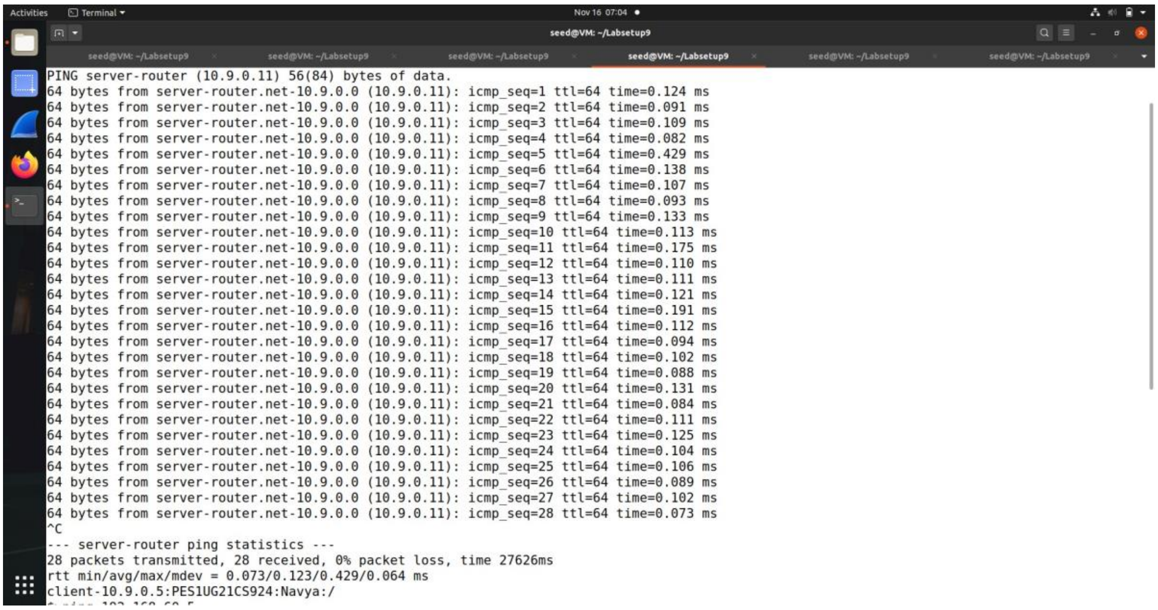
CNS LAB 9

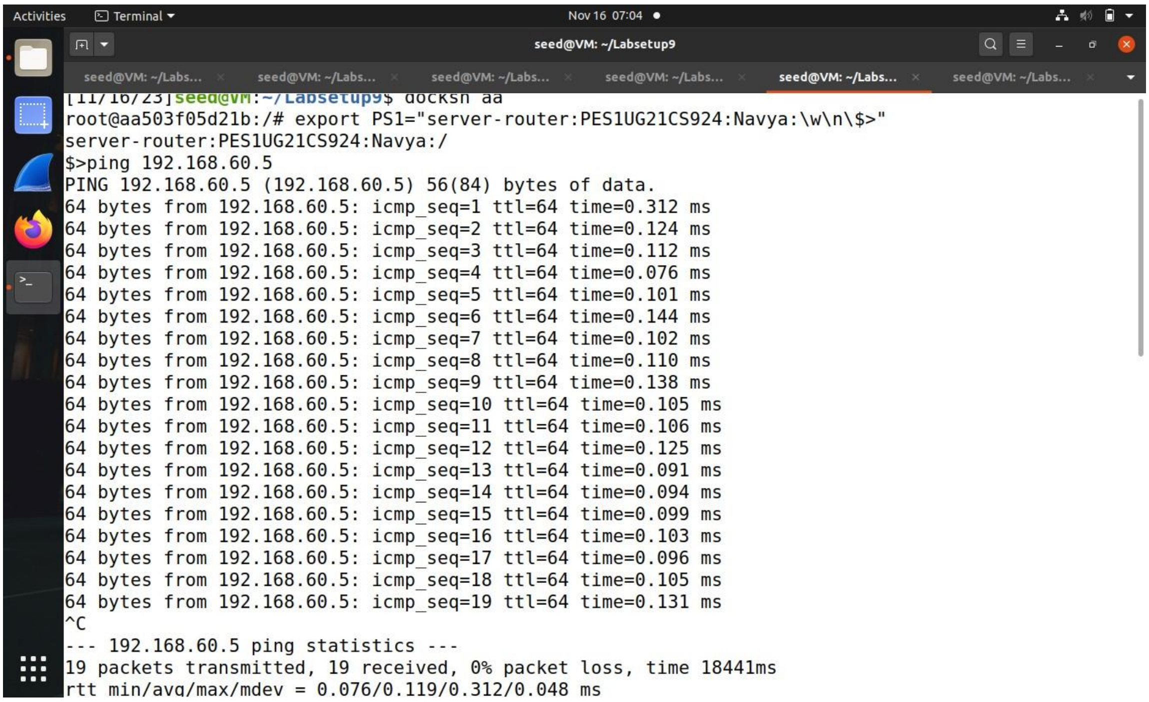
NAME: NAVYA PERAM

SRN: PES1UG21CS924

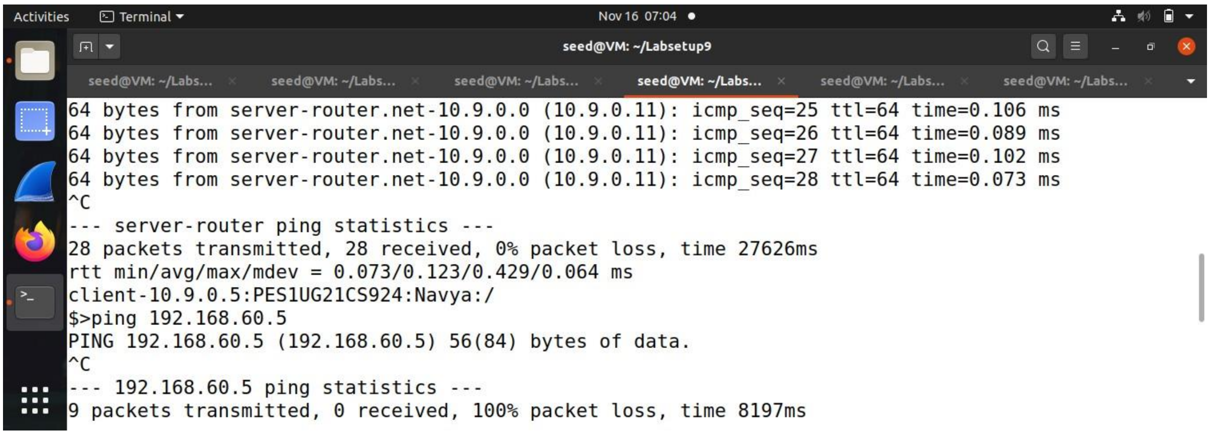
Task 1



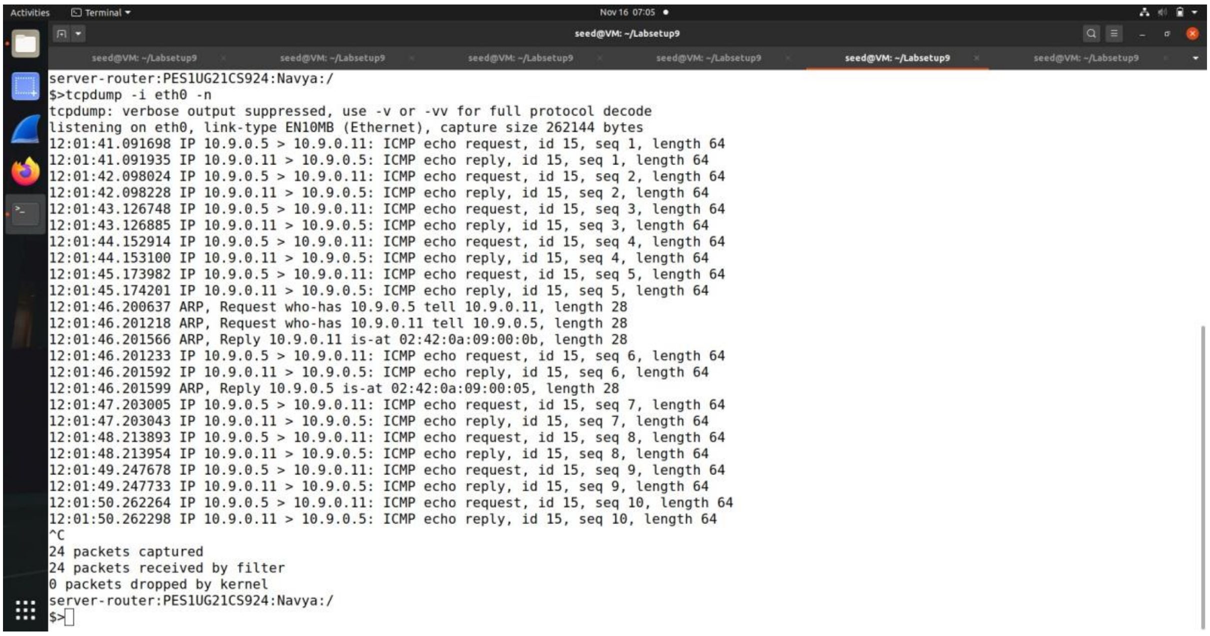
On pinging the server, we can reach it immediately which shows that the server is properly configured



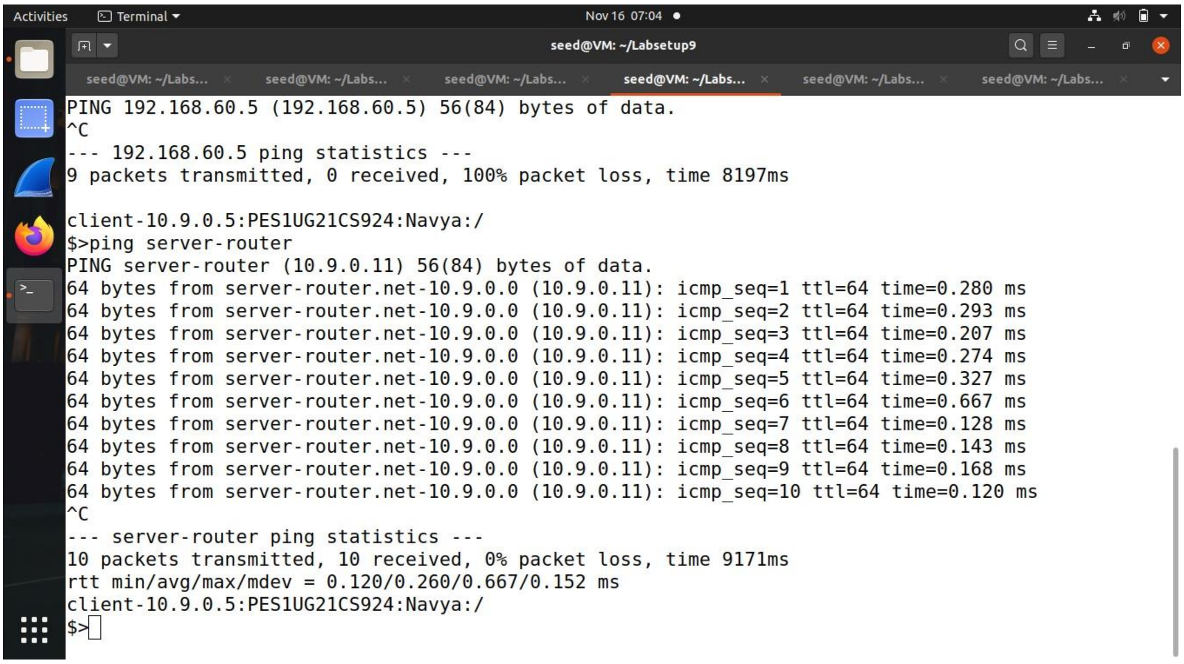
Here, we can ping the host V from the VPN server which shows that the VPN server is configured in the correct manner.



We cannot ping between host U and host V, since they are present on different subnets.

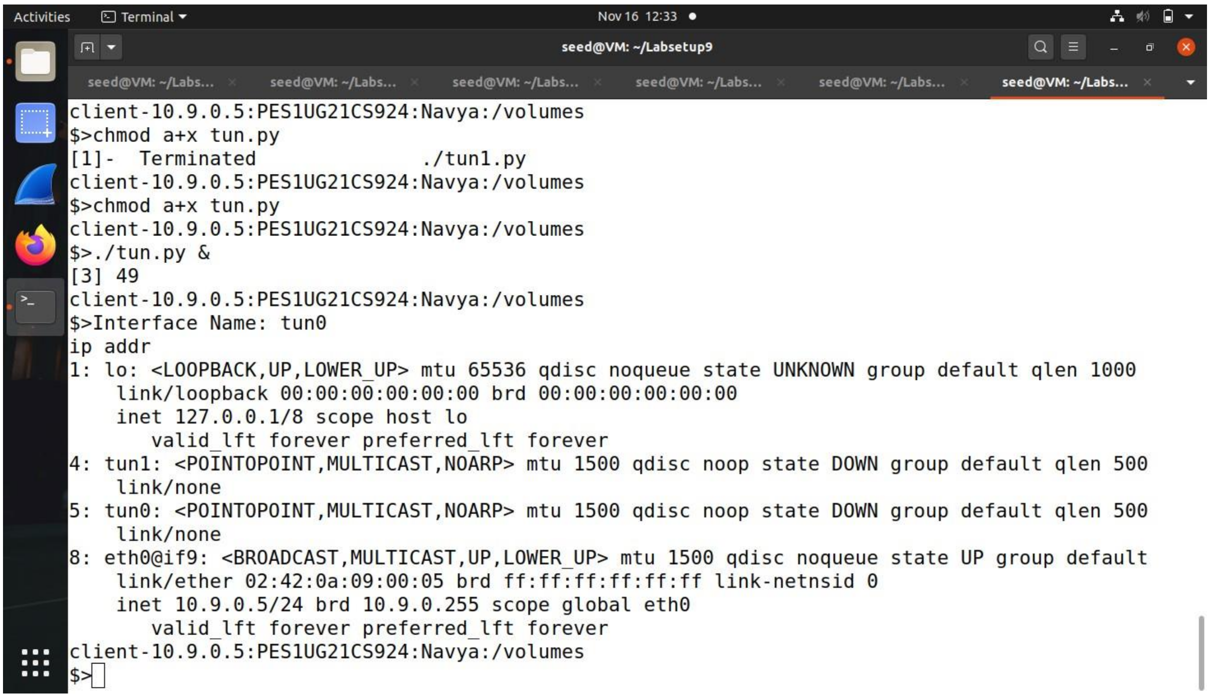


The tcpdump command is used to capture and analyze network traffic. The output of the tcpdump command shows the sending and receiving of ICMP echo requests and replies, ping packets, to and from the IP address 10.9.0.11, through the router.

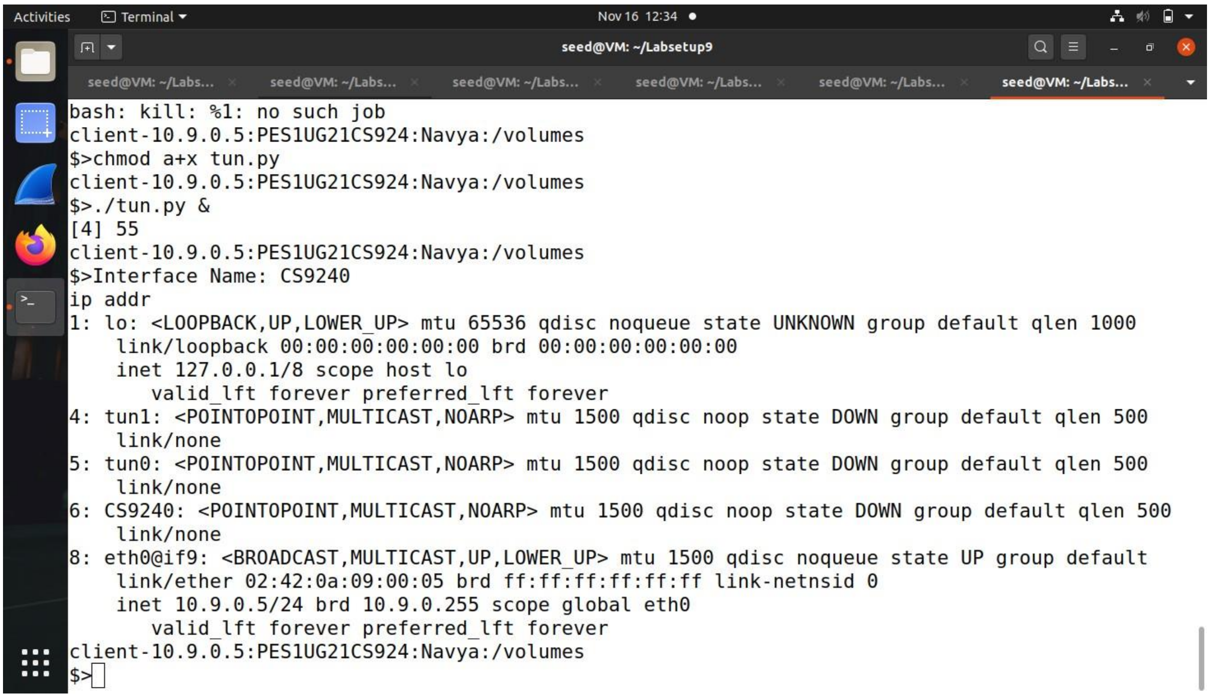


Task 2

Task 2a

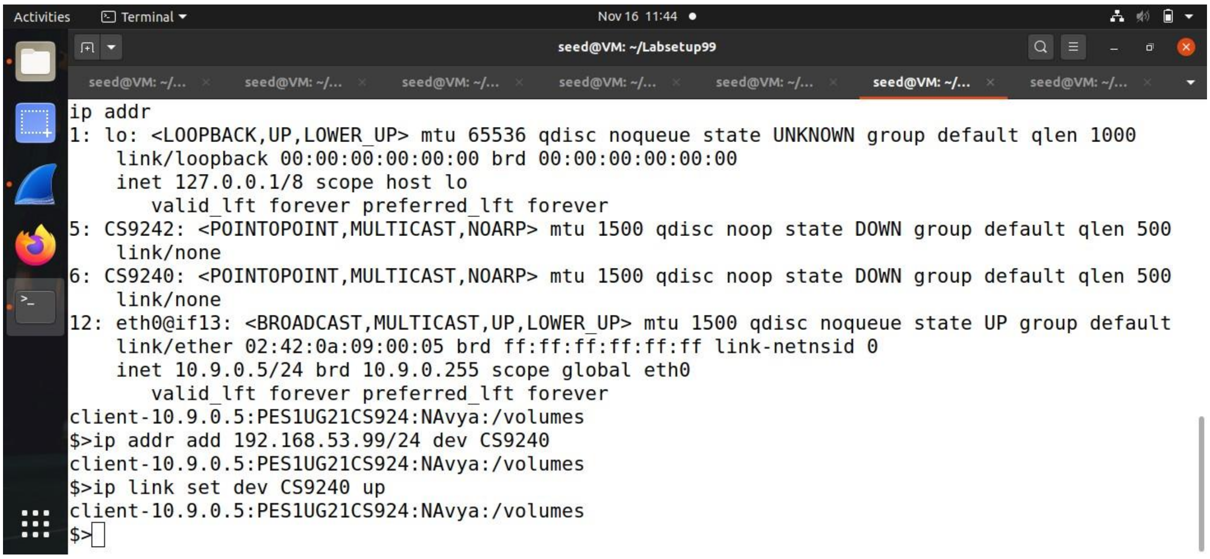


In this picture we observe the interface name to be tun0, based on the code provided in tun.py



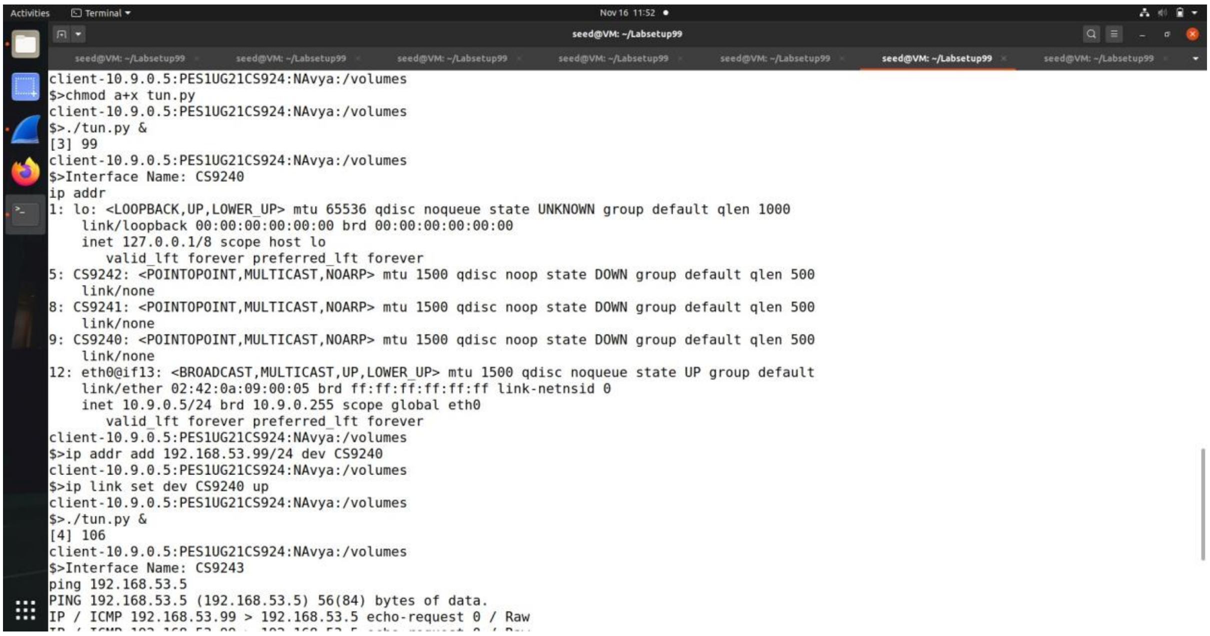
In the first picture, we observe the interface name to be of tun0. Here, however the interface name is displayed to be CS9240, which occurred after changing the code in tun.py.

Task 2.b

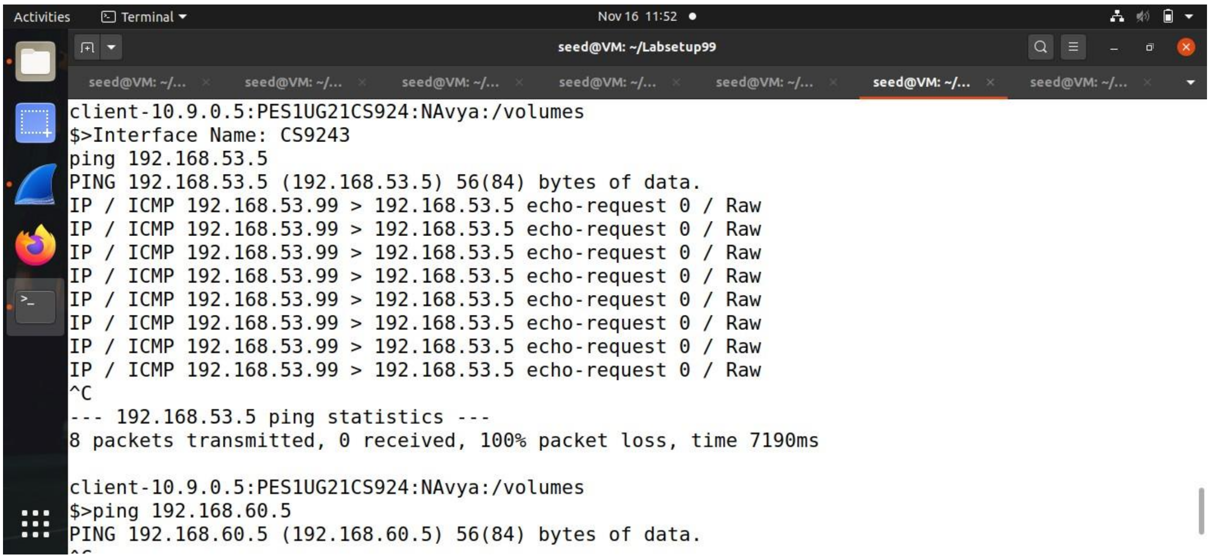


To be able to make the TUN interface usable we need to configure it. This can be done by assigning an IP address and then calling it up as shown above. This has been done to make the above interface usable. Since the interface is typically present in a down state, it has been called up to bring it to a usable state. Hence, an IP address is assigned to the interface CS9240 and is brought up to make it usable.

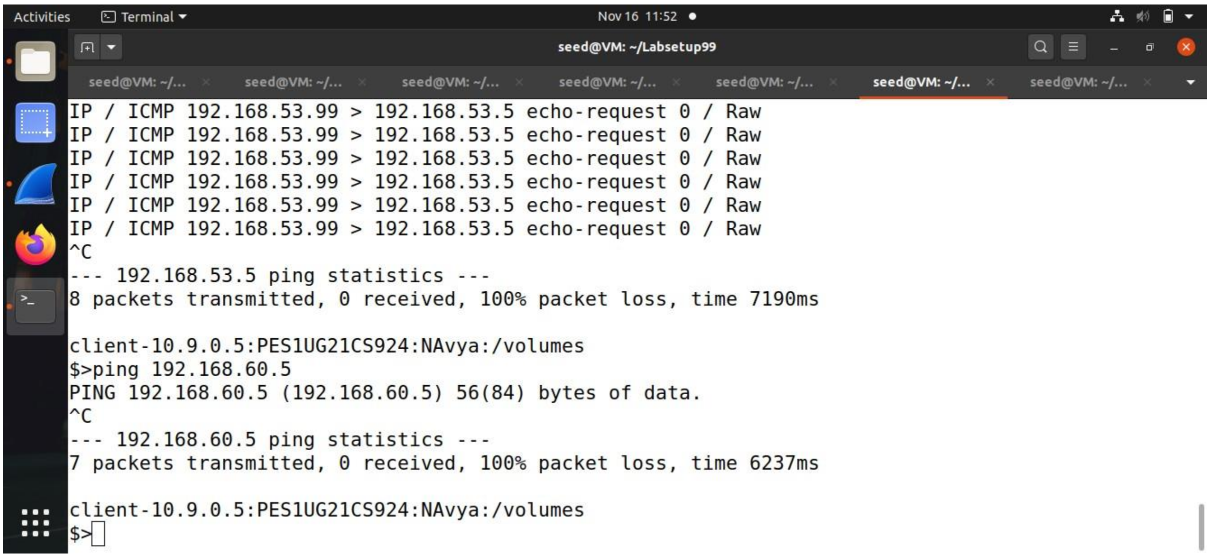
Task 2.c



Here, we first assign an IP address and call it up to make the interface usable.



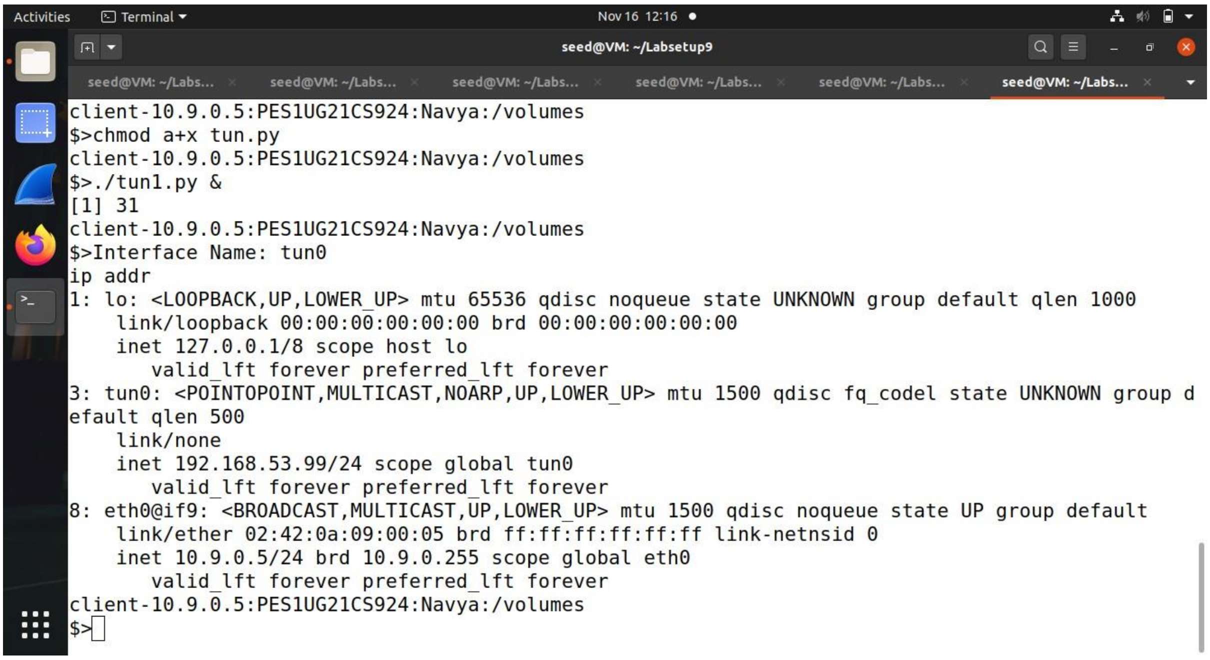
We observe that the first ping is successful and that the packets are being sent, considering that tun is configured in the same network as the ping, the network being 192.168..53.0/24.



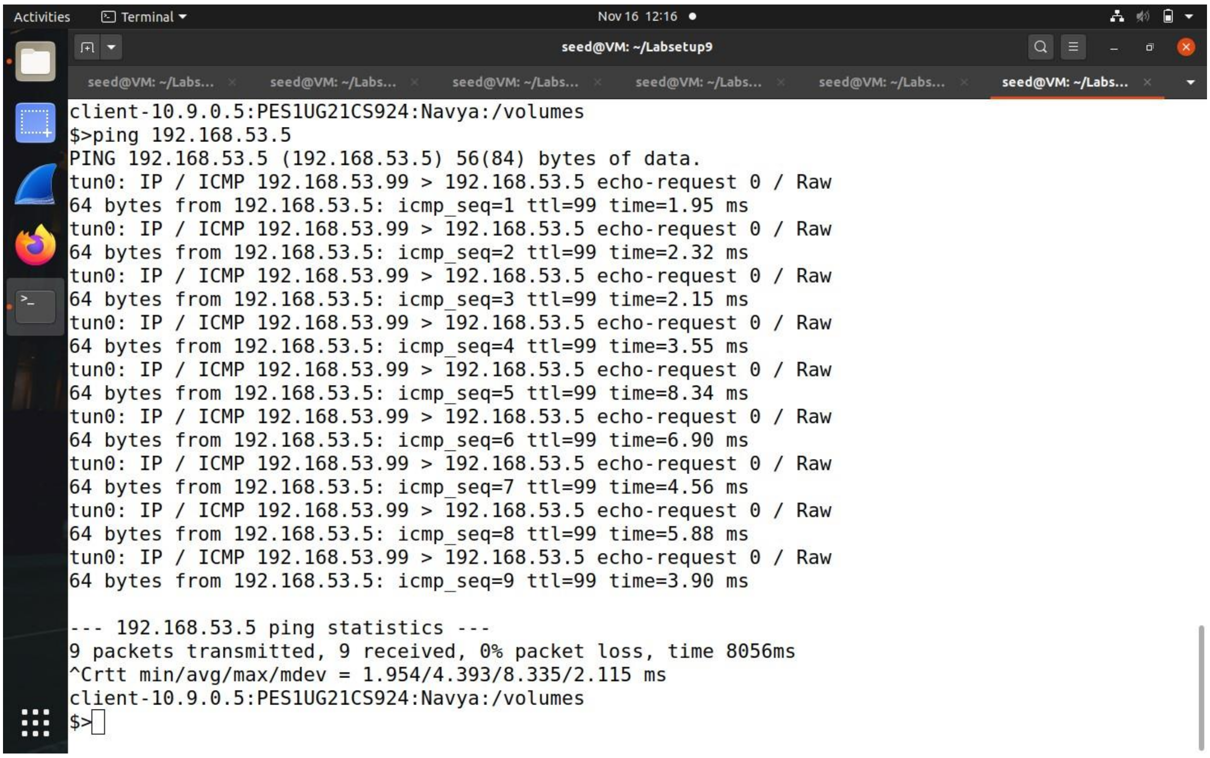
```
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
^C
--- 192.168.53.5 ping statistics ---
8 packets transmitted, 0 received, 100% packet loss, time 7190ms

client-10.9.0.5:PES1UG21CS924:NAvya:/volumes
$>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 ---
7 packets transmitted, 0 received, 100% packet loss, time 6237ms

client-10.9.0.5:PES1UG21CS924:NAvya:/volumes
$>
```

However, we find that the above ping fails as it isn't configured on the same network**.**

Task 2.d



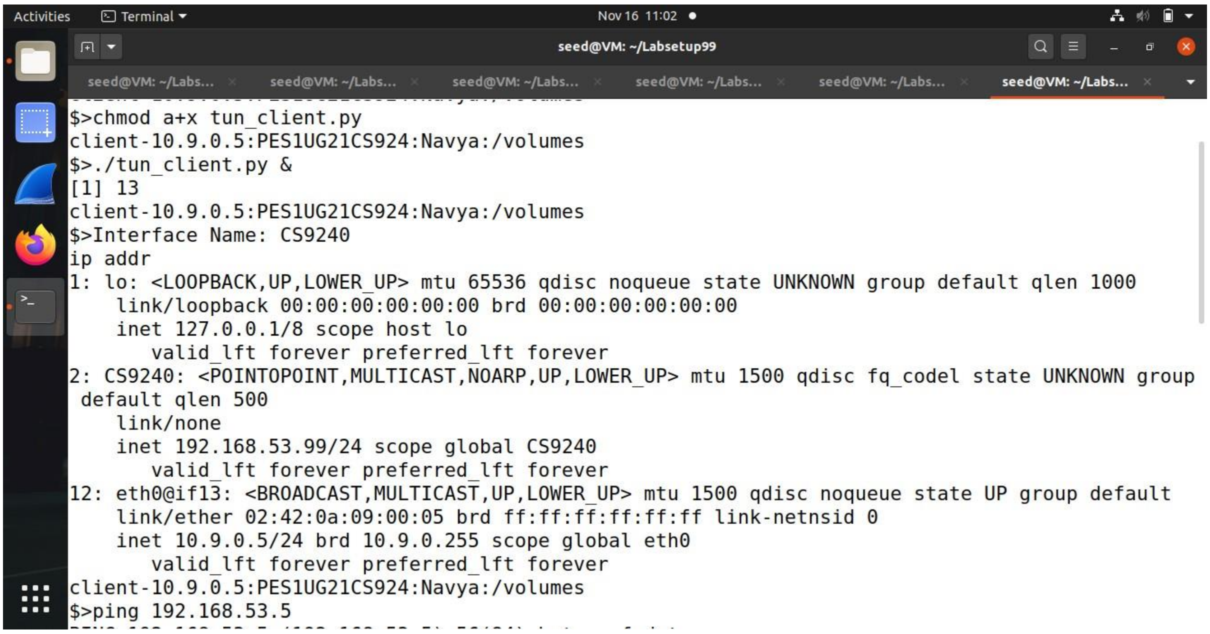
```
client-10.9.0.5:PES1UG21CS924:Navya:/volumes
$>chmod a+x tun.py
client-10.9.0.5:PES1UG21CS924:Navya:/volumes
$>./tun1.py &
[1] 31
client-10.9.0.5:PES1UG21CS924:Navya:/volumes
$>Interface Name: tun0
ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    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
3: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group d
efault qlen 500
    link/none
    inet 192.168.53.99/24 scope global tun0
       valid_lft forever preferred_lft forever
8: eth0@if9: <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
client-10.9.0.5:PES1UG21CS924:Navya:/volumes
$>
```

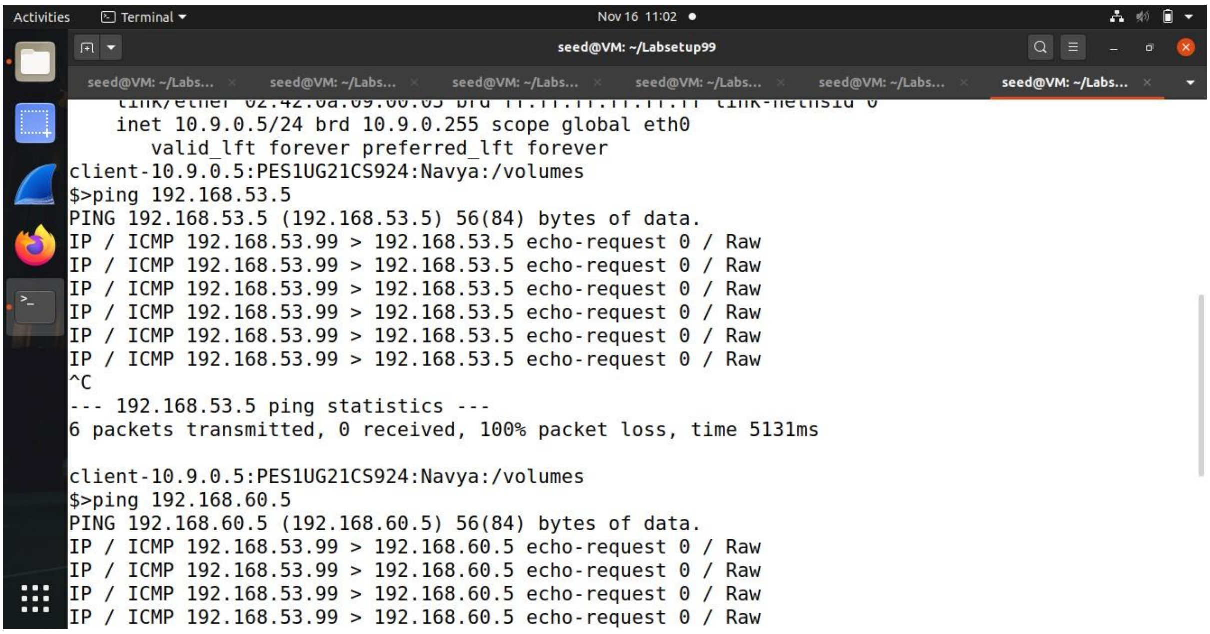


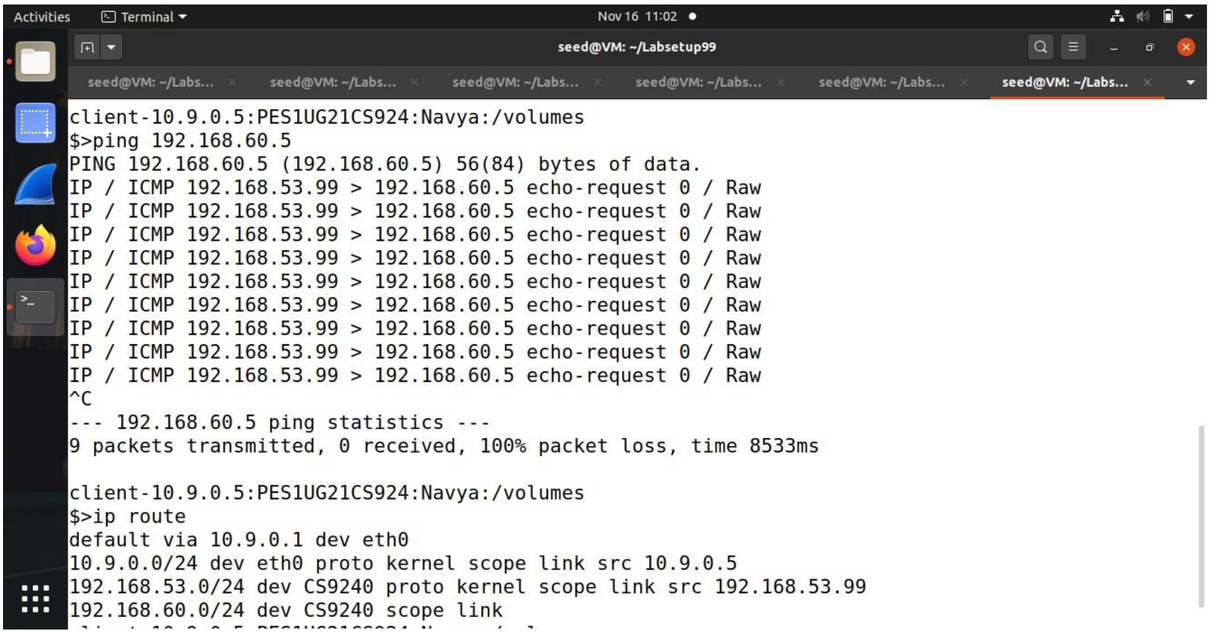
In this procedure, the ping is successful as it is able to print both the packets sent and received. The packets from the source and destination of the ICMP request are seen.

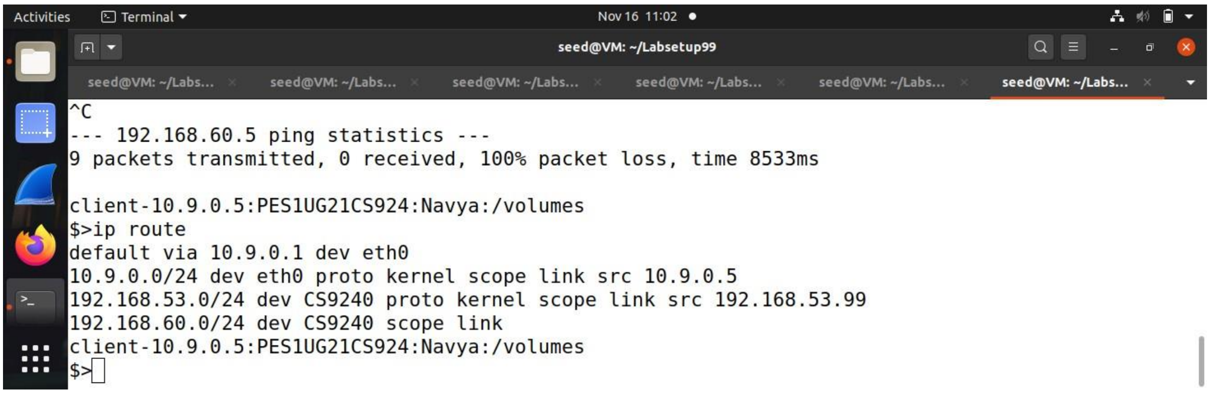
Task 3

On the client

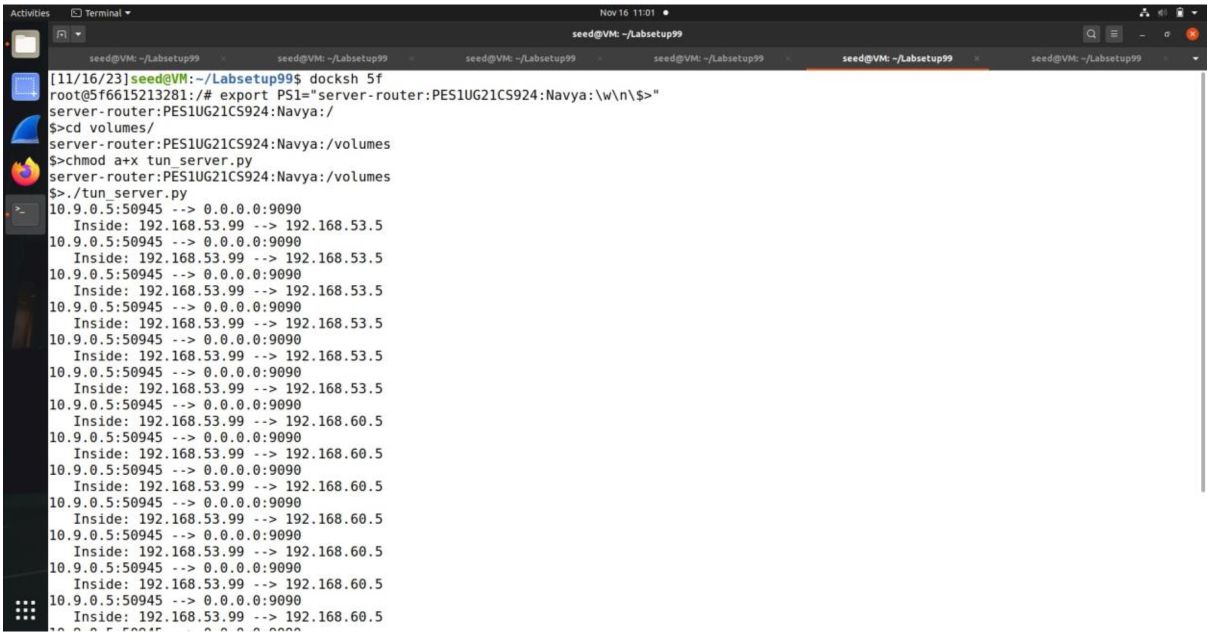






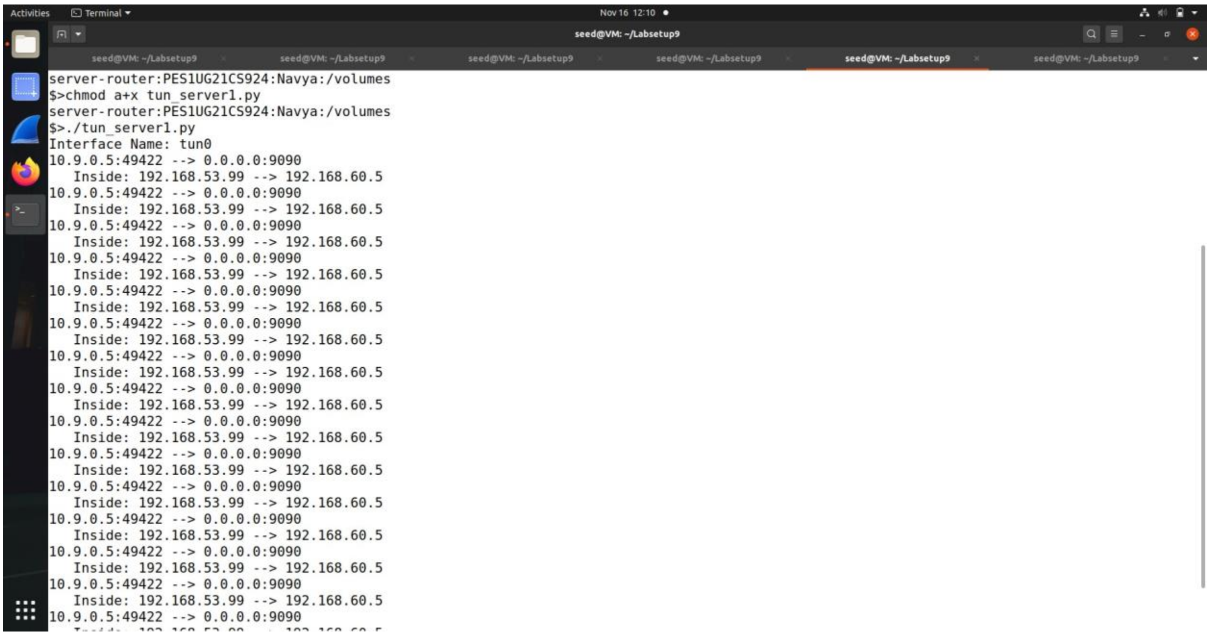


On the server



In the above task, the ping packets which are being sent to both 192.168.53.5 and 192.168.60.5 are encapsulated in UDP segments and are sent. These are then received at the server where, it extracts the ICMP packets and forwards it to the correct and desired IP.

Task 4



```
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
^C
--- 192.168.60.5 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5032ms

client-10.9.0.5:PES1UG21CS924:Navya:/volumes
$>ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
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
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
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
^C
--- 192.168.60.5 ping statistics ---
16 packets transmitted, 0 received, 100% packet loss, time 15339ms

client-10.9.0.5:PES1UG21CS924:Navya:/volumes
$>
```

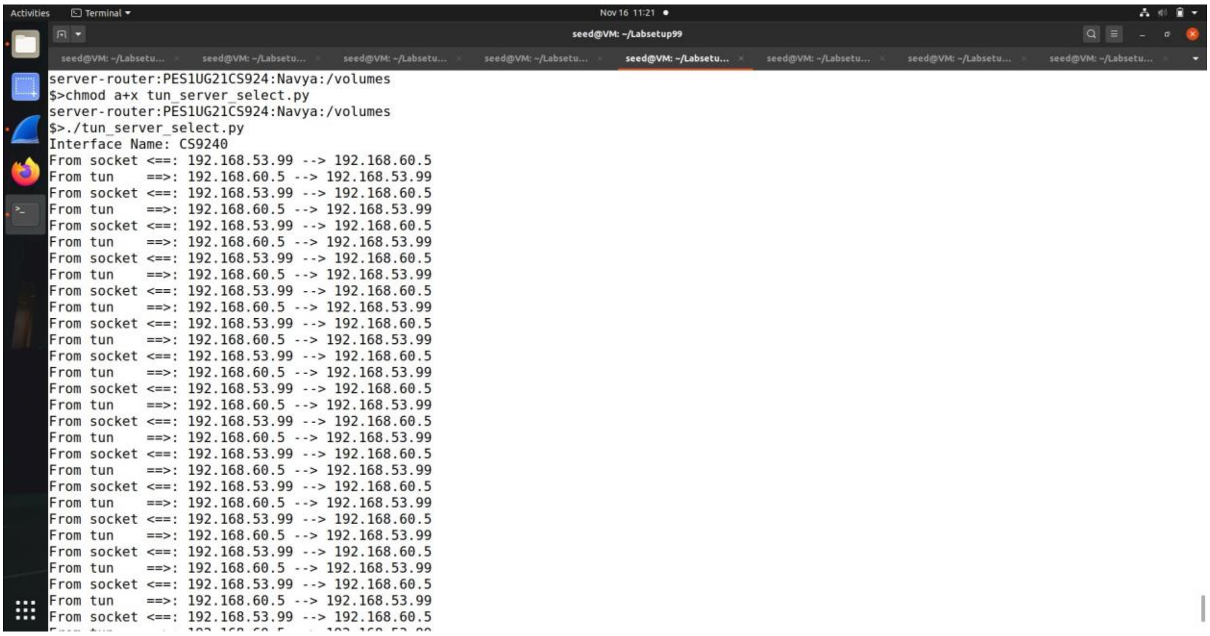
```
host-192.168.60.5:PES1UG21CS924:Navya:/
$>tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
-vv
17:10:25.746292 ARP, Request who-has 192.168.60.5 tell 192.168.60.11, length 28
17:10:25.746506 ARP, Reply 192.168.60.5 is-at 02:42:c0:a8:3c:05, length 28
17:10:25.746534 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 28, seq 1, length 64
17:10:25.746555 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 28, seq 1, length 64
17:10:26.764070 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 28, seq 2, length 64
17:10:26.764103 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 28, seq 2, length 64
17:10:27.773313 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 28, seq 3, length 64
17:10:27.773354 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 28, seq 3, length 64
17:10:28.793935 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 28, seq 4, length 64
17:10:28.794953 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 28, seq 4, length 64
17:10:29.822840 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 28, seq 5, length 64
17:10:29.822886 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 28, seq 5, length 64
17:10:30.807517 ARP, Request who-has 192.168.60.11 tell 192.168.60.5, length 28
17:10:30.807629 ARP, Reply 192.168.60.11 is-at 02:42:c0:a8:3c:0b, length 28
17:10:30.842037 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 28, seq 6, length 64
17:10:30.842060 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 28, seq 6, length 64
17:10:31.866914 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 28, seq 7, length 64
17:10:31.867047 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 28, seq 7, length 64
17:10:32.889305 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 28, seq 8, length 64
17:10:32.889542 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 28, seq 8, length 64
17:10:33.918073 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 28, seq 9, length 64
17:10:33.918426 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 28, seq 9, length 64
17:10:34.938015 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 28, seq 10, length 64
17:10:34.938043 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 28, seq 10, length 64
17:10:35.972893 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 28, seq 11, length 64
17:10:35.972929 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 28, seq 11, length 64
17:10:36.990061 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 28, seq 12, length 64
17:10:36.990096 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 28, seq 12, length 64
17:10:38.012481 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 28, seq 13, length 64
```

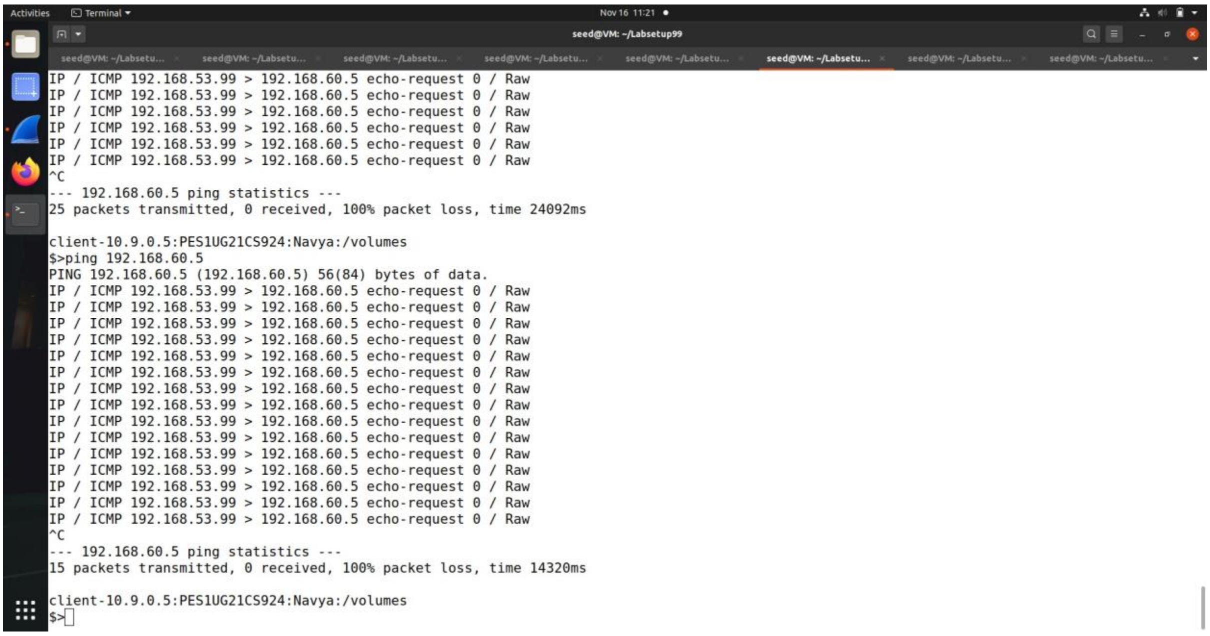
```
17:10:39.034697 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 28, seq 14, length 64
17:10:39.034760 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 28, seq 14, length 64
17:10:40.069555 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 28, seq 15, length 64
17:10:40.069825 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 28, seq 15, length 64
17:10:41.086310 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 28, seq 16, length 64
17:10:41.086350 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 28, seq 16, length 64
17:11:18.167911 IP6 fe80::8c0d:8bff:fea2:4a4d > ff02::2: ICMP6, router solicitation, length 16
17:12:41.805161 IP 192.168.60.1.5353 > 224.0.0.251.5353: 0 [2q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
17:12:43.896599 IP6 fe80::42:33ff:fe45:f0f3.5353 > ff02::fb.5353: 0 [2q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
17:12:47.561246 IP6 fe80::8c0d:8bff:fea2:4a4d.5353 > ff02::fb.5353: 0 [2q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
^C
40 packets captured
40 packets received by filter
0 packets dropped by kernel
host-192.168.60.5:PES1UG21CS924:Navya:/
$>
```

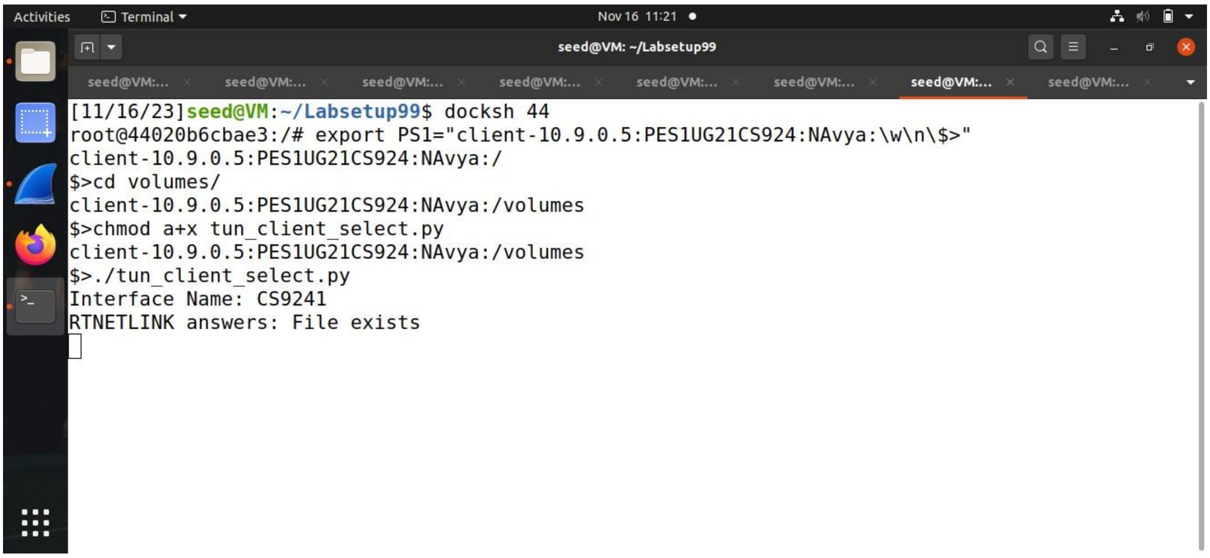
In the above task, we are able to ping from host U to host V, since the packets are passing through the tunnel. However, we can see the request part and not the reply part of the messages as it still hasn't been configured yet.

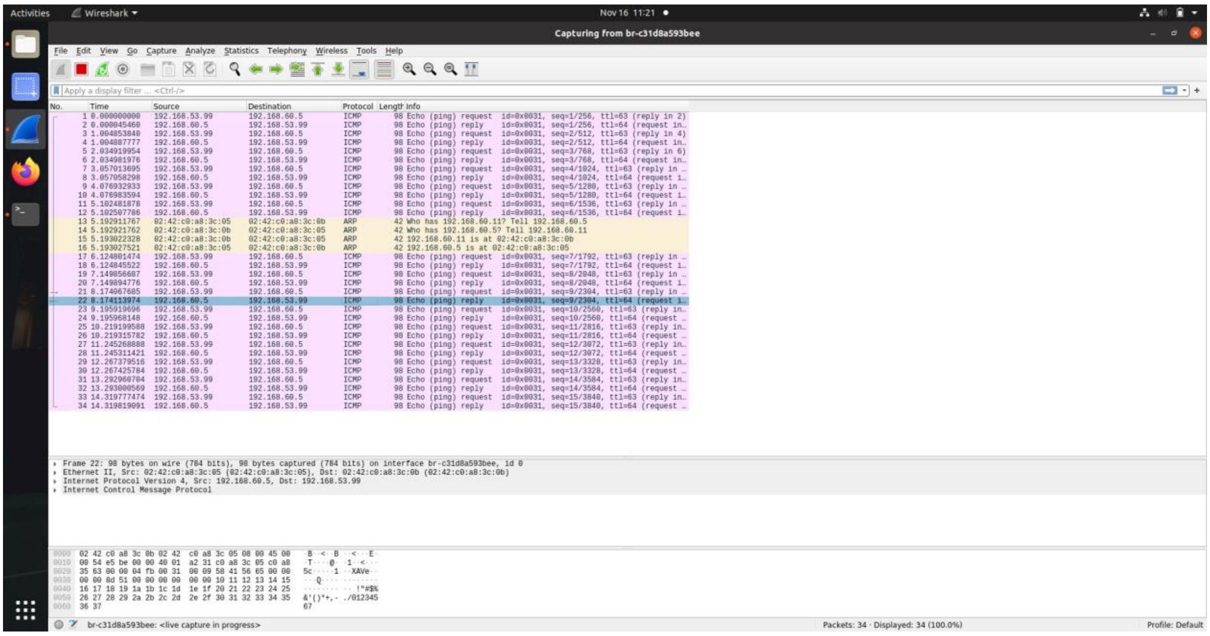
Task 5

In the previous task , we were only able to get the requests on the terminal and not the reply. Since the terminal was one directional. In this task we run the program on the client side to establish connection from both sides.

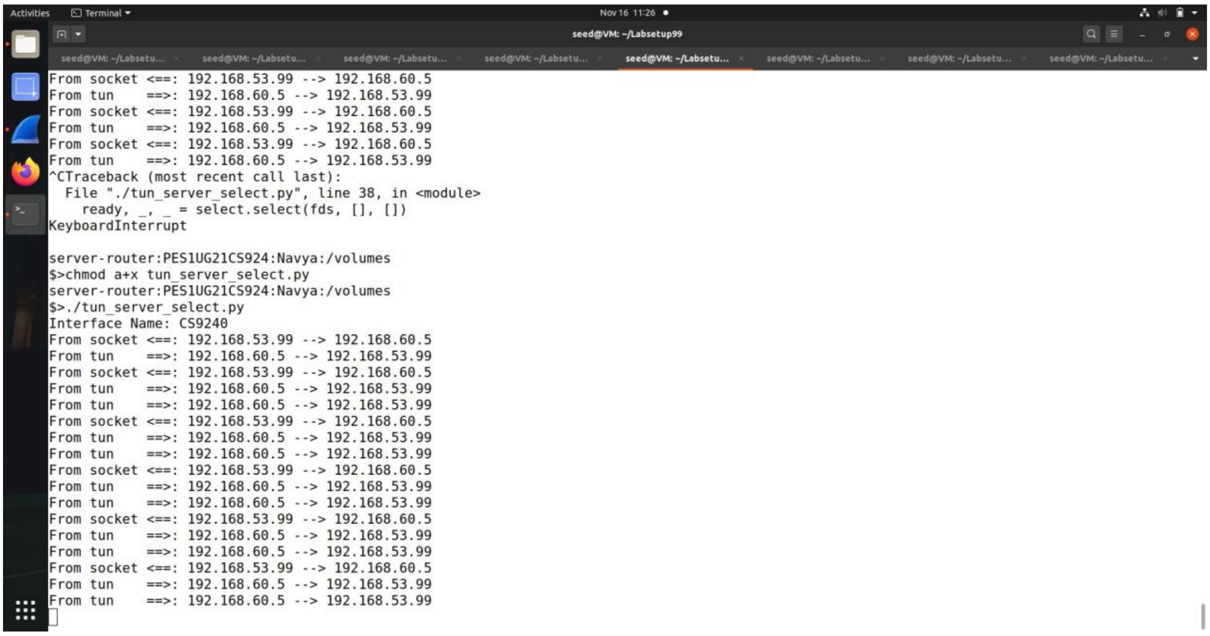




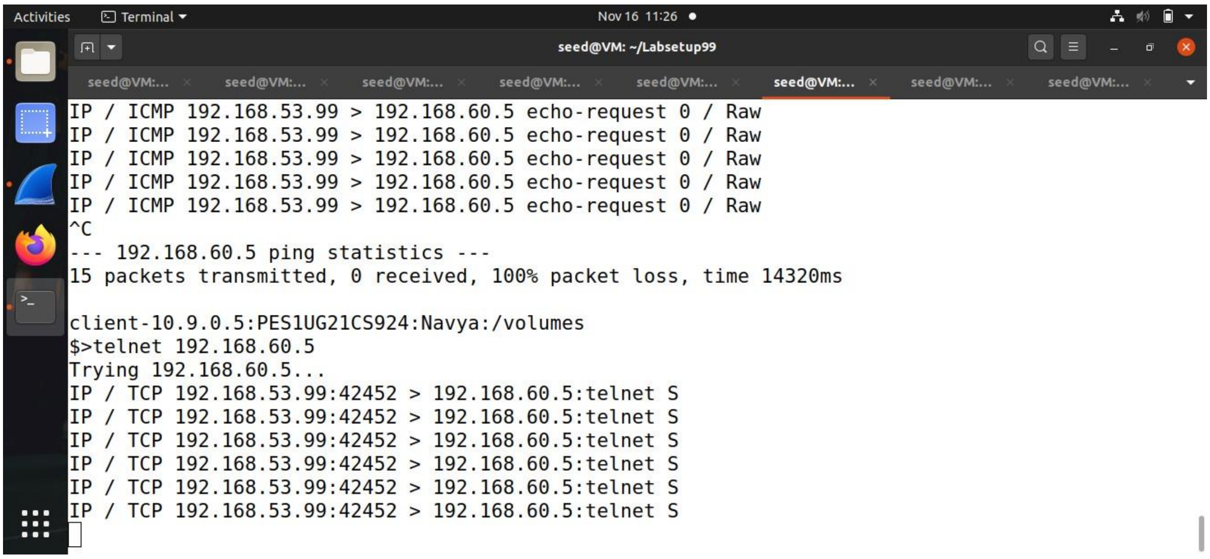




In the above wireshark image, we can observe the requests being sent by the host, client. Which shows that it is established in the correct manner.



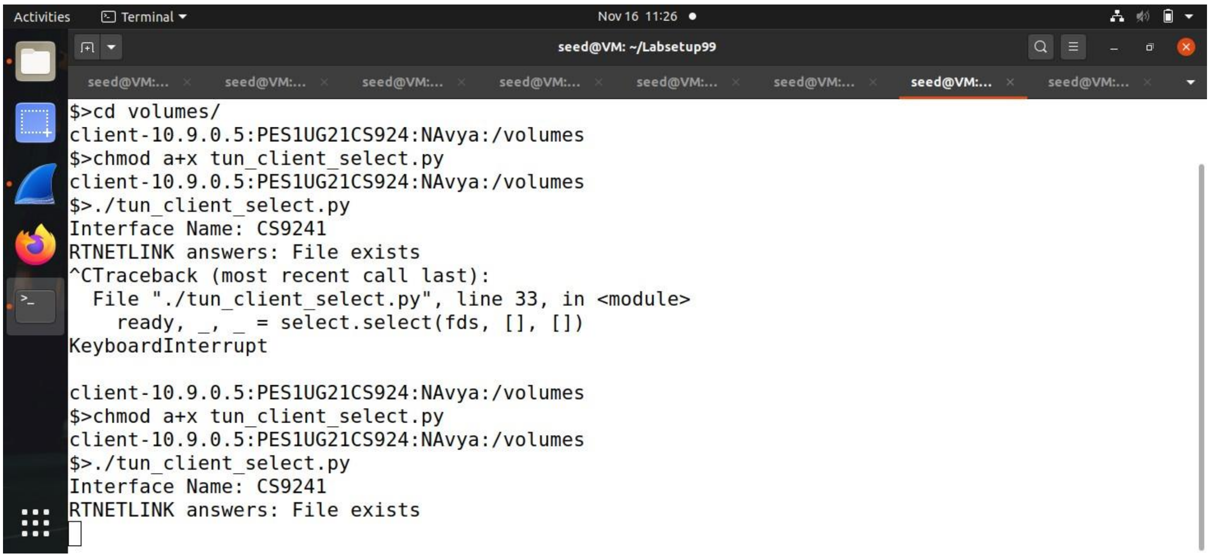
```
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun    ==>: 192.168.60.5 --> 192.168.53.99
^CTraceback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

server-router:PES1UG21CS924:Navya:/volumes
$>chmod a+x tun_server_select.py
server-router:PES1UG21CS924:Navya:/volumes
$>./tun_server_select.py
Interface Name: CS9240
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 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 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
```

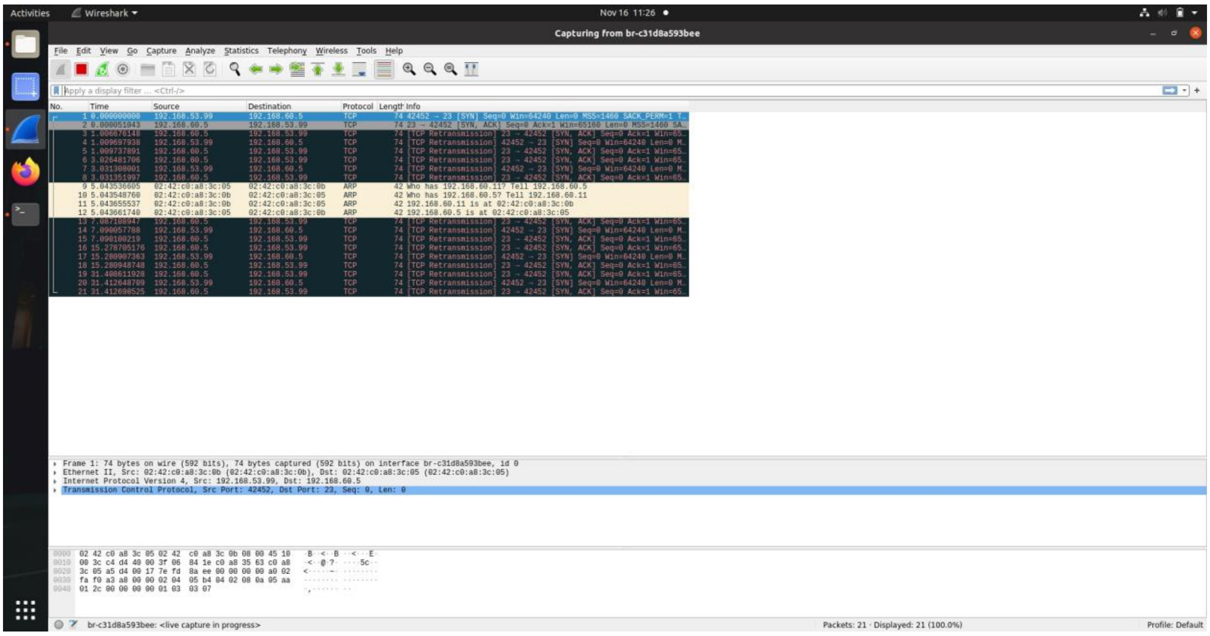


```
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
^C
--- 192.168.60.5 ping statistics ---
15 packets transmitted, 0 received, 100% packet loss, time 14320ms

client-10.9.0.5:PES1UG21CS924:Navya:/volumes
$>telnet 192.168.60.5
Trying 192.168.60.5...
IP / TCP 192.168.53.99:42452 > 192.168.60.5:telnet S
IP / TCP 192.168.53.99:42452 > 192.168.60.5:telnet S
IP / TCP 192.168.53.99:42452 > 192.168.60.5:telnet S
IP / TCP 192.168.53.99:42452 > 192.168.60.5:telnet S
IP / TCP 192.168.53.99:42452 > 192.168.60.5:telnet S
IP / TCP 192.168.53.99:42452 > 192.168.60.5:telnet S
```

We observe that from host U we are able to reach host V and vice versa through tunnelling.

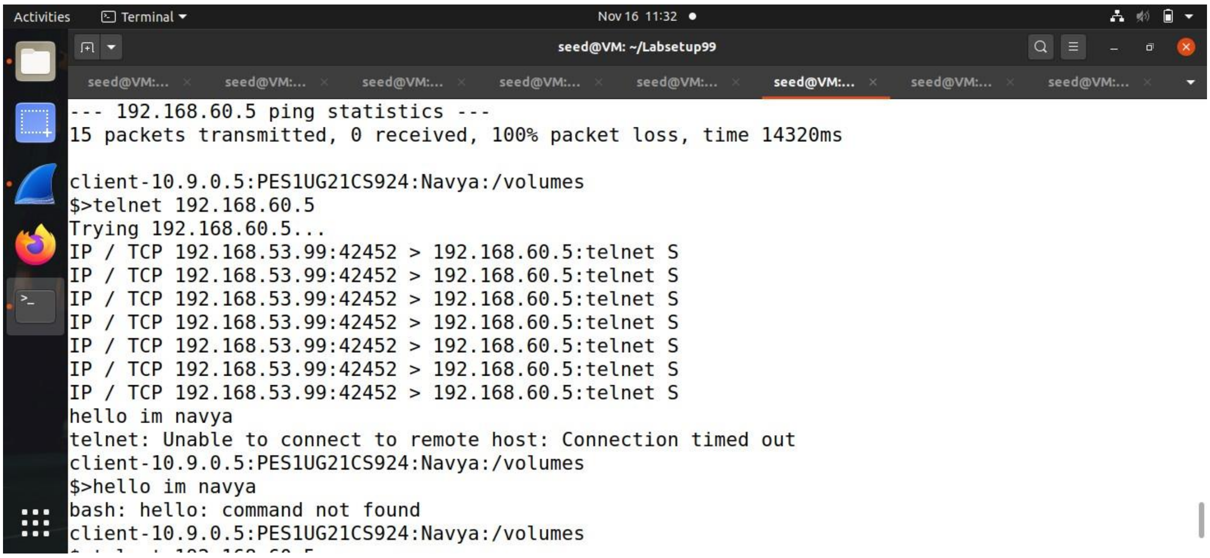
We observe that we are able to view both the reply and request messages from both sides, showing that the tunnel is two sided. We can get both the ping request and reply in the wireshark terminal.

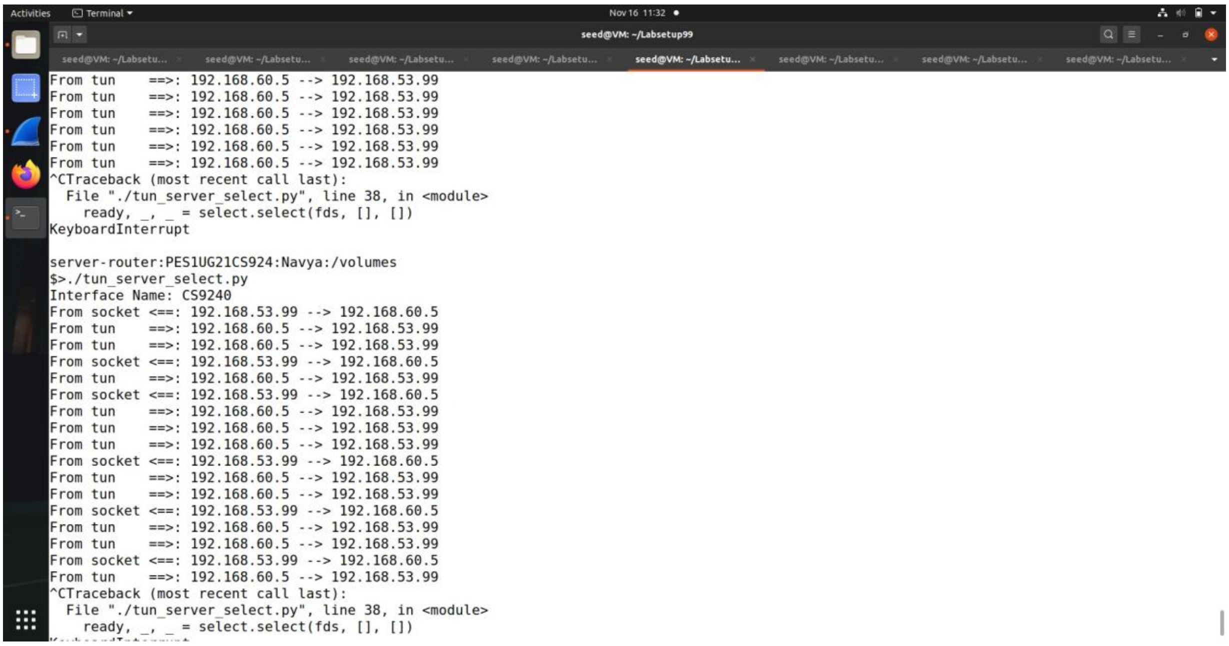




The output shows the sending and receiving ICMP echo requests and replies. Which shows that the tunnel is properly established.

Task 6





In this task, we have currently established a telnet connection from host U to host V. However, when the VPN connection breaks, the telnet connection also breaks as shown above. With messages and requests no longer being sent. Hence, in spite of typing a message nothing is viewed on the other host. After re-establishing the connection, we observe that even though it takes some time and does not appear immediately, we can see that the message is sent and that the requests are sent. Which shows that the connection between the two hosts has been established.