PES UNIVERSITY

Department of Computer Science & Engineering

Computer Network Security

UE23CS343AB6

Assignment 9 Submission

Name of the Student	Pranav Hemanth
SRN	PES1UG23CS433
Section	G
Department	CSE
Campus	RR

VPN Tunneling Lab

UE23CS343AB6

Explanation of the config files as part of this lab setup:

This lab uses a small multi-container environment (router container + helper code) implemented with Docker images, source code and host-mounted volumes. The top-level tree is:

```
pranavhemanth@Pranavs-MacBook-Pro-M3 Lab8 %ls
docker-compose.yml
docker-compose2.yml
UE22CS343AB6_cef3d63c-5637-11ef-bfdf-0aefbf151e9b_20241025125756.pdf
volumes
pranavhemanth@Pranavs-MacBook-Pro-M3 Lab8 %tree
.
├── docker-compose.yml
├── docker-compose2.yml
├── UE22CS343AB6_cef3d63c-5637-11ef-bfdf-0aefbf151e9b_20241025125756.pdf
└── volumes
    ├── tun_client_select.py
    ├── tun_client.py
    ├── tun_server_select.py
    ├── tun_server.py
    ├── tun_server1.py
    ├── tun.py
    └── tun1.py

2 directories, 10 files
pranavhemanth@Pranavs-MacBook-Pro-M3 Lab8 %
```

Below is the role and purpose of each file and its configurations:

docker-compose.yml

Defines the multi-container VPN lab environment. It specifies services including VPN clients, hosts, and routers with specific network configurations. The file mounts the ./volumes directory into containers for code sharing, grants necessary capabilities (cap_add: ALL) for network operations, and configures multiple Docker networks (net-10.9.0.0 and net-192.168.60.0) to simulate different network segments. This is the main configuration for launching the VPN tunnel testing environment with docker-compose up.

volumes/

- Host-side directory mounted into containers via docker-compose.yml. Contains all the Python scripts for implementing VPN tunnels using TUN interfaces and UDP socket programming. Using volumes enables easy development: edit files on the host system and changes are immediately available in containers.
- volumes/tun_client_select.py
  Advanced VPN client implementation using Python's `select()` for handling multiple I/O sources simultaneously. Creates a TUN interface, configures IP address and routing, then efficiently handles both tunnel interface and UDP socket traffic using non-blocking I/O.
- volumes/tun_client.py
  Basic VPN client that creates a TUN interface, configures networking (192.168.53.99/24), sets up routing for the 192.168.60.0/24 network, and forwards packets between the TUN interface and a UDP tunnel to the VPN server.
- volumes/tun_server_select.py
  Advanced VPN server using `select()` for handling multiple connections. Creates a TUN interface (192.168.53.1/24), binds to UDP port 9090, and efficiently manages traffic between multiple clients and the TUN interface using non-blocking I/O.
- volumes/tun_server.py
  Basic VPN server that listens on UDP port 9090, receives encapsulated packets from VPN clients, and prints packet information for monitoring tunnel traffic.
- volumes/tun_server1.py
  Enhanced VPN server that combines TUN interface creation with UDP tunneling. Creates a TUN interface, binds to UDP port 9090, and forwards received packets directly to the TUN interface.
- volumes/tun.py
  Simple TUN interface creation script for testing and demonstration. Creates a TUN device and keeps it active with periodic sleep intervals.
- volumes/tun1.py
  Advanced TUN interface script with packet manipulation capabilities. Creates a TUN interface with IP configuration and implements ICMP packet spoofing - intercepts ping requests and generates spoofed replies.

Change shell name:

PS1="host-192.168.60.5:PES1UG23CS433:PranavHemanth:\w\n\$>"

PS1="host-192.168.60.6:PES1UG23CS433:PranavHemanth:\w\n\$>"

PS1="client-10.9.0.5:PES1UG23CS433:PranavHemanth:\w\n\$>"

PS1="server-router:PES1UG23CS433:PranavHemanth:\w\n\$>"

Task 1: Network Setup

We are only using docker-compose.yml, please delete docker-compose2.yml and open all the docker container terminals using the given Labsetup folder.

```
seed@seedvm2004:~$ cd Desktop/
seed@seedvm2004:~/Desktop$ cd Lab8
seed@seedvm2004:~/Desktop/Lab8$ ls
docker-compose2.yml
docker-compose.yml
UE22CS343AB6_cef3d63c-5637-11ef-bfdf-0aefbf151e9b_20241025125756.pdf
volumes
seed@seedvm2004:~/Desktop/Lab8$ rm docker-compose2.yml
seed@seedvm2004:~/Desktop/Lab8$ ls
docker-compose.yml                                                volumes
UE22CS343AB6_cef3d63c-5637-11ef-bfdf-0aefbf151e9b_20241025125756.pdf
seed@seedvm2004:~/Desktop/Lab8$
```

```
 ✔ Host1 Pulled                                                    35.1s
   ✔ 82d728d38b98 Pull complete                                    15.1s
   ✔ c06a3dc7ba9f Pull complete                                    28.2s
   ✔ 1622d55204e9 Pull complete                                    28.3s
   ✔ 04e16e3db838 Pull complete                                    28.9s
   ✔ 408a37ef18eb Pull complete                                    28.9s
   ✔ 768c10fd7ef1 Pull complete                                    28.9s
   ✔ c79874963658 Pull complete                                    28.9s
 ✔ Host2 Pulled                                                    35.1s
 ✔ Router Pulled                                                   35.1s
[+] Running 6/6
 ✔ Network net-10.9.0.0          Created                            0.1s
 ✔ Network net-192.168.60.0      Created                            0.0s
 ✔ Container client-10.9.0.5     Created                            0.4s
 ✔ Container host-192.168.60.5   Created                            0.4s
 ✔ Container server-router       Created                            0.4s
 ✔ Container host-192.168.60.6   Created                            0.4s
Attaching to client-10.9.0.5, host-192.168.60.5, host-192.168.60.6, server-route
r
host-192.168.60.5  |  * Starting internet superserver inetd                [ OK ]
host-192.168.60.6  |  * Starting internet superserver inetd                [ OK ]

w Enable Watch
```

Certain requirements for the lab pre-configured are:

- Shared Folder
- Packet Sniffing:
  - To sniff the packets going through a particular interface, we just need to find out the interface name, and then do the following (assume that the interface name is eth0):
  - Command: # tcpdump -i eth0 -n
- Testing

Testing:

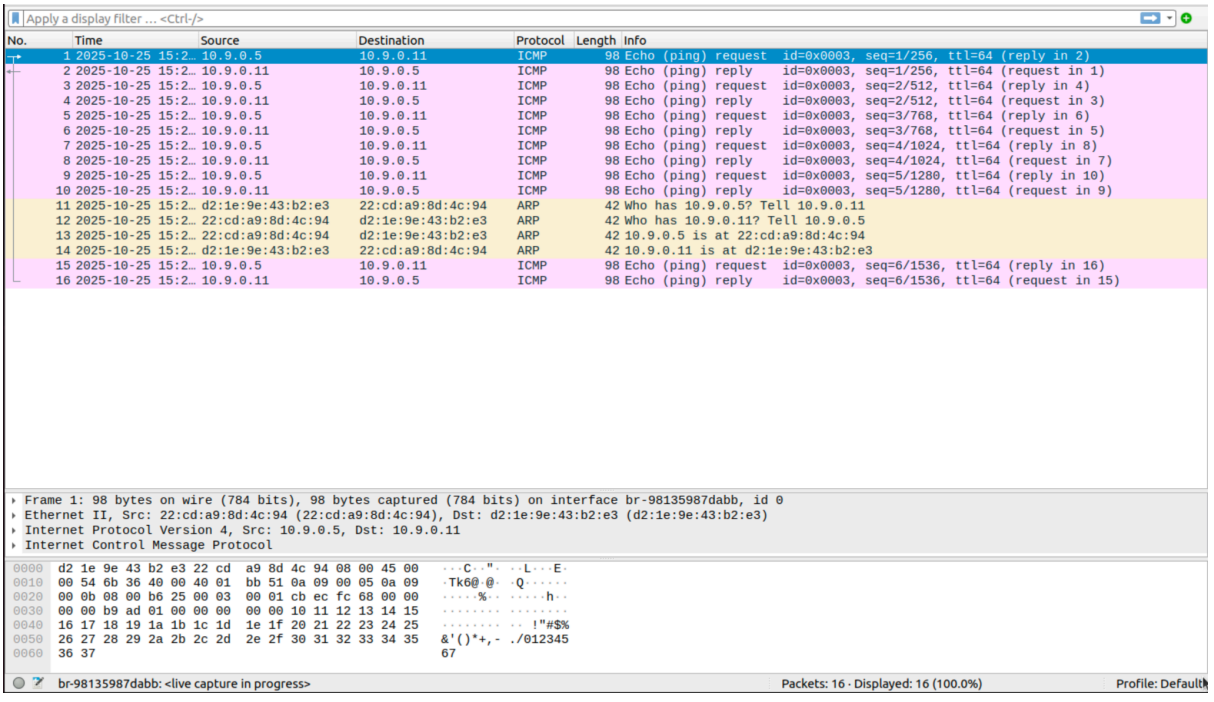
Please conduct the following testings to ensure that the lab environment is set up correctly:

Step1: Host U - 10.9.0.5 can communicate with VPN Server (server-router)

1) Command: ping server-router (On Client-10.9.0.5)

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$>ping server-router
PING server-router (10.9.0.11) 56(84) bytes of data.
64 bytes from server-router.net-10.9.0.0 (10.9.0.11): icmp_seq=1 ttl=64 time=0.2
73 ms
64 bytes from server-router.net-10.9.0.0 (10.9.0.11): icmp_seq=2 ttl=64 time=0.1
31 ms
64 bytes from server-router.net-10.9.0.0 (10.9.0.11): icmp_seq=3 ttl=64 time=0.1
06 ms
64 bytes from server-router.net-10.9.0.0 (10.9.0.11): icmp_seq=4 ttl=64 time=0.1
06 ms
64 bytes from server-router.net-10.9.0.0 (10.9.0.11): icmp_seq=5 ttl=64 time=0.1
37 ms
64 bytes from server-router.net-10.9.0.0 (10.9.0.11): icmp_seq=6 ttl=64 time=0.0
99 ms
^C
--- server-router ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5082ms
rtt min/avg/max/mdev = 0.099/0.142/0.273/0.060 ms
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$>
```

Wireshark:

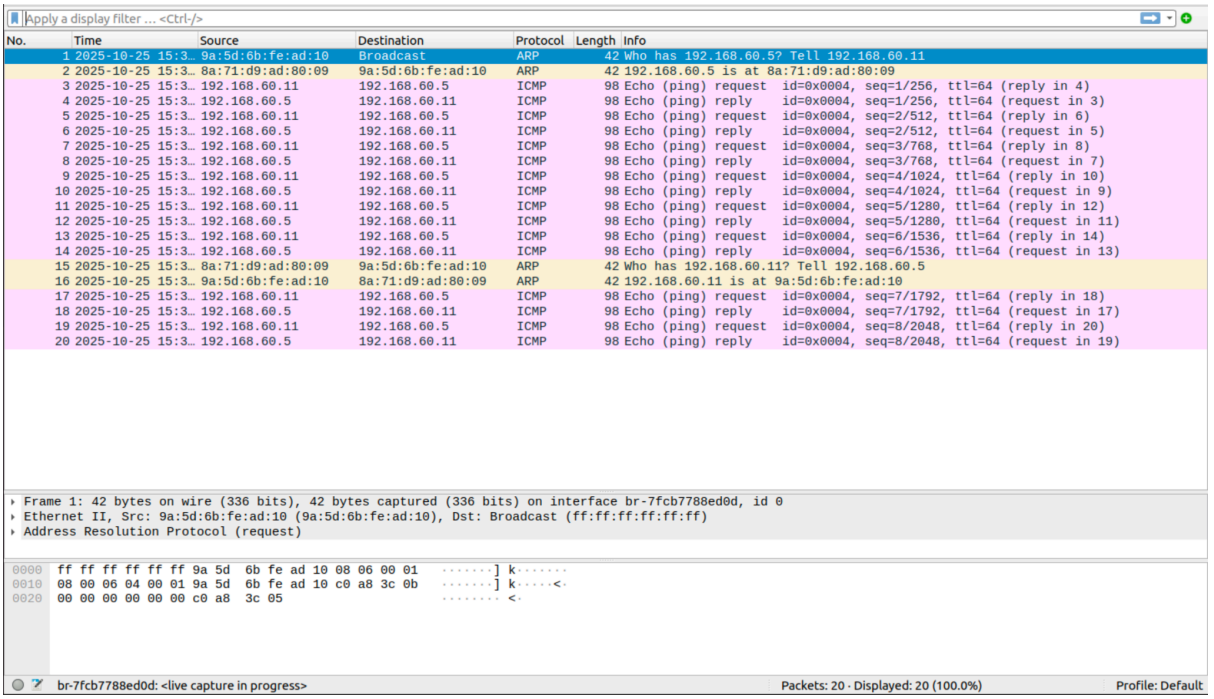


Step2: VPN Server (server-router) can communicate with Host V (host-192.168.60.5)

2) Command: ping 192.168.60.5 (On server-router)

```
server-router:PES1UG23CS433:PranavHemanth:/
$>ping 192.168.60.5
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=64 time=1.43 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=64 time=0.160 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=64 time=0.135 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=64 time=0.170 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=64 time=0.140 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=64 time=0.218 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=64 time=0.102 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=64 time=0.140 ms
^C
--- 192.168.60.5 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7124ms
rtt min/avg/max/mdev = 0.102/0.311/1.430/0.423 ms
server-router:PES1UG23CS433:PranavHemanth:/
$>
```

Wireshark:

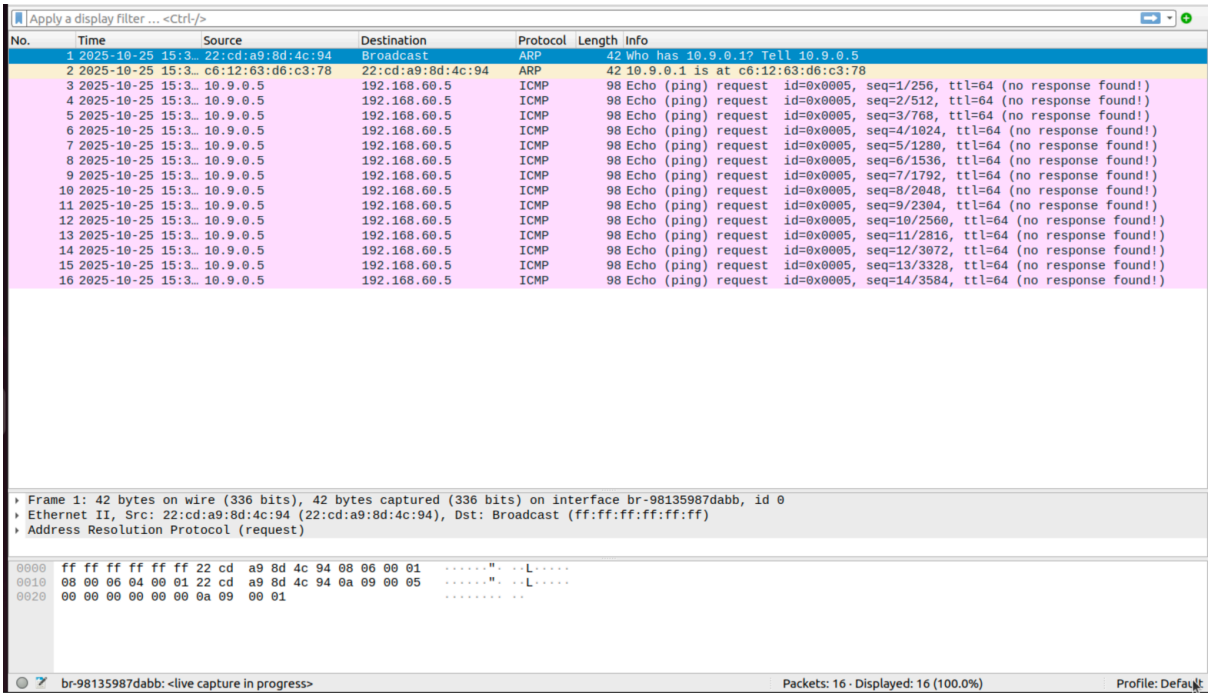


Step3: Host U (Client - 10.9.0.5) should not be able to communicate with Host V (host 192.168.60.5)

3) Command: ping 192.168.60.5 (On Client-10.9.0.5)

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$>ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
14 packets transmitted, 0 received, 100% packet loss, time 13314ms

client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$>
```

Wireshark:



Step4: Run tcpdump on the router, and sniff the traffic on each of the networks. Show that you can capture packets.

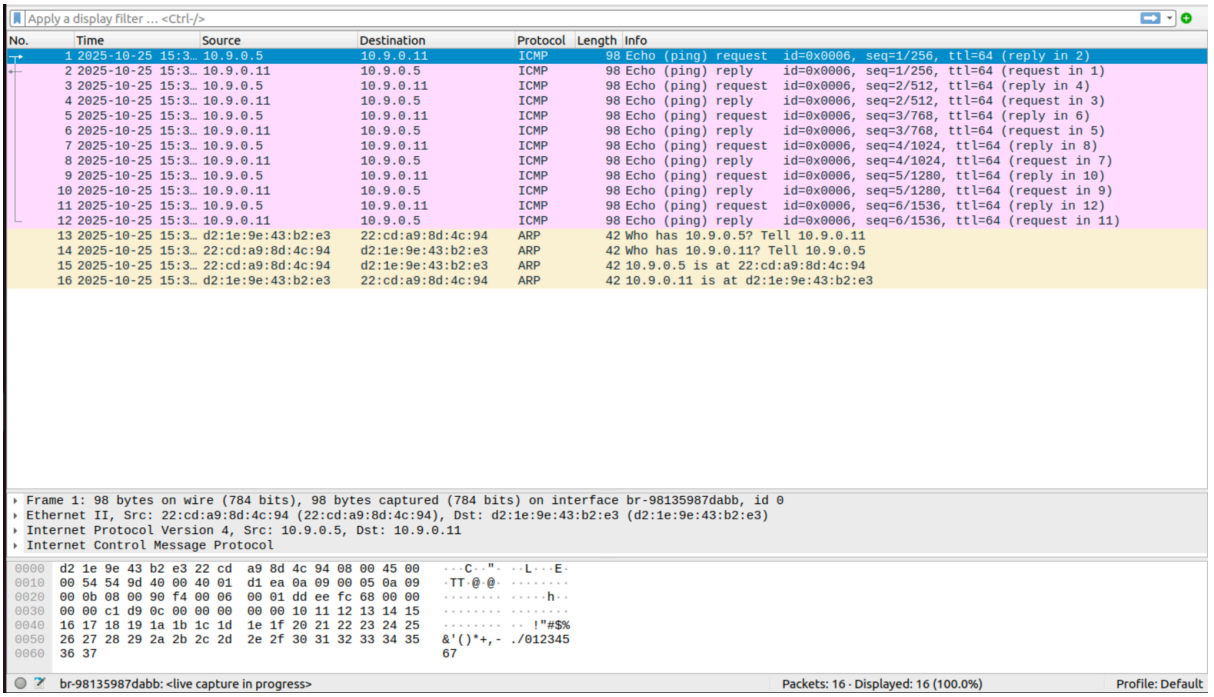
4) Command: tcpdump -i eth0 -n (On server-router)

```
server-router:PES1UG23CS433:PranavHemanth:/
$>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
15:38:05.842350 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 6, seq 1, length 64
15:38:05.842532 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 6, seq 1, length 64
15:38:06.843717 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 6, seq 2, length 64
15:38:06.843885 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 6, seq 2, length 64
15:38:07.871532 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 6, seq 3, length 64
15:38:07.871655 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 6, seq 3, length 64
15:38:08.895697 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 6, seq 4, length 64
15:38:08.895823 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 6, seq 4, length 64
15:38:09.919241 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 6, seq 5, length 64
15:38:09.919333 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 6, seq 5, length 64
15:38:10.943752 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 6, seq 6, length 64
15:38:10.943868 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 6, seq 6, length 64
15:38:11.072144 ARP, Request who-has 10.9.0.5 tell 10.9.0.11, length 28
15:38:11.072422 ARP, Request who-has 10.9.0.11 tell 10.9.0.5, length 28
15:38:11.072455 ARP, Reply 10.9.0.11 is-at d2:1e:9e:43:b2:e3, length 28
15:38:11.072470 ARP, Reply 10.9.0.5 is-at 22:cd:a9:8d:4c:94, length 28
^C
16 packets captured
16 packets received by filter
0 packets dropped by kernel
server-router:PES1UG23CS433:PranavHemanth:/
$>
```

5) Command: ping server-router (On Client-10.9.0.5)

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$>ping server-router
PING server-router (10.9.0.11) 56(84) bytes of data.
64 bytes from server-router.net-10.9.0.0 (10.9.0.11): icmp_seq=1 ttl=64 time=0.5
10 ms
64 bytes from server-router.net-10.9.0.0 (10.9.0.11): icmp_seq=2 ttl=64 time=0.3
29 ms
64 bytes from server-router.net-10.9.0.0 (10.9.0.11): icmp_seq=3 ttl=64 time=0.2
61 ms
64 bytes from server-router.net-10.9.0.0 (10.9.0.11): icmp_seq=4 ttl=64 time=0.2
36 ms
64 bytes from server-router.net-10.9.0.0 (10.9.0.11): icmp_seq=5 ttl=64 time=0.2
22 ms
64 bytes from server-router.net-10.9.0.0 (10.9.0.11): icmp_seq=6 ttl=64 time=0.2
44 ms
^C
--- server-router ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5101ms
rtt min/avg/max/mdev = 0.222/0.300/0.510/0.099 ms
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$>
```

Wireshark:



Explanation:
This output from the client machine performs two key tests. It first proves it can reach the server-router (0% packet loss) but then correctly fails to reach Host V (100% packet loss), confirming that the networks are isolated.

On the router side, the terminal output shows a successful connection to host 192.168.60.5, demonstrating correct inter-network connectivity.

Additionally, tcpdump running on the router captures ICMP request and reply packets from the client's ping test, verifying that packet sniffing is functioning correctly.

Task 2: Create and Configure TUN Interface

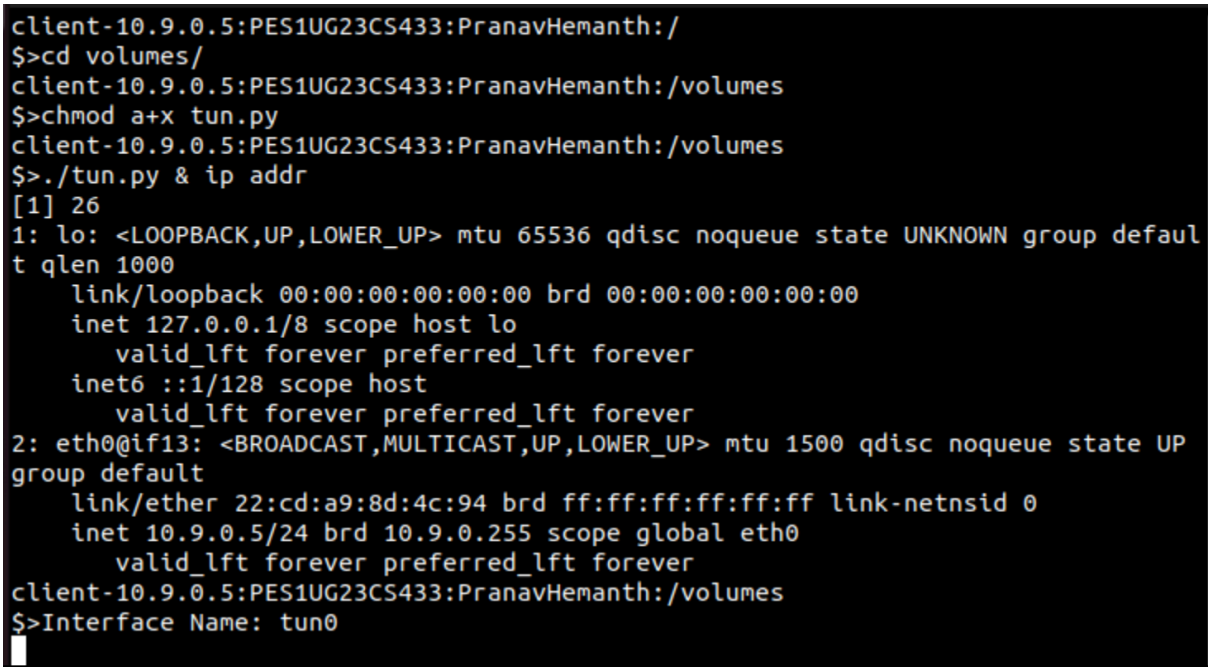
The VPN tunnel that we are going to build is based on the TUN/TAP technologies. TUN and TAP are virtual network kernel drivers; they implement network devices that are supported entirely in software. TUN (as in network TUNnel) simulates a network layer device and it operates with layer-3 packets such as IP packets. With TUN/TAP, we can create virtual network interfaces.

The objective of this task is to get familiar with the TUN technology. We will conduct several experiments to learn the technical details of the TUN interface. We will use the following Python program as the basis for the experiments, and we will modify this base code throughout this lab. The code is already included in the volumes folder in the zip file.

Task 2.A: Name of the Interface

Step1: We will run the tun.py program on Host U (Client-10.9.0.5). Make the above tun.py program executable, and run it using the root privilege.

6) Command: (On Client - 10.9.0.5)
   chmod a+x tun.py
   ./tun.py &
   ip addr



Explanation:
Here we can see that we have successfully given permissions to, and are able to run tun.py on the client

Step2: Kill the earlier Tunnel Process - (On Client - 10.9.0.5)

7) Command: kill %1
   In case you get an error, run
   # jobs
   Note down the number for ./tun.py, then run # kill %[the number]

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>kill %1
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>jobs
[1]+  Terminated              ./tun.py
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>
```

Step3: Instead of using tun as the prefix of the interface name, use the last 5 digits of your SRN as the prefix

8) Edit: In tun.py replace “tun” with the last 5 characters of your SRN in line 16.
   Line 16 - ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)

```
seed@seedvm2004:~$ cd Desktop/
seed@seedvm2004:~/Desktop$ cd Lab8
seed@seedvm2004:~/Desktop/Lab8$ cd volumes/
seed@seedvm2004:~/Desktop/Lab8/volumes$ ls
tun1.py        tun_client_select.py  tun_server1.py  tun_server_select.py
tun_client.py  tun.py                tun_server.py
seed@seedvm2004:~/Desktop/Lab8/volumes$ nano tun.py
seed@seedvm2004:~/Desktop/Lab8/volumes$
```

```
 GNU nano 6.2                                                    tun.py
#!/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'CS433%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))

while True:
   time.sleep(10)
```

Step1: Now run the following commands again and see the new tunnel interface, it should be “SRN0” (Client-10.9.0.5). Make the above tun.py program executable, and run it using the root privilege.

9) Command: (On Client - 10.9.0.5)
   chmod a+x tun.py
   ./tun.py &
   ip addr
10) Command:
   kill %1
   jobs

client-10.9.0.5:

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>chmod a+x tun.py
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>./tun.py & ip addr
[1] 31
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
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0@if13: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 22:cd:a9:8d:4c:94 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
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>Interface Name: CS4330
^C
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>kill %1
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>jobs
[1]+  Terminated              ./tun.py
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>
```

Explanation:
This output shows the unmodified tun.py script being executed successfully. It creates a new virtual interface tun0, as verified by ip addr. This interface acts as a software-based gateway to the kernel’s network stack. Any packet the kernel sends to tun0 will be redirected to the running Python script.

After modifying the interface prefix to CS433, rerunning the script creates a custom virtual interface CS4330, showing that the TUN interface name is user-defined and configurable.

Task 2.B: Set up the TUN Interface

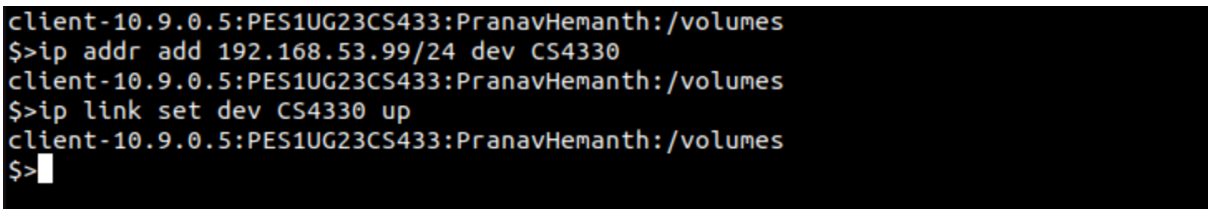
At this point, the TUN interface is not usable, because it has not been configured yet. There are two things that we need to do before the interface can be used. First, we need to assign an IP address to it. Second, we need to bring up the interface, because the interface is still in the down state. We can use the following two commands for the configuration:

Step1: Configure the TUN interface

11) Command: (On Client - 10.9.0.5)

```
# ip addr add 192.168.53.99/24 dev CS4330
# ip link set dev CS4330 up
```

client-10.9.0.5:



Explanation:

In this task, we configured iptables on the seed-router to protect the internal network When the first two commands are executed, they successfully create and configure the TUN interface. Running ip addr show CS4330 afterwards confirms that the interface has been assigned the IP address 192.168.53.99/24and its state is UP. This verifies that the virtual interface has been properly initialized, assigned a valid IP configuration, and is now active and ready to handle network traffic.

Task 2.C: Read from the TUN Interface

In this task, we will read from the TUN interface. Whatever coming out from the TUN interface is an IP packet. We can cast the data received from the interface into a Scapy IP object, so we can print out each field of the IP packet. Please use the following while loop to replace the one in tun.py:

Step1: Replace the following in tun.py -

```
while True:
        time.sleep(10)
```

With -

```
while True:
# Get a packet from the tun interface
        packet = os.read(tun, 2048)
        if packet:
                ip = IP(packet)
                print(ip.summary())
```

tun.py:

```
  GNU nano 6.2                                  tun.py

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'CS433%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))

#while True:
#   time.sleep(10)

while True:
# Get a packet from the tun interface
        packet = os.read(tun, 2048)
        if packet:
                ip = IP(packet)
                print(ip.summary())

                                [ Wrote 32 lines ]
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^/ Go To Line
```

```
seed@seedvm2004:~$ cd Desktop/
seed@seedvm2004:~/Desktop$ cd Lab8
seed@seedvm2004:~/Desktop/Lab8$ ls
docker-compose.yml                                                  volumes
UE22CS343AB6_cef3d63c-5637-11ef-bfdf-0aefbf151e9b_20241025125756.pdf
seed@seedvm2004:~/Desktop/Lab8$ cd volumes/
seed@seedvm2004:~/Desktop/Lab8/volumes$ ls
tun1.py        tun_client_select.py  tun_server1.py  tun_server_select.py
tun_client.py  tun.py                tun_server.py
seed@seedvm2004:~/Desktop/Lab8/volumes$ nano tun.py
seed@seedvm2004:~/Desktop/Lab8/volumes$
```

Step2: Kill the earlier Tunnel Process - (On Client - 10.9.0.5)

12)Command: (On Client - 10.9.0.5)

kill %1

jobs

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>kill %1
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>jobs
[1]+  Terminated              ./tun.py
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>
```

Step3: Setup the TUN interface

13)Command: (On Client - 10.9.0.5)

ip addr add 192.168.53.99/24 dev CS4330

ip link set dev CS4330 up

client-10.9.0.5:

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>ip addr add 192.168.53.99/24 dev CS4330
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>ip link set dev CS4330 up
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>
```

Step4: Run the revised tun.py program -

14)Command: ./tun.py & ip addr (On Client - 10.9.0.5)

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>ip addr add 192.168.53.99/24 dev CS4330
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>ip link set dev CS4330 up
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>./tun.py & ip addr
[3] 55
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
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0@if6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:c9:4d:5c:4a:e2 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
4: CS4330: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group default qlen 500
    link/none
    inet 192.168.53.99/24 scope global CS4330
       valid_lft forever preferred_lft forever
    inet6 fe80::2540:17aa:c8df:6001/64 scope link stable-privacy
       valid_lft forever preferred_lft forever
6: CS4331: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group default qlen 500
    link/none
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>Interface Name: CS4332
$>0.0.0.0 > 37.64.23.170 128 frag:6911 / Padding
$>
```

Step5: On Host U, ping a host in the 192.168.53.0/24 network.

15)Command: ping 192.168.53.5 (On Client - 10.9.0.5)

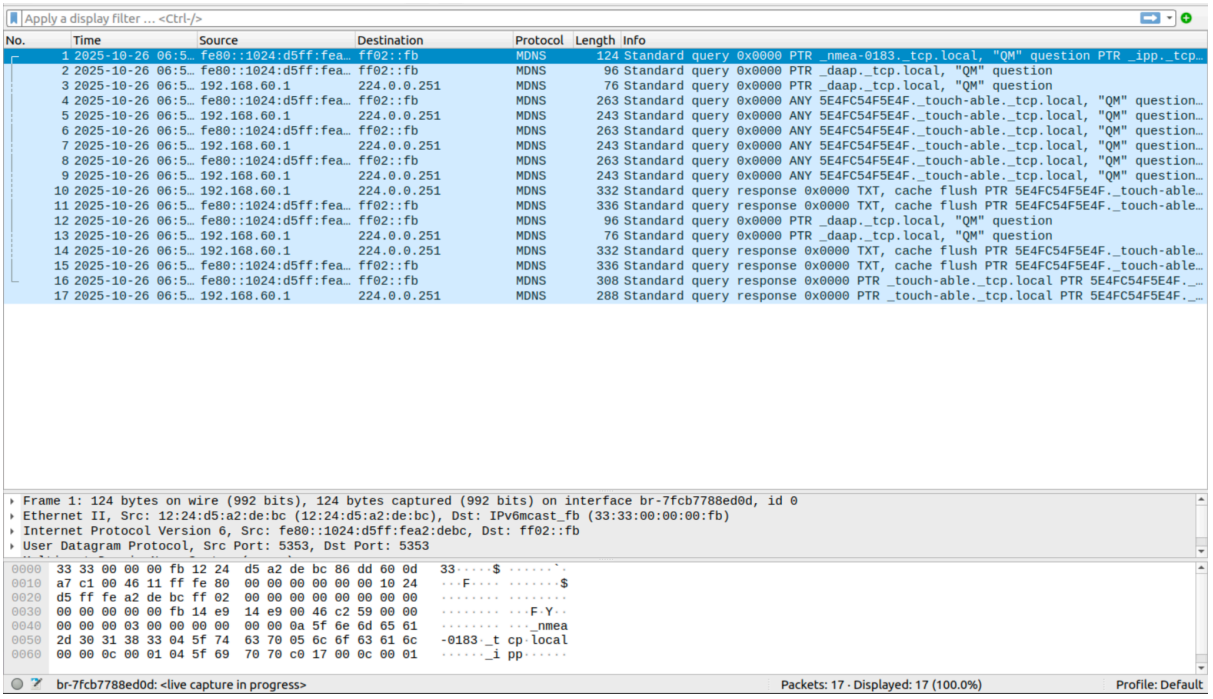
client-10.9.0.5 (pinging 192.168.53.5):

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$ping 192.168.53.5
PING 192.168.53.5 (192.168.53.5) 56(84) bytes of data.
^C
--- 192.168.53.5 ping statistics ---
8 packets transmitted, 0 received, 100% packet loss, time 7166ms

client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$
```

client-10.9.0.5 (running the tun.py script):

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>Interface Name: CS4332
$>0.0.0.0 > 37.64.23.170 128 frag:6911 / Padding
$>IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
```

Wireshark:



(Capture not relevant to experiment. only mdns queries observed)

Questions:

What is printed out by the tun.py program? What has happened? Why?

Answer: The tun.py program prints IP/ICMP packet summaries such as IP 192.168.53.99 > 192.168.53.5: ICMP echo request. This happens because when the client pings 192.168.53.5, the outgoing ICMP packets are routed through the virtual interface CS4330. The kernel sends these packets to the TUN device, and the Python script captures them and prints their summaries. No replies are seen because the script only reads packets it doesn't generate or forward responses yet.

Step6: On Host U, ping a host in the internal 192.168.60.0/24 network.

16)Command: ping 192.168.60.5 (On Client - 10.9.0.5)

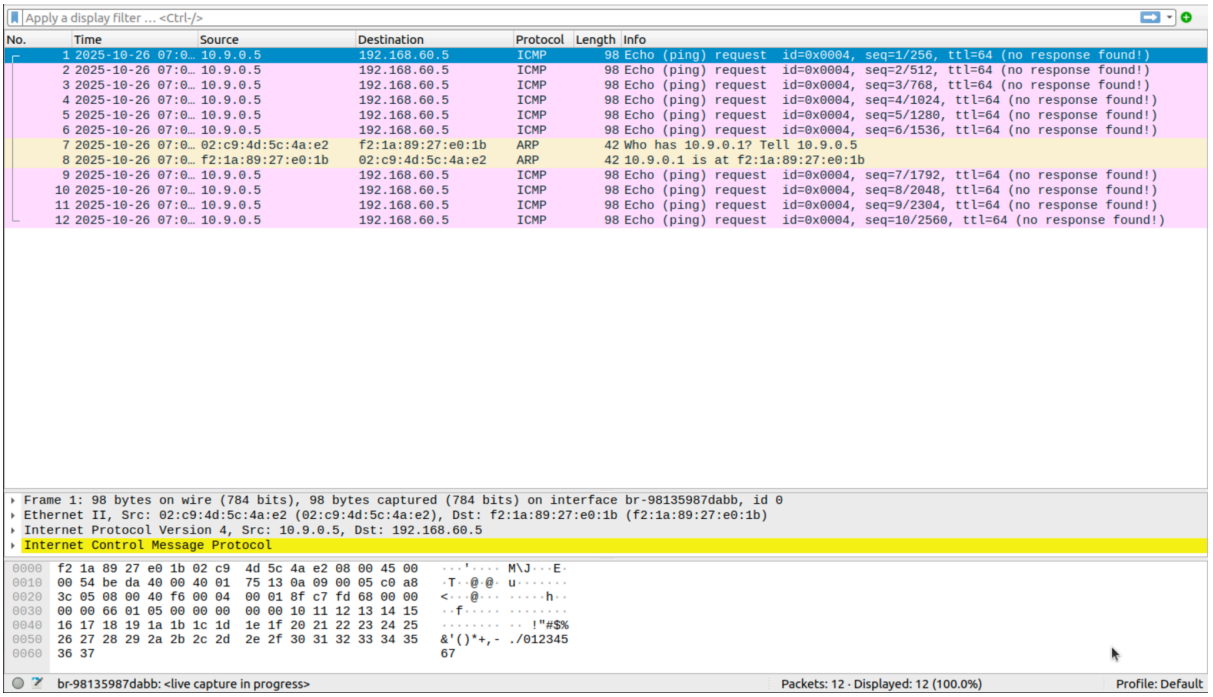
client-10.9.0.5 (pinging 192.168.60.5):

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
10 packets transmitted, 0 received, 100% packet loss, time 9205ms

client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$
```

client-10.9.0.5 (running the tun.py script):

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>Interface Name: CS4333
```

Wireshark:



Questions:

What is printed out by the tun.py program? What has happened? Why?

Answer: Nothing is printed this time. The packets never reach the TUN interface because the destination (192.168.60.5) is outside the configured 192.168.53.0/24 subnet. Hence, the kernel routes them via the default gateway instead of the virtual interface, meaning the script doesn't intercept or print anything.

Explanation: This experiment proves that the TUN interface only receives traffic In this task, we will write to the TUN interface. Since this is a virtual network interface, whatever is written to the interface by the application will appear in the kernel as an IP packet. We will modify the tun.py program, so after getting a packet from the TUN interface, we construct a new packet based on the received packet. We then write the new packet to the TUN interface.specifically routed to its configured subnet.

It behaves just like a real network interface; packets must be addressed to its subnet for the kernel to deliver them through it.

Step7: Kill the Tunnel Process - (On Client - 10.9.0.5)

17) Command: (On Client - 10.9.0.5)

kill %1

jobs

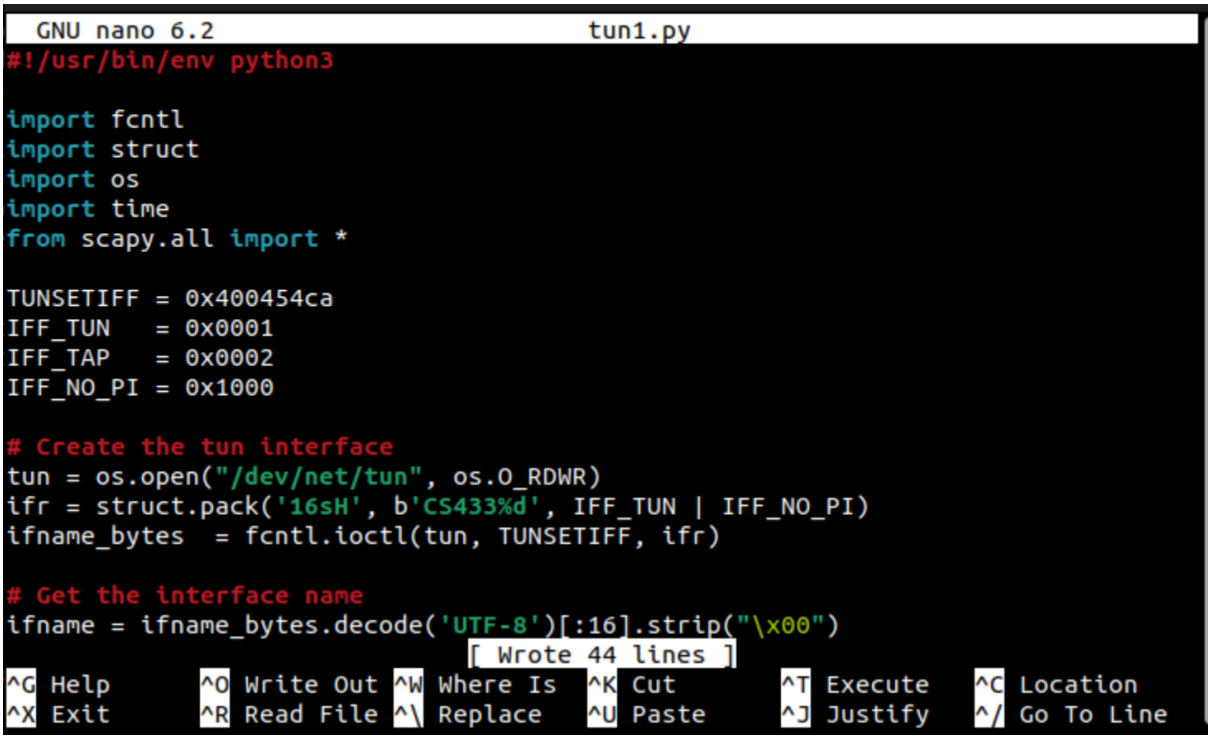
```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>kill %1
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>jobs
[1]   Terminated              ./tun.py
[2]   Running                 ./tun.py &
[3]-  Running                 ./tun.py &
[4]+  Running                 ./tun.py &
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>kill %2
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>kill %3
[2]   Terminated              ./tun.py
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>kill %4
[3]-  Terminated              ./tun.py
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>jobs
[4]+  Terminated              ./tun.py
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>
```

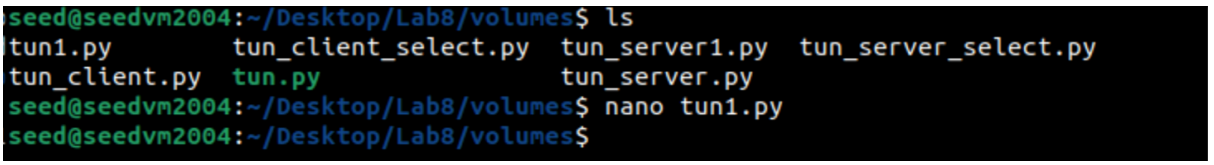
Task 2.D: Write to the TUN Interface

Your job in this task is to change the tun1.py program, so instead of using tun as the prefix of the interface name, use the last 5 digits of your SRN as the prefix. Please show your results with appropriate screenshots.

Step1: Instead of using tun as the prefix of the interface name, use the last 5 digits of your SRN as the prefix

18)Edit: In tun1.py replace “tun” with the last 5 characters of your SRN in line 16.
Line 16 - ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)





Step2: Now run the following commands and see the new tunnel interface, it should be “SRN0” (Client-10.9.0.5). Make the above tun1.py program executable, and run it using the root privilege.

19)Command: (On Client - 10.9.0.5)

```
chmod a+x tun1.py
./tun1.py &
ip addr
```

client-10.9.0.5:

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>chmod a+x tun1.py
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>./tun1.py & ip addr
[1] 84
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t 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
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0@if6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP g
roup default
    link/ether 02:c9:4d:5c:4a:e2 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
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>Interface Name: CS4330
CS4330: 0.0.0.0 > 178.119.113.2 128 frag:6911 / Padding
CS4330: 0.0.0.0 > 178.119.113.2 128 frag:6911 / Padding
CS4330: 0.0.0.0 > 178.119.113.2 128 frag:6911 / Padding
```

Step3: On Host U, ping a host in the 192.168.53.0/24 network.

20)Command: ping 192.168.53.5 (On Client - 10.9.0.5)

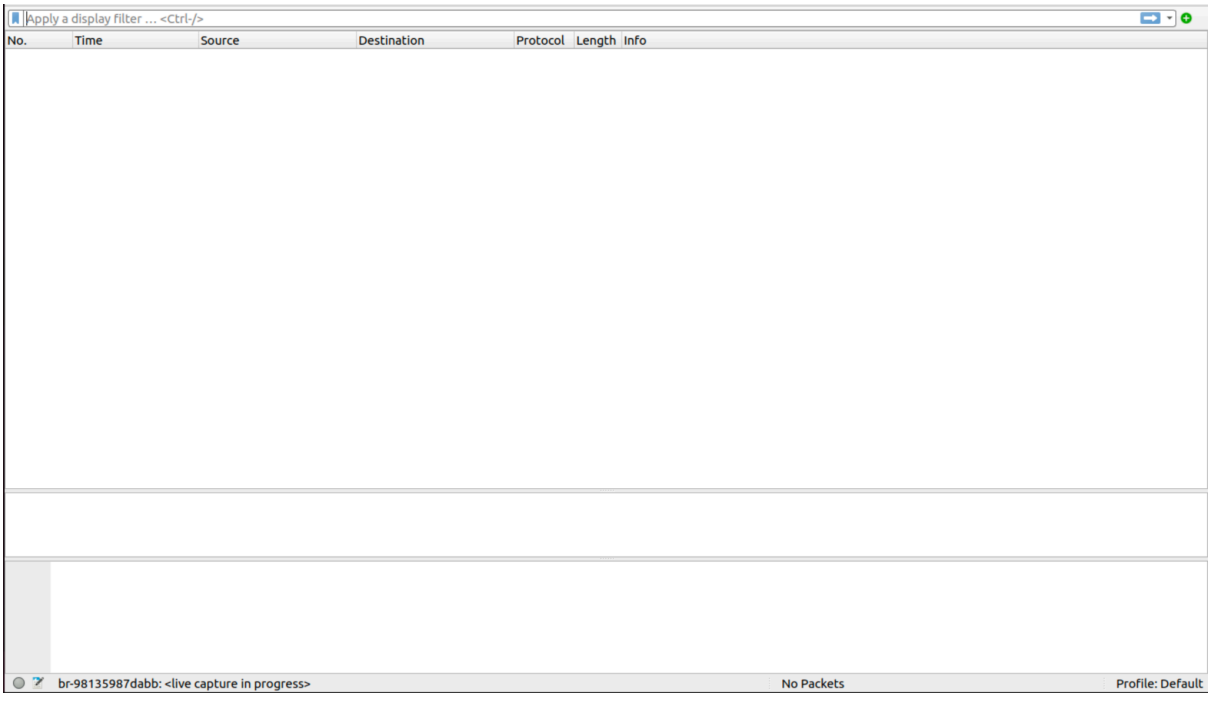
client-10.9.0.5 (pinging 192.168.53.5):

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$ping 192.168.53.5
PING 192.168.53.5 (192.168.53.5) 56(84) bytes of data.
64 bytes from 192.168.53.5: icmp_seq=1 ttl=99 time=42.4 ms
64 bytes from 192.168.53.5: icmp_seq=2 ttl=99 time=4.13 ms
64 bytes from 192.168.53.5: icmp_seq=3 ttl=99 time=4.60 ms
64 bytes from 192.168.53.5: icmp_seq=4 ttl=99 time=4.23 ms
64 bytes from 192.168.53.5: icmp_seq=5 ttl=99 time=3.79 ms
64 bytes from 192.168.53.5: icmp_seq=6 ttl=99 time=3.72 ms
64 bytes from 192.168.53.5: icmp_seq=7 ttl=99 time=14.3 ms
^C
--- 192.168.53.5 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6011ms
rtt min/avg/max/mdev = 3.717/11.020/42.380/13.280 ms
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$
```

client-10.9.0.5 (running the tun1.py script):

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>./tun1.py & ip addr
[2] 93
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
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0@if6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:c9:4d:5c:4a:e2 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
9: CS4330: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group default qlen 500
    link/none
    inet 192.168.53.99/24 scope global CS4330
       valid_lft forever preferred_lft forever
    inet6 fe80::b277:7102:2de0:2728/64 scope link stable-privacy
       valid_lft forever preferred_lft forever
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>Interface Name: CS4331
CS4331: 0.0.0.0 > 241.90.24.236 128 frag:6911 / Padding
CS4331: 0.0.0.0 > 241.90.24.236 128 frag:6911 / Padding
CS4330: IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
CS4330: IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
CS4330: IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
CS4330: IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
CS4330: IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
CS4331: 0.0.0.0 > 241.90.24.236 128 frag:6911 / Padding
CS4330: IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
CS4330: IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
$>
```

Wireshark:



(Capture not relevant to experiment observed)

Explanation:
The modified tun1.py script now implements bidirectional communication. It captures ICMP echo requests from the client (192.168.53.99 → 192.168.53.5) and then crafts echo replies that are written back to the TUN interface.

The successful ping replies confirm that the interface is functioning as a simulated host reading incoming packets, generating valid responses, and integrating seamlessly with the kernel's networking stack.

Step4: Kill the Tunnel Process - (On Client - 10.9.0.5)

21) Command: (On Client - 10.9.0.5)

kill %1

jobs

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>jobs
[1]-  Running                 ./tun1.py &
[2]+  Running                 ./tun1.py &
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>kill %1
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>kill %2
[1]-  Terminated              ./tun1.py
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>jobs
[2]+  Terminated              ./tun1.py
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>
```

Explanation:

Once bidirectional ICMP handling works in Task 2.D, the next step is to extend communication across machines.

This is achieved in Task 3 by encapsulating IP packets inside UDP packets and transmitting them to a remote VPN server.

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

In this task, we will put the IP packet received from the TUN interface into the UDP payload field of a new IP packet, and send it to another computer. Namely, we place the original packet inside a new packet. This is called IP tunneling. The tunnel implementation is just standard client/server programming. It can be built on top of TCP or UDP. In this task, we will use UDP. Namely, we put an IP packet inside the payload field of a UDP packet.

The server program tun_server.py - We will run the tun_server.py program on VPN Server. This program is just a standard UDP server program. It listens to port 9090 and prints out whatever is received. The program assumes that the data in the UDP payload field is an IP packet, so it casts the payload to a Scapy IP object, and prints out the source and destination IP address of the enclosed IP packet.

Step1: On server-router run (change directory to ./volumes)

22) Command:

chmod a+x tun_server.py

./tun_server.py

```
server-router:PES1UG23CS433:PranavHemanth:/
$>cd volumes/
server-router:PES1UG23CS433:PranavHemanth:/volumes
$>chmod a+x tun_server.py
server-router:PES1UG23CS433:PranavHemanth:/volumes
$>./tun_server.py
```

Implement the client program tun_client.py - First, we need to modify the TUN program tun.py. Let's rename it, and call it tun_client.py. Sending data to another computer using UDP can be done using the standard socket programming. Replace the while loop in the program with the following: The SERVER IP and SERVER PORT should be replaced with the actual IP address and port number of the server program running on VPN Server

Note - In tun_client.py replace "tun" with the last 5 characters of your SRN in line 16.

Step2: Instead of using tun as the prefix of the interface name, use the last 5 digits of your SRN as the prefix

23) Edit: In tun1.py replace "tun" with the last 5 characters of your SRN in line 16.

Line 16 - ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)

```
  GNU nano 6.2                          tun_client.py
#!/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'CS433%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")
                            [ Wrote 42 lines ]
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^/ Go To Line
```

```
seed@seedvm2004:~/Desktop/Lab8/volumes$ ls
tun1.py         tun_client_select.py  tun_server1.py  tun_server_select.py
tun_client.py   tun.py                tun_server.py
seed@seedvm2004:~/Desktop/Lab8/volumes$ nano tun_client.py
seed@seedvm2004:~/Desktop/Lab8/volumes$
```

Step3: Run the following commands and see the new tunnel interface, it should be “SRN0” (Client-10.9.0.5). Make the above tun_client.py program executable, and run it using the root privilege.

24)Command: (On Client - 10.9.0.5)

chmod a+x tun_client.py

./tun_client.py &

ip addr

client-10.9.0.5:

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>chmod a+x tun_client.py
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>./tun_client.py & ip addr
[1] 105
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t 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
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0@if6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP g
roup default
    link/ether 02:c9:4d:5c:4a:e2 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
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>Interface Name: CS4330
0.0.0.0 > 12.44.88.212 128 frag:6911 / Padding
0.0.0.0 > 12.44.88.212 128 frag:6911 / Padding
0.0.0.0 > 12.44.88.212 128 frag:6911 / Padding
```

server-router:

```
server-router:PES1UG23CS433:PranavHemanth:/
$>cd volumes/
server-router:PES1UG23CS433:PranavHemanth:/volumes
$>chmod a+x tun_server.py
server-router:PES1UG23CS433:PranavHemanth:/volumes
$>./tun_server.py
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 0.0.0.0 --> 12.44.88.212
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 0.0.0.0 --> 12.44.88.212
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 0.0.0.0 --> 12.44.88.212
```

To test whether the tunnel works or not, ping any IP address belonging to the 192.168.53.0/24 network.

Step4: On Host U, ping a host in the 192.168.53.0/24 network.

25)Command: ping 192.168.53.5 (On Client - 10.9.0.5)

client-10.9.0.5 (pinging 192.168.53.5):

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$ping 192.168.53.5
PING 192.168.53.5 (192.168.53.5) 56(84) bytes of data.
^C
--- 192.168.53.5 ping statistics ---
10 packets transmitted, 0 received, 100% packet loss, time 9227ms

client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$
```

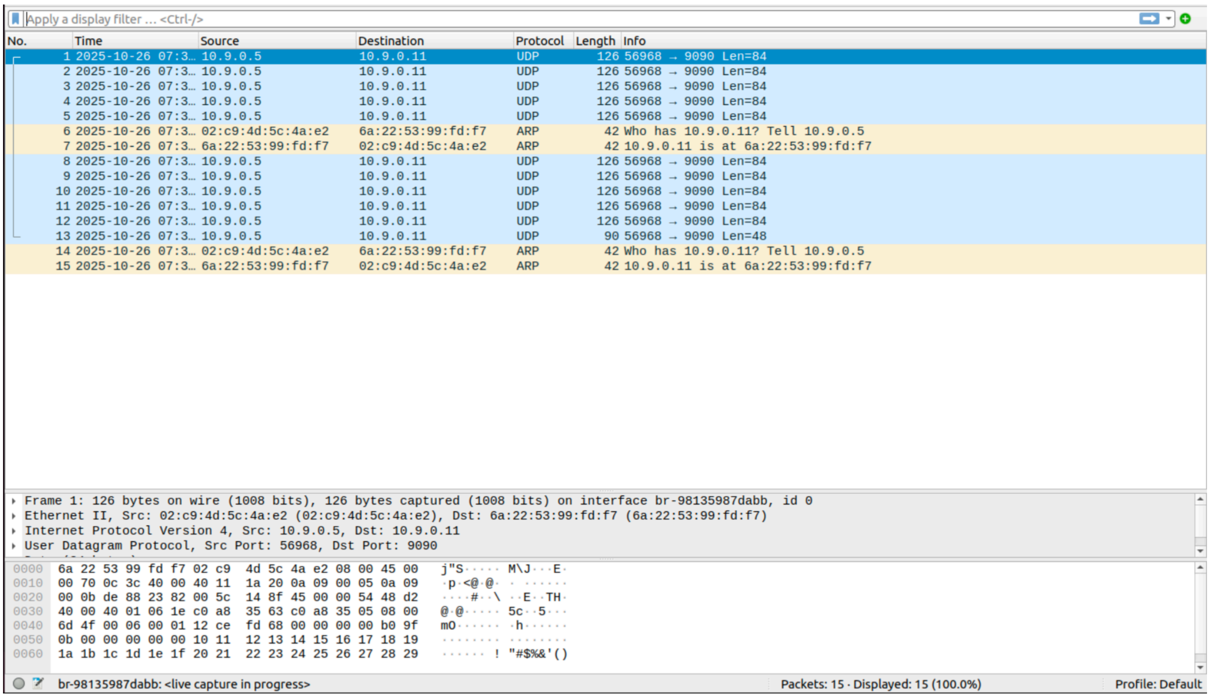
client-10.9.0.5 (running the tun_client.py script):

```
$>Interface Name: CS4330
0.0.0.0 > 12.44.88.212 128 frag:6911 / Padding
0.0.0.0 > 12.44.88.212 128 frag:6911 / Padding
0.0.0.0 > 12.44.88.212 128 frag:6911 / Padding
0.0.0.0 > 12.44.88.212 128 frag:6911 / Padding
0.0.0.0 > 12.44.88.212 128 frag:6911 / Padding
0.0.0.0 > 12.44.88.212 128 frag:6911 / Padding
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
0.0.0.0 > 12.44.88.212 128 frag:6911 / Padding
```

server-router:

```
   Inside: 0.0.0.0 --> 12.44.88.212
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 0.0.0.0 --> 12.44.88.212
```

Wireshark:



Apply a display filter … <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	2025-10-26 07:3…	10.9.0.5	10.9.0.11	UDP	126	56968 → 9090 Len=84
2	2025-10-26 07:3…	10.9.0.5	10.9.0.11	UDP	126	56968 → 9090 Len=84
3	2025-10-26 07:3…	10.9.0.5	10.9.0.11	UDP	126	56968 → 9090 Len=84
4	2025-10-26 07:3…	10.9.0.5	10.9.0.11	UDP	126	56968 → 9090 Len=84
5	2025-10-26 07:3…	10.9.0.5	10.9.0.11	UDP	126	56968 → 9090 Len=84
6	2025-10-26 07:3…	02:c9:4d:5c:4a:e2	6a:22:53:99:fd:f7	ARP	42	Who has 10.9.0.11? Tell 10.9.0.5
7	2025-10-26 07:3…	6a:22:53:99:fd:f7	02:c9:4d:5c:4a:e2	ARP	42	10.9.0.11 is at 6a:22:53:99:fd:f7
8	2025-10-26 07:3…	10.9.0.5	10.9.0.11	UDP	126	56968 → 9090 Len=84
9	2025-10-26 07:3…	10.9.0.5	10.9.0.11	UDP	126	56968 → 9090 Len=84
10	2025-10-26 07:3…	10.9.0.5	10.9.0.11	UDP	126	56968 → 9090 Len=84
11	2025-10-26 07:3…	10.9.0.5	10.9.0.11	UDP	126	56968 → 9090 Len=84
12	2025-10-26 07:3…	10.9.0.5	10.9.0.11	UDP	126	56968 → 9090 Len=84
13	2025-10-26 07:3…	10.9.0.5	10.9.0.11	UDP	90	56968 → 9090 Len=48
14	2025-10-26 07:3…	02:c9:4d:5c:4a:e2	6a:22:53:99:fd:f7	ARP	42	Who has 10.9.0.11? Tell 10.9.0.5
15	2025-10-26 07:3…	6a:22:53:99:fd:f7	02:c9:4d:5c:4a:e2	ARP	42	10.9.0.11 is at 6a:22:53:99:fd:f7

```
Frame 1: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits) on interface br-98135987dabb, id 0
Ethernet II, Src: 02:c9:4d:5c:4a:e2 (02:c9:4d:5c:4a:e2), Dst: 6a:22:53:99:fd:f7 (6a:22:53:99:fd:f7)
Internet Protocol Version 4, Src: 10.9.0.5, Dst: 10.9.0.11
User Datagram Protocol, Src Port: 56968, Dst Port: 9090

0000  6a 22 53 99 fd f7 02 c9  4d 5c 4a e2 08 00 45 00   j"S····M\J···E·
0010  00 70 0c 3c 40 00 40 11  1a 20 0a 09 00 05 0a 09   ·p·<@·@· ·······
0020  00 0b de 88 23 82 00 5c  14 8f 45 00 00 54 48 d2   ····#··\ ··E··TH·
0030  40 00 40 01 06 1e c0 a8  35 63 c0 a8 35 05 08 00   @·@·····5c··5···
0040  6d 4f 00 06 00 01 12 ce  fd 68 00 00 00 00 b0 9f   mO······ ·h······
0050  0b 00 00 00 00 00 10 11  12 13 14 15 16 17 18 19   ········ ········
0060  1a 1b 1c 1d 1e 1f 20 21  22 23 24 25 26 27 28 29   ······ ! "#$%&'()
```

br-98135987dabb: <live capture in progress> Packets: 15 · Displayed: 15 (100.0%) Profile: Default

Question: What is printed out on VPN Server? Why?
Answer: The VPN server prints lines like:

From UDP: 192.168.53.99 → 192.168.53.5 and From UDP: 192.168.53.99 → 192.168.60.5. These represent the original IP packets encapsulated inside UDP and

sent by the client. This output confirms that the client's script (tun_client.py) successfully captured IP packets from the TUN interface, encapsulated them inside UDP, and sent them to the server where they were received and decapsulated correctly.

Step5: On Host U, let us ping Host V, and see whether the ICMP packet is sent to VPN Server through the tunnel.

26)Command: ping 192.168.60.5 (On Client - 10.9.0.5)

client-10.9.0.5 (pinging 192.168.60.5):

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
10 packets transmitted, 0 received, 100% packet loss, time 9225ms

client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$
```

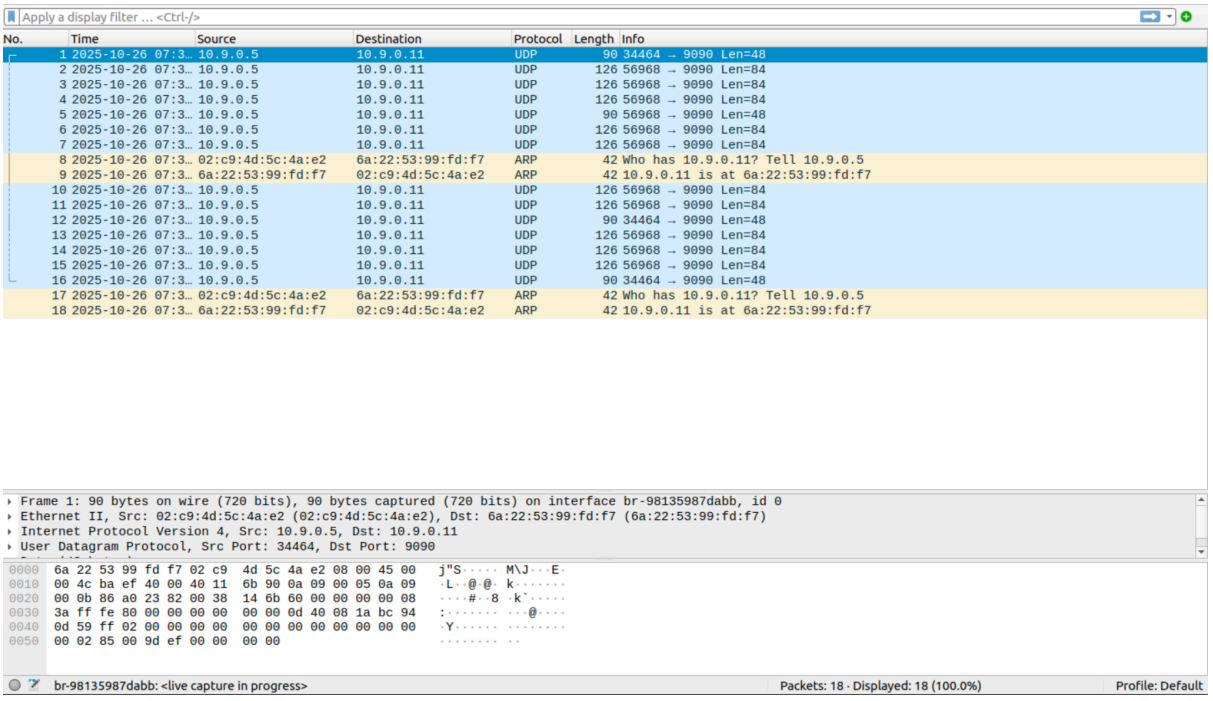
client-10.9.0.5 (running the tun_client.py script):

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>Interface Name: CS4331
RTNETLINK answers: File exists
0.0.0.0 > 13.64.8.26 128 frag:6911 / Padding
0.0.0.0 > 13.64.8.26 128 frag:6911 / Padding
0.0.0.0 > 13.64.8.26 128 frag:6911 / Padding
0.0.0.0 > 13.64.8.26 128 frag:6911 / Padding
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
0.0.0.0 > 12.44.88.212 128 frag:6911 / Padding
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
0.0.0.0 > 13.64.8.26 128 frag:6911 / Padding
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
```

server-router:

```
   Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 0.0.0.0 --> 12.44.88.212
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:34464 --> 0.0.0.0:9090
   Inside: 0.0.0.0 --> 13.64.8.26
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.60.5
```

Wireshark:



Explanation:

At this stage, only the client-to-server direction of the VPN tunnel is complete. Packets reach the server and are decapsulated, but replies are not yet sent back because the server does not handle returning traffic.

Step6: check the routing run the following command on Client - 10.9.0.5

27)Command: ip route (On Client - 10.9.0.5)

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$ip route
default via 10.9.0.1 dev eth0
10.9.0.0/24 dev eth0 proto kernel scope link src 10.9.0.5
192.168.53.0/24 dev CS4330 proto kernel scope link src 192.168.53.99
192.168.53.0/24 dev CS4331 proto kernel scope link src 192.168.53.99
192.168.60.0/24 dev CS4330 scope link
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$
```

Explanation:

On the server-router, tun_server.py receives UDP packets from the client and prints their inner IP details (e.g., 192.168.53.99 → 192.168.53.5). This verifies that client-side encapsulation and UDP transmission are functioning correctly.

On the client side, tun_client.py captures ICMP packets, encapsulates them into UDP, and sends them to the server. The ping fails because the server does not yet send replies.

The routing table shows that traffic destined for 192.168.60.0/24 is redirected through the virtual interface, confirming that routing into the tunnel is working correctly.

Task 4: Set Up the VPN Server

tun_server.py gets a packet from the tunnel, it needs to feed the packet to the kernel, so the kernel can route the packet towards its final destination. This needs to be done through a TUN interface, just like what we did in Task 2. We have modified 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.

Note - In tun_server1.py replace “tun” with the last 5 characters of your SRN in line 18.

Step1: Instead of using tun as the prefix of the interface name, use the last 5 digits of your SRN as the prefix

28)Edit: In tun_server1.py replace “tun” with the last 5 characters of your SRN in line 16.

Line 18 - ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)

```
  GNU nano 6.2                          tun_server1.py
#!/usr/bin/python3

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

IP_A = "0.0.0.0"
PORT = 9090

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

# Create a tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'CS433%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
ifname  = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
                              [ Wrote 35 lines ]
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^/ Go To Line
```

```
seed@seedvm2004:~/Desktop/Lab8/volumes$ ls
tun1.py        tun_client_select.py  tun_server1.py  tun_server_select.py
tun_client.py  tun.py                tun_server.py
seed@seedvm2004:~/Desktop/Lab8/volumes$ nano tun_server1.py
seed@seedvm2004:~/Desktop/Lab8/volumes$
```

Step2: Run the following commands and see the new tunnel interface, it should be “SRN0” (server-router). Make the above tun_server1.py program executable, and run it using the root privilege.

29) Command: (On server-router)

chmod a+x tun_server1.py

./tun_server1.py &

ip addr

server-router:

```
server-router:PES1UG23CS433:PranavHemanth:/volumes
$>chmod a+x tun_server1.py
server-router:PES1UG23CS433:PranavHemanth:/volumes
$>./tun_server1.py
Interface Name: CS4330
```

Step3: On host-192.168.60.5 run -

30) Command: tcpdump -i eth0 -n

```
host-192.168.60.5:PES1UG23CS433:PranavHemanth:/
$>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
```

Step4: On Host U, ping a host in the 192.168.60.0/24 network.

31)Command: ping 192.168.60.5 (On Client - 10.9.0.5)

client-10.9.0.5 (pinging 192.168.60.5):

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5110ms

client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$
```

client-10.9.0.5 (running tun script):

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>0.0.0.0 > 13.64.8.26 128 frag:6911 / Padding
0.0.0.0 > 12.44.88.212 128 frag:6911 / Padding
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
```

server-router:

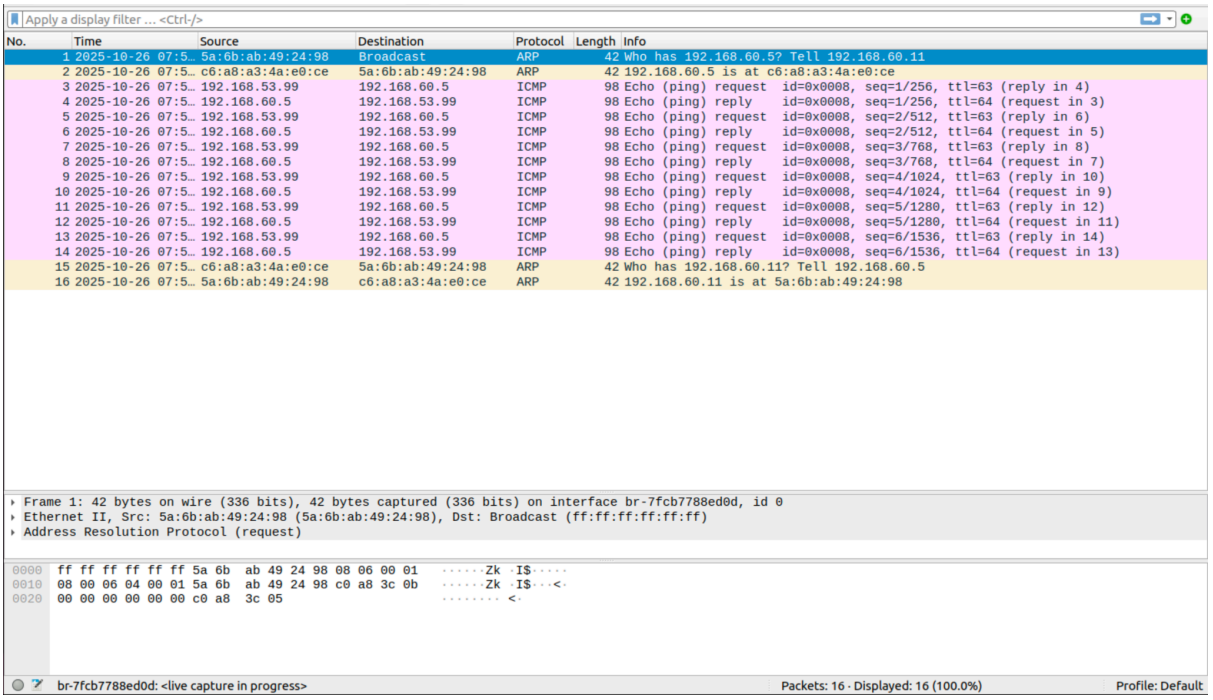
```
server-router:PES1UG23CS433:PranavHemanth:/volumes
$>chmod a+x tun_server1.py
server-router:PES1UG23CS433:PranavHemanth:/volumes
$>./tun_server1.py
Interface Name: CS4330
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:56968 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.60.5
```

host-192.168.60.5:

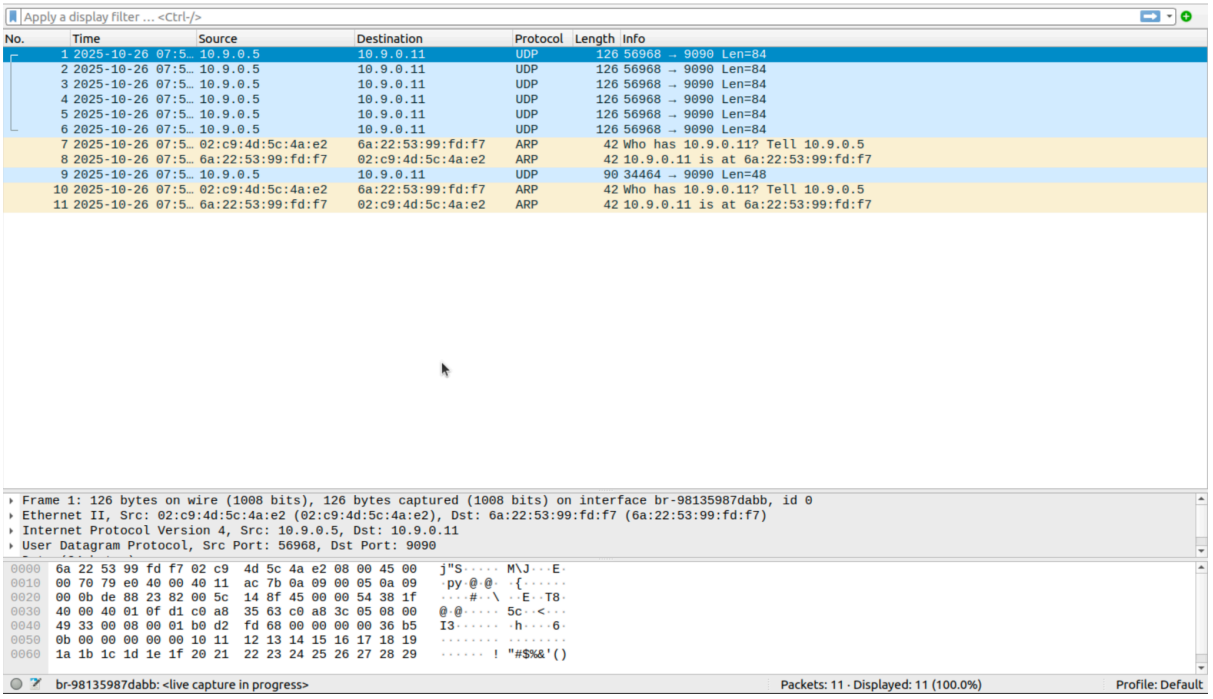
```
host-192.168.60.5:PES1UG23CS433:PranavHemanth:/
$>tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
07:50:08.786283 ARP, Request who-has 192.168.60.5 tell 192.168.60.11, length 28
07:50:08.786410 ARP, Reply 192.168.60.5 is-at c6:a8:a3:4a:e0:ce, length 28
07:50:08.786455 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 8, seq 1, length 64
07:50:08.786613 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 8, seq 1, length 64
07:50:09.784041 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 8, seq 2, length 64
07:50:09.784208 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 8, seq 2, length 64
07:50:10.808373 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 8, seq 3, length 64
07:50:10.808480 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 8, seq 3, length 64
07:50:11.831302 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 8, seq 4, length 64
07:50:11.831374 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 8, seq 4, length 64
07:50:12.855873 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 8, seq 5, length 64
07:50:12.856065 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 8, seq 5, length 64
07:50:13.880521 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 8, seq 6, length 64
07:50:13.880632 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 8, seq 6, length 64
07:50:14.005128 ARP, Request who-has 192.168.60.11 tell 192.168.60.5, length 28
07:50:14.005330 ARP, Reply 192.168.60.11 is-at 5a:6b:ab:49:24:98, length 28
```

```
07:50:14.005128 ARP, Request who-has 192.168.60.11 tell 192.168.60.5, length 28
07:50:14.005330 ARP, Reply 192.168.60.11 is-at 5a:6b:ab:49:24:98, length 28
^C
16 packets captured
16 packets received by filter
0 packets dropped by kernel
host-192.168.60.5:PES1UG23CS433:PranavHemanth:/
$>
```

Wireshark:



Wireshark (on client - not very relevant to experiment):



Explanation:
On the server-router, tun_server1.py receives encapsulated UDP packets from the client and decapsulates them, writing the inner IP packets into its virtual interface. This allows the kernel's routing table to forward them to their final destination (e.g., Host V).

On the client side, tun_client.py shows outbound ICMP packets being captured and encapsulated into UDP, while Host V's tcpdump confirms receiving ICMP echo requests.

Even though the client sees 100% packet loss, this confirms a fully functional one-way VPN tunnel from client to server.

Step5: Kill the Tunnel Process - (On Client - 10.9.0.5)

32) Command: (On Client - 10.9.0.5)

kill %1

jobs

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>jobs
[1]-  Running                 ./tun_client.py &
[2]+  Running                 ./tun_client.py &
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>kill %1
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>kill %2
[1]-  Terminated              ./tun_client.py
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>jobs
[2]+  Terminated              ./tun_client.py
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>
```

Explanation:
After Task 4, the VPN tunnel works in only one direction (client → server → Host V).

In Task 5, the setup is enhanced with non-blocking I/O using select() to handle both directions simultaneously, creating a fully functional, bidirectional VPN

Task 5: Handling Traffic in Both Directions

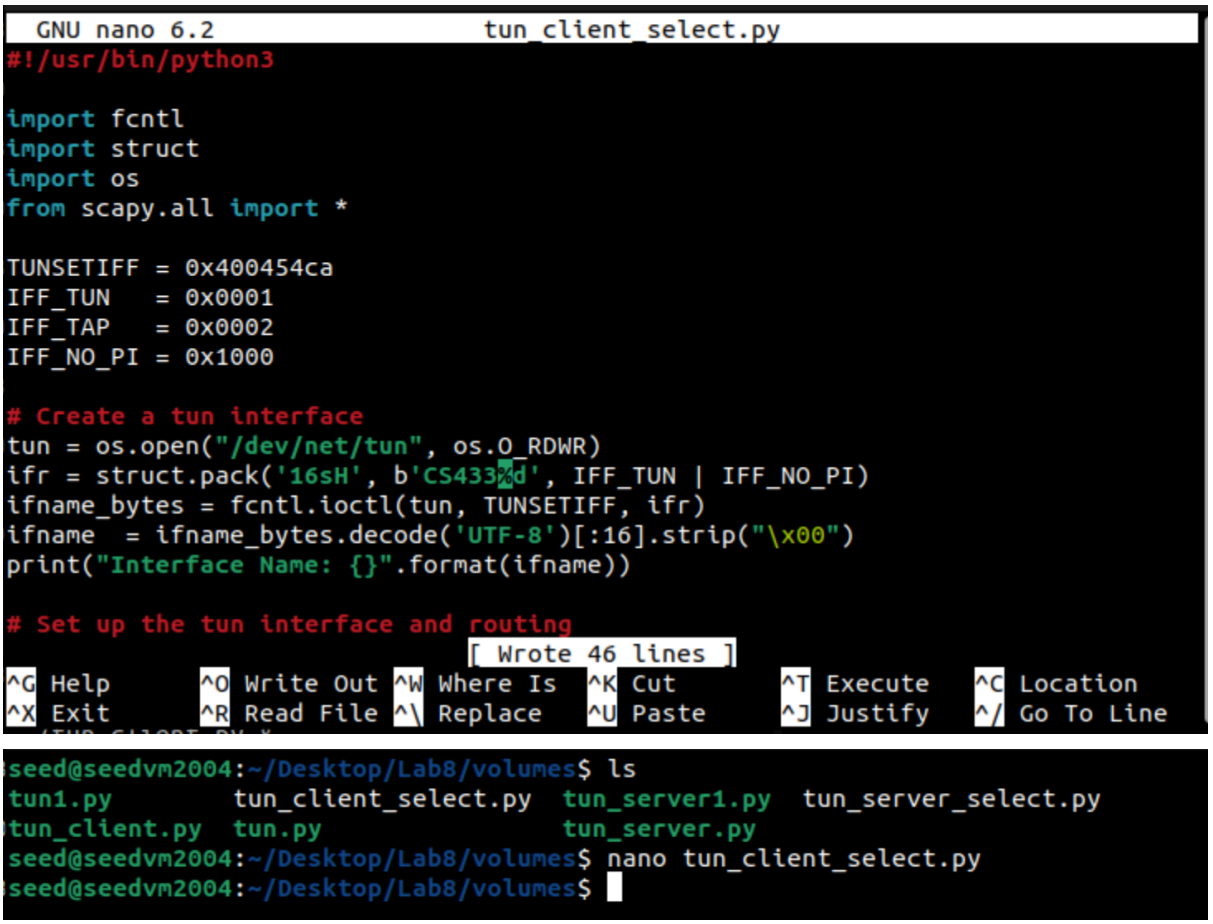
After getting to this point, one direction of your tunnel is complete, i.e., we can send packets from Host U to Host V via the tunnel. If we look at the Wireshark trace on Host V, we can see that Host V has sent out the response, but the packet gets dropped somewhere. This is because our tunnel is only one directional; we need to set up its other direction, so returning traffic can be tunneled back to Host U.

To achieve that, our TUN client and server programs need to read data from two interfaces, the TUN interface and the socket interface. All these interfaces are represented by file descriptors, so we need to monitor them to see whether there is data coming from them. One way to do that is to keep polling them, and see whether there is data on each of the interfaces. The performance of this approach is undesirable, because the process has to keep running in an idle loop when there is no data. Another way is to read from an interface. By default, read is blocking, i.e The process will be suspended if there is no data. When data becomes available, the process will be unblocked, and its execution will continue. This way, it does not waste CPU time when there is no data.

We use two new programs tun_client_select.py and tun_server_select.py. In both the mentioned programs replace “tun” with the last 5 characters of your SRN in line 15 (client) and line 19 (server).
The line is - “ ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)”

Step1: Instead of using tun as the prefix of the interface name, use the last 5 digits of your SRN as the prefix

33)Edit: In tun_client_select.py replace “tun” with the last 5 characters of your SRN in line 16.
Line 18 - ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)



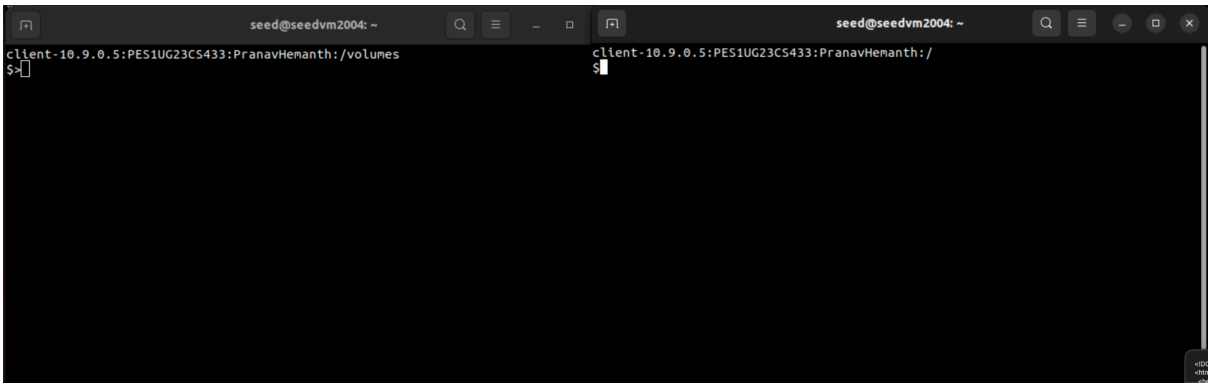
Step2: Instead of using tun as the prefix of the interface name, use the last 5 digits of your SRN as the prefix

34)Edit: In tun_server_select.py replace “tun” with the last 5 characters of your SRN in line 16.
Line 18 - ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)

```
  GNU nano 6.2                    tun_server_select.py
#!/usr/bin/python3

#import select
import fcntl
import struct
import os
from scapy.all import *

IP_A = "0.0.0.0"
PORT = 9090

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

# Create a tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'CS433%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
                              [ Wrote 51 lines ]
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^/ Go To Line
```

```
seed@seedvm2004:~/Desktop/Lab8/volumes$ ls
tun1.py         tun_client_select.py  tun_server1.py  tun_server_select.py
tun_client.py   tun.py                tun_server.py
seed@seedvm2004:~/Desktop/Lab8/volumes$ nano tun_server_select.py
seed@seedvm2004:~/Desktop/Lab8/volumes$
```

Open two terminals of the Client - 10.9.0.5 Machine, this improves comprehensibility for what we are about to execute.



You will need wireshark for this task, capturing the packets on the client interface.

Step3: Run the following commands and see the new tunnel interface, it should be “SRN0” (server-router). Make the above tun_server_select.py program executable, and run it using the root privilege.

35)Command: (On server-router)

chmod a+x tun_server_select.py

./tun_server_select.py

server-router:

```
server-router:PES1UG23CS433:PranavHemanth:/volumes
$>chmod a+x tun_server_select.py
server-router:PES1UG23CS433:PranavHemanth:/volumes
$>./tun_server_select.py
Interface Name: CS4330
From tun    ==>: 0.0.0.0 --> 114.150.103.97
From tun    ==>: 0.0.0.0 --> 114.150.103.97
```

Step4: Run the following commands and see the new tunnel interface, it should be "SRN0" (On Client - 10.9.0.5). Make the above tun_client_select.py program executable, and run it using the root privilege.

36)Command: (On Client - 10.9.0.5)

chmod a+x tun_client_select.py

./tun_client_select.py

client - 10.9.0.5:

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>chmod a+x tun_client_select.py
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>./tun_client_select.py
Interface Name: CS4330
From tun    ==>: 0.0.0.0 --> 13.161.148.163
From tun    ==>: 0.0.0.0 --> 13.161.148.163
```

Step5: On Host U, let us ping Host V

37)Command: ping 192.168.60.5 (On Client - 10.9.0.5)

client-10.9.0.5 (pinging 192.168.60.5):

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$ping 192.168.60.5
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=47.3 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=5.50 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=5.24 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=2.90 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=5.31 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=4.92 ms
^C
--- 192.168.60.5 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5010ms
rtt min/avg/max/mdev = 2.898/11.859/47.296/15.871 ms
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$
```

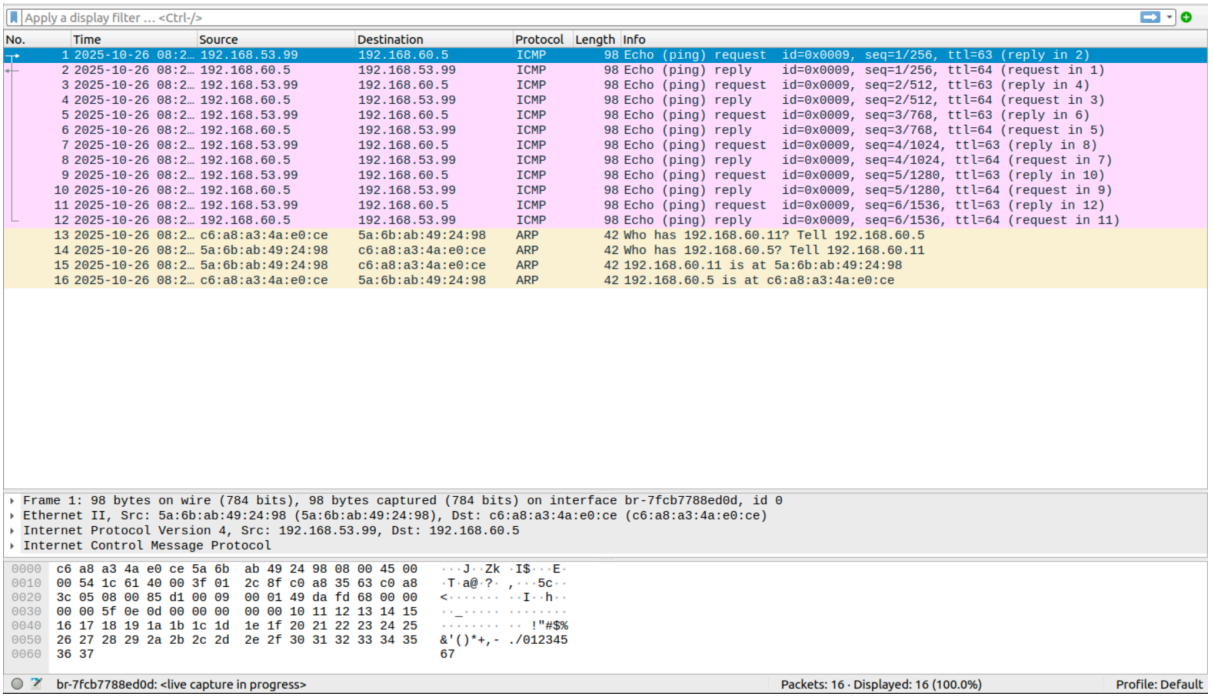
client-10.9.0.5 (running the tun_client_select.py script):

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>chmod a+x tun_client_select.py
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/volumes
$>./tun_client_select.py
Interface Name: CS4330
From tun    ==>: 0.0.0.0 --> 13.161.148.163
From tun    ==>: 0.0.0.0 --> 13.161.148.163
From tun    ==>: 0.0.0.0 --> 13.161.148.163
From socket <==: 0.0.0.0 --> 114.150.103.97
From tun    ==>: 0.0.0.0 --> 13.161.148.163
From tun    ==>: 0.0.0.0 --> 13.161.148.163
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 socket <==: 192.168.60.5 --> 192.168.53.99
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 socket <==: 192.168.60.5 --> 192.168.53.99
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 socket <==: 192.168.60.5 --> 192.168.53.99
From tun    ==>: 0.0.0.0 --> 13.161.148.163
```

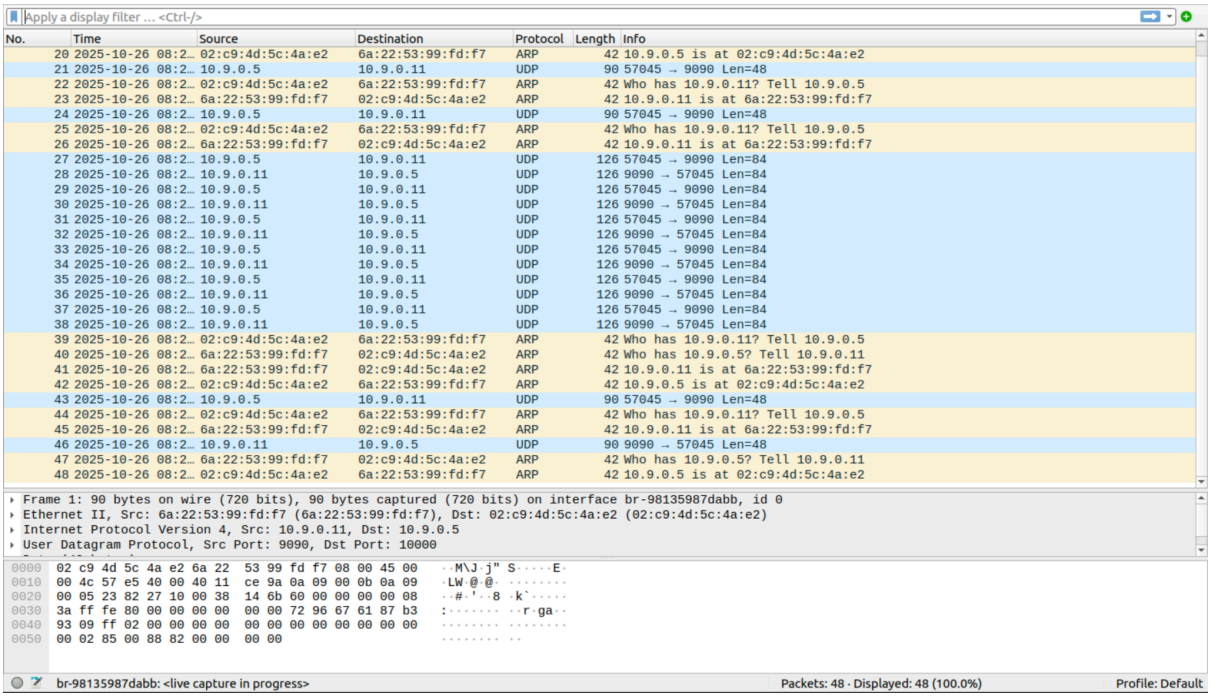
server-router (running the tun_server_select.py script):

```
server-router:PES1UG23CS433:PranavHemanth:/volumes
$>chmod a+x tun_server_select.py
server-router:PES1UG23CS433:PranavHemanth:/volumes
$>./tun_server_select.py
Interface Name: CS4330
From tun    ==>: 0.0.0.0 --> 114.150.103.97
From tun    ==>: 0.0.0.0 --> 114.150.103.97
From tun    ==>: 0.0.0.0 --> 114.150.103.97
From tun    ==>: 0.0.0.0 --> 114.150.103.97
From tun    ==>: 0.0.0.0 --> 114.150.103.97
From socket <==: 0.0.0.0 --> 13.161.148.163
From socket <==: 0.0.0.0 --> 13.161.148.163
From socket <==: 0.0.0.0 --> 13.161.148.163
From tun    ==>: 0.0.0.0 --> 114.150.103.97
From socket <==: 0.0.0.0 --> 13.161.148.163
From socket <==: 0.0.0.0 --> 13.161.148.163
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 0.0.0.0 --> 13.161.148.163
From tun    ==>: 0.0.0.0 --> 114.150.103.97
```

Wireshark:



Wireshark (on client not relevant to experiment):



Explanation:

In this stage, the tunnel becomes bidirectional. The server (tun_server_select.py)

Step5: On Host U, let us telnet Host V

38)Command: telnet 192.168.60.5 (On Client - 10.9.0.5)

client-10.9.0.5 (telnet-ing 192.168.60.5):

```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.6 LTS
6942a84cfb59 login: seed
Password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-160-generic aarch64)

 * 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.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

seed@6942a84cfb59:~$
```

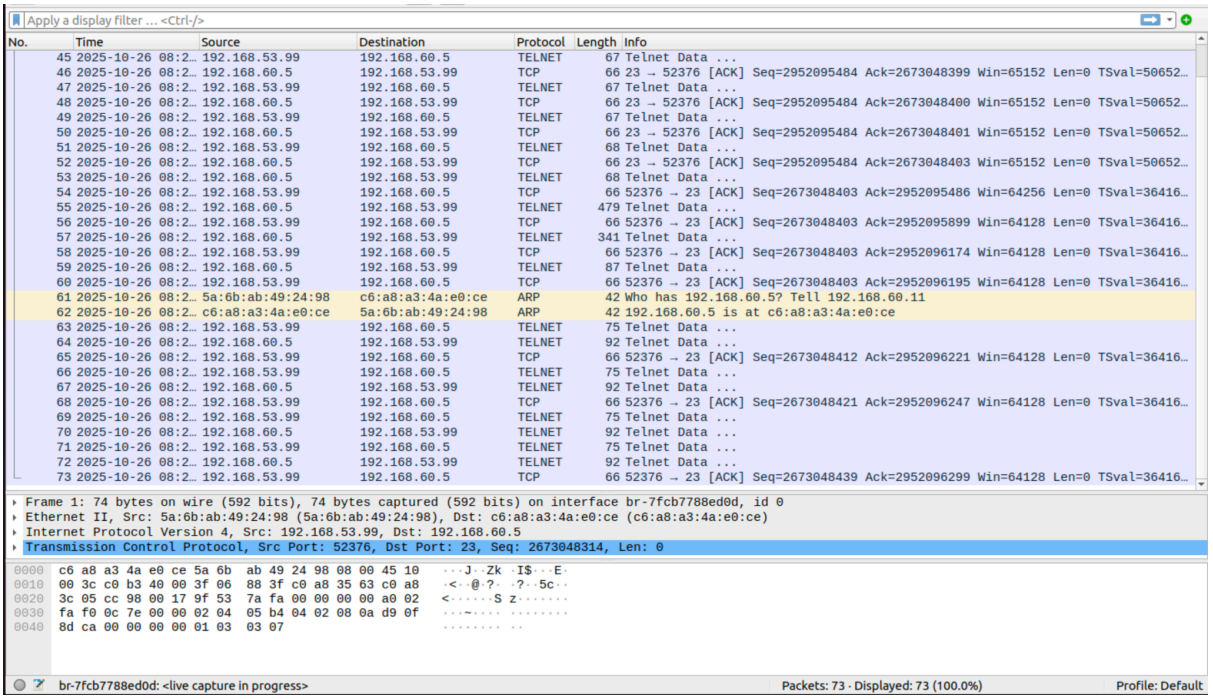
client-10.9.0.5 (running the tun_client_select.py script):

```
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
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
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
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
From socket <==: 192.168.60.5 --> 192.168.53.99
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 socket <==: 192.168.60.5 --> 192.168.53.99
From tun    ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.60.5 --> 192.168.53.99
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 socket <==: 192.168.60.5 --> 192.168.53.99
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
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
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun    ==>: 192.168.53.99 --> 192.168.60.5
```

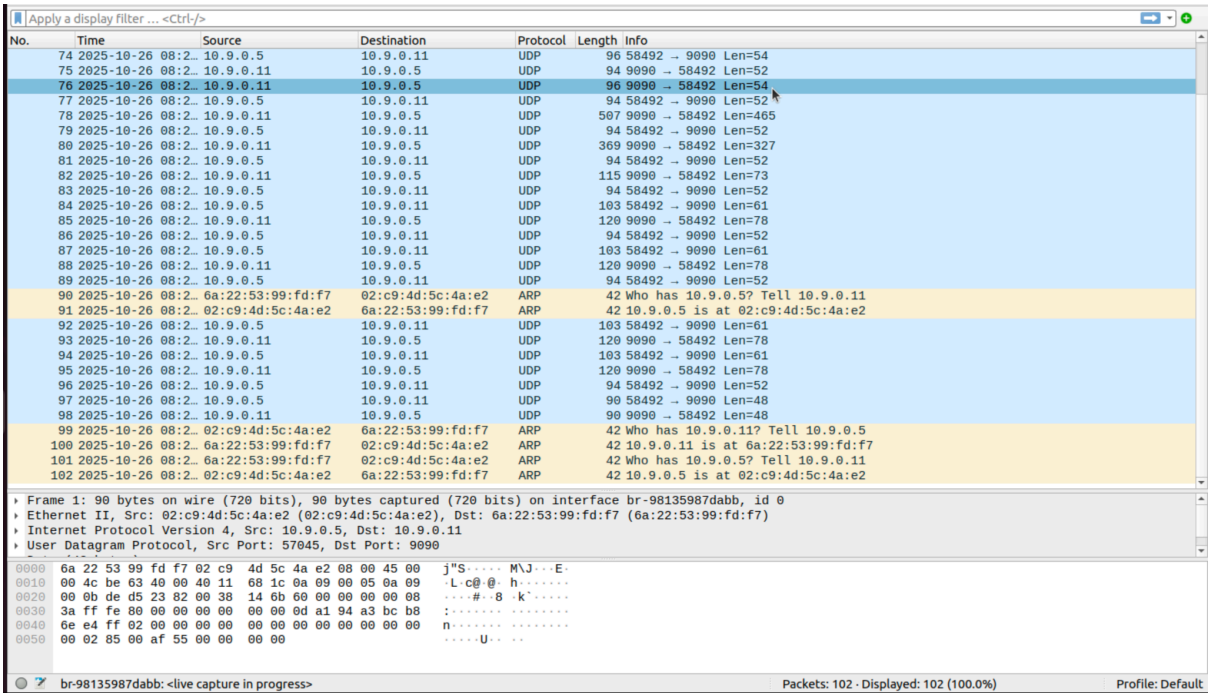
server-router (running the tun_server_select.py script):

```
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 0.0.0.0 --> 255.80.54.210
```

Wireshark:



Wireshark (on client not relevant to experiment):



Explanation:

In this stage, the tunnel becomes bidirectional. The server (tun_server_select.py) and client (tun_client_select.py) both use select() to handle simultaneous reads and writes from the TUN interface and UDP socket.

The client's ping to 192.168.60.5 now shows 0% packet loss, meaning that encapsulation, transmission, decapsulation, and routing are all functioning correctly in both directions.

Wireshark captures show only UDP packets between 10.9.0.5 and 10.9.0.11 (port 9090), demonstrating that the ICMP traffic is successfully encapsulated and hidden within UDP confirming the VPN's stealth and functionality.

Finally, a successful telnet session proves that the tunnel can carry full TCP traffic, not just ICMP, showing that the VPN supports complete two-way communication for any network protocol.

Explanation:
The final experiment tests tunnel resilience under real network conditions.

By intentionally breaking and restoring the tunnel mid-session, we can observe how higher-level protocols like TCP behave during VPN disruptions.

Task 6: Tunnel-Breaking Experiment

On Host U (10.9.0.5), telnet to Host V (192.168.60.5) .
(Perform Task 5 again, incase you have closed both the Client and Server Tunnel Connections)

Step1: Log on and while keeping the telnet connection alive, we break the VPN tunnel by stopping the tun_server_select.py program - Ctrl + C in server-router

39)Command: telnet 192.168.60.5 (On Client - 10.9.0.5)

server-router:

```
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 0.0.0.0 --> 255.80.54.210
From tun    ==>: 0.0.0.0 --> 169.43.218.250
From tun    ==>: 0.0.0.0 --> 169.43.218.250
From socket <==: 0.0.0.0 --> 255.80.54.210
^CTraceback (most recent call last):
  File "./tun_server_select.py", line 38, in <module>
    ready, _, _ = select.select(fds, [], [])
KeyboardInterrupt

server-router:PES1UG23CS433:PranavHemanth:/volumes
$>
```

Step2: We then type something in the telnet window.

```
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.6 LTS
6942a84cfb59 login: seed
Password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-160-generic aarch64)

 * 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.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

seed@6942a84cfb59:~$
```

(Typed characters are not visible here)

```
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.6 LTS
6942a84cfb59 login: seed
Password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-160-generic aarch64)

 * 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.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

seed@6942a84cfb59:~$ this is lab8
```

```
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 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
From socket <==: 192.168.60.5 --> 192.168.53.99
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 socket <==: 192.168.60.5 --> 192.168.53.99
From tun    ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.60.5 --> 192.168.53.99
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 socket <==: 192.168.60.5 --> 192.168.53.99
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
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
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun    ==>: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 0.0.0.0 --> 255.80.54.210
From socket <==: 0.0.0.0 --> 169.43.218.250
From socket <==: 0.0.0.0 --> 169.43.218.250
From tun    ==>: 0.0.0.0 --> 255.80.54.210
From tun    ==>: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.53.99 --> 192.168.60.5
```

(Typing in the client telnet session causes packet to move through the tunnel and can be seen above)

Explanation:

In this experiment, the router is configured to distribute incoming UDP traffic on port 8080 evenly across three internal servers (192.168.60.5, 192.168.60.6, 192.168.60.7) using the statistic module in nth mode. Three DNAT rules are added to the PREROUTING chain of the nat table. The first rule redirects every third packet

Questions:

Do you see what you type? What happens to the TCP connection? Is the connection broken? Explain.
Answer: When the server-side VPN script is stopped, the telnet window still echoes keystrokes locally, but no output or response appears. This is because the tunnel is broken- the packets from the client are still being sent out, but no server process is listening on port 9090 to receive them.

The TCP connection isn't terminated immediately; instead, it becomes "frozen." TCP's reliability mechanisms cause it to keep retransmitting packets, waiting for acknowledgments that never arrive.

Step3: Let us now reconnect the VPN tunnel (do not wait for too long).
40)Command: ./tun_server_select.py (On server-router)

```
From tun    ==>: 0.0.0.0 --> 169.43.218.250
From tun    ==>: 0.0.0.0 --> 169.43.218.250
From socket <==: 0.0.0.0 --> 255.80.54.210
^CTraceback (most recent call last):
  File "./tun_server_select.py", line 38, in <module>
    ready, _, _ = select.select(fds, [], [])
KeyboardInterrupt

server-router:PES1UG23CS433:PranavHemanth:/volumes
$>./tun_server_select.py
Interface Name: CS4330
From tun    ==>: 0.0.0.0 --> 154.164.223.131
From tun    ==>: 0.0.0.0 --> 154.164.223.131
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 0.0.0.0 --> 154.164.223.131
From tun    ==>: 0.0.0.0 --> 154.164.223.131
From tun    ==>: 0.0.0.0 --> 154.164.223.131
From socket <==: 0.0.0.0 --> 255.80.54.210
From tun    ==>: 0.0.0.0 --> 154.164.223.131
```

Step4: Log on and while keeping the telnet connection alive we type in the telnet session

41)Command: telnet 192.168.60.5 (On Client - 10.9.0.5)

server-router:

```
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 0.0.0.0 --> 154.164.223.131
```

client-10.9.0.5 (with the telnet session):

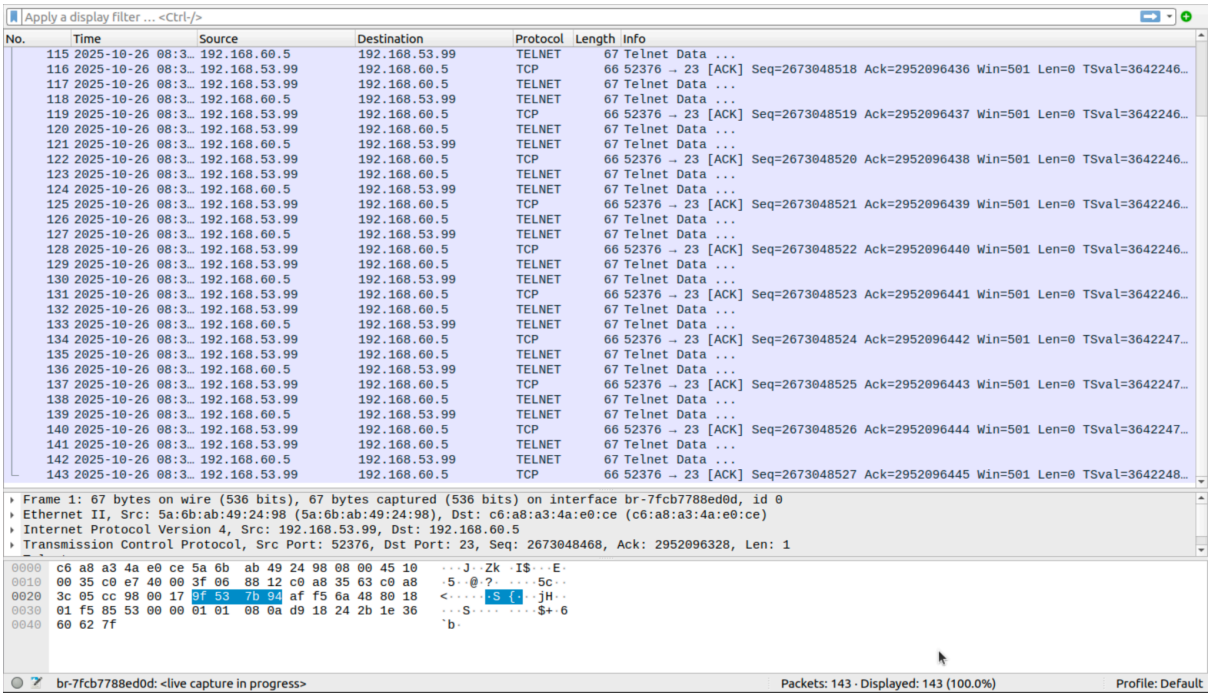
```
client-10.9.0.5:PES1UG23CS433:PranavHemanth:/
$telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.6 LTS
6942a84cfb59 login: seed
Password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-160-generic aarch64)

 * 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.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

seed@6942a84cfb59:~$ this is lab8
```

client-10.9.0.5 (running the tun_client_select.py script):

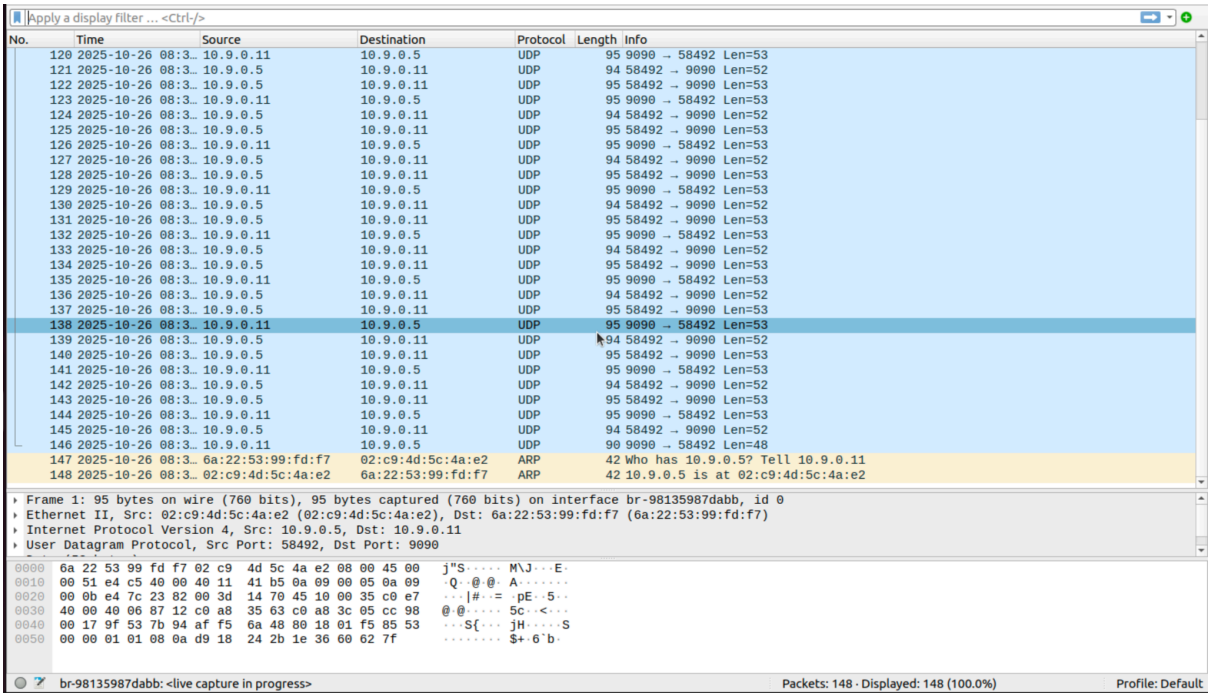
```
From tun    ==>: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun    ==>: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun    ==>: 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
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
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
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
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
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
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
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 socket <==: 0.0.0.0 --> 154.164.223.131
```

Aug -Dec 2025 Assignment SUBMISSION_UE23CS343AB6

Wireshark:



Wireshark (not relevant to experiment):



Explanation:

During an active telnet session, when the VPN server (tun_server_select.py) is terminated, the client's terminal stops receiving responses. The session appears frozen because the client's TCP stack keeps retransmitting packets that never reach the remote host.

Wireshark confirms this by showing the client sending UDP packets and receiving ICMP “Port unreachable” messages from the server (since no process is listening on port 9090).

When the server script is restarted, the tunnel reestablishes immediately. The pending TCP retransmissions finally succeed, and the telnet session resumes seamlessly, demonstrating TCP’s reliability and the tunnel’s resilience to temporary interruptions.

Questions:

Once the tunnel is re-established, what is going to happen to the telnet connection? Please describe and explain your observations.
Answer: When the VPN server is restarted, the tunnel becomes active again. The client’s pending TCP retransmissions finally reach the destination, and the telnet session resumes seamlessly without requiring a new connection.

This shows how TCP maintains session reliability over temporary network failures; the connection “pauses” during tunnel downtime and “revives” when connectivity is restored.

Explanation:

This experiment illustrates TCP’s reliability over unreliable transport mediums. Even when the VPN tunnel breaks, TCP preserves session state. Once the tunnel comes back up, the protocol automatically continues the session, proving that the VPN can handle transient interruptions gracefully.