VPN TUNNELLING

UFCFVN-30-M - Computer & Network Security

A Research Coursework

Prepared by

Kiran Kumar Mohanraj - 20054954

MSc Cyber Security

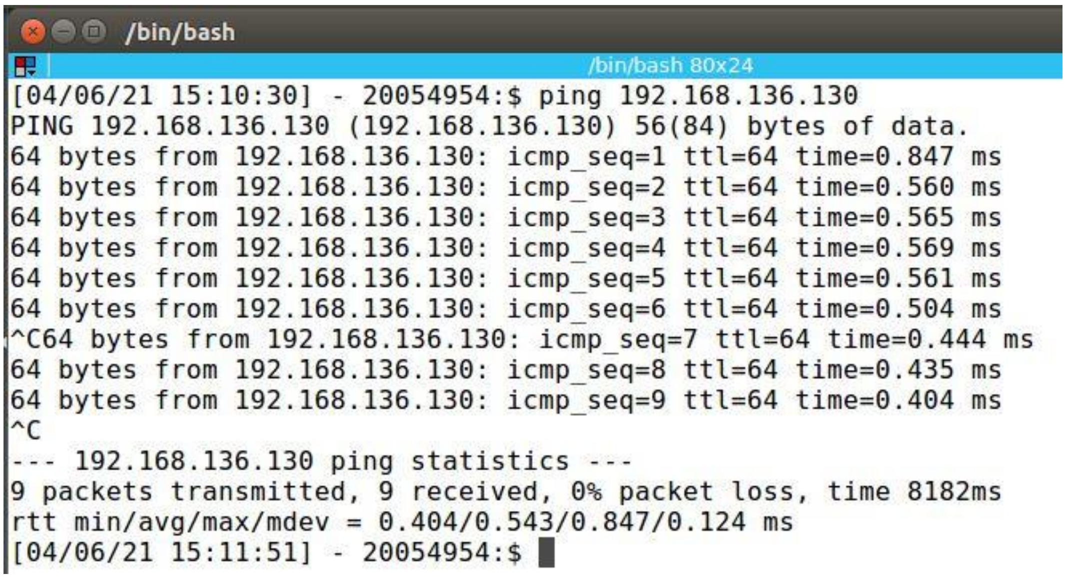
University of the West of England, Bristol

TABLE OF CONTENTS

1. LAB TASK

1.1 Task 1: Testing the Configuration

a) Testing connection between VPN Client and VPN Server over NAT network:

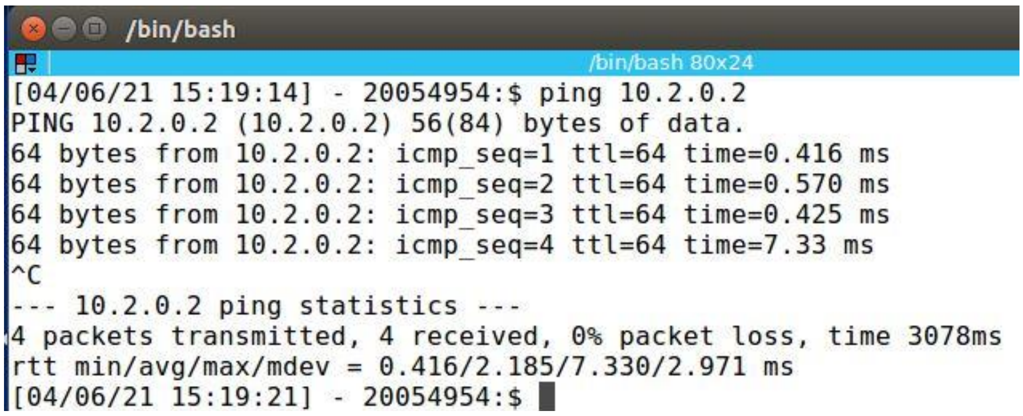


```
[04/06/21 15:10:30] - 20054954:$ ping 192.168.136.130
PING 192.168.136.130 (192.168.136.130) 56(84) bytes of data.
64 bytes from 192.168.136.130: icmp_seq=1 ttl=64 time=0.847 ms
64 bytes from 192.168.136.130: icmp_seq=2 ttl=64 time=0.560 ms
64 bytes from 192.168.136.130: icmp_seq=3 ttl=64 time=0.565 ms
64 bytes from 192.168.136.130: icmp_seq=4 ttl=64 time=0.569 ms
64 bytes from 192.168.136.130: icmp_seq=5 ttl=64 time=0.561 ms
64 bytes from 192.168.136.130: icmp_seq=6 ttl=64 time=0.504 ms
^C64 bytes from 192.168.136.130: icmp_seq=7 ttl=64 time=0.444 ms
64 bytes from 192.168.136.130: icmp_seq=8 ttl=64 time=0.435 ms
64 bytes from 192.168.136.130: icmp_seq=9 ttl=64 time=0.404 ms
^C
--- 192.168.136.130 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8182ms
rtt min/avg/max/mdev = 0.404/0.543/0.847/0.124 ms
[04/06/21 15:11:51] - 20054954:$
```

Figure 1 VPN Client ping VPN Server over NAT network

Figure 1 shows that the VPN client can connect with the VPN Server over NAT network, 9 packets has been sent from the VPN Client to VPN Server and all these were received without any packet loss. The 192.168.136.130 is the IP address of VPN Server's NAT network.

b) Testing connection between VPN Server and Host V over private network:

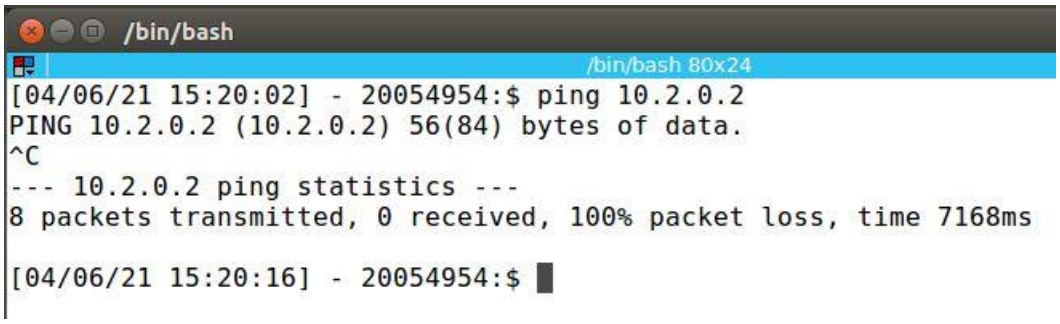


```
[04/06/21 15:19:14] - 20054954:$ ping 10.2.0.2
PING 10.2.0.2 (10.2.0.2) 56(84) bytes of data.
64 bytes from 10.2.0.2: icmp_seq=1 ttl=64 time=0.416 ms
64 bytes from 10.2.0.2: icmp_seq=2 ttl=64 time=0.570 ms
64 bytes from 10.2.0.2: icmp_seq=3 ttl=64 time=0.425 ms
64 bytes from 10.2.0.2: icmp_seq=4 ttl=64 time=7.33 ms
^C
--- 10.2.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3078ms
rtt min/avg/max/mdev = 0.416/2.185/7.330/2.971 ms
[04/06/21 15:19:21] - 20054954:$
```

Figure 2 VPN Server pinging Host V over private network

Figure 2 shows that the VPN Server can connect with the Host V over private network, 4 packets has been sent from the VPN Server to Host V and all these were received without any packet loss. The 10.2.0.2 is the IP address of Host V and the gateway is VPN Server whose private network IP address is 10.2.0.1.

c) Testing connection between VPN Client and Host V:



```
[04/06/21 15:20:02] - 20054954:$ ping 10.2.0.2
PING 10.2.0.2 (10.2.0.2) 56(84) bytes of data.
^C
--- 10.2.0.2 ping statistics ---
8 packets transmitted, 0 received, 100% packet loss, time 7168ms

[04/06/21 15:20:16] - 20054954:$
```

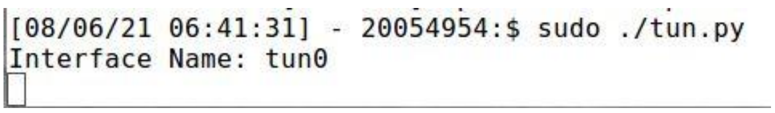
Figure 3 VPN Client cannot ping Host V

From Figure 3, VPN Client is unable to connect to Host V because Host V is in a private network and only VPN Server can connect to it. For VPN Client to connect with Host V, we need to connect through VPN Server.

1.2 Task 2: Create and Configure the TUN interface

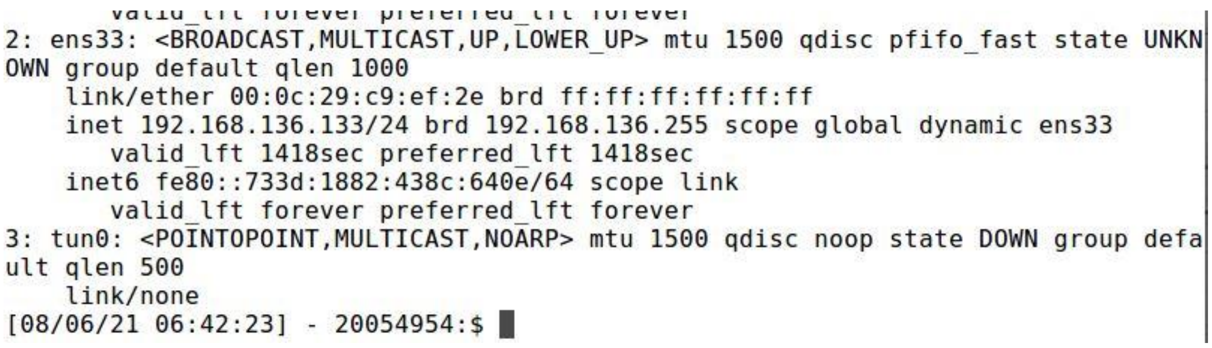
1.2.1 Name the TUN interface:

For this task, the provided python program was downloaded and executed as shown in Figure 4. Once the program was executed a new interface named tun0 was created. Figure 5 shows the list of interfaces which include tun0.



```
[08/06/21 06:41:31] - 20054954:$ sudo ./tun.py
Interface Name: tun0
```

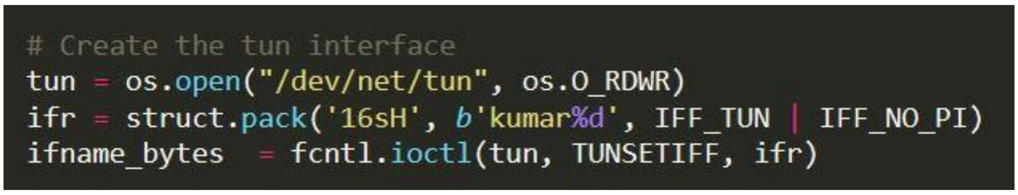
Figure 4 Execution of provided python code for creating interface



```
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKN
OWN group default qlen 1000
    link/ether 00:0c:29:c9:ef:2e brd ff:ff:ff:ff:ff:ff
    inet 192.168.136.133/24 brd 192.168.136.255 scope global dynamic ens33
       valid_lft 1418sec preferred_lft 1418sec
    inet6 fe80::733d:1882:438c:640e/64 scope link
       valid_lft forever preferred_lft forever
3: tun0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group defa
ult qlen 500
    link/none
[08/06/21 06:42:23] - 20054954:$
```

Figure 5 List of Network interfaces

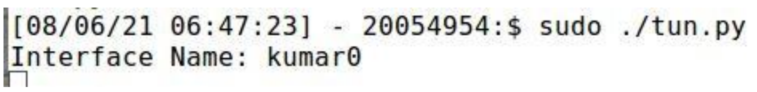
This task requires us to change the name of the interface to my last name (kumar). For this, the python code was changed from "tun%d" to "kumar%d". The modified part of the python code is shown in Figure 6.



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

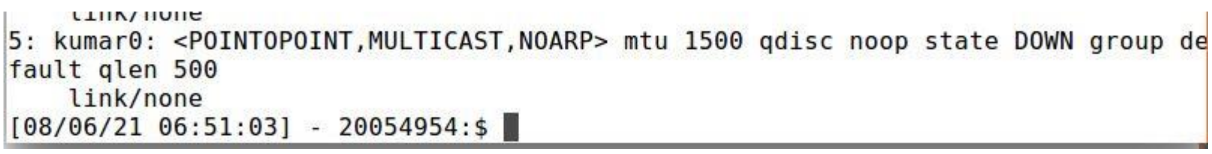
Figure 6 Modified python code with kumar as interface name

When this modified code was executed, the interface with kumar as prefix was created. Figure 7 and 8 shows the results of the execution.



```
[08/06/21 06:47:23] - 20054954:$ sudo ./tun.py
Interface Name: kumar0
```

Figure 7 Execution of modified python code

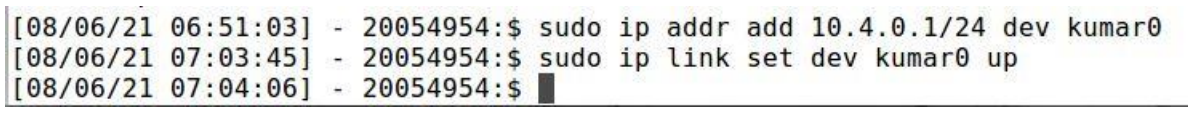


```
5: kumar0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group de
fault qlen 500
    link/none
[08/06/21 06:51:03] - 20054954:$
```

Figure 8 New network interface with kumar as prefix

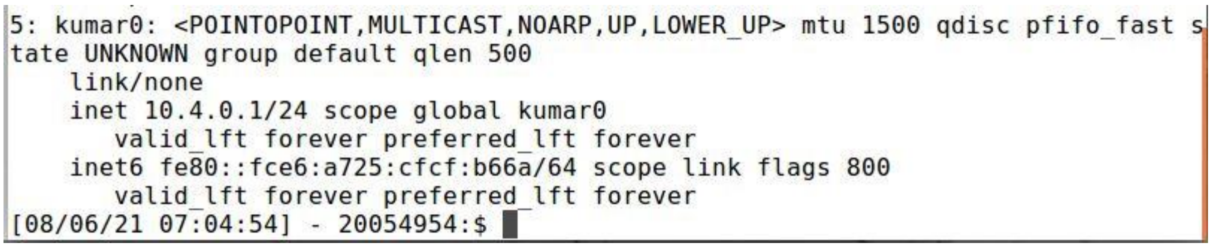
1.2.2 Setup the TUN interface:

Figure 9 shows the commands used to assign IP address to kumar0 interface and to bring up the interface. When we execute the command "ip addr", we could see that the provided IP address has been assigned to the kumar0 interface. Figure 10 shows the results of this.



```
[08/06/21 06:51:03] - 20054954:$ sudo ip addr add 10.4.0.1/24 dev kumar0
[08/06/21 07:03:45] - 20054954:$ sudo ip link set dev kumar0 up
[08/06/21 07:04:06] - 20054954:$ 
```

Figure 9 Commands for assigning IP address and bringing up the interface

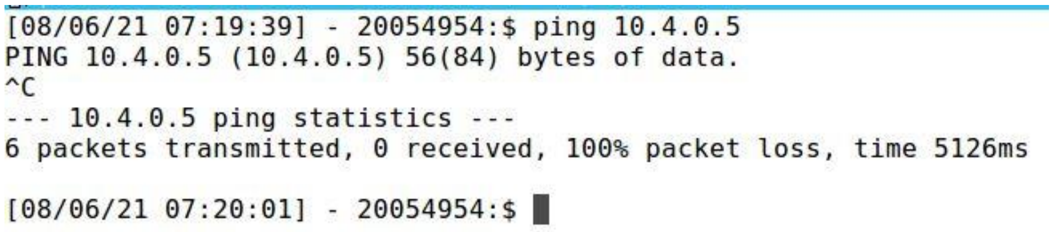


```
5: kumar0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast s
tate UNKNOWN group default qlen 500
    link/none
    inet 10.4.0.1/24 scope global kumar0
       valid_lft forever preferred_lft forever
    inet6 fe80::fce6:a725:cfcf:b66a/64 scope link flags 800
       valid_lft forever preferred_lft forever
[08/06/21 07:04:54] - 20054954:$ 
```

Figure 10 kumar0 interface with the assigned IP

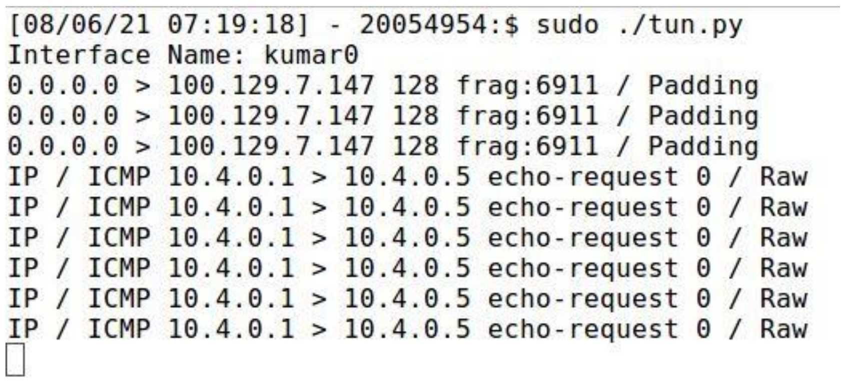
1.2.3 Read from the TUN interface:

Before conducting the task, the provided code snippet was added to the tun.py program and was executed in VPN Client. When we try to ping any host in the same network (10.4.0.5), we couldn't get any reply, all the packets were lost. This is because no other host was present in the network. Thus, no reply was received as shown in Figure 11. Figure 12 shows that the python program prints details of the packet sent by VPN Client which include source and destination IP address, type of request and the data.



```
[08/06/21 07:19:39] - 20054954:$ ping 10.4.0.5
PING 10.4.0.5 (10.4.0.5) 56(84) bytes of data.
^C
--- 10.4.0.5 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5126ms

[08/06/21 07:20:01] - 20054954:$ 
```

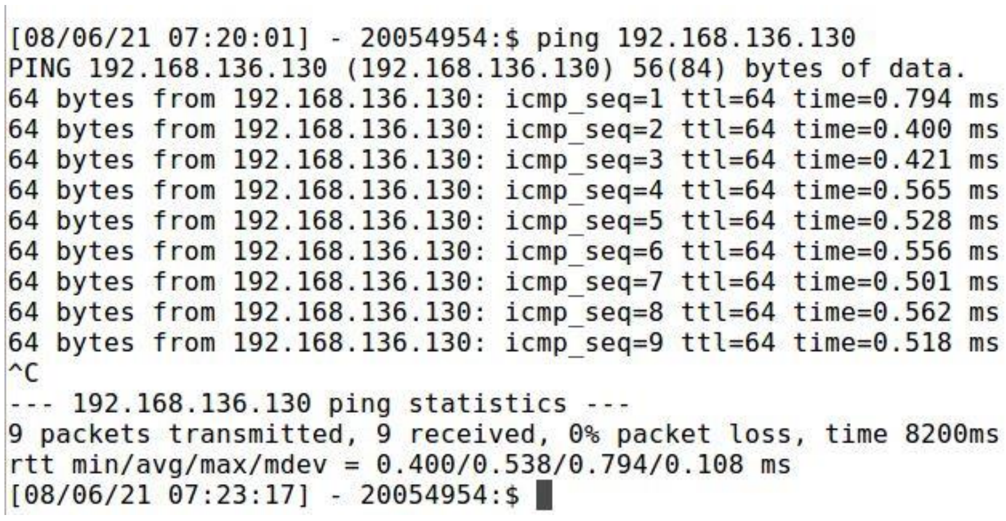
Figure 11 Pinging another host in the same network



```
[08/06/21 07:19:18] - 20054954:$ sudo ./tun.py
Interface Name: kumar0
0.0.0.0 > 100.129.7.147 128 frag:6911 / Padding
0.0.0.0 > 100.129.7.147 128 frag:6911 / Padding
0.0.0.0 > 100.129.7.147 128 frag:6911 / Padding
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
```

Figure 12 Output of tun.py program

When we try to ping the VPN Server which is on the NAT network, we could get a response. This is because the tunnel has been setup and the packets were able to reach the Server (192.168.136.130). Thus, we get a response as shown in Figure 13.

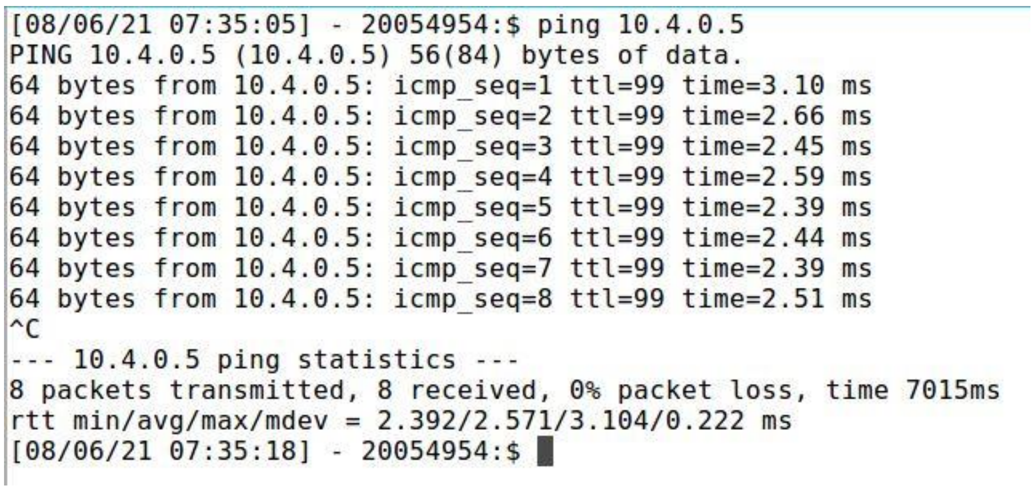


```
[08/06/21 07:20:01] - 20054954:$ ping 192.168.136.130
PING 192.168.136.130 (192.168.136.130) 56(84) bytes of data.
64 bytes from 192.168.136.130: icmp_seq=1 ttl=64 time=0.794 ms
64 bytes from 192.168.136.130: icmp_seq=2 ttl=64 time=0.400 ms
64 bytes from 192.168.136.130: icmp_seq=3 ttl=64 time=0.421 ms
64 bytes from 192.168.136.130: icmp_seq=4 ttl=64 time=0.565 ms
64 bytes from 192.168.136.130: icmp_seq=5 ttl=64 time=0.528 ms
64 bytes from 192.168.136.130: icmp_seq=6 ttl=64 time=0.556 ms
64 bytes from 192.168.136.130: icmp_seq=7 ttl=64 time=0.501 ms
64 bytes from 192.168.136.130: icmp_seq=8 ttl=64 time=0.562 ms
64 bytes from 192.168.136.130: icmp_seq=9 ttl=64 time=0.518 ms
^C
--- 192.168.136.130 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8200ms
rtt min/avg/max/mdev = 0.400/0.538/0.794/0.108 ms
[08/06/21 07:23:17] - 20054954:$
```

Figure 13 Pinging VPN Server on the NAT network

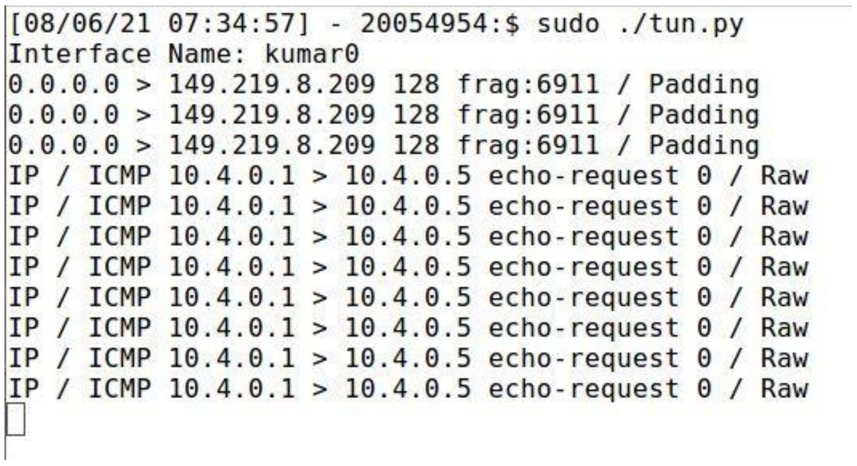
1.2.4 Write to the TUN interface

The code snippet provided in the lab sheet was added to tun.py program. This program will spoof the ICMP echo-request to create a ICMP echo-reply which is sent back to the VPN Client. The program was executed as shown in Figure 15. When we try to ping the previous host which was unreachable (10.2.0.5), this time we could get a response. This is because of the python program which creates a spoofed packet to provide a response. This illustration is shown in Figure 14.



```
[08/06/21 07:35:05] - 20054954:$ ping 10.4.0.5
PING 10.4.0.5 (10.4.0.5) 56(84) bytes of data.
64 bytes from 10.4.0.5: icmp_seq=1 ttl=99 time=3.10 ms
64 bytes from 10.4.0.5: icmp_seq=2 ttl=99 time=2.66 ms
64 bytes from 10.4.0.5: icmp_seq=3 ttl=99 time=2.45 ms
64 bytes from 10.4.0.5: icmp_seq=4 ttl=99 time=2.59 ms
64 bytes from 10.4.0.5: icmp_seq=5 ttl=99 time=2.39 ms
64 bytes from 10.4.0.5: icmp_seq=6 ttl=99 time=2.44 ms
64 bytes from 10.4.0.5: icmp_seq=7 ttl=99 time=2.39 ms
64 bytes from 10.4.0.5: icmp_seq=8 ttl=99 time=2.51 ms
^C
--- 10.4.0.5 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7015ms
rtt min/avg/max/mdev = 2.392/2.571/3.104/0.222 ms
[08/06/21 07:35:18] - 20054954:$
```

Figure 14 Pinging a host in the same network

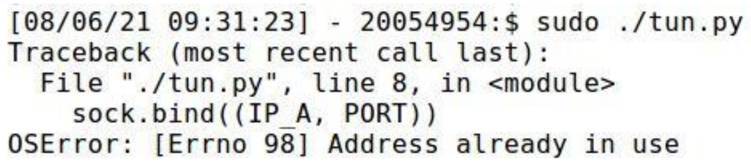


```
[08/06/21 07:34:57] - 20054954:$ sudo ./tun.py
Interface Name: kumar0
0.0.0.0 > 149.219.8.209 128 frag:6911 / Padding
0.0.0.0 > 149.219.8.209 128 frag:6911 / Padding
0.0.0.0 > 149.219.8.209 128 frag:6911 / Padding
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
```

Figure 15 Output of the tun.py program

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

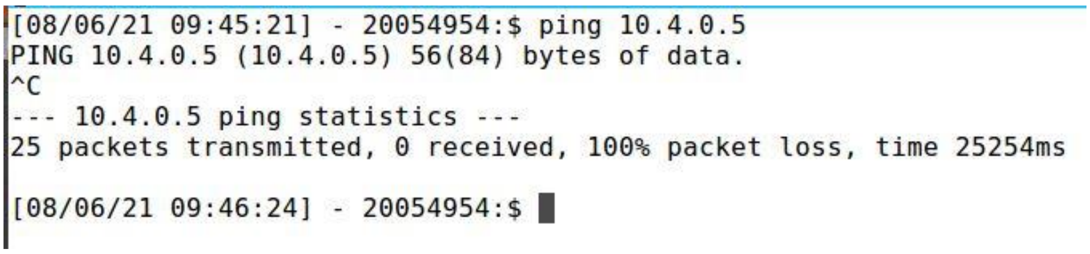
The provided VPN Server and VPN Client program was compiled and executed in their respective VMs. Initially, the Server program produced an error as shown in Figure 16. The error was because of port 9090 which was already in use. To overcome this error, I assigned port 9080 for listening all request.



```
[08/06/21 09:31:23] - 20054954:$ sudo ./tun.py
Traceback (most recent call last):
  File "./tun.py", line 8, in <module>
    sock.bind((IP_A, PORT))
OSError: [Errno 98] Address already in use
```

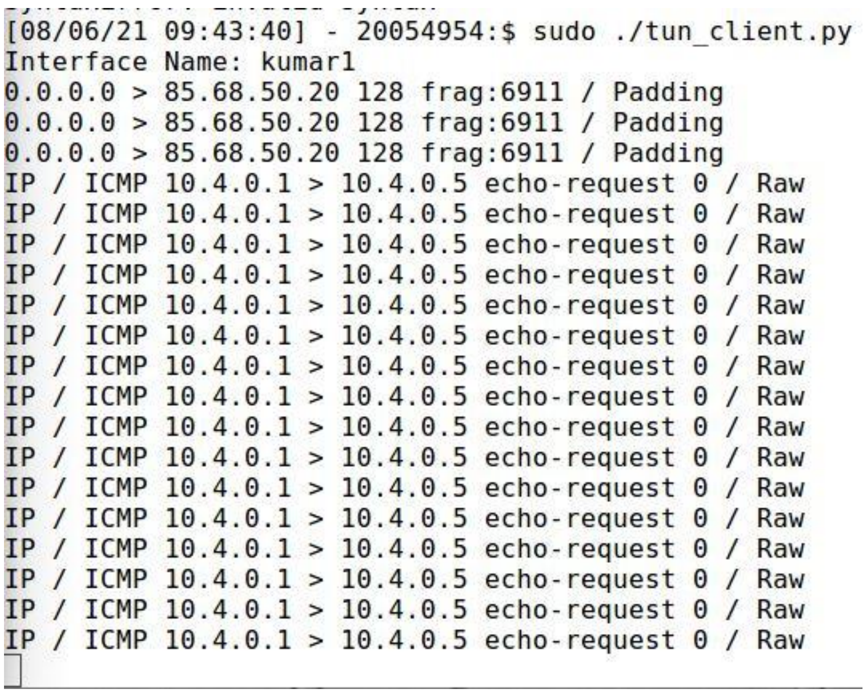
Figure 16 Port already in use error

Once the setup was done, I tried pinging the host (10.4.0.5) which is on the same network (as shown in Figure 17). This time we couldn’t get any reply because we remove our code for spoofing a reply and there were no host to reply back. Similar to the previous results we get details of packet sent by the VPN Client as output of the tun_client.py program (Figure 18).



```
[08/06/21 09:45:21] - 20054954:$ ping 10.4.0.5
PING 10.4.0.5 (10.4.0.5) 56(84) bytes of data.
^C
--- 10.4.0.5 ping statistics ---
25 packets transmitted, 0 received, 100% packet loss, time 25254ms

[08/06/21 09:46:24] - 20054954:$
```

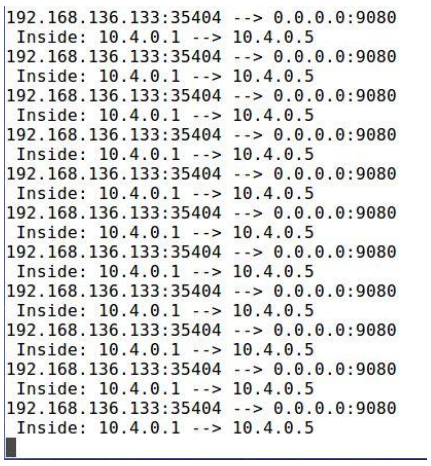
Figure 17 Pinging host in the same network



```
[08/06/21 09:43:40] - 20054954:$ sudo ./tun_client.py
Interface Name: kumar1
0.0.0.0 > 85.68.50.20 128 frag:6911 / Padding
0.0.0.0 > 85.68.50.20 128 frag:6911 / Padding
0.0.0.0 > 85.68.50.20 128 frag:6911 / Padding
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.4.0.5 echo-request 0 / Raw
```

Figure 18 Output of tun_client.py program

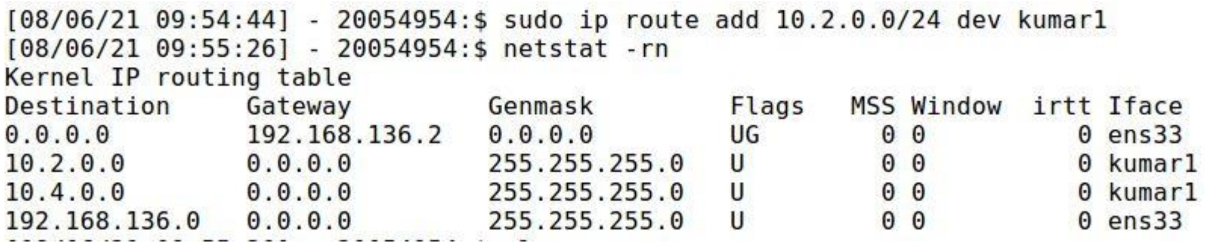
In VPN Server, we could see details of the packet sent by VPN Client as output of tun_server.py program. This proves that the tunnel has been setup successfully and the packet could reach the VPN Server. Figure 19 shows the results of this.



```
192.168.136.133:35404 --> 0.0.0.0:9080
 Inside: 10.4.0.1 --> 10.4.0.5
192.168.136.133:35404 --> 0.0.0.0:9080
 Inside: 10.4.0.1 --> 10.4.0.5
192.168.136.133:35404 --> 0.0.0.0:9080
 Inside: 10.4.0.1 --> 10.4.0.5
192.168.136.133:35404 --> 0.0.0.0:9080
 Inside: 10.4.0.1 --> 10.4.0.5
192.168.136.133:35404 --> 0.0.0.0:9080
 Inside: 10.4.0.1 --> 10.4.0.5
192.168.136.133:35404 --> 0.0.0.0:9080
 Inside: 10.4.0.1 --> 10.4.0.5
192.168.136.133:35404 --> 0.0.0.0:9080
 Inside: 10.4.0.1 --> 10.4.0.5
192.168.136.133:35404 --> 0.0.0.0:9080
 Inside: 10.4.0.1 --> 10.4.0.5
192.168.136.133:35404 --> 0.0.0.0:9080
 Inside: 10.4.0.1 --> 10.4.0.5
192.168.136.133:35404 --> 0.0.0.0:9080
 Inside: 10.4.0.1 --> 10.4.0.5
192.168.136.133:35404 --> 0.0.0.0:9080
 Inside: 10.4.0.1 --> 10.4.0.5
```

Figure 19 Output of tun_server.py

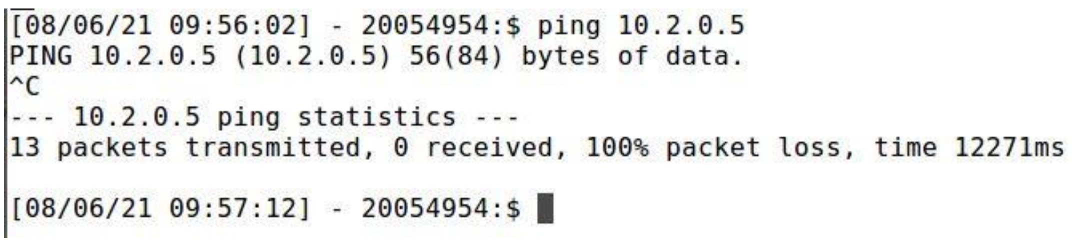
Now we try to ping any host in Host V network from the VPN Client. Initially, the ICMP request could reach the server because the packets was travelling through the 0.0.0.0 default route. To overcome this, we need to add our route to the routing table so that the packets travel through our interface kumar0. The commands and execution of adding an entry to the routing table is shown in Figure 20.



```
[08/06/21 09:54:44] - 20054954:$ sudo ip route add 10.2.0.0/24 dev kumar1
[08/06/21 09:55:26] - 20054954:$ netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         192.168.136.2   0.0.0.0         UG        0 0          0 ens33
10.2.0.0        0.0.0.0         255.255.255.0   U         0 0          0 kumar1
10.4.0.0        0.0.0.0         255.255.255.0   U         0 0          0 kumar1
192.168.136.0   0.0.0.0         255.255.255.0   U         0 0          0 ens33
```

Figure 20 Adding an entry to the routing table

Once this setup is done, we try to pinging a host in the Host V network (10.2.0.5). Now, we could see that the packets reach the VPN Server as shown in Figure 22. This proves the packets travel through our tunnel. Since the Server is not configured to provide a response, we couldn’t get a response at VPN Client (as shown in Figure 21).



```
[08/06/21 09:56:02] - 20054954:$ ping 10.2.0.5
PING 10.2.0.5 (10.2.0.5) 56(84) bytes of data.
^C
--- 10.2.0.5 ping statistics ---
13 packets transmitted, 0 received, 100% packet loss, time 12271ms

[08/06/21 09:57:12] - 20054954:$ 
```

Figure 21 Pinging host in Host V network

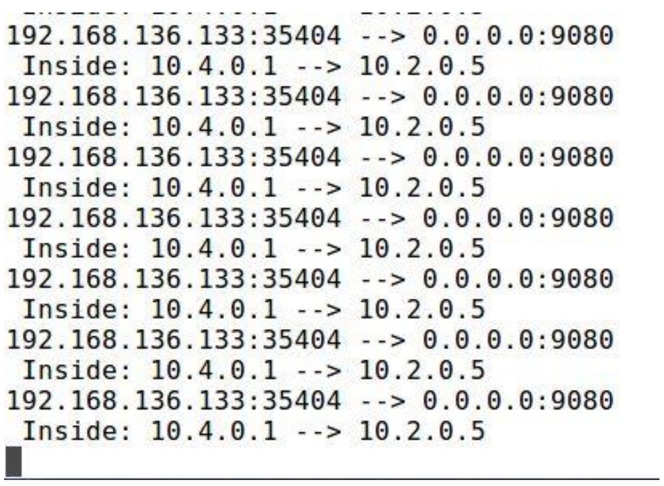
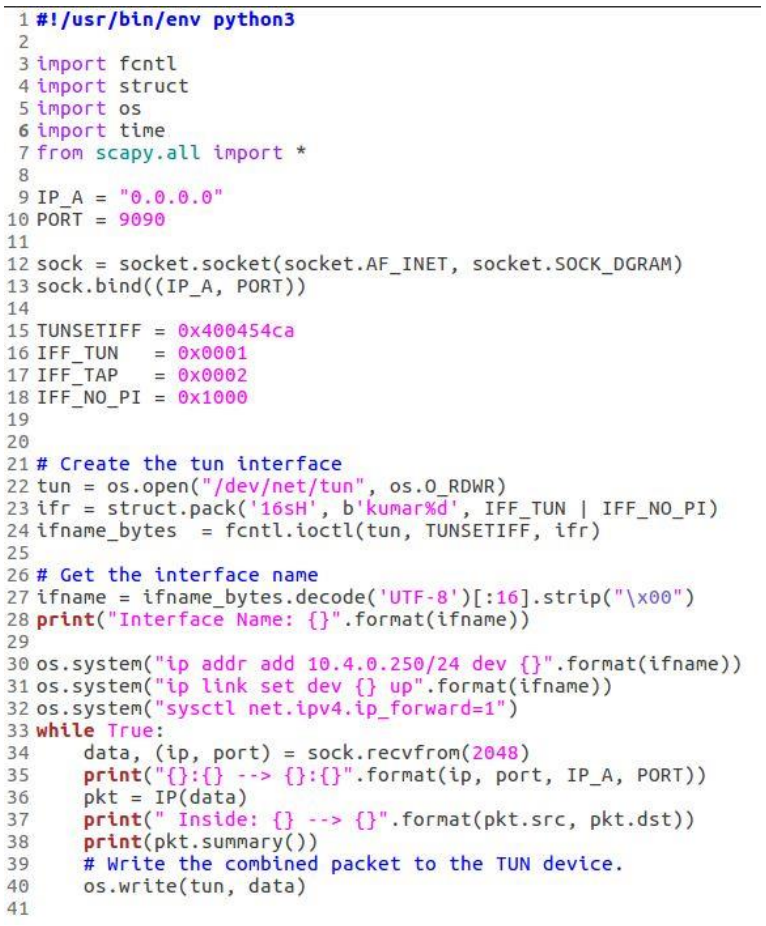

```
192.168.136.133:35404 --> 0.0.0.0:9080
 Inside: 10.4.0.1 --> 10.2.0.5
192.168.136.133:35404 --> 0.0.0.0:9080
 Inside: 10.4.0.1 --> 10.2.0.5
192.168.136.133:35404 --> 0.0.0.0:9080
 Inside: 10.4.0.1 --> 10.2.0.5
192.168.136.133:35404 --> 0.0.0.0:9080
 Inside: 10.4.0.1 --> 10.2.0.5
192.168.136.133:35404 --> 0.0.0.0:9080
 Inside: 10.4.0.1 --> 10.2.0.5
192.168.136.133:35404 --> 0.0.0.0:9080
 Inside: 10.4.0.1 --> 10.2.0.5
192.168.136.133:35404 --> 0.0.0.0:9080
 Inside: 10.4.0.1 --> 10.2.0.5
```

Figure 22 Results of server program with packet details

1.4 Task 4: Setup the VPN Server

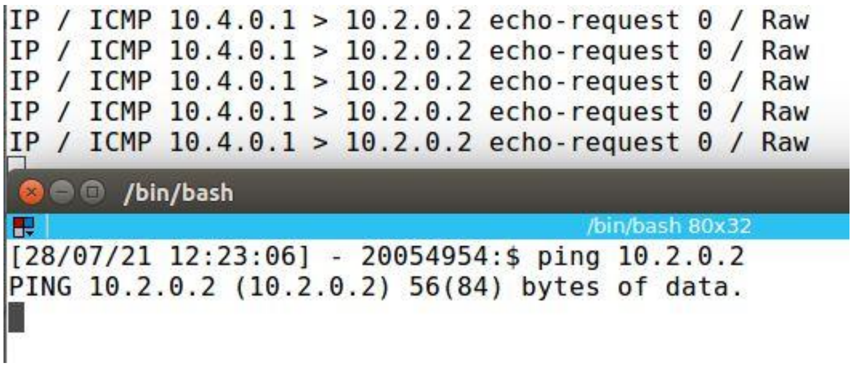
For this task, the tun_server.py program was modified as shown in Figure 23. This changes in the code are to write a response back to the TUN interface. Because of this the ICMP request sent by VPN Client is received by the server and writes it to the Host V.



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

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

IP_A = "0.0.0.0"
PORT = 9090

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))

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'kumar%d', IFF_TUN | IFF_NO_PI)
ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)

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

os.system("ip addr add 10.4.0.250/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
os.system("sysctl net.ipv4.ip_forward=1")
while True:
    data, (ip, port) = sock.recvfrom(2048)
    print("{}:{} --> {}:{}".format(ip, port, IP_A, PORT))
    pkt = IP(data)
    print(" Inside: {} --> {}".format(pkt.src, pkt.dst))
    print(pkt.summary())
    # Write the combined packet to the TUN device.
    os.write(tun, data)

```

Figure 23 Modified tun_server.py for task 4

Figure 24 provides the execution of tun_client.py program and the pinging from VPN Client.

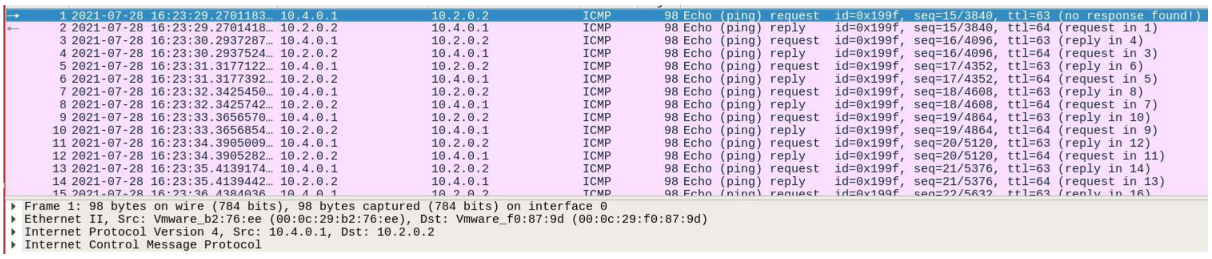


```
IP / ICMP 10.4.0.1 > 10.2.0.2 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.2.0.2 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.2.0.2 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.2.0.2 echo-request 0 / Raw
IP / ICMP 10.4.0.1 > 10.2.0.2 echo-request 0 / Raw
```

```
[28/07/21 12:23:06] - 20054954:$ ping 10.2.0.2
PING 10.2.0.2 (10.2.0.2) 56(84) bytes of data.
```

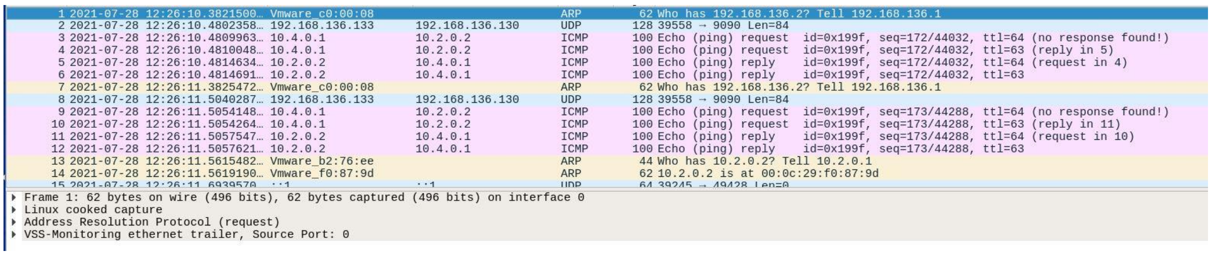
Figure 24 Pinging the Host V from VPN Client

The Wireshark trace results of packets sent from VPN Client to Host V is shown in Figure 25. This proves that the packet has reached the Host V. Figure 26 shows the trace of these packets in Wireshark of VPN Server. We could see the request from VPN Client and a response from the Host V.



```
→   1 2021-07-28 16:23:29.2701183… 10.4.0.1     10.2.0.2     ICMP    98 Echo (ping) request  id=0x199f, seq=15/3840, ttl=63 (no response found!)
←   2 2021-07-28 16:23:29.2701418… 10.2.0.2     10.4.0.1     ICMP    98 Echo (ping) reply    id=0x199f, seq=15/3840, ttl=64 (request in 1)
    3 2021-07-28 16:23:30.2937287… 10.4.0.1     10.2.0.2     ICMP    98 Echo (ping) request  id=0x199f, seq=16/4096, ttl=63 (reply in 4)
    4 2021-07-28 16:23:30.2937524… 10.2.0.2     10.4.0.1     ICMP    98 Echo (ping) reply    id=0x199f, seq=16/4096, ttl=64 (request in 3)
    5 2021-07-28 16:23:31.3177122… 10.4.0.1     10.2.0.2     ICMP    98 Echo (ping) request  id=0x199f, seq=17/4352, ttl=63 (reply in 6)
    6 2021-07-28 16:23:31.3177392… 10.2.0.2     10.4.0.1     ICMP    98 Echo (ping) reply    id=0x199f, seq=17/4352, ttl=64 (request in 5)
    7 2021-07-28 16:23:32.3425450… 10.4.0.1     10.2.0.2     ICMP    98 Echo (ping) request  id=0x199f, seq=18/4608, ttl=63 (reply in 8)
    8 2021-07-28 16:23:32.3425742… 10.2.0.2     10.4.0.1     ICMP    98 Echo (ping) reply    id=0x199f, seq=18/4608, ttl=64 (request in 7)
    9 2021-07-28 16:23:33.3656570… 10.4.0.1     10.2.0.2     ICMP    98 Echo (ping) request  id=0x199f, seq=19/4864, ttl=63 (reply in 10)
   10 2021-07-28 16:23:33.3656854… 10.2.0.2     10.4.0.1     ICMP    98 Echo (ping) reply    id=0x199f, seq=19/4864, ttl=64 (request in 9)
   11 2021-07-28 16:23:34.3905009… 10.4.0.1     10.2.0.2     ICMP    98 Echo (ping) request  id=0x199f, seq=20/5120, ttl=63 (reply in 12)
   12 2021-07-28 16:23:34.3905282… 10.2.0.2     10.4.0.1     ICMP    98 Echo (ping) reply    id=0x199f, seq=20/5120, ttl=64 (request in 11)
   13 2021-07-28 16:23:35.4139174… 10.4.0.1     10.2.0.2     ICMP    98 Echo (ping) request  id=0x199f, seq=21/5376, ttl=63 (reply in 14)
   14 2021-07-28 16:23:35.4139442… 10.2.0.2     10.4.0.1     ICMP    98 Echo (ping) reply    id=0x199f, seq=21/5376, ttl=64 (request in 13)
   15 2021-07-28 16:23:36.4384036… 10.4.0.1     10.2.0.2     ICMP    98 Echo (ping) request  id=0x199f, seq=22/5632, ttl=63 (reply in 16)
▸ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
▸ Ethernet II, Src: Vmware_b2:76:ee (00:0c:29:b2:76:ee), Dst: Vmware_f0:87:9d (00:0c:29:f0:87:9d)
▸ Internet Protocol Version 4, Src: 10.4.0.1, Dst: 10.2.0.2
▸ Internet Control Message Protocol
```

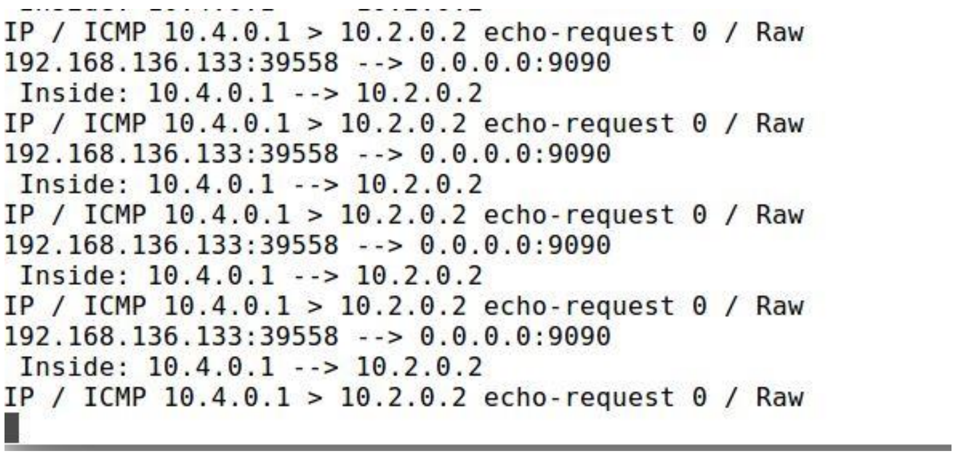
Figure 25 Host V Wireshark trace of packet reaching Host V



```
    1 2021-07-28 12:26:10.3821500… Vmware_c0:00:08                     ARP      62 Who has 192.168.136.2? Tell 192.168.136.1
    2 2021-07-28 12:26:10.4802358… 192.168.136.133   192.168.136.130   UDP     128 39558 → 9090 Len=84
    3 2021-07-28 12:26:10.4809963… 10.4.0.1          10.2.0.2          ICMP    100 Echo (ping) request  id=0x199f, seq=172/44032, ttl=64 (no response found!)
    4 2021-07-28 12:26:10.4810048… 10.4.0.1          10.2.0.2          ICMP    100 Echo (ping) request  id=0x199f, seq=172/44032, ttl=63 (reply in 5)
    5 2021-07-28 12:26:10.4814634… 10.2.0.2          10.4.0.1          ICMP    100 Echo (ping) reply    id=0x199f, seq=172/44032, ttl=64 (request in 4)
    6 2021-07-28 12:26:10.4814691… 10.2.0.2          10.4.0.1          ICMP    100 Echo (ping) reply    id=0x199f, seq=172/44032, ttl=63
    7 2021-07-28 12:26:11.3825472… Vmware_c0:00:08                     ARP      62 Who has 192.168.136.2? Tell 192.168.136.1
    8 2021-07-28 12:26:11.5040287… 192.168.136.133   192.168.136.130   UDP     128 39558 → 9090 Len=84
    9 2021-07-28 12:26:11.5054148… 10.4.0.1          10.2.0.2          ICMP    100 Echo (ping) request  id=0x199f, seq=173/44288, ttl=64 (no response found!)
   10 2021-07-28 12:26:11.5054264… 10.4.0.1          10.2.0.2          ICMP    100 Echo (ping) request  id=0x199f, seq=173/44288, ttl=63 (reply in 11)
   11 2021-07-28 12:26:11.5057547… 10.2.0.2          10.4.0.1          ICMP    100 Echo (ping) reply    id=0x199f, seq=173/44288, ttl=64 (request in 10)
   12 2021-07-28 12:26:11.5057621… 10.2.0.2          10.4.0.1          ICMP    100 Echo (ping) reply    id=0x199f, seq=173/44288, ttl=63
   13 2021-07-28 12:26:11.5615482… Vmware_b2:76:ee                     ARP      44 Who has 10.2.0.2? Tell 10.2.0.1
   14 2021-07-28 12:26:11.5619190… Vmware_f0:87:9d                     ARP      62 10.2.0.2 is at 00:0c:29:f0:87:9d
   15 2021-07-28 12:26:11.6939570… ::1               ::1               UDP      64 39245 → 49428 Len=0
▸ Frame 1: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface 0
▸ Linux cooked capture
▸ Address Resolution Protocol (request)
▸ VSS-Monitoring ethernet trailer, Source Port: 0
```

Figure 26 VPN Server Wireshark trace of packets reaching Host V

Figure 27 shows the output of tun_server.py program which shows the packet details.

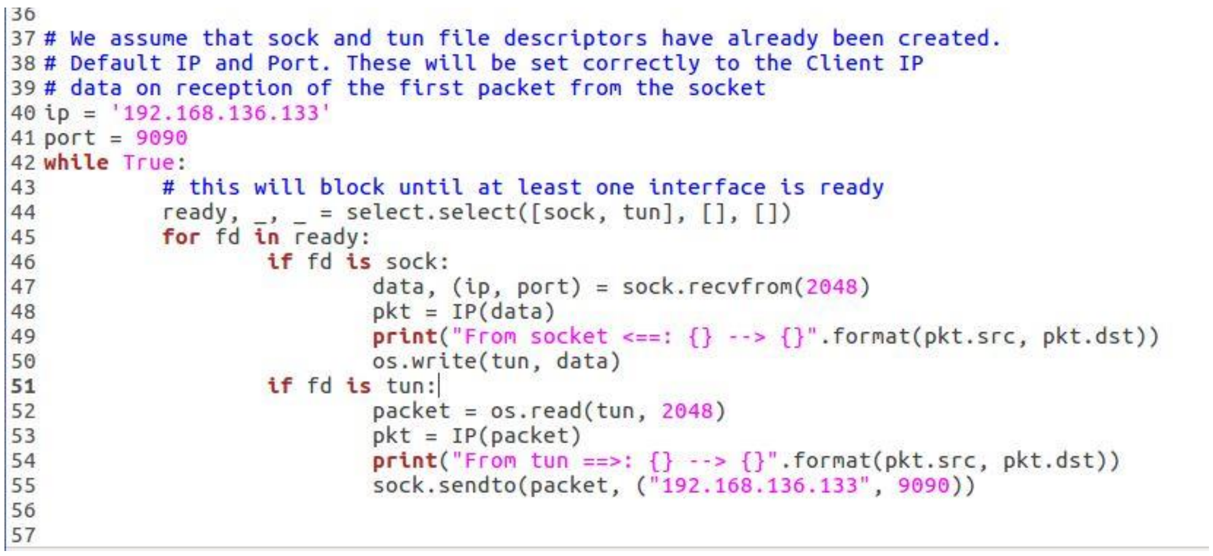


```
IP / ICMP 10.4.0.1 > 10.2.0.2 echo-request 0 / Raw
192.168.136.133:39558 --> 0.0.0.0:9090
 Inside: 10.4.0.1 --> 10.2.0.2
IP / ICMP 10.4.0.1 > 10.2.0.2 echo-request 0 / Raw
192.168.136.133:39558 --> 0.0.0.0:9090
 Inside: 10.4.0.1 --> 10.2.0.2
IP / ICMP 10.4.0.1 > 10.2.0.2 echo-request 0 / Raw
192.168.136.133:39558 --> 0.0.0.0:9090
 Inside: 10.4.0.1 --> 10.2.0.2
IP / ICMP 10.4.0.1 > 10.2.0.2 echo-request 0 / Raw
192.168.136.133:39558 --> 0.0.0.0:9090
 Inside: 10.4.0.1 --> 10.2.0.2
IP / ICMP 10.4.0.1 > 10.2.0.2 echo-request 0 / Raw
```

Figure 27 Output of tun_server.py

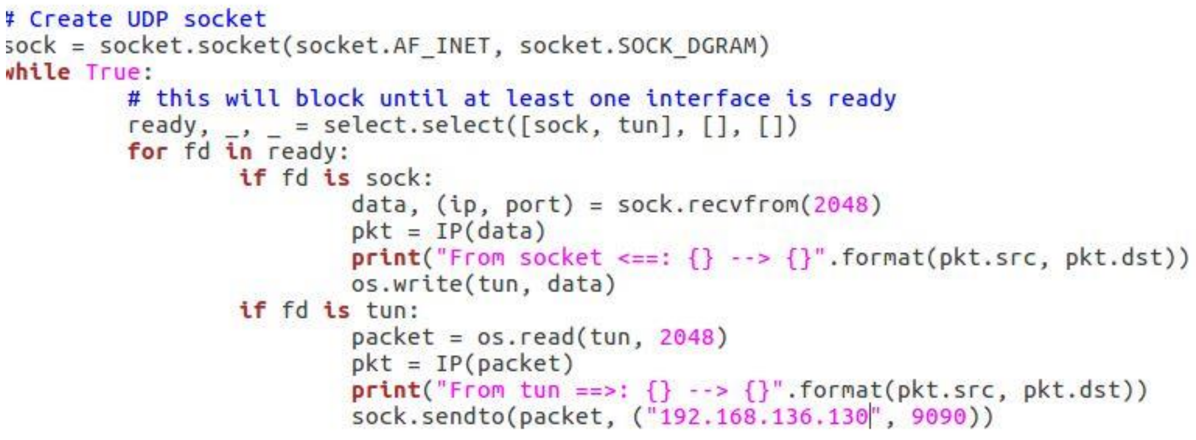
1.5 Task 5: Handling Traffic in Both Directions

For this task, the provided python code was used. Both the tun_server.py and tun_client.py code was modified to the code shown in Figure 28 and 29. The changes were made to write the data out to correct interface.



```

# We assume that sock and tun file descriptors have already been created.
# Default IP and Port. These will be set correctly to the Client IP
# data on reception of the first packet from the socket
ip = '192.168.136.133'
port = 9090
while True:
        # this will block until at least one interface is ready
        ready, _, _ = select.select([sock, tun], [], [])
        for fd in ready:
                if fd is sock:
                        data, (ip, port) = sock.recvfrom(2048)
                        pkt = IP(data)
                        print("From socket <==: {} --> {}".format(pkt.src, pkt.dst))
                        os.write(tun, data)
                if fd is tun:
                        packet = os.read(tun, 2048)
                        pkt = IP(packet)
                        print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))
                        sock.sendto(packet, ("192.168.136.133", 9090))


```

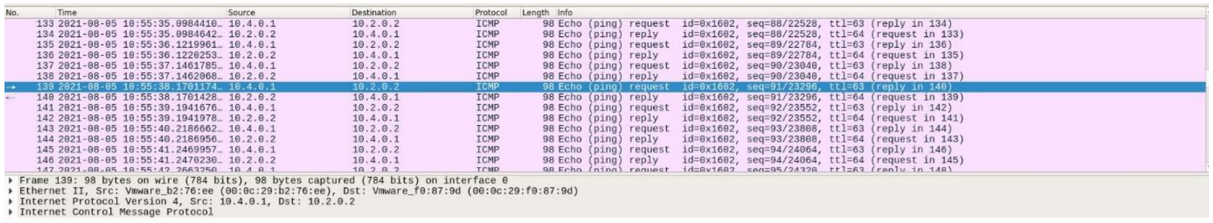
Figure 28 tun_server.py code



```
# Create UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
while True:
        # this will block until at least one interface is ready
        ready, _, _ = select.select([sock, tun], [], [])
        for fd in ready:
                if fd is sock:
                        data, (ip, port) = sock.recvfrom(2048)
                        pkt = IP(data)
                        print("From socket <==: {} --> {}".format(pkt.src, pkt.dst))
                        os.write(tun, data)
                if fd is tun:
                        packet = os.read(tun, 2048)
                        pkt = IP(packet)
                        print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))
                        sock.sendto(packet, ("192.168.136.130", 9090))
```

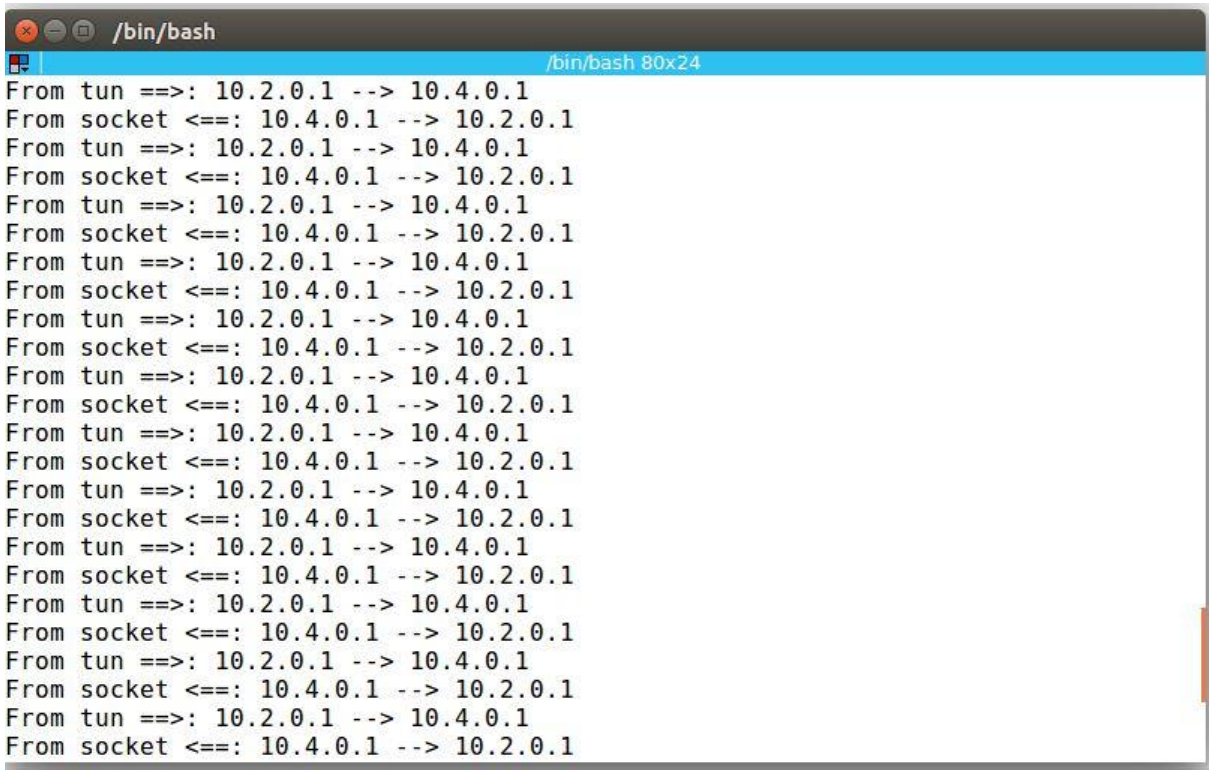
Figure 29 tun_client.py

When these code was executed, we try to ping Host V from VPN client. This time we were able to get a response in VPN Client as shown in Figure 30. This is because the python code was able to read the request from the socket and provide the response to the tunnel so that VPN Client could get it. Figure 31 shows the wireshark trace of the ICMP request and response in the Host V. This proves that the ICMP request has reached Host V and it replies with ICMP response.



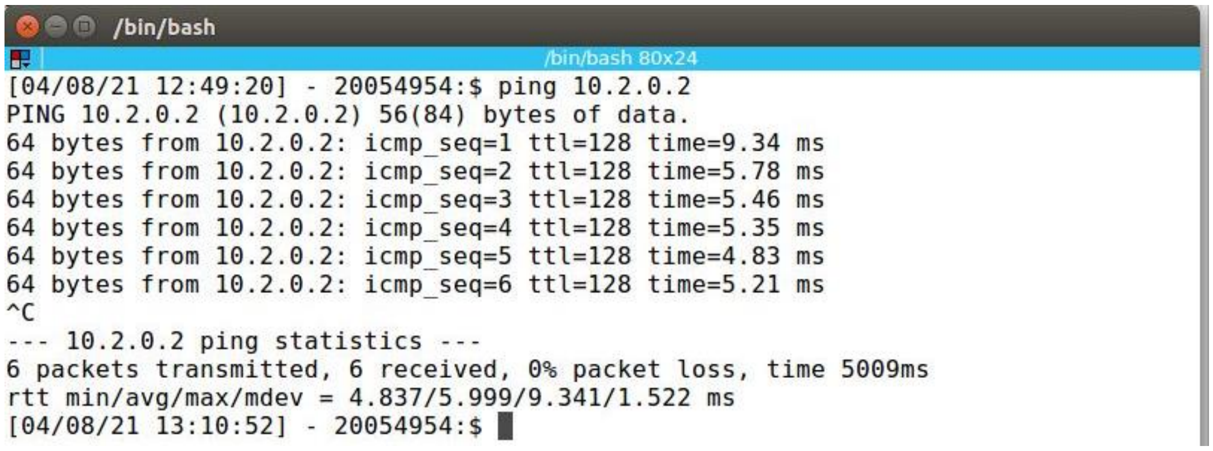
```
No.  Time                          Source     Destination  Protocol  Length Info
 133 2021-08-05 10:55:35.0984410…  10.4.0.1   10.2.0.2     ICMP      98 Echo (ping) request  id=0x1602, seq=88/22528, ttl=63 (reply in 134)
 134 2021-08-05 10:55:35.0984642…  10.2.0.2   10.4.0.1     ICMP      98 Echo (ping) reply    id=0x1602, seq=88/22528, ttl=64 (request in 133)
 135 2021-08-05 10:55:36.1219961…  10.4.0.1   10.2.0.2     ICMP      98 Echo (ping) request  id=0x1602, seq=89/22784, ttl=63 (reply in 136)
 136 2021-08-05 10:55:36.1220253…  10.2.0.2   10.4.0.1     ICMP      98 Echo (ping) reply    id=0x1602, seq=89/22784, ttl=64 (request in 135)
 137 2021-08-05 10:55:37.1461785…  10.4.0.1   10.2.0.2     ICMP      98 Echo (ping) request  id=0x1602, seq=90/23040, ttl=63 (reply in 138)
 138 2021-08-05 10:55:37.1462068…  10.2.0.2   10.4.0.1     ICMP      98 Echo (ping) reply    id=0x1602, seq=90/23040, ttl=64 (request in 137)
 139 2021-08-05 10:55:38.1701174…  10.4.0.1   10.2.0.2     ICMP      98 Echo (ping) request  id=0x1602, seq=91/23296, ttl=63 (reply in 140)
 140 2021-08-05 10:55:38.1701428…  10.2.0.2   10.4.0.1     ICMP      98 Echo (ping) reply    id=0x1602, seq=91/23296, ttl=64 (request in 139)
 141 2021-08-05 10:55:39.1941676…  10.4.0.1   10.2.0.2     ICMP      98 Echo (ping) request  id=0x1602, seq=92/23552, ttl=63 (reply in 142)
 142 2021-08-05 10:55:39.1941978…  10.2.0.2   10.4.0.1     ICMP      98 Echo (ping) reply    id=0x1602, seq=92/23552, ttl=64 (request in 141)
 143 2021-08-05 10:55:40.2186662…  10.4.0.1   10.2.0.2     ICMP      98 Echo (ping) request  id=0x1602, seq=93/23808, ttl=63 (reply in 144)
 144 2021-08-05 10:55:40.2186956…  10.2.0.2   10.4.0.1     ICMP      98 Echo (ping) reply    id=0x1602, seq=93/23808, ttl=64 (request in 143)
 145 2021-08-05 10:55:41.2469957…  10.4.0.1   10.2.0.2     ICMP      98 Echo (ping) request  id=0x1602, seq=94/24064, ttl=63 (reply in 146)
 146 2021-08-05 10:55:41.2470230…  10.2.0.2   10.4.0.1     ICMP      98 Echo (ping) reply    id=0x1602, seq=94/24064, ttl=64 (request in 145)
 147 2021-08-05 10:55:42.2663250…  10.4.0.1   10.2.0.2     ICMP      98 Echo (ping) request  id=0x1602, seq=95/24320, ttl=63 (reply in 148)
▸ Frame 139: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
▸ Ethernet II, Src: Vmware_b2:76:ee (00:0c:29:b2:76:ee), Dst: Vmware_f0:87:9d (00:0c:29:f0:87:9d)
▸ Internet Protocol Version 4, Src: 10.4.0.1, Dst: 10.2.0.2
▸ Internet Control Message Protocol
```

Figure 30 Wireshark trace of packets in Host V



```
From tun ==>: 10.2.0.1 --> 10.4.0.1
From socket <==: 10.4.0.1 --> 10.2.0.1
From tun ==>: 10.2.0.1 --> 10.4.0.1
From socket <==: 10.4.0.1 --> 10.2.0.1
From tun ==>: 10.2.0.1 --> 10.4.0.1
From socket <==: 10.4.0.1 --> 10.2.0.1
From tun ==>: 10.2.0.1 --> 10.4.0.1
From socket <==: 10.4.0.1 --> 10.2.0.1
From tun ==>: 10.2.0.1 --> 10.4.0.1
From socket <==: 10.4.0.1 --> 10.2.0.1
From tun ==>: 10.2.0.1 --> 10.4.0.1
From socket <==: 10.4.0.1 --> 10.2.0.1
From tun ==>: 10.2.0.1 --> 10.4.0.1
From socket <==: 10.4.0.1 --> 10.2.0.1
From tun ==>: 10.2.0.1 --> 10.4.0.1
From socket <==: 10.4.0.1 --> 10.2.0.1
From tun ==>: 10.2.0.1 --> 10.4.0.1
From socket <==: 10.4.0.1 --> 10.2.0.1
From tun ==>: 10.2.0.1 --> 10.4.0.1
From socket <==: 10.4.0.1 --> 10.2.0.1
From tun ==>: 10.2.0.1 --> 10.4.0.1
From socket <==: 10.4.0.1 --> 10.2.0.1
From tun ==>: 10.2.0.1 --> 10.4.0.1
From socket <==: 10.4.0.1 --> 10.2.0.1
```

Figure 31 Output of tun_server.py showing packets movements



```
[04/08/21 12:49:20] - 20054954:$ ping 10.2.0.2
PING 10.2.0.2 (10.2.0.2) 56(84) bytes of data.
64 bytes from 10.2.0.2: icmp_seq=1 ttl=128 time=9.34 ms
64 bytes from 10.2.0.2: icmp_seq=2 ttl=128 time=5.78 ms
64 bytes from 10.2.0.2: icmp_seq=3 ttl=128 time=5.46 ms
64 bytes from 10.2.0.2: icmp_seq=4 ttl=128 time=5.35 ms
64 bytes from 10.2.0.2: icmp_seq=5 ttl=128 time=4.83 ms
64 bytes from 10.2.0.2: icmp_seq=6 ttl=128 time=5.21 ms
^C
--- 10.2.0.2 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5009ms
rtt min/avg/max/mdev = 4.837/5.999/9.341/1.522 ms
[04/08/21 13:10:52] - 20054954:$ █
```

Figure 32 Successful ping in VPN CLient

1.7 Choosing between an SSL/TLS VPN vs IPsec VPN (Research Task)

INTRODUCTION

Virtual Private Network (VPN) is widely used to connect a private network to a public network. It allows the user connected to a public network to send and receive data to other users connected to a private network as if they are part of the private network without compromising the security and management policies. VPN provides additional security and privacy for the exchange of data. The privacy of data is provided through tunnelling protocols. The additional security is provided through encryption, the sender's data, sender, and receiver network address are encrypted and forwarded through the tunnel; this is then decrypted by the receiver(Chawla, Gupta and Sawhney, 2014). The VPN is commonly used for connecting between two remote sites of an organization (site-to-site) or between user and the organization (point-to-point). The most widely used approaches for building a VPN are Internet Protocol Security (IPsec) and Secure Socket Layer (SSL) (Korhonen, 2019).

IPSEC VPN

Internet Protocol Security (IPsec) is a security protocol proposed by Internet Engineering Task force (IETF) in several Request for Comments (RFC) documents to provide secure communication for IP layer. IPsec protocol includes security protocol, which defines the methods for protecting the communication, and key negotiation, which defines the parameters for negotiation and identity authentication (Mao, Zhu and Qin, 2012). The three primary components of IPsec protocol are Authentication Header (AH), Encapsulating Security Payload (ESP) and Internet Key Exchange (IKE) protocols(Chawla, Gupta and Sawhney, 2014). The AH and ESP are mechanisms for providing communication protection, whereas IKE provides security parameters negotiation (Mao, Zhu and Qin, 2012).

1. Authentication Header (AH):

AH provides data authentication and integrity protection. It helps in protecting all fields of an IP datagram. Encryption is not provided because AH does not guarantee data confidentiality. To improve the level of security, AH uses MD5 or SHA1 as hashing algorithm. In tunnel mode, a new IP Header is assigned for every packet by the AH (Chawla, Gupta and Sawhney, 2014).

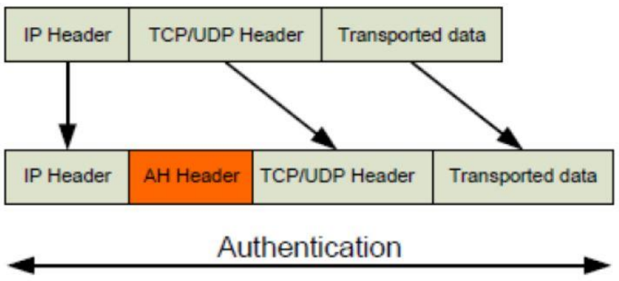


Figure 33 AH in Transport mode (Chawla, Gupta and Sawhney, 2014)

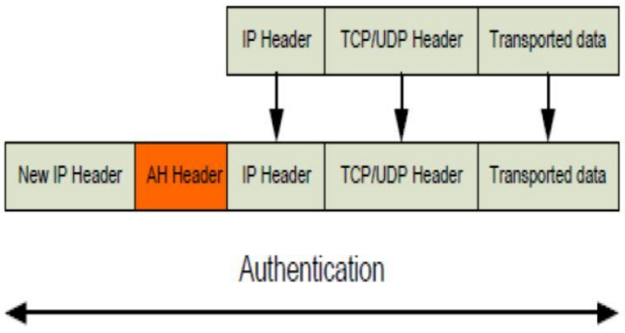


Figure 34 AH in Tunnel mode (Chawla, Gupta and Sawhney, 2014)

2. Encapsulating Security Payload (ESP):

The ESP mechanism provides data confidentiality and integrity protection. For this, ESP uses encryption services. This mechanism can prevent anti-replay attack (Mao, Zhu and Qin, 2012). Symmetric encryption is used by ESP for providing data privacy. ESP supports two configurations: authentication-only and encryption-only. AH and ESP may be used individually or together but the most common practice is to use only ESP (Chawla, Gupta and Sawhney, 2014).

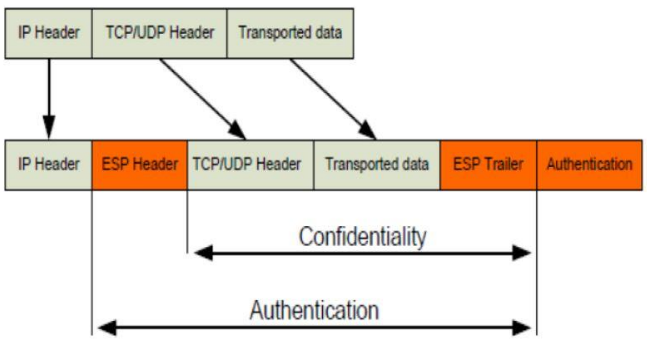


Figure 35 ESP in Transport mode (Chawla, Gupta and Sawhney, 2014)

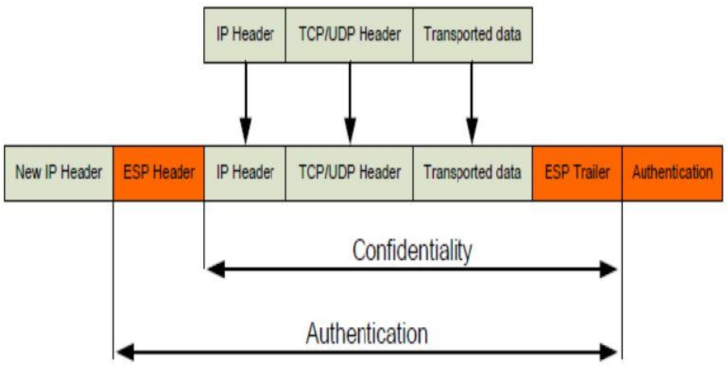


Figure 36 ESP in Tunnel mode (Chawla, Gupta and Sawhney, 2014)

3. Internet Key Exchange (IKE):

IPsec uses Security Associations (SA), which is logical simplex connection between IPsec endpoints. The connection has to be established before endpoints can exchange any data. Both the endpoints agree on how to establish communication and exchange data. IPsec supports IKE protocol which allows dynamic configuration of SA (Korhonen, 2019). IKE is used to determine and negotiate algorithm, keys and protocols, and to authenticate two endpoints.

TLS/SSL VPN

Secure Socket Layer (SSL) VPN is based on SSL protocol and can be used with standard web browser. Transport Layer Security (TLS) protocol is the successor of SSL, which is commonly used for securing TCP traffic (Korhonen, 2019).

TLS protocol:

TLS fulfils the basic requirements of VPN namely, encryption through symmetric encryption, authentication through public-key cryptography and integrity through message authentication code. The TLS protocol can be divided into TLS Handshake protocol and TLS Record protocol. TLS Handshake protocols handles the handshake, authentication of parties, key-related information exchange. TLS Record protocol handles the exchange of application data (Korhonen, 2019).

TLS/SSL VPN Architecture:

TLS/SSL support two types of operating modes, one is portal/transport mode and the other is tunnel mode. In portal mode, the user connects to the web portal through the web browser, once successfully logged in, they can access the services through the web portal. The tunnel mode uses client software or browser to connect to the VPN endpoint. This client software authenticates the user and creates a TLS connection between the user and VPN endpoint. The authentication is established with combination of X.509 certificate, username, password and one-time-password (Korhonen, 2019).

ADVANTAGES AND DISADVANTAGES ANALYSIS FOR TLS/SSL AND IPSEC VPN

TLS/SSL VPN:

Advantages:

1. TLS/SSL VPN in portal mode is easy to implement, the user only needs a web browser which can support TLS and credentials. This reduces the maintenance and management cost.
2. The tunnel mode allows users to use all kinds of applications to access resources and services in the intranet of the organization. If all traffic is routed to the tunnel, organization can monitor and protect their VPN users with organization firewalls and other security appliances.
3. SSL VPN can work in NAT proxy device in portal mode (Mao, Zhu and Qin, 2012).

Disadvantages:

1. In portal mode, the user can only access services via the VPN portal with browser and all other traffic is routed through public network.
2. SSL VPN is mostly used for remote access and is not suitable for site-to-site VPN.
3. SSL VPN usually requires additional security configuration, such as the third party to authenticate security of un-trusted equipment (Mao, Zhu and Qin, 2012).

IPsec VPN:

Advantages:

1. IPsec VPN is suitable for LAN-to-LAN communication. IPsec allows to monitor all the traffic in the network.
2. IPsec VPN does not depend on particular application because of the VPN client supports all IP layer protocol (Mao, Zhu and Qin, 2012).
3. IPsec VPN provides data confidentiality, integrity and privacy.

Disadvantages:

1. IPsec VPN client needs a client software for communication. This increases the management cost of IPsec client.
2. In remote access and enterprise network, the user is provided with full access to all application of a company, this increases the security risk.
3. IPsec VPN is not suitable for network translation, firewall traversal and broadband access.

CONCLUSION

With the understanding of both the VPN, the SSL VPN is well suited for remote access, where the user is provided with access to particular application rather than the complete network. Whereas, IPsec is suitable if the application is IP specific and can have an IPsec gateway located at the organization.

1.8 Reference

Chawla, B. K., Gupta, O. P. and Sawhney, B. K. (2014) 'A Review on IPsec and SSL VPN', *International Journal of Scientific & Engineering Research*, 5(11), pp. 21–24.

Korhonen, V. (2019) 'FUTURE AFTER OPENVPN AND IPSEC', (August).

Mao, H., Zhu, L. and Qin, H. (2012) 'A comparative research on SSL VPN and IPSec VPN', *2012 International Conference on Wireless Communications, Networking and Mobile Computing, WiCOM 2012*, 2012, pp. 31–34. doi: 10.1109/WiCOM.2012.6478270.