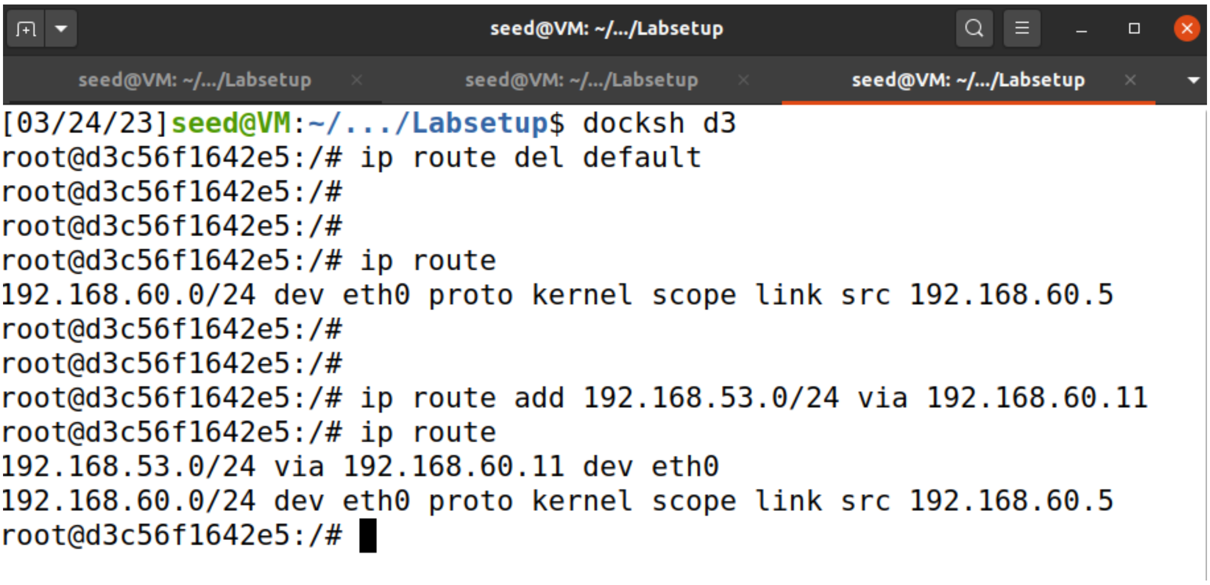
NAME: YASH SNEHAL SHETIYA

SUID: 9276568741

VPN LAB PART 2
**************************************************************************

Task 7:

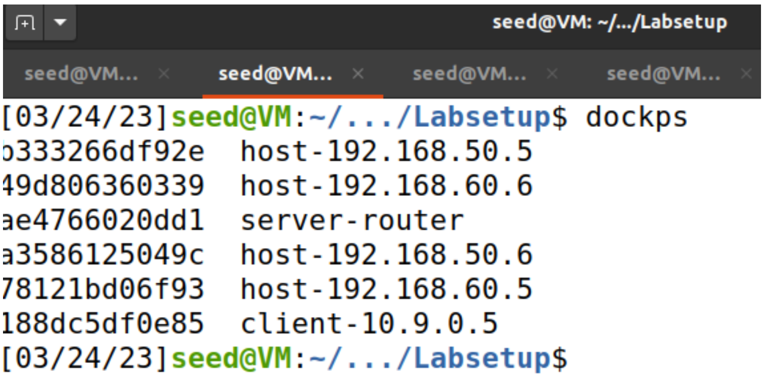
In this task we delete the default route and then add a custom route that is needed which will add a more specific entry to the routing table, so that the return packets can be sent to the VPN server.



```
[03/24/23]seed@VM:~/.../Labsetup$ docksh d3
root@d3c56f1642e5:/# ip route del default
root@d3c56f1642e5:/#
root@d3c56f1642e5:/#
root@d3c56f1642e5:/# ip route
192.168.60.0/24 dev eth0 proto kernel scope link src 192.168.60.5
root@d3c56f1642e5:/#
root@d3c56f1642e5:/#
root@d3c56f1642e5:/# ip route add 192.168.53.0/24 via 192.168.60.11
root@d3c56f1642e5:/# ip route
192.168.53.0/24 via 192.168.60.11 dev eth0
192.168.60.0/24 dev eth0 proto kernel scope link src 192.168.60.5
root@d3c56f1642e5:/#
```

Task 8:

This task is performed in a new setup environment.

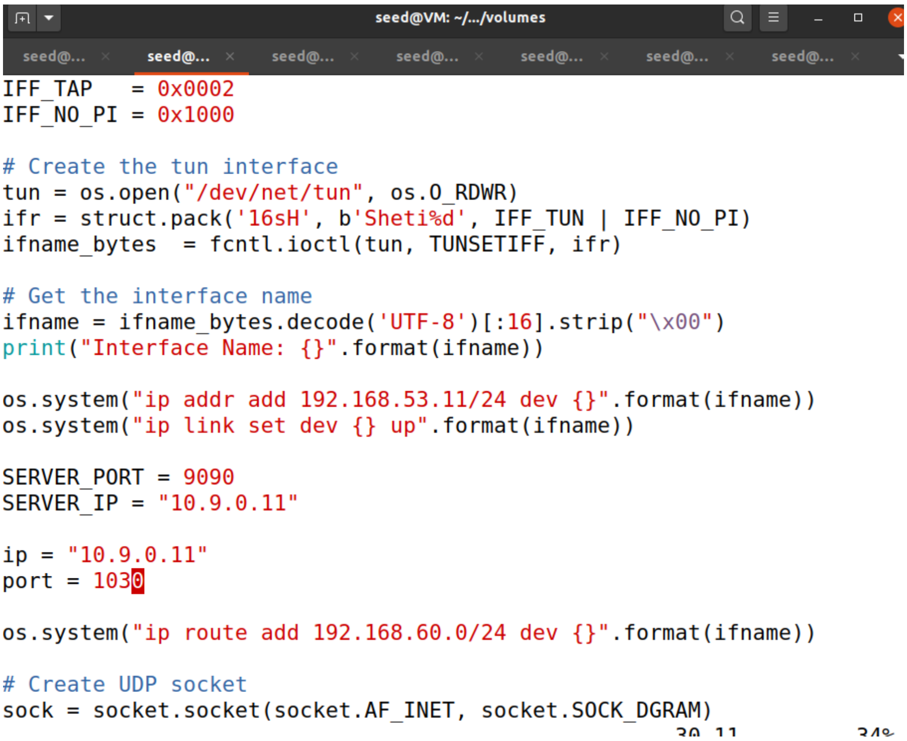


```
[03/24/23]seed@VM:~/.../Labsetup$ dockps
b333266df92e  host-192.168.50.5
49d806360339  host-192.168.60.6
ae4766020dd1  server-router
a3586125049c  host-192.168.50.6
78121bd06f93  host-192.168.60.5
188dc5df0e85  client-10.9.0.5
[03/24/23]seed@VM:~/.../Labsetup$
```

It is required to establish a tunnel to achieve intercommunication at both ends.

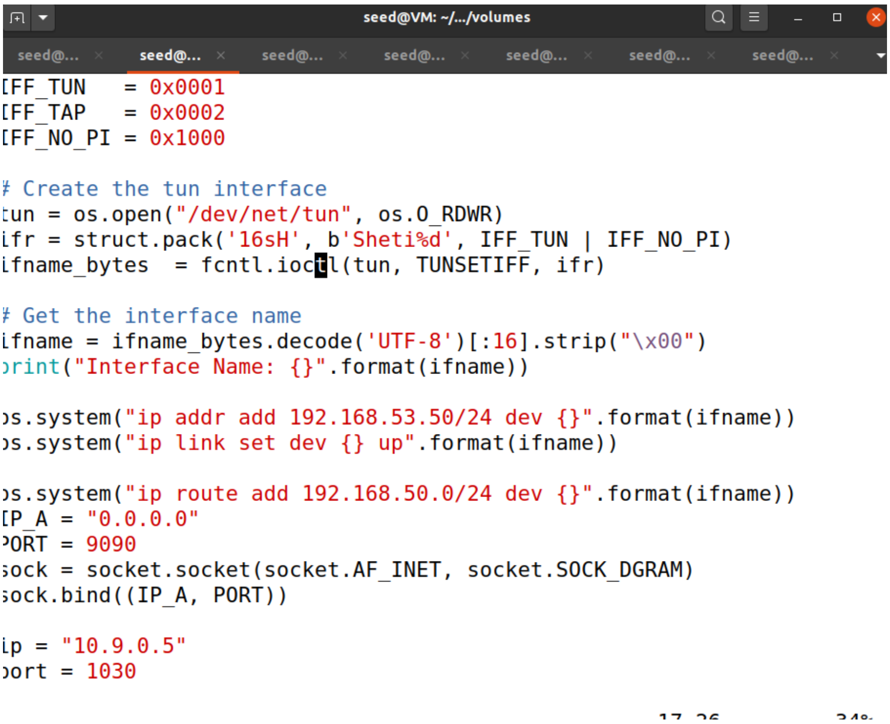
When tried to ping 192.168.60.5 from Host U, it could be observed that packets were transmitted for sure but were not received. After this, We did the modifications required in the code and are as follows:

Client side:



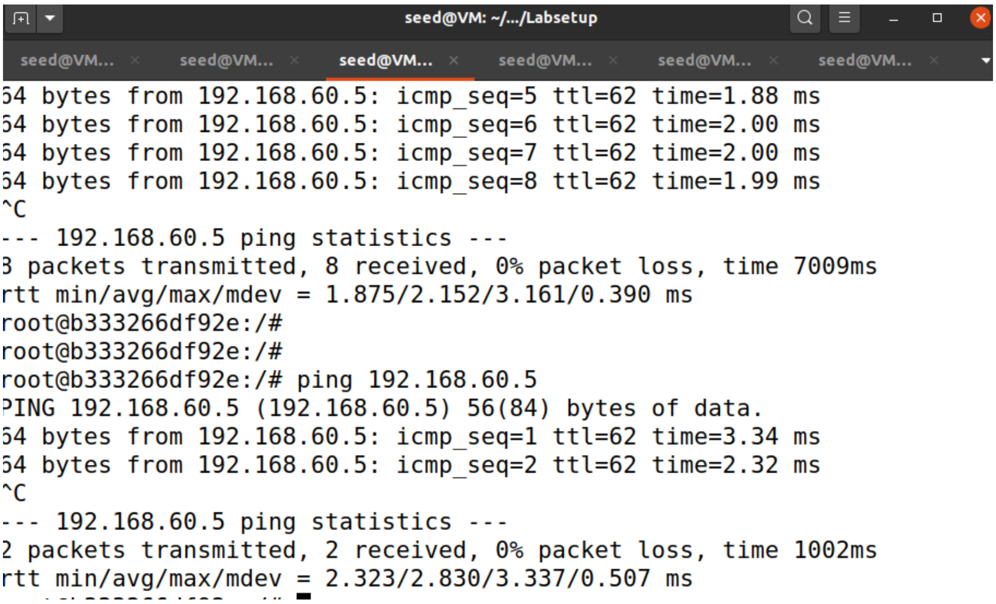
```
IFF_TAP   = 0x0002
IFF_NO_PI = 0x1000

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

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

os.system("ip addr add 192.168.53.11/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

SERVER_PORT = 9090
SERVER_IP = "10.9.0.11"

ip = "10.9.0.11"
port = 1030

os.system("ip route add 192.168.60.0/24 dev {}".format(ifname))

# Create UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
                                                      30,11          34%
```

Server side:

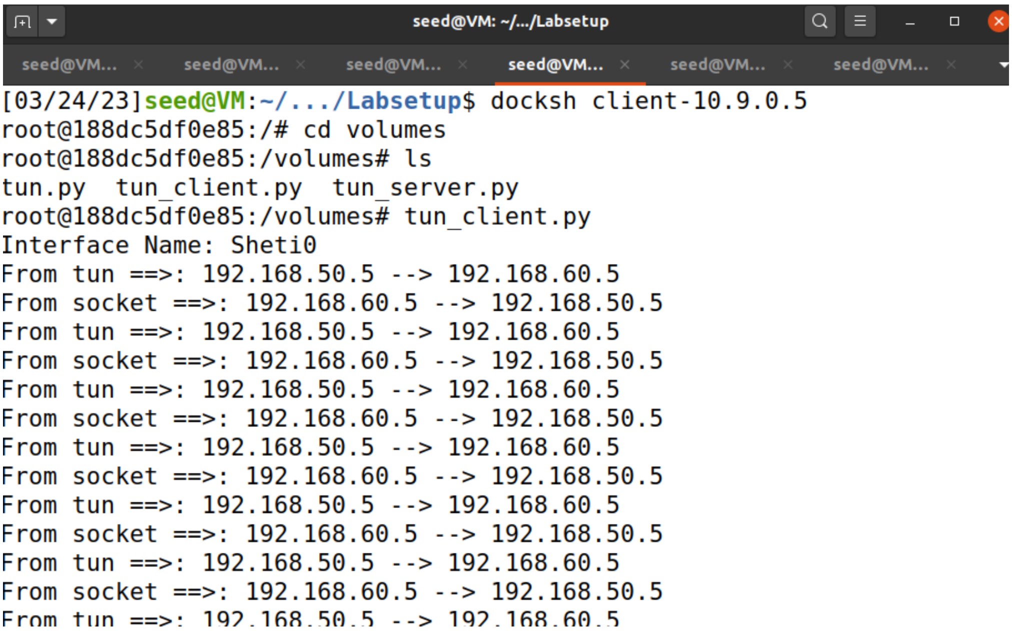


```
IFF_TUN   = 0x0001
IFF_TAP   = 0x0002
IFF_NO_PI = 0x1000

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

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

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

os.system("ip route add 192.168.50.0/24 dev {}".format(ifname))
IP_A = "0.0.0.0"
PORT = 9090
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))

ip = "10.9.0.5"
port = 1030
```

At Client side when we run the code while pinging 192.168.60.5, i.e from host U to Host V:

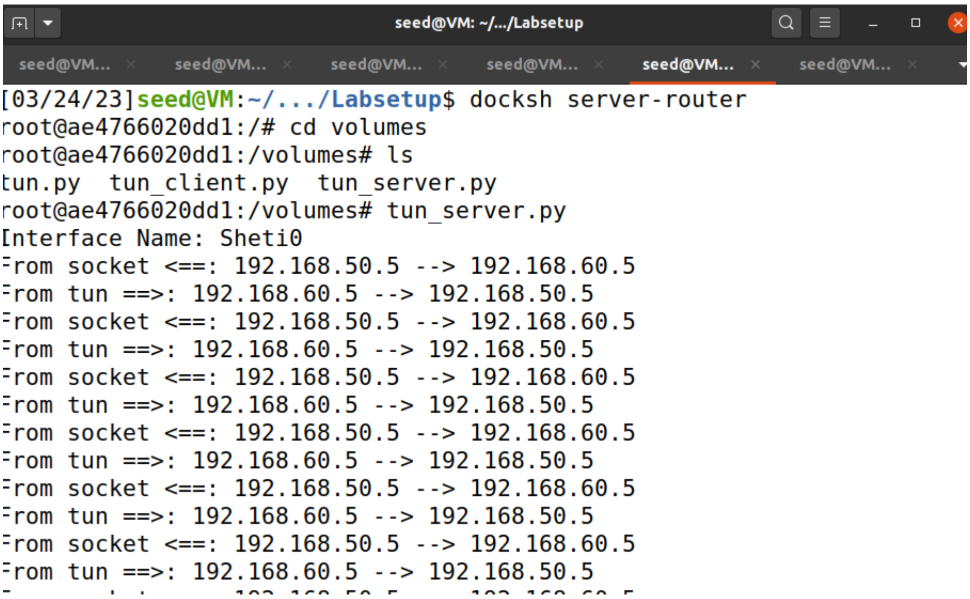


```
64 bytes from 192.168.60.5: icmp_seq=5 ttl=62 time=1.88 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=62 time=2.00 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=62 time=2.00 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=62 time=1.99 ms
^C
--- 192.168.60.5 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7009ms
rtt min/avg/max/mdev = 1.875/2.152/3.161/0.390 ms
root@b333266df92e:/#
root@b333266df92e:/#
root@b333266df92e:/# 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=62 time=3.34 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=62 time=2.32 ms
^C
--- 192.168.60.5 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 2.323/2.830/3.337/0.507 ms
```



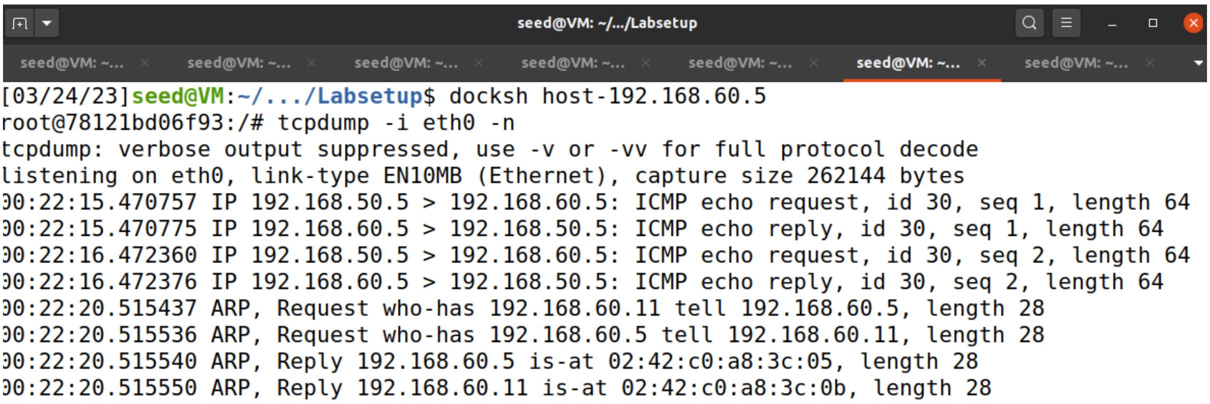
```
[03/24/23]seed@VM:~/.../Labsetup$ docksh client-10.9.0.5
root@188dc5df0e85:/# cd volumes
root@188dc5df0e85:/volumes# ls
tun.py  tun_client.py  tun_server.py
root@188dc5df0e85:/volumes# tun_client.py
Interface Name: Sheti0
From tun ==>: 192.168.50.5 --> 192.168.60.5
From socket ==>: 192.168.60.5 --> 192.168.50.5
From tun ==>: 192.168.50.5 --> 192.168.60.5
From socket ==>: 192.168.60.5 --> 192.168.50.5
From tun ==>: 192.168.50.5 --> 192.168.60.5
From socket ==>: 192.168.60.5 --> 192.168.50.5
From tun ==>: 192.168.50.5 --> 192.168.60.5
From socket ==>: 192.168.60.5 --> 192.168.50.5
From tun ==>: 192.168.50.5 --> 192.168.60.5
From socket ==>: 192.168.60.5 --> 192.168.50.5
From tun ==>: 192.168.50.5 --> 192.168.60.5
From socket ==>: 192.168.60.5 --> 192.168.50.5
From tun ==>: 192.168.50.5 --> 192.168.60.5
```

At server side:



```
[03/24/23]seed@VM:~/.../Labsetup$ docksh server-router
root@ae4766020dd1:/# cd volumes
root@ae4766020dd1:/volumes# ls
tun.py  tun_client.py  tun_server.py
root@ae4766020dd1:/volumes# tun_server.py
Interface Name: Sheti0
From socket <==: 192.168.50.5 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.50.5
From socket <==: 192.168.50.5 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.50.5
From socket <==: 192.168.50.5 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.50.5
From socket <==: 192.168.50.5 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.50.5
From socket <==: 192.168.50.5 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.50.5
From socket <==: 192.168.50.5 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.50.5
```

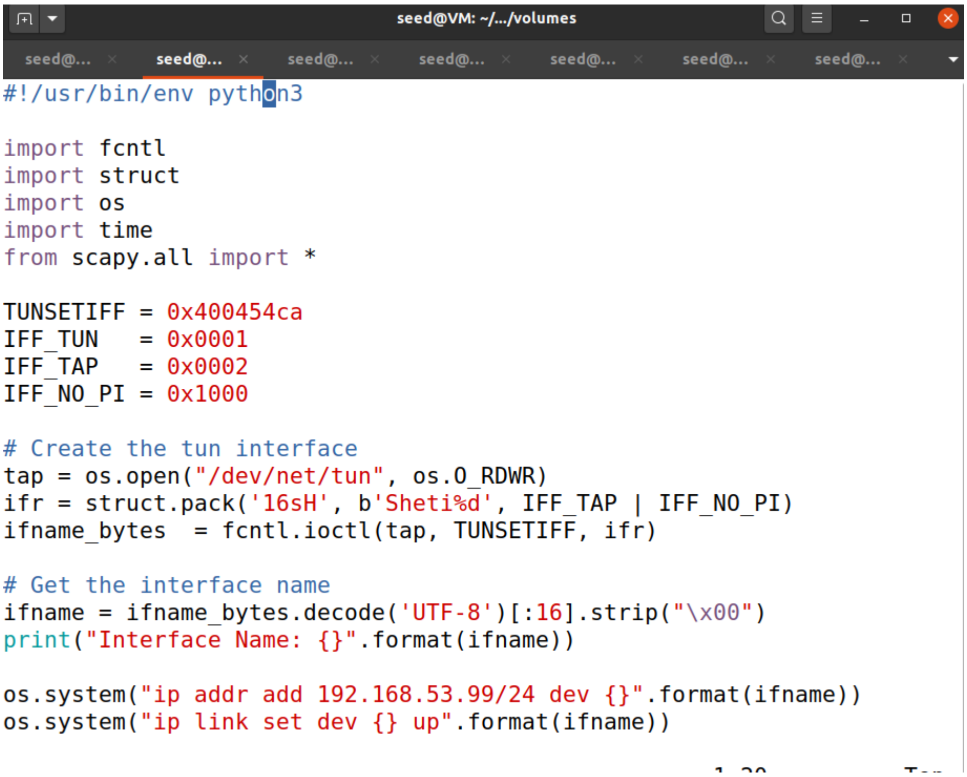
By running tcpdump at 192.168.60.5 we can see that a proper transmission and reception of icmp packet has taken place:



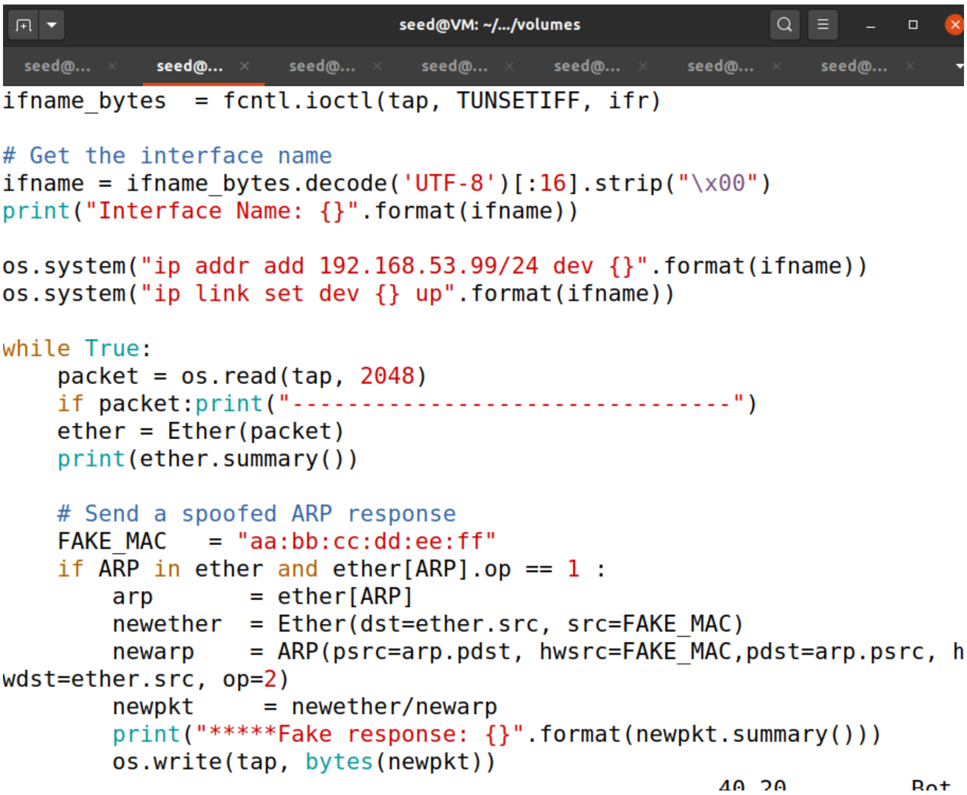
```
[03/24/23]seed@VM:~/.../Labsetup$ docksh host-192.168.60.5
root@78121bd06f93:/# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
00:22:15.470757 IP 192.168.50.5 > 192.168.60.5: ICMP echo request, id 30, seq 1, length 64
00:22:15.470775 IP 192.168.60.5 > 192.168.50.5: ICMP echo reply, id 30, seq 1, length 64
00:22:16.472360 IP 192.168.50.5 > 192.168.60.5: ICMP echo request, id 30, seq 2, length 64
00:22:16.472376 IP 192.168.60.5 > 192.168.50.5: ICMP echo reply, id 30, seq 2, length 64
00:22:20.515437 ARP, Request who-has 192.168.60.11 tell 192.168.60.5, length 28
00:22:20.515536 ARP, Request who-has 192.168.60.5 tell 192.168.60.11, length 28
00:22:20.515540 ARP, Reply 192.168.60.5 is-at 02:42:c0:a8:3c:05, length 28
00:22:20.515550 ARP, Reply 192.168.60.11 is-at 02:42:c0:a8:3c:0b, length 28
```

Hence, looking at the outputs it can be said that this task was successful.

TASK 9:



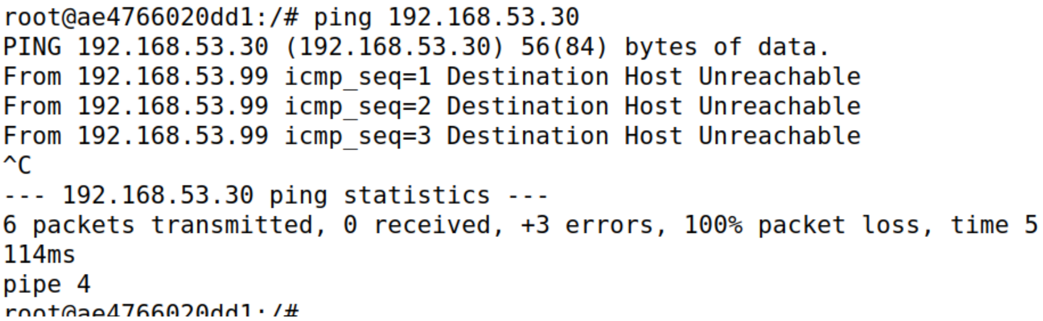
```
#!/usr/bin/env python3

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
tap = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'Sheti%d', IFF_TAP | IFF_NO_PI)
ifname_bytes  = fcntl.ioctl(tap, TUNSETIFF, ifr)

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

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



```
ifname_bytes  = fcntl.ioctl(tap, TUNSETIFF, ifr)

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

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

while True:
    packet = os.read(tap, 2048)
    if packet:print("---------------------------------")
    ether = Ether(packet)
    print(ether.summary())

    # Send a spoofed ARP response
    FAKE_MAC    = "aa:bb:cc:dd:ee:ff"
    if ARP in ether and ether[ARP].op == 1 :
        arp       = ether[ARP]
        newether  = Ether(dst=ether.src, src=FAKE_MAC)
        newarp    = ARP(psrc=arp.pdst, hwsrc=FAKE_MAC,pdst=arp.psrc, h
wdst=ether.src, op=2)
        newpkt      = newether/newarp
        print("*****Fake response: {}".format(newpkt.summary()))
        os.write(tap, bytes(newpkt))
```

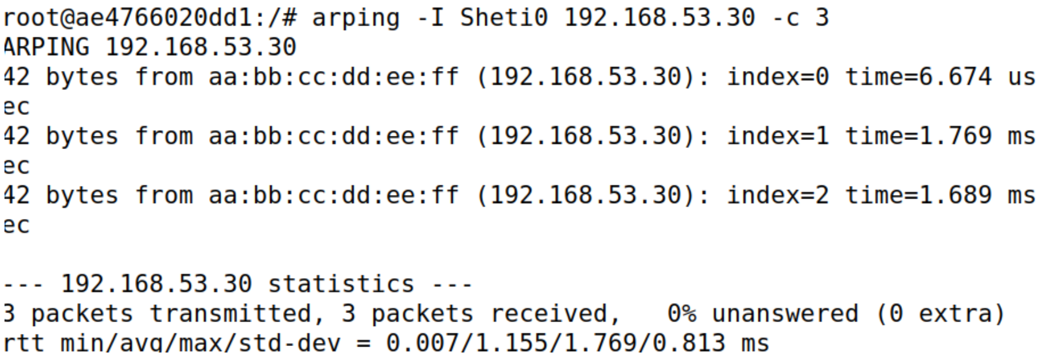
We did try to ping IP 192.168.53.30 but the interface routes the packets to the tunnel interface which doesn’t know that ip address belongs to whom and thereby sends a host unreachable text.

This screenshot is before we add the ARP spoof code snippet to our code:

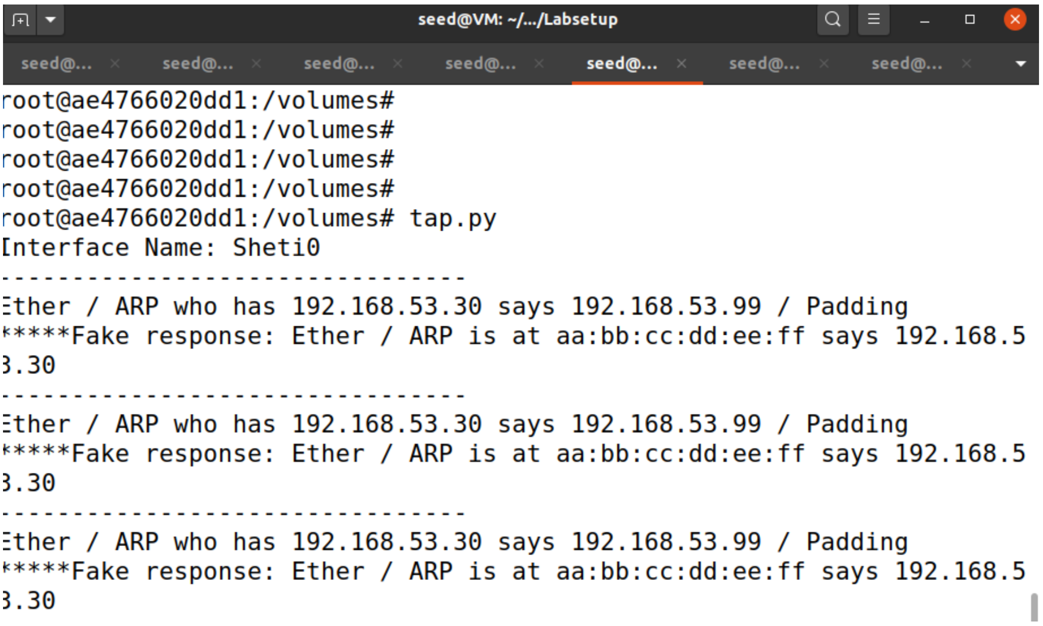


```
root@ae4766020dd1:/# ping 192.168.53.30
PING 192.168.53.30 (192.168.53.30) 56(84) bytes of data.
From 192.168.53.99 icmp_seq=1 Destination Host Unreachable
From 192.168.53.99 icmp_seq=2 Destination Host Unreachable
From 192.168.53.99 icmp_seq=3 Destination Host Unreachable
^C
--- 192.168.53.30 ping statistics ---
6 packets transmitted, 0 received, +3 errors, 100% packet loss, time 5
114ms
pipe 4
root@ae4766020dd1:/#
```

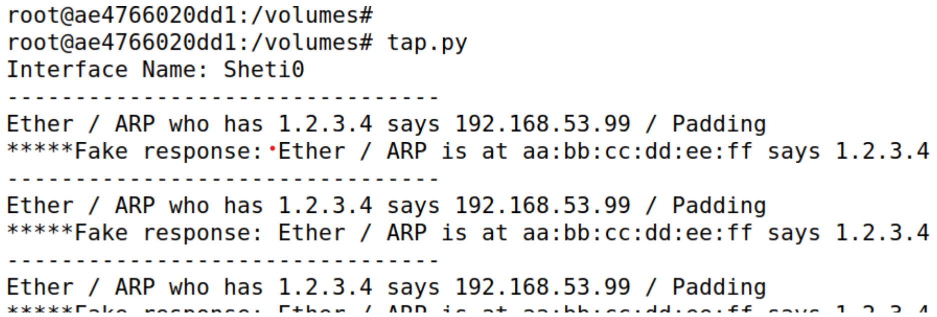
After we add the ARP spoof code, we modify the code to send ARP packtes from a fake MAC address :



```
root@ae4766020dd1:/# arping -I Sheti0 192.168.53.30 -c 3
ARPING 192.168.53.30
42 bytes from aa:bb:cc:dd:ee:ff (192.168.53.30): index=0 time=6.674 us
ec
42 bytes from aa:bb:cc:dd:ee:ff (192.168.53.30): index=1 time=1.769 ms
ec
42 bytes from aa:bb:cc:dd:ee:ff (192.168.53.30): index=2 time=1.689 ms
ec

--- 192.168.53.30 statistics ---
3 packets transmitted, 3 packets received,   0% unanswered (0 extra)
rtt min/avg/max/std-dev = 0.007/1.155/1.769/0.813 ms
```



```
root@ae4766020dd1:/volumes#
root@ae4766020dd1:/volumes#
root@ae4766020dd1:/volumes#
root@ae4766020dd1:/volumes#
root@ae4766020dd1:/volumes# tap.py
Interface Name: Sheti0
--------------------------------
Ether / ARP who has 192.168.53.30 says 192.168.53.99 / Padding
*****Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 192.168.5
3.30
--------------------------------
Ether / ARP who has 192.168.53.30 says 192.168.53.99 / Padding
*****Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 192.168.5
3.30
--------------------------------
Ether / ARP who has 192.168.53.30 says 192.168.53.99 / Padding
*****Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 192.168.5
3.30
```



```
root@ae4766020dd1:/volumes#
root@ae4766020dd1:/volumes# tap.py
Interface Name: Sheti0
--------------------------------
Ether / ARP who has 1.2.3.4 says 192.168.53.99 / Padding
*****Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 1.2.3.4
--------------------------------
Ether / ARP who has 1.2.3.4 says 192.168.53.99 / Padding
*****Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 1.2.3.4
--------------------------------
Ether / ARP who has 1.2.3.4 says 192.168.53.99 / Padding
*****Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 1.2.3.4
```

We ran the code and tried to arping 192.168.53.30 and we did receive message replies from the MAC sent for the request.