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

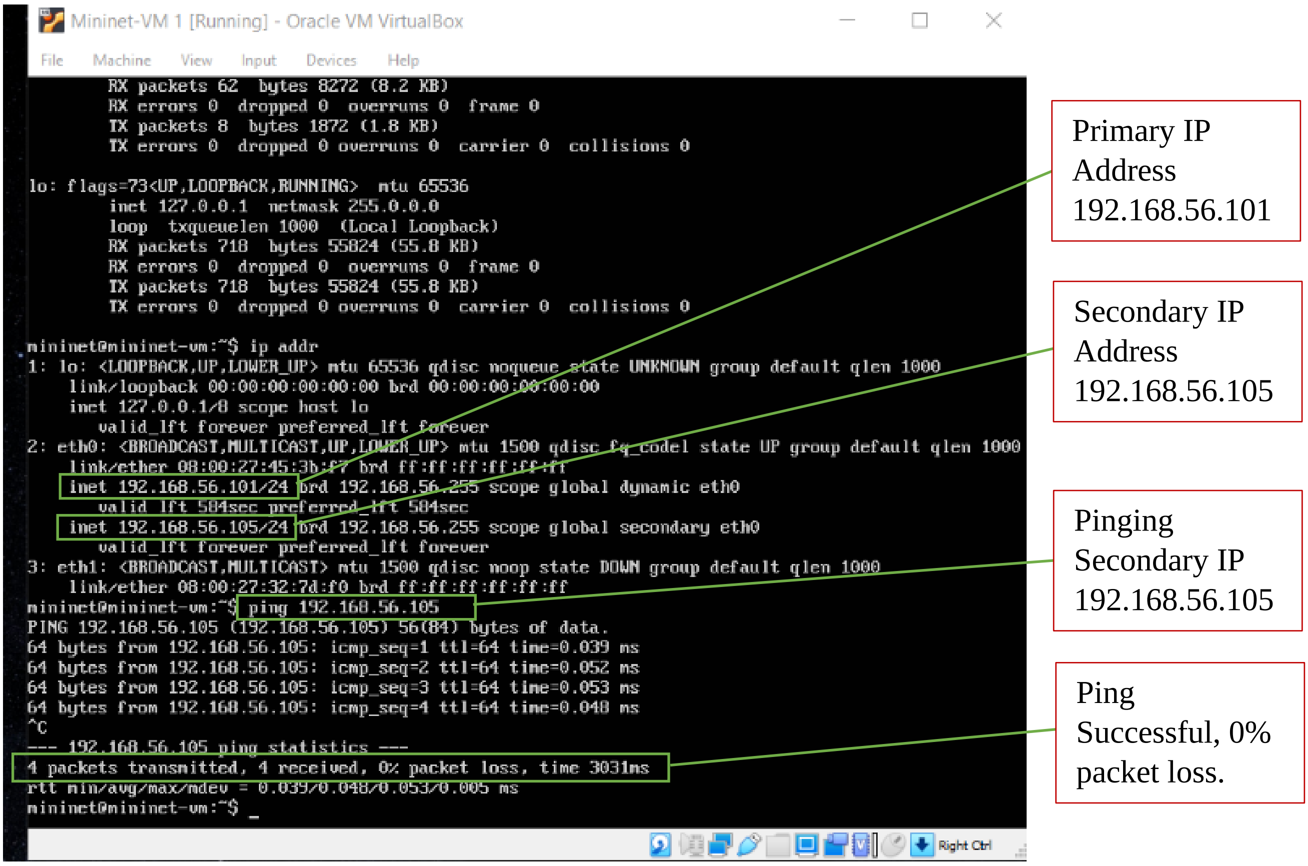


Figure 1. IP Aliasing, Adding Secondary IP Address and ping to check it’s working or not.

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

2. 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.

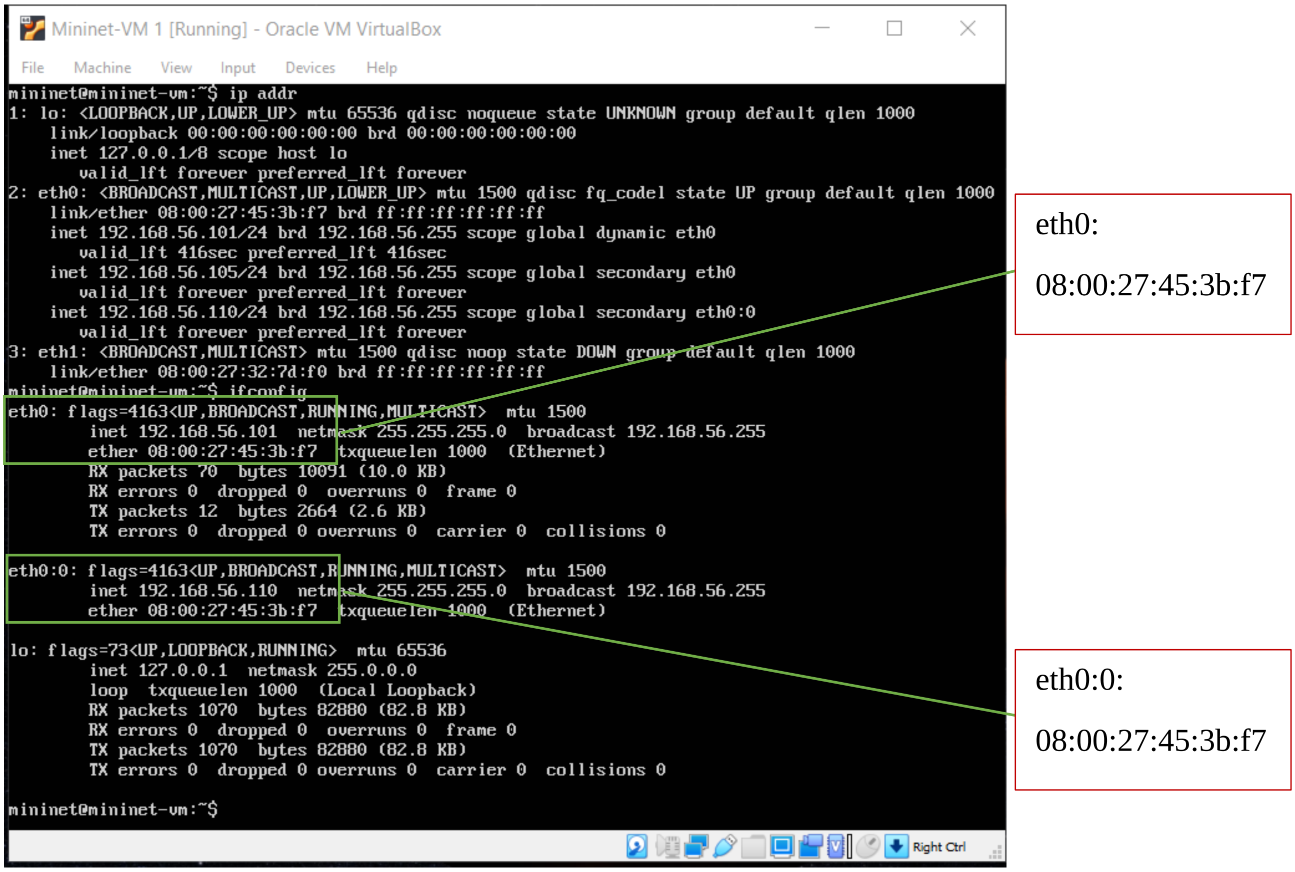


Figure 2. MAC aliasing, adding another interface using the same mac address as that of available one.

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.

<u>Single Connection:</u>

Fork Model

Server File Name: *server_fork.py*
Client File Name: *client.py*

Starting only one server and one client.

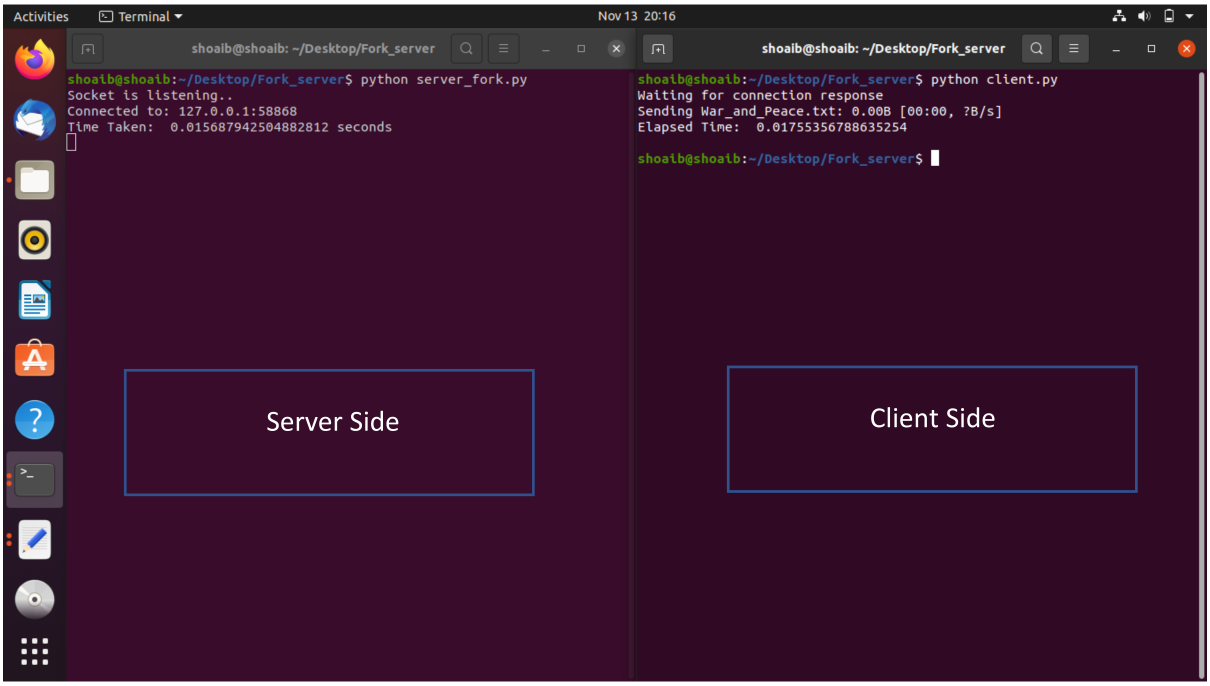


Figure 3. Concurrent Server implementation using Forks. Each new client will be serve as independent process using fork.

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.

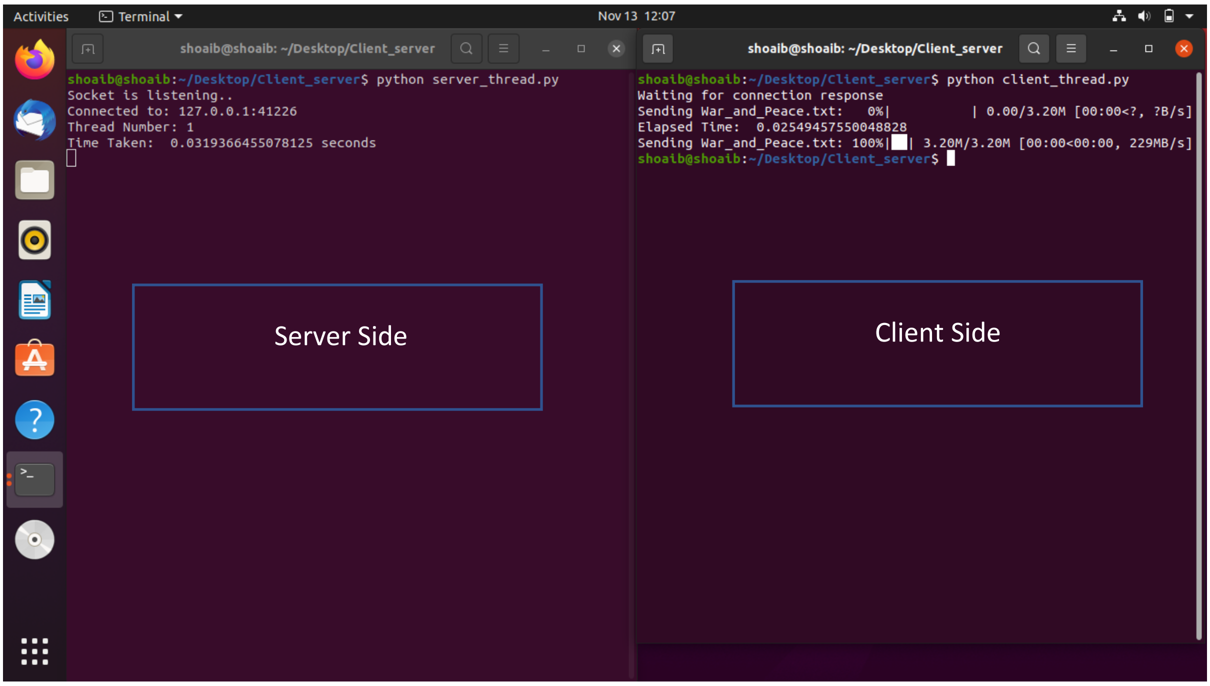


Figure 4. Implementation of concurrent server using threads. Each new client will be assigned new thread and parent thread will handle all the new requests.

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.

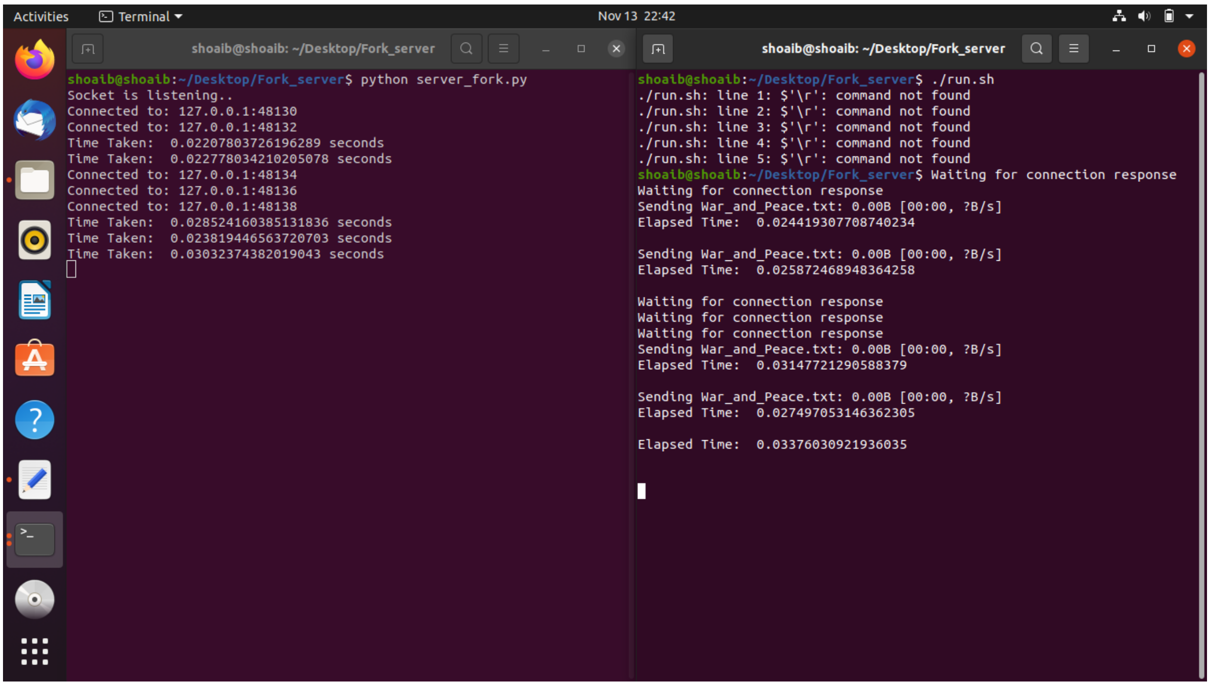


Figure 5. Starting many clients together and sending them in background so that all the clients can start simultaneously. This is an example of handling concurrent client server model using fork. On the left it's server who is receiving the requests and on right are the multiple clients who are trying 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.

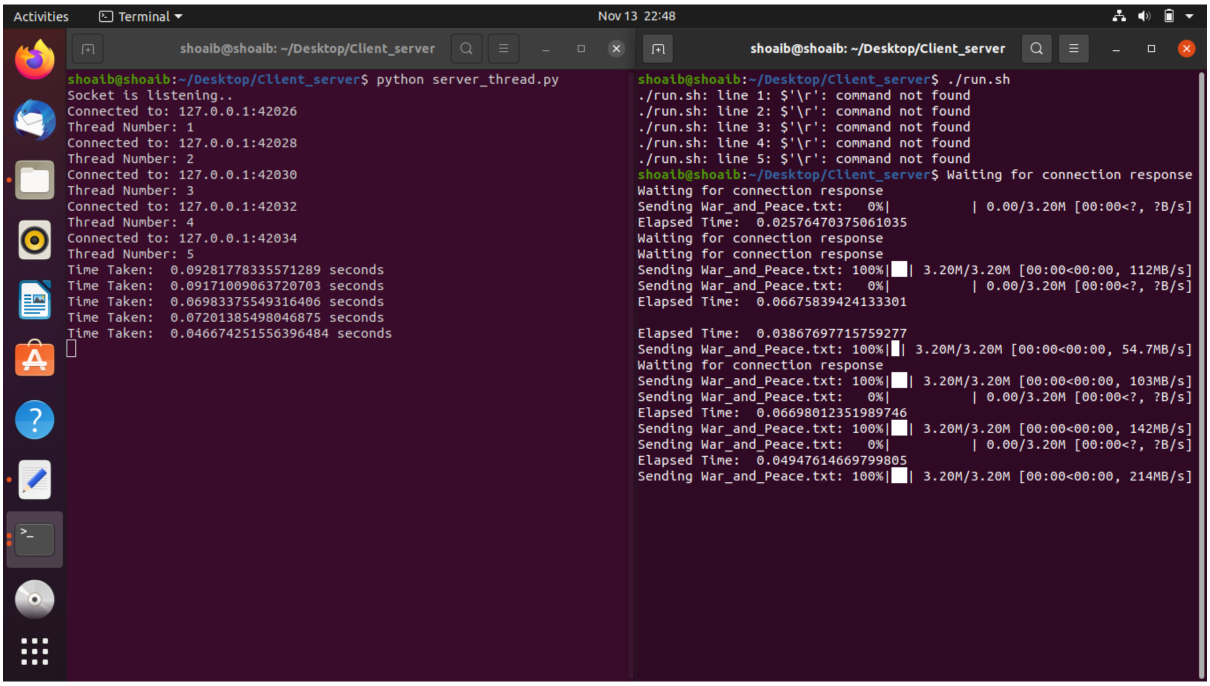


Figure 6. . Starting many clients together and sending them in background so that all the clients can start simultaneously. This is an example of handling concurrent client server model using Threads. On the left it's server who is receiving the requests and on right are the multiple clients who are 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 a 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
```

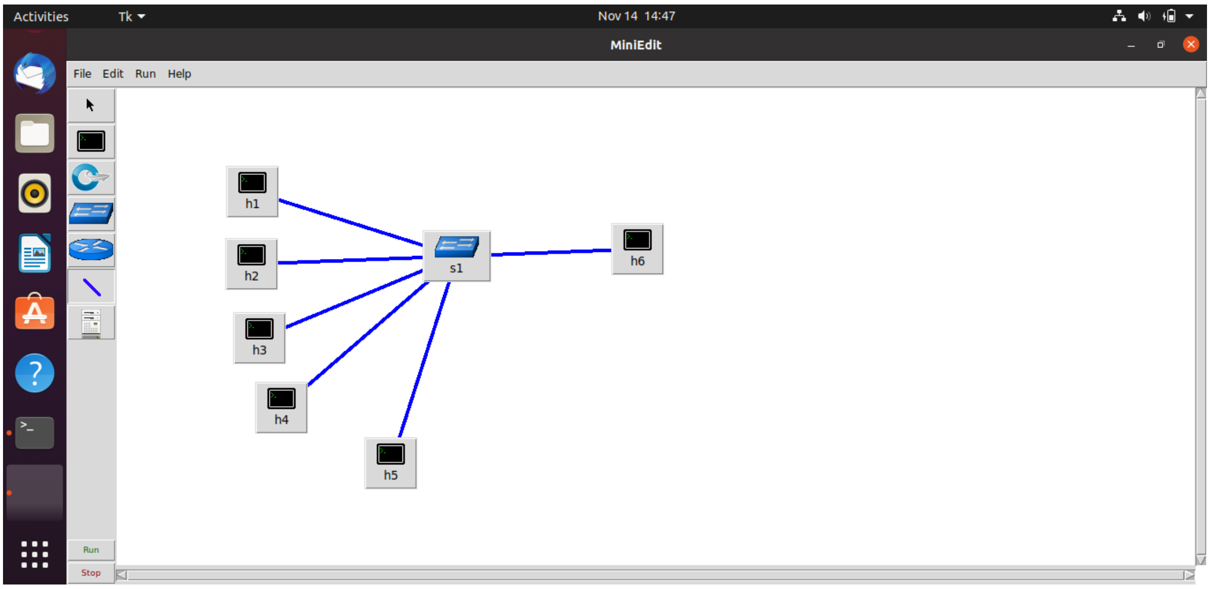


Figure 7. Create a single topology of 6 hosts. This is for visualization how our topology looks like. It is made using miniedit a simple light weight GUI based tool provided by mininet.

Step 2: Open xterm terminal for each node

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

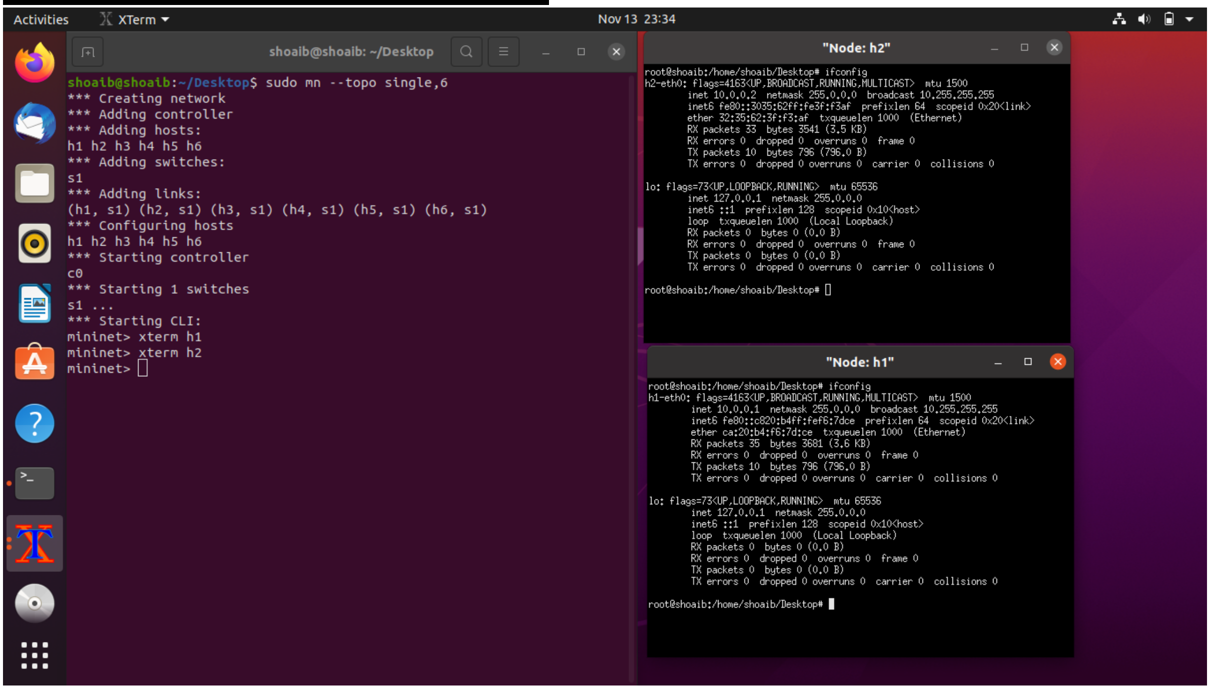


Figure 8.Use xterm command to open xterm terminal for each nodes separately. From here we can control each nodes and can configure them according to our needs.

Step 3: Check IP address of each nodes

#ifconfig

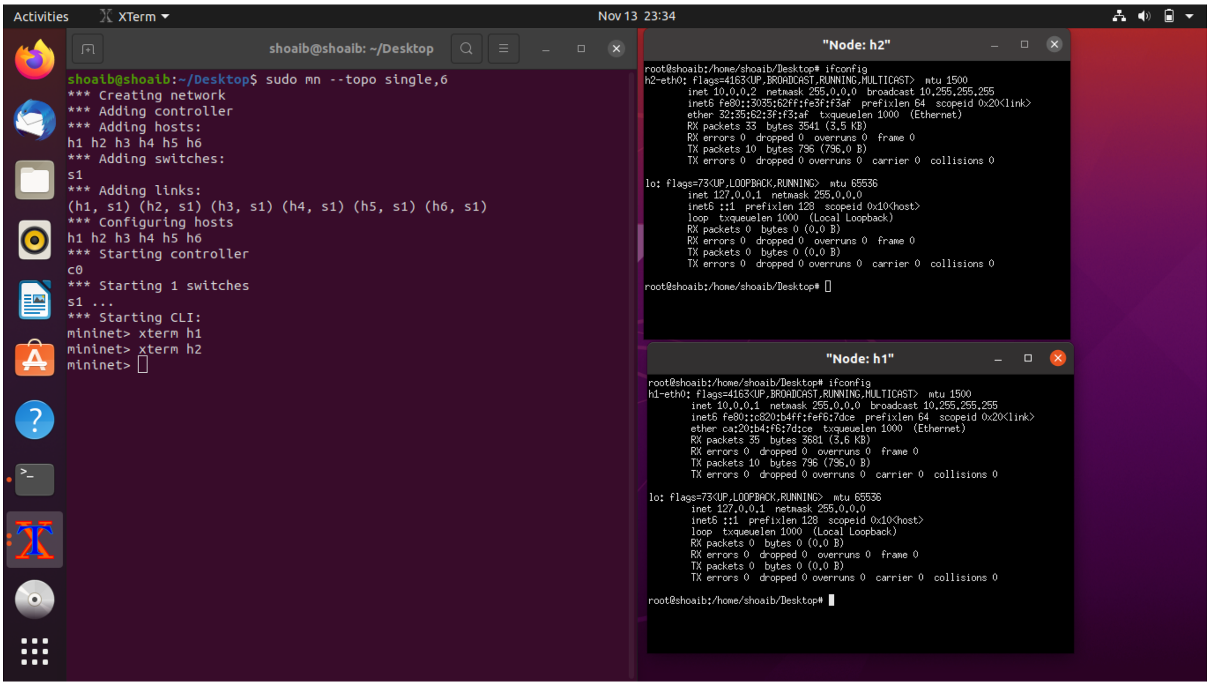


Figure 9. Once created open xterm terminal for each of the hosts. You can also check the IP's of each host using ifconfig that will help us to establish the connection.

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

#iperf3 -s

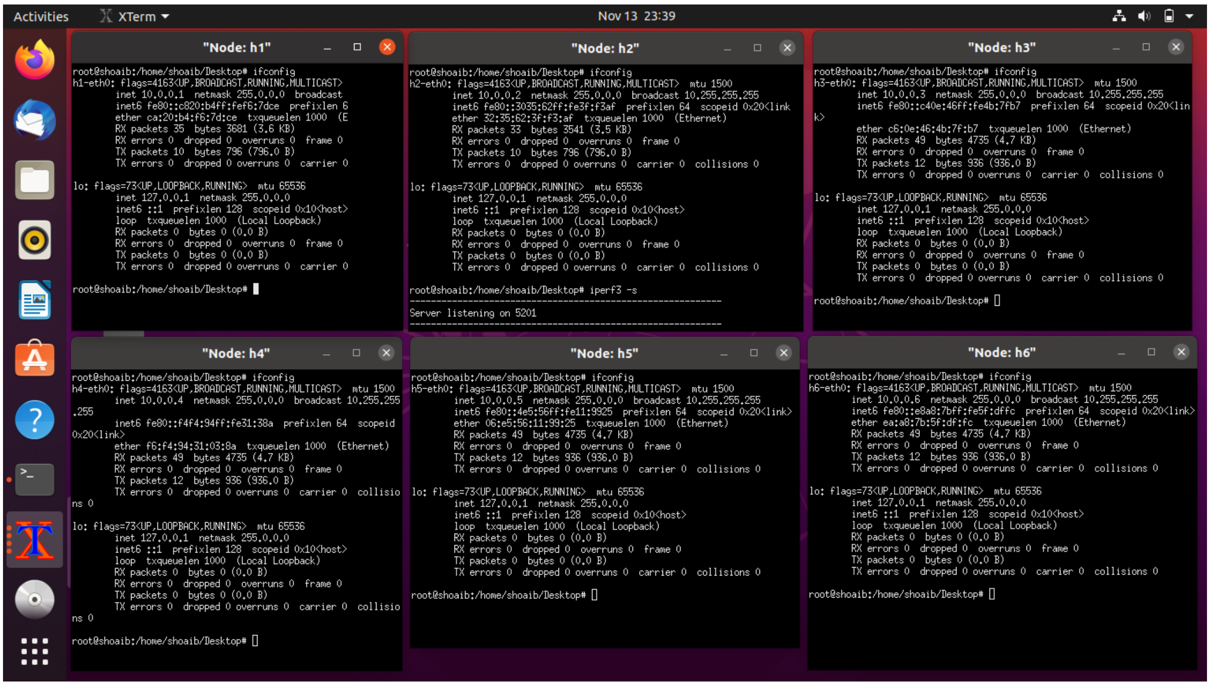


Figure 10. Using iperf3 we are testing the connection between each of the hosts. In this case "Node:h2" is acting as server and rest other nodes are working like clients.

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

`#iperf3 -c 10.0.0.2`

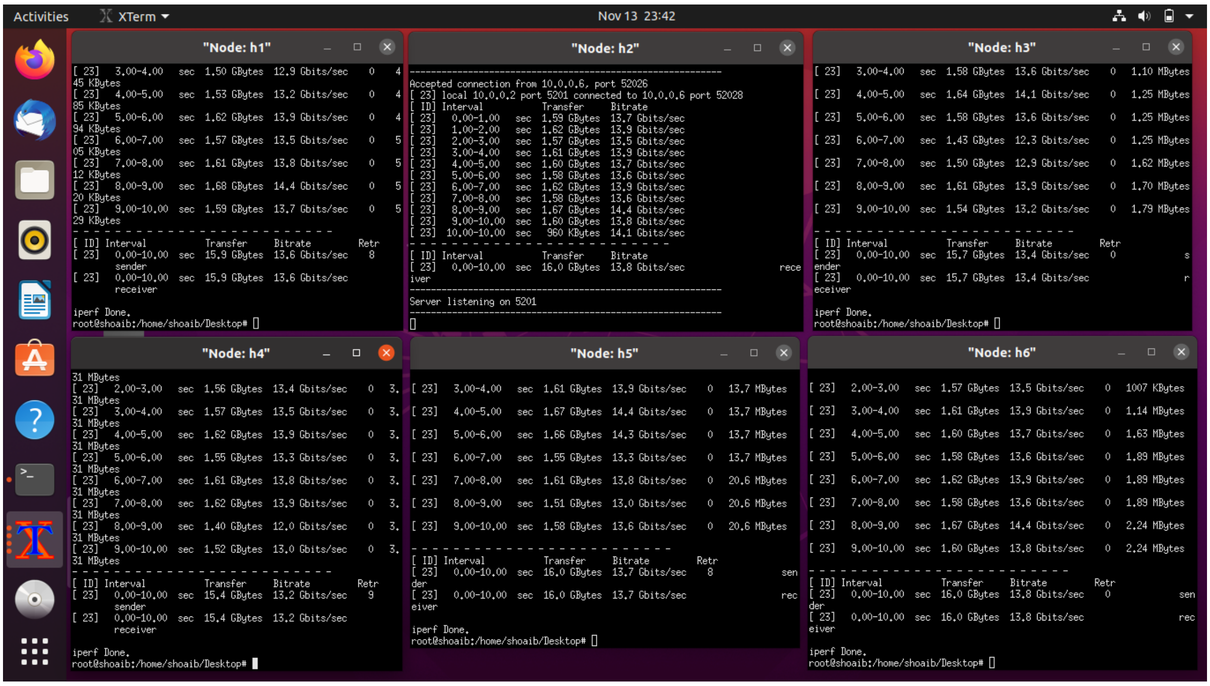


Figure 11. Each of the Nodes has their data and the summarized information are there on their respective terminals, i.e., Transfer Rates, Bitrates, cnwd, etc.

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 noes 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
```

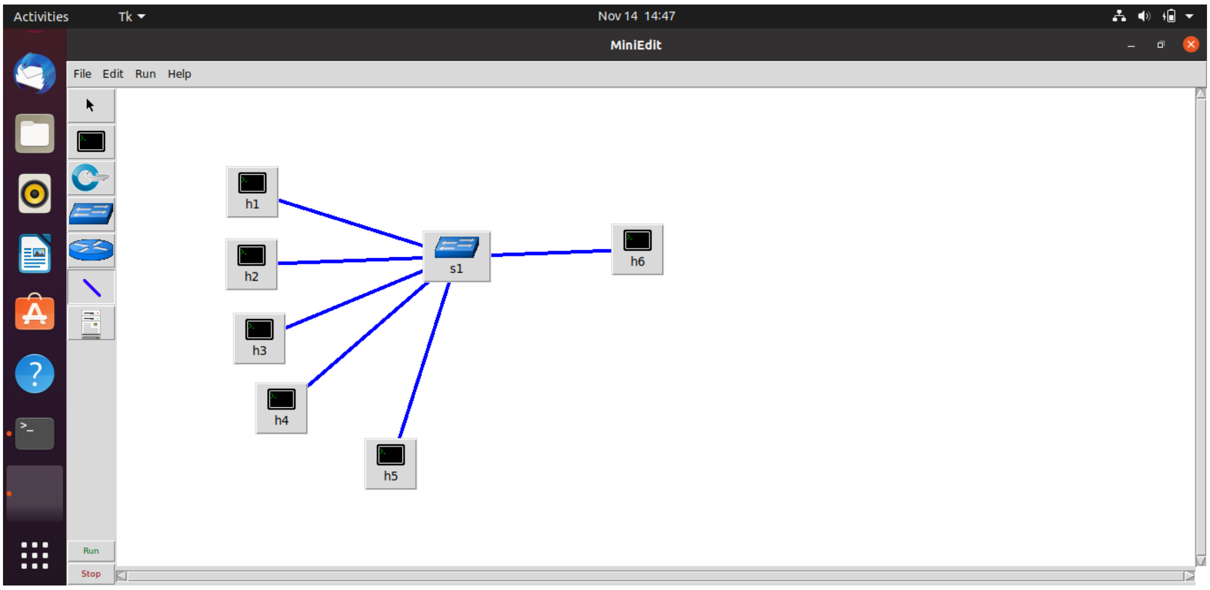


Figure 12. . Create a single topology of 6 hosts. This is for visualization how our topology looks like. It is made using miniedit a simple light weight GUI based tool provided by mininet.

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

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

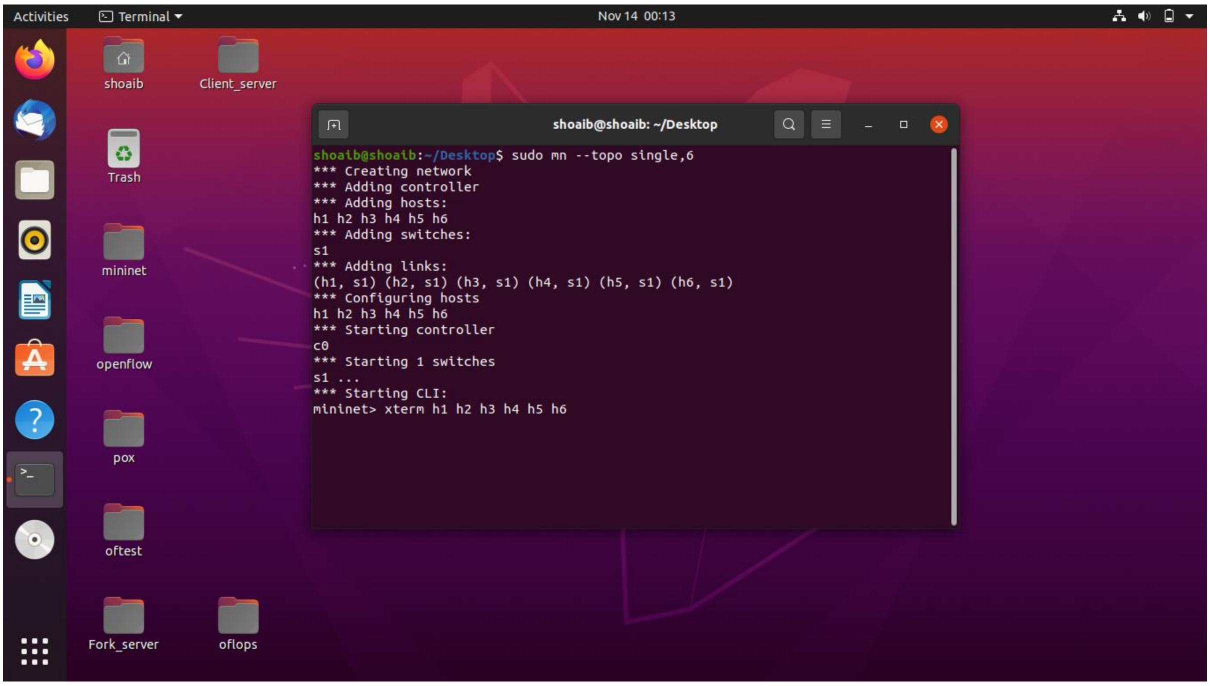


Figure 13. Use xterm command to open xterm terminal for each nodes separately. From here we can control each nodes and can configure them according to our needs

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

```
#iperf3 -s
```

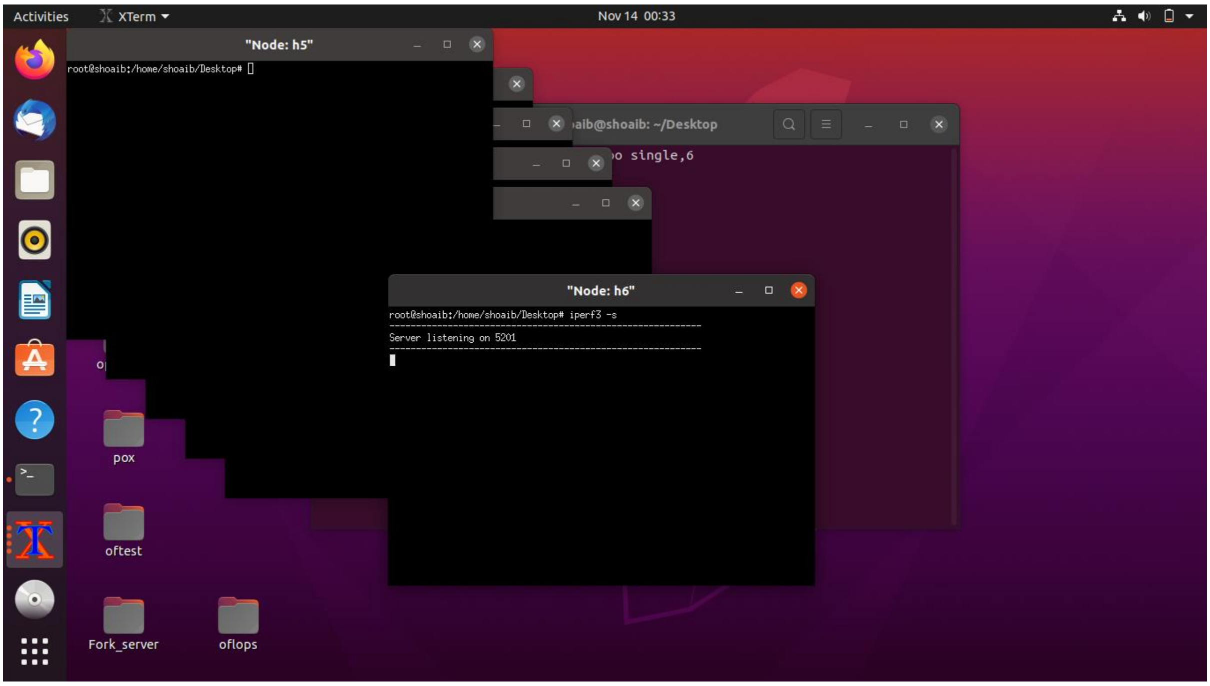


Figure 14. Using iperf3 we are testing the connection between each of the hosts. In this case "Node:h6" is acting as server and rest other nodes are working like clients.

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
```

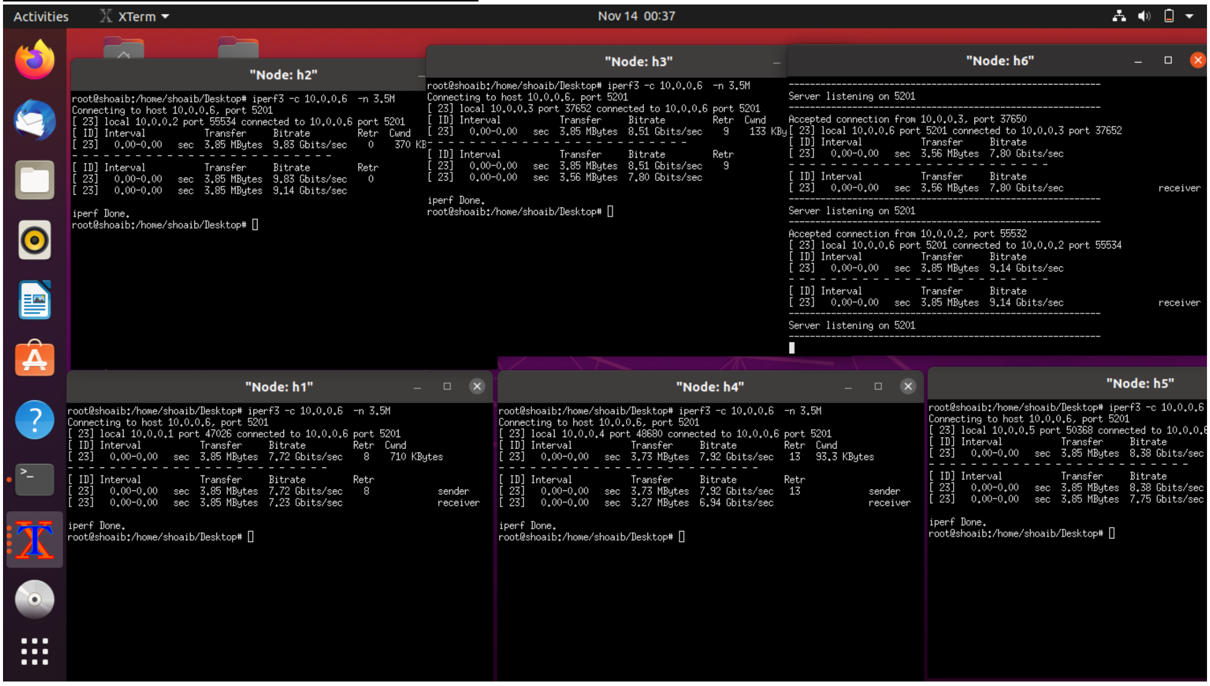


Figure 15. As we want to compare the mininet model and non-mininet model, we have send the same size of file across the different clients. So, we have used -n option of iperf3 which allows us to specify the file size that we want to send.

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, <u>the mininet system performs much better</u> 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.

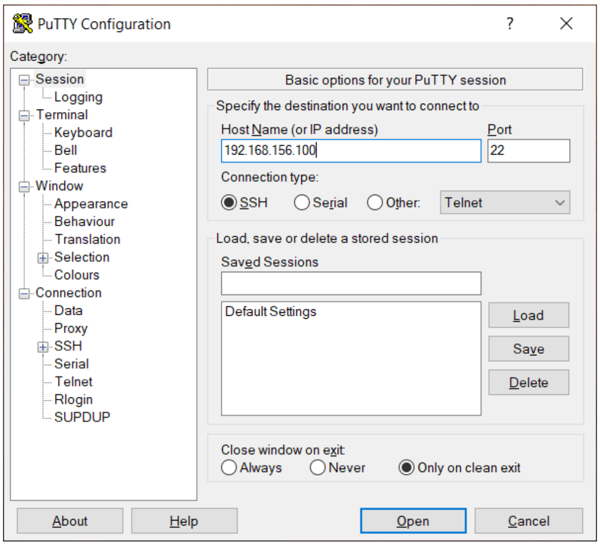


Figure 16. PuTTY Configuration window. Enter Host IP in order to connect it with particular host.

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.

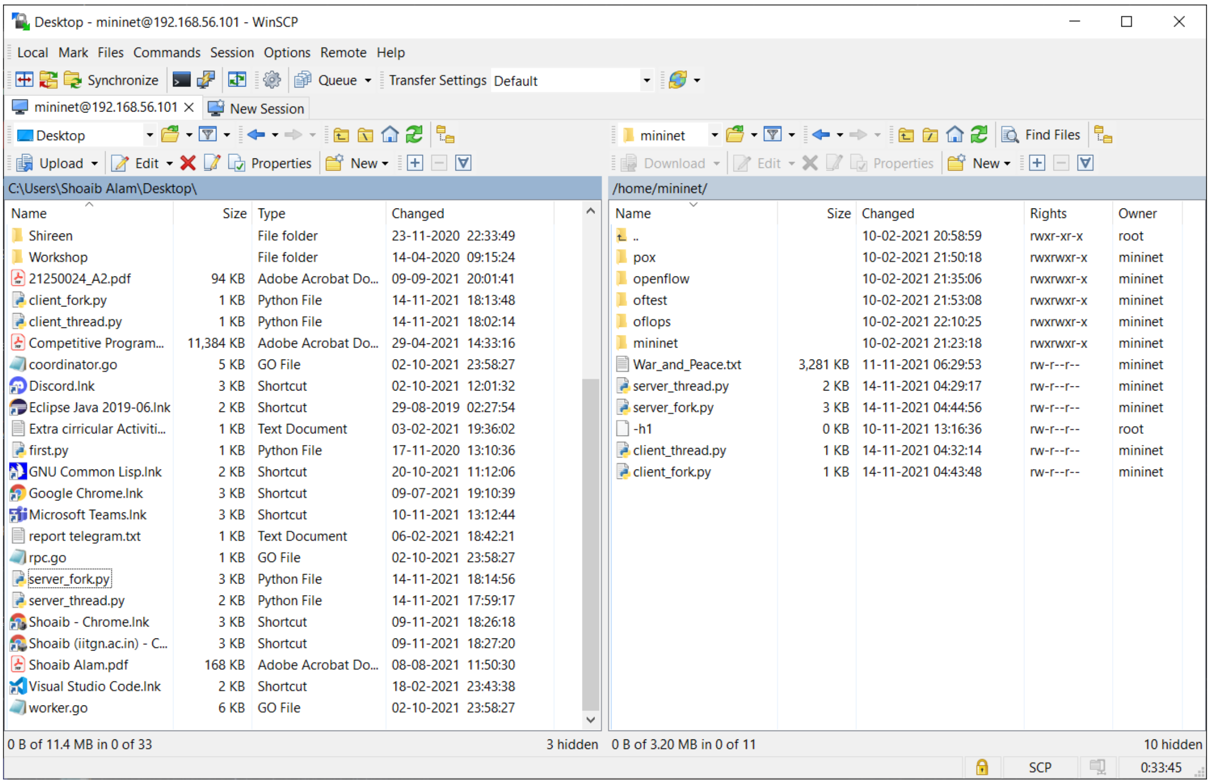
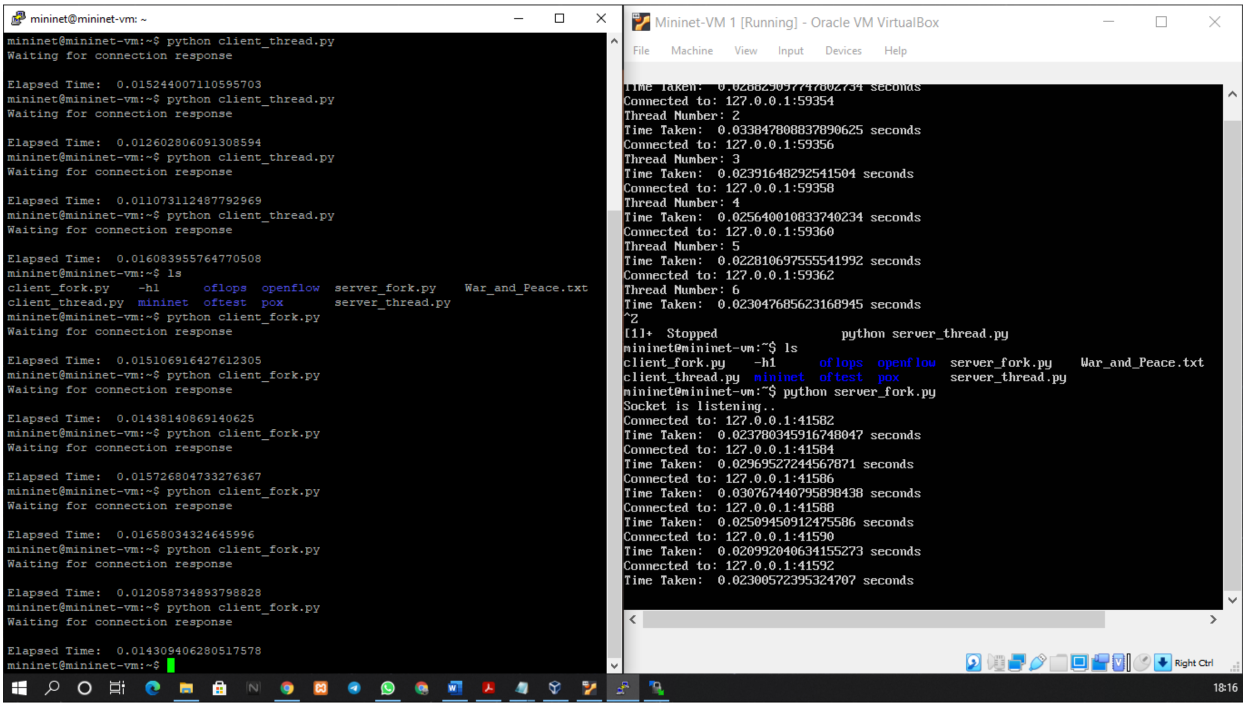


Figure 17. WinSCP Program. Use this to transfer the files to mininet system.

Now start the clients and server as you did previously.

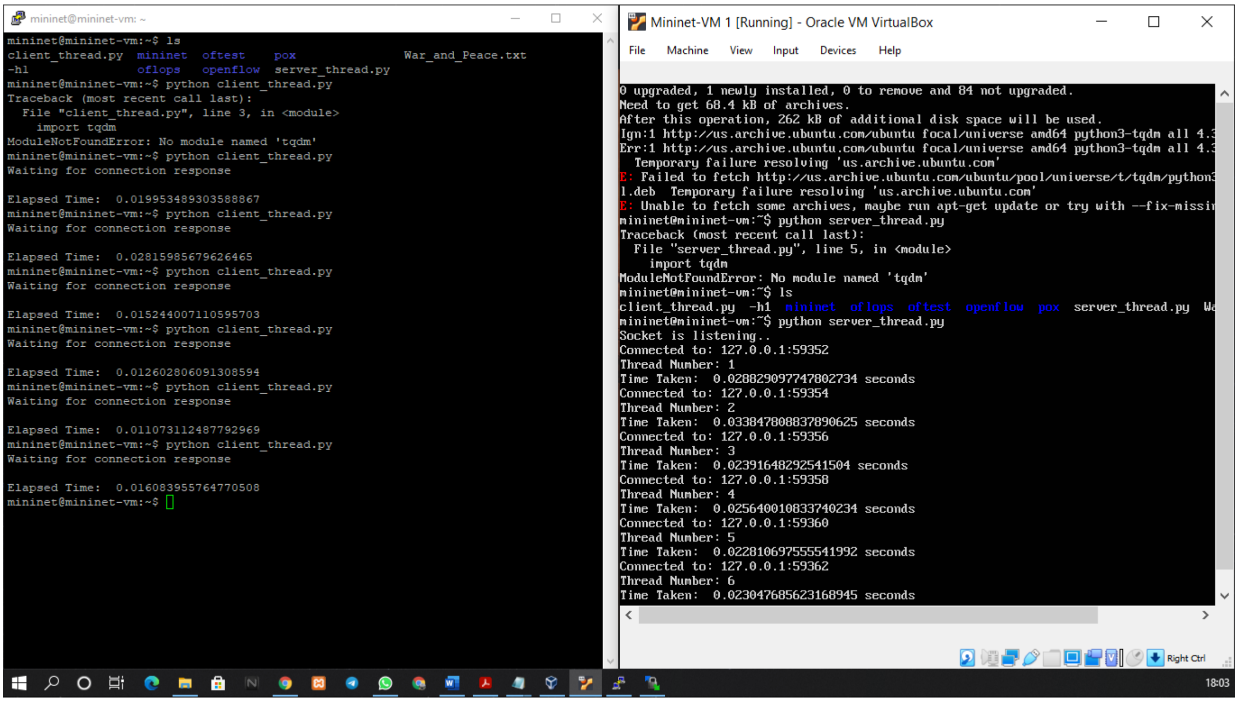
Fork Results:



Avg. Time: 0.025s

Avg. Throughput: 140MB/s

Thread Results:



Avg. Time: 0.022s

Avg. Throughput: 158MB/s