means that packets arrive in-order at the receiver.	There is no sequencing of data in UDP. If ordering is required, it has to be managed by the application layer.
TCP is comparatively slower than UDP.	UDP is faster, simpler and more efficient than TCP.
Retransmission of lost packets is possible in TCP, but not in UDP.	There is no retransmission of lost packets in User Datagram Protocol (UDP).
TCP has a (20-80) bytes variable length header.	UDP has a 8 bytes fixed length header.

Question 5.4: In your opinions which are the main advantages and disadvantages of working with mininet? Provide at least 3

Answer:

Advantages:

- **Boots faster**: seconds instead of minutes
- **Scales larger**: hundreds of hosts and switches vs. single digits
- **Provides more bandwidth**: typically 2Gbps total bandwidth on modest hardware
- **Installs easily**: a prepackaged VM is available that runs on VMware or VirtualBox for Mac/Win/Linux with OpenFlow v1.0 tools already installed.

Disadvantages:

Mininet-based networks cannot (currently) exceed the CPU or bandwidth available on a single server.

Mininet cannot (currently) run non-Linux-compatible OpenFlow switches or applications; this has not been a major issue in practice.

Question 5.6: What is the advantage of having a programmable Controller?

Answer:

- many inputs and outputs, excellent for controlling and monitoring many processes.
- designed for industrial environments, very robust and reliable.
- reprogrammable.
- modular.
- ideally suited to supervisory control.
- easy to set up and good for FMS environment.