CN Assignment-3

Shobhit Raj (2022482)

For this assignment, we have to setup this network structure [fig. 1]:

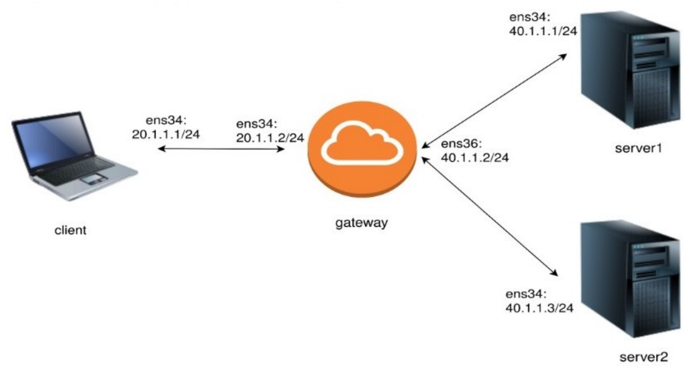


Figure 1: Network Diagram

1 Question 1

In this setup, four virtual machines (VMs) were created using VirtualBox, each configured with 1 CPU core, 512 MB of RAM, and 20 GB of hard disk space. The operating system installed on all VMs was Debian (64-bit). The VMs were named `client`, `gateway`, `server1`, and `server2`.

Two host-only networks were configured within VirtualBox:

- **Host-Only Ethernet Adapter** – IP range: 20.1.1.0/24
- **Host-Only Ethernet Adapter #2** – IP range: 40.1.1.0/24

In both networks, DHCP was disabled.

Each VM was connected to the networks as follows:

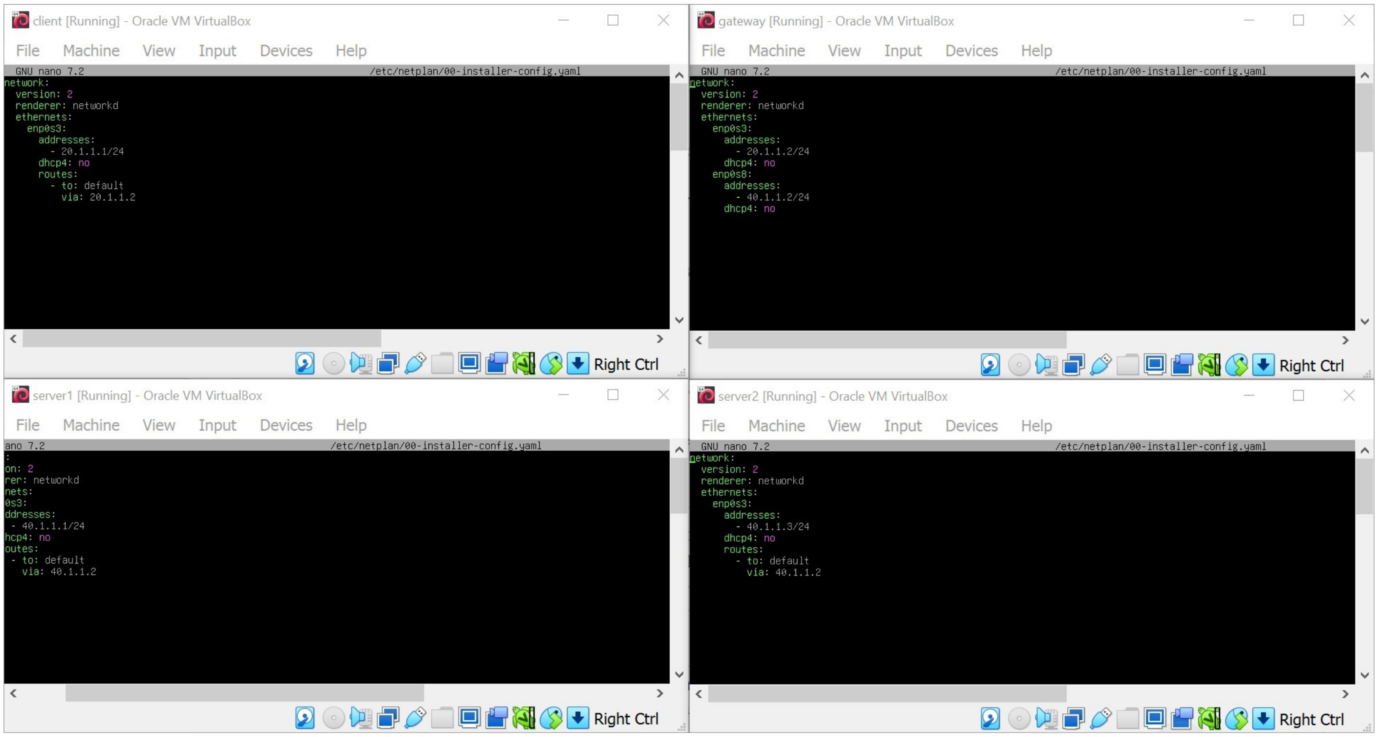
- The `client` VM was connected to *Host-Only Ethernet Adapter* using interface `enp0s3`.
- The `gateway` VM had two network interfaces: `enp0s3` – connected to the network 20.1.1.0/24 and `enp0s8` – connected to the network 40.1.1.0/24
- Both `server1` and `server2` VMs were attached to *Host-Only Ethernet Adapter #2* through `enp0s3`.

This multi-network configuration facilitated isolated communication between the VMs, with the gateway acting as an intermediary between the two networks.

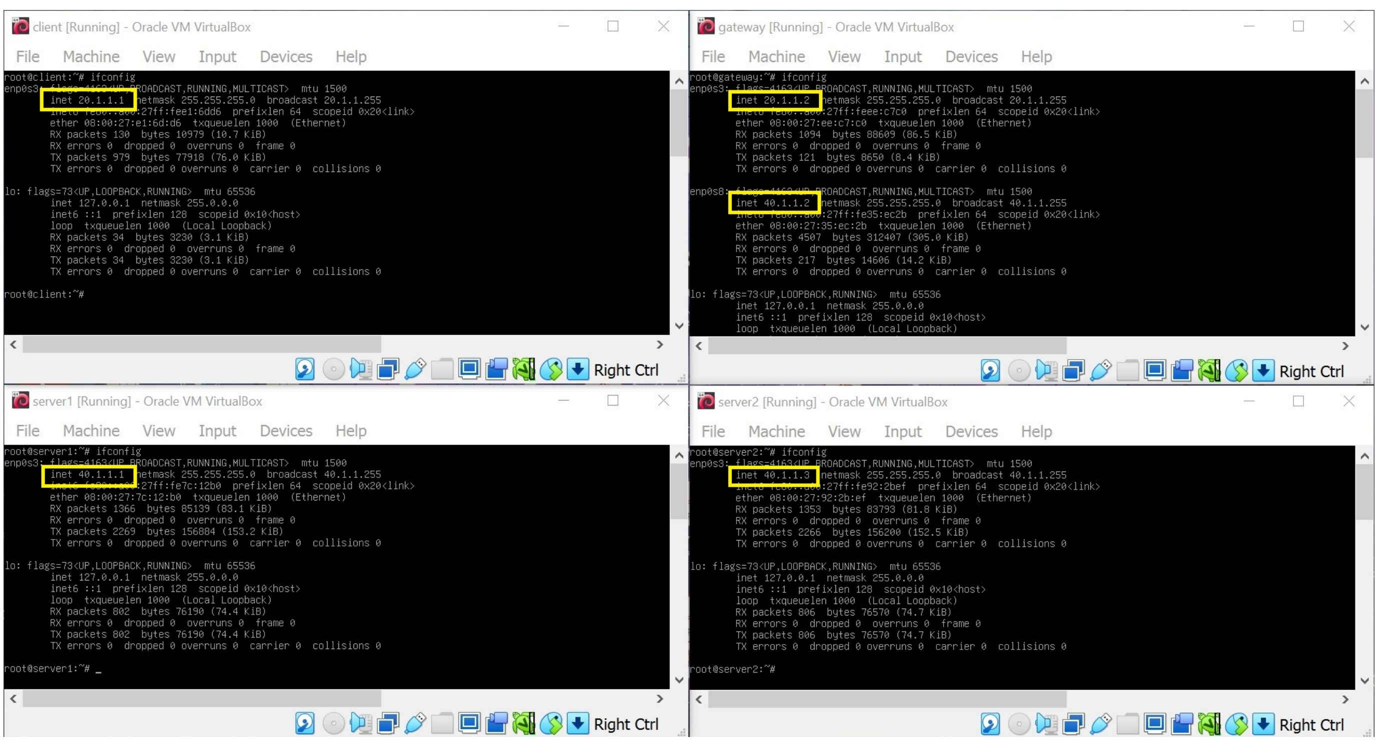
1.1 a)

In this question, I configured the IP addresses and routing for all four VMs by modifying the Netplan YAML configuration files. Using the command `nano /etc/netplan/00-installer-config.yaml`, I edited the files on each VM to set up the network interfaces, assigning the specified IP addresses and routes [fig. 2 (a)]. The `addresses` field was used to specify each IP address and subnet, while the `routes` field was utilized to define the default gateway for proper routing. This configuration ensured that each VM adhered to the network structure presented in the assignment diagram.

After saving the changes in the YAML files, I applied the configurations using the `sudo netplan apply` command to activate the new settings. To ensure the networking services were fully updated, I executed `systemctl restart networking.service` on each VM. Finally, I verified the network configuration changes by running the `ifconfig` command on all four VMs, confirming [fig. 2 (b)] that the IP addresses and routes matched the assignment requirements.



(a) netplan YAML configuration files



(b) ifconfig to verify the IP addresses

Figure 2: Configuration and Verification Steps

1.2 b)

To enable IP forwarding on the gateway, I used this command on the `gateway`:

```
/sbin/sysctl -w net.ipv4.ip_forward=1
```

This command sets a system parameter that controls IP forwarding. Here, `/sbin/sysctl` is the utility used to modify kernel parameters at runtime, `-w` is a flag that specifies writing (setting) a parameter, and `net.ipv4.ip_forward` is the parameter name that controls whether the kernel forwards IPv4 traffic. Setting this to `=1` enables IP forwarding, allowing the gateway to forward packets between interfaces.

To verify that IP forwarding was functioning correctly, I tested connectivity from the client VM to server1 using the following ping command on `client`:

```
ping -c 3 40.1.1.1
```

Before enabling IP forwarding, the `ping` command did not receive replies (fails), indicating that the packets were not being routed. After running the command to enable IP forwarding, pinging `40.1.1.1` resulted in successful replies, confirming that the gateway was correctly forwarding traffic between the networks [fig. 3].

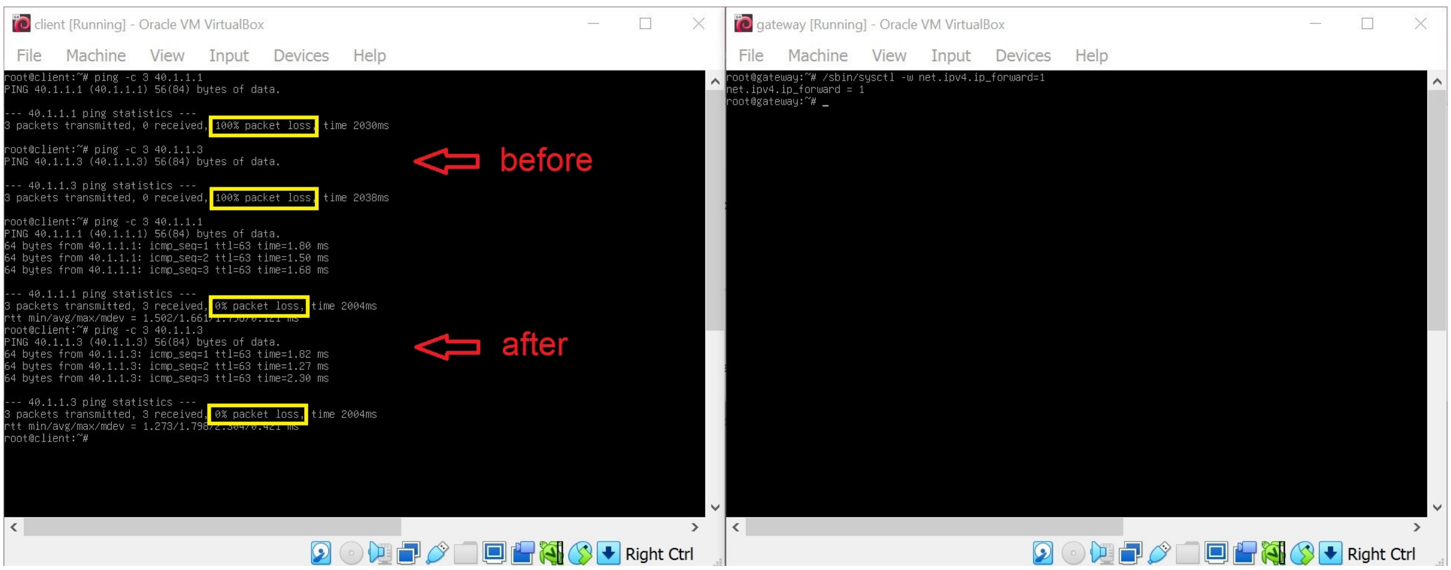


Figure 3: Setting up Forwarding functionality

2 Question 2

2.1 a)

For the gateway to block all traffic (except for ping) destined to the server 40.1.1.1/24, I used these commands on the `gateway`:

```
/sbin/iptables -A FORWARD -d 40.1.1.1 -p icmp -j ACCEPT
/sbin/iptables -A FORWARD -d 40.1.1.1 -j DROP
```

Here, `iptables` is the utility used for managing Linux kernel packet filtering rules. The flag `-A FORWARD` adds a rule to the `FORWARD` chain, which is used for packets routed through the gateway. The option `-d 40.1.1.1` specifies that the rule applies to packets with a destination IP address within the range 40.1.1.1/24 (the `server1` network).

In the first command, `-p icmp` indicates that the rule applies to ICMP protocol traffic (used by `ping`), and `-j ACCEPT` accepts packets that match the conditions. In the second command, `-j DROP` drops all other traffic destined for 40.1.1.1.

The first command allows ping traffic (ICMP) to reach the server. The second command drops all other types of traffic destined for the server network, effectively blocking everything except ping.

To check if the rule works, I used the ping command on the client VM, which replied successfully, indicating that ICMP traffic is allowed. But, when used netcat to setup connection between client and server1, it failed, confirming that the traffic filtering is in place [fig. 4].

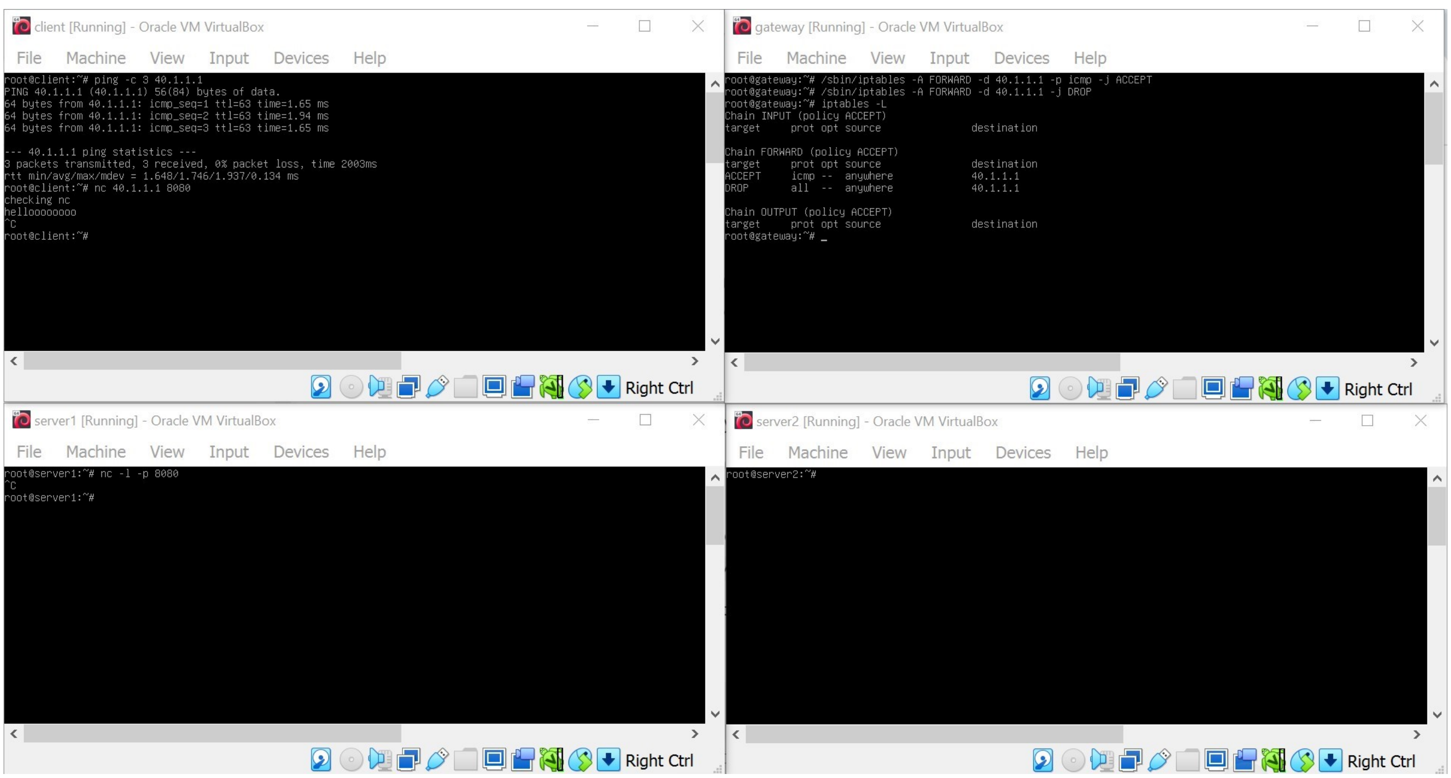


Figure 4: Ping is working but netcat failed

2.2 b)

For the gateway to block only TCP traffic initiated by 20.1.1.1/24 (client), I used this command on the `gateway`:

```
/sbin/iptables -A FORWARD -s 20.1.1.1 -p tcp -j DROP
```

Here, `iptables` is the utility used for managing Linux kernel packet filtering rules. The flag `-A FORWARD` adds a rule to the `FORWARD` chain, which is used for packets routed through the gateway. The option `-s 20.1.1.1` Specifies the source address, meaning this rule applies to traffic originating from the IP range 20.1.1.1/24 (the `client` network), `-p tcp` indicates that the rule applies to TCP protocol traffic and `-j DROP` drops the matching packets, thereby blocking TCP traffic.

This command blocks all TCP traffic initiated from the client network (20.1.1.1/24). It won't affect other types of traffic (e.g., ICMP or UDP), ensuring that only TCP packets are filtered. To check if it is working, I used netcat to create listening port on server2 (as server1 is blocked for all traffic) and setup TCP and UDP connection from client [fig. 5]. The TCP connection failed verifying iptables rules is effectively blocking TCP traffic initiated by the client network, and UDP connection was successfully receiving messages, confirming that the UDP traffic was not affected by the iptables rule.

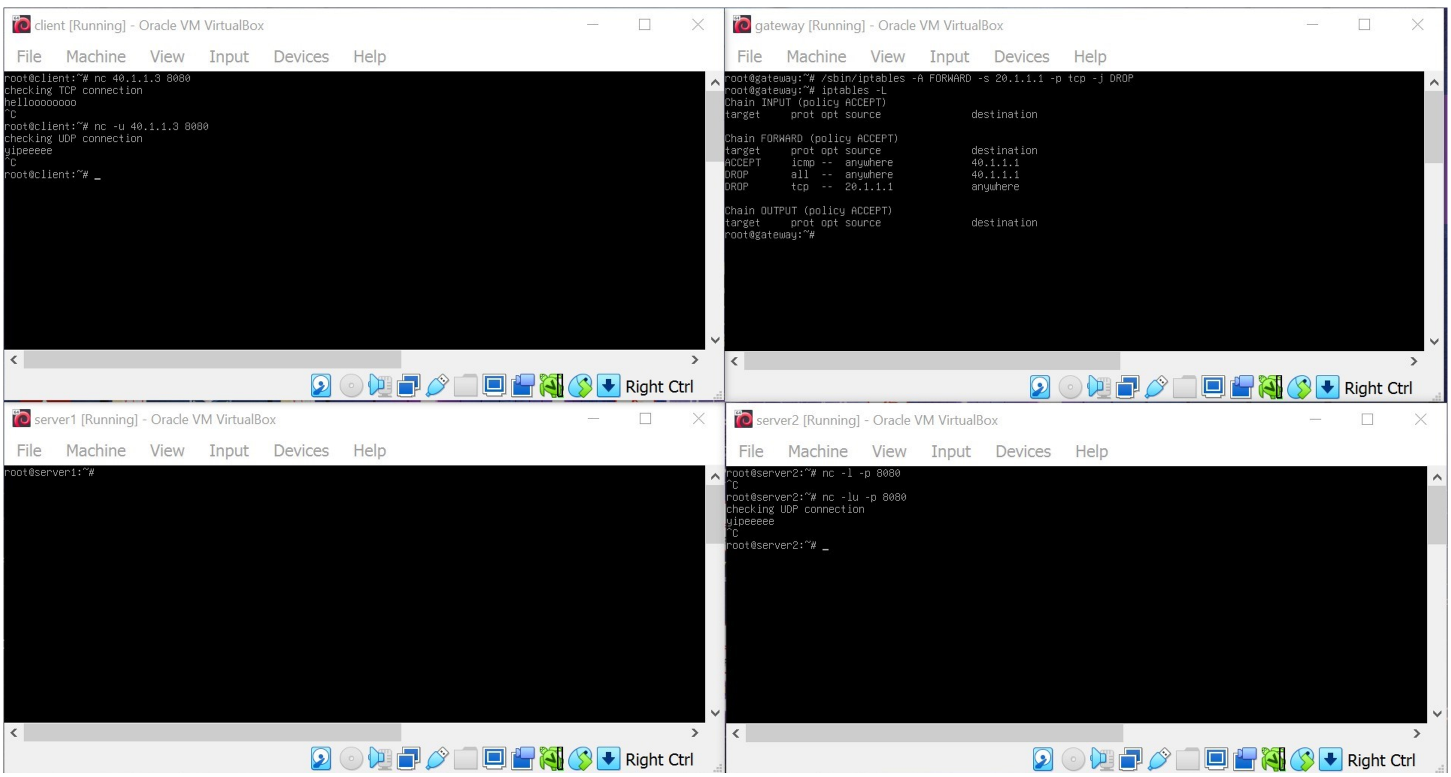


Figure 5: TCP connection failed but UDP working

3 Question 3

3.1 a)

To test the TCP and UDP Bandwidth using iperf between client and server2,

I ran these commands on the `server2`:

```
For TCP : iperf -s
For UDP : iperf -s -u
```

I ran these commands on the `client`:

```
For TCP : iperf -c 40.1.1.3
For UDP : iperf -c 40.1.1.3 -u
```

Here, `iperf` is the tool used for network performance measurement. The flag `-s` starts the iperf server mode, `-c` runs the client mode, connecting to the specified server IP and `-u` indicates UDP mode.

The commands above measure the bandwidth between the client (20.1.1.1) and Server2 (40.1.1.3) for both TCP and UDP traffic, showing the achievable data transfer rates under the current network configuration.

For TCP, no results were obtained as TCP traffic was blocked in the previous question from client as source. This prevents the establishment of TCP connection in this case. For UDP, successful connection is set up and the bandwidth measured is `1.05 Mbits/sec` [fig. 6].

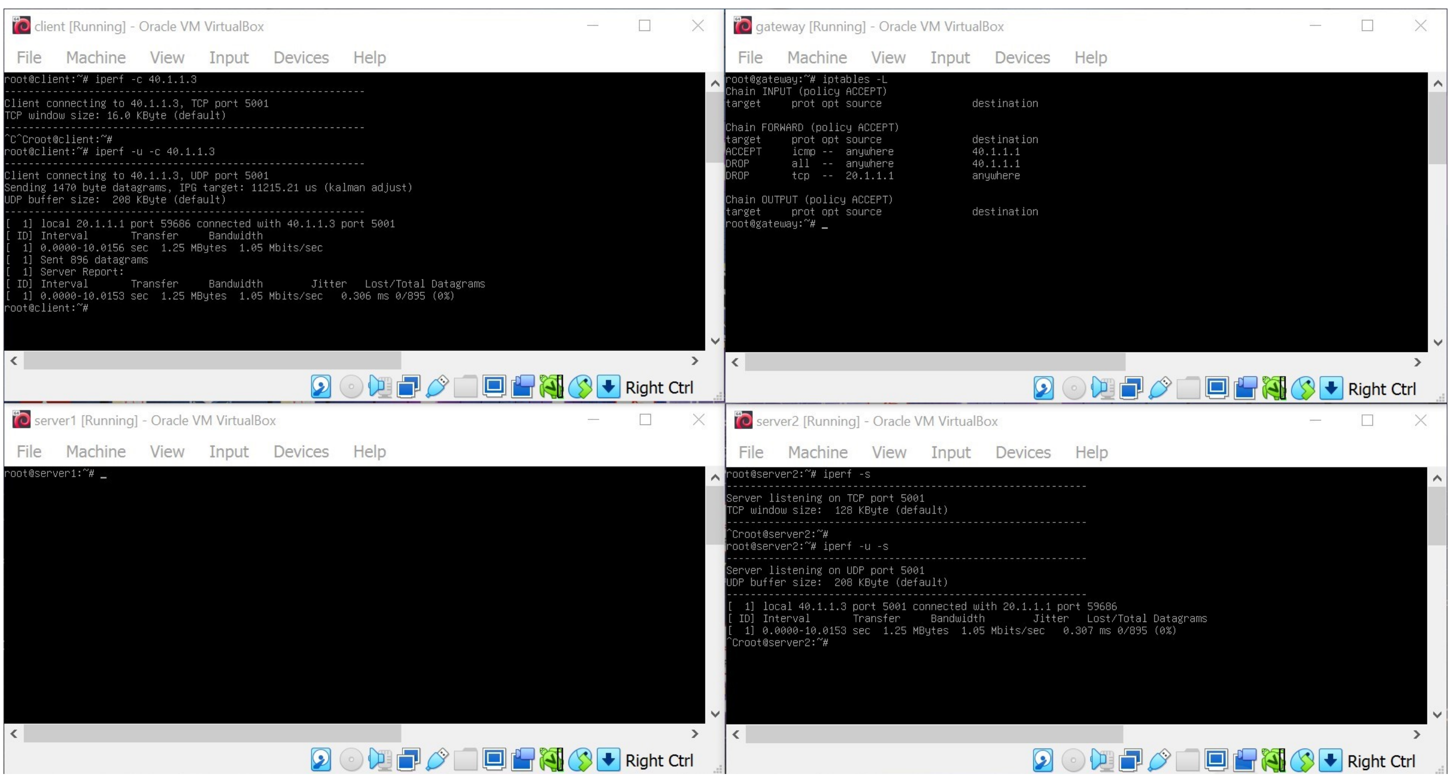


Figure 6: Bandwidth measurement using iperf

3.2 b)

Here, `ping` is the tool for testing network connectivity and measuring RTT. The ping commands measure the round-trip time for packets sent from the client to each server, providing insights into network latency. The results give minimum, average, and maximum RTT values for comparison.

3.2.1 (i)

To measure RTT from 20.1.1.1 to 40.1.1.1 (server1), I used this command on `client` [fig. 7]:

```
ping -c 10 40.1.1.1
```

- **Minimum RTT:** 1.248 ms
- **Average RTT:** 1.455 ms
- **Maximum RTT:** 1.652 ms

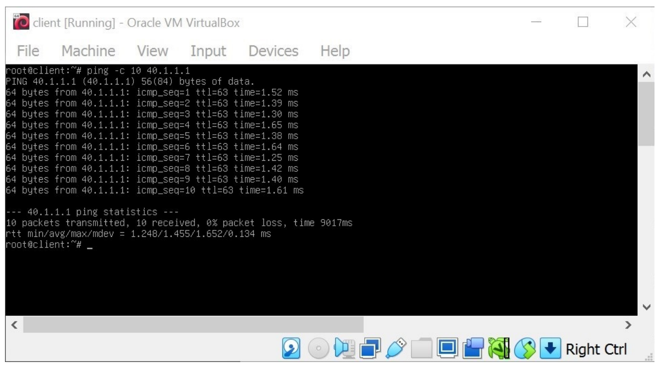


Figure 7: pinging server1

3.2.2 (ii)

To measure RTT from 20.1.1.1 to 40.1.1.3 (server2), I used this command on `client` [fig. 8]:

```
ping -c 10 40.1.1.3
```

- **Minimum RTT:** 1.287 ms
- **Average RTT:** 1.447 ms
- **Maximum RTT:** 1.735 ms

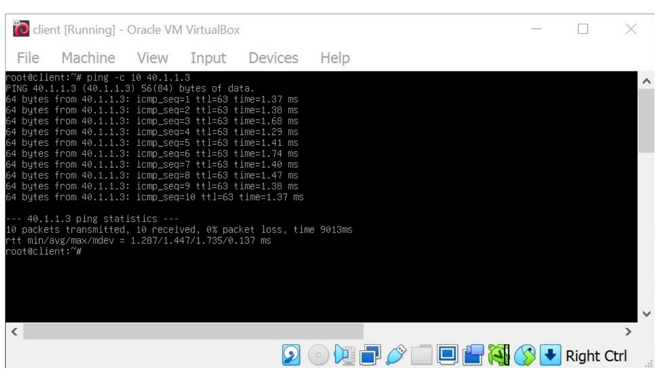


Figure 8: pinging server2

3.2.3 (iii)

There is not significant difference between the minimum, average and maximum RTTs of the 2 servers because both servers are connected to the same network segment (40.1.1.0/24) and are accessed through the same gateway (20.1.1.1). Since the network path from the client VM to both servers is essentially identical, with no additional hops or differing link characteristics (no filtering or rule blocking on icmp packets), the latency remains similar for both destinations.

The average RTT of server2 (1.447 ms) is slightly lower than that of server1 (1.455 ms) likely due to the heavier use of iptables rules applied on 40.1.1.1, resulting in slightly increased processing time for packets destined for Server1. [This difference is used to decide the load balancing in question 5]

4 Question 4

The existing iptables rules were flushed using `iptables -F` so as to clear rules such as blocking tcp connections from source or any connection destined to server1, both of which are done in this question while using tcpdump.

4.1 a)

To change the source IP address of every packet from 20.1.1.1 to 40.1.1.2, I ran this command on the `gateway`:

```
/sbin/iptables -t nat -A POSTROUTING -s 20.1.1.1 -j SNAT --to-source 40.1.1.2
```

Here, `iptables` is the utility used for managing Linux kernel packet filtering rules, `-t nat` specifies that the command applies to the NAT table, `-A POSTROUTING` appends a rule to the POSTROUTING chain, which modifies packets after routing decisions have been made. The option `-s 20.1.1.1` specifies the source address, meaning this rule applies to traffic originating from the IP range 20.1.1.1/24 (the `client` network), `-j SNAT` indicates that Source NAT (SNAT) is being used to modify the source IP address of the packet and `--to-source 40.1.1.2` changes the source IP address of the packet to 40.1.1.2.

This command ensures that all packets originating from the client network (20.1.1.1) will have their source IP changed to 40.1.1.2 when they leave through the gateway.

4.2 b)

To revert the destination IP back to the original when the response arrives, I ran this command on the `gateway`:

```
/sbin/iptables -t nat -A PREROUTING -d 40.1.1.2 -j DNAT --to-destination 20.1.1.1
```

Here, `iptables` is the utility used for managing Linux kernel packet filtering rules, `-t nat` specifies that the command applies to the NAT table, `-A PREROUTING` appends a rule to the PREROUTING chain, which modifies packets before routing decisions are made. The option `-d 40.1.1.2` specifies that the rule applies to packets with the destination IP 40.1.1.2 (the address used for SNAT), `-j DNAT` indicates that Destination NAT (DNAT) is being used to modify the destination IP address of the packet and `--to-destination 20.1.1.1` changes the destination IP back to 20.1.1.1.

This ensures that when the response packets arrive at the gateway with a destination IP of 40.1.1.2, the gateway will revert the destination IP back to 20.1.1.1, so the client can receive the response correctly.

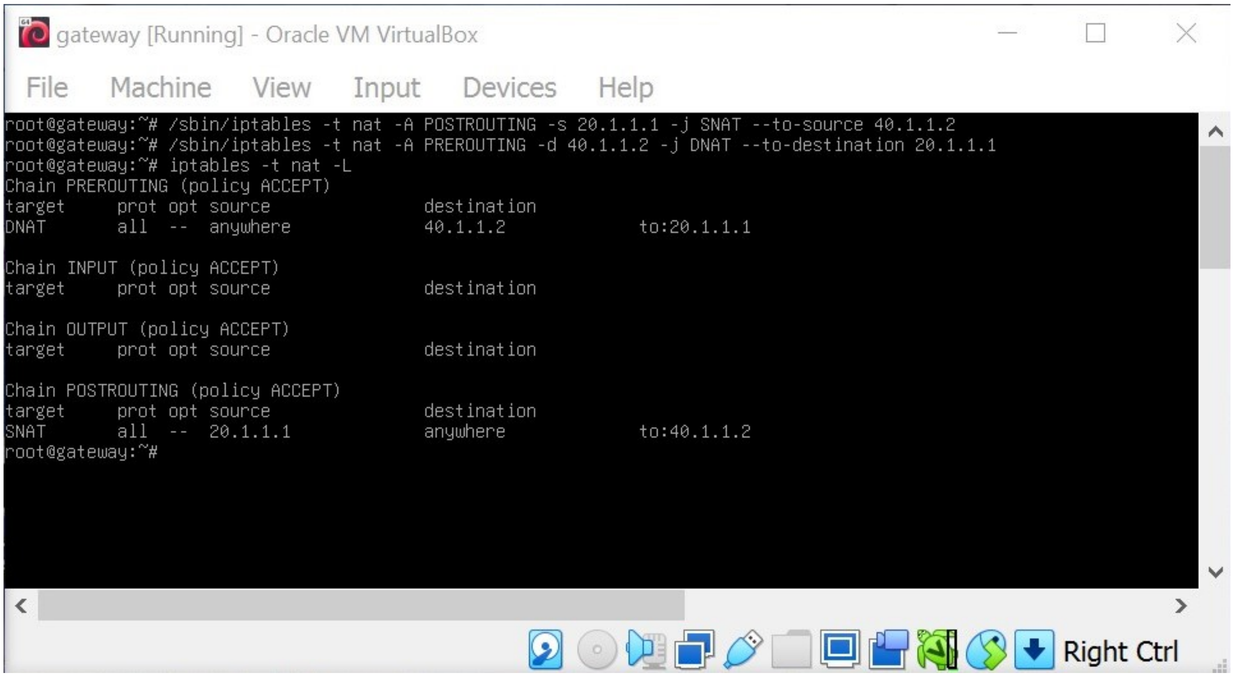


Figure 9: configuring NAT table

4.3 c)

To validate the Network Address Translation (NAT) setup configured in parts (a) and (b), I utilized tcpdump on the client, gateway, and server VMs to monitor network traffic and verify the changes. Additionally, I established a TCP connection between the client and server using nc (netcat) to generate traffic and observe the packet flow.

On the client, gateway, and server1, I initiated tcpdump to capture packets in the background:

```
tcpdump tcp &
```

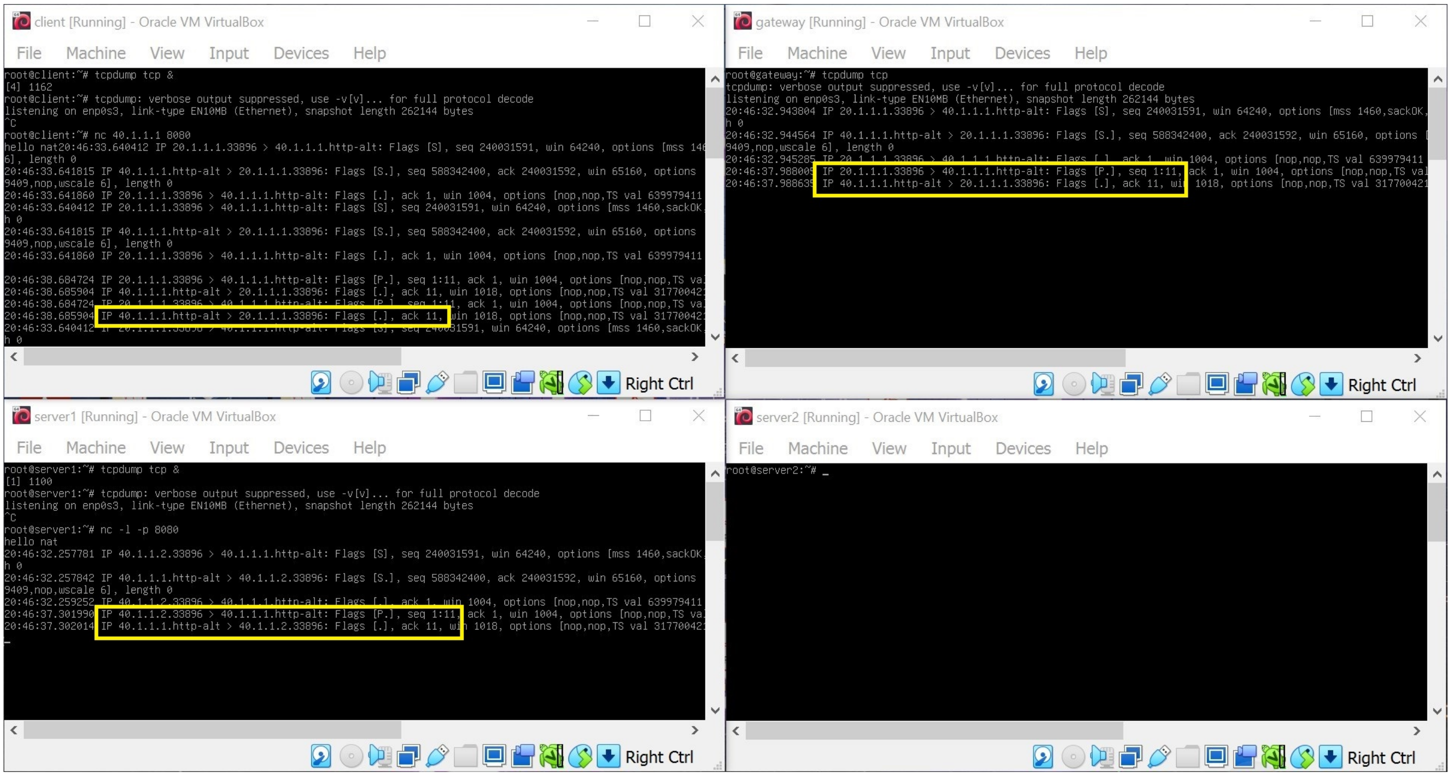
To generate traffic and test the NAT configuration, I used netcat (nc) to create a TCP connection between the client and server1:

```
On Server : nc -l -p 8080
On Client : nc 40.1.1.1 8080
```

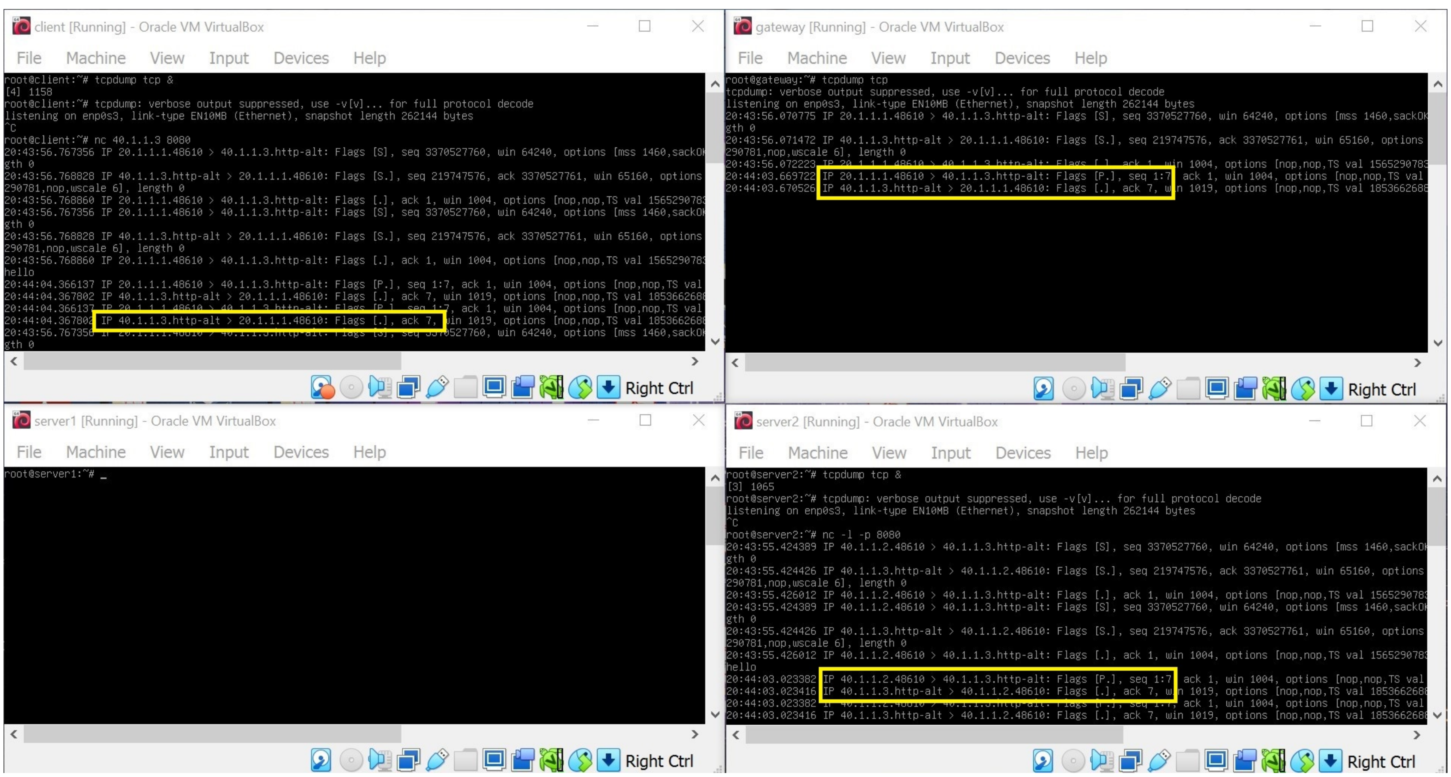
Observing the results of tcpdump [fig. 10], it shows :-

- On Server, the received packets appeared with source IP 40.1.1.2 instead of 20.1.1.1, confirming that SNAT was applied.
- On Client, the source IP of the acknowledgement is 40.1.1.1 even though the ack (response) was sent from server to gateway (40.1.1.2) as observed on the server tcpdump, but client received it from source as server, confirming the destination IP was reverted back to 20.1.1.1.

This validates that the NAT rules were correctly applied and the communication was valid.



(a) tcpdump result of client-server1



(b) tcpdump result of client-server2

Figure 10: Validating NAT configuration

5 Question 5

The existing NAT rules were flushed using `iptables -t nat -F` to clear the configurations of question 4, ensuring no previous rules interfere with the new load balancing configuration of this question.

5.1 a)

To implement load balancing between Server1 (40.1.1.1/24) and Server2 (40.1.1.3/24), the iptables statistic module can be used to distribute traffic with different probabilities (0.8 and 0.2). As observed in question 3, the average RTT of server2 (1.447 ms) was slightly lower than that of server1 (1.455 ms), hence we assign 80% of traffic to it, and the remaining 20% goes to Server1.

To route 20% of traffic to Server1 (40.1.1.1) & remaining 80% of traffic to Server2 (40.1.1.3), I used these commands on the `gateway` [fig. 11 (a)]:

```
/sbin/iptables -t nat -A PREROUTING -d 20.1.1.2 -m statistic --mode random
--probability 0.2 -j DNAT --to-destination 40.1.1.1
/sbin/iptables -t nat -A PREROUTING -d 20.1.1.2 -j DNAT --to-destination 40.1.1.3
/sbin/iptables -t nat -A POSTROUTING -d 40.1.1.1 -j SNAT --to-source 40.1.1.2
/sbin/iptables -t nat -A POSTROUTING -d 40.1.1.3 -j SNAT --to-source 40.1.1.2
```

Here, `iptables` is the utility used for managing Linux kernel packet filtering rules, `-t nat` specifies that the command applies to the NAT table, `-A PREROUTING` appends a rule to the PREROUTING chain, which modifies packets before routing decisions are made. The option `-s 20.1.1.1` specifies this rule applies to packets originating from the client subnet, `-m statistic` tells to use the statistic module to make routing decisions based on probability, `--mode random` is used to apply random selection for load balancing, `--probability 0.2` is used to route 20% of the packets to specified destination IP, `-j DNAT` indicates that Destination NAT (DNAT) is being used to modify the source IP address of the packet and `--to-destination 40.1.1.1` changes the destination IP to 40.1.1.1 (Server1).

The POSTROUTING commands ensure that the outgoing packets from gateway to servers have their source IP address changed to 40.1.1.2 (the gateway's IP). This is required so that the response from server is correctly routed back to the gateway, which can then forward it to the client.

5.2 b)

To validate the load balancing configuration, I ran tcpdump on both servers to monitor ICMP packets:

```
tcpdump icmp
```

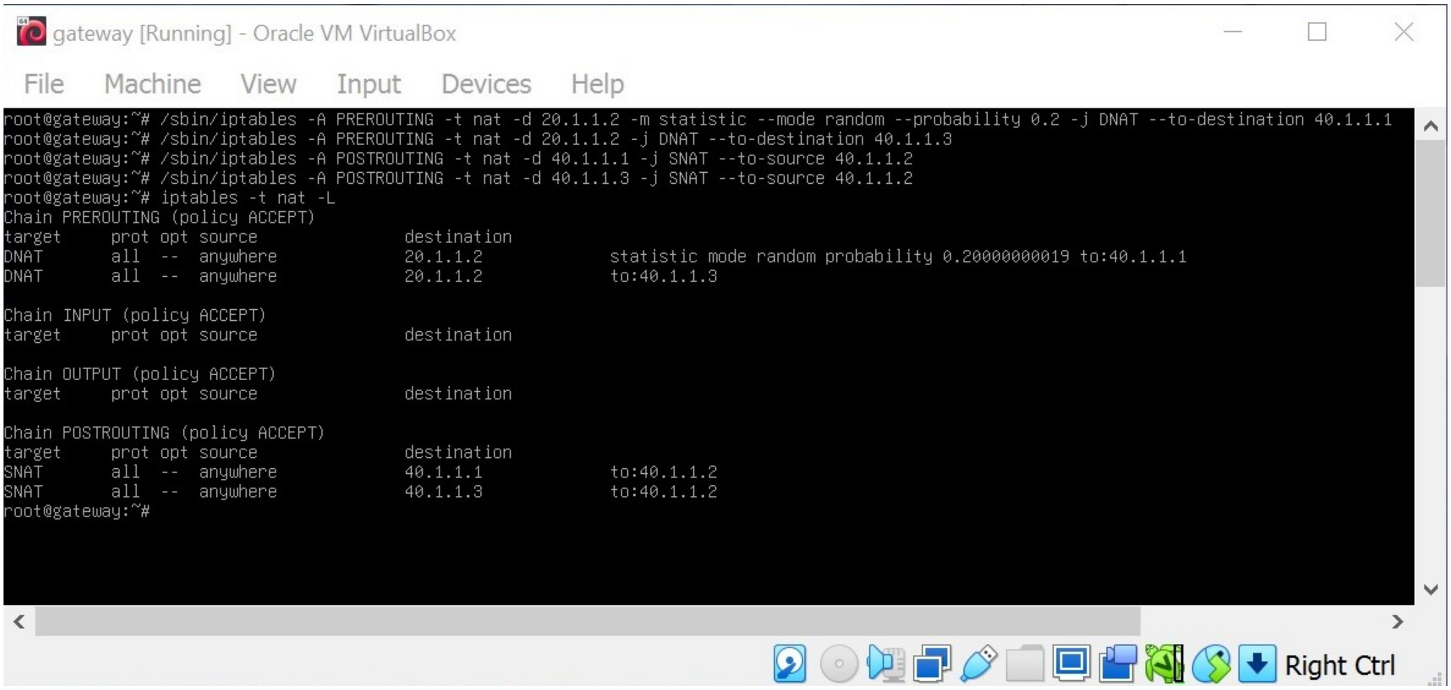
and then issued 10 separate ping requests one at a time (to maintain the state as NEW for each request) from client:

```
ping -c 1 20.1.1.2
```

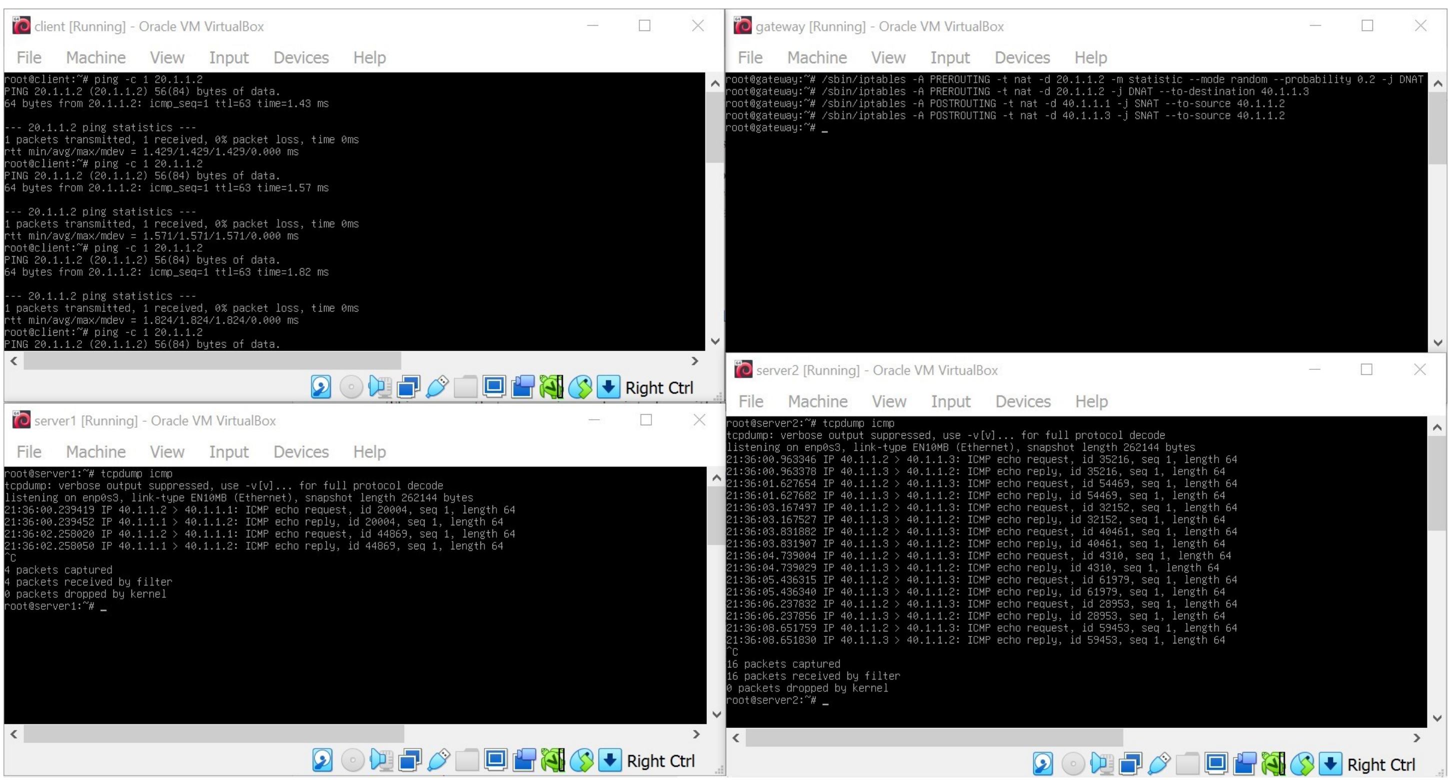
After sending 10 individual pings from the client, the following packet distribution was observed [fig. 11 (b)] from the tcpdump outputs on the servers:

- 2 ICMP packets were routed to Server1 (40.1.1.1).
- 8 ICMP packets were routed to Server2 (40.1.1.3).

This outcome aligns with the load balancing probabilities configured.



(a) configuring load balancing



(b) tcpdump output demonstrating load balancing

Figure 11: Load Balancing at gateway