CSE 644 Internet Security Lab-5 (Vpn lab)

Sneden Rebello

Task 1

Please conduct the following testings to ensure that the lab environment is set up correctly:

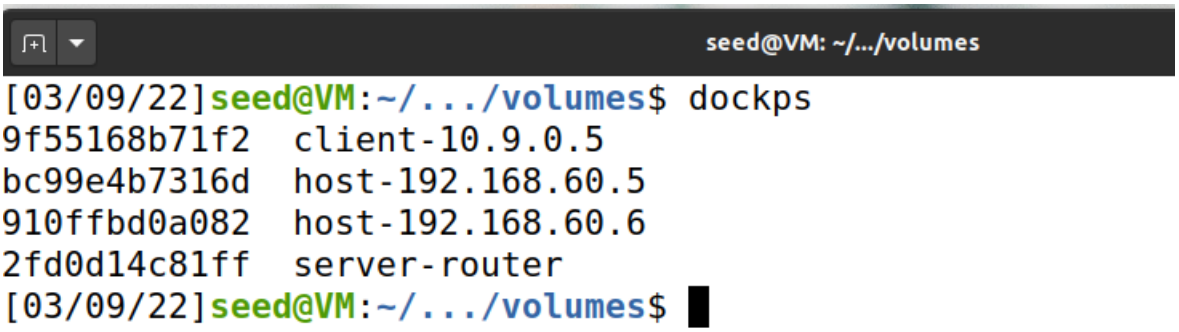
• Host U can communicate with VPN Server.

• VPN Server can communicate with Host V.

• Host U should not be able to communicate with Host V.

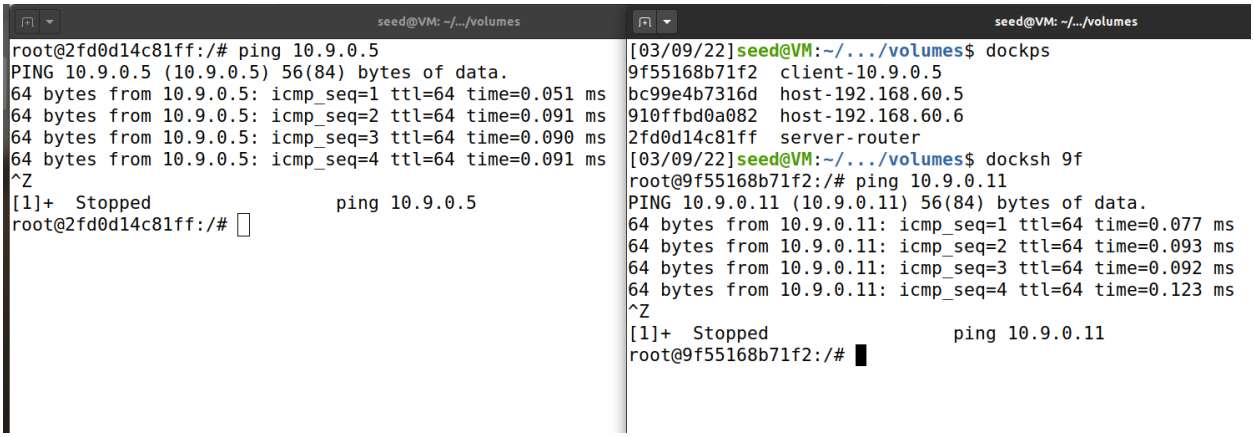
• Run tcpdump on the router, and sniff the traffic on each of the network. Show that you can capture packets.

Environment :

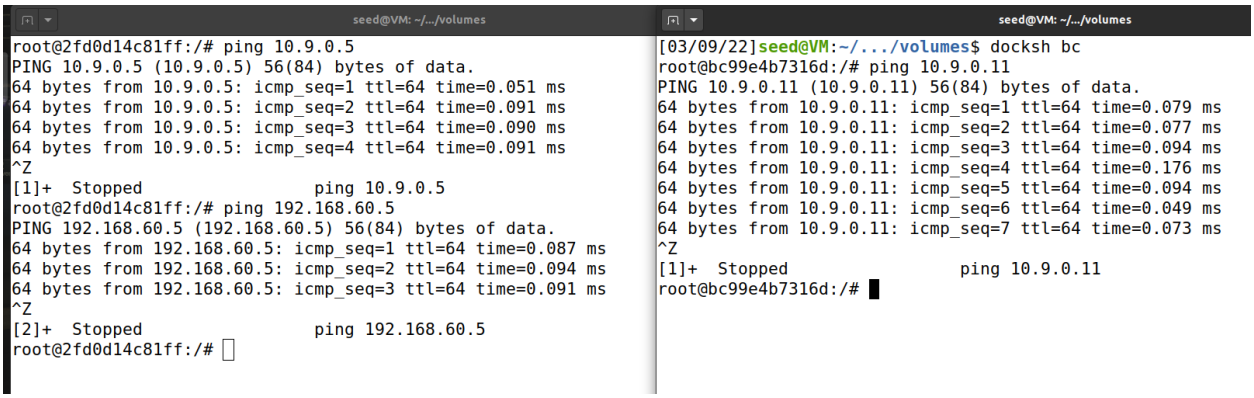


I build and run the docker file and run dockps to check all the containers. I make sure the lab is setup as expected.

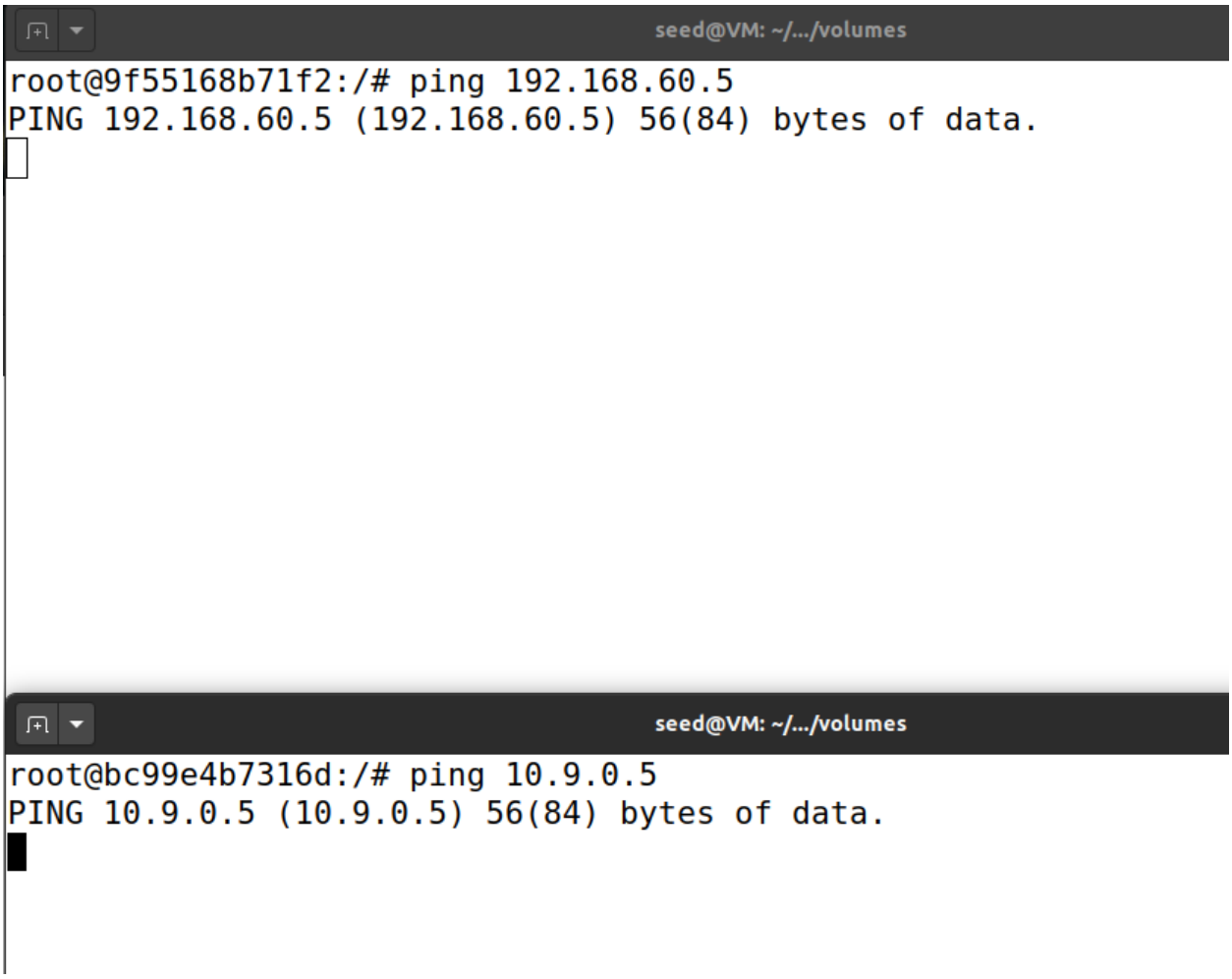
Host U can communicate with VPN Server.



VPN Server can communicate with Host V.



Host U should not be able to communicate with Host V.



I run tcpdump on the router, and sniff the traffic on each of the network. I can sniff the packets.

```
root@2fd0d14c81ff:/# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
17:38:43.836089 IP client-10.9.0.5.net-10.9.0.0 > 2fd0d14c81ff: ICMP echo reques
t, id 16, seq 1, length 64
17:38:43.836108 IP 2fd0d14c81ff > client-10.9.0.5.net-10.9.0.0: ICMP echo reply,
 id 16, seq 1, length 64
17:38:44.858646 IP client-10.9.0.5.net-10.9.0.0 > 2fd0d14c81ff: ICMP echo reques
t, id 16, seq 2, length 64
17:38:44.858680 IP 2fd0d14c81ff > client-10.9.0.5.net-10.9.0.0: ICMP echo reply,
 id 16, seq 2, length 64
17:38:45.882754 IP client-10.9.0.5.net-10.9.0.0 > 2fd0d14c81ff: ICMP echo reques
t, id 16, seq 3, length 64
17:38:45.882778 IP 2fd0d14c81ff > client-10.9.0.5.net-10.9.0.0: ICMP echo reply,
 id 16, seq 3, length 64
17:38:46.907101 IP client-10.9.0.5.net-10.9.0.0 > 2fd0d14c81ff: ICMP echo reques
t, id 16, seq 4, length 64
17:38:46.907137 IP 2fd0d14c81ff > client-10.9.0.5.net-10.9.0.0: ICMP echo reply,
```

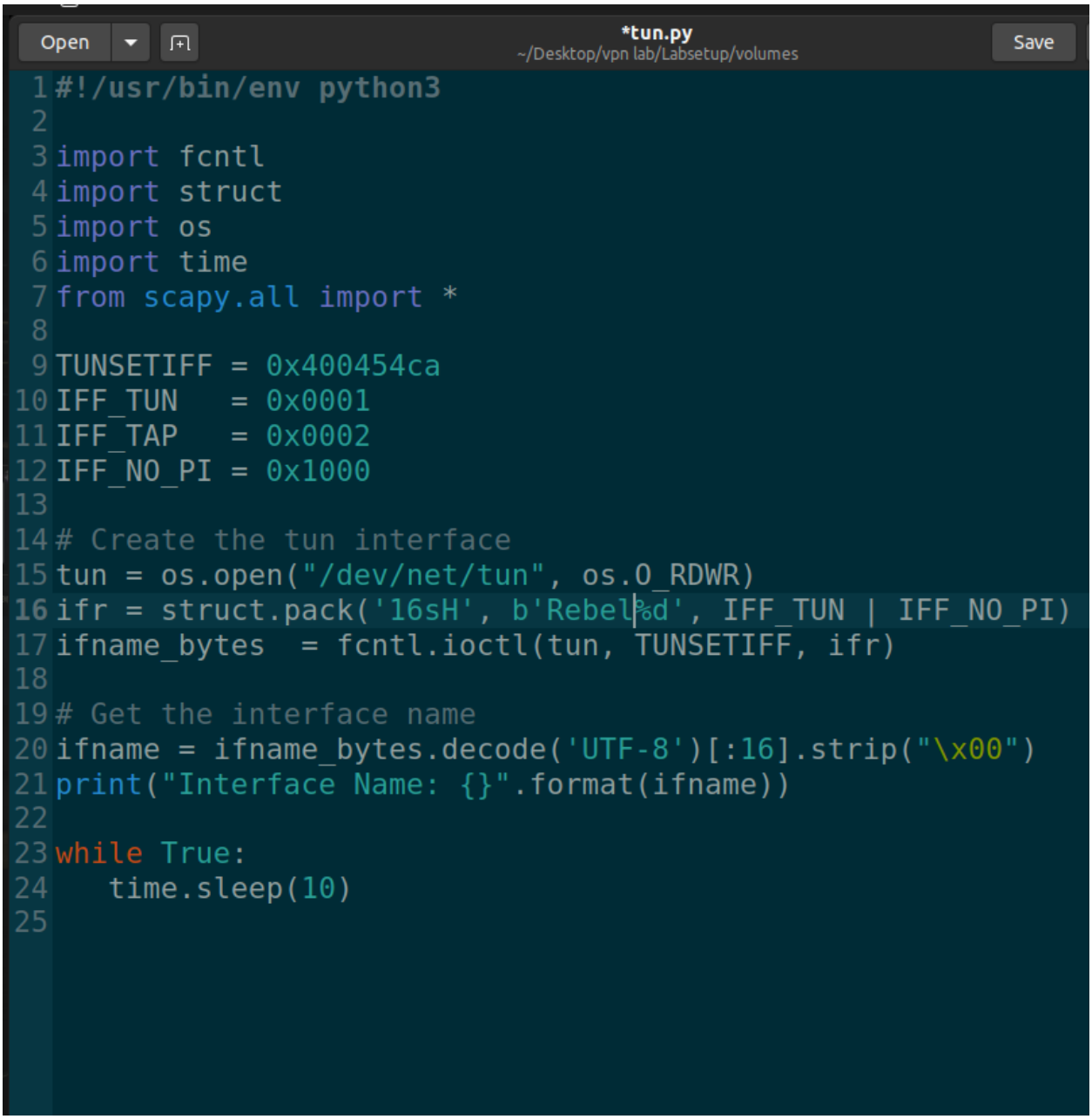
```
.0.0: ICMP echo request, id 36, seq 6, length 64
17:40:48.826529 ARP, Request who-has client-10.9.0.5.net-10.9.0.0 tell 2fd0d14c8
1ff, length 28
17:40:48.826692 ARP, Reply client-10.9.0.5.net-10.9.0.0 is-at 02:42:0a:09:00:05
(oui Unknown), length 28
17:40:48.858604 IP host-192.168.60.5.net-192.168.60.0 > client-10.9.0.5.net-10.9
.0.0: ICMP echo request, id 36, seq 7, length 64
17:40:49.882705 IP host-192.168.60.5.net-192.168.60.0 > client-10.9.0.5.net-10.9
.0.0: ICMP echo request, id 36, seq 8, length 64
17:40:50.907139 IP host-192.168.60.5.net-192.168.60.0 > client-10.9.0.5.net-10.9
.0.0: ICMP echo request, id 36, seq 9, length 64
17:41:01.371305 IP6 fe80::348d:30ff:fefd:c5be > ip6-allrouters: ICMP6, router so
licitation, length 16
```

Task 2

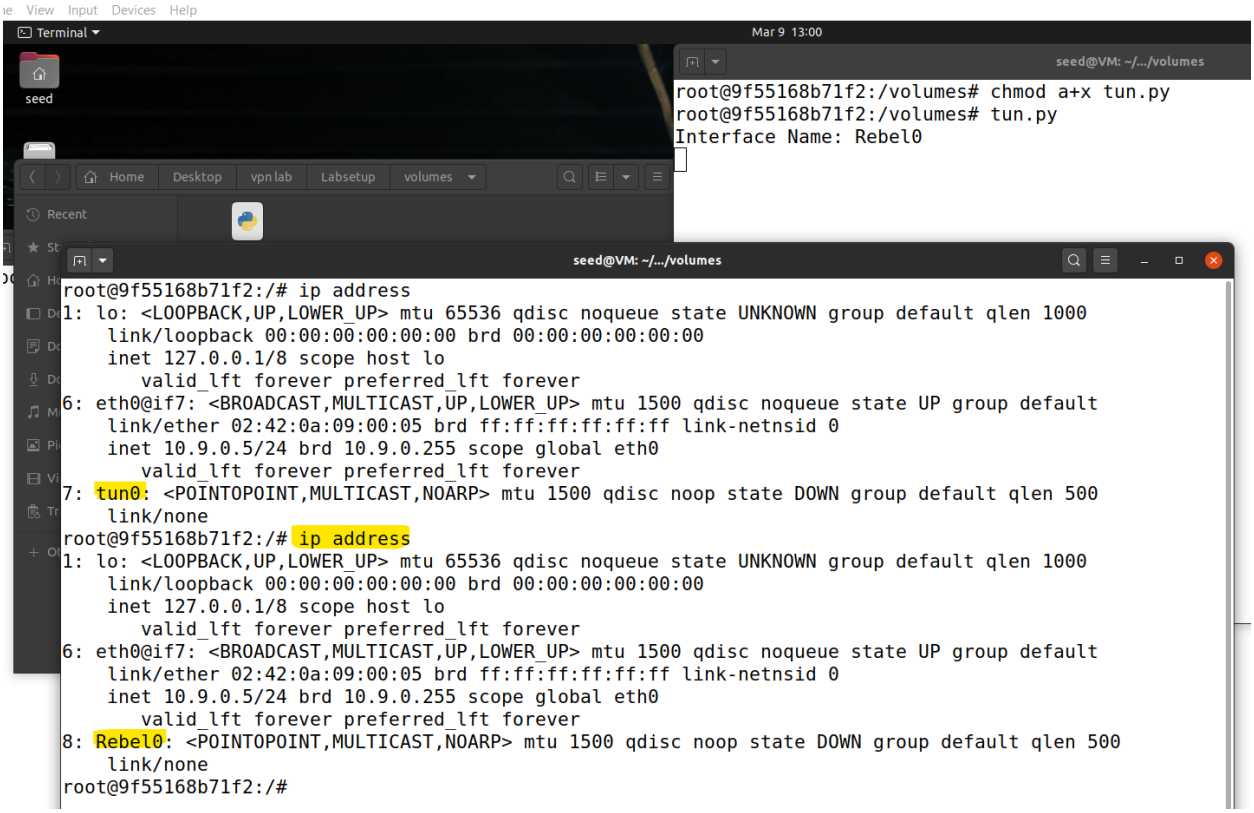
2a. Name of the Interface

I run the existing code and then run ip address to check for all interfaces.

The code is as shown below. I also change the interface name from Tun to Rebel (my last name).

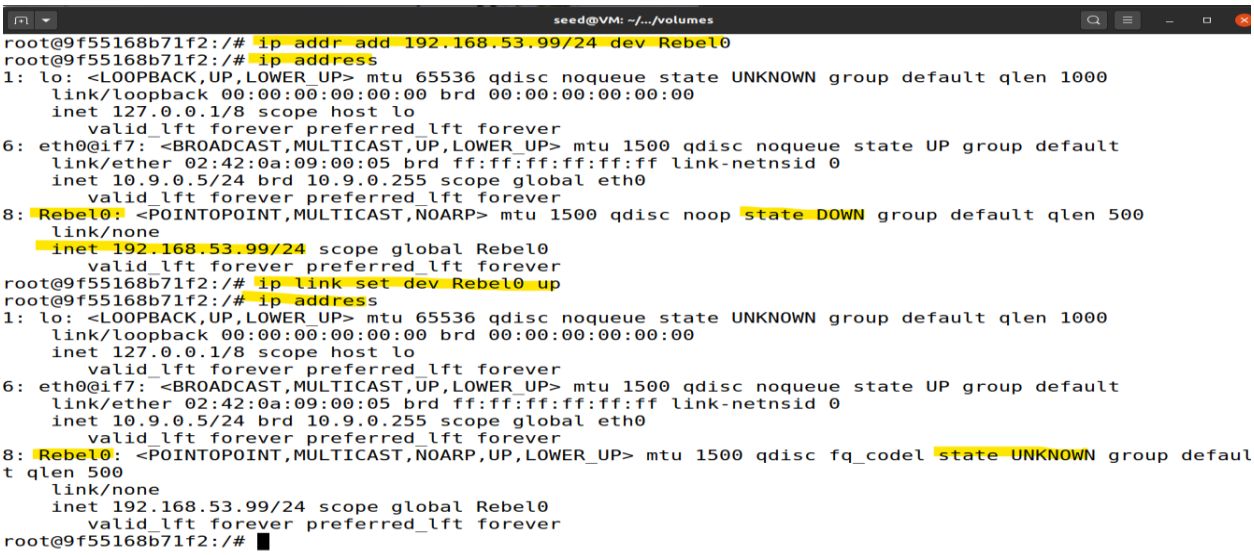


Below I run ip address command again after re running the code and I see the change in interface name.



Task 2.b: Set up the TUN Interface

I add an ip address via the command “ip addr add”. After this the ip address is assigned but the interface is still down. I run “ ip link set up” command to get the interface up and running.



Below, I add the lines of code and delete the existing interface I created via command “ip link delete Rebel0”. I then run the code and check if it foes all the above automatically.

Open ~/Desktop/vpn lab/Labsetup/volumes tun.py Save

```
#!/usr/bin/env python3

import fcntl
import struct
import os
import time
from scapy.all import *

TUNSETIFF = 0x400454ca
IFF_TUN   = 0x0001
IFF_TAP   = 0x0002
IFF_NO_PI = 0x1000

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'Rebel%d', IFF_TUN | IFF_NO_PI)
ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)

# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface Name: {}".format(ifname))

os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
```

Below shows that it worked and got the network up and running with just running the code.

seed@VM: ~/.../volumes

```
root@9f55168b71f2:/# ip address
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
6: eth0@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:0a:09:00:05 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.9.0.5/24 brd 10.9.0.255 scope global eth0
       valid_lft forever preferred_lft forever
10: Rebel0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group defau
lt qlen 500
    link/none
    inet 192.168.53.99/24 scope global Rebel0
       valid_lft forever preferred_lft forever
root@9f55168b71f2:/#
```

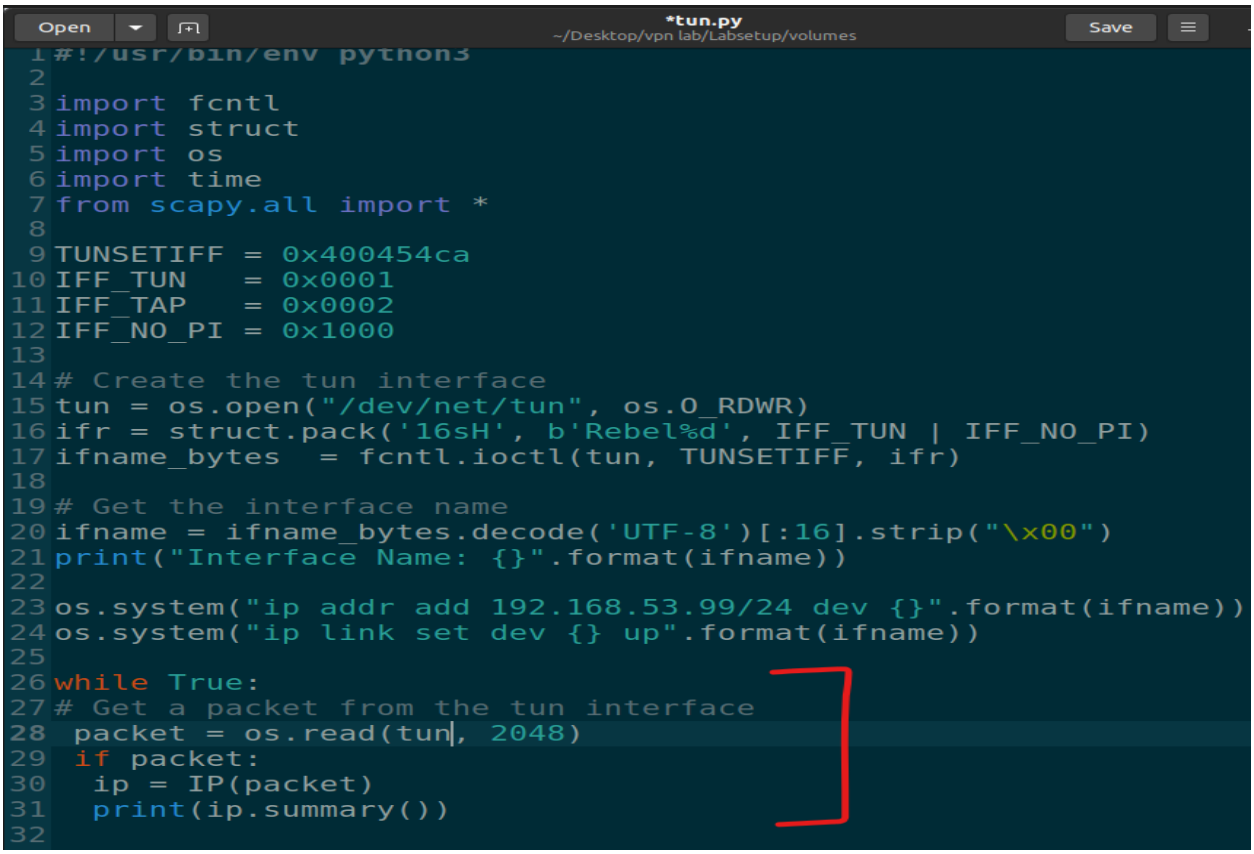
2c Read from the TUN Interface

configure the TUN interface accordingly, and then conduct the following experiments. Please describe your observations:

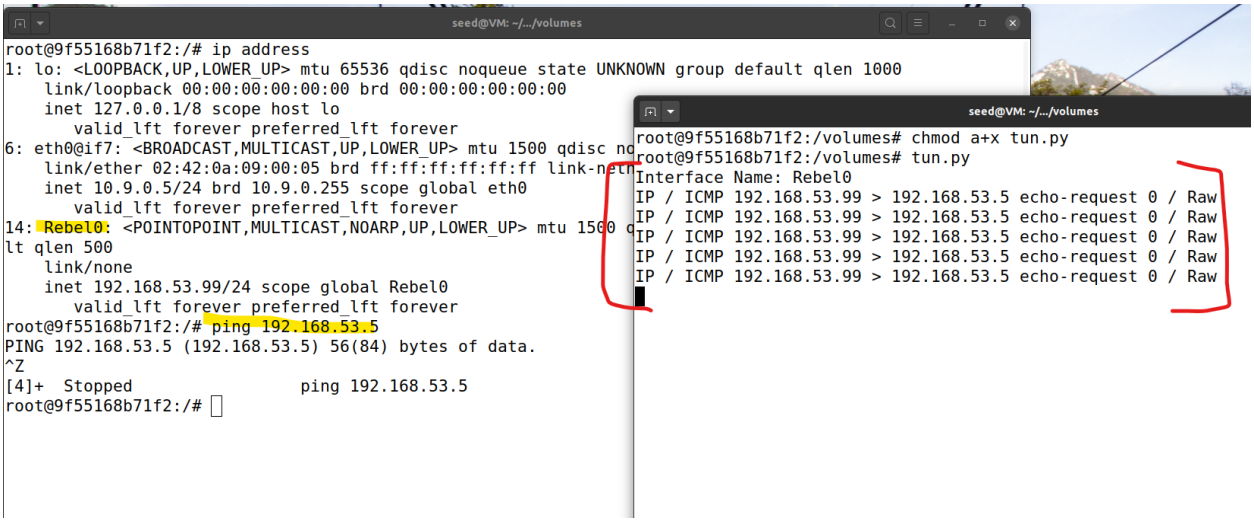
• On Host U, ping a host in the 192.168.53.0/24 network. What are printed out by the tun.py program? What has happened? Why?

• On Host U, ping a host in the internal network 192.168.60.0/24, Does tun.py print out anything? Why?

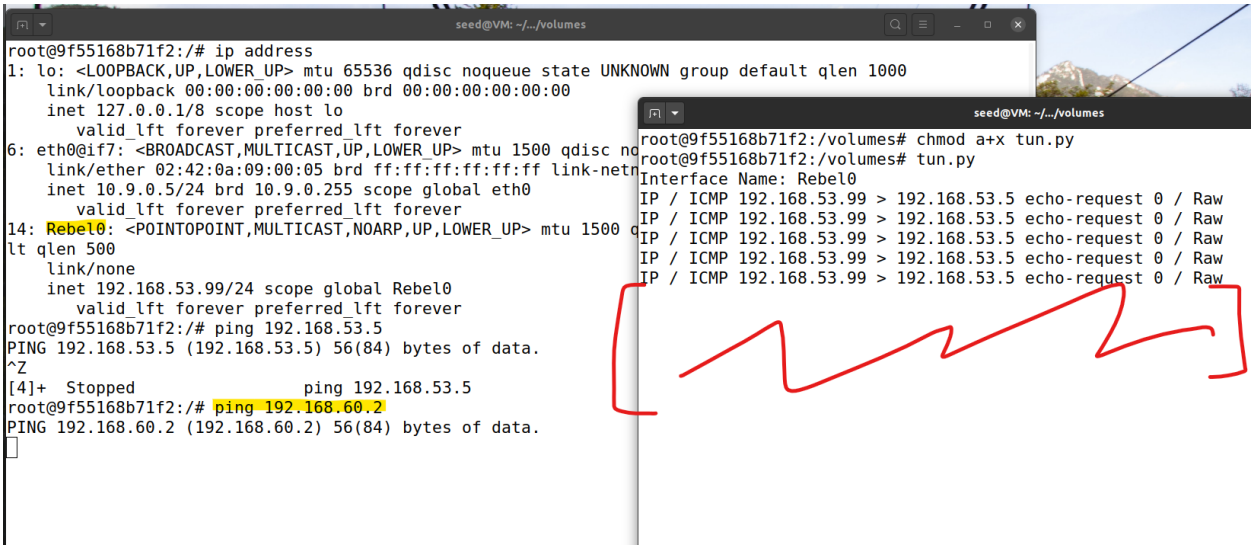
Below is the code, wherein I replace the while loop in order to cast the data received from the interface into a Scapy IP object, so we can print out each field of the IP packet.



On Host U, I ping a host in the 192.168.53.0/24 network. I ping 192.168.53.5 network. I notice that there were no packets that were received back. Hence the ping didn't get a reply. This is because the interface hasn't been configured entirely yet to receive packets. I see IP/ ICMP echo request packets going from the tun interface ip to the network I try to ping. The packets get sent because they are in the same LAN.



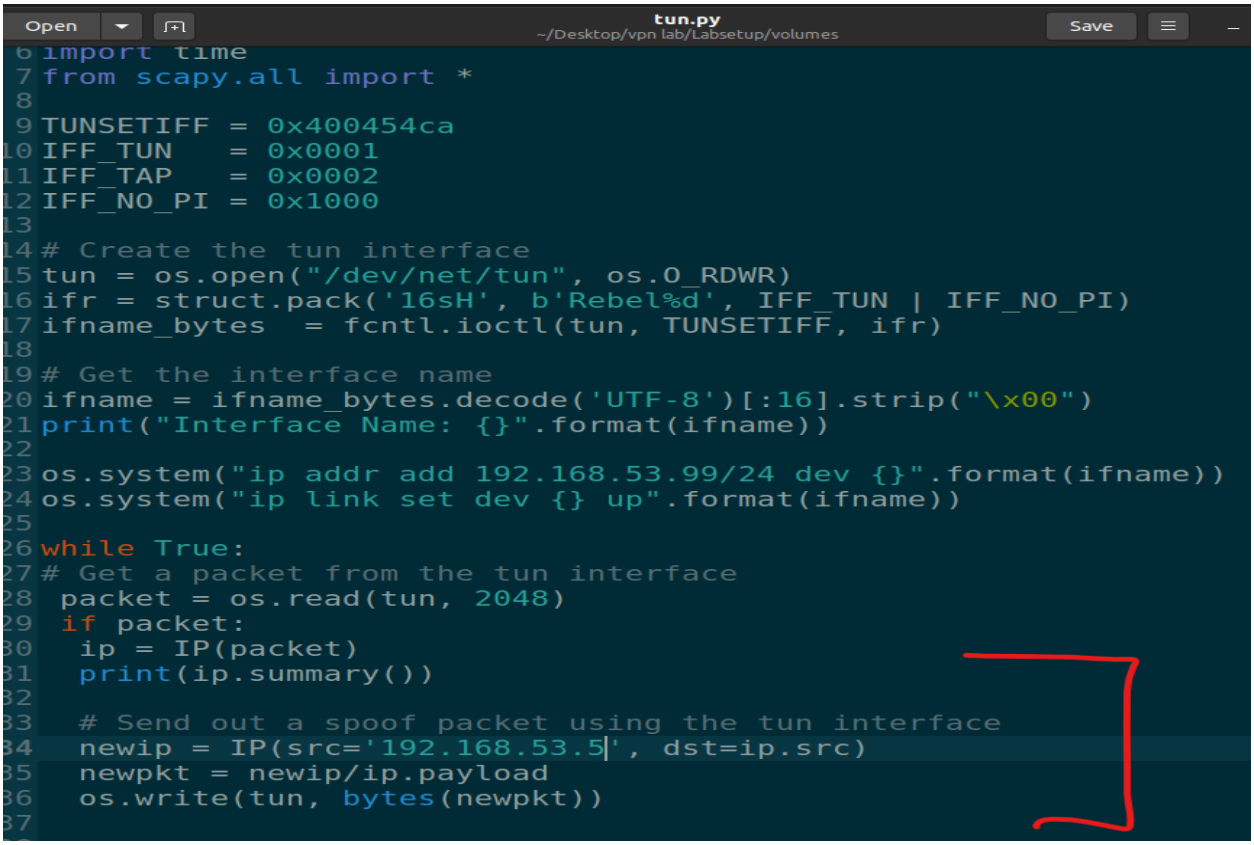
On Host U, I ping a host in the internal network 192.168.60.0/24, hence I ping 192.168.60.2 network. I notice that again I receive no reply back. What is different in this case is that I do not see any packets that get sent out and is blank, this is because the network I try to ping is outside the network of Host U and cannot be reached.



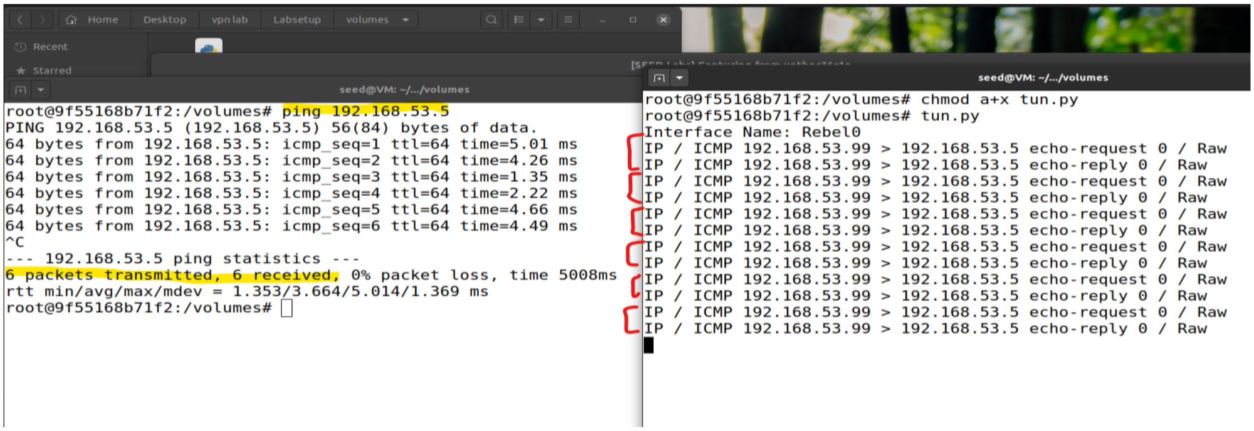
2d. Write to the TUN Interface

After getting a packet from the TUN interface, if this packet is an ICMP echo request packet, construct a corresponding echo reply packet and write it to the TUN interface. Please provide evidence to show that the code works as expected.

I add in the code to the while loop that enables me to send a spoofed packet with source ip as 192.168.53.5. Below we can see the changed code.



Below I try pinging 192.168.53.5 and notice that packets were received. The spoofed packets were generated and came back in as replies from 192.168.53.5 to the requests that were sent.

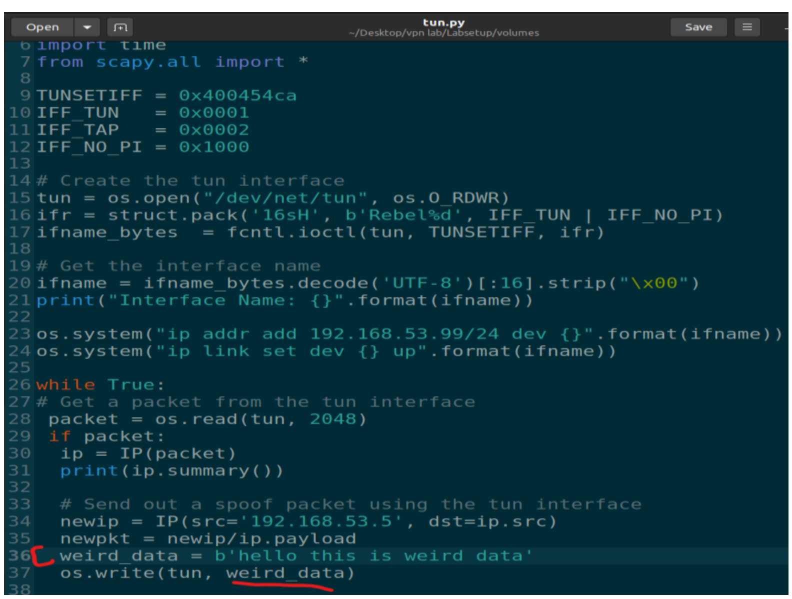


Instead of writing an IP packet to the interface, write some arbitrary data to the interface, and report your observation.

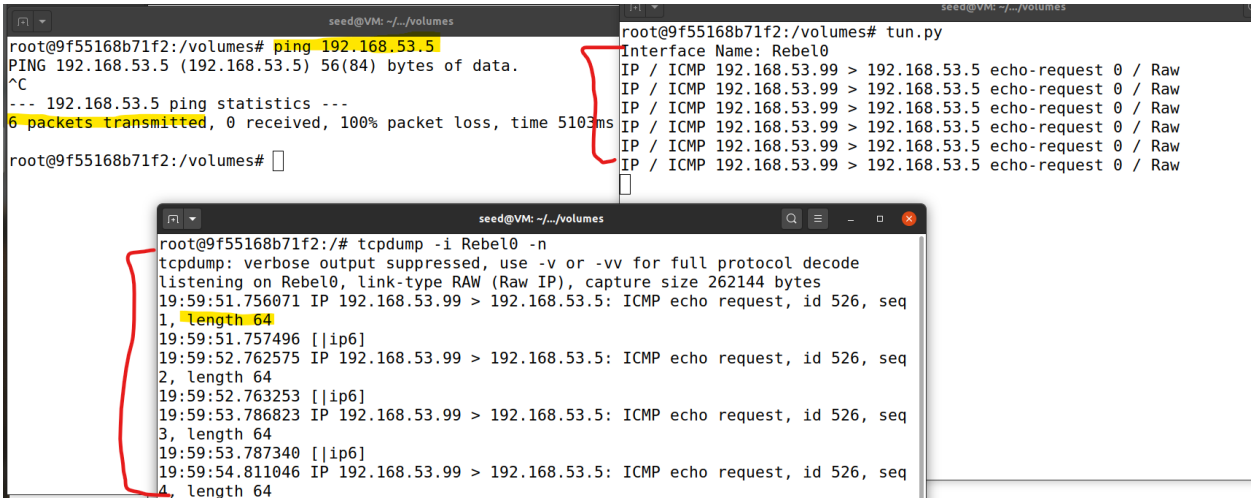
I do this task in two ways.

1. I put in some arbitrary data in the program itself and run the program and check the output.
2. I modify the same code to take input and then use netcat -u to connect to 192.168.53.5 network and see the output.

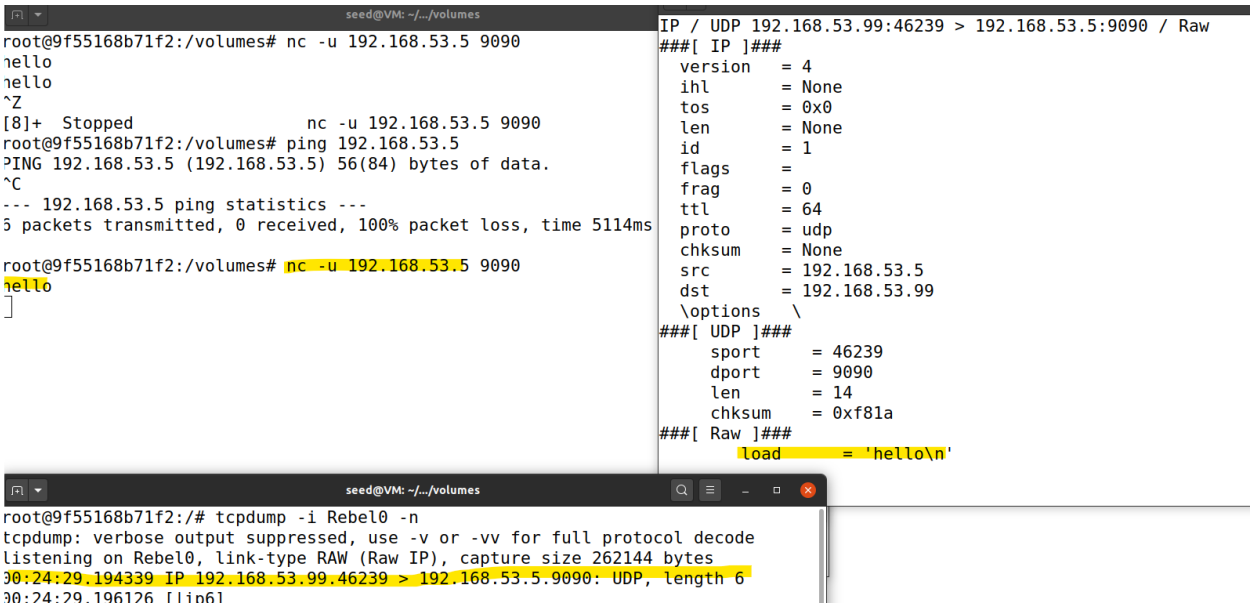
Below is the code for the first case, wherein I create a variable called weird data and provide the data I want to sent in the IP packet through bytes. I then run the program.



I then ping 192.168.53.5 to check and see the output. I notice the packets were successfully transmitted.



Now for the second case, I modify the same code, to accept input and send that input as data to the packet, I also add newpkt.show() at the end of the program, to show the details of the packet. I run tcpdump to sniff packets on the same machine. I notice that after the connection is successful and I give a input as 'hello' I am able to view the message sent and also in the packet sniff I am able to view the length of the packet.



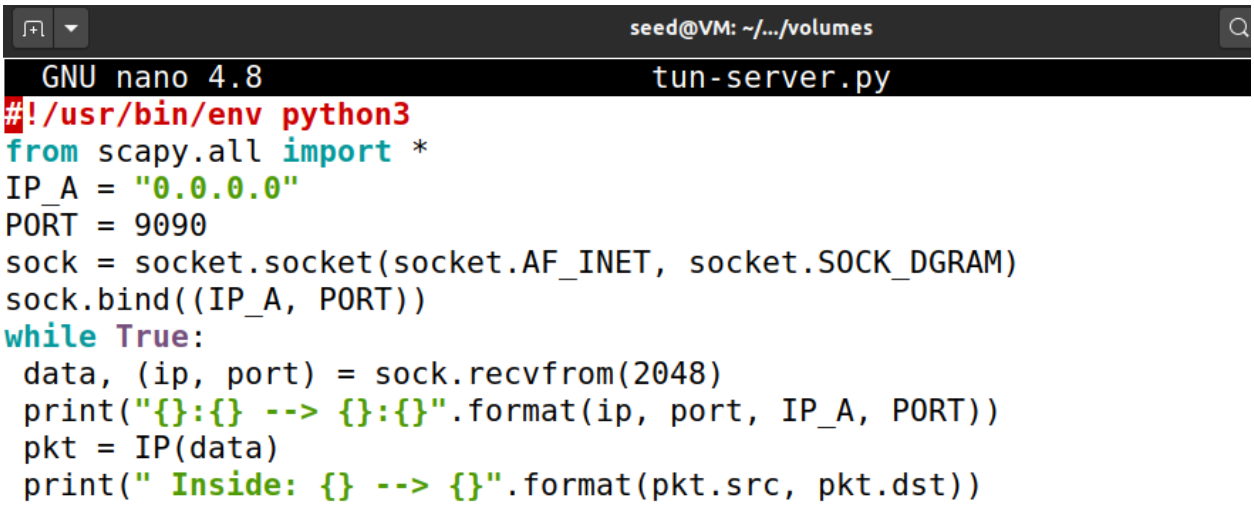
Task 3 Send the IP Packet to VPN Server Through a Tunnel

Run the tun server.py program on VPN Server, and then run tun client.py on Host U. To test whether the tunnel works or not, ping any IP address belonging to the 192.168.53.0/24 network. What is printed out on VPN Server? Why?

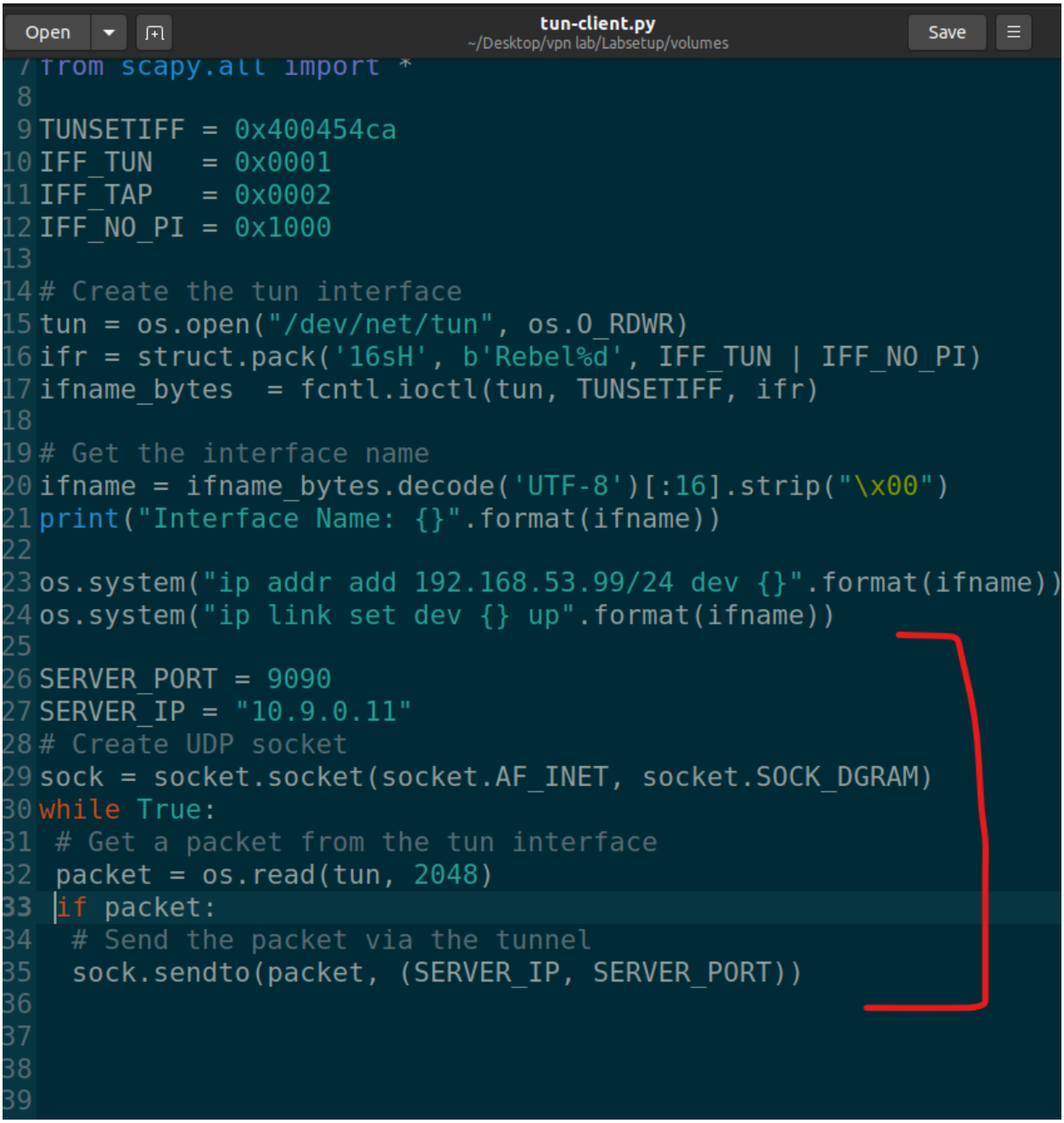
Our ultimate goal is to access the hosts inside the private network 192.168.60.0/24 using the tunnel. Let us ping Host V, and see whether the ICMP packet is sent to VPN Server through the tunnel. If not, what are the problems? You need to solve this problem, so the ping packet can be sent through the tunnel. This is done through routing, i.e., packets going to the 192.168.60.0/24 network should be routed to the TUN interface and be given to the tun client.py program.

The following command shows how to add an entry to the routing table: # ip route add dev via Please provide proofs to demonstrate that when you ping an IP address in the 192.168.60.0/24 network, the ICMP packets are received by tun server.py through the tunnel.

Tun-server.py script :

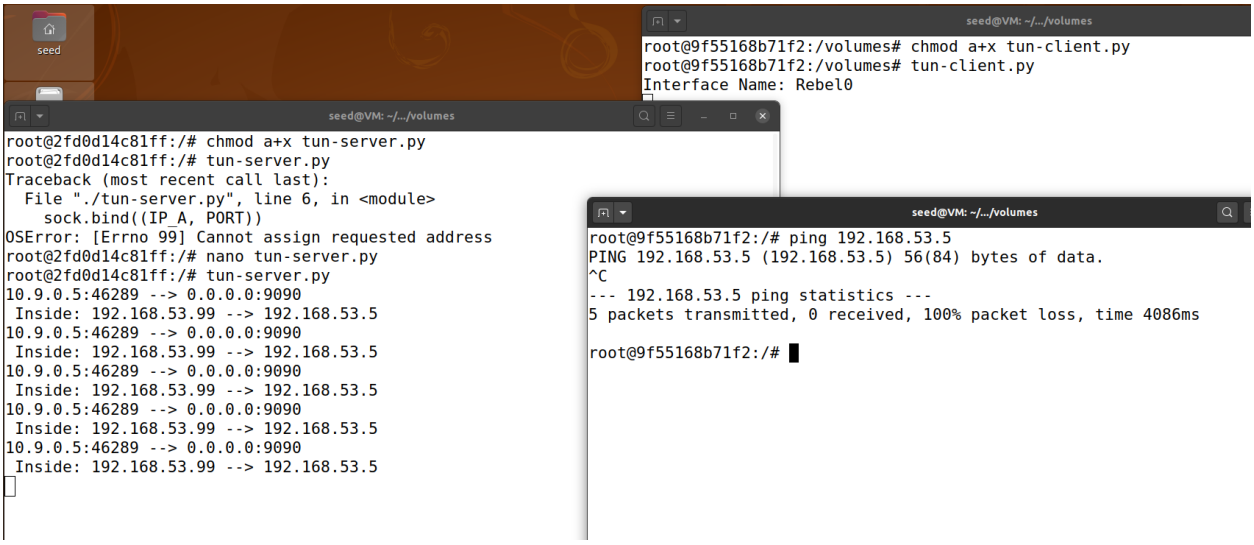


Tun-client.py script :



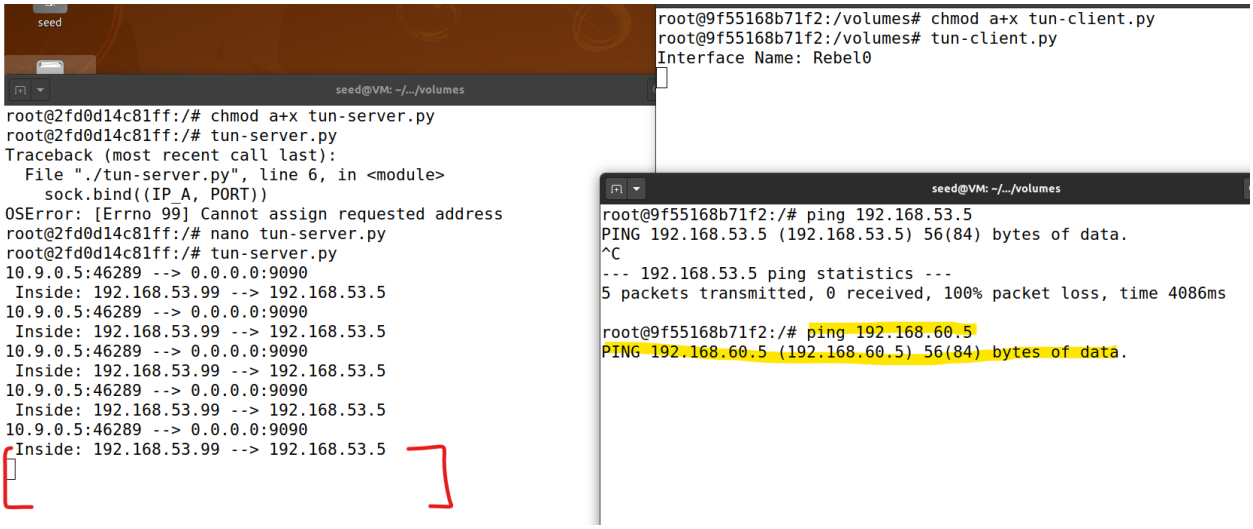
I replace the tun.py program with tun-client.py and changed the while loop to add in the client script. I added 10.9.0.11 as the ip for the server with 9090 as the port number.

I then run the server and client script and try to ping 192.168.53.5 IP and notice that the VPN server prints the source and destination and port along with the inner packet source and destination.

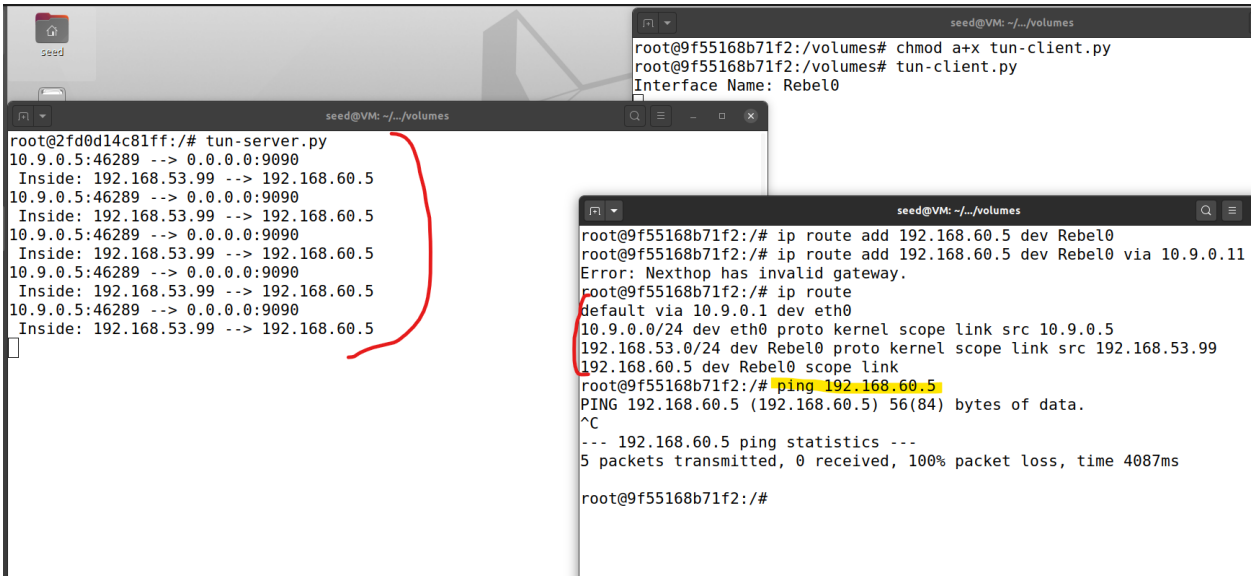


This happens because hout U has a tun interface with routing route that passes any data to this interface. Pinging an IP in this network passes through the tun interface and sent to the listening socket.

Below I try to ping host V in the private network, I notice that nothing is printed on the vpn server, this is because the routing terminal is not configured to pass packets through the Rebel0 interface.



After adding the routing via the commands mentioned below, I notice that now the packets have been configured to pass through the Rebel0 interface and hence can be noticed at the VPN server.



Task 4 : **Set Up the VPN Server**

Please modify tun server.py, so it can do the following:

- **Create a TUN interface and configure it.**
- **Get the data from the socket interface; treat the received data as an IP packet.**
- **Write the packet to the TUN interface.**

Testing: If everything is set up properly, we can ping Host V from Host U. The ICMP echo request packets should eventually arrive at Host V through the tunnel. Please show your proof. It should be noted that although Host V will respond to the ICMP packets, the reply will not get back to Host U, because we have not set up everything yet. Therefore, for this task, it is sufficient to show (using Wireshark or tcpdump) that the ICMP packets have arrived at Host V.

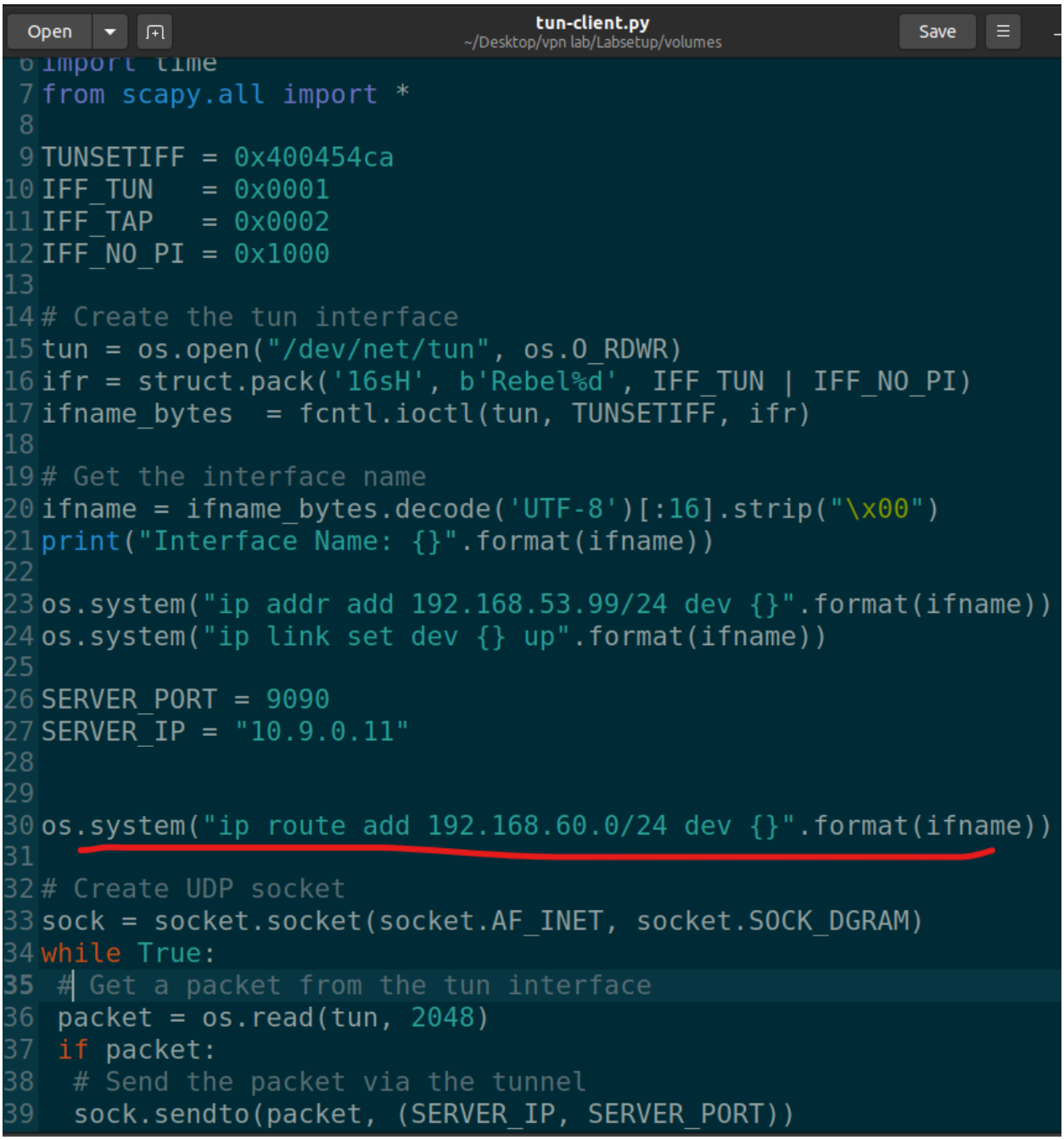
Tun-server.py script :

seed@VM: ~/.../volumes

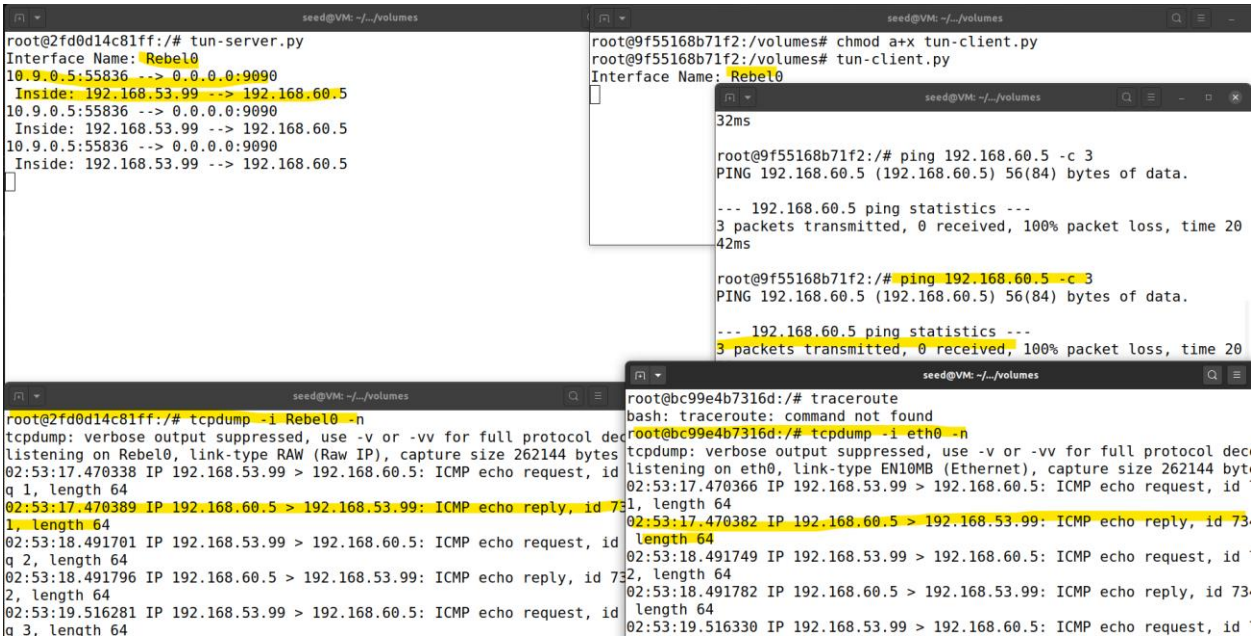
GNU nano 4.8 tun-server.py

```
TUNSETIFF = 0x400454ca
IFF_TUN   = 0x0001
IFF_TAP   = 0x0002
IFF_NO_PI = 0x1000

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'Rebel%d', IFF_TUN | IFF_NO_PI)
ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)

# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface Name: {}".format(ifname))

os.system("ip addr add 192.168.53.50/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

#UDP server
IP_A = "0.0.0.0"
PORT = 9090
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))
while True:
 data, (ip, port) = sock.recvfrom(2048)
 print("{}:{} --> {}:{}".format(ip, port, IP_A, PORT))
 pkt = IP(data)
 print(" Inside: {} --> {}".format(pkt.src, pkt.dst))

 os.write(tun, bytes(pkt))
```

Tun-Client.py script :



Below I try to ping the host V in the private network and I notice that I was able to ping host V from host U and I do receive replies as well. However, Host U is still unable to receive replies from Host V.



I run tcpdump on the eth0 interface on Host V to sniff the packets.

```
root@bc99e4b7316d:/# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
02:53:17.470366 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 734, seq
1, length 64
02:53:17.470382 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 734, seq 1,
 length 64
02:53:18.491749 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 734, seq
2, length 64
02:53:18.491782 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 734, seq 2,
 length 64
02:53:19.516330 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 734, seq
3, length 64
02:53:19.516362 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 734, seq 3,
 length 64
02:53:22.618725 ARP, Request who-has 192.168.60.11 tell 192.168.60.5, length 28
02:53:22.618798 ARP, Request who-has 192.168.60.5 tell 192.168.60.11, length 28
02:53:22.618813 ARP, Reply 192.168.60.5 is-at 02:42:c0:a8:3c:05, length 28
02:53:22.618821 ARP, Reply 192.168.60.11 is-at 02:42:c0:a8:3c:0b, length 28
```

Task 5 Handling Traffic in Both Directions

Testing: We should be able to communicate with Machine V from Machine U, and the VPN tunnel (un-encrypted) is now complete. Please show your wireshark proof using about ping and telnet commands. In your proof, you need to point out how your packets flow.

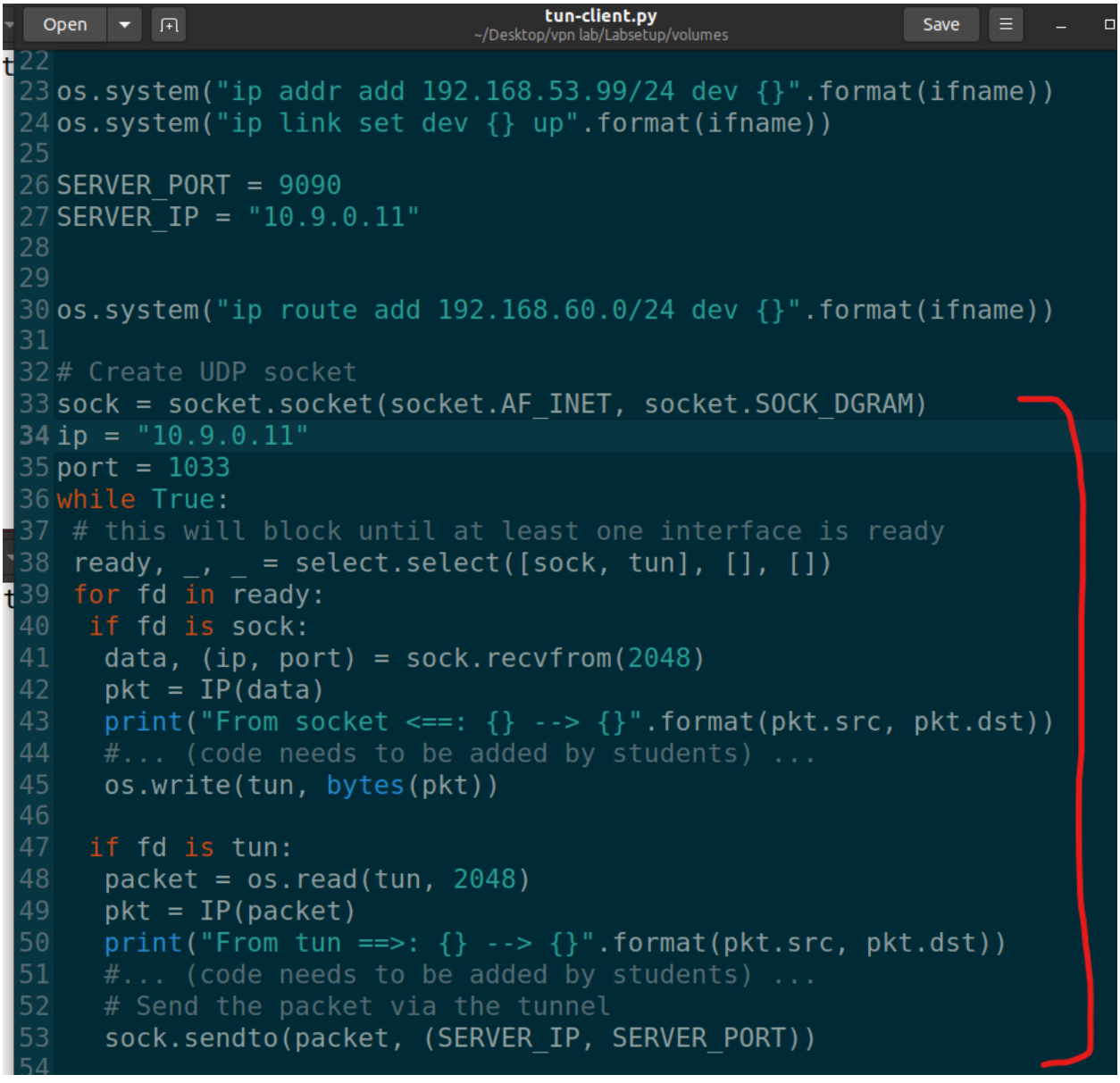
Tun-Server.py script :

seed@VM: ~/.../volumes

GNU nano 4.8 tun-server.py

```
print("Interface Name: {}".format(ifname))

os.system("ip addr add 192.168.53.50/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

#UDP server
IP_A = "0.0.0.0"
PORT = 9090
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))
ip = "10.9.0.5"
port = 1033
while True:
 # this will block until at least one interface is ready
 ready, _, _ = select.select([sock, tun], [], [])
 for fd in ready:
  if fd is sock:
   data, (ip, port) = sock.recvfrom(2048)
   pkt = IP(data)
   print("From socket <==: {} --> {}".format(pkt.src, pkt.dst))
   #... (code needs to be added by students) ...
   os.write(tun, bytes(pkt))

  if fd is tun:
   packet = os.read(tun, 2048)
   pkt = IP(packet)
   print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))
   #... (code needs to be added by students) ...
   sock.sendto(packet, (ip, port))
```

Tun-Client.py script :



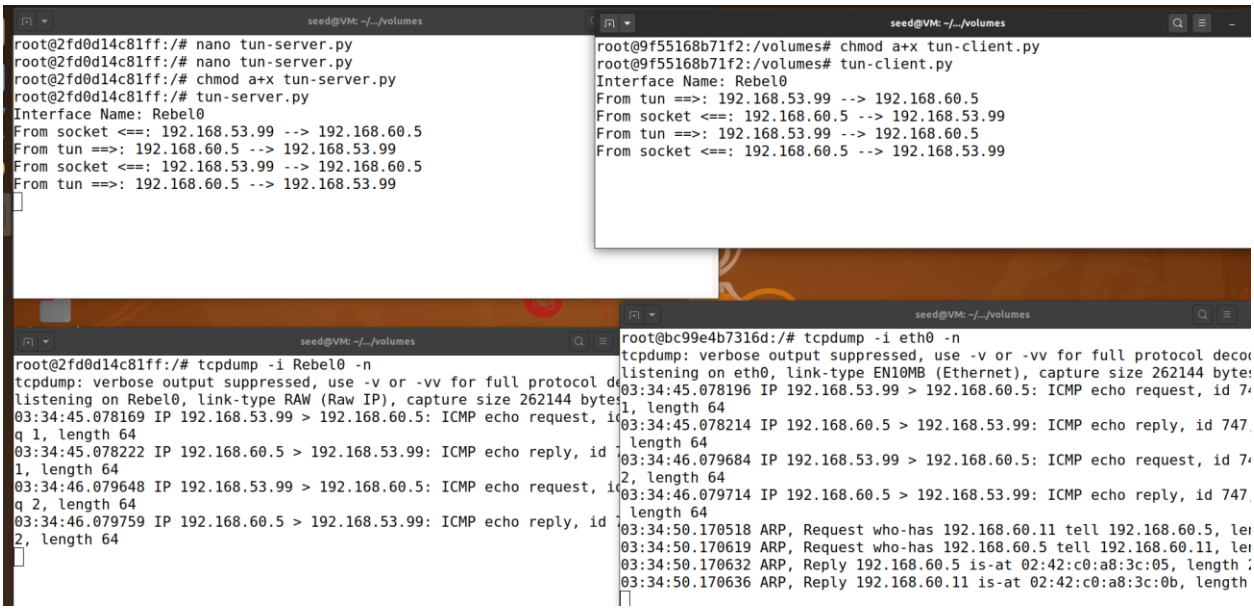
Ping

Below, I try to ping the private network host from Host U before the tun interface is setup. As shown I receive not replies. Now I setup the tun interface and then try to ping again.

```
root@9f55168b71f2:/# ping 192.168.60.5 -c 3
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 20
46ms

root@9f55168b71f2:/# ping 192.168.60.5 -c 2
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=4.29 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=5.78 ms

--- 192.168.60.5 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002
ms
rtt min/avg/max/mdev = 4.289/5.036/5.783/0.747 ms
root@9f55168b71f2:/#
```

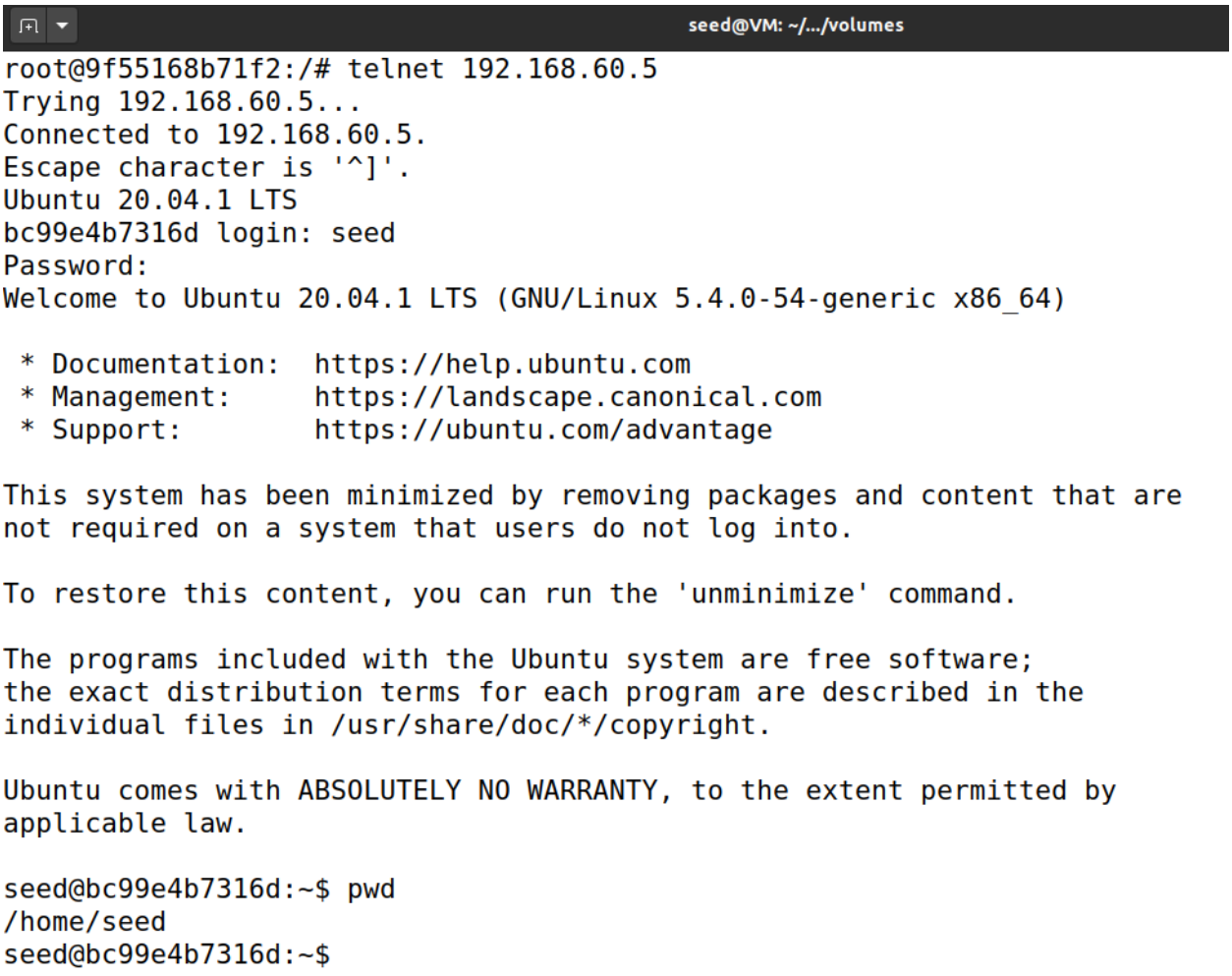
I notice that I received replies and the ping was successful. This means that I was able to connect to the Host V machine in the private network.

Below, I show the setup that shows everything setup and running and the information's for the successful ping.



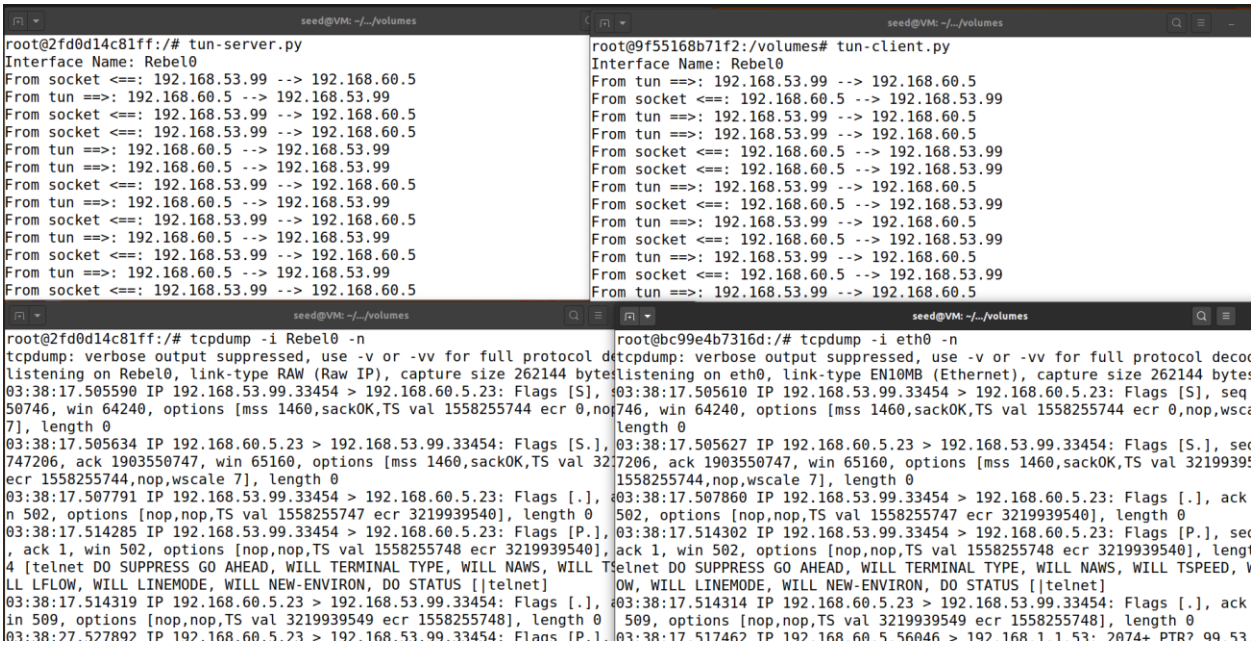
Telnet :

Now I try to telnet to Host V directly from Host U to see if it is now possible since the tunnel is now setup.



Yes, Host U was successfully able to telnet to Host V in the private network.

Below, is the information after the setup of the server and client script along with tcpdump on Host V machine to view the information during the telnet connection.

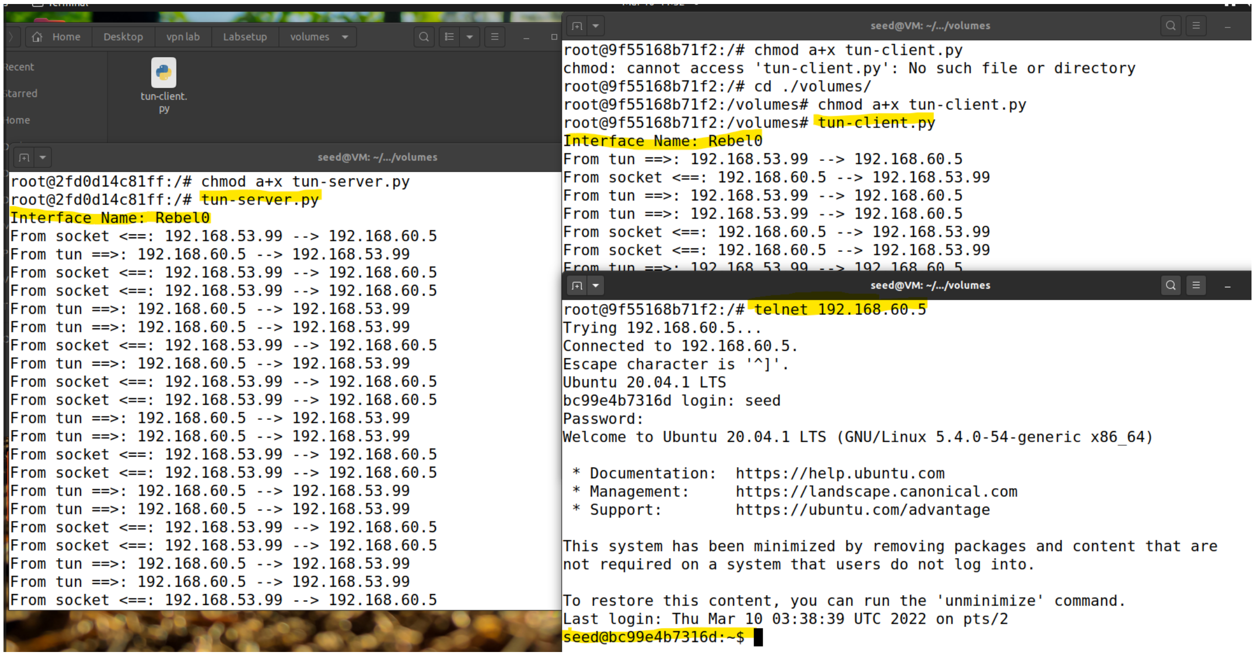


<u>Task 6</u> Tunnel-Breaking Experiment

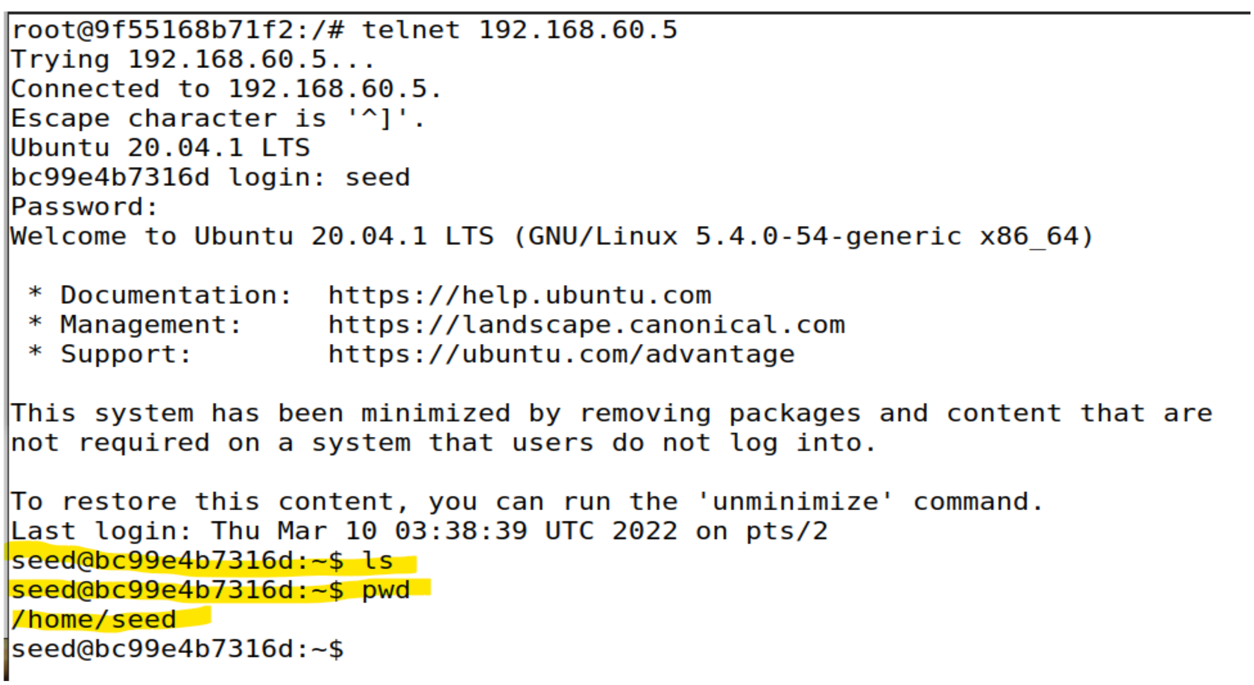
On Host U, telnet to Host V. While keeping the telnet connection alive, we break the VPN tunnel by stopping the tun client.py or tun server.py program. We then type something in the telnet window. Do you see what you type? What happens to the TCP connection? Is the connection broken? Let us now reconnect the VPN tunnel (do not wait for too long).

We will run the tun client.py and tun server.py programs again, and set up their TUN interfaces and routing. Once the tunnel is re-established, what is going to happen to the telnet connection? Please describe and explain your observations.

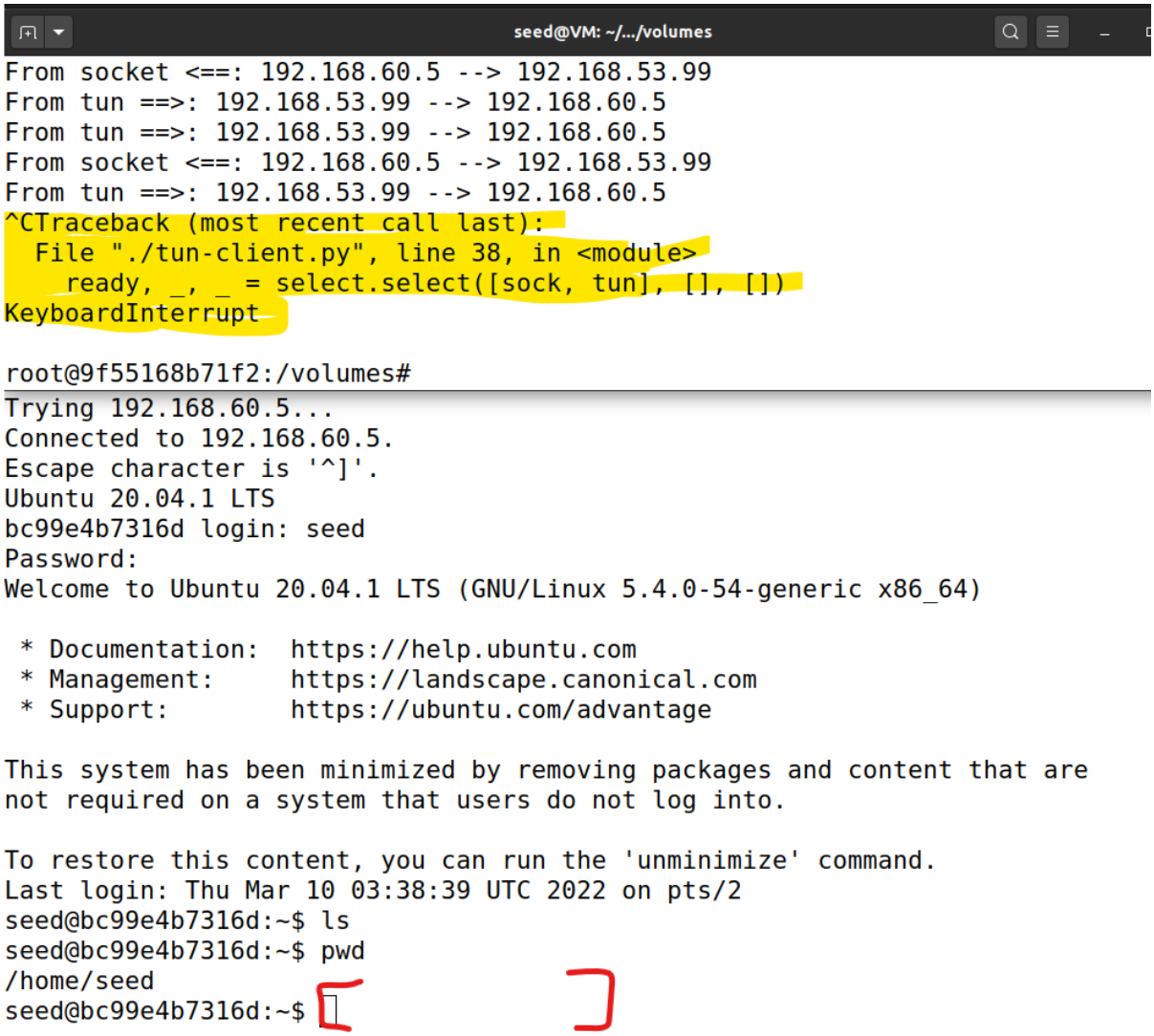
Below, I setup the tunnel by running the server and client scripts on the respective machines like in the previous task. I then telnet to Host V from Host U.



We see that the telnet connection was established successfully. And we can run commands on the Host V machine from Host U.

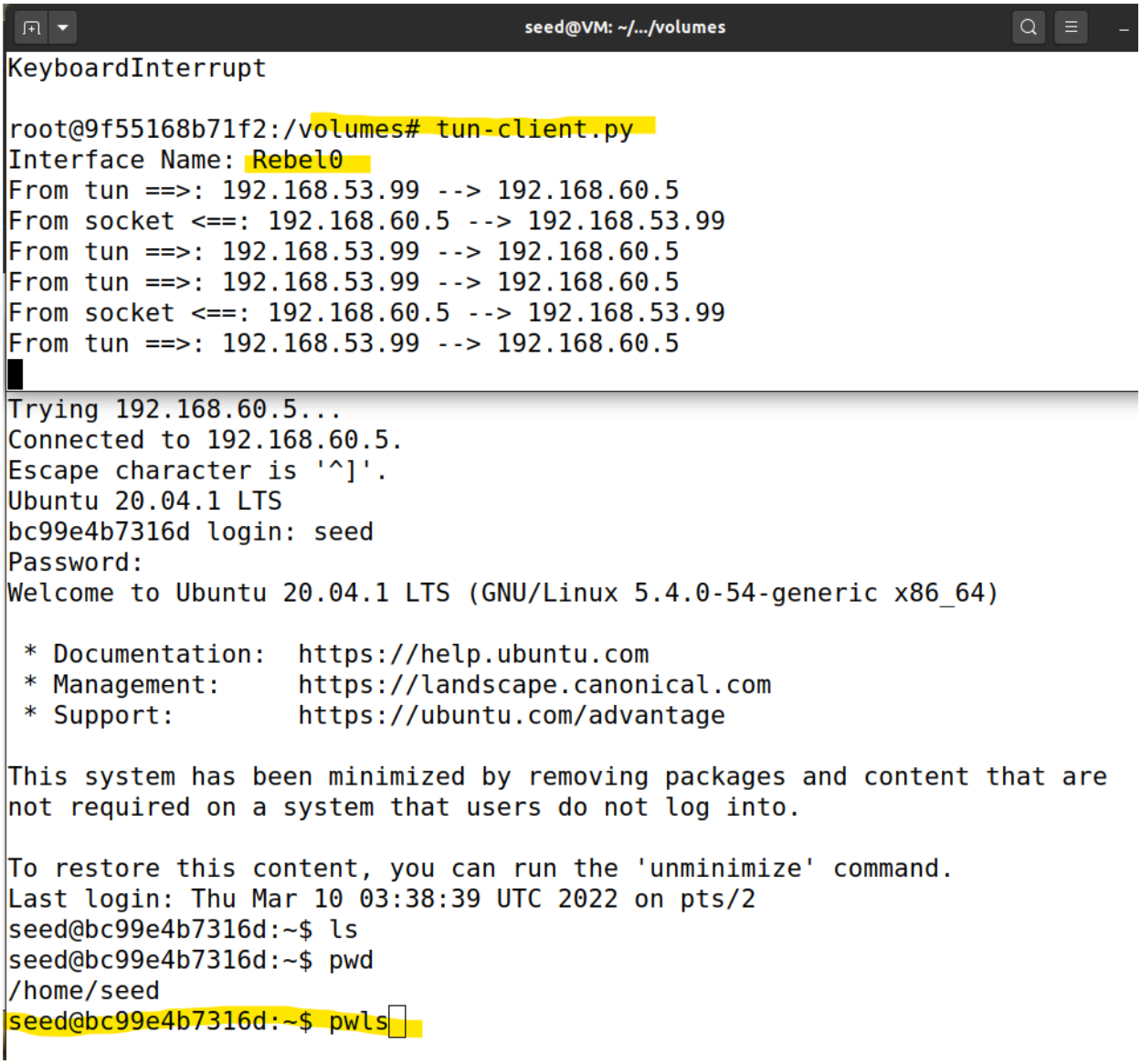


Now, I abruptly stop the client script on the Host U machine as shown below.



I try typing now and notice that there was no information was typed in that I tried to type. There was nothing getting displayed.

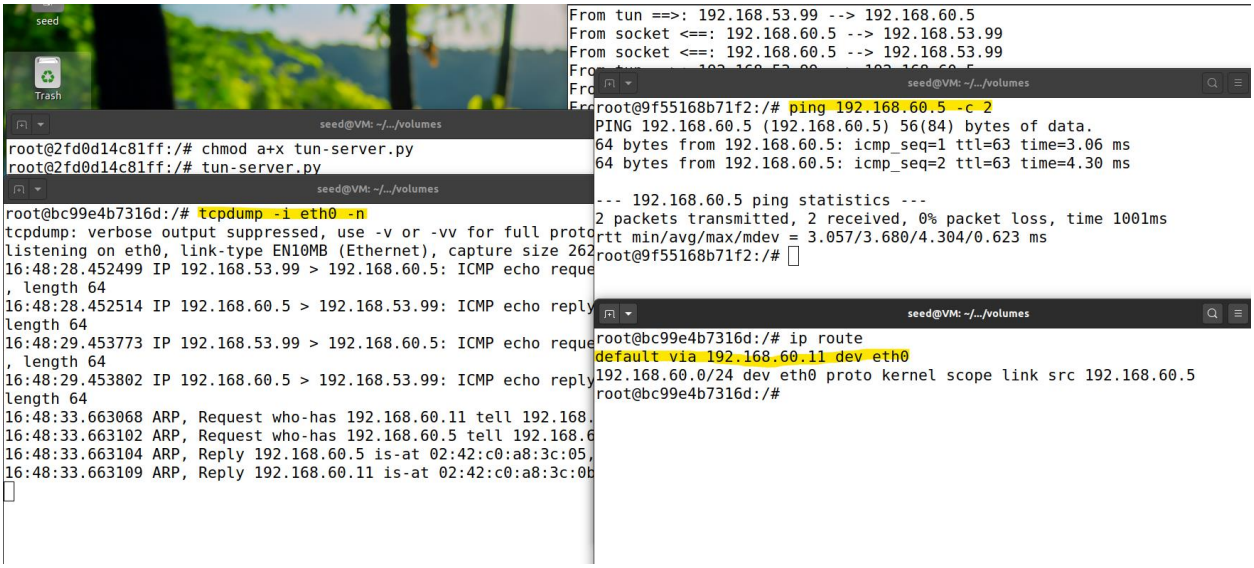
I then re run the client script on Host U as sown below. I notice that what ever I had typed initially was buffered and got pasted as soon as I regained connection.


seed@VM: ~/.../volumes
KeyboardInterrupt
root@9f55168b71f2:/volumes# tun-client.py
Interface Name: Rebel0
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
bc99e4b7316d login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage
This system has been minimized by removing packages and content that are
not required on a system that users do not log into.
To restore this content, you can run the 'unminimize' command.
Last login: Thu Mar 10 03:38:39 UTC 2022 on pts/2
seed@bc99e4b7316d:~$ ls
seed@bc99e4b7316d:~$ pwd
/home/seed
seed@bc99e4b7316d:~$ pwls

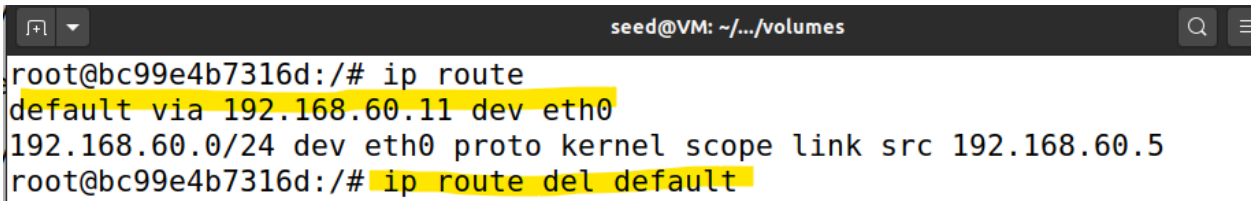
<u>Task 7</u> Routing Experiment on Host V

Routing tables inside a private network have to be set up properly to ensure that packets going to the other end of the tunnel will be routed to the VPN server. To simulate this scenario, we will remove the default entry from Host V, and add a more specific entry to the routing table, so the return packets can be routed back to the VPN server.

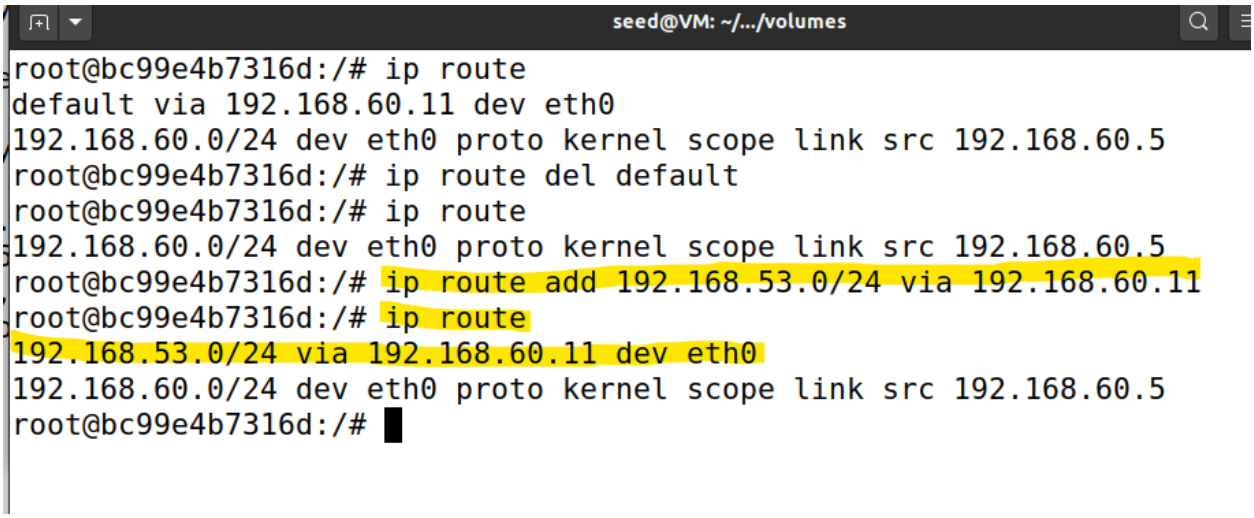
Below I setup the client and server scripts, on Host V I run the command ip route to check for the current or default route.



I notice that the default route is via 192.168.60.11 as shown below. I then run the command to delete the default. Now if I try to ping Host V from Host U I notice that a specific route is required to get things back working.



I then run the command below to add a more specific route. I run ip route again to check for the current route.

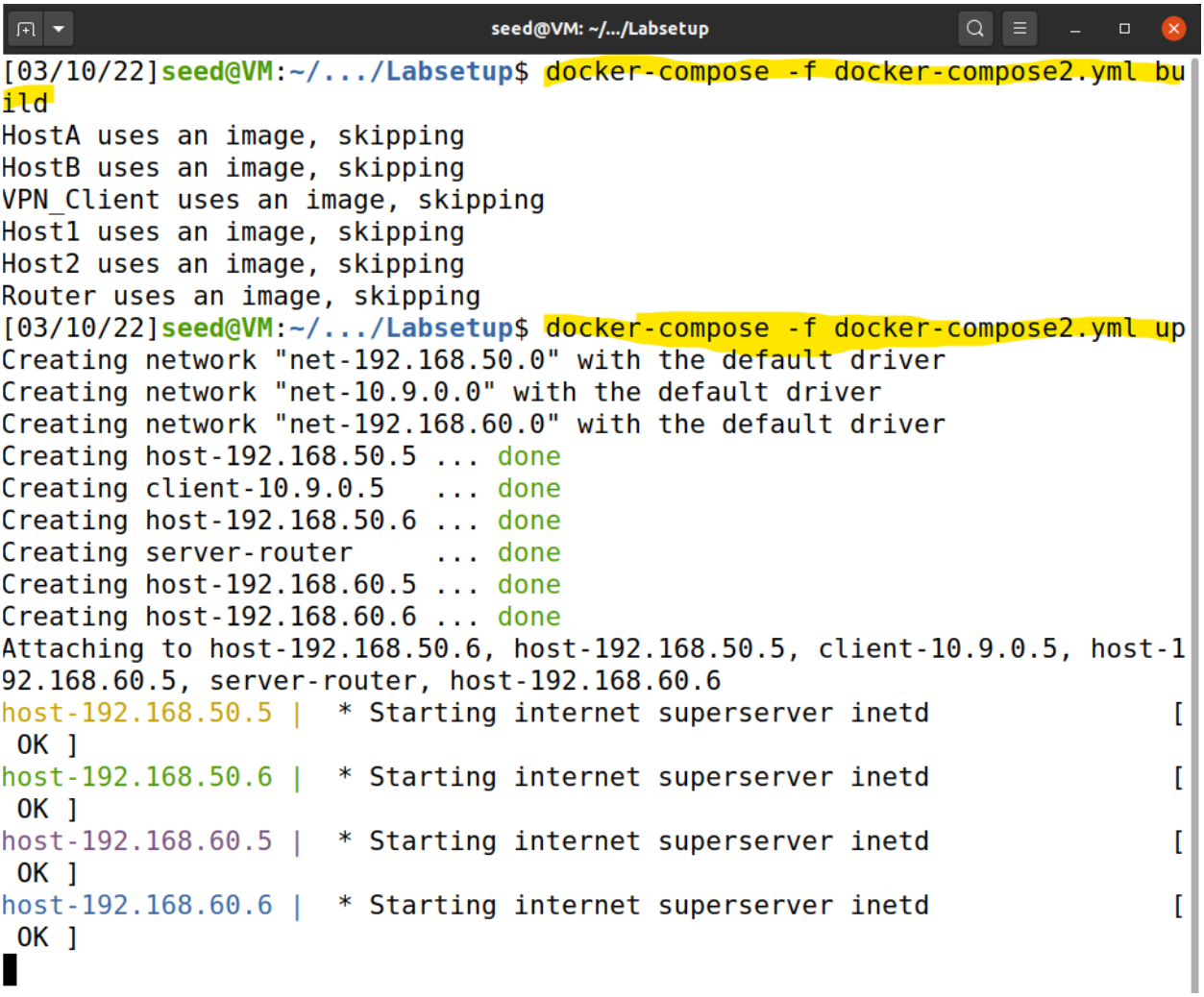


Now if I try to ping again I notice that the ping works again.

Task 8 VPN Between Private Networks

This setup simulates a situation where an organization has two sites, each having a private network. The only way to connect these two networks is through the Internet. Your task is to set up a VPN between these two sites, so the communication between these two networks will go through a VPN tunnel. You can use the code developed earlier, but you need to think about how to set up the correct routing, so packets between these two private networks can get routed into the VPN tunnel. In your report, please describe and explain what you did. You need to provide proofs to show that the packets between the two private networks are indeed going through a VPN tunnel.

In this task I setup a new environment using the new docker compose file and I then try to setup the vpn properly inorder for the communication to work successfully.



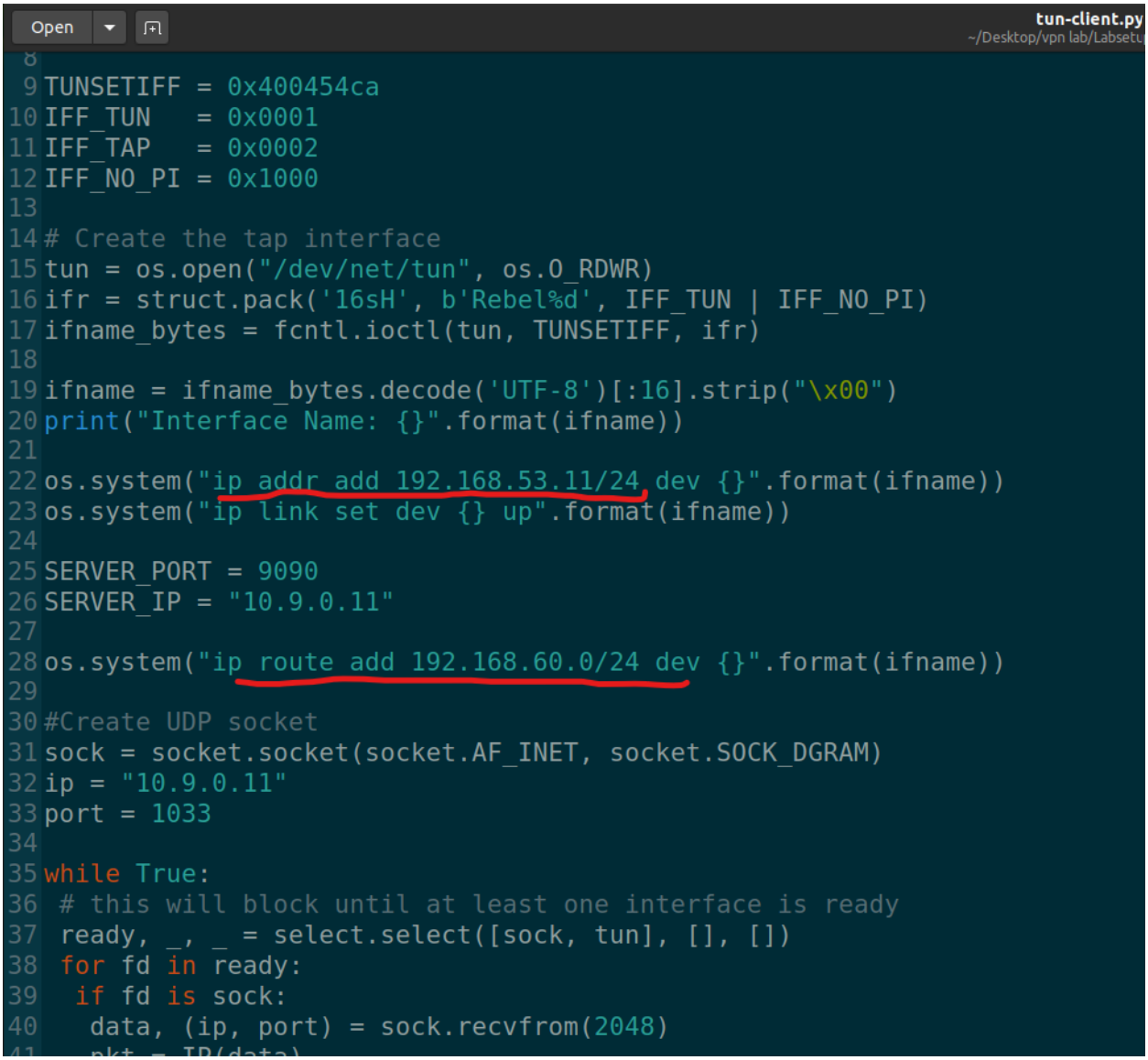
Environment :

```
[03/10/22]seed@VM:~/.../volumes$ dockps
2971d2609c7a  host-192.168.60.5
1b9d1a5f2105  host-192.168.60.6
4a2e5333d701  server-router
d36d3f344d78  host-192.168.50.6
c6fbe439443c  host-192.168.50.5
5323920a8491  client-10.9.0.5
```

Below, I try to ping the host V in the private network from Host U. Unfortunately since the vpn is not setup, we do not receive a reply.

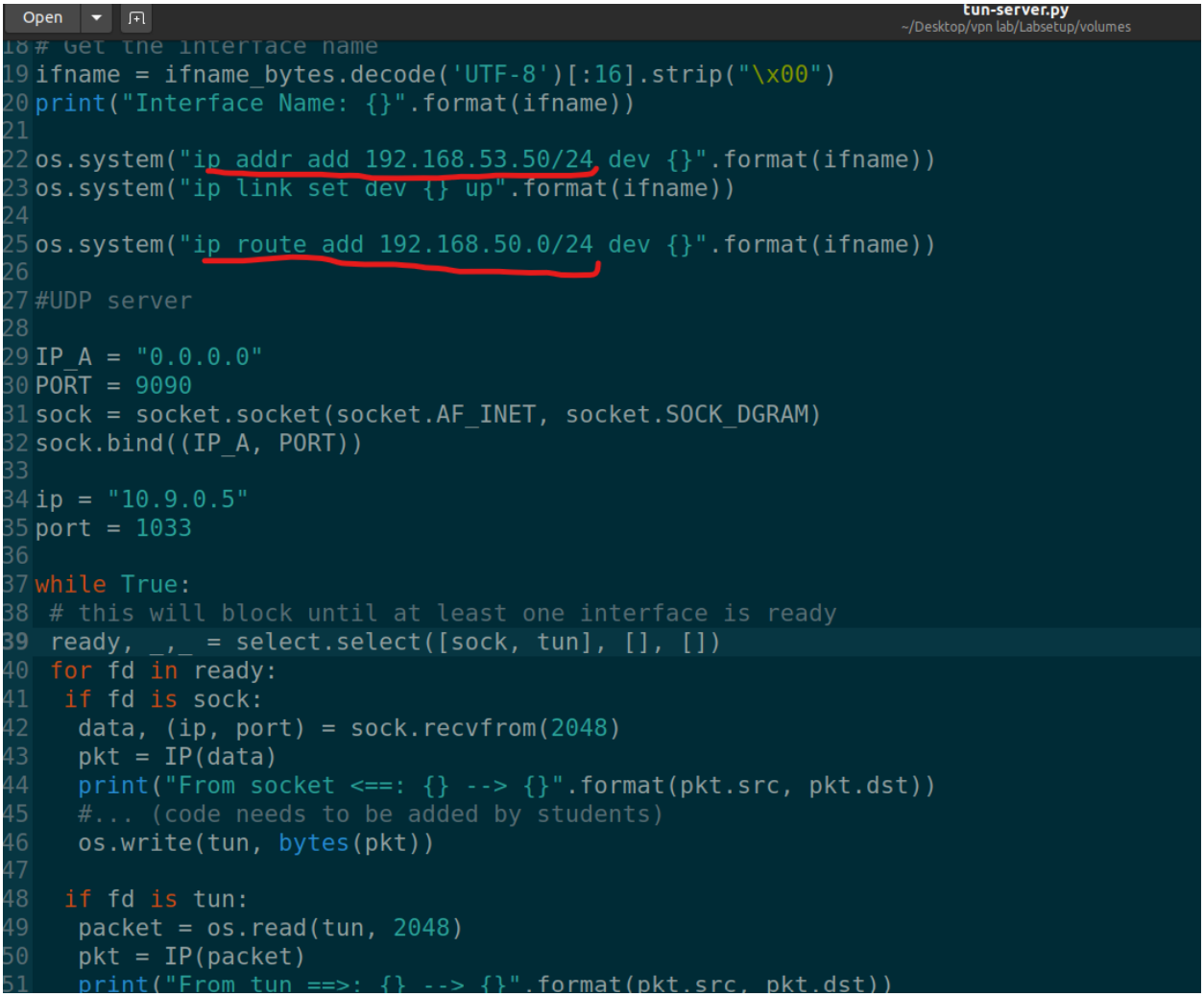
```
root@c6fbe439443c:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
```

Tun-Client.py script :



In the above code I route the packets to the private network through the ip route command.

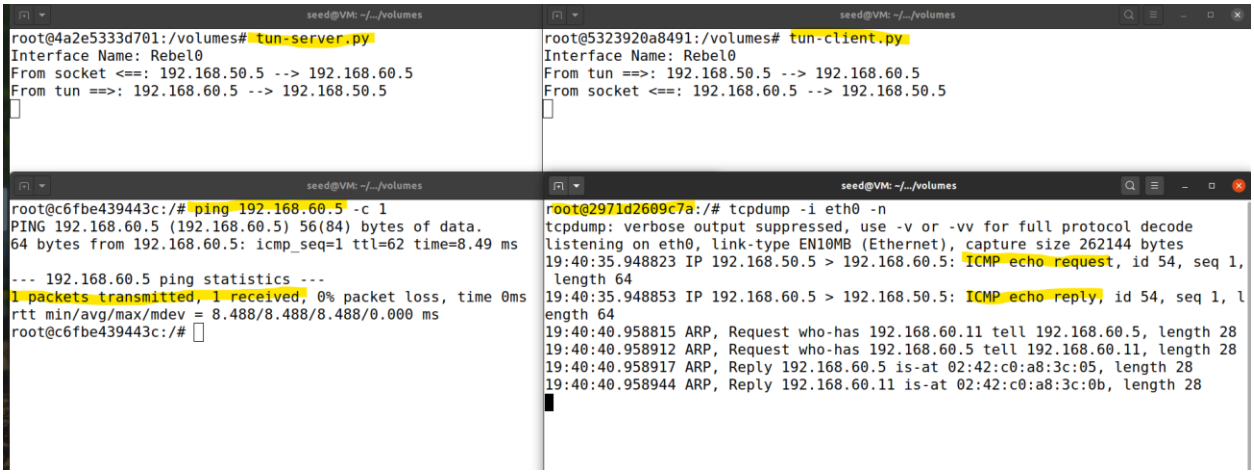
Tun-Server.py script :



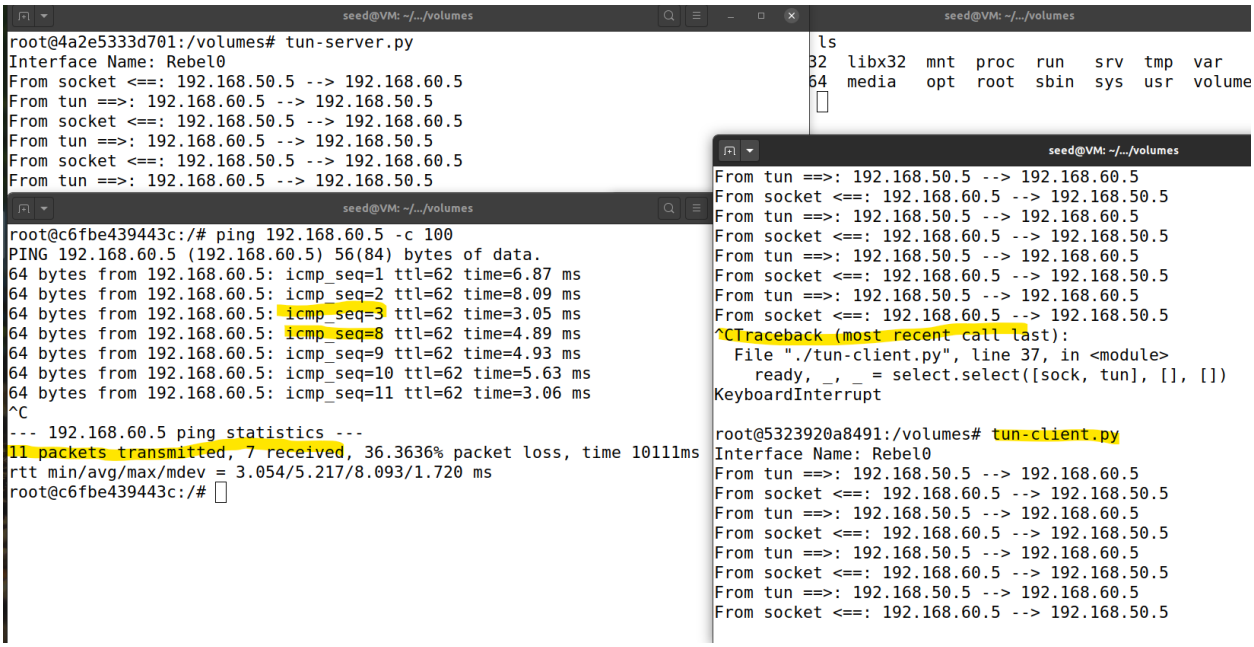
In the above code I added ip route command to redirect traffic to the tun interface.

In this task we need to setup routing properly so that the host can direct traffic to the vpn client / server.

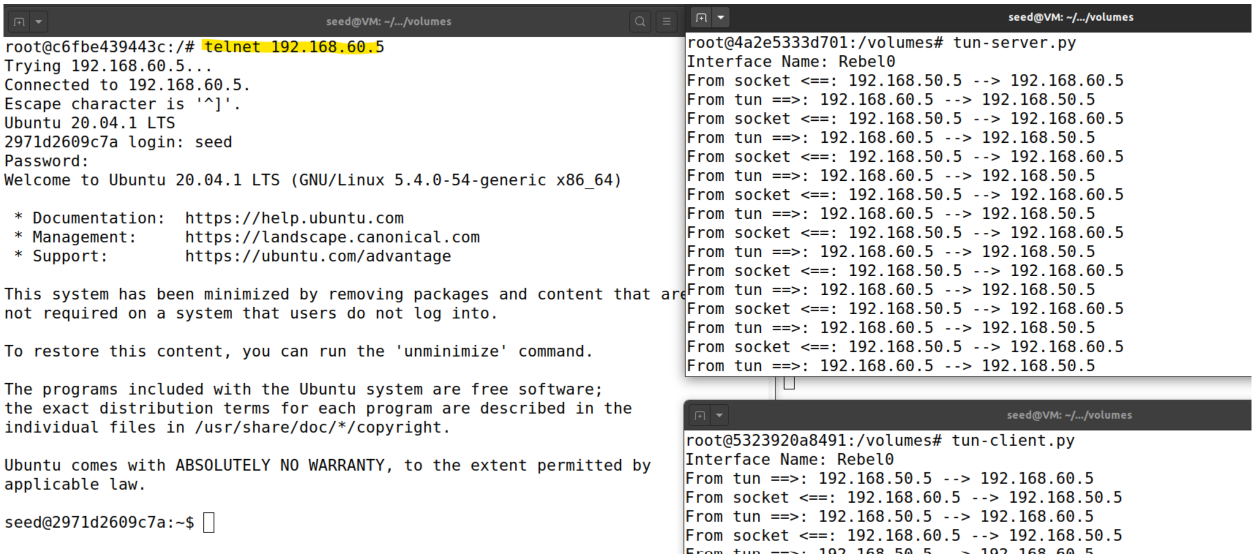
Below, I run the client and server scripts on the respective machines and notice that when I do ping Host V from Host U I am able to receive an ICMP reply.



Below, I do a test to see if the VPN is set up properly. Similarly, to the task wherein I break the telnet connection and re establish the connection, I do that here as well through the ping command. I keep the pinging Host V from Host U and abruptly stop the client program and then re run the program. The connection starts back again. Notice the difference in icmp sequence numbers as well as the net packets transmitted and received.



I now try to telnet to host V from Host U to see if the VPN was configured properly. Thereafter below screen shot shows that the telnet was successful between both the machines.

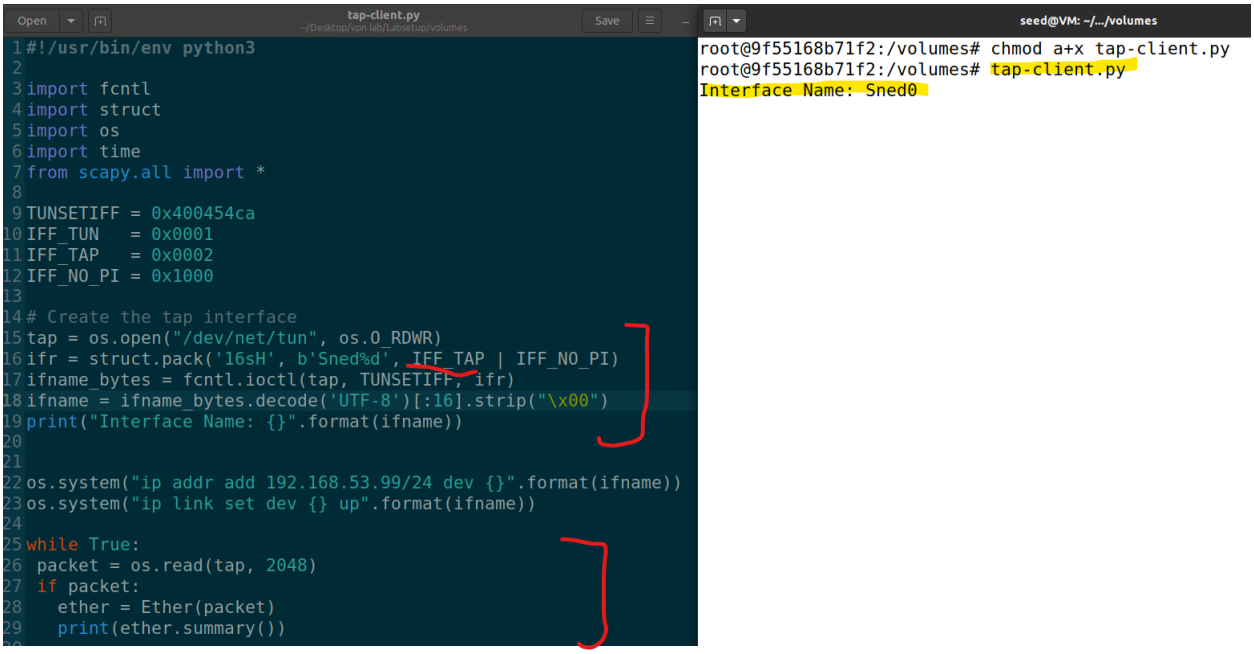


<u>Task 9</u> Experiment with the TAP Interface

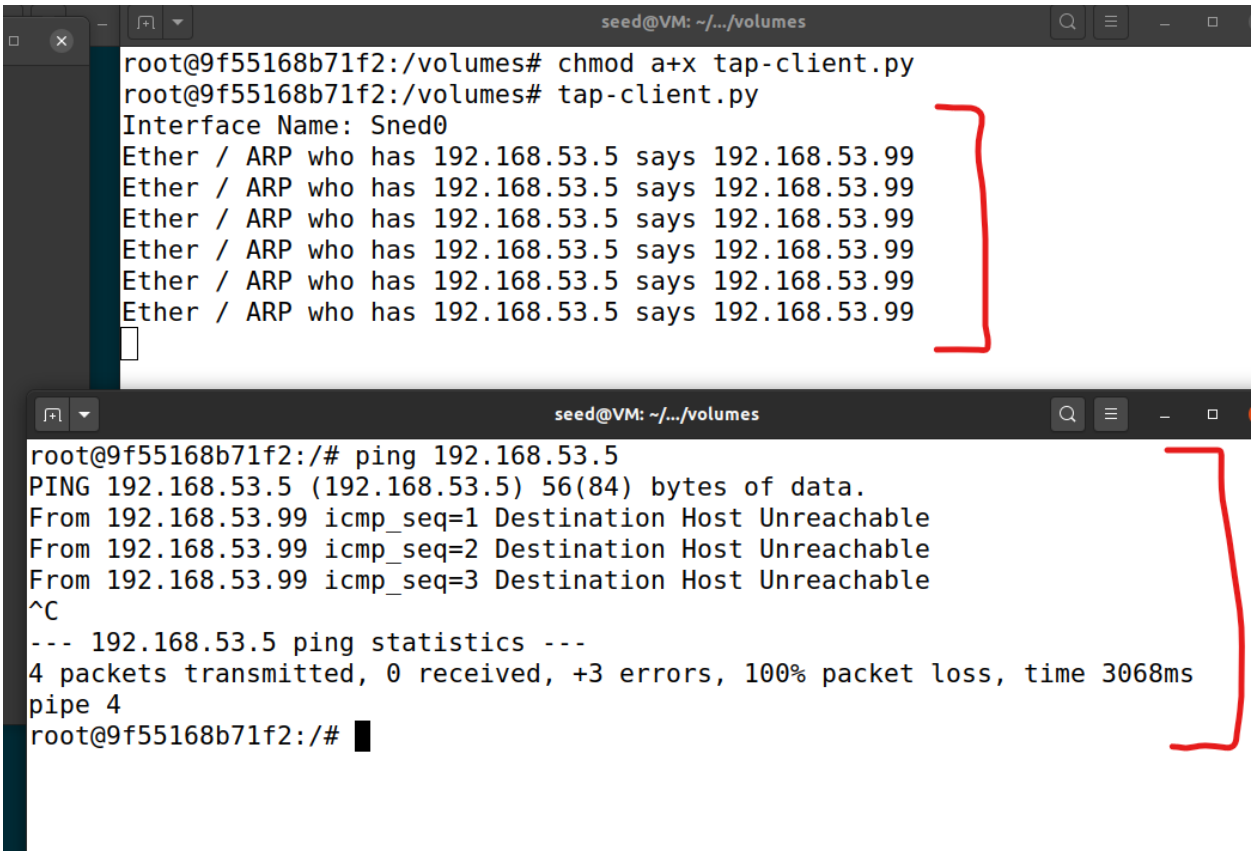
The code above simply reads from the TAP interface. It then casts the data to a Scapy Ether object, and prints out all its fields. Try to ping an IP address in the 192.168.53.0/24 network; report and explain your observations. To make this more interesting, once you get an ethernet frame from the TAP interface, you can check whether it is an ARP request; if it is, generate a corresponding ARP reply and write it to the TAP interface.

test your TAP program, you can run the arping command on any IP address. This command sends out an ARP request for the specified IP address via the specified interface. If your spoof-arp-reply TAP program works, you should be able to get a response.

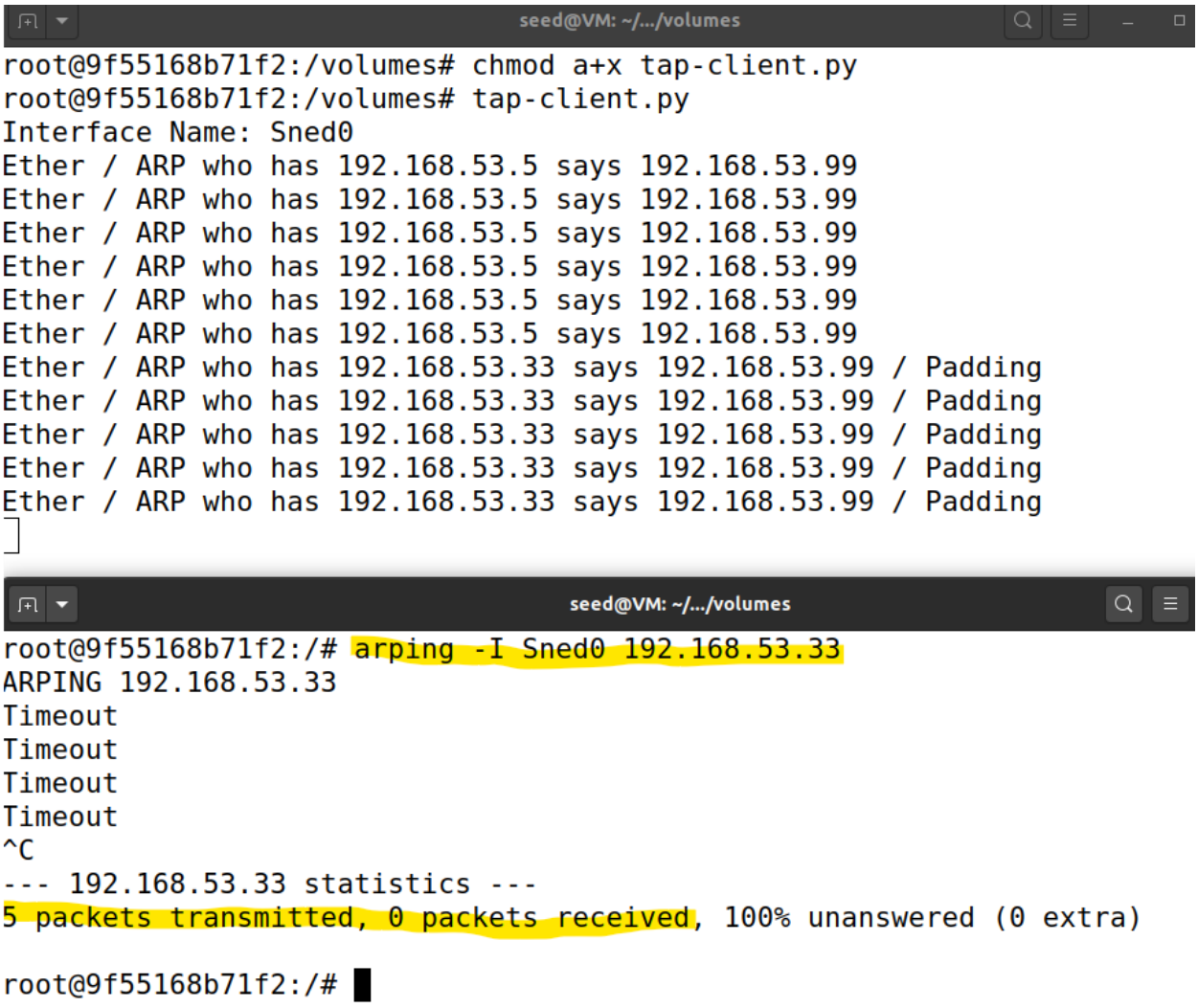
Below, shows the client program with the Tap interface setup as Sned0 (my first name).



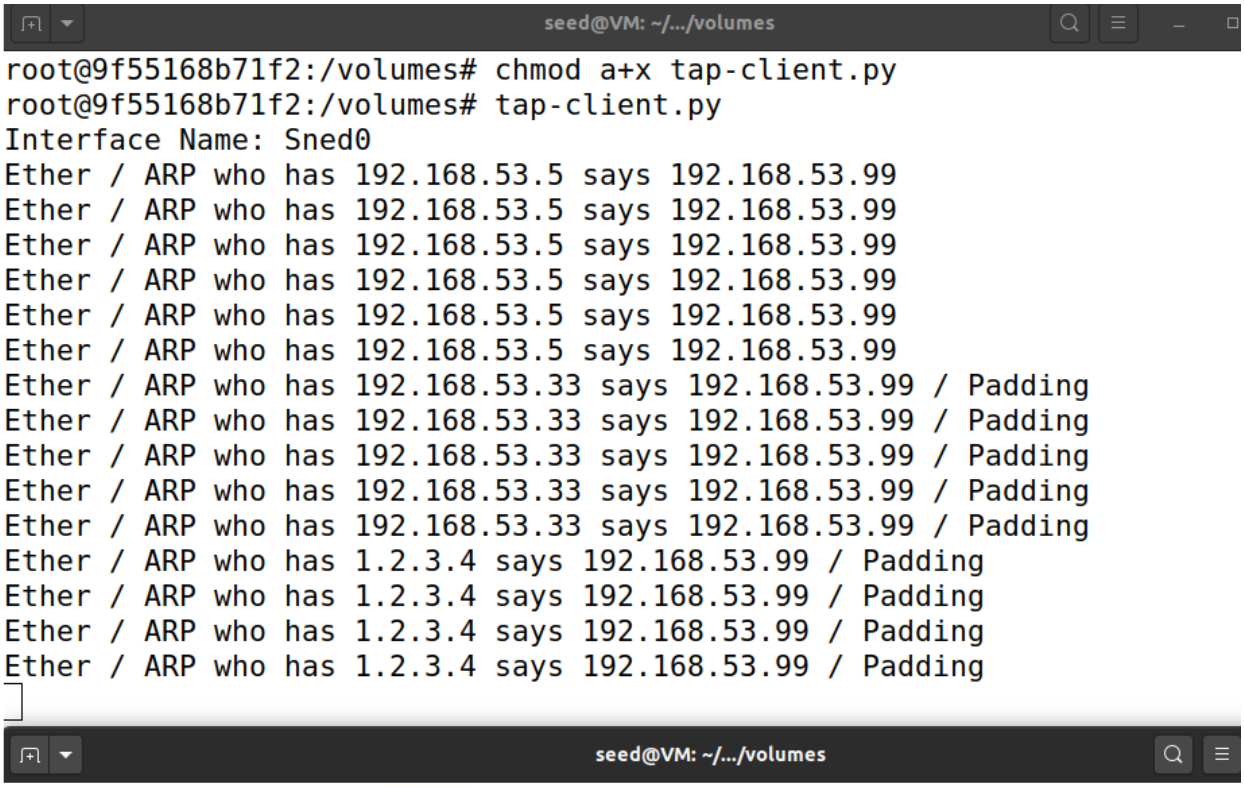
Below, I ping the 192.168.53.5 network from Host U. I notice that the tap Sned0 interface detects ARP frames. The packets are routed to the tunnel interface. The client is unsure of any machine with that ip hence ARP requests were sent with a message of hosts unreachable error.

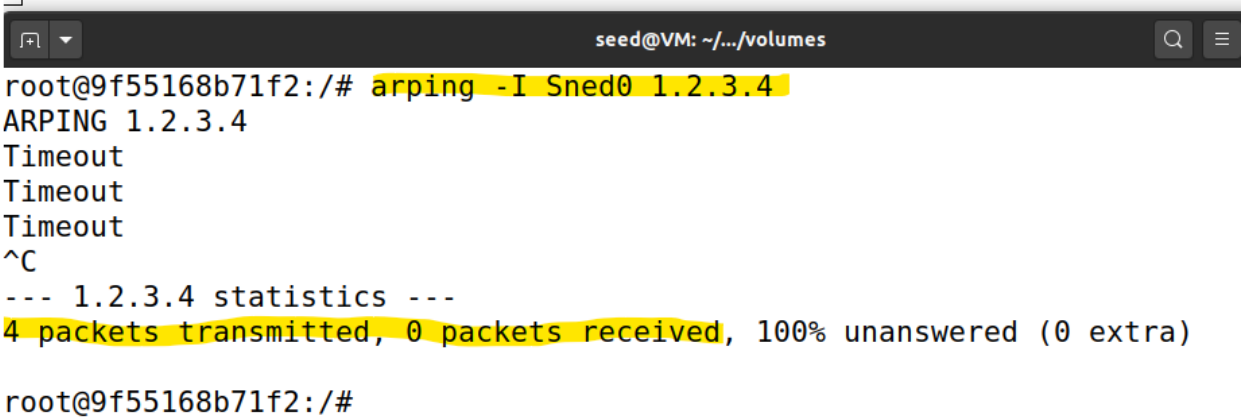


Below I try to arp ping without sending spoofed arp packets. First to 192.168.53.33 ip and then to 1.2.3.4 ip.

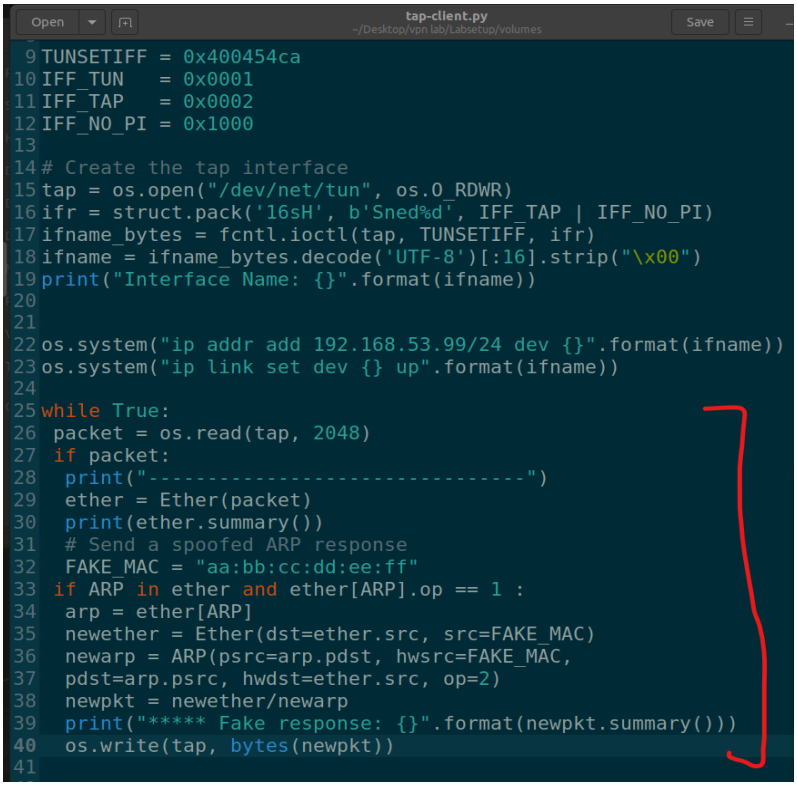


Below I try to arping 1.2.3.4 ip.

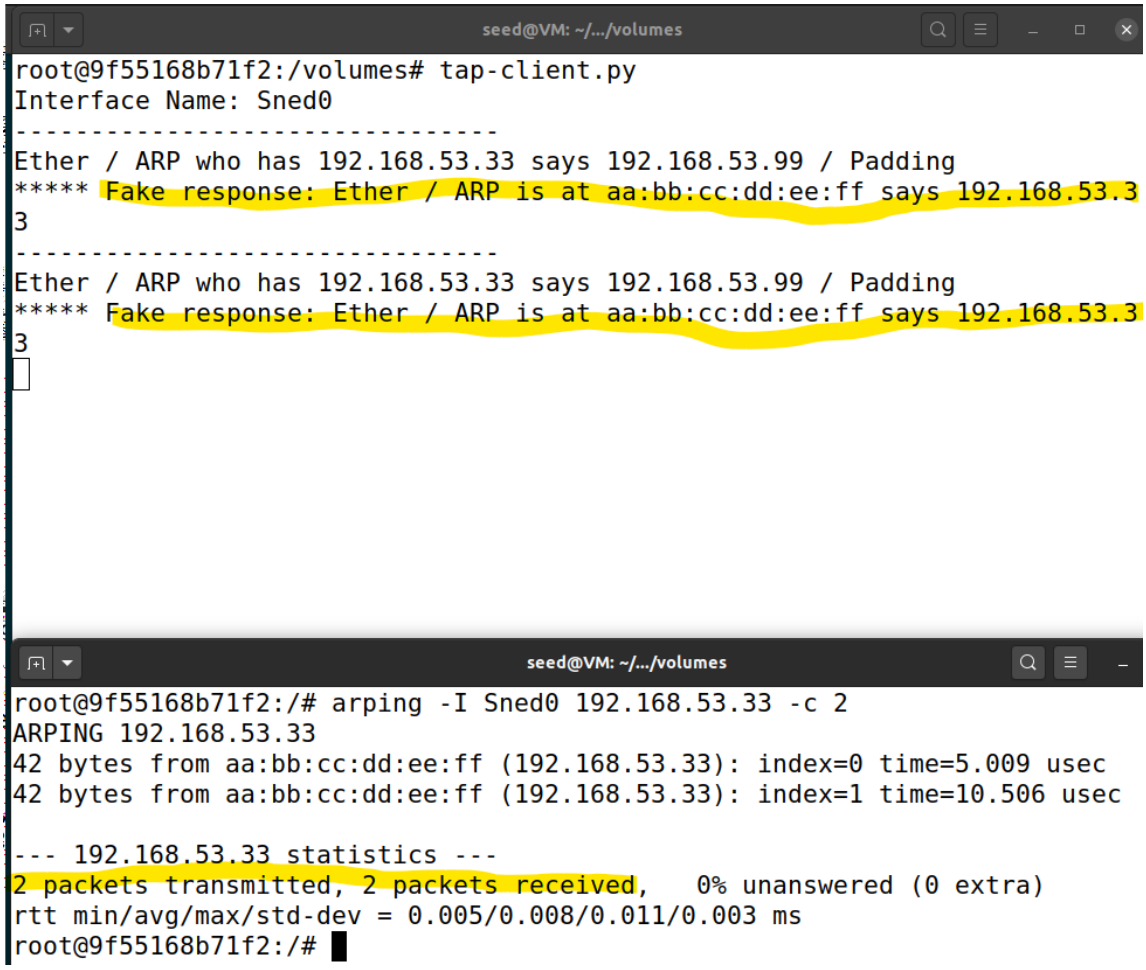




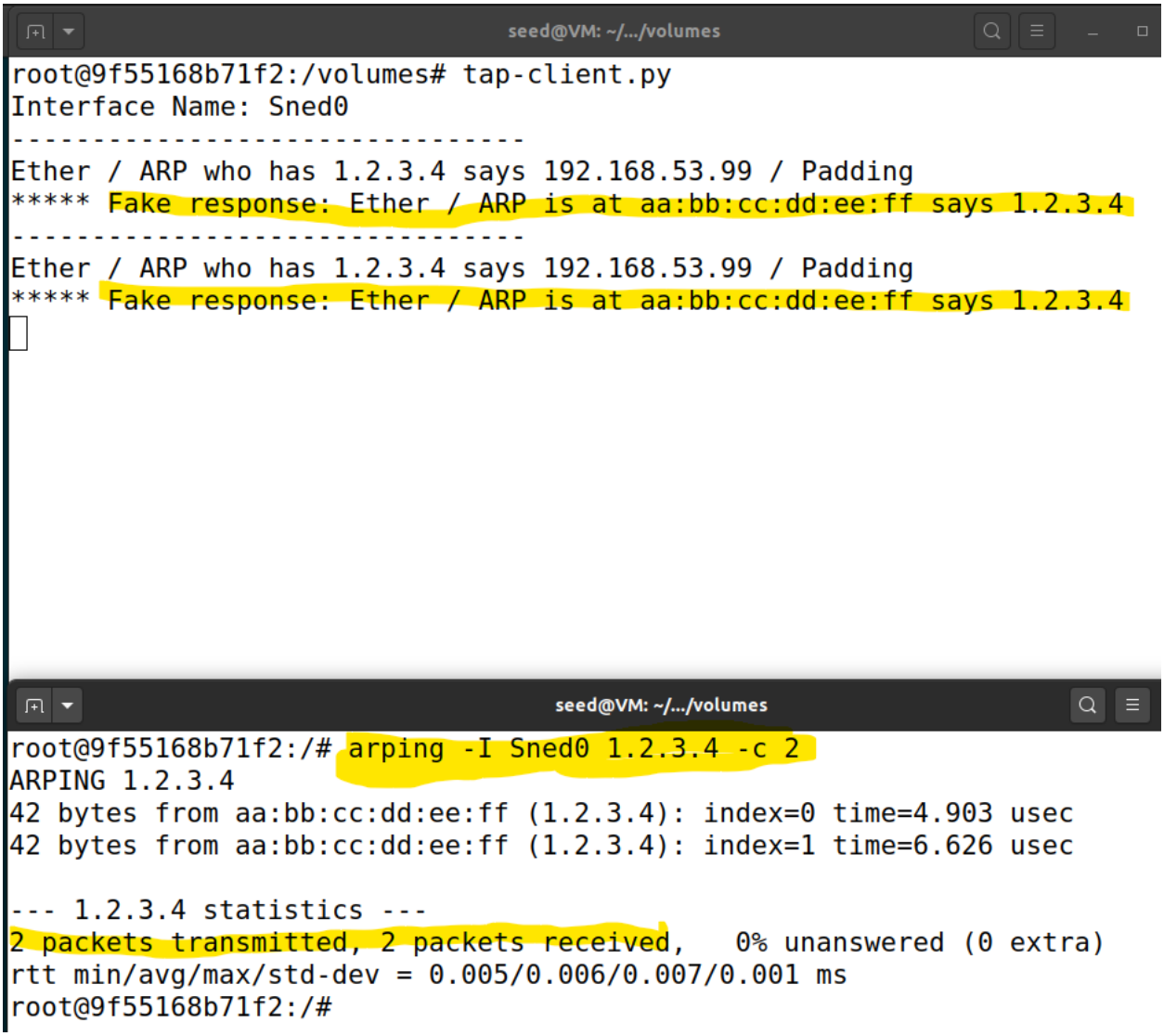
Now in the client program I add the code for sending spoofed arp packets from a fake mac address.



I then run the client program and do a arping to 192.168.53.33 and receive the spoofed replies from the faked mac address for every request sent.



I then do a arping to 1.2.3.4 and receive the spoofed replies from the faked mac address for every request sent.


seed@VM: ~/.../volumes
root@9f55168b71f2:/volumes# tap-client.py
Interface Name: Sned0

Ether / ARP who has 1.2.3.4 says 192.168.53.99 / Padding
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 1.2.3.4

Ether / ARP who has 1.2.3.4 says 192.168.53.99 / Padding
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 1.2.3.4
seed@VM: ~/.../volumes
root@9f55168b71f2:/# arping -I Sned0 1.2.3.4 -c 2
ARPING 1.2.3.4
42 bytes from aa:bb:cc:dd:ee:ff (1.2.3.4): index=0 time=4.903 usec
42 bytes from aa:bb:cc:dd:ee:ff (1.2.3.4): index=1 time=6.626 usec
--- 1.2.3.4 statistics ---
2 packets transmitted, 2 packets received,   0% unanswered (0 extra)
rtt min/avg/max/std-dev = 0.005/0.006/0.007/0.001 ms
root@9f55168b71f2:/#