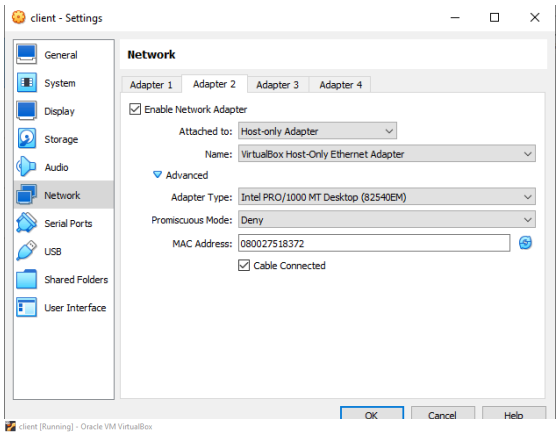
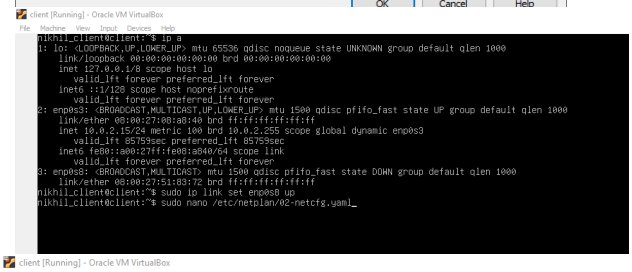
1) a) For the initial setup of four VM's I created 4 different VM's with each having OS Ubuntu server edition. After creating them I added a new Network adaptor available to each of the machines with the following configurations:-

For Client:-

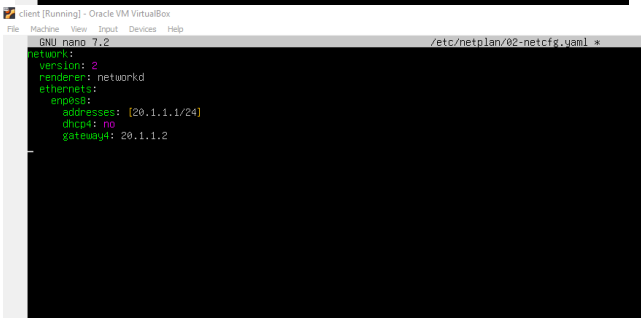


I added a new network adaptor with the following configuration.

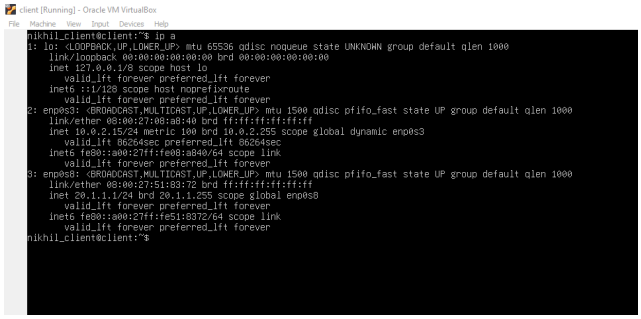
After this I ran the command "ip a" and then activated the newly connected adaptor using "sudo ip link set enp0s8 up" here enp0s8 was the name of the network adaptor.



After this I ran "sudo nano /etc/netplan/02-netcfg.yaml" to add a new configuration for this network adaptor.



These are the things I added to the newly created yaml file. This sets the static IP assigned to client as "20.1.1.1" with subnet mask value as 24 and gateway IP as "20.1.1.2"

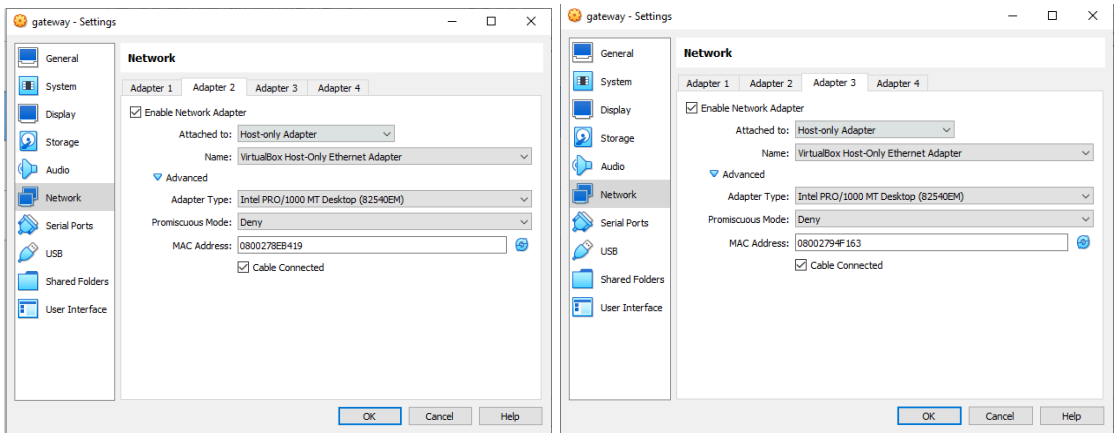


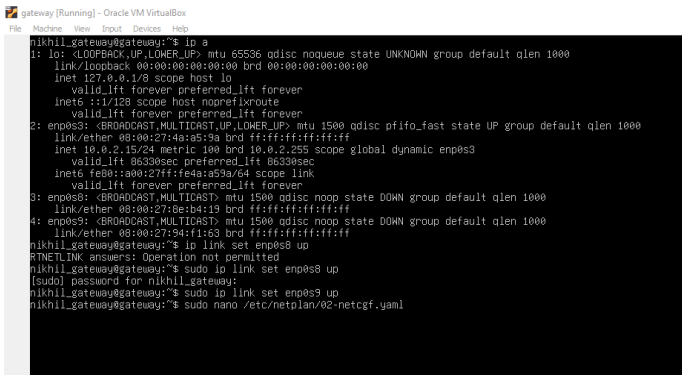
After this i applied this using command "sudo netplan apply" and then rebooted the system. This was the final output for "ip a".

And then added the route for communication with servers.

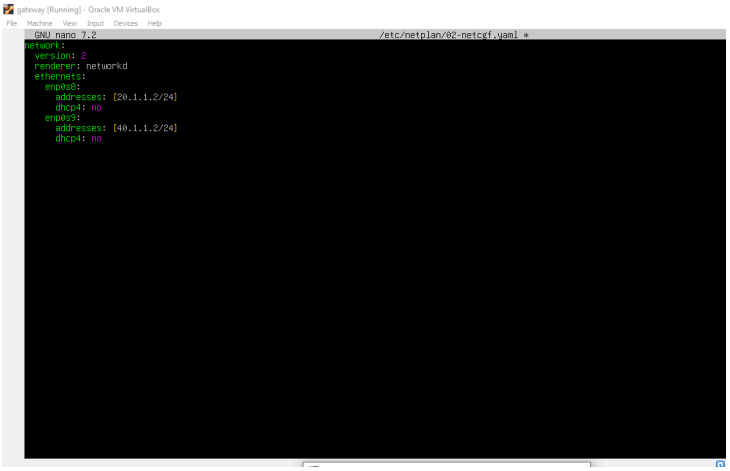
For gateway:-

There were 2 adaptors for the gateway, one for communicating with client and one for servers. Following are the configurations for both adaptors. (Adaptor 2&3).

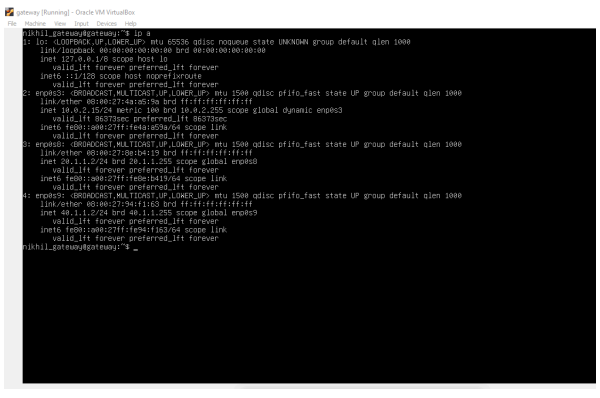




After this I ran the command “ip a” and then activated the newly connected adaptors using “sudo ip link set enp0s8 up” and “sudo ip link set enp0s9 up” here, enp0s8 and enp0s9 were the name of the network adaptor. Then after this i added the new configurations for yaml files.

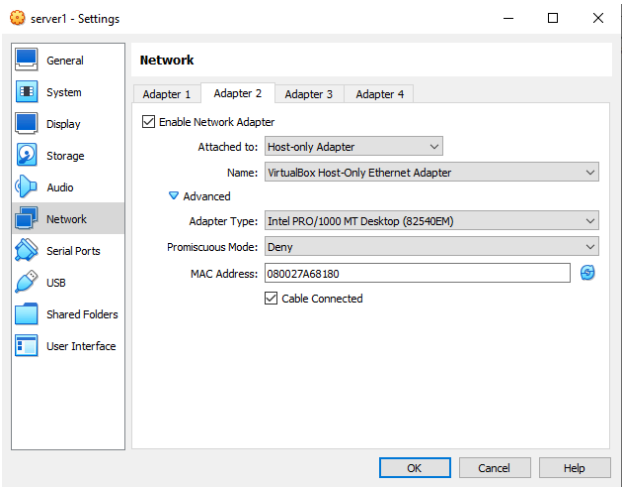


These are the things I added to the newly created yaml file. This sets the static IP assigned to adaptor 2 (which communicates with client) as “20.1.1.2” with subnet mask value as 24 and adaptor 3 (for communicating with servers) as “40.1.1.2” with subnet mask as 24. And after this i applied this using “sudo netplan apply” and rebooted the system.

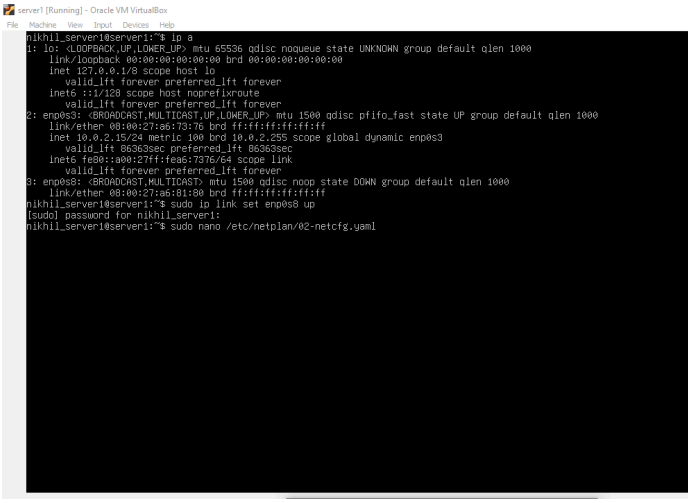


This is the final output for “ip a”

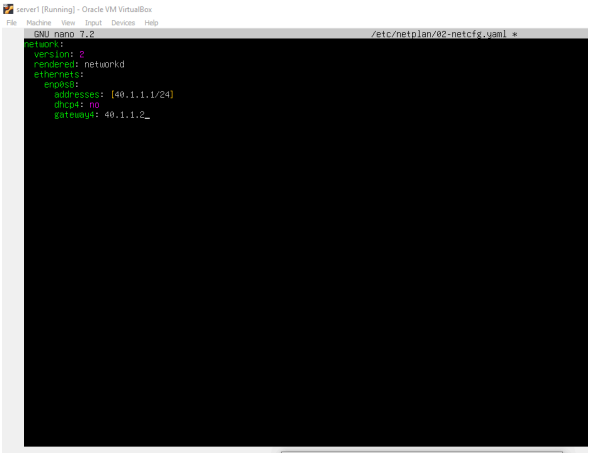
For server1:-



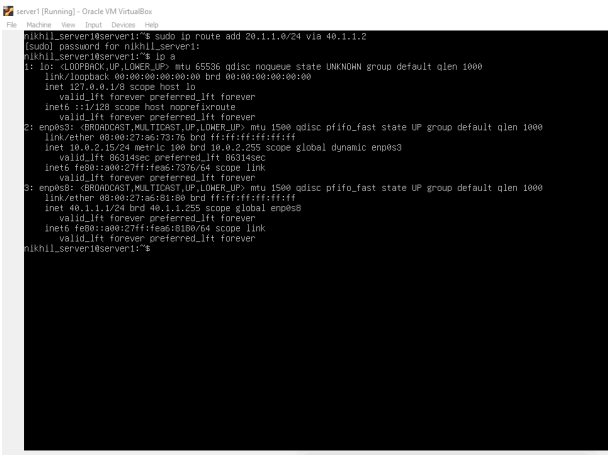
I added a new network adaptor with the following configuration.



After this I ran the command “ip a” and then activated the newly connected adaptor using “sudo ip link set enp0s8 up” here enp0s8 was the name of the network adaptor.



These are the things I added to the newly created yaml file. This sets the static IP assigned to server1 as “40.1.1.1” with subnet mask value as 24 and gateway IP as “40.1.1.2”. And after this applied these configurations.

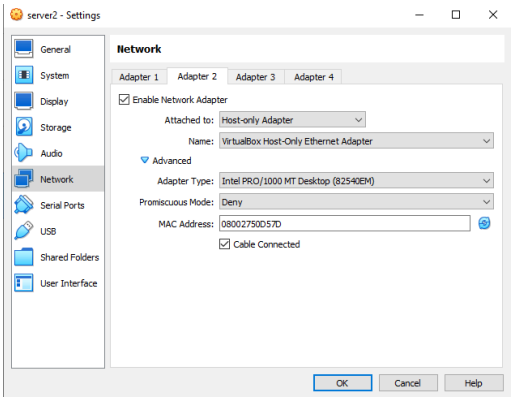


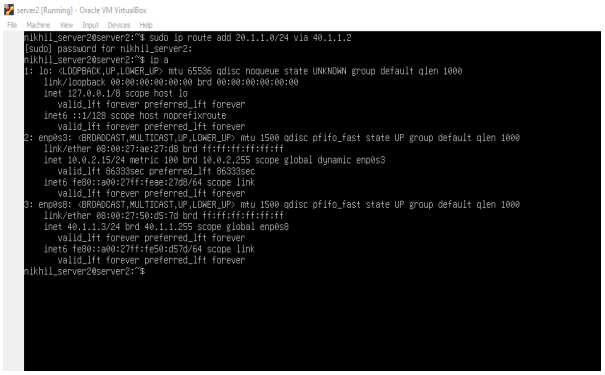
Here i added the route to connect to client using gateway using command “ip route add 20.1.1.0/24” via 40.1.1.2”. This is the final output for “ip a”

For server2:-

(Same things were done for server2 by just changing the route command and static IP address in yaml file).

enp0s8 is the new network adaptor.



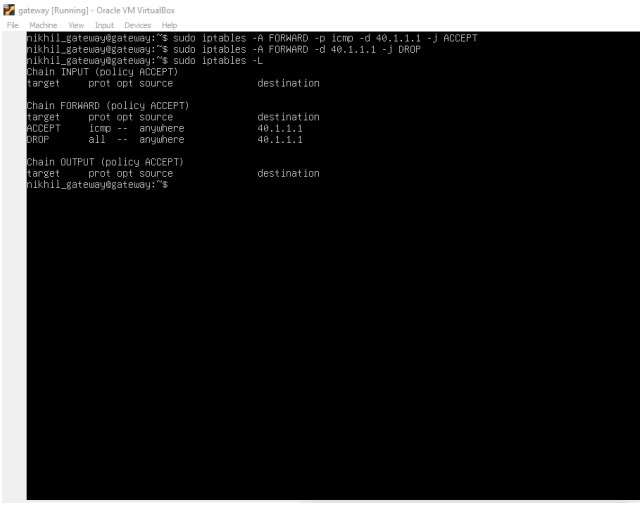


So, for part (a) we just need to add a new yaml file configuration and configure the routes using command “sudo ip route add <destination_ip> via <gateway_ip>”.

b) To add the forwarding functionality we just need to run this on gateway “sudo sysctl -w net.ipv4.ip_forward=1”. Following is the screenshot for same:-

2) a) To filter the traffic at the gateway VM we use the iptables command. At the gateway VM, we execute the following commands:-
sudo iptables -A FORWARD -p icmp -d 40.1.1.1 -j ACCEPT
sudo iptables -A FORWARD -d 40.1.1.1 -j DROP

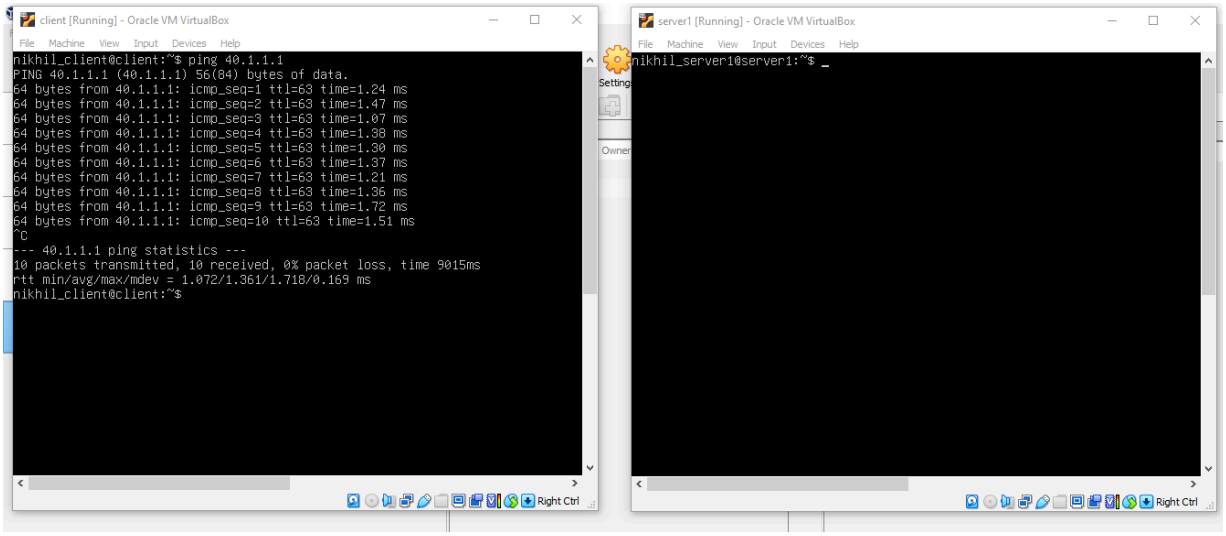
Here the first command allows the ICMP packets (ping) to server1 and the next command drops every packet that has destination as server1 IP's address. Following is the command on the gateway:-



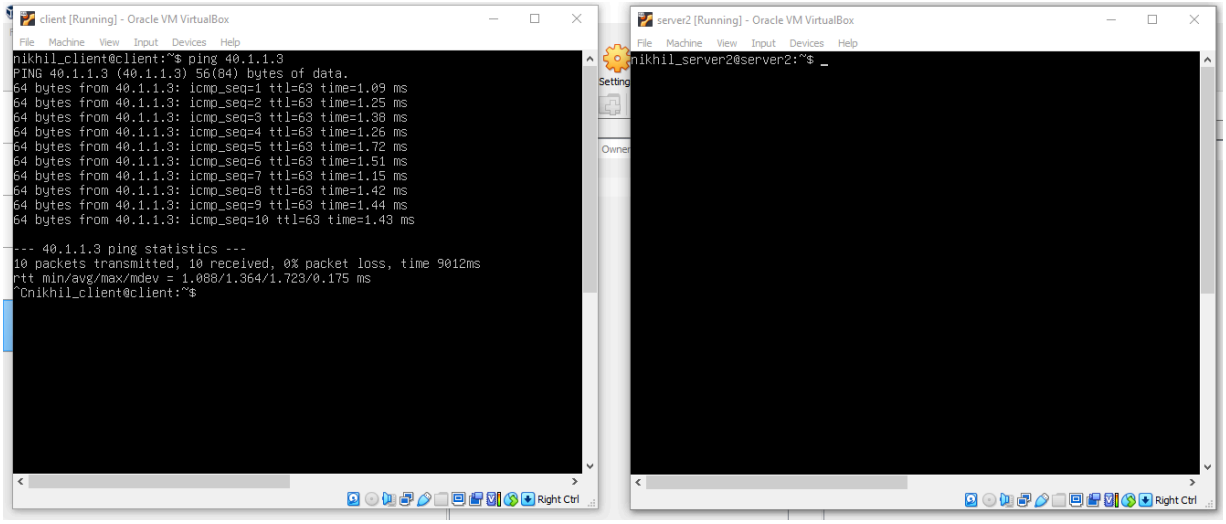
<u>Tests after applying the traffic filtering commands:-</u>

<u>Test for ping on server1 and server2:-</u>

Server1:-



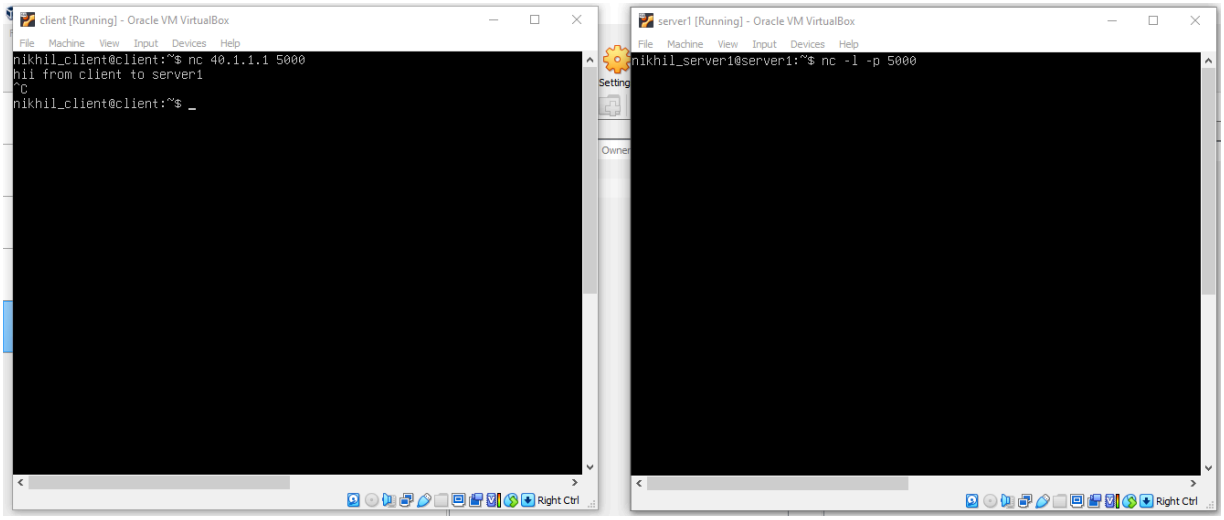
Server2:-



From these, it is clear that the ping worked for both the servers.

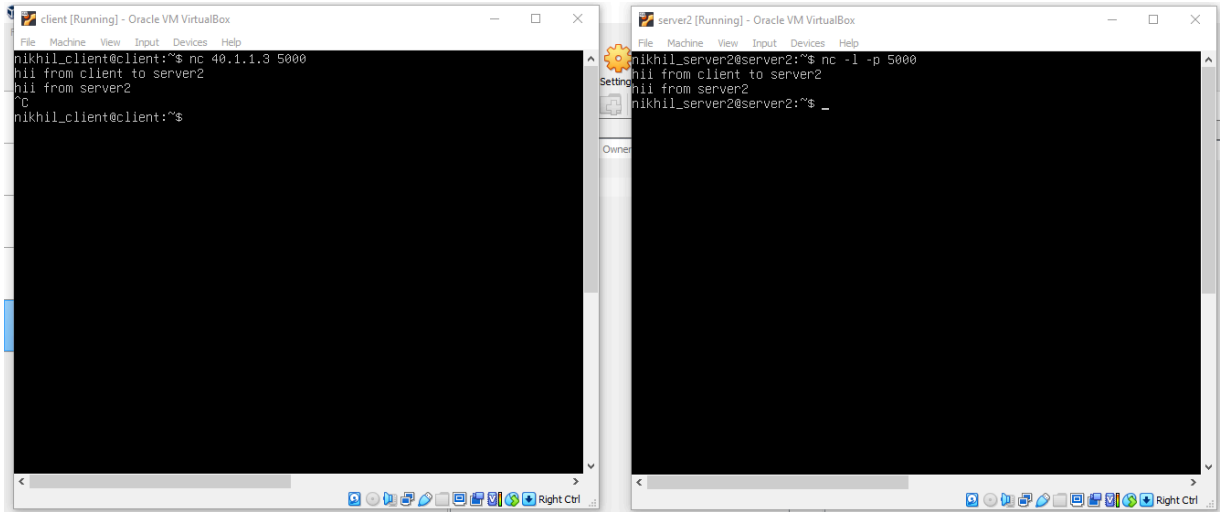
<u>Test for TCP requests:-</u>

Server1:-



Here the tcp connection between client and server1 fails, which proves that traffic was filtered at the gateway and everything was blocked.

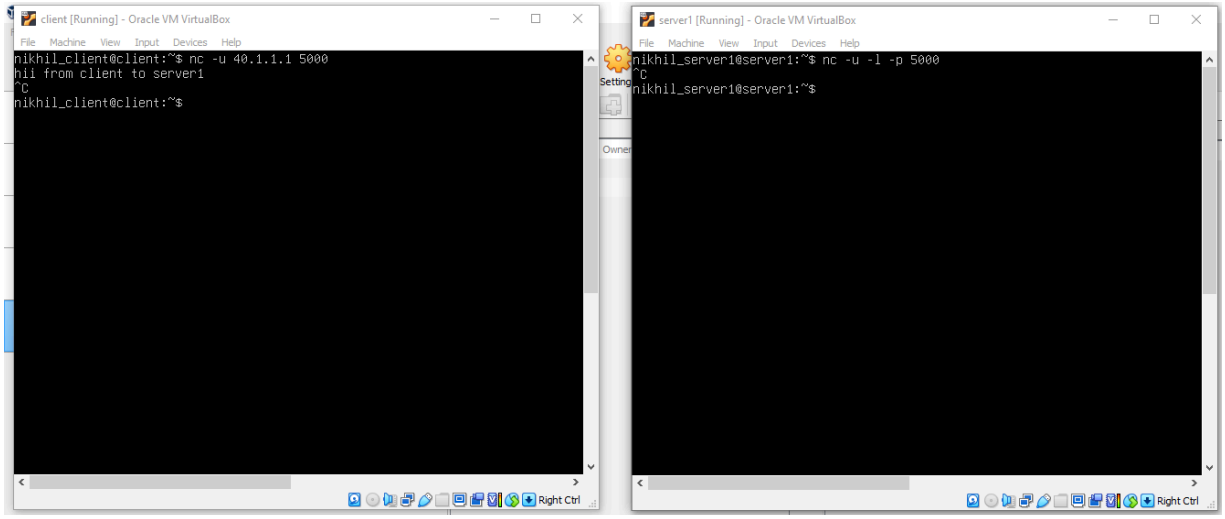
Server2:-



Here the TCP connection was successful, which indicates that traffic filtering is successful, as everything was allowed for server2.

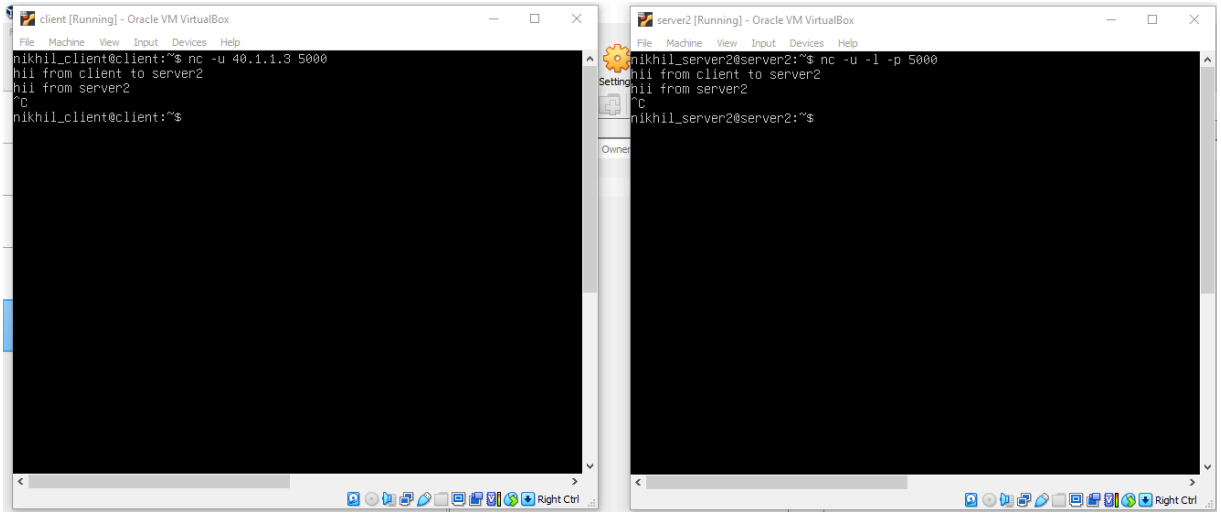
<u>Test for UDP requests:-</u>

Server1:-



Here the UDP connection between client and server1 fails, which proves that traffic was filtered at the gateway and everything was blocked.

Server2:-



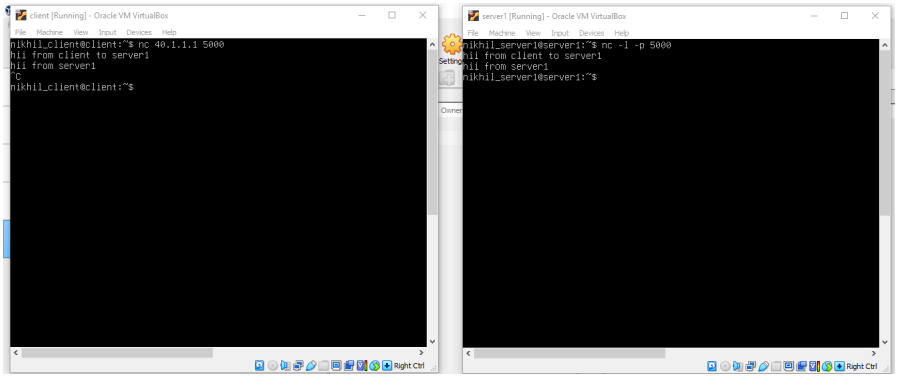
Here the UDP connection was successful, which indicates that traffic filtering is successful, as everything was allowed for server2.

<u>Tests after removing the traffic filtering commands:-</u>

Now I removed the earlier iptable entries from the gateway using the command: sudo iptables -F

<u>Test for TCP:-</u>

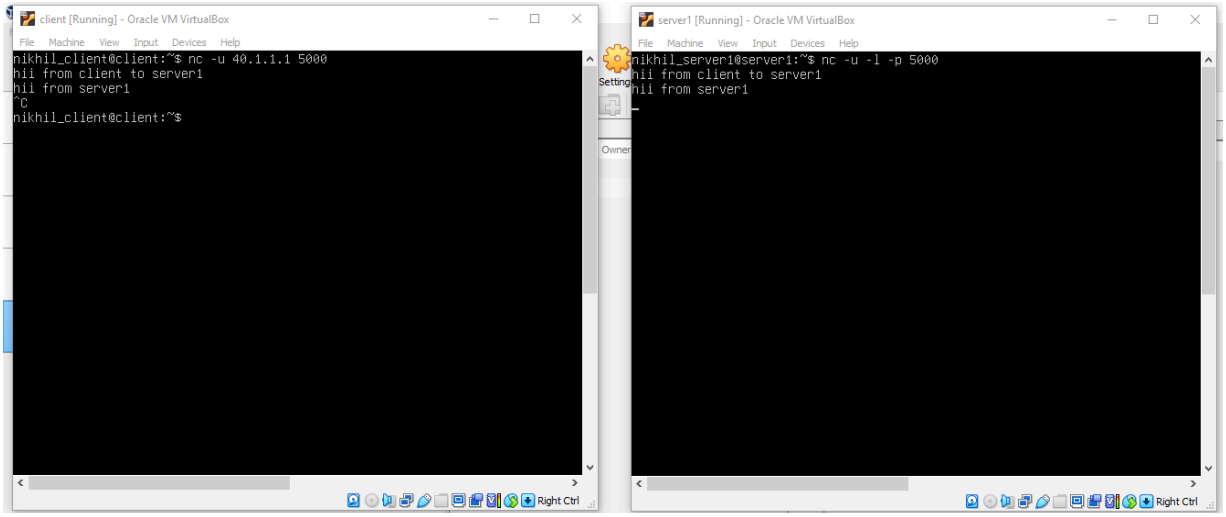
Server1:-



After removing the entries from the iptables for gateway, TCP connection between the client and server1 was successful.

Test for UDP:-

Server1:-



After removing the entries from the iptables for gateway, UDP connection between the client and server1 was successful.

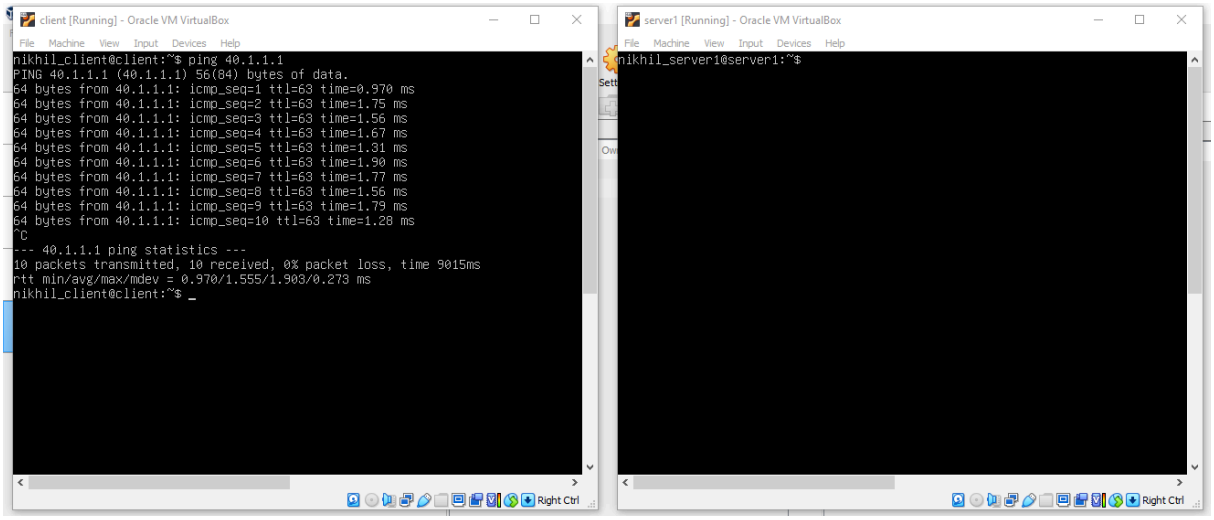
Both of these tests, after adding traffic filtering using iptables and after removing them from iptable at the gateway, clearly show that traffic filtering was successful and we were successfully able to block every traffic which was destined to server1 except ping.

b) To block all the traffic generated from the client that is: 20.1.1.1, we run the following commands on the gateway:-
sudo iptables -A FORWARD -p tcp -s 20.1.1.1 -j DROP
sudo iptables -A FORWARD -s 20.1.1.1 -j ACCEPT
Here the first command ensures that we drop all the TCP traffic generated by the client using the -p as tcp and -j as DROP for blocking the TCP traffic.
Following are the commands on the gateway:-

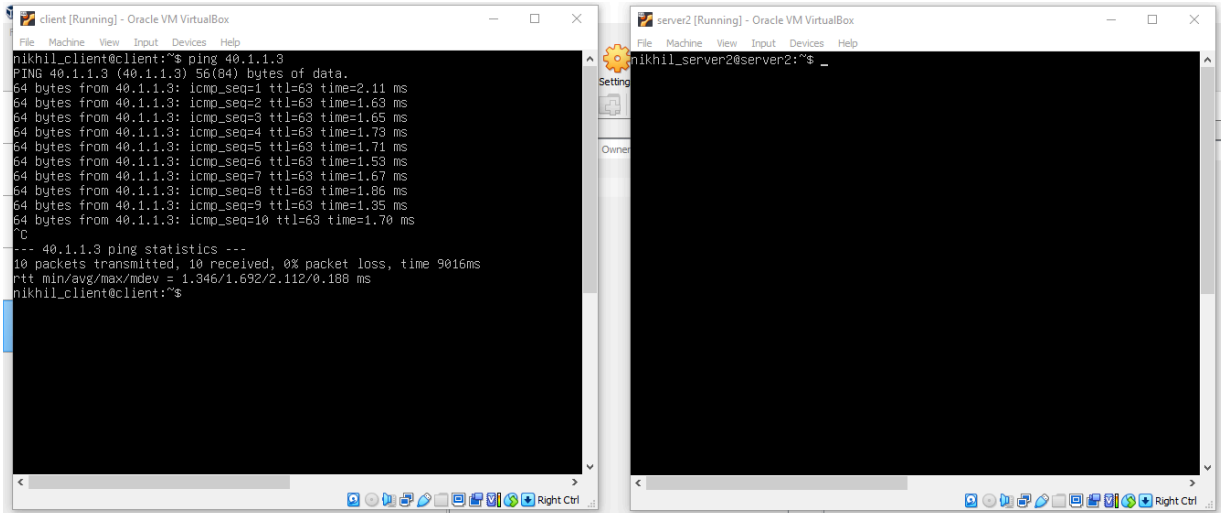
Tests after applying the traffic filtering commands:-

Test for ping on server1 and server2:-

Server1:-



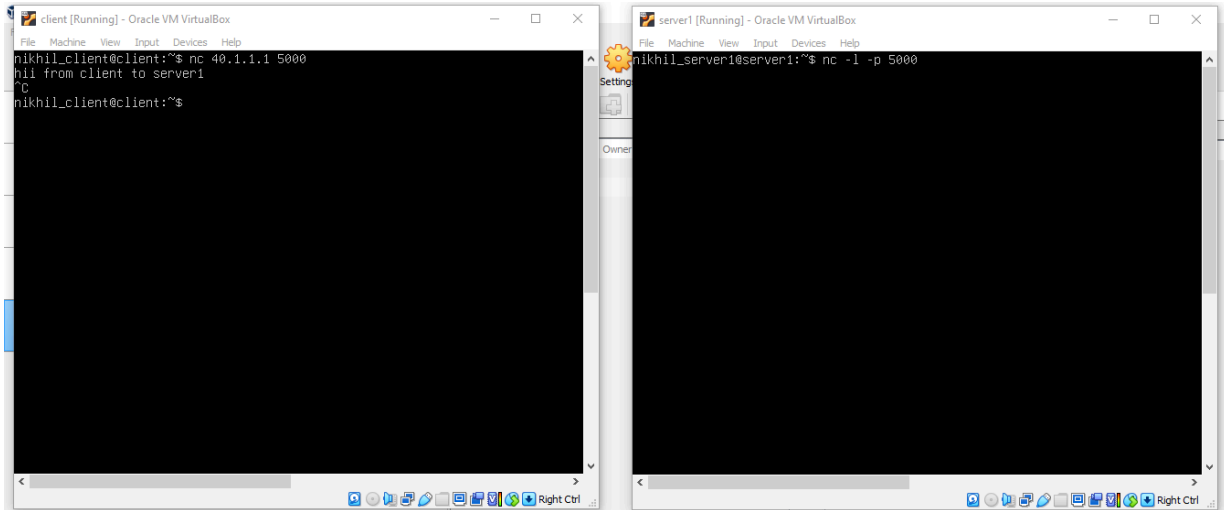
Server2:-



From these, it is clear that the ping worked for both the servers.

Test for TCP requests:-

Server1:-



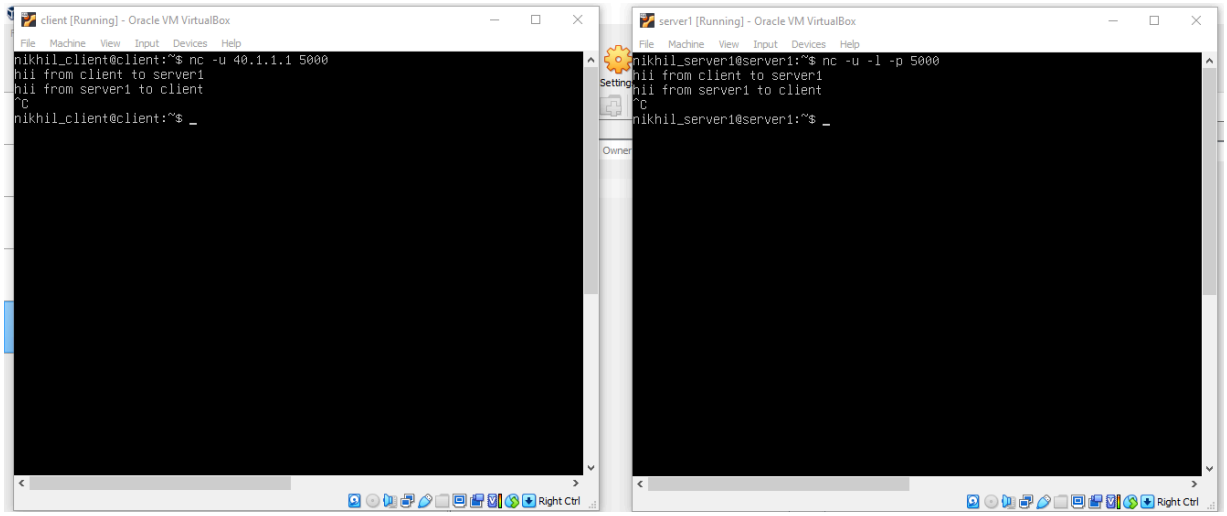
Here the TCP connection between client and server1 fails, which proves that traffic was filtered at the gateway and only the TCP traffic was blocked which was generated by client.

Server2:-

Here the TCP connection between client and server2 fails, which proves that traffic was filtered at the gateway and only the TCP traffic was blocked which was generated by client.

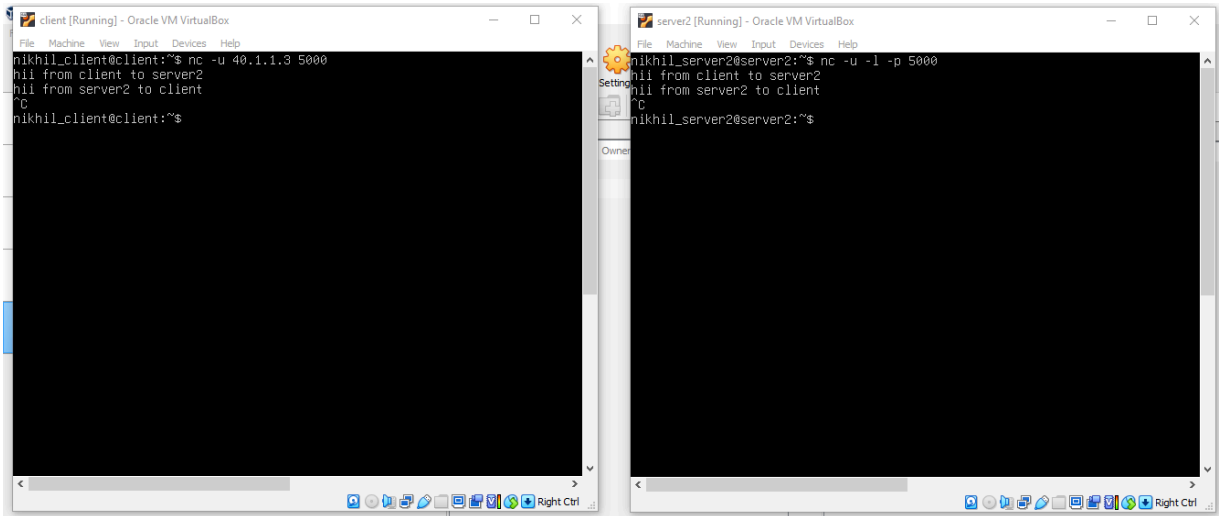
<u>Test for UDP requests:-</u>

Server1:-



Here the UDP connection between client and server1 is successful, which proves that traffic was filtered at the gateway and only the TCP traffic is getting blocked, and this time it was a UDP connection and hence was successful.

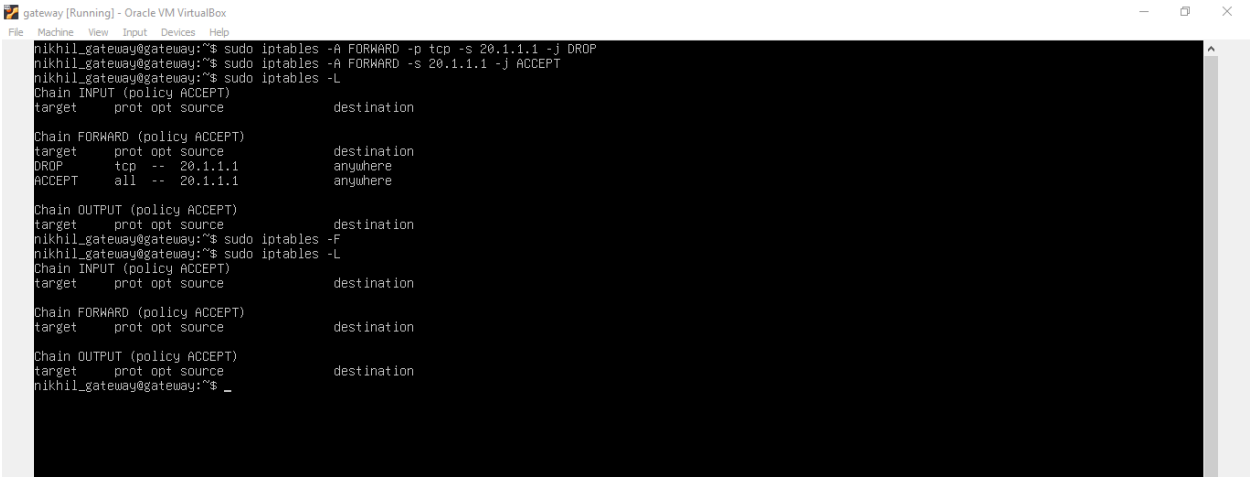
Server2:-



Here the UDP connection between client and server2 is successful, which proves that traffic was filtered at the gateway and only the TCP traffic is getting blocked, and this time it was a UDP connection and hence was successful.

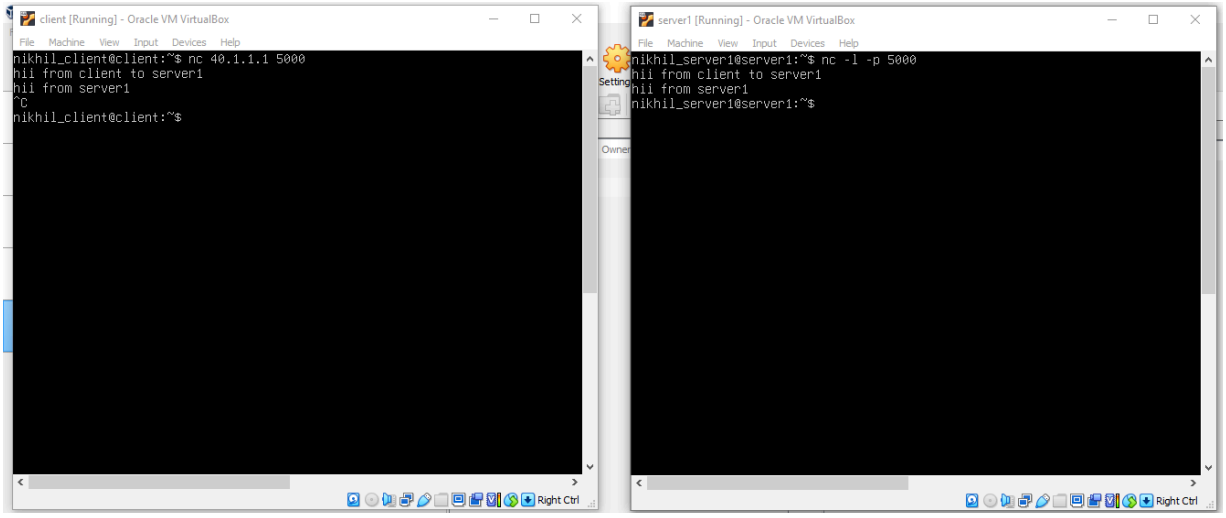
<u>Tests after removing the traffic filtering commands:-</u>

Now I removed the earlier iptable entries from the gateway using the command:
sudo iptables -F



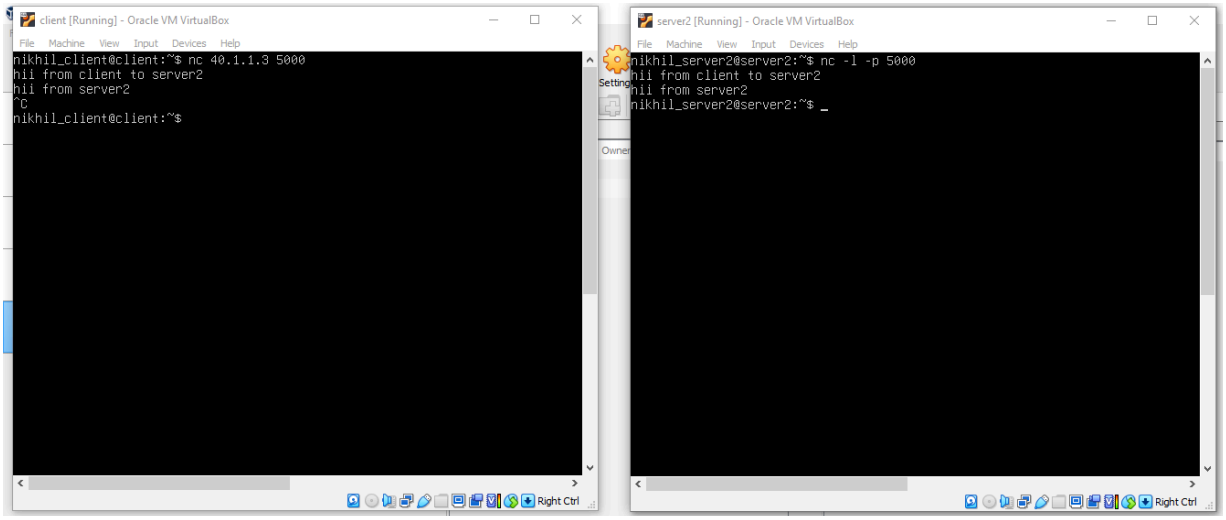
<u>Test for TCP requests:-</u>

Server1:-



After removing the entries from the iptables for gateway, TCP connection between the client and server1 was successful.

Server2:-



After removing the entries from the iptables for gateway, TCP connection between the client and server2 was successful.

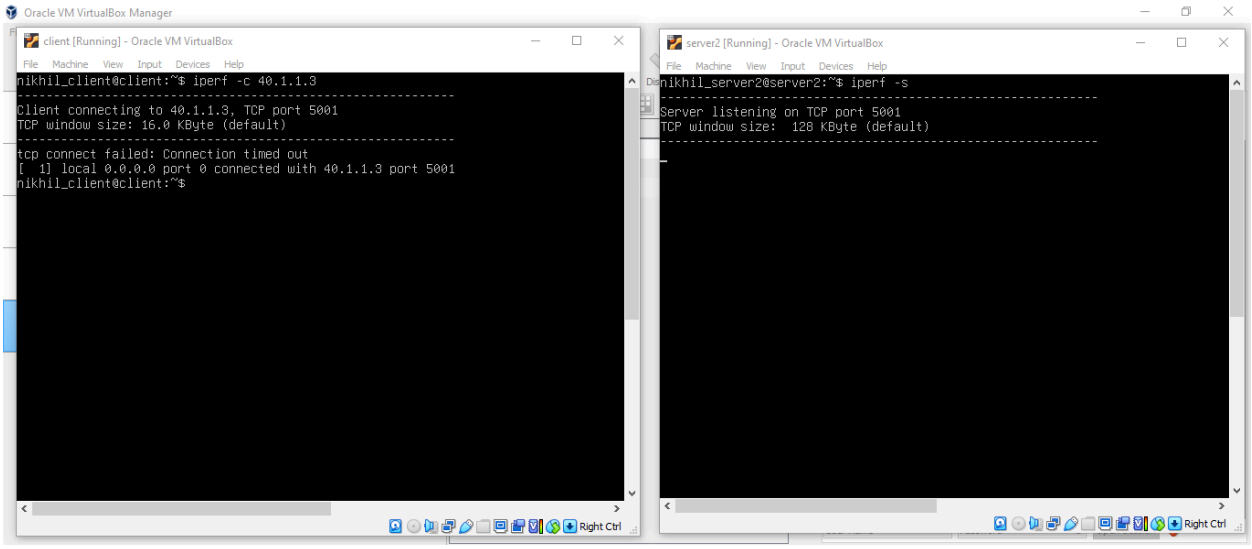
Both of these tests, after adding traffic filtering using iptables and after removing them from iptable at the gateway, clearly show that traffic filtering was successful and we were successfully able to block only the TCP traffic which was generated by the client-20.1.1.1/24.

3) a) For TCP:-

When I ran the iperf tests between the client (20.1.1.1) and server2 (40.1.1.3) for TCP, then it was not able to connect successfully, this is because, as we are using the same configuration obtained in q2, and hence there we have blocked the TCP connections originating from the client (20.1.1.1) by the command “sudo iptables -A FORWARD -p tcp -s 20.1.1.1 -j DROP”. Following is the screenshot for the same:-

Command used at server: iperf -s

Command used at client: iperf -c 40.1.1.3



For UDP:-

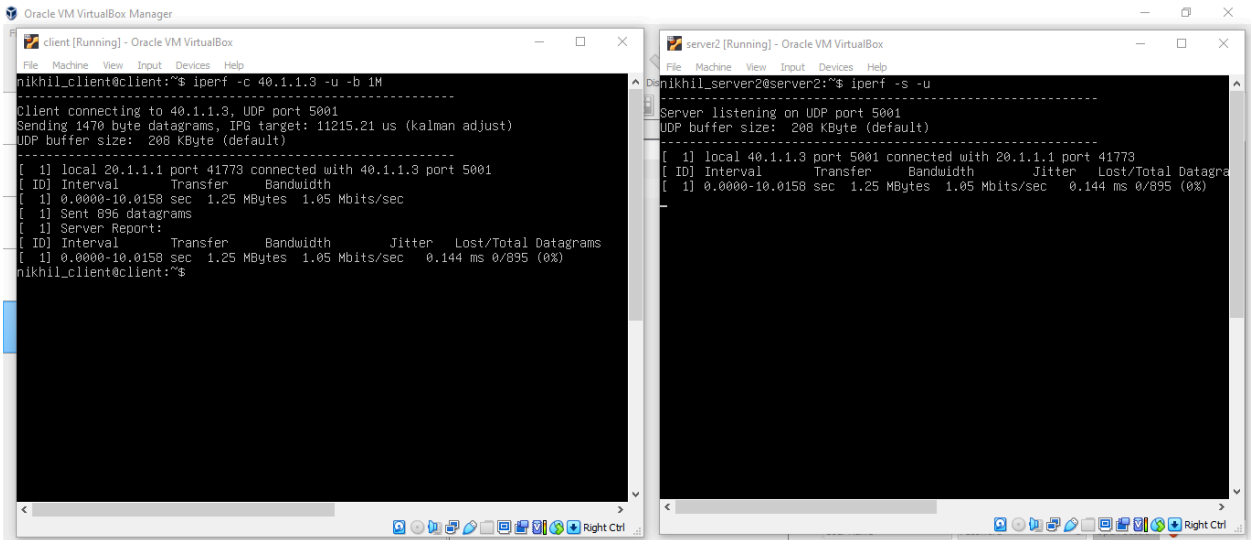
I did a lot 2 tests for UDP connections between client (20.1.1.1) and server2 (40.1.1.1). The following are these:-

<u>Tests for different bandwidths:-</u>

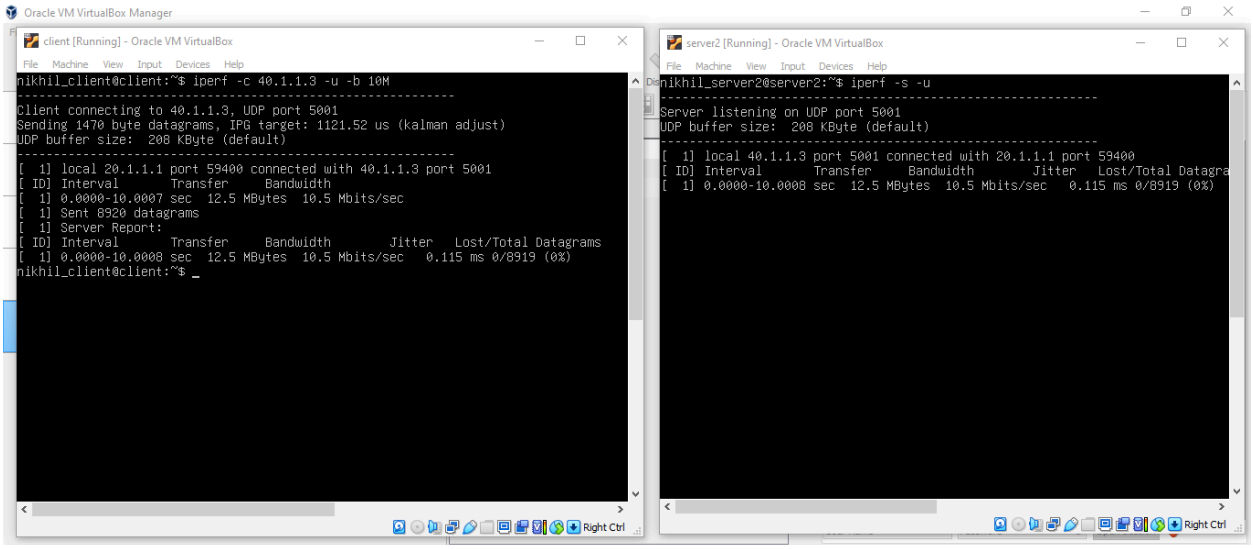
i) For 1Mbits/sec

Command used at server: iperf -s -u

Command used at client: iperf -c 40.1.1.3 -u -b 1M

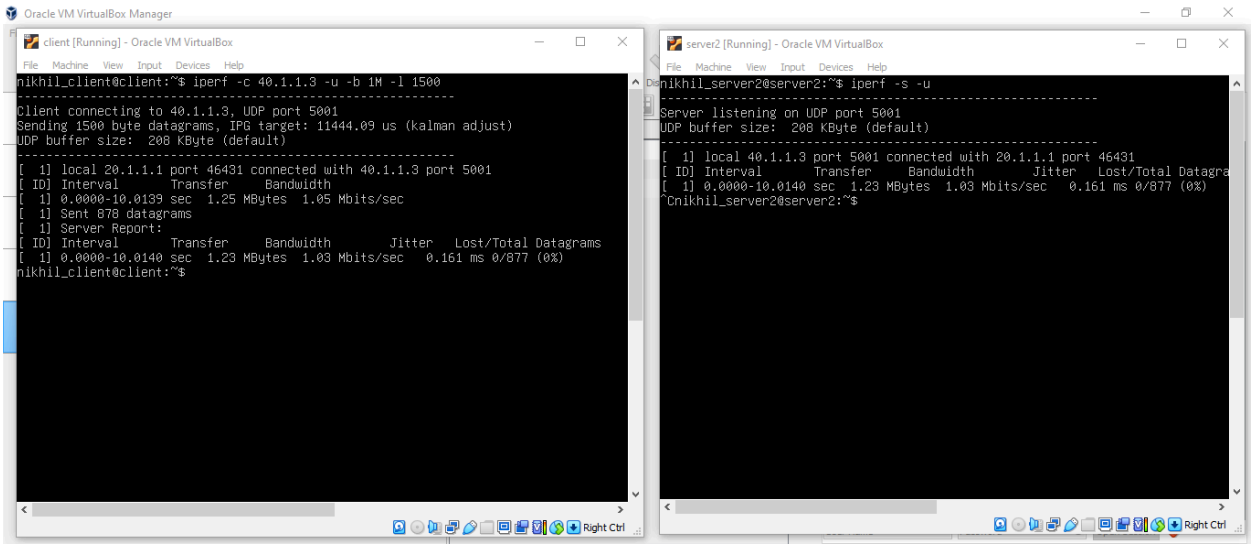


ii) For 10Mbits/sec
Command used at server: iperf -s -u
Command used at client: iperf -c 40.1.1.3 -u -b 10M

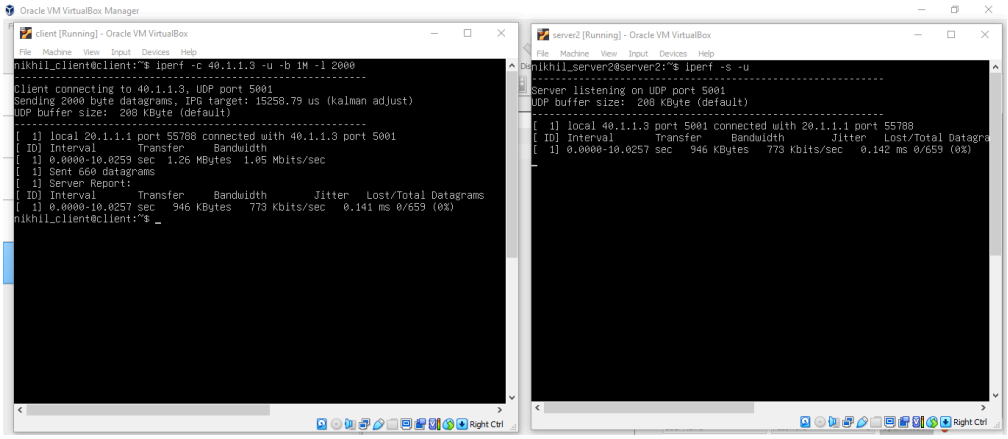


Interpretation:-
In both cases the target bandwidth was reached with .05 more for 1Mbps and 0.5 more for 10Mbps with 0% data loss between client and server. Jitter was almost the same for both cases. The more bandwidth achieved may be because of, iperf tries to achieve the target bandwidth by sending packets at calculated intervals. However, due to small variations in network timing (like packet scheduling or network congestion), packets may arrive at slightly different rates than expected.

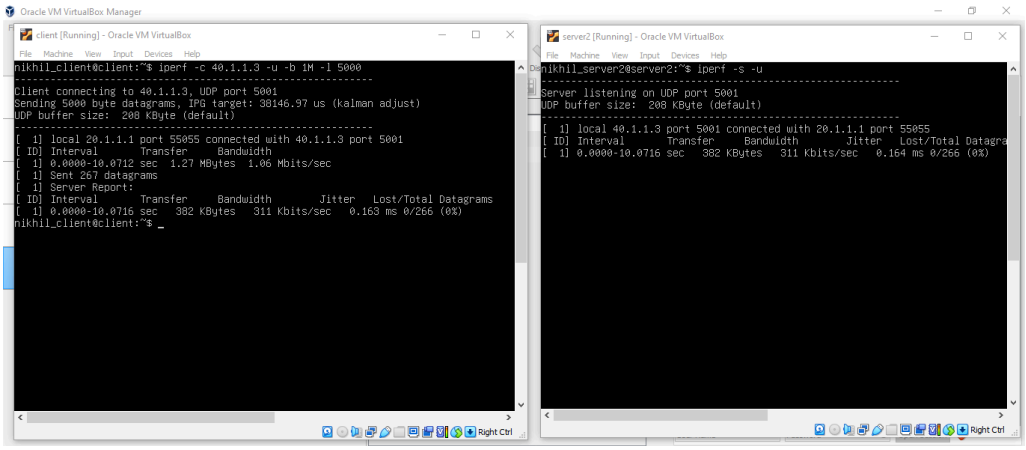
<u>Tests for different packet sizes with the same bandwidth:-</u>
i) For 1500 bytes size:-



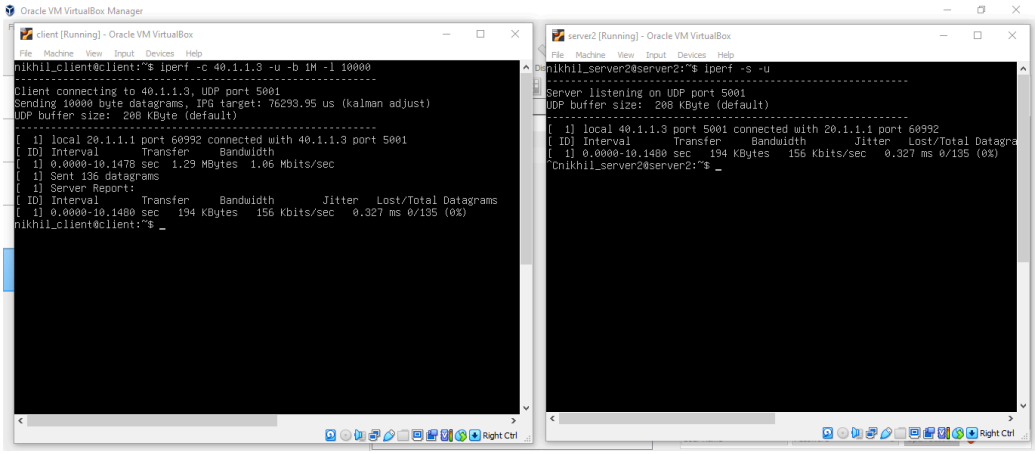
ii) For 2000 bytes size:-



iii) For 5000 bytes size:-



iv) For 10000 bytes size:-

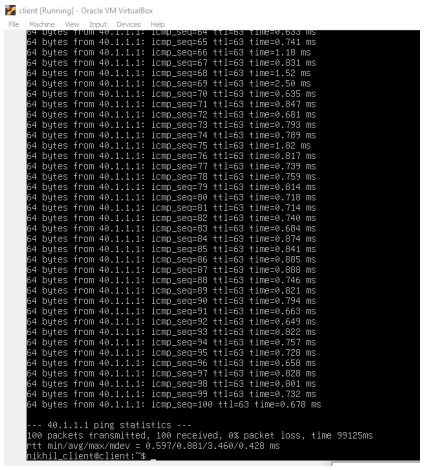


Packet size in Bytes	Bandwidth in Mbps	Jitter in ms
1500	1.03	0.161
2000	0.731	0.141
5000	0.311	0.163
10000	0.156	0.327

Interpretation:-

1. Bandwidth decreases with increasing packet size because larger packets are harder to process, more likely to encounter fragmentation, cause more congestion, and require more buffer space. This reduces the efficiency of the network, causing a lower effective throughput.
2. Jitter tends to increase for larger packets as network delays become more variable, especially when the network struggles to keep up with the large data load.

b) i) Following is the data for the same using ping 40.1.1.1 -c 100



Minimum RTT: 0.597 ms
Maximum RTT: 3.460 ms
Avg RTT: 0.881 ms

ii) Following is the data for the same using ping 40.1.1.3 -c 100

Minimum RTT: 0.649 ms
Maximum RTT: 8.610 ms
Avg RTT: 0.922 ms

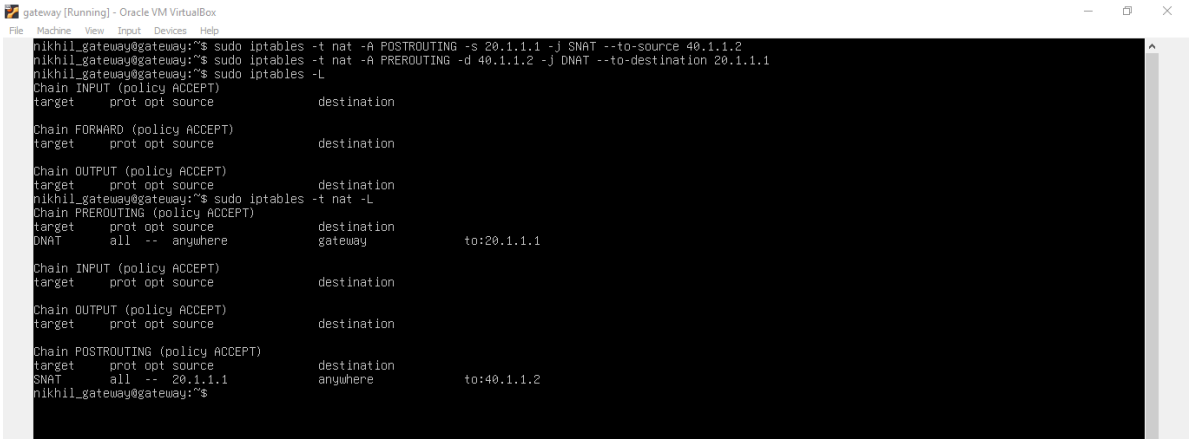
iii) There was no such “significant” difference between the two, but the maximum RTT for 40.1.1.3 was almost 3 times higher than 40.1.1.1. But the average RTT was almost the same, reasons for this are:-

1. Similar network path: Both IP addresses (40.1.1.1 and 40.1.1.3) are on the same subnet, meaning they follow the same route through the network infrastructure, resulting in similar round-trip times.
2. Same physical location: Both the IPs are hosted on VMs within the same physical machine, the RTT will be nearly identical because the physical distance between the source and destination is almost the same.

4) a) Command used: sudo iptables -t nat -A POSTROUTING -s 20.1.1.1/24 -j SNAT --to-source 40.1.1.2

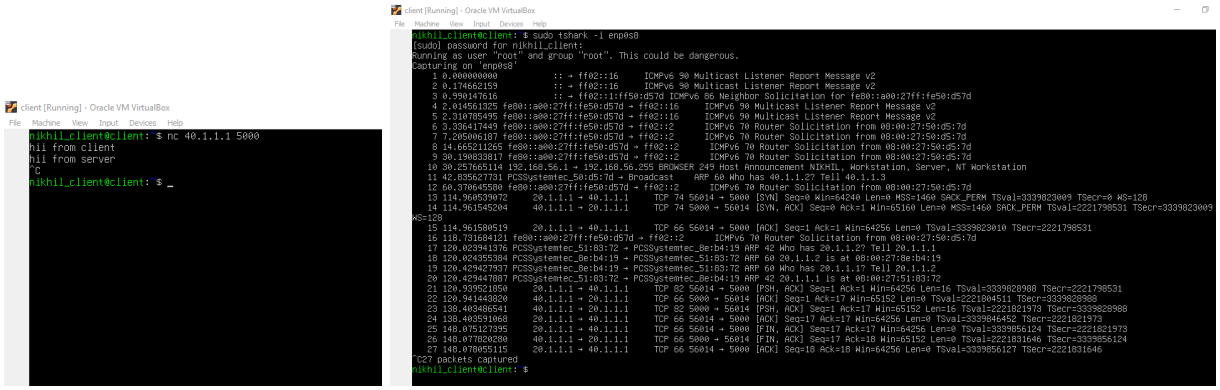
b) Command used: sudo iptables -t nat -A PREROUTING -d 40.1.1.2 -j DNAT --to-destination 20.1.1.1

Below is the screenshot for these commands on gateway:-



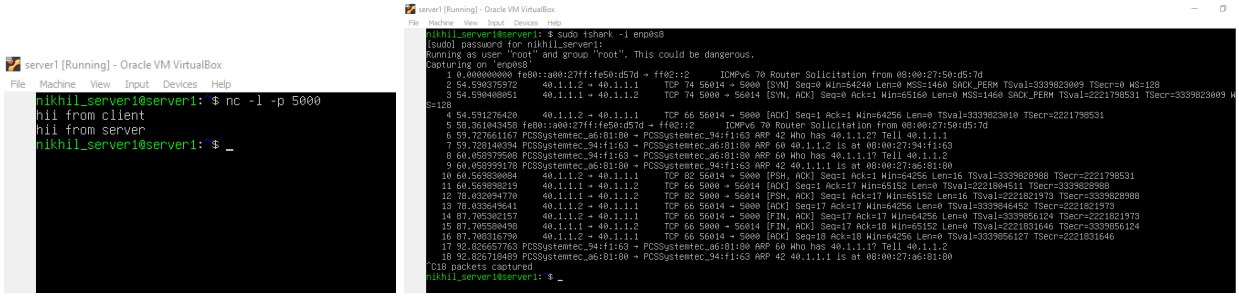
c) Following is the traffic observed at each network interface using tcpdump:-

i) At Client's Network Interface



This shows that TCP connection between client and server1 was successful. And the right side image shows that client (20.1.1.1) is communicating with server1 (40.1.1.1), here the source and destination packet address didn't change, which is expected, as this change should only happen on gateway network interfaces.

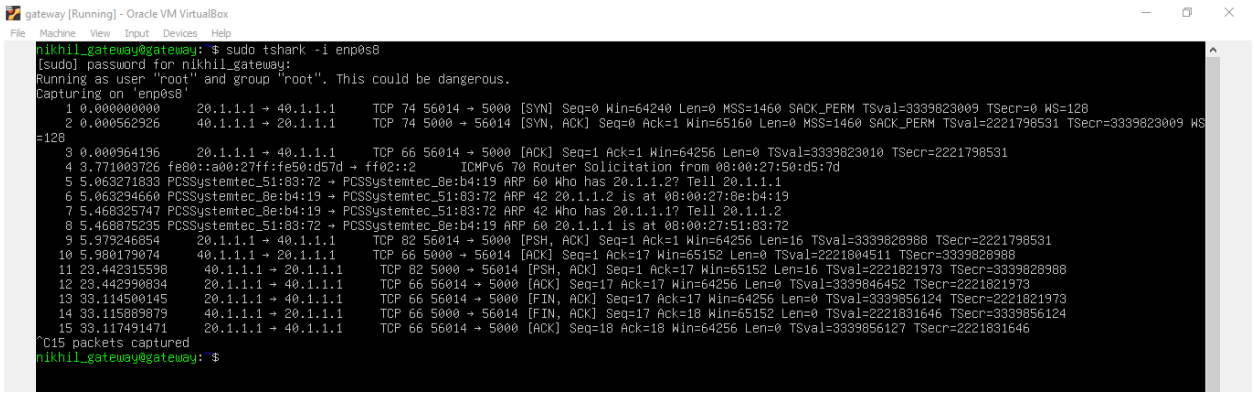
ii) At Server1's Network Interface



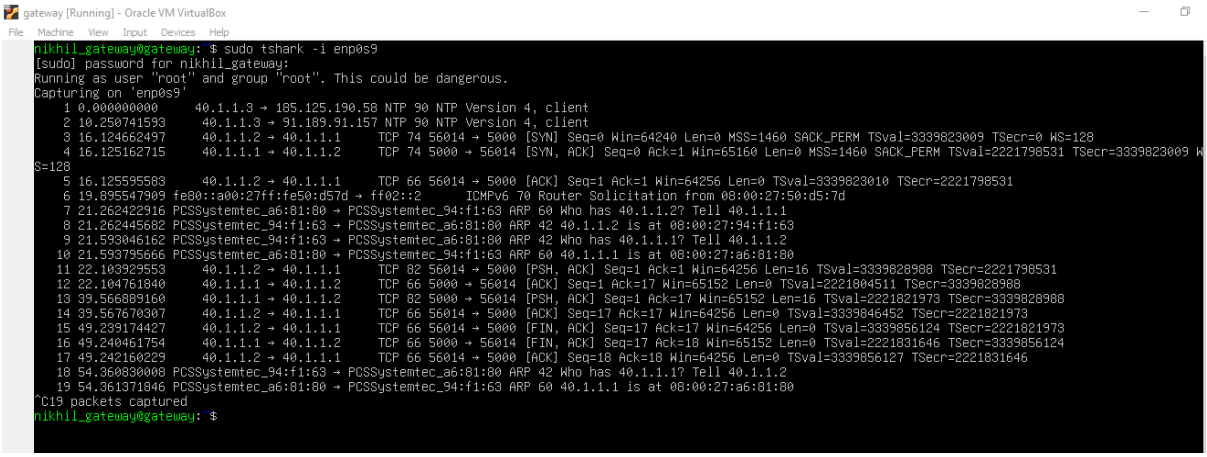
Left side image confirms that TCP connection between client and server1 was successful. Now, the right side image shows the source and destination packet address, here the connection is between 40.1.1.1 and 40.1.1.2, which means that it is between server1 and gateway's server-side network interface, and thus it proves that the source IP address was changed to 40.1.1.2 when packet was sent by the client.

iii) At Gateway's Network Interface

Client-side Interface



Server side Interface



From these images it is clear that source IP was changed to “40.1.1.2” and then after the response it was again reverted back to the original source IP address.

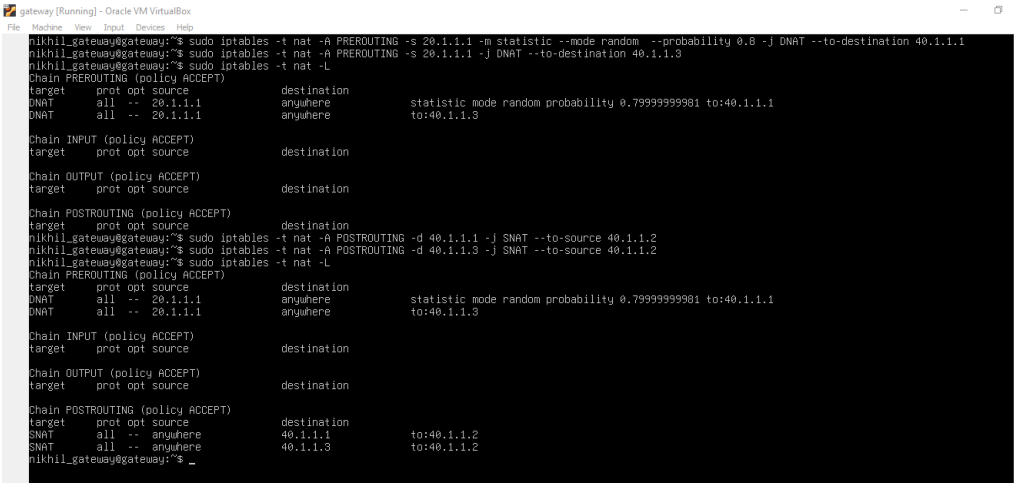
5) From q3 (b), both the servers have almost the same average RTT values with server1 having a slightly less time, and hence, we assign a probability of 0.8 to the server1 for load balancing.
Command used: -

1. sudo iptables -t nat -A PREROUTING -s 20.1.1.1/24 -m statistic --mode random --probability 0.8 -j DNAT --to-destination 40.1.1.1
2. sudo iptables -t nat -A PREROUTING -s 20.1.1.1/24 -j DNAT --to-destination 40.1.1.3

Here the first command routes 80% of the traffic to server1 (40.1.1.1) and second command assigns the remaining 20% of the traffic to server2 (40.1.1.3).
After this, we need to ensure that responses return through the gateway and hence we set up the source NAT:-

1. sudo iptables -t nat -A POSTROUTING -d 40.1.1.1/24 -j SNAT --to-source 40.1.1.2
2. sudo iptables -t nat -A POSTROUTING -d 40.1.1.3/24 -j SNAT --to-source 40.1.1.2

These will ensure that both the servers (40.1.1.1 and 40.1.1.3) send responses back through the gateway.



Now, to test it, we established a series of TCP connections between the client and the gateway's client interface (20.1.1.2) and then sent some messages to the 20.1.1.2, but some of these client connections get connected to the server1 and some to server2, with the majority being connected to the server1 as shown in the below image:-