

## Assignment 1

Shoaib Alam – 21250024

### Experimenting with IP, MAC addresses:

1. Add a secondary IP address to a chosen interface on your device. If your primary interface shows IP address as 192.168.0.1 then add another IP address 10.0.0.1 with subnet mask 255.255.255.0 to the same interface. Verify that your device is operating and responding to the configured IP address using the “ping” command. Note: when the device is not set with secondary IP, the ping for the secondary IP would fail

```
Mininet-VM 1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

RX packets 62 bytes 8272 (8.2 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 8 bytes 1872 (1.8 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
loop txqueuelen 1000 (Local Loopback)
RX packets 718 bytes 55824 (55.8 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 718 bytes 55824 (55.8 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

mininet@mininet-vm:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:45:3b:c7 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.101/24 brd 192.168.56.255 scope global dynamic eth0
        valid_lft 504sec preferred_lft 504sec
    inet 192.168.56.105/24 brd 192.168.56.255 scope global secondary eth0
        valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 08:00:27:32:7d:f0 brd ff:ff:ff:ff:ff:ff

mininet@mininet-vm:~$ ping 192.168.56.105
PING 192.168.56.105 (192.168.56.105) 56(84) bytes of data:
64 bytes from 192.168.56.105: icmp_seq=1 ttl=64 time=0.039 ms
64 bytes from 192.168.56.105: icmp_seq=2 ttl=64 time=0.052 ms
64 bytes from 192.168.56.105: icmp_seq=3 ttl=64 time=0.053 ms
64 bytes from 192.168.56.105: icmp_seq=4 ttl=64 time=0.048 ms
^C
--- 192.168.56.105 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3031ms
rtt min/avg/max/mdev = 0.039/0.048/0.053/0.005 ms
mininet@mininet-vm:~$ _
```

Primary IP Address  
192.168.56.101

Secondary IP Address  
192.168.56.105

Pinging Secondary IP  
192.168.56.105

Ping Successful, 0% packet loss.

Figure 1. IP Aliasing, Adding Secondary IP Address and ping to check it's working or not.

### Commands Used:

- To create Secondary IP address

```
$sudo ip addr add 192.168.56.105/24 broadcast 192.168.56.255 dev eth0
```

- To check if new configured IP is working correctly or not

```
$ping 192.168.56.105
```

### Analysis:

- This experiment shows that there can be multiple IP's of a interface and both IP can work together.

### Tool:

- Mininet 2.3.0

- For an existing ethernet device, create an alias network interface (multiple L2 interfaces) i.e. You will be able to see multiple network interfaces, but all will have same MAC address. You can verify the result by running the ifconfig command.

eth0:  
08:00:27:45:3b:f7

eth0:0:  
08:00:27:45:3b:f7

Figure 2. MAC aliasing, adding another interface using the same mac address as that of available one.

#### Commands Used:

- To create multiple interface using the same MAC address

```
$sudo ifconfig eth0:0 192.168.56.110 up
```

- To check if new configured IP is working correctly or not

```
$ifconfig
```

#### Analysis:

- This experiment shows that there can be multiple interfaces having the same mac address and mac address need not be unique for all the interface.

#### Tool:

- Mininet 2.3.0

## Concurrent Servers: Fork vs Threads (TCP)

3. Implement a simple File server that can handle multiple requests concurrently. You have the following two models to build a concurrent server.

### Single Connection:

#### Fork Model

Server File Name: *server\_fork.py*

Client File Name: *client.py*

Starting only one server and one client.

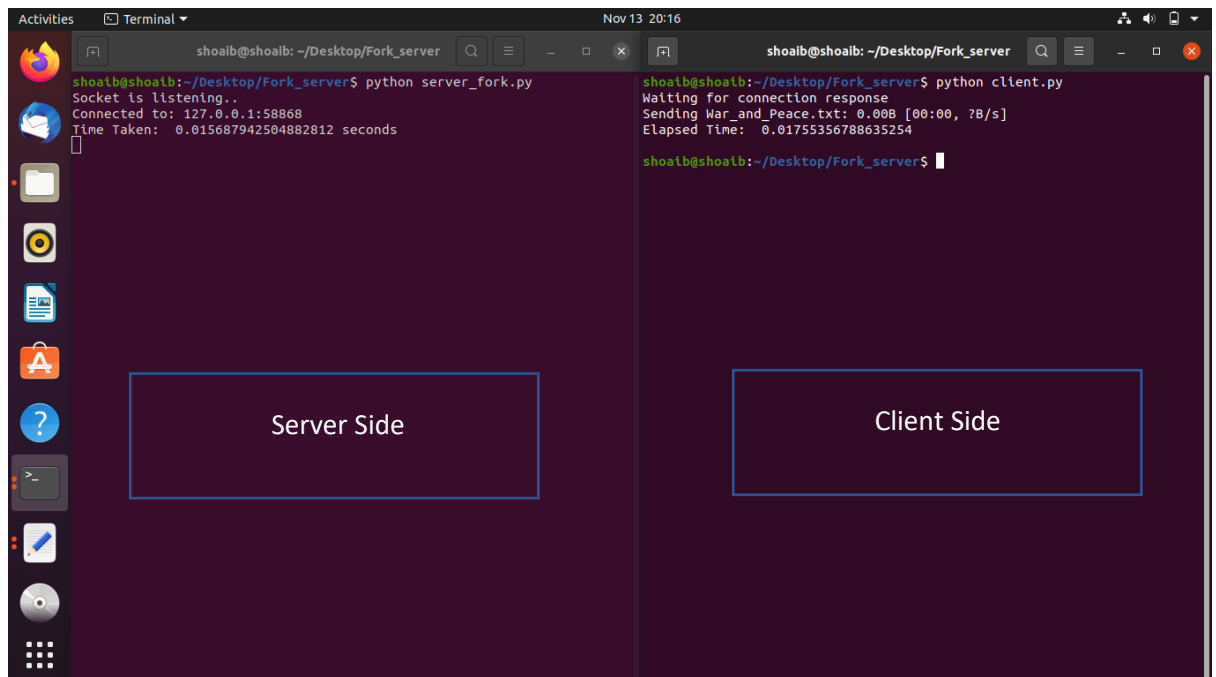


Figure 3. Concurrent Server implementation using Forks. Each new client will be serve as independent process using fork.

#### Program Description

- File Name: War\_and\_Peace.txt
- File Size: 3.20MB
- Time Taken: 0.017 seconds
- Throughput: 188.23MB/s

#### Tools Used:

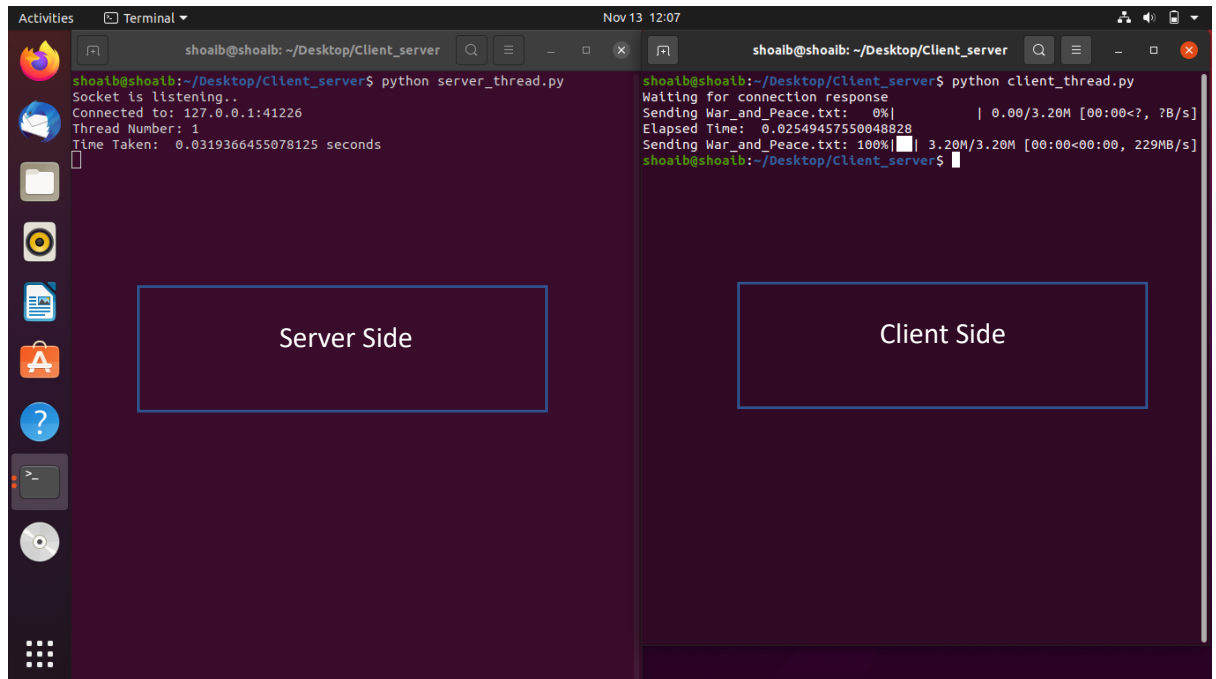
- Python3 (programming language)

## Thread Model

Server File Name: *server\_thread.py*

Client File Name: *client\_thread.py*

Starting the Server and only one client. Client wants to download a file name *War\_and\_Peace.txt* from server.



The image shows two terminal windows side-by-side. The left window, titled 'Server Side', shows the execution of `python server_thread.py`. It displays 'Socket is listening..', 'Connected to: 127.0.0.1:41226', 'Thread Number: 1', and 'Time Taken: 0.0319366455078125 seconds'. The right window, titled 'Client Side', shows the execution of `python client_thread.py`. It displays 'Waiting for connection response', 'Sending War\_and\_Peace.txt: 0%| 0.00/3.20M [00:00<?, ?B/s]', 'Elapsed Time: 0.02549457550048828', and 'Sending War\_and\_Peace.txt: 100%| 3.20M/3.20M [00:00<00:00, 229MB/s]'. Both windows have a dark background with light text.

Figure 4. Implementation of concurrent server using threads. Each new client will be assigned new thread and parent thread will handle all the new requests.

## Program Description

- File Name: *War\_and\_Peace.txt*
- File Size: 3.20MB
- Time Taken: 0.025 seconds
- Throughput: 128MB/s

## Tools Used:

- Python3 (programming language)

## Multiple Connections:

In order to initiate multiple clients I have used a script file which allows each clients to start and send them in background to let run other clients. And that's how I tried to start multiple clients work simultaneously.

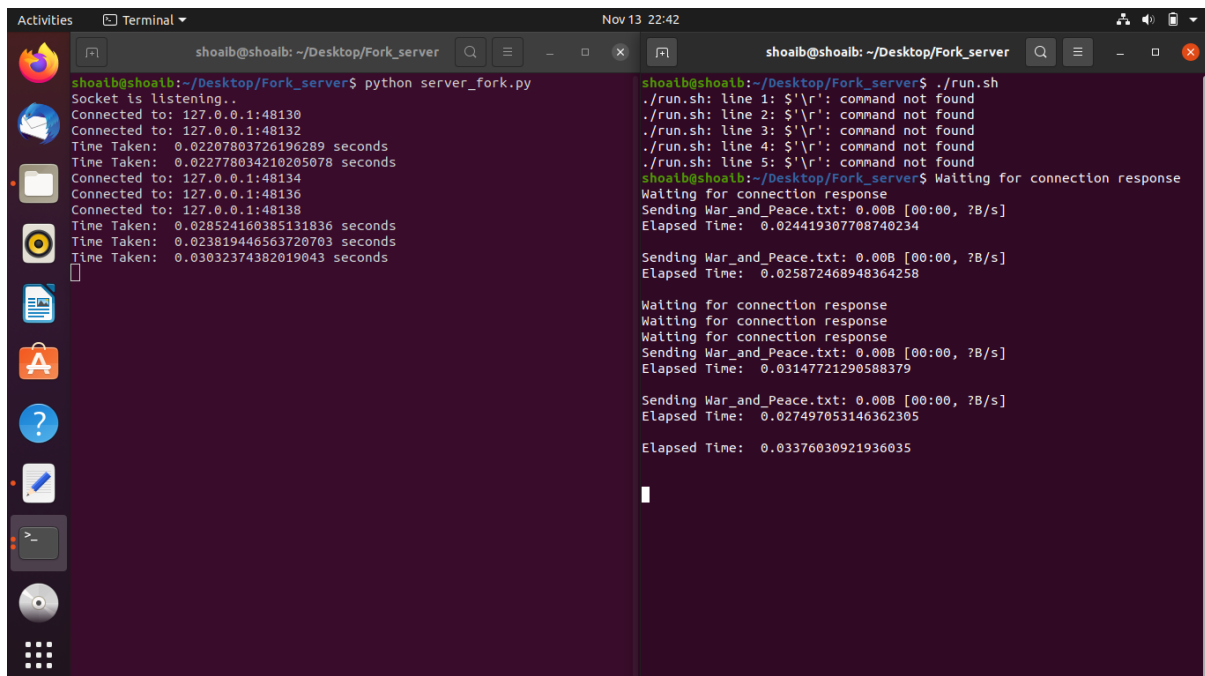
Name of Script File: *run.sh*

## **Fork Model:**

Server File Name: *server\_fork.py*

Client File Name: *client.py*

Starting only one server and multiple clients.



```
shoalb@shoalb: ~/Desktop/Fork_server
shoalb@shoalb:~/Desktop/Fork_server$ python server_fork.py
Socket is listening..
Connected to: 127.0.0.1:48130
Connected to: 127.0.0.1:48132
Time Taken: 0.02207803726196289 seconds
Time Taken: 0.022778034210205078 seconds
Connected to: 127.0.0.1:48134
Connected to: 127.0.0.1:48136
Connected to: 127.0.0.1:48138
Time Taken: 0.028524160395131836 seconds
Time Taken: 0.023819446563728793 seconds
Time Taken: 0.03032374382019043 seconds

shoalb@shoalb:~/Desktop/Fork_server$ ./run.sh
./run.sh: line 1: '$\r': command not found
./run.sh: line 2: '$\r': command not found
./run.sh: line 3: '$\r': command not found
./run.sh: line 4: '$\r': command not found
./run.sh: line 5: '$\r': command not found
shoalb@shoalb:~/Desktop/Fork_server$ Waiting for connection response
Waiting for connection response
Sending War_and_Peace.txt: 0.00B [00:00, ?B/s]
Elapsed Time: 0.024419307708740234

Sending War_and_Peace.txt: 0.00B [00:00, ?B/s]
Elapsed Time: 0.025872468948364258

Waiting for connection response
Waiting for connection response
Waiting for connection response
Sending War_and_Peace.txt: 0.00B [00:00, ?B/s]
Elapsed Time: 0.03147721290588379

Sending War_and_Peace.txt: 0.00B [00:00, ?B/s]
Elapsed Time: 0.027497053146362305

Elapsed Time: 0.03376030921936035
```

Figure 5. Starting many clients together and sending them in background so that all the clients can start simultaneously. This is an example of handling concurrent client server model using fork. On the left it's server who is receiving the requests and on right are the multiple clients who are trying to access the same file i.e., War\_and\_Peace.txt, Size 3.5MB).

## **Program Description**

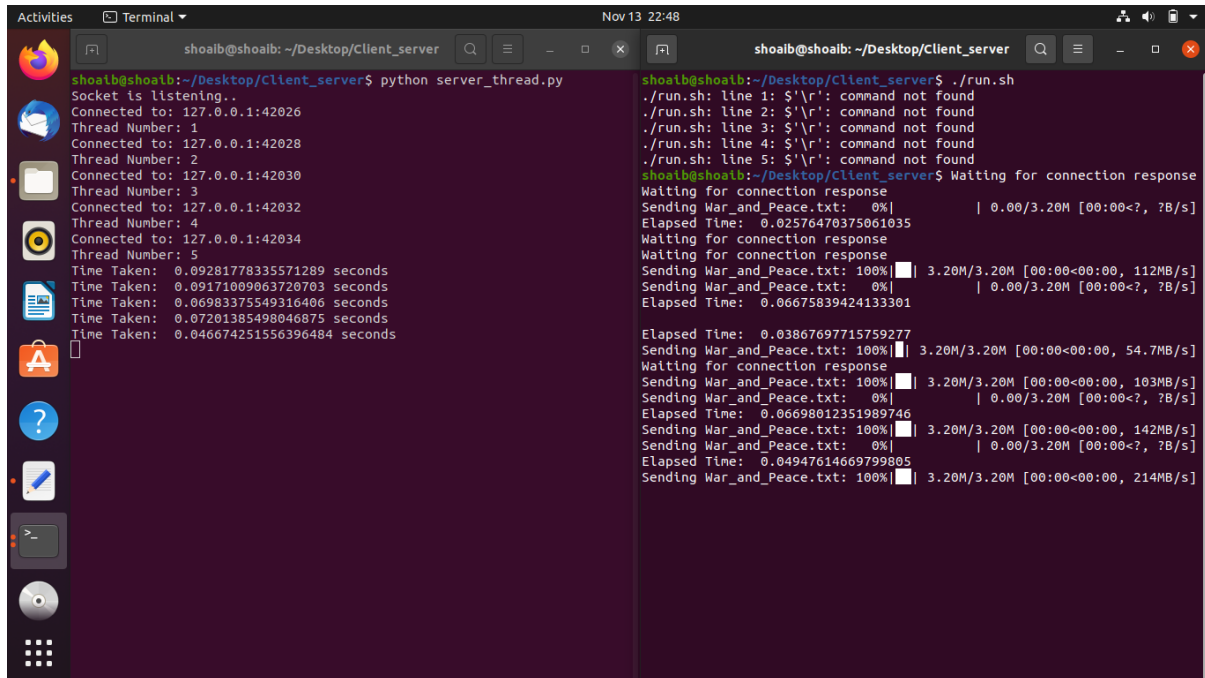
- File Name: War\_and\_Peace.txt
- File Size: 3.20MB
- Avg. Time Taken: 0.02546 seconds
- Avg. Throughput: 137.47MB/s

## Thread Model:

Server File Name: *server\_thread.py*

Client File Name: *client\_thread.py*

Starting only one server and multiple clients.



```
shoalb@shoalb: ~/Desktop/Client_server$ python server_thread.py
Socket is listening..
Connected to: 127.0.0.1:42026
Thread Number: 1
Connected to: 127.0.0.1:42028
Thread Number: 2
Connected to: 127.0.0.1:42030
Thread Number: 3
Connected to: 127.0.0.1:42032
Thread Number: 4
Connected to: 127.0.0.1:42034
Thread Number: 5
Time Taken: 0.09281778335571289 seconds
Time Taken: 0.09171009063720703 seconds
Time Taken: 0.06983375549316406 seconds
Time Taken: 0.07201385498046875 seconds
Time Taken: 0.046674251556396484 seconds

shoalb@shoalb: ~/Desktop/Client_server$ ./run.sh
./run.sh: line 1: $'\r': command not found
./run.sh: line 2: $'\r': command not found
./run.sh: line 3: $'\r': command not found
./run.sh: line 4: $'\r': command not found
./run.sh: line 5: $'\r': command not found
shoalb@shoalb: ~/Desktop/Client_server$ Waiting for connection response
Waiting for connection response
Sending War_and_Peace.txt: 0%| 0.00/3.20M [00:00<?, ?B/s]
Elapsed Time: 0.02576470375061035
Waiting for connection response
Waiting for connection response
Sending War_and_Peace.txt: 100%| 3.20M/3.20M [00:00<00:00, 112MB/s]
Sending War_and_Peace.txt: 0%| 0.00/3.20M [00:00<?, ?B/s]
Elapsed Time: 0.06675839424133301
Elapsed Time: 0.03867697715759277
Sending War_and_Peace.txt: 100%| 3.20M/3.20M [00:00<00:00, 54.7MB/s]
Waiting for connection response
Sending War_and_Peace.txt: 100%| 3.20M/3.20M [00:00<00:00, 103MB/s]
Sending War_and_Peace.txt: 0%| 0.00/3.20M [00:00<?, ?B/s]
Elapsed Time: 0.06698012351989746
Sending War_and_Peace.txt: 100%| 3.20M/3.20M [00:00<00:00, 142MB/s]
Sending War_and_Peace.txt: 0%| 0.00/3.20M [00:00<?, ?B/s]
Elapsed Time: 0.04947614669799805
Sending War_and_Peace.txt: 100%| 3.20M/3.20M [00:00<00:00, 214MB/s]
```

Figure 6. . Starting many clients together and sending them in background so that all the clients can start simultaneously. This is an example of handling concurrent client server model using Threads. On the left it's server who is receiving the requests and on right are the multiple clients who are trying to access the same file i.e., War\_and\_Peace.txt, Size 3.5MB.

## Program Description

- File Name: War\_and\_Peace.txt
- File Size: 3.20MB
- Avg. Time Taken: 0.0742 seconds
- Avg. Throughput: 47.169MB/s

## Analysis

After running both the files for multiple times I found that fork Method works best for this use-case. The other difference that arise due to fork and threading is in threads all share the common memory space so variables can easily be shared among all its threads which is not there in case of forks.

Forking is usually is faster in single core CPU's than threading as there are no overheads of context switching. But, fork comes up with its major drawback that it has it's own memory and address space which makes it difficult to coordinate with other processes.

Threading is good it's initialization is usually faster as it don't requires any separate spaces for memory and address. Its most effective with multiprocessor systems where each process can run across several processors and hence gives faster processing. But, the major drawbacks of threads are there are chances of race conditions.

## Experimenting with Mininet. (TCP)

Implement a Mininet network, where you instantiate a desired (parameterized) topology, where one of the host node acts as server and subset of the remaining nodes act as clients. First, detail precisely the changes you need to make for the client and server to operate with Mininet.

Step 1: Creating single topology with 6 nodes.

```
$ sudo mn --topo single,6
```

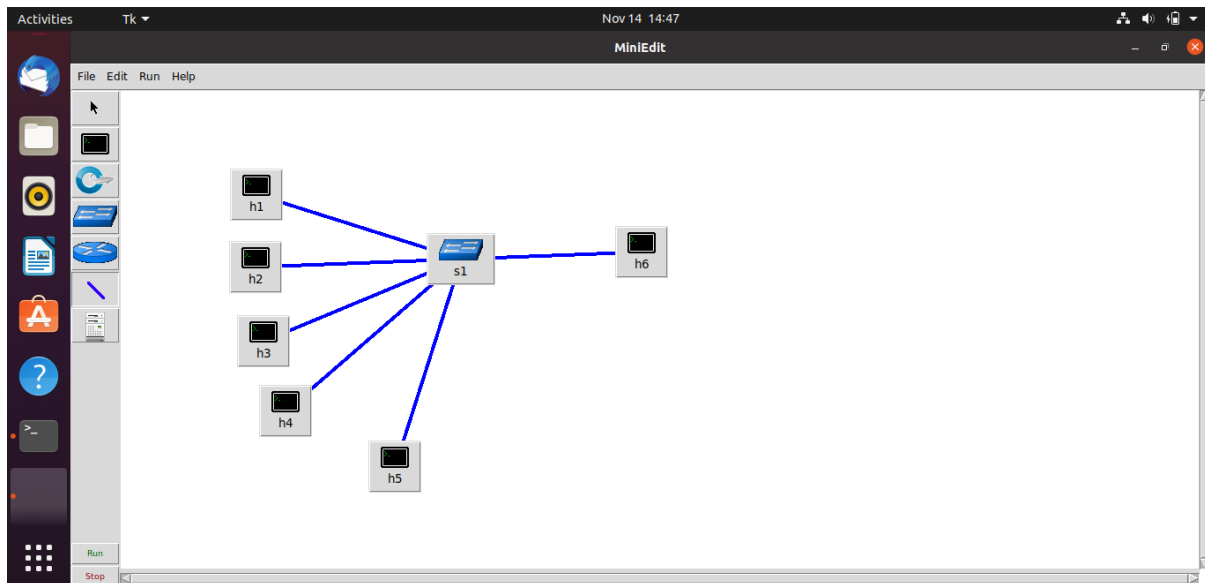


Figure 7. Create a single topology of 6 hosts. This is for visualization how our topology looks like. It is made using miniedit a simple light weight GUI based tool provided by mininet.

Step 2: Open xterm terminal for each node

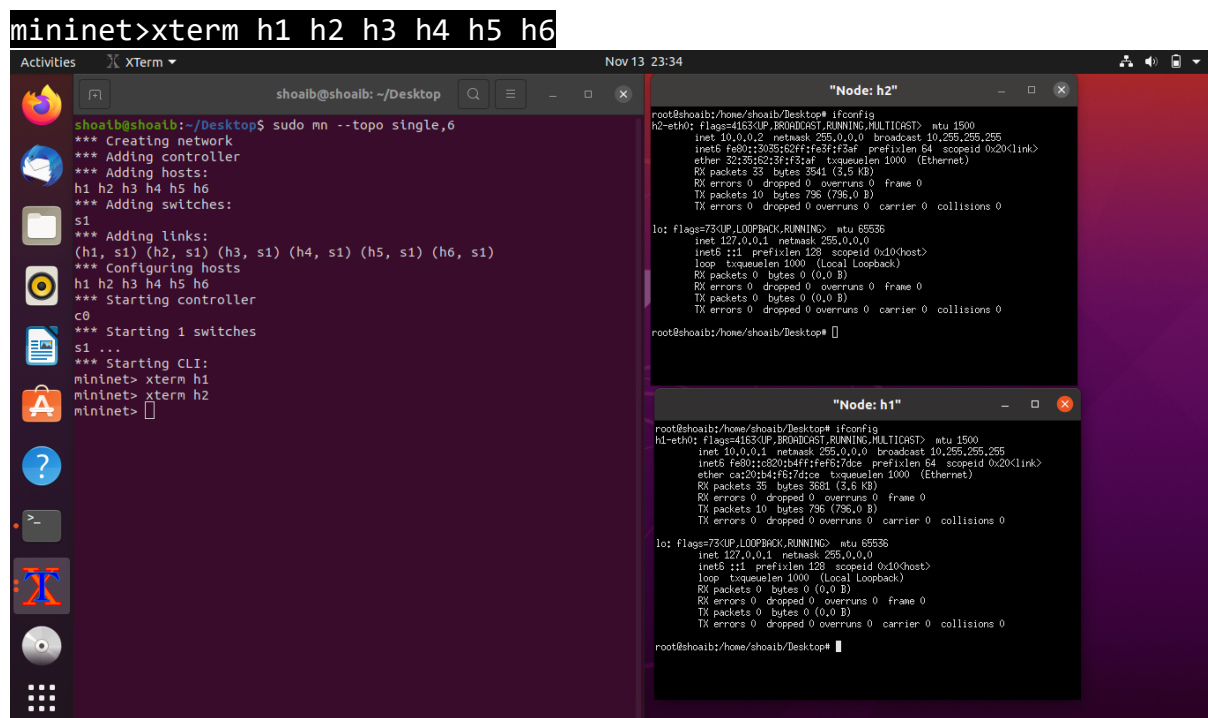
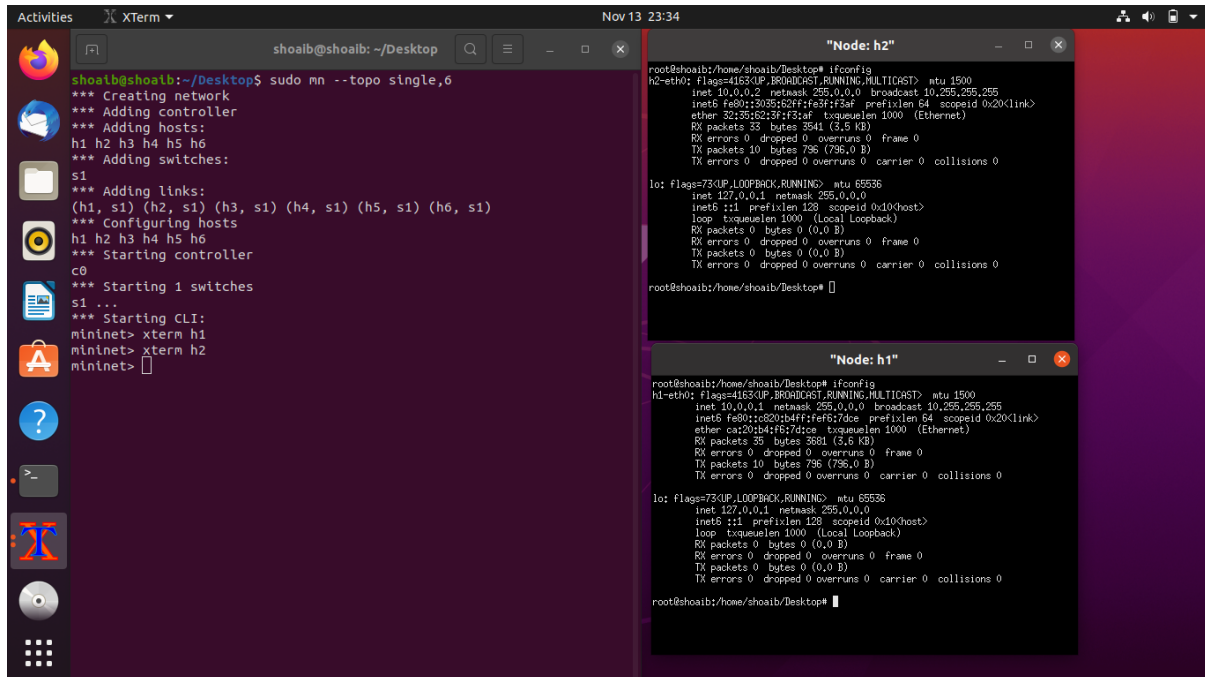


Figure 8. Use xterm command to open xterm terminal for each nodes separately. From here we can control each nodes and can configure them according to our needs.



### Step 3: Check IP address of each nodes

#### #ifconfig



```
shoalb@shoalb: ~/Desktop
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1) (h6, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> xterm h1
mininet> xterm h2
mininet>

"Node: h2"
root@shoalb:/home/shoalb/Desktop# ifconfig
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.2 netmask 255.0.0.0 broadcast 10.255.255.255
    inet6 fe80::3035:62ff:fe3f:f3af prefixlen 64 scopeid 0x20<link>
    ether 32:35:62:f3:f3:af txqueuelen 1000 (Ethernet)
    RX packets 33 bytes 3541 (3.5 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 10 bytes 736 (736.0 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

root@shoalb:/home/shoalb/Desktop#

"Node: h1"
root@shoalb:/home/shoalb/Desktop# ifconfig
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::c20:b4ff:fe6f:7dce prefixlen 64 scopeid 0x20<link>
    ether ca:20:b4:f6:7d:ce txqueuelen 1000 (Ethernet)
    RX packets 35 bytes 3681 (3.6 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 10 bytes 736 (736.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

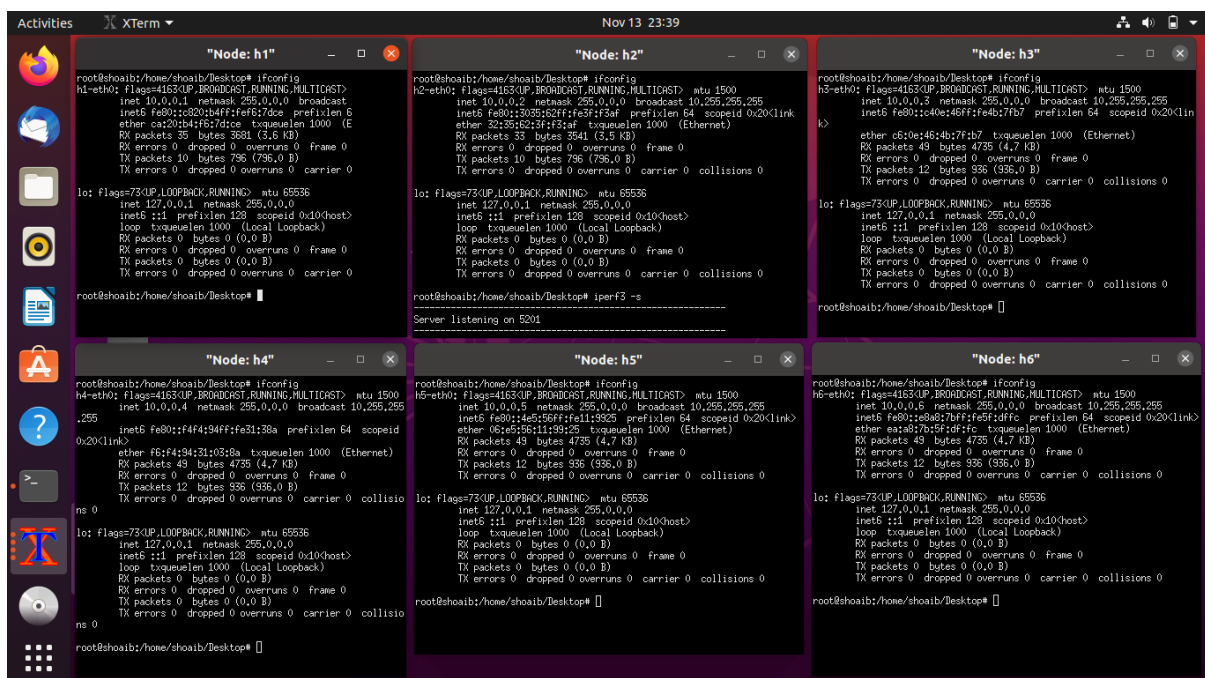
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped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

root@shoalb:/home/shoalb/Desktop#
```

Figure 9. Once created open xterm terminal for each of the hosts. You can also check the IP's of each host using ifconfig that will help us to establish the connection.

### Step 4: Start iPerf3 in server mode in host h2 using

#### #iperf3 -s



```
"Node: h1"
root@shoalb:/home/shoalb/Desktop# ifconfig
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::c20:b4ff:fe6f:7dce prefixlen 64 scopeid 0x20<link>
    ether ca:20:b4:f6:7d:ce txqueuelen 1000 (Ethernet)
    RX packets 35 bytes 3681 (3.6 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 10 bytes 736 (736.0 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

root@shoalb:/home/shoalb/Desktop#

"Node: h2"
root@shoalb:/home/shoalb/Desktop# ifconfig
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.2 netmask 255.0.0.0 broadcast 10.255.255.255
    inet6 fe80::3035:62ff:fe3f:f3af prefixlen 64 scopeid 0x20<link>
    ether 32:35:62:f3:f3:af txqueuelen 1000 (Ethernet)
    RX packets 33 bytes 3541 (3.5 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 10 bytes 736 (736.0 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

root@shoalb:/home/shoalb/Desktop# iperf3 -s
Server listening on 5201

"Node: h3"
root@shoalb:/home/shoalb/Desktop# ifconfig
h3-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.3 netmask 255.0.0.0 broadcast 10.255.255.255
    inet6 fe80::1c40e:46ff:fe4b:7b77 prefixlen 64 scopeid 0x20<link>
    ether c6:0e:46:b7:b7:77 txqueuelen 1000 (Ethernet)
    RX packets 49 bytes 4735 (4.7 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 12 bytes 336 (336.0 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

root@shoalb:/home/shoalb/Desktop#

"Node: h4"
root@shoalb:/home/shoalb/Desktop# ifconfig
h4-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.4 netmask 255.0.0.0 broadcast 10.255.255.255
    inet6 fe80::f4f4:94ff:fe31:138a prefixlen 64 scopeid 0x20<link>
    ether f6:f4:94:31:03:8a txqueuelen 1000 (Ethernet)
    RX packets 49 bytes 4735 (4.7 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 12 bytes 336 (336.0 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

root@shoalb:/home/shoalb/Desktop#

"Node: h5"
root@shoalb:/home/shoalb/Desktop# ifconfig
h5-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.5 netmask 255.0.0.0 broadcast 10.255.255.255
    inet6 fe80::4465:56ff:fe11:1325 prefixlen 64 scopeid 0x20<link>
    ether 64:45:56:11:13:25 txqueuelen 1000 (Ethernet)
    RX packets 49 bytes 4735 (4.7 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 12 bytes 336 (336.0 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

root@shoalb:/home/shoalb/Desktop#

"Node: h6"
root@shoalb:/home/shoalb/Desktop# ifconfig
h6-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.6 netmask 255.0.0.0 broadcast 10.255.255.255
    inet6 fe80::e8a8:70ff:fe5f:dfdc prefixlen 64 scopeid 0x20<link>
    ether ea:48:70:5f:df:dc txqueuelen 1000 (Ethernet)
    RX packets 49 bytes 4735 (4.7 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 12 bytes 336 (336.0 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

root@shoalb:/home/shoalb/Desktop#
```

Figure 10. Using iperf3 we are testing the connection between each of the hosts. In this case "Node:h2" is acting as server and rest other nodes are working like clients.



Step 5: Now start iPerf in rest of the nodes in client mode

```
#iperf3 -c 10.0.0.2
```

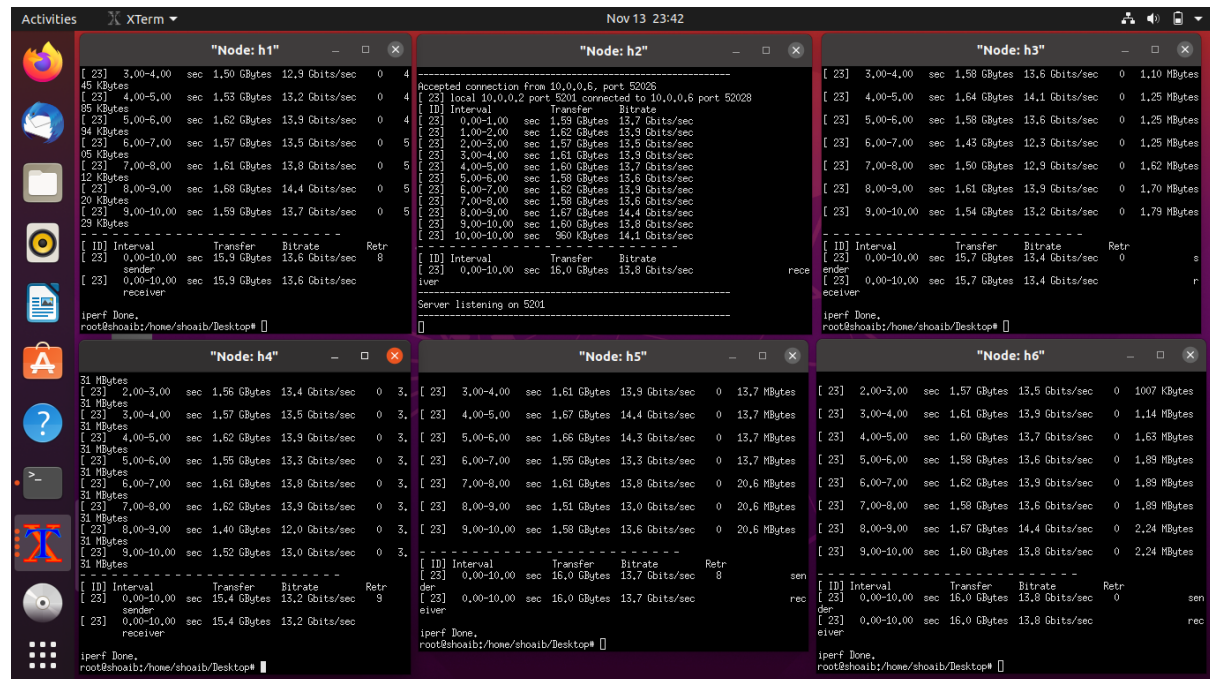


Figure 11. Each of the Nodes has their data and the summarized information are there on their respective terminals, i.e., Transfer Rates, Bitrates, cnwd, etc.

Step 6: Once done you can see the summarized data in both server and client terminals.

Result: This experiments shows that a proper connection between all the nodes has been established and, a server and all the clients are able to communicate each other. The Objective of this experiment has been achieved.

(a) Single topology (i): Make use of the single topology of the mininet that allows you to instantiate 1 switch and 6 host nodes. Designate one of the host nodes to run a server program that you implemented and the remaining 5 nodes to run the client program. All the Clients connect with the server concurrently and request to download the same file. Measure the download time and throughput for each of these clients. What do you observe?

Step 1: Creating single topology with 6 nodes.

```
$ sudo mn --topo single,6
```

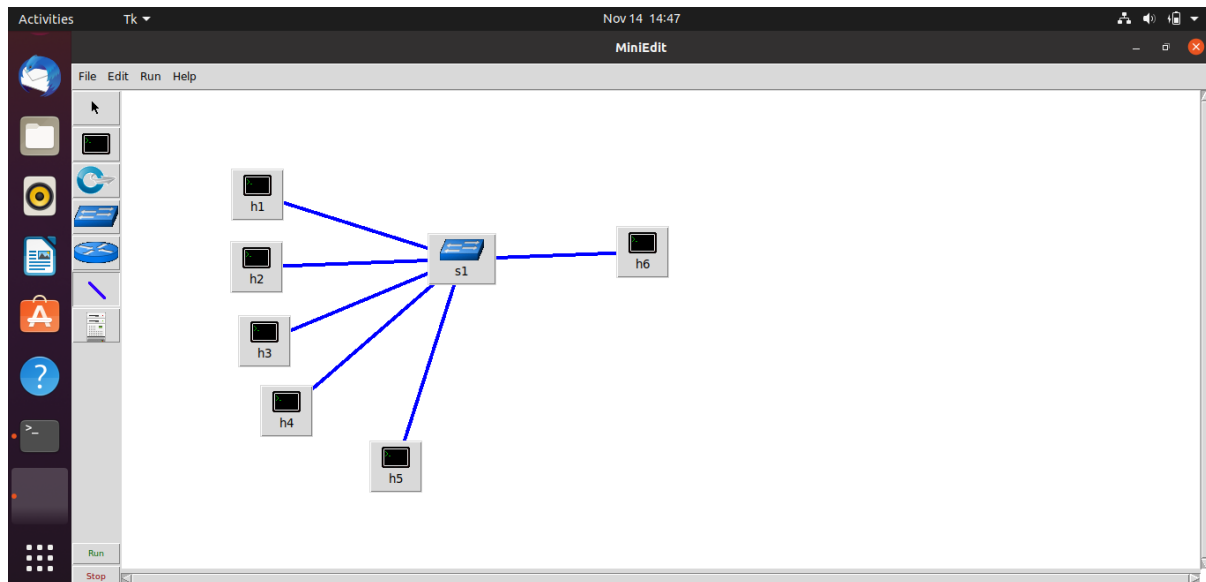


Figure 12. . Create a single topology of 6 hosts. This is for visualization how our topology looks like. It is made using miniedit a simple light weight GUI based tool provided by mininet.

Step 2: Open xterm terminal for each of the Nodes.

```
mininet>xterm h1 h2 h3 h4 h5 h6
```

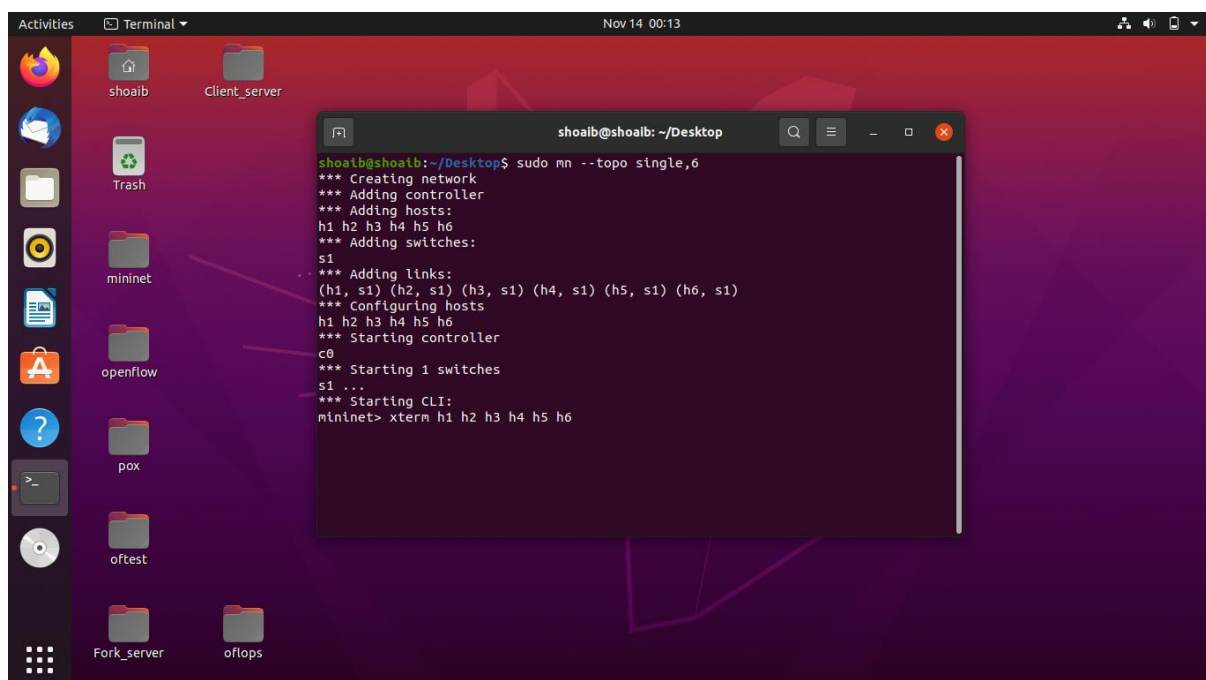


Figure 13. Use xterm command to open xterm terminal for each nodes separately. From here we can control each nodes and can configure them according to our needs

Step 3: Start iPerf3 in server mode in host h6 using

```
#iperf3 -s
```

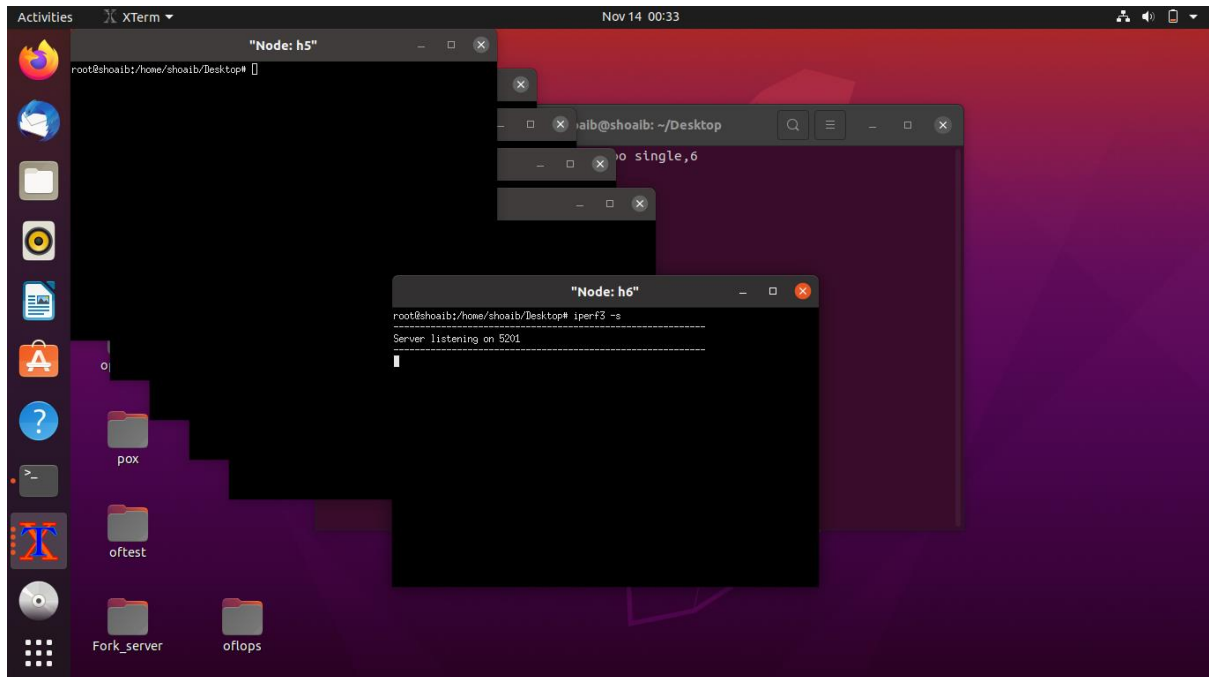


Figure 14. Using iPerf3 we are testing the connection between each of the hosts. In this case "Node:h6" is acting as server and rest other nodes are working like clients.

Since we have to transfer a file of particular size as we did earlier in Q2 so I have decided to use a transfer the same file size using iPerf3 so that It will be easy for calculations.

Step 4: Make all the rest nodes as client. We will use -n option to specify the file size.

```
#iperf3 -c 10.0.0.6 -n 3.5M
```

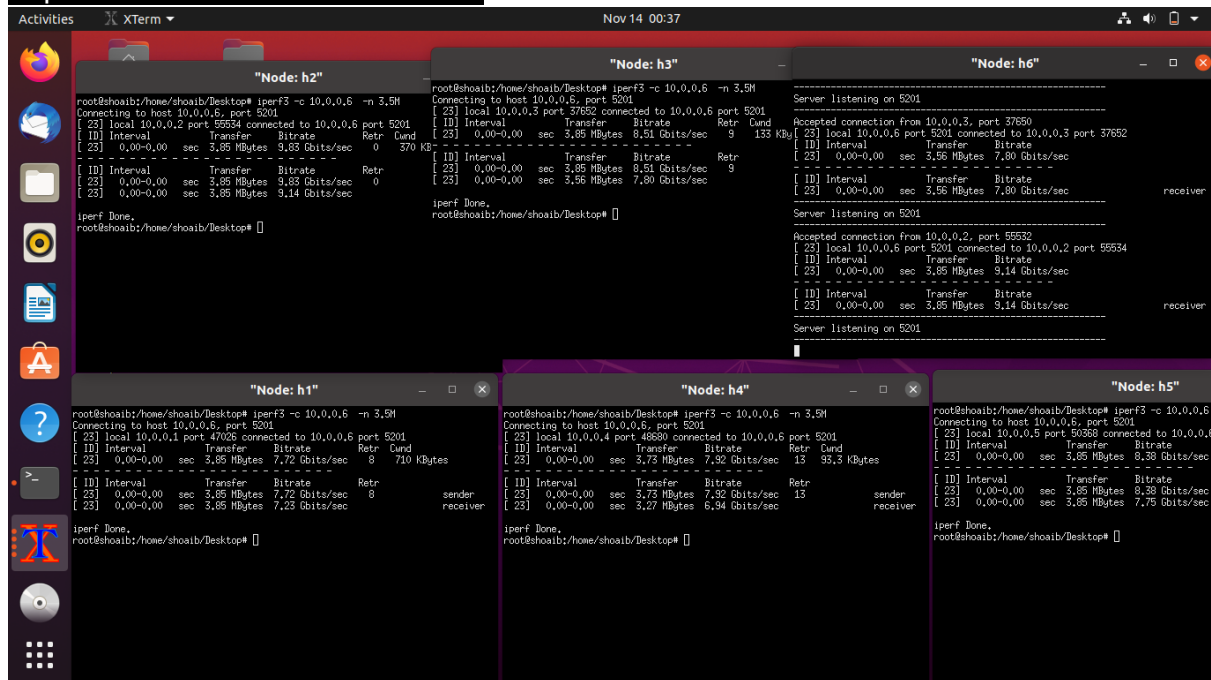


Figure 15. As we want to compare the mininet model and non-mininet model, we have send the same size of file across the different clients. So, we have used -n option of iperf3 which allows us to specify the file size that we want to send.

Step 5: Once done you can see the summarized data in both server and client terminals.

Analysis:

File Transfer Rate and Bitrate are mention on each terminal separately. Now, on comparing it with non-mininet configuration, the mininet system performs much better in in terms of speed and throughput.

In order to establish the connection between clients and server we will be needing the PuTTY that help us to connect between remote clients and servers.

Install PuTTY and enter the machine IP address you want to connect in session login. In our case it is the IP address of mininet system.

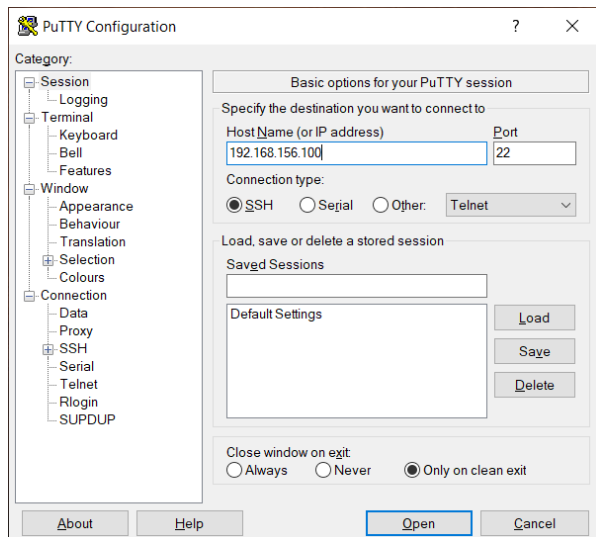


Figure 16. PuTTY Configuration window. Enter Host IP in order to connect it with particular host.

Also you need to transfer files to mininet in order to execute them. To do so we will be using Windows SCP for this purpose.

To establish connection between mininet and SCP you will be needing IP address of mininet and its username and password. Once connected transfer all the files you need to execute the client server model.

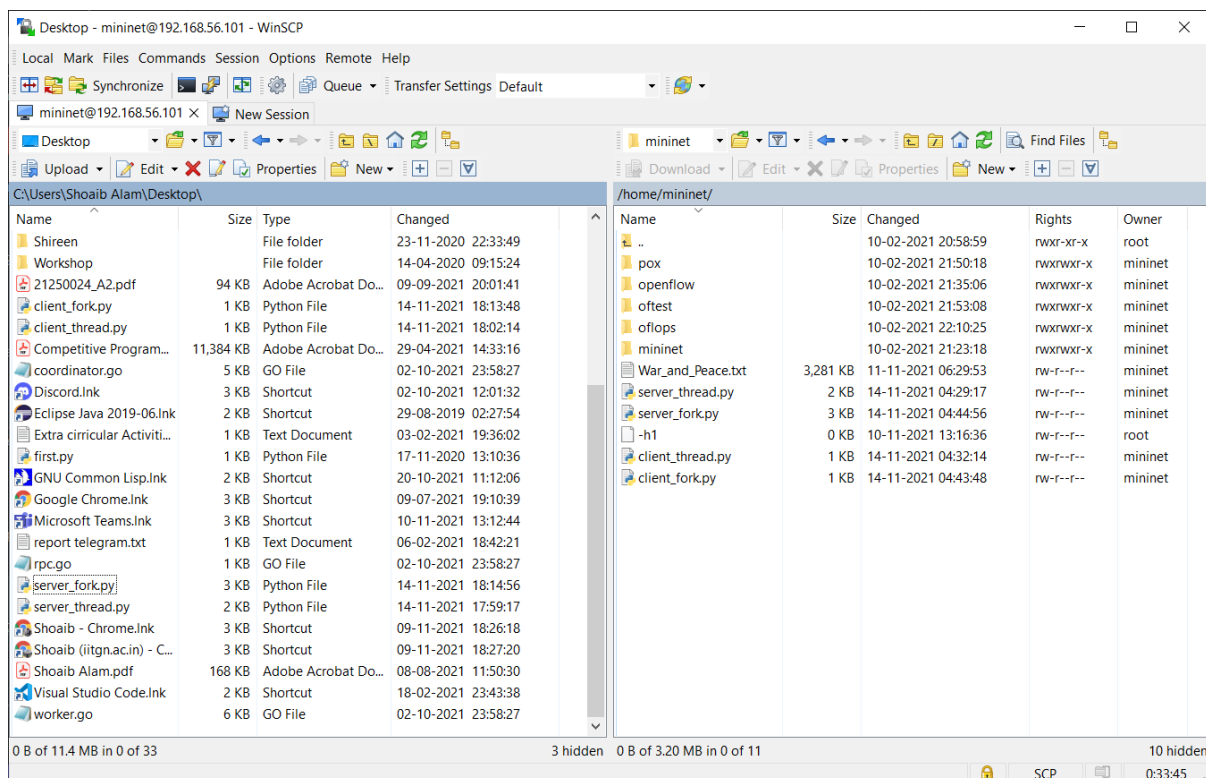
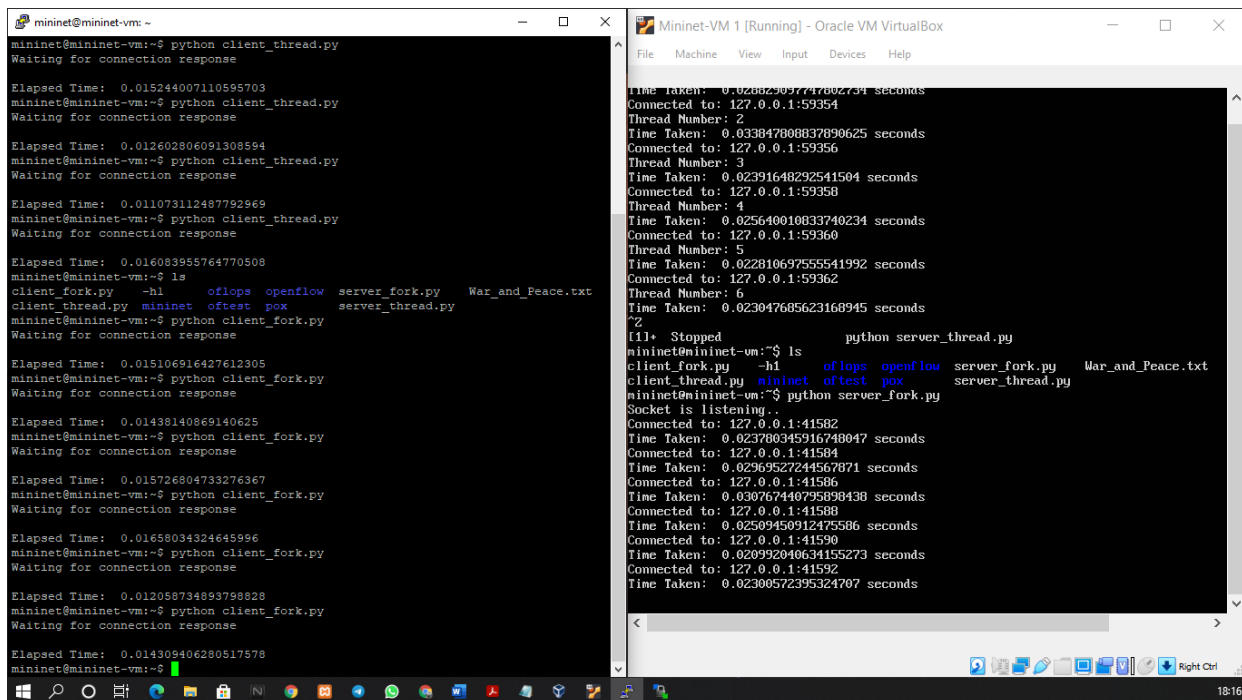


Figure 17. WinSCP Program. Use this to transfer the files to mininet system.

Now start the clients and server as you did previously.

## Fork Results:



```
mininet@mininet-vm:~$ python client_thread.py
Waiting for connection response

Elapsed Time: 0.015244007110595703
mininet@mininet-vm:~$ python client_thread.py
Waiting for connection response

Elapsed Time: 0.012602806091308594
mininet@mininet-vm:~$ python client_thread.py
Waiting for connection response

Elapsed Time: 0.011073112487792969
mininet@mininet-vm:~$ python client_thread.py
Waiting for connection response

Elapsed Time: 0.016083955764770508
mininet@mininet-vm:~$ ls
client_thread.py  -hl  oflops  openflow  server_fork.py  War_and_Peace.txt
client_thread.py  mininet  oftest  pox  server_thread.py
mininet@mininet-vm:~$ python client_fork.py
Waiting for connection response

Elapsed Time: 0.015106916427612305
mininet@mininet-vm:~$ python client_fork.py
Waiting for connection response

Elapsed Time: 0.01438140869140625
mininet@mininet-vm:~$ python client_fork.py
Waiting for connection response

Elapsed Time: 0.015726804733276367
mininet@mininet-vm:~$ python client_fork.py
Waiting for connection response

Elapsed Time: 0.01658034324645996
mininet@mininet-vm:~$ python client_fork.py
Waiting for connection response

Elapsed Time: 0.012058734893798828
mininet@mininet-vm:~$ python client_fork.py
Waiting for connection response

Elapsed Time: 0.014309406280517578
mininet@mininet-vm:~$
```

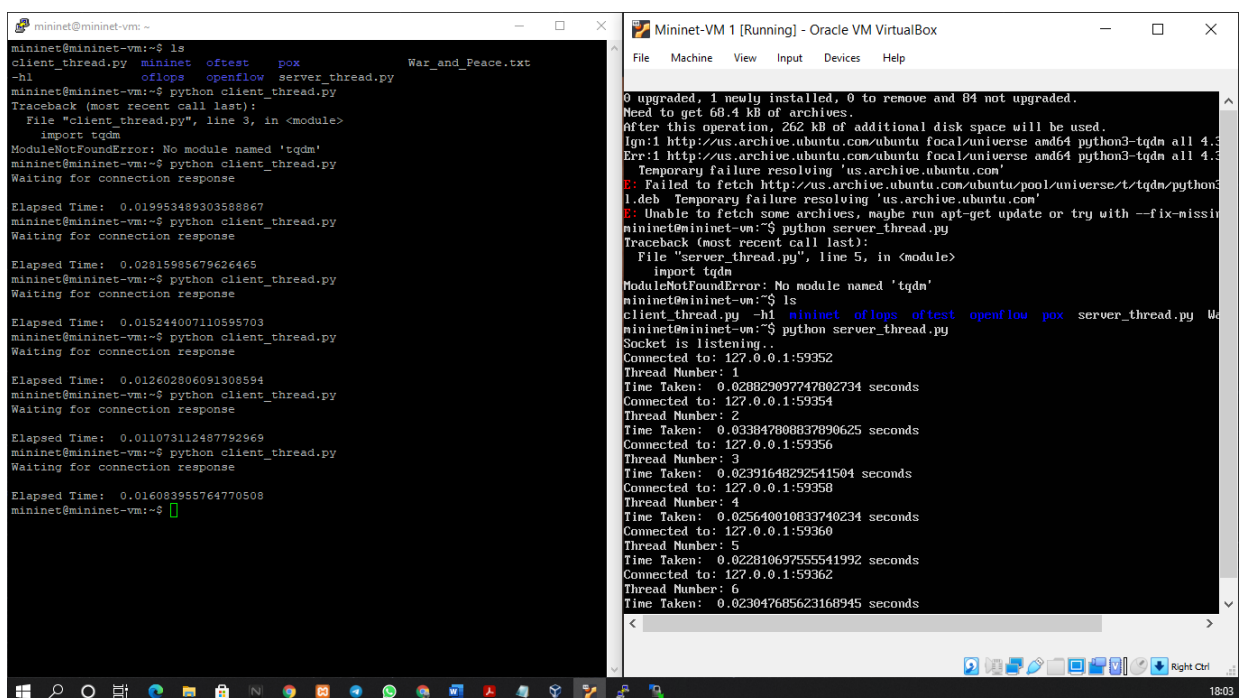
```
Mininet-VM 1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

Time Taken: 0.028829097747802734 seconds
Connected to: 127.0.0.1:59354
Thread Number: 2
Time Taken: 0.033847808837890625 seconds
Connected to: 127.0.0.1:59356
Thread Number: 3
Time Taken: 0.02391648292541504 seconds
Connected to: 127.0.0.1:59358
Thread Number: 4
Time Taken: 0.025640010833740234 seconds
Connected to: 127.0.0.1:59360
Thread Number: 5
Time Taken: 0.022810697555541992 seconds
Connected to: 127.0.0.1:59362
Thread Number: 6
Time Taken: 0.023047685623168945 seconds
Thread Number: 7
[1]: Stopped python server_thread.py
mininet@mininet-vm:~$ ls
client_fork.py  -hl  oflops  openflow  server_fork.py  War_and_Peace.txt
client_thread.py  mininet  oftest  pox  server_thread.py
mininet@mininet-vm:~$ python server_fork.py
Socket is listening..
Connected to: 127.0.0.1:41582
Time Taken: 0.023780345916748047 seconds
Connected to: 127.0.0.1:41584
Time Taken: 0.02969527244567871 seconds
Connected to: 127.0.0.1:41586
Time Taken: 0.03076744079508438 seconds
Connected to: 127.0.0.1:41588
Time Taken: 0.02509450912475586 seconds
Connected to: 127.0.0.1:41590
Time Taken: 0.020992040634155273 seconds
Connected to: 127.0.0.1:41592
Time Taken: 0.02300572395324707 seconds
```

Avg. Time: 0.025s

Avg. Throughput: 140MB/s

## Thread Results:



```
mininet@mininet-vm:~$ ls
client_thread.py  mininet  oftest  pox  War_and_Peace.txt
-hl  oflops  openflow  server_thread.py
mininet@mininet-vm:~$ python client_thread.py
Traceback (most recent call last):
  File "client_thread.py", line 3, in <module>
    import tqdm
ModuleNotFoundError: No module named 'tqdm'
mininet@mininet-vm:~$ python client_thread.py
Waiting for connection response

Elapsed Time: 0.019953489303588867
mininet@mininet-vm:~$ python client_thread.py
Waiting for connection response

Elapsed Time: 0.02815985679626465
mininet@mininet-vm:~$ python client_thread.py
Waiting for connection response

Elapsed Time: 0.015244007110595703
mininet@mininet-vm:~$ python client_thread.py
Waiting for connection response

Elapsed Time: 0.012602806091308594
mininet@mininet-vm:~$ python client_thread.py
Waiting for connection response

Elapsed Time: 0.011073112487792969
mininet@mininet-vm:~$ python client_thread.py
Waiting for connection response

Elapsed Time: 0.016083955764770508
mininet@mininet-vm:~$
```

```
Mininet-VM 1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

0 upgraded, 1 newly installed, 0 to remove and 84 not upgraded.
Need to get 60.4 kB of archives.
After this operation, 262 kB of additional disk space will be used.
Ign:1 http://us.archive.ubuntu.com/ubuntu focal/universe amd64 python3-tqdm all 4.3
Err:1 http://us.archive.ubuntu.com/ubuntu focal/universe amd64 python3-tqdm all 4.3
Temporary failure resolving 'us.archive.ubuntu.com'
E: Failed to fetch http://us.archive.ubuntu.com/ubuntu/pool/universe/t/tqdm/python3
l.deb Temporary failure resolving 'us.archive.ubuntu.com'
E: Unable to fetch some archives, maybe run apt-get update or try with --fix-missin
mininet@mininet-vm:~$ python server_thread.py
Traceback (most recent call last):
  File "server_thread.py", line 5, in <module>
    import tqdm
ModuleNotFoundError: No module named 'tqdm'
mininet@mininet-vm:~$ ls
client_thread.py  -hl  mininet  oflops  oftest  openflow  pox  server_thread.py  Wa
mininet@mininet-vm:~$ python server_thread.py
Socket is listening..
Connected to: 127.0.0.1:59352
Thread Number: 1
Time Taken: 0.028829097747802734 seconds
Connected to: 127.0.0.1:59354
Thread Number: 2
Time Taken: 0.033847808837890625 seconds
Connected to: 127.0.0.1:59356
Thread Number: 3
Time Taken: 0.02391648292541504 seconds
Connected to: 127.0.0.1:59358
Thread Number: 4
Time Taken: 0.025640010833740234 seconds
Connected to: 127.0.0.1:59360
Thread Number: 5
Time Taken: 0.022810697555541992 seconds
Connected to: 127.0.0.1:59362
Thread Number: 6
Time Taken: 0.023047685623168945 seconds
```

Avg. Time: 0.022s

Avg. Throughput: 158MB/s