Computer Networks Assignment 3

Souparno Ghose (2022506)

October 26, 2024

Question 1

Set up four VMs as shown in the figure. Use the same setup for the entire assignment

(a) Configure the IP addresses and routes for all VMs, as shown in the figure

(b) Configure VM2 as the gateway such that it can forward the incoming traffic to one of the servers – add forwarding functionality

Solution

(a) Four virtual machines (VMs) were created in VirtualBox, each configured with 1 CPU core, 512 MB of RAM, a 20 GB hard disk, and Debian (64-bit) as the operating system. The VMs were named *client*, *gateway*, *server1*, and *server2*.

I set up two host-only networks within VirtualBox. The first network was assigned the IP range `20.1.1.0/24`, and the second network was assigned `40.1.1.0/24`, both with DHCP disabled. These networks were labeled *VirtualBox Host-Only Ethernet Adapter* and *VirtualBox Host-Only Ethernet Adapter #2*, respectively.

To configure the network connections for the VMs using Netplan, I first downloaded the necessary packages for Netplan. All VMs were then set to use host-only adapters. Using the command `ip a`, I identified the available network interfaces on each VM.

I created and edited the configuration file for Netplan by running:

```
sudo nano /etc/netplan/00-installer-config.yaml
```

In the YAML file, I specified the network configurations for each VM. For instance, the *client* VM was connected to the *VirtualBox Host-Only Ethernet Adapter*, while the *gateway* VM had two interfaces: one for each of the host-only networks. The *server1* and *server2* VMs were both connected to the second host-only network.

After setting up the YAML configuration, I started the systemd network service with the following command:

```
sudo systemctl start systemd-networkd
```

I then enabled the service to start on boot:

```
sudo systemctl enable systemd-networkd
```

To apply the network configuration, I executed:

```
sudo netplan apply
```

Finally, I restarted the networking service on all VMs to ensure the changes took effect:

```
sudo systemctl restart networking.service
```

By disabling DHCP on both host-only networks and manually configuring the IP addresses, I ensured that the IP addresses remained static and consistent across reboots. This setup is crucial for networking experiments and custom routing configurations where predictable IP addresses are essential. The use of Netplan allowed for a streamlined and modern approach to managing network configurations in the Debian environment.

The network interface names in this setup, such as `enp0s3` and `enp0s8`, adhere to the predictable network interface naming convention in Linux, ensuring consistency across reboots and hardware configurations. In this naming convention:

- `en` denotes an Ethernet interface.
- `p0` indicates the physical slot number on the system's motherboard where the network device is connected.
- `s3` or `s8` represents the specific number or location of the interface on the device.

This naming scheme replaces the older `eth0`, `eth1`, etc., to enhance predictability, especially when multiple network interfaces are involved.

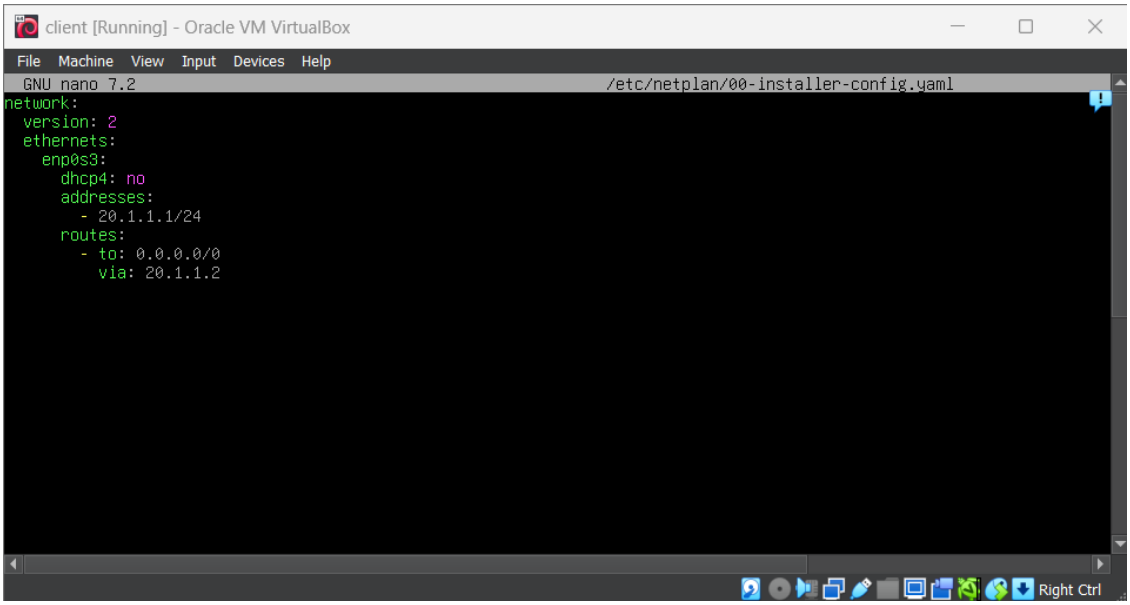

Figure 1: Client Configuration

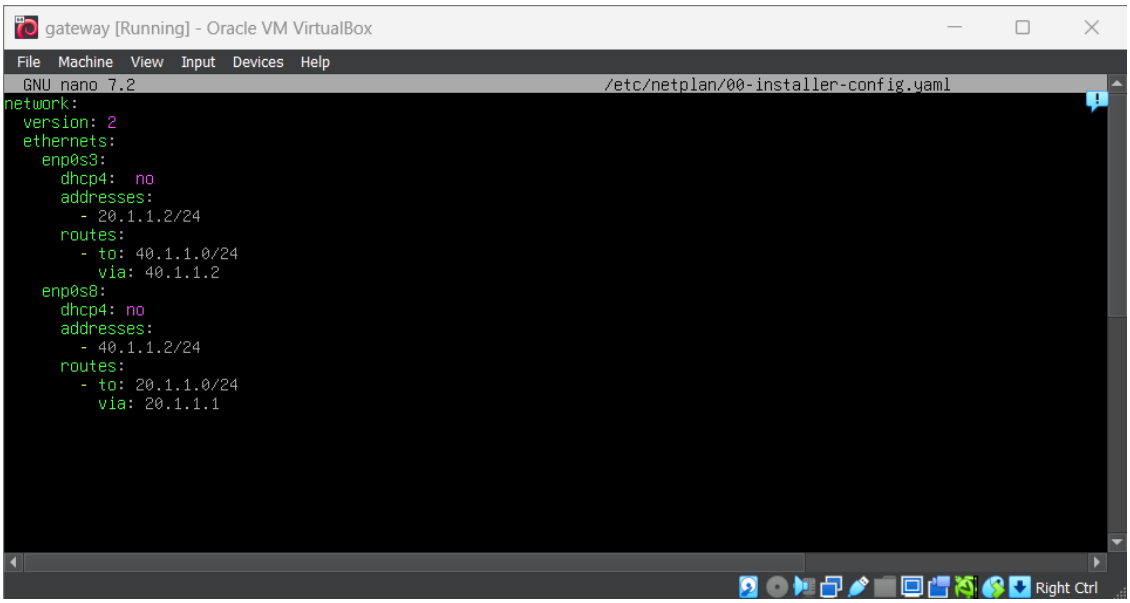

Figure 2: Gateway Configuration

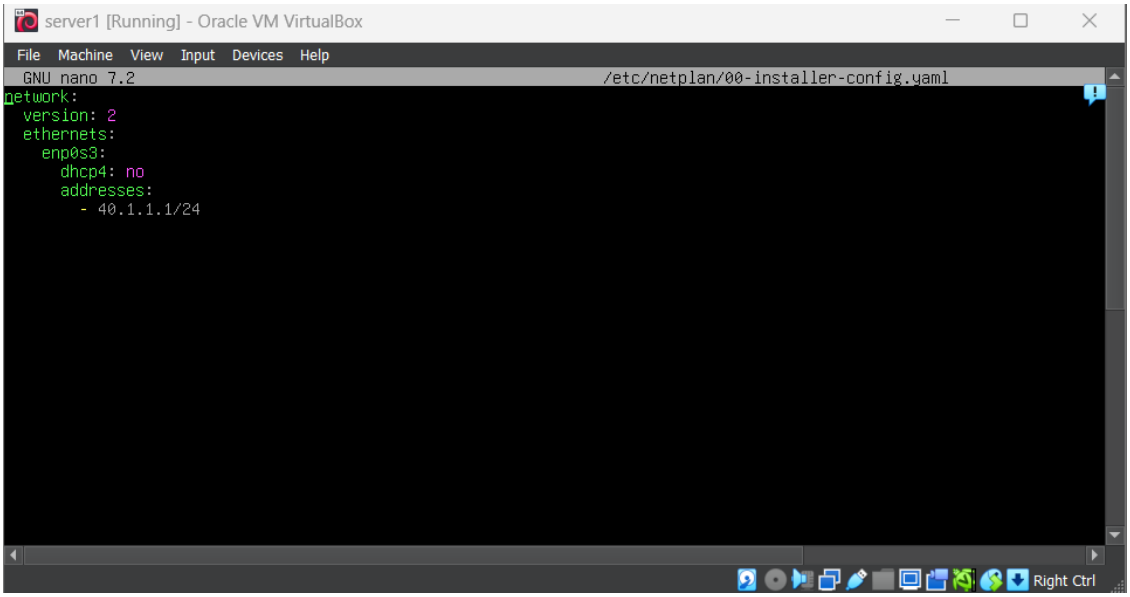

Figure 3: Server 1 Configuration

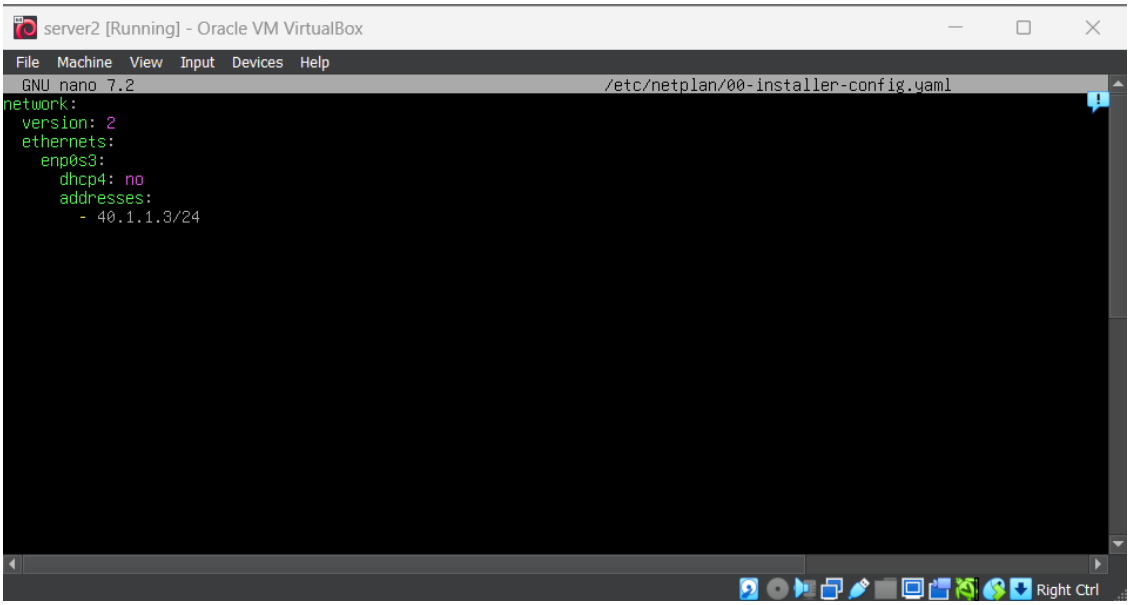

Figure 4: Server 2 Configuration

(b) Initially, when attempting to ping the IP addresses `40.1.1.1` (server1) and `40.1.1.3` (server2) from the *client* VM, the pings failed. This was because the *gateway* VM (VM2) had not been configured to forward traffic between the two networks it is connected to.

To enable packet forwarding on the *gateway* VM, the following command was executed:

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

This command temporarily enables IPv4 forwarding on the system, allowing the *gateway* to forward incoming traffic from the `20.1.1.0/24` network to the `40.1.1.0/24` network, and vice versa. After executing this command, both `40.1.1.1` and `40.1.1.3` became reachable from the *client* VM, as the *gateway* was now able to route the packets properly between the two networks.

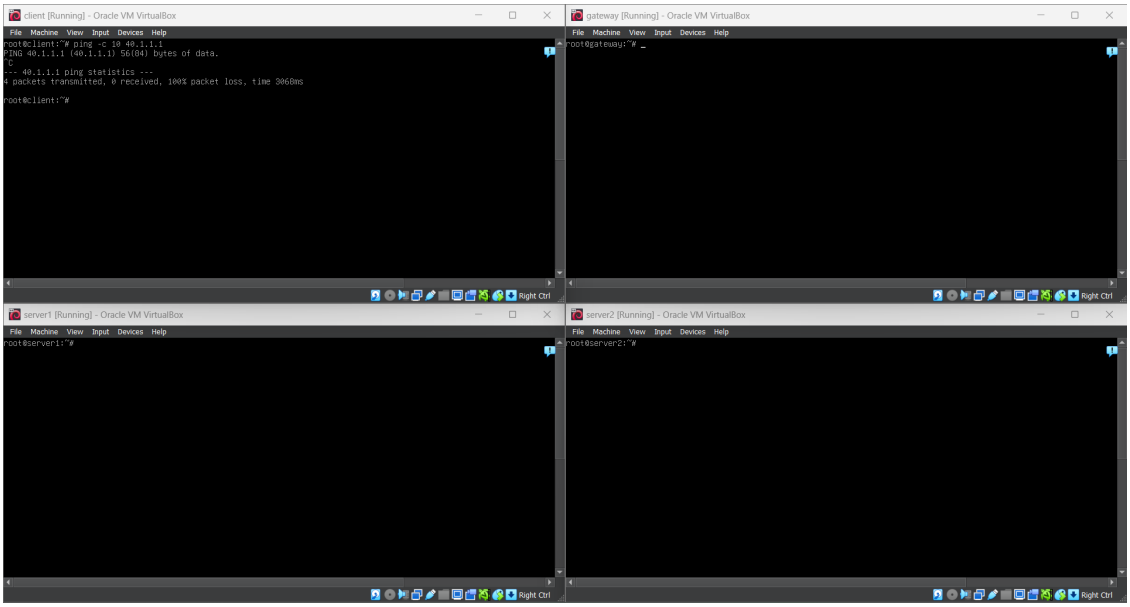

Figure 5: Showing how ping in not working before forwarding is set

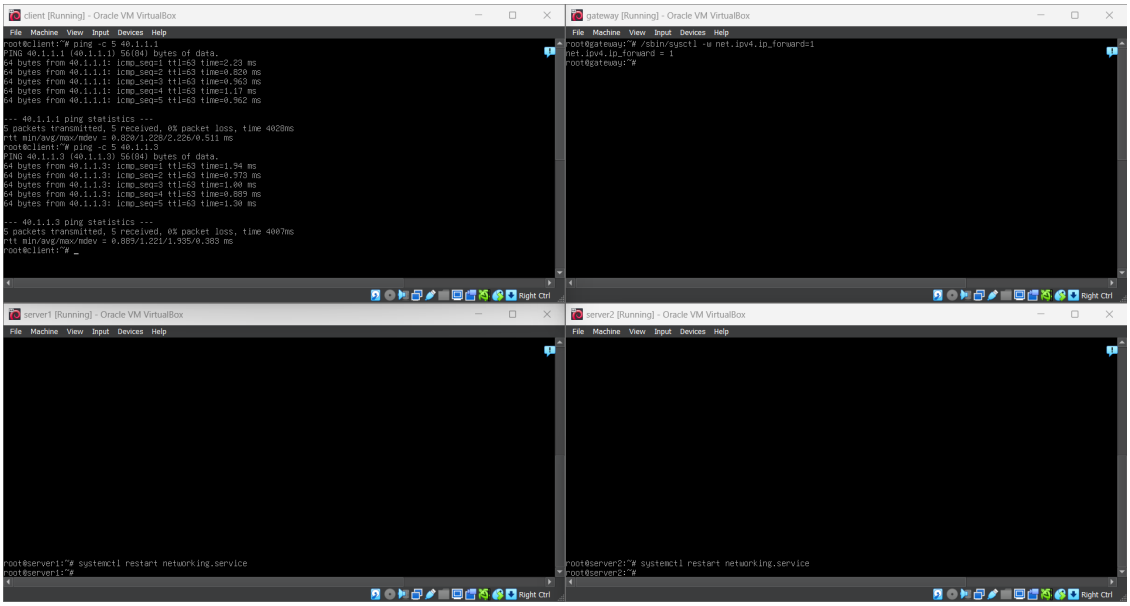

Figure 6: After forwarding is set

Question 2

Traffic filtering at the gateway VM

(a) The gateway must block all traffic (except for ping) destined to the server 40.1.1.1/24.Show that this works; attach the screenshot.

(b) The gateway must block only TCP traffic initiated by 20.1.1.1/24. Show that this works, attach the screenshot.

Solution

(a) Blocking all traffic except for ping to the server 40.1.1.1/24

To accomplish the task of allowing only `ping` (ICMP) traffic to server `40.1.1.1` while blocking all other types of traffic, the following `iptables` rules were applied on the *gateway* VM:

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

The first rule explicitly allows ICMP (ping) traffic destined for `40.1.1.1`. The second rule blocks all other traffic to `40.1.1.1`, regardless of protocol, by dropping the packets. This ensures that tools like `curl`, `wget`, or any non-ICMP traffic (such as TCP or UDP) will be blocked, while pings (ICMP echo requests) will still succeed.
After applying these rules, I was able to successfully `ping 40.1.1.1` from the *client* VM, but any attempt to access the server using other protocols (like HTTP or SSH) was blocked, confirming the functionality of the rule.

(b) Blocking only TCP traffic initiated by 20.1.1.1/24

To block only TCP traffic originating from the `20.1.1.1/24` network (the *client* VM's subnet), while allowing other types of traffic like UDP, the following `iptables` rule was applied on the *gateway* VM:

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

This rule drops all TCP traffic originating from the *client* VM with source IP `20.1.1.1`, preventing the *client* from establishing any TCP connections, such as HTTP or SSH, to any other machine. However, non-TCP traffic, such as UDP, remains unaffected and can still be forwarded.
After applying this rule, any TCP-based connections (e.g., trying to SSH or access a web server) failed, but UDP traffic (such as `traceroute` or other UDP-based protocols) still worked, demonstrating the effectiveness of the filtering rule.

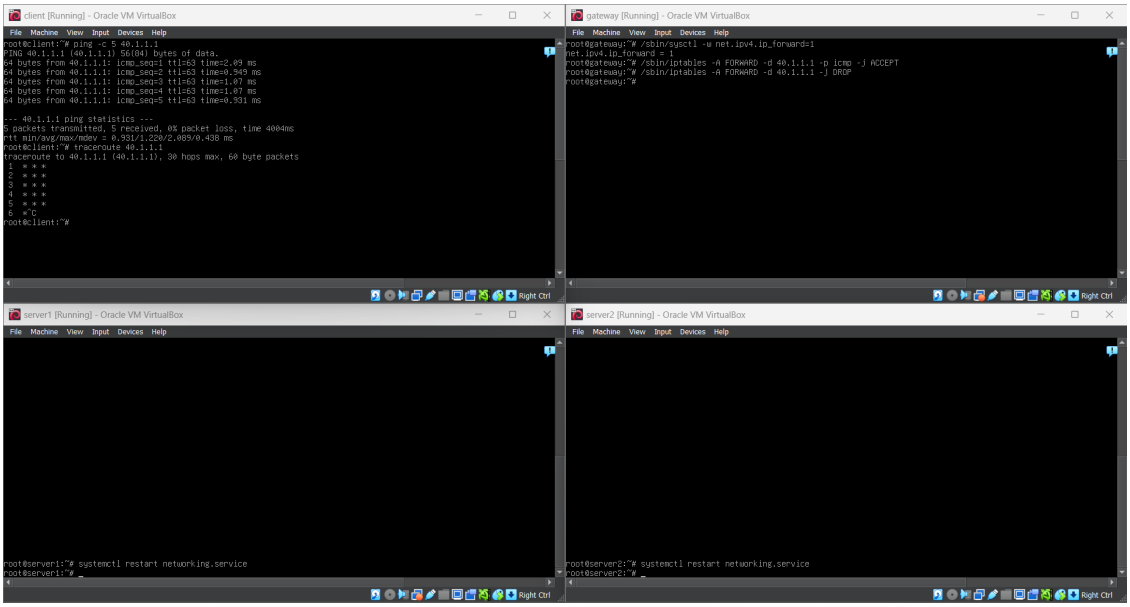

Figure 7: Q2 Part A Ping is working but traceroute is not

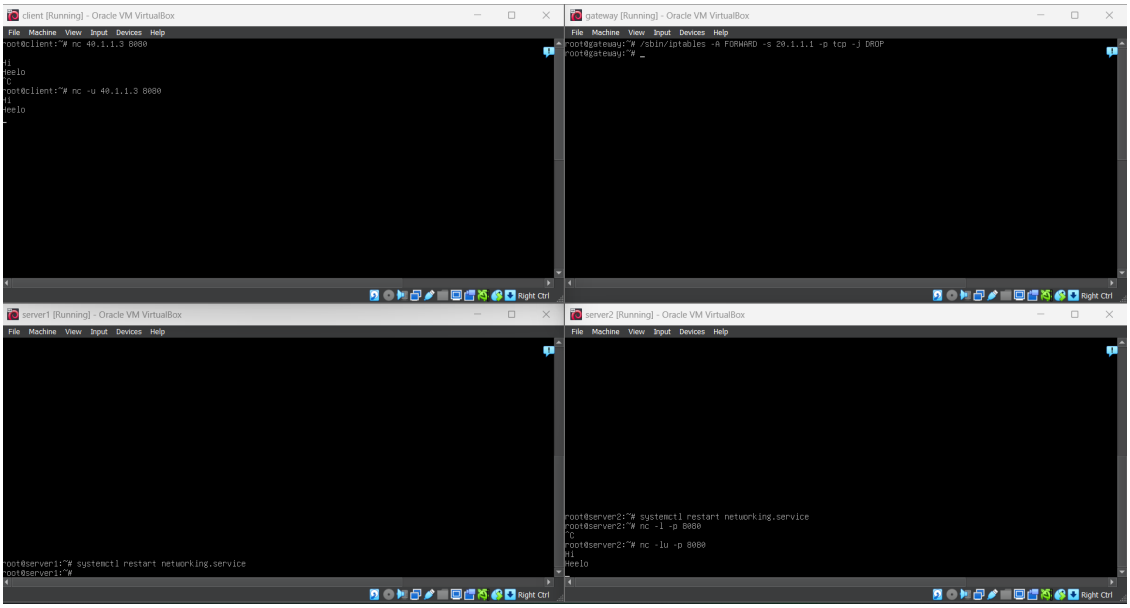

Figure 8: Q2 Part B TCP Packets are not sent while UDP Packets are being sent

Question 3

Use the configuration obtained in Q.2. to solve this question

(a) Use "iperf2" tool to test the TCP and UDP bandwidth between 20.1.1.1/24 and 40.1.1.3/24. Attach the screenshot.

(b) What is the minimum, average, and maximum RTT (Attach the screenshot) (i) from 20.1.1.1/24 to 40.1.1.1/24 (ii) from 20.1.1.1/24 to 40.1.1.3/24 (iii) Did you find a significant difference between (i) and (ii)? If so, why?

Solution

(a) Testing TCP and UDP Bandwidth between 20.1.1.1/24 and 40.1.1.3/24

To test the TCP and UDP bandwidth between the *client* (`20.1.1.1`) and *server2* (`40.1.1.3`), the `iperf2` tool was used.

TCP Bandwidth Test: For the TCP bandwidth test, the following commands were executed:

- On `server2` (`40.1.1.3`), the server mode of `iperf2` was initiated:

```
iperf -s
```

- On the *client* (`20.1.1.1`), the following command was used to run the bandwidth test:

```
iperf -c 40.1.1.3
```

However, no results were obtained from this test because, as per the configuration applied in Q.2(b), all TCP packets originating from `20.1.1.1` are blocked by the following `iptables` rule:

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

This rule drops all TCP traffic from the `client`, preventing any TCP connection from being established for the bandwidth test.

UDP Bandwidth Test: For the UDP bandwidth test, the following commands were used:

- On `server2` (`40.1.1.3`), the `iperf2` server was initiated in UDP mode:

  ```
  iperf -u -s
  ```

- On the *client* (`20.1.1.1`), the UDP bandwidth test was initiated with the following command:

  ```
  iperf -u -c 40.1.1.3
  ```

The UDP test showed a successful data transfer with a measured bandwidth of approximately **1.05 Mbits/sec**. Since UDP traffic is not affected by the TCP blocking rule set in Q.2(b), the packets were able to flow between the client and server without restriction.

(b) Measuring Round Trip Time (RTT)

To measure the minimum, average, and maximum Round Trip Time (RTT), `ping` was used from the *client* (`20.1.1.1`) to both *server1* (`40.1.1.1`) and *server2* (`40.1.1.3`).

- **(i)** From `20.1.1.1` to `40.1.1.1`, the minimum, average, and maximum RTT were measured with `ping`.
- **(ii)** From `20.1.1.1` to `40.1.1.3`, the minimum, average, and maximum RTT were also measured using `ping`.

(iii) RTT Comparison: There was no significant difference in the minimum, average, or maximum RTT between the pings sent to `40.1.1.1` and those sent to `40.1.1.3`. The similarity in RTTs can be attributed to the fact that both servers are connected to the same network segment `40.1.1.0/24`, resulting in similar physical distances and network latencies. The gateway handles traffic between the `20.1.1.0/24` and `40.1.1.0/24` networks equally for both servers, hence there is no significant difference in the RTT measurements.

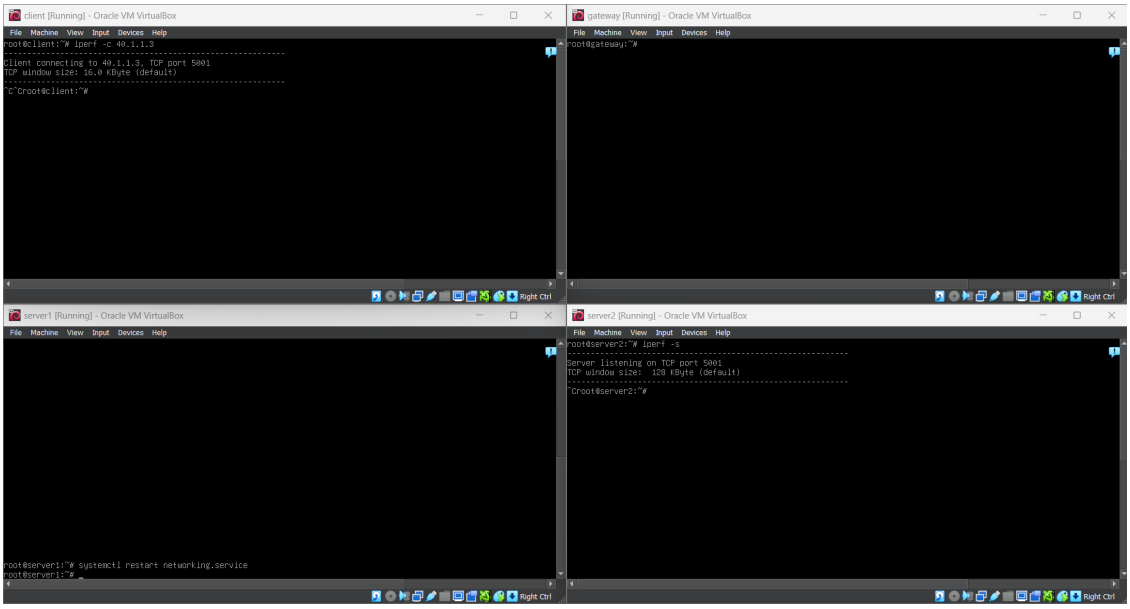

Figure 9: Q3 Part A (TCP Bandwidth)

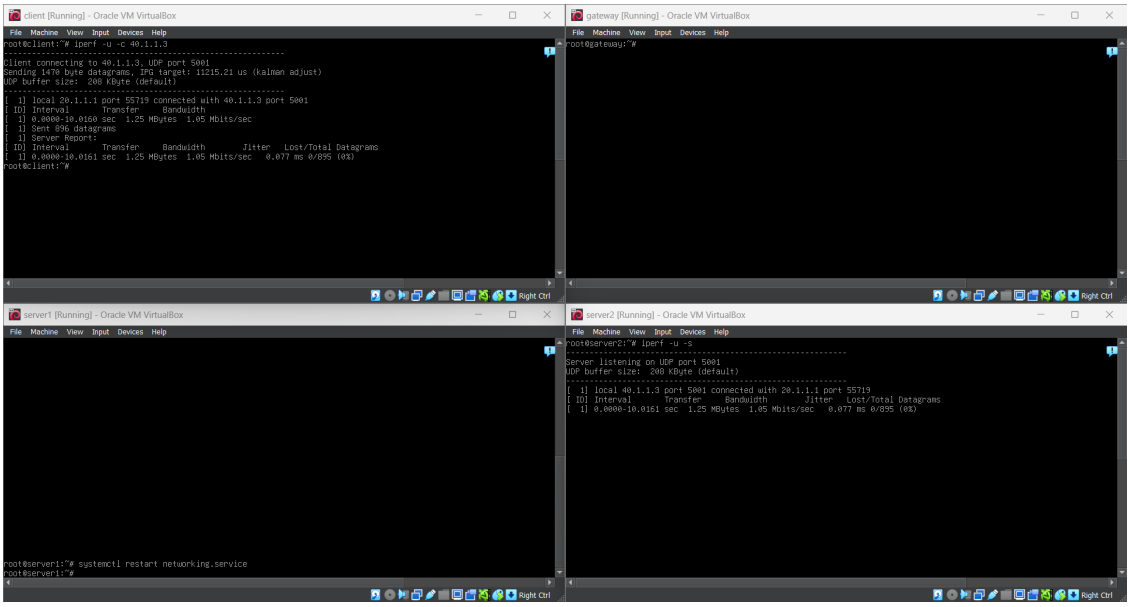

Figure 10: Q3 Part A (UDP Bandwidth)

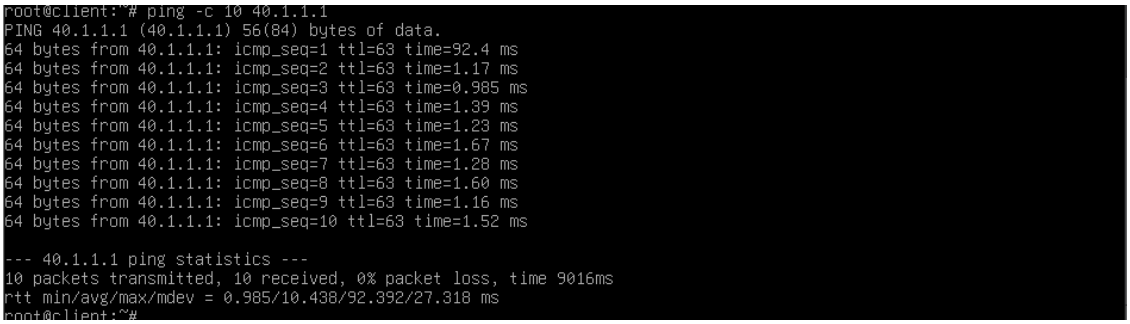

Figure 11: Q3 Part B Pinging 40.1.1.1

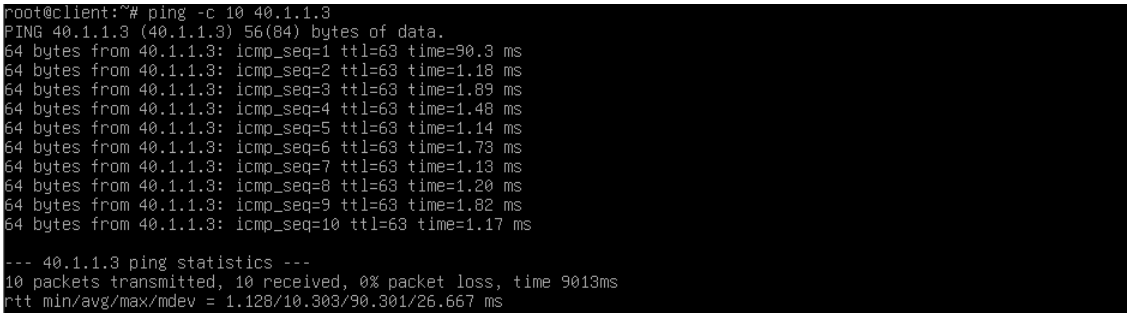

Figure 12: Q3 Part B Pinging 40.1.1.3

Question 4

Network address translation at the gateway VM

(a) Change the source IP address of every packet from 20.1.1.1/24 to 40.1.1.2/24

(b) When the packet response for the packet from step "a" arrives at the gateway, revert the destination IP address to the original.

(c) Validate the above by sending traffic and observing the packets at each VM using Wireshark/tcpdump. Attach the screenshot.

Solution

Flushing Existing iptables Rules

Before configuring the Network Address Translation (NAT) rules, the existing `iptables` rules were flushed using the following command:

```
/sbin/iptables -F
```

This ensures that the system starts with a clean state, free of any previously set firewall or NAT rules.

(a) Changing the Source IP Address

To change the source IP address of every outgoing packet from the `20.1.1.1/24` network to `40.1.1.2/24` (the *gateway* VM's IP), the following `iptables` rule was applied:

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

This rule is part of the NAT table and applies source network address translation (SNAT) to packets originating from `20.1.1.1`. It changes the source IP address to `40.1.1.2` when the packet is routed out of the gateway. This allows the server receiving the packet to believe it came from `40.1.1.2` instead of the `client` VM's real IP.

(b) Reverting the Destination IP Address on Response

When a response packet destined for `40.1.1.2` (modified in step (a)) returns to the gateway, we need to revert the destination IP address to the original source, `20.1.1.1`. This is accomplished using the following `iptables` rule:

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

This rule performs destination network address translation (DNAT) in the `PREROUTING` chain. It changes the destination IP address from `40.1.1.2` back to `20.1.1.1` so that the response reaches the original client VM.

(c) Validating the NAT Configuration with tcpdump and netcat

To validate the above NAT configuration, traffic was sent between the `client` (`20.1.1.1`) and *server1* (`40.1.1.1`) while observing the packets on both systems using `tcpdump`. The following steps were taken:

- On `40.1.1.1`, I started capturing the TCP packets using:

  ```
  tcpdump tcp &
  ```

- On `20.1.1.1`, I also initiated a packet capture:

  ```
  tcpdump tcp &
  ```

- Next, I used `netcat` (`nc`) to create a simple TCP connection. On `40.1.1.1`, I started a listener on port 8080:

  ```
  nc -l -p 8080
  ```

- On `20.1.1.1`, I initiated a connection to `40.1.1.1` using `nc`:

  ```
  nc 40.1.1.1 8080
  ```

- Using `tcpdump`, I observed the routes taken by the packets. The source IP was initially modified from `20.1.1.1` to `40.1.1.2` when the packets reached the gateway. On their return, the destination IP was reverted back to `20.1.1.1`, confirming that the NAT rules were correctly applied and the communication was valid.

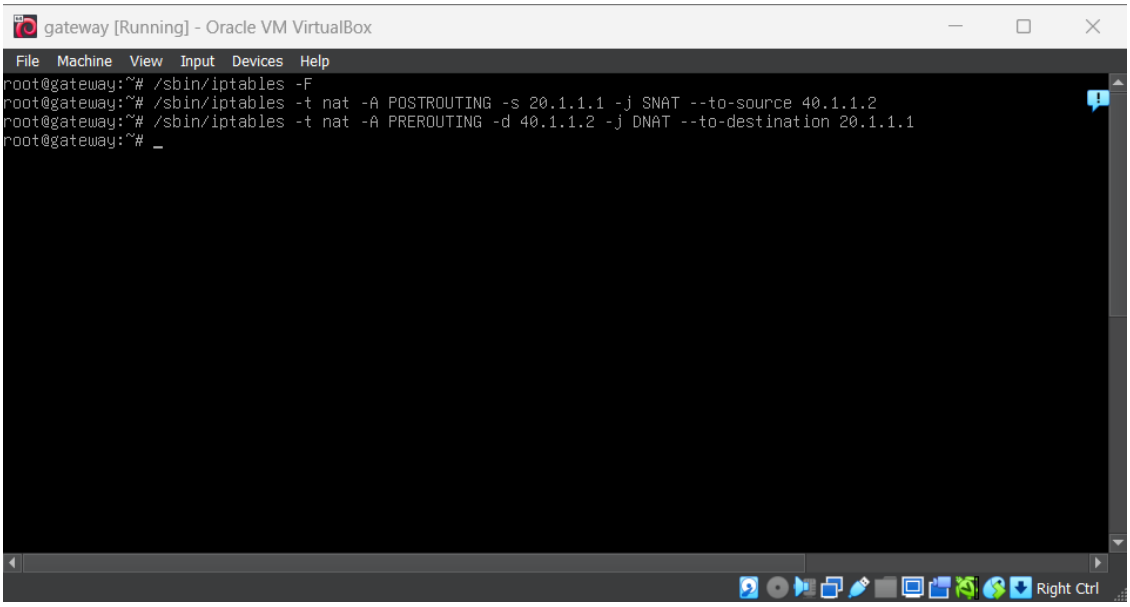

Figure 13: Q4 Part A and B

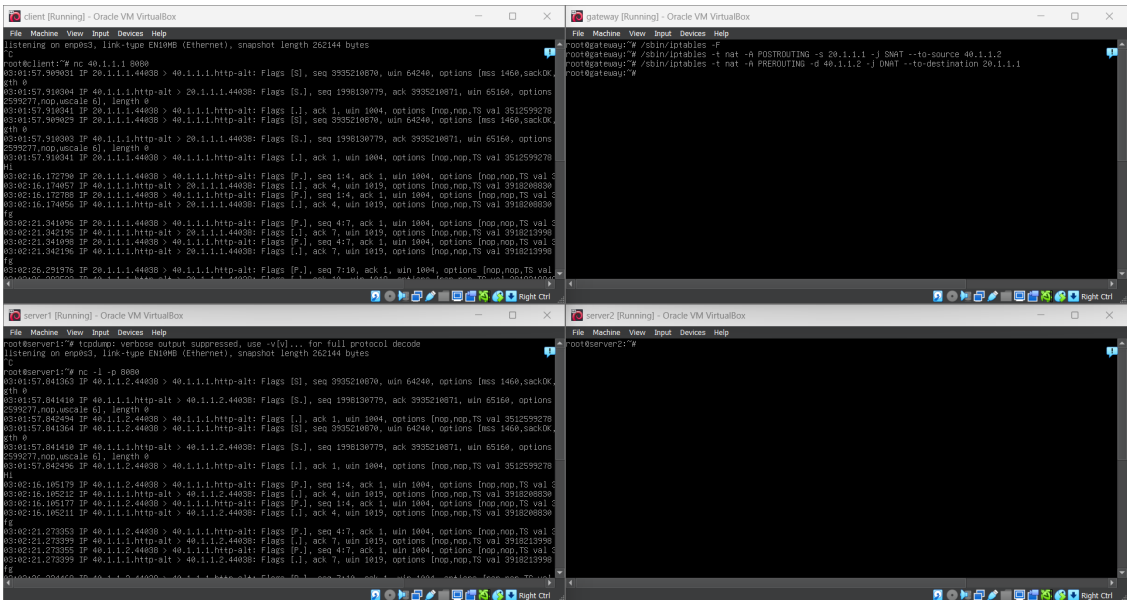

Figure 14: Q4 Part C

Question 5

Load balancing at the gateway VM. Attach screenshots

(a) Using the information obtained from Q.5.b., balance the traffic from 20.1.1.1/24 to the servers, 40.1.1.1/24 and 40.1.1.3/24. The probability of assigning the packet to the servers is 0.8 and 0.2, i.e., assign a high probability to the server with lower RTT.

(b) Test the above configuration using a series of "ping" packets.

Solution

Before setting up the load balancing rules, I flushed the existing NAT table from the configuration of Q.4 using the following command:

```
/sbin/iptables -t nat -F
```

This ensures that no previous rules interfere with the new load balancing configuration.

(a) Configuring Load Balancing with 80-20 Traffic Split

To balance traffic from the `client` (`20.1.1.1/24`) to `server1` (`40.1.1.1/24`) and `server2` (`40.1.1.3/24`) based on the RTT, I used the following `iptables` rules:

```
/sbin/iptables -A PREROUTING -t nat -d 20.1.1.2 -m statistic --mode random --probabili-
-ty 0.2 -j DNAT --to-destination 40.1.1.1
/sbin/iptables -A PREROUTING -t nat -d 20.1.1.2 -j DNAT --to-destination 40.1.1.3
```

These two rules perform Destination Network Address Translation (DNAT) in the `PREROUTING` chain. The first rule sends 20% of the traffic to `server1` (`40.1.1.1`) by using the `--probability 0.2` option, and the second rule sends the remaining 80% of the traffic to `server2` (`40.1.1.3`), which had a lower RTT based on prior measurements.
To ensure that the response packets are returned with the correct source IP address, the following `iptables` rules were applied to perform Source Network Address Translation (SNAT) in the `POSTROUTING` chain:

```
/sbin/iptables -A POSTROUTING -t nat -d 40.1.1.1 -j SNAT --to-source 40.1.1.2
/sbin/iptables -A POSTROUTING -t nat -d 40.1.1.3 -j SNAT --to-source 40.1.1.2
```

These rules modify the source IP address of the outgoing packets to `40.1.1.2` so that the responses are directed back to the gateway VM.
Through the use of `ping` testing, it was observed that `40.1.1.3` was less congested with lower RTT values compared to `40.1.1.1`, confirming that 80% of the traffic was directed to `server2` (`40.1.1.3`), as expected.

(b) Validating Load Balancing using ping and tcpdump

To validate the traffic distribution, I used the following commands to capture ICMP traffic on both servers:

- On `server1` (`40.1.1.1`), I ran:

```
tcpdump icmp &
```

- On `server2` (`40.1.1.3`), I ran:

```
tcpdump icmp &
```

On the *client* (`20.1.1.1`), I used the following command to send 10 ping requests to `20.1.1.2`:

```
ping -c 1 20.1.1.2
```

After executing the command 10 times, I observed that out of 10 ICMP echo requests, 8 packets were sent to `server2` (`40.1.1.3`) and 2 packets were sent to `server1` (`40.1.1.1`), confirming the expected 80-20 traffic split based on RTT.

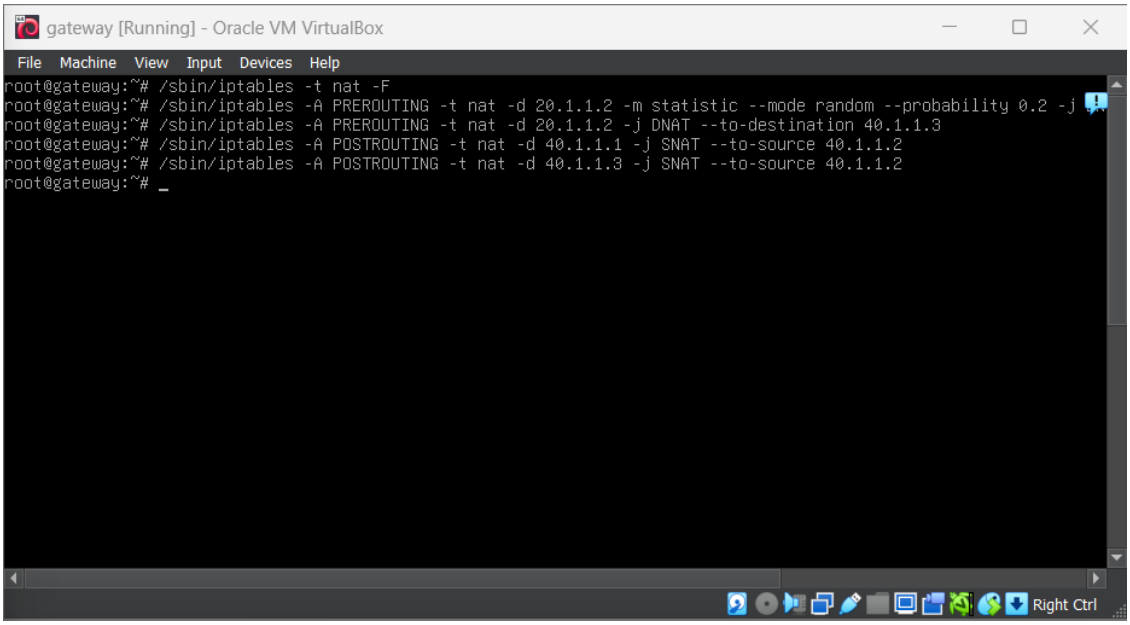

Figure 15: Q5 Part A

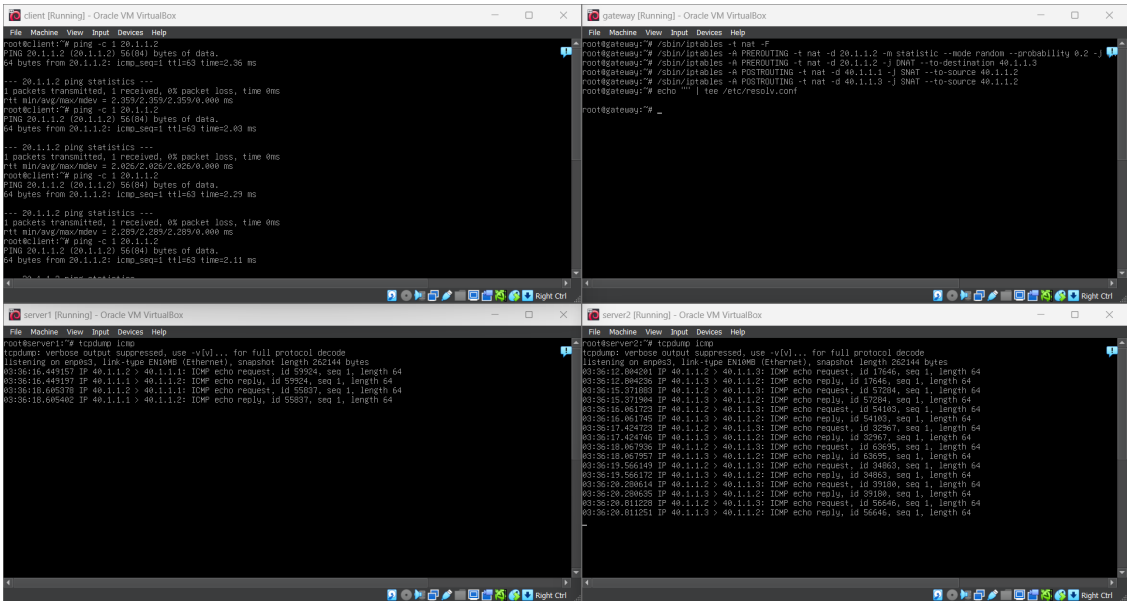

Figure 16: Q5 Part B