Mawlana Bhashani Science and Technology University

Lab-Report

Report No: 04

Course code:ICT-3110

Course title: Operating Systems Lab

Lab Report Name: SDN Controllers and Mininet

Date of Performance:

Date of Submission: 11/09/2020

Submitted by

Name:Golam Kibria Tuhin

ID:IT-18015

3th year 1ndsemester

Session: 2017-2018

Dept. of ICT

MBSTU.

Submitted To

Nazrul Islam

Assistant Professor

Dept. of ICT

MBSTU.

Theory:

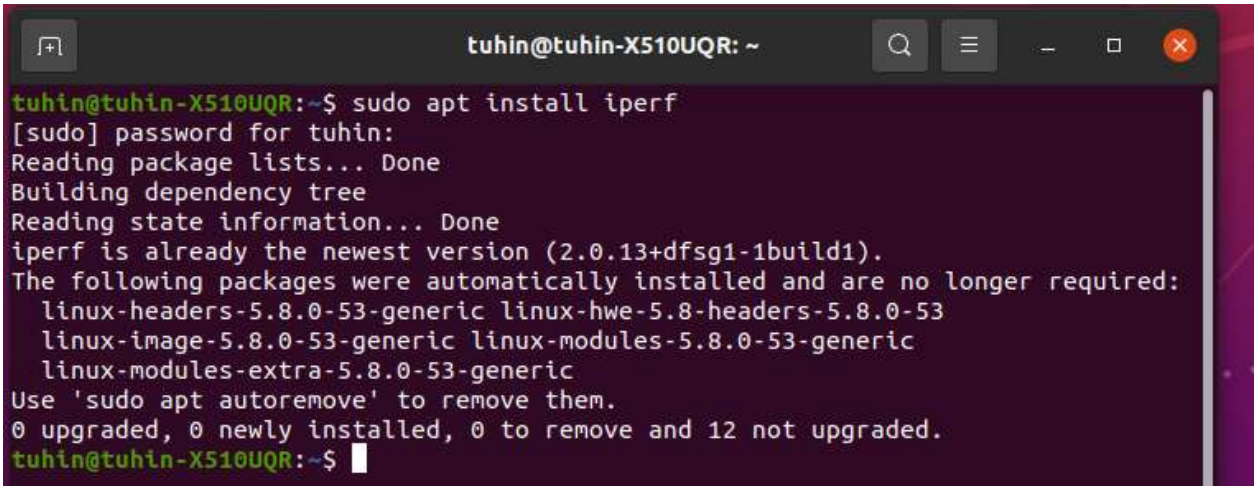
What is iPerf?

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. 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:



Install Mininet:

```
tuhin@tuhin-X510UQR: ~
tuhin@tuhin-X510UQR:~$ sudo apt 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 packages were automatically installed and are no longer required:
  linux-headers-5.8.0-53-generic linux-hwe-5.8-headers-5.8.0-53
  linux-image-5.8.0-53-generic linux-modules-5.8.0-53-generic
  linux-modules-extra-5.8.0-53-generic
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 12 not upgraded.
tuhin@tuhin-X510UQR:~$
```

4. Exercise

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

```
tuhin@tuhin-X510UQR: ~
tuhin@tuhin-X510UQR:~$ 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 p
er second
  -e, --enhancedreports    use enhanced reporting giving more tcp/udp and traffi
c 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 (Default
s: 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 specifie
d 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 multicas
t address) and optional port and device
  -C, --compatibility      for use with older versions does not sent extra msgs
```

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:

```
tuhin@tuhin-X510UQR: ~
tuhin@tuhin-X510UQR:~$ iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size:  128 KByte (default)
------------------------------------------------------------
```

For terminal -2:

```
tuhin@tuhin-X510UQR: ~
tuhin@tuhin-X510UQR:~$ 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 59646 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
tuhin@tuhin-X510UQR:~$
```

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?

```
tuhin@tuhin-X510UQR: ~
tuhin@tuhin-X510UQR:~$ 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 59646 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
tuhin@tuhin-X510UQR:~$
```

```
tuhin@tuhin-X510UQR: ~
tuhin@tuhin-X510UQR:~$ 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

```
tuhin@tuhin-X510UQR: ~
tuhin@tuhin-X510UQR:~$ iperf -c 127.0.0.1 -u -l 1000 -t 20 -b 1 -p 9900
WARNING: delay too large, reducing from 8000.0 to 1.0 seconds.
------------------------------------------------------------
Client connecting to 127.0.0.1, UDP port 9900
Sending 1000 byte datagrams, IPG target: 8000000000.00 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 127.0.0.1 port 39311 connected with 127.0.0.1 port 9900
read failed: Connection refused
[  3] WARNING: did not receive ack of last datagram after 2 tries.
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-20.0 sec  19.5 KBytes  8.00 Kbits/sec
[  3] Sent 20 datagrams
tuhin@tuhin-X510UQR:~$
```

For terminal 2:

Iperf –s –u –p 9900

```
tuhin@tuhin-X510UQR: ~
tuhin@tuhin-X510UQR:~$ 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 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?

```
tuhin@tuhin-X510UQR: ~
tuhin@tuhin-X510UQR:~$ ifconfig
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 8633  bytes 2853928 (2.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8633  bytes 2853928 (2.8 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp2s0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 60:f6:77:ee:e5:fd  txqueuelen 1000  (Ethernet)
        RX packets 308882  bytes 197901297 (197.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 313692  bytes 93904009 (93.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

tuhin@tuhin-X510UQR:~$
```

```
tuhin@tuhin-X510UQR:~$ sudo mn
[sudo] password for tuhin:
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

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

```
tuhin@tuhin-X510UQR: ~
tuhin@tuhin-X510UQR: ~    tuhin@tuhin-X510UQR: ~
tuhin@tuhin-X510UQR:~$ ifconfig
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 8834  bytes 2867892 (2.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8834  bytes 2867892 (2.8 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::e80b:6bff:fe02:7ffa  prefixlen 64  scopeid 0x20<link>
        ether ea:0b:6b:02:7f:fa  txqueuelen 1000  (Ethernet)
        RX packets 9  bytes 726 (726.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 30  bytes 3972 (3.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::dc09:58ff:fed1:91a3  prefixlen 64  scopeid 0x20<link>
        ether de:09:58:d1:91:a3  txqueuelen 1000  (Ethernet)
        RX packets 9  bytes 726 (726.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
```

Exercise 4.2.2:

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

mininet> help

```
tuhin@tuhin-X510UQR: ~
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> nodes

```
mininet> nodes
available nodes are:
h1 h2 s1
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=17049>
<Host h2: h2-eth0:10.0.0.2 pid=17051>
<OVSBridge s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=17056>
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::a0e6:4fff:fe7d:c219  prefixlen 64  scopeid 0x20<link>
        ether a2:e6:4f:7d:c2:19  txqueuelen 1000  (Ethernet)
        RX packets 36  bytes 4658 (4.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 11  bytes 866 (866.0 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

```
tuhin@tuhin-X510UQR: ~
mininet> s1 ifconfig -a
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 9259  bytes 2897784 (2.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9259  bytes 2897784 (2.8 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ovs-system: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether b2:82:cc:81:c4:29  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 2e:32:fd:cb:14:4b  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 19  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
tuhin@tuhin-X510UQR: ~
s1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::e80b:6bff:fe02:7ffa  prefixlen 64  scopeid 0x20<link>
        ether ea:0b:6b:02:7f:fa  txqueuelen 1000  (Ethernet)
        RX packets 11  bytes 866 (866.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 36  bytes 4658 (4.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::dc09:58ff:fed1:91a3  prefixlen 64  scopeid 0x20<link>
        ether de:09:58:d1:91:a3  txqueuelen 1000  (Ethernet)
        RX packets 11  bytes 866 (866.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 37  bytes 4744 (4.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp2s0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 60:f6:77:ee:e5:fd  txqueuelen 1000  (Ethernet)
        RX packets 308882  bytes 197901297 (197.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 313692  bytes 93904009 (93.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

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=0.638 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.092 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.088 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.100 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.096 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4077ms
rtt min/avg/max/mdev = 0.088/0.202/0.638/0.217 ms
mininet>
```

Conclusion:

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 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 . 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.