Mawlana Bhashani Science and Technology University

Lab-Report

Report No:04

Course code: ICT-3207

Course title: Computer Networks

Date of Performance:01/02/2021

Date of Submission:

Submitted by

Name: Afrin Zaman & Mahfuza Talukdar

ID:IT-18008 & IT-18009

3rd year 2nd semester

Session: 2017-2018

Dept. of ICT

Submitted To

Nazrul Islam

Assistant Professor

Dept. of ICT

MBSTU.

Theory:

Traffic Generator:

iPerf : iPerf is a tool for active measurements of the maximum achievable bandwidth on IP networks. It supports tuning of various parameters related to timing, buffers and protocols (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:

[AfrinZamanRima@webminal.org ~]$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).
The following packages were automatically installed and are no longer required:
  efibootmgr gir1.2-geocodeglib-1.0 libegl1-mesa
  libfwup1 libllvm8 libllvm9 libwayland-egl1-mesa
  linux-headers-5.0.0-23
  linux-headers-5.0.0-23-generic
  linux-image-5.0.0-23-generic
  linux-modules-5.0.0-23-generic
  linux-modules-extra-5.0.0-23-generic
  ubuntu-web-launchers
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 189 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 (2.9.5-0ubuntu0.18.04.1) ...
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)
```

Install Mininet:

[AfrinZamanRima@webminal.org ~]$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).
The following packages were automatically installed and are no longer required:
  efibootmgr gir1.2-geocodeglib-1.0 libegl1-mesa libfwup1 libllvm8 libllvm9 libwayland-egl1-mesa linux-headers-5.0.0-23
  linux-headers-5.0.0-23-generic linux-image-5.0.0-23-generic linux-modules-5.0.0-23-generic linux-modules-extra-5.0.0-23-generic
  ubuntu-web-launchers
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 189 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 (2.9.5-0ubuntu0.18.04.1) ...
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)
```

4. Exercises Exercise

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

[AfrinZamanRima@webminal.org ~]$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
```

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:

```
[AfrinZamanRima@webminal.org ~]$iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size:  128 KByte (default)
------------------------------------------------------------
```

For terminal -2:

```
[AfrinZamanRima@webminal.org ~]$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 32935 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.
```

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?

```
[AfrinZamanRima@webminal.org ~]$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 32935 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.
```

[AfrinZamanRima@webminal.org ~]$perf -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

[AfrinZamanRima@webminal.org ~]$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 44882 connected with 127.0.0.1 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.
```

For terminal 2:

Iperf –s –u –p 9900

[AfrinZamanRima@webminal.org ~]$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?

[AfrinZamanRima@webminal.org ~]$ifconfig

```
eno1: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 48:ba:4e:5a:67:dd  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 5199  bytes 3894630 (3.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5199  bytes 3894630 (3.8 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlo1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.43.34  netmask 255.255.255.0  broadcast 192.168.43.255
        inet6 fe80::c4e8:894:a905:442d  prefixlen 64  scopeid 0x20<link>
        ether 28:c6:3f:25:b7:19  txqueuelen 1000  (Ethernet)
        RX packets 27187  bytes 30799056 (30.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 19119  bytes 2661495 (2.6 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

[AfrinZamanRima@webminal.org ~]$sudo mn

```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

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

mininet> help

[AfrinZamanRima@webminal.org ~]$ifconfig

```
eno1: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 48:ba:4e:5a:67:dd  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 5199  bytes 3894630 (3.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5199  bytes 3894630 (3.8 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlo1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.43.34  netmask 255.255.255.0  broadcast 192.168.43.255
        inet6 fe80::c4e8:894:a905:442d  prefixlen 64  scopeid 0x20<link>
        ether 28:c6:3f:25:b7:19  txqueuelen 1000  (Ethernet)
        RX packets 27187  bytes 30799056 (30.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 19119  bytes 2661495 (2.6 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
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:
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=3785>
<Host h2: h2-eth0:10.0.0.2 pid=3787>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=3792>
<Controller c0: 127.0.0.1:6653 pid=3778>
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::585d:ccff:fece:d1c9  prefixlen 64  scopeid 0x20<link>
        ether 5a:5d:cc:ce:d1:c9  txqueuelen 1000  (Ethernet)
        RX packets 32  bytes 3631 (3.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 796 (796.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

mininet>
```

mininet> s1 ifconfig –a

```
mininet> s1 ifconfig -a
eno1: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 48:ba:4e:5a:67:dd  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 5782  bytes 3942989 (3.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5782  bytes 3942989 (3.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ovs-system: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether 3e:ed:da:9e:8f: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 ee:95:6c:3e:fc:42  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 31  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
Termi
File  Edit  View  Search  Terminal  Help
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether ee:95:6c:3e:fc:42  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 31  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::4476:7aff:feb4:2d72  prefixlen 64  scopeid 0x20<link>
        ether 46:76:7a:b4:2d:72  txqueuelen 1000  (Ethernet)
        RX packets 11  bytes 866 (866.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 34  bytes 3808 (3.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::580f:c5ff:fe98:26e  prefixlen 64  scopeid 0x20<link>
        ether 5a:0f:c5:98:02:6e  txqueuelen 1000  (Ethernet)
        RX packets 11  bytes 866 (866.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 35  bytes 3878 (3.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlo1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.43.34  netmask 255.255.255.0  broadcast 192.168.43.255
        inet6 fe80::c4e8:894:a905:442d  prefixlen 64  scopeid 0x20<link>
        ether 28:c6:3f:25:b7:19  txqueuelen 1000  (Ethernet)
        RX packets 27383  bytes 30834178 (30.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 19386  bytes 2702033 (2.7 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=2 ttl=64 time=3022 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=2001 ms
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=4053 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=2000 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=2000 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4073ms
rtt min/avg/max/mdev = 2000.272/2615.562/4053.501/820.755 ms, pipe 4
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?

```
[AfrinZamanRima@webminal.org ~]$sudo mn –-link tc,bw=10,delay=500ms
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(10.00Mbit 500ms delay) (10.00Mbit 500ms delay) (h1, s1) (10.00Mbit 500ms delay) (10.00Mbit 500ms delay) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...(10.00Mbit 500ms delay) (10.00Mbit 500ms delay)
*** Starting CLI:
```

```
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=2 ttl=64 time=3022 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=2001 ms
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=4053 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=2000 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=2000 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4073ms
rtt min/avg/max/mdev = 2000.272/2615.562/4053.501/820.755 ms, pipe 4
mininet>
```

Discussion :

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 supports research, development, learning, prototyping, testing, debugging, and any other tasks that could benefit from having a complete experimental network on a laptop or other PC.

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 the extent supported by your hardware, performance) and to experiment with topologies.