Mawlana Bhashani Science and Technology University

Lab-Report

Lab Report No: 04

Lab Report Name: SDN Controllers and Mininet.

Course Title: Network Planning and Designing Lab

Course code: ICT-3208

Submitted by

Name: Md Masud Rana

ID: IT-18002

3rd year 2nd semester

Session: 2017-18

Dept. of ICT

Submitted To

Nazrul Islam

Assistant Professor

Dept. of ICT,

MBSTU.

Theory:

Traffic Generator:

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:

```
masud@masud-VirtualBox:~$ sudo apt-get install iperf
Reading package lists... Done
Building dependency tree
Reading state information... Done
iperf is already the newest version (2.0.10+dfsg1-1ubuntu0.18.04.2).
iperf set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 480 not upgraded.
```

Install Mininet:

```
masud@masud-VirtualBox:~$ sudo apt-get install mininet
Reading package lists... Done
Building dependency tree
Reading state information... Done
mininet is already the newest version (2.2.2-2ubuntu1).
0 upgraded, 0 newly installed, 0 to remove and 480 not upgraded.
masud@masud-VirtualBox:~$ --help
```

4. Exercises Exercise

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

```
masud@masud-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 t
raffic (default not set)
  -U, --single_udp         run in single threaded UDP mode
  -D, --daemon             run the server as a daemon
```

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:

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

For terminal -2:

```
^Cmasud@masud-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 41227 connected with 127.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.44 KBytes  1.18 Kbits/sec
[  3] Sent 1 datagrams
read failed: Connection refused
[  3] WARNING: did not receive ack of last datagram after 2 tries.
masud@masud-VirtualBox:~$
```

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?

```
^Cmasud@masud-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 41227 connected with 127.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.44 KBytes  1.18 Kbits/sec
[  3] Sent 1 datagrams
read failed: Connection refused
[  3] WARNING: did not receive ack of last datagram after 2 tries.
masud@masud-VirtualBox:~$
```

```
masud@masud-VirtualBox:~$ 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

```
masud@masud-VirtualBox:~$ iperf -c 127.0.0.1 -u -l 1000 t 20 -b 1 -p 9900
iperf: ignoring extra argument -- t
iperf: ignoring extra argument -- 20
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 34067 connected with 127.0.0.1 port 9900
read failed: Connection refused
[  3] WARNING: did not receive ack of last datagram after 1 tries.
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-11.0 sec  1000 Bytes   727 bits/sec
[  3] Sent 1 datagrams
masud@masud-VirtualBox:~$
```

For terminal 2:

Iperf –s –u –p 9900

```
masud@masud-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 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?

```
masud@masud-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::1256:d071:904:f492  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:a9:5a:ab  txqueuelen 1000  (Ethernet)
        RX packets 152720  bytes 186456054 (186.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 30633  bytes 1983427 (1.9 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 2353  bytes 1933305 (1.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2353  bytes 1933305 (1.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
masud@masud-VirtualBox:~$ sudo mn
[sudo] password for masud:
*** 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>
```

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> 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=4184>
<Host h2: h2-eth0:10.0.0.2 pid=4186>
<OVSBridge s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=4191>
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::a0df:25ff:fe72:dfa5  prefixlen 64  scopeid 0x20<link>
        ether a2:df:25:72:df:a5  txqueuelen 1000  (Ethernet)
        RX packets 40  bytes 4241 (4.2 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

```
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::1256:d071:904:f492  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:a9:5a:ab  txqueuelen 1000  (Ethernet)
        RX packets 166891  bytes 202065098 (202.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 34896  bytes 2249375 (2.2 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 2394  bytes 1937164 (1.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2394  bytes 1937164 (1.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ovs-system: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether 42:dc:ab:16:4c:6e  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 06:b5:21:ae:b1:43  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::7ca6:6aff:fe3c:1064  prefixlen 64  scopeid 0x20<link>
        ether 7e:a6:6a:3c:10:64  txqueuelen 1000  (Ethernet)
        RX packets 12  bytes 936 (936.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 40  bytes 4241 (4.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::9ca2:e7ff:fee6:71e3  prefixlen 64  scopeid 0x20<link>
        ether 9e:a2:e7:e6:71:e3  txqueuelen 1000  (Ethernet)
        RX packets 12  bytes 936 (936.0 B)
        RX errors 0  dropped 0  overruns 0  frame 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=0.230 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.103 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.076 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.185 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.127 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4082ms
rtt min/avg/max/mdev = 0.076/0.144/0.230/0.056 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?

```
masud@masud-VirtualBox:~$ sudo mn --link tc,bw=10,delay=500ms
[sudo] password for masud:
*** 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 500m
s delay) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

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

```
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=2996 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=2001 ms
--------------------------------------------------------------------------------
Caught exception. Cleaning up...
```

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.