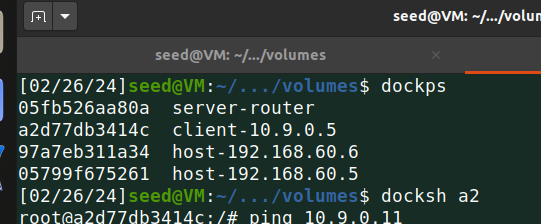
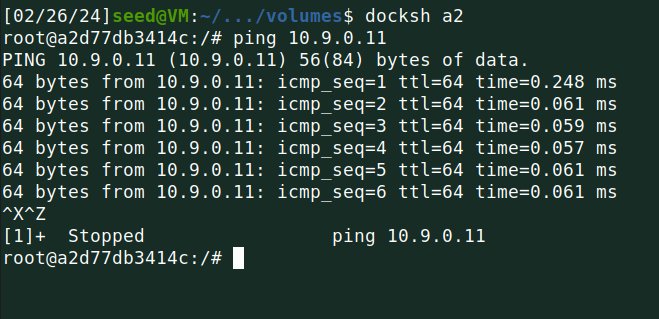
**Lab 6: VPN Lab: The Container Version**

**Task 1: Lab Setup**

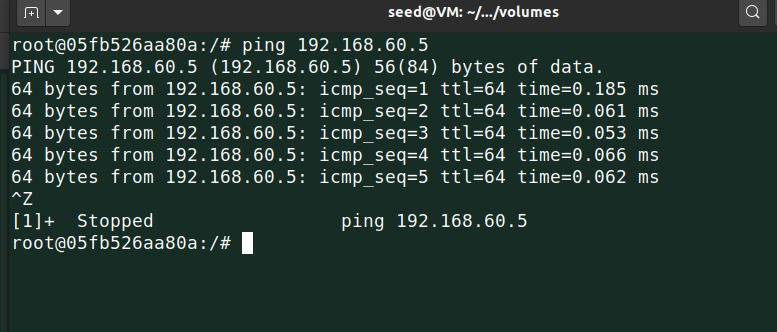
In this task, I have to do a lab setup, I started by docker containers and now I have to test whether the connection is correct.



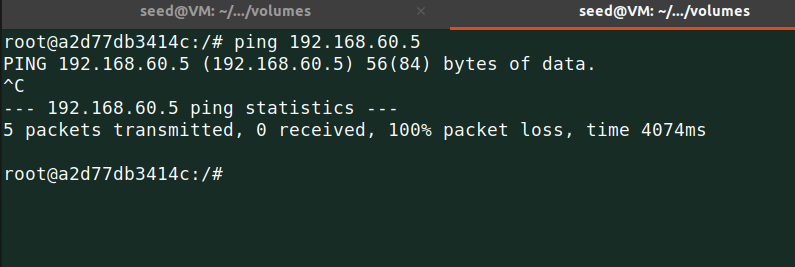
1. Host U is communicating with the VPN Server - checked by pinging to 10.9.0.11 from Host U (10.9.0.5)



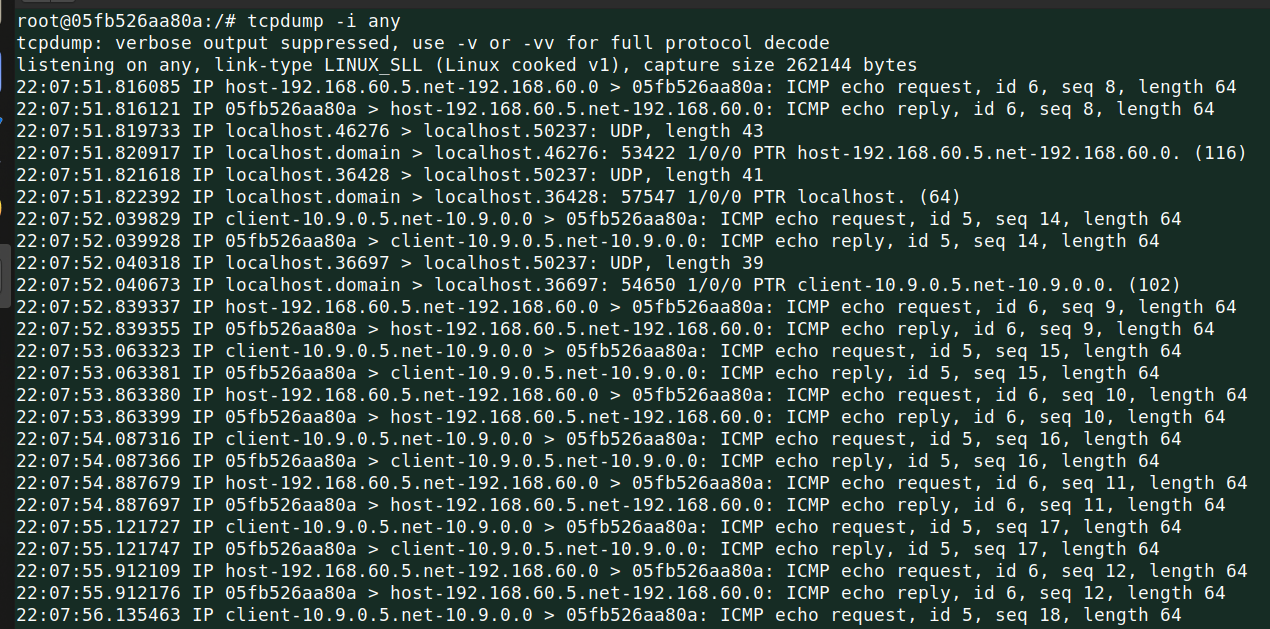
1. VPN Server can communicate with Host V - checked by pinging to 192.168.60.5 from VPN Server



1. Host U is not connecting to Host V which is as expected



1. After running ***tcpdump*** on the router, I sniffed the traffic on the network,



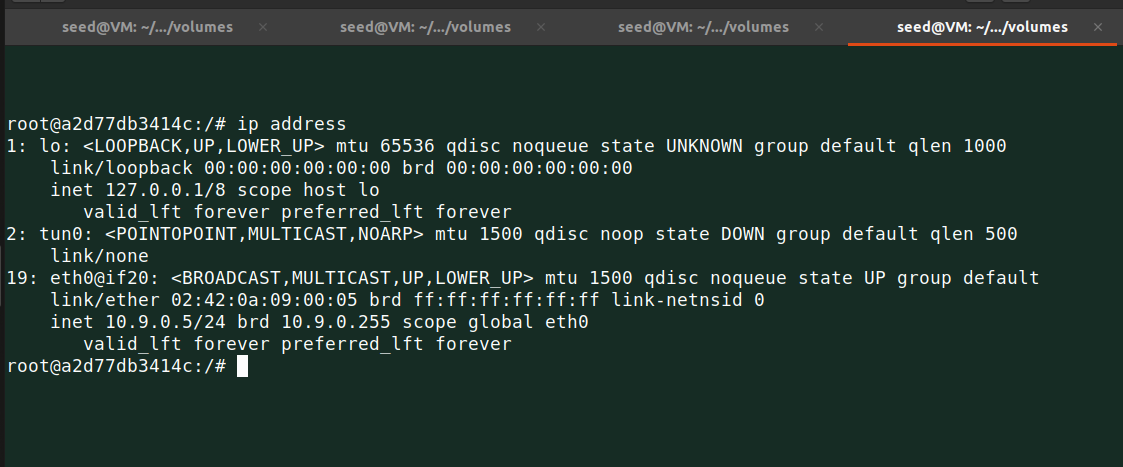
**Task 2: Create and Configure the TUN interface**

In this task, I am going to build a VPN tunnel using the TUN interface which is nothing but virtual kernel drivers.

**Task 2. a: Name of the interface**

I made it executable and ran the code.

Then using the command ***ip address*** I found that the tun0 interface was there.



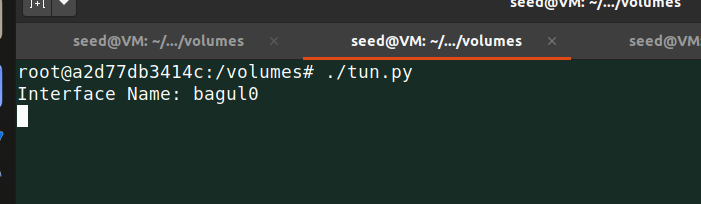
Now let’s do some fun and add my last name as the interface name.

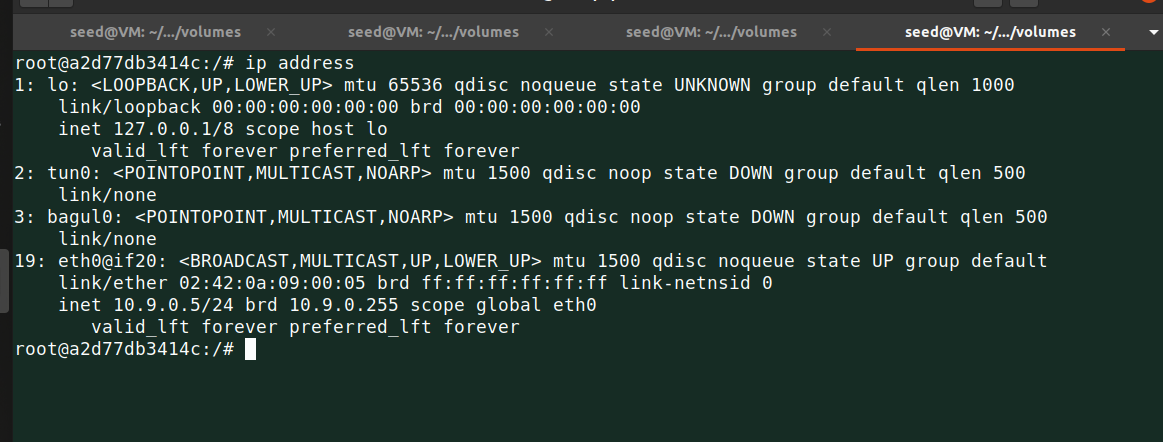
Below is the code

**Code: 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'bagul%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) |
| --- |

Ran the code and yes I found a new interface with my name.





**Task 2. b: Set up the TUN Interface**

Until now, the tun interface is not in use because configuration is still remaining.

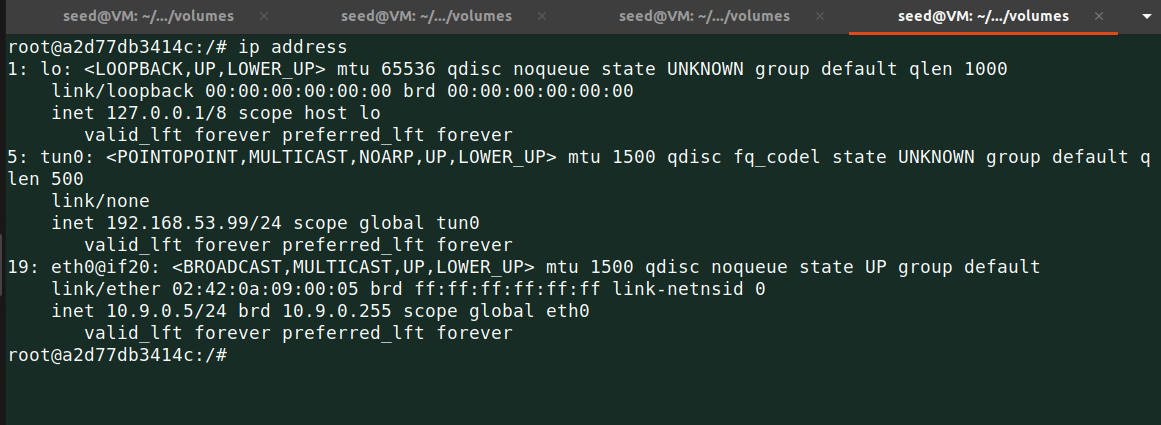
Let’s assign an IP address to it.

Below is the modified code:

**Code: 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'tun%d', IFF\_TUN | IFF\_NO\_PI) ifname\_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)  # Get the interface name ifname = ifname\_bytes.decode('UTF-8')[:16].strip("\x00") print("Interface Name: {}".format(ifname))  os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname)) os.system("ip link set dev {} up".format(ifname))  while True:  time.sleep(10) |
| --- |

Run the code and let’s see using ***ip address*** command what I got.



As I see an interface is assigned to IP address 192.168.53.99/24 and the state is UNKNOWN.

So using this task I was able to set up the interface.

**Task 2. c: Read from the TUN interface**

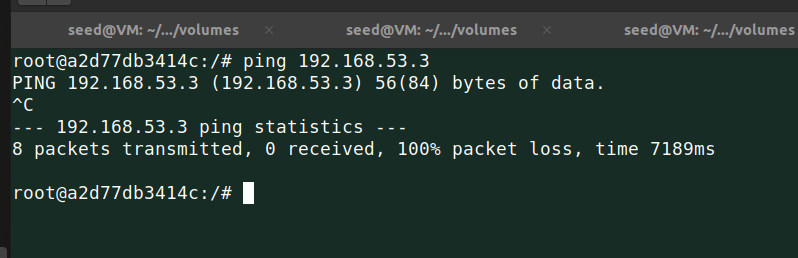
In this task, I modified the tun.py code to read the data from the TUN interface.

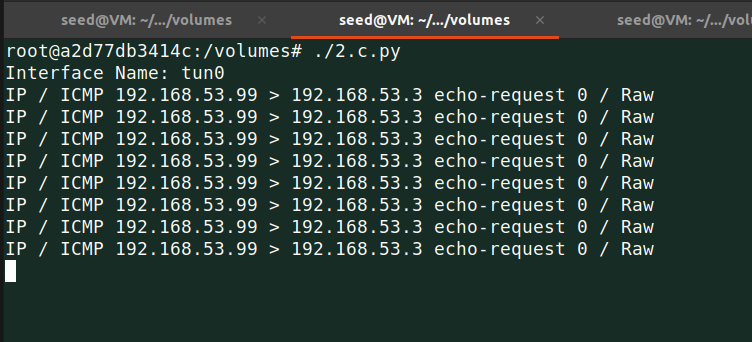
**Code: tun.py**

| #!/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'tun%d', IFF\_TUN | IFF\_NO\_PI) ifname\_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)  # Get the interface name ifname = ifname\_bytes.decode('UTF-8')[:16].strip("\x00") print("Interface Name: {}".format(ifname))  os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname)) os.system("ip link set dev {} up".format(ifname))  while True:  # Get a packet from the tun interface  packet = os.read(tun, 2048)  if packet:  ip = IP(packet)  print(ip.summary()) |
| --- |

I tested whether my new code is working or not using the below:

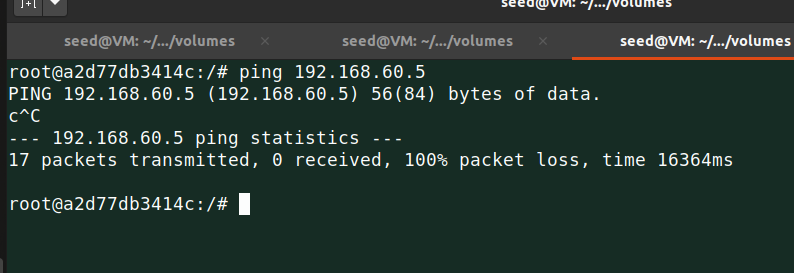
1. On Host U, I pinged to 192.168.53.3

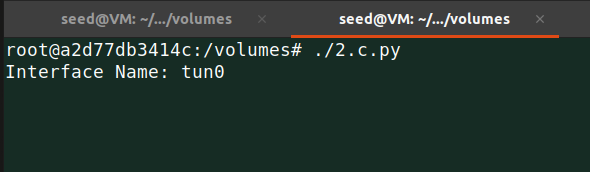




As I see TUN interface printed the packet because TUN interface is configured with IP address 192.168.53.99 and I pinged to IP address within that network.

1. On Host U, I pinged to 192.168.60.5





Hmm, As I expected it didn’t print anything because there is no full connection to 192.168.60.5.

**Task 2.d: Write to the TUN interface**

In this task, I will write to the TUN interface. I modified the tun.py code for this task.

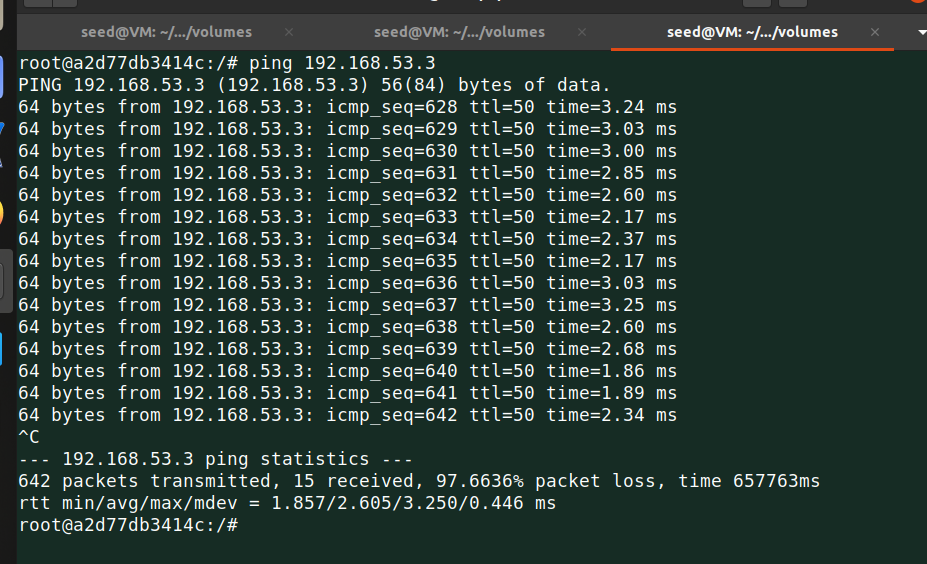
I have to check packet is an ICMP echo request or not if the packet is a request then I will create a reply packet and write it into the interface.

**Code: tun.py**

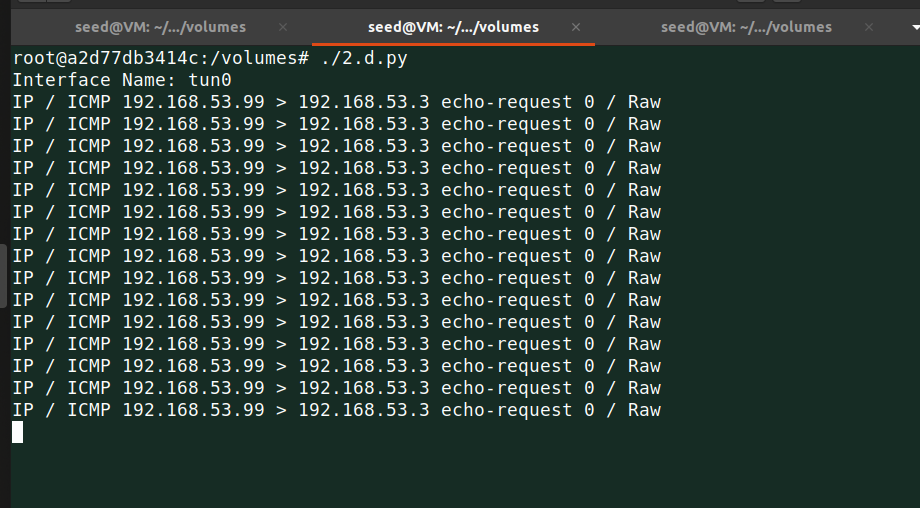
| #!/usr/bin/env python3  import fcntl import struct import os import time from scapy.all import \*  TUNSETIFF = 0x400454ca IFF\_TUN = 0x0001 IFF\_TAP = 0x0002 IFF\_NO\_PI = 0x1000  # Create the tun interface tun = os.open("/dev/net/tun", os.O\_RDWR) ifr = struct.pack('16sH', b'tun%d', IFF\_TUN | IFF\_NO\_PI) ifname\_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)  # Get the interface name ifname = ifname\_bytes.decode('UTF-8')[:16].strip("\x00") print("Interface Name: {}".format(ifname))  os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname)) os.system("ip link set dev {} up".format(ifname))  def new\_packet(pkt):  if ICMP in pkt and pkt[ICMP].type == 8:   ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl, ttl=50)  icmp = ICMP(id = pkt[ICMP].id,type = 0, seq = pkt[ICMP].seq )  data = pkt[Raw].load  newpkt = ip/icmp/data  os.write(tun, bytes(newpkt))  else:  data = b"INVALID"  os.write(tun, bytes(data))  while True:  # Get a packet from the tun interface  packet = os.read(tun, 2048)  if packet:  ip\_pkt = IP(packet)  print(ip\_pkt.summary())  new\_packet(ip\_pkt) |
| --- |

Now, I ran this code and ***tcpdump*** on the tun0 interface to check if it worked or not.

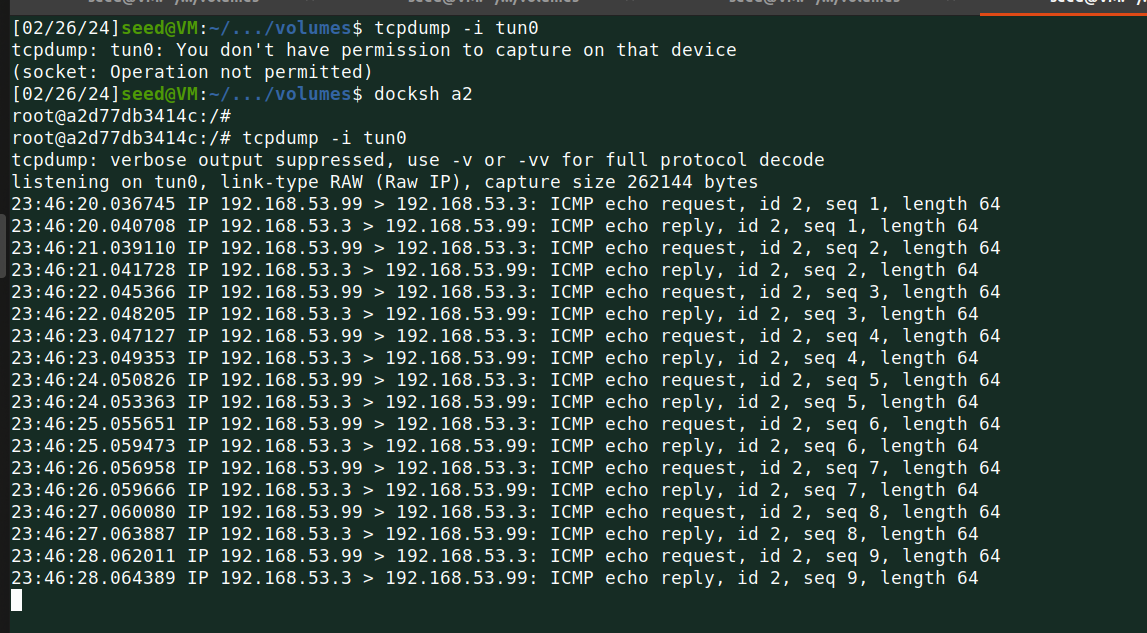
As I see ping command worked for 192.168.53.3 which was obvious.



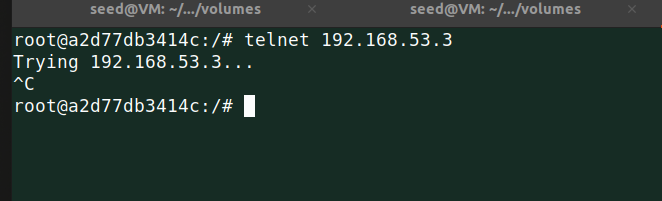
TUN interface printed below details, which shows all ICMP requests also it sends reply packet captured in tcpdump.

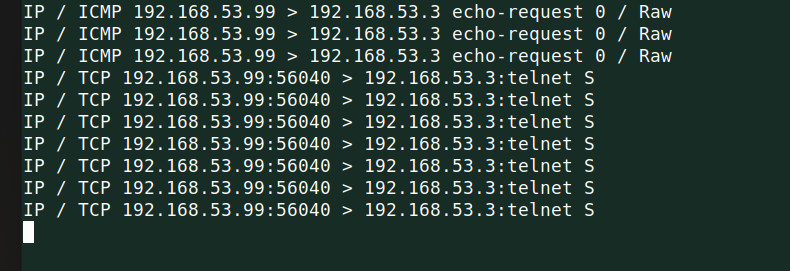


Below shows tcpdump captured information as we see for each request there is the reply.



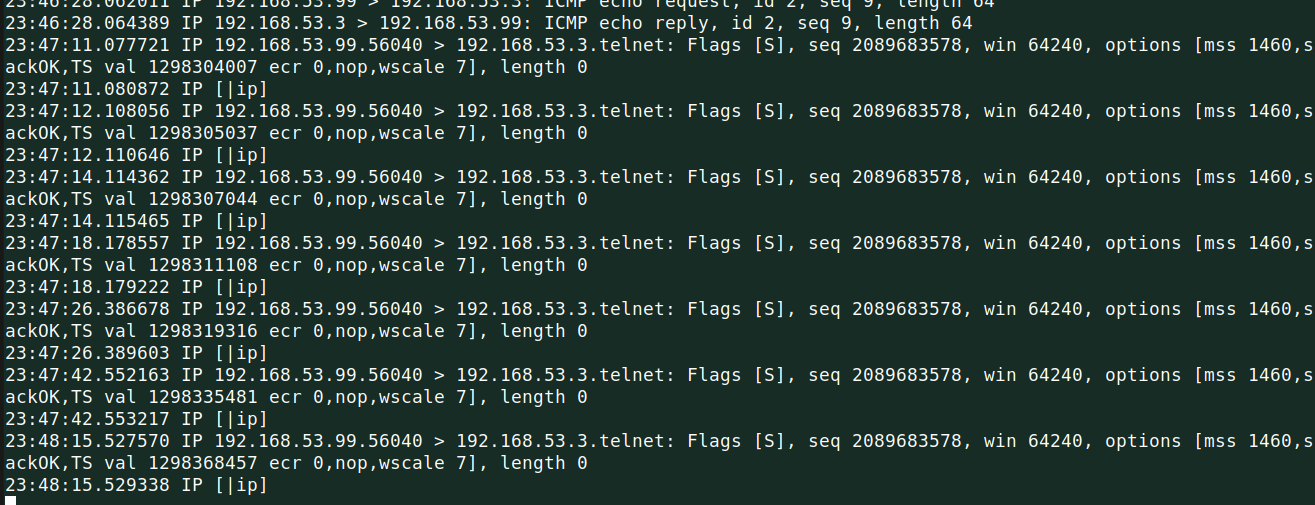
Now I tried to connect 192.168.53.3 which didn’t work.





Below are the tcpdump details.

This demonstrates that when Host U sends an IP request packet, which is not an ICMP packet, to the network, the tun0 interface intercepts it. This interception occurs because I have programmatically linked it to do so. Subsequently, it generates a response by sending back an IP packet containing the data we predefined in the code. The capture of this traffic can be observed in tcpdump, illustrating how the response packet is then transmitted through the interface.



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

In this task, I am going to do IP tunneling which is I will place the original packet inside a new packet. In detail, I will put the IP packet received from the TUN interface into the UDP payload field of the new IP packet.

**Code: tun\_server.py**

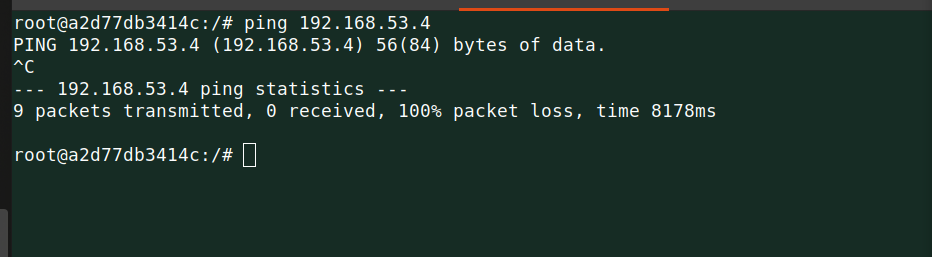
| #!/usr/bin/env python3  from scapy.all import \*  IP\_A = "0.0.0.0" PORT = 9090  sock =socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM) sock.bind((IP\_A,PORT))  while True:  data, (ip, port) = sock.recvfrom(2048)  print("{}:{} --> {}:{}".format(ip, port, IP\_A, PORT))  pkt = IP(data)  print(" Inside: {} -->".format(pkt.src, pkt.dst)) |
| --- |

**Code: tun\_client.py**

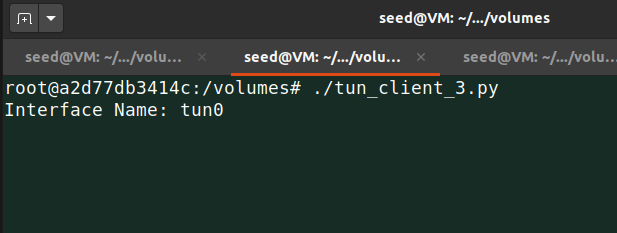
| #!/usr/bin/env python3  import fcntl import struct import os import time from scapy.all import \*  SERVER\_IP = "10.9.0.11" SERVER\_PORT = 9090  TUNSETIFF = 0x400454ca IFF\_TUN = 0x0001 IFF\_TAP = 0x0002 IFF\_NO\_PI = 0x1000  # Create the tun interface tun = os.open("/dev/net/tun", os.O\_RDWR) ifr = struct.pack('16sH', b'tun%d', IFF\_TUN | IFF\_NO\_PI) ifname\_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)  # Get the interface name ifname = ifname\_bytes.decode('UTF-8')[:16].strip("\x00") print("Interface Name: {}".format(ifname))  os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname)) os.system("ip link set dev {} up".format(ifname))  # Create UDP socket sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  while True:  # Get a packet from the tun interface  packet = os.read(tun, 2048)  if packet:  # Send the packet via the tunnel  sock.sendto(packet, (SERVER\_IP, SERVER\_PORT)) |
| --- |

Now to complete the task, I ran the tun\_server.py program on the VPN server and tun\_client.py on Host U.

And on Host U, I pinged 192.168.53.4 to check what happened.

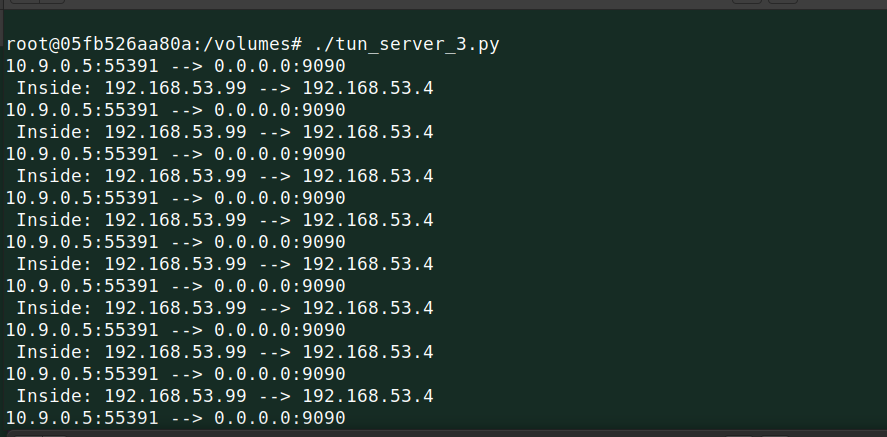


With that nothing is shown on the client side.

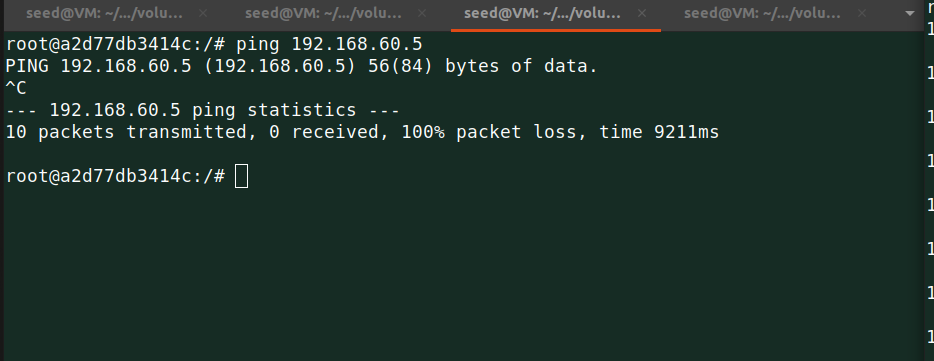


But on the server side, there are a lot of things captured or I can say printed.

This is because I already configured our network with 192.168.53.0/24.

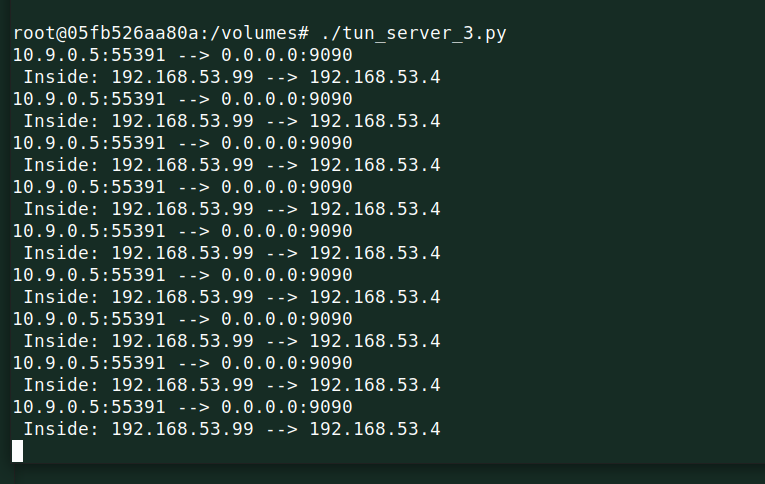


Now, let’s try to ping Host V and see what happens as I am expecting nothing will show as there is no full connection yet with Host V.



Below are the details from the server side, which shows nothing is printed.

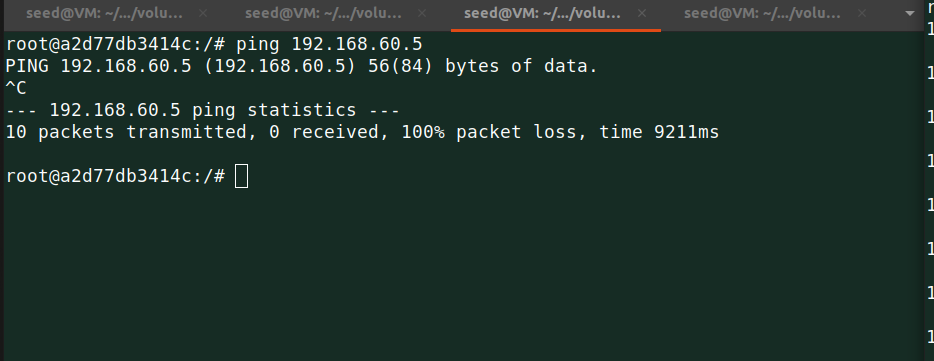
Also, nothing got printed on the client side.



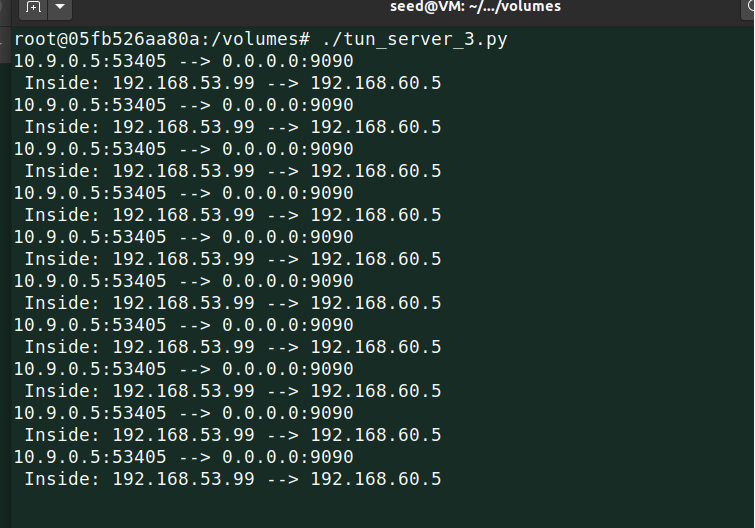
To work out this, I have to route the packets through the TUN interface to achieve this I have to add a new line in my tun\_client.py program which is below.

| # Set up routing os.system("ip route add 192.168.60.0/24 dev {}".format(ifname)) |
| --- |

Now let’s test this again, hope this time it will work.



As I dreamed of it worked, when I ping 192.168.60.5 the ICMP packets are received by the tun\_server.py program.



And with that, I was able to connect partially with Host V.

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

In this task, I have to complete my connection for that once tun\_server.py gets a packet from the tunnel, I need to feed the packet to the kernel so that the kernel can route that towards its final destination.

Below are the modified code

**Code: tun\_server.py**

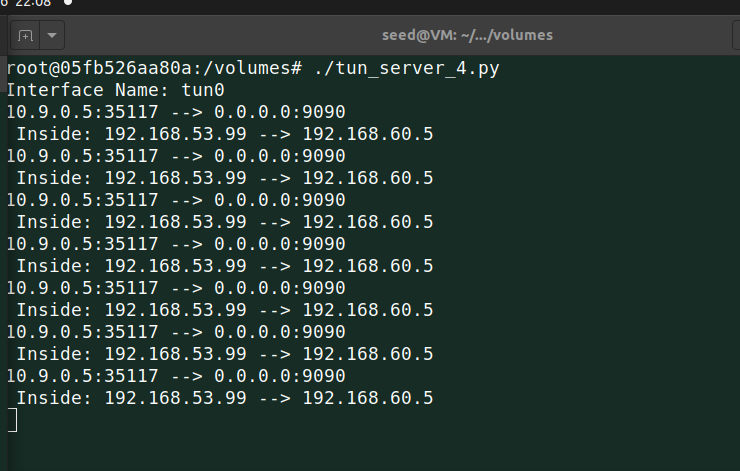
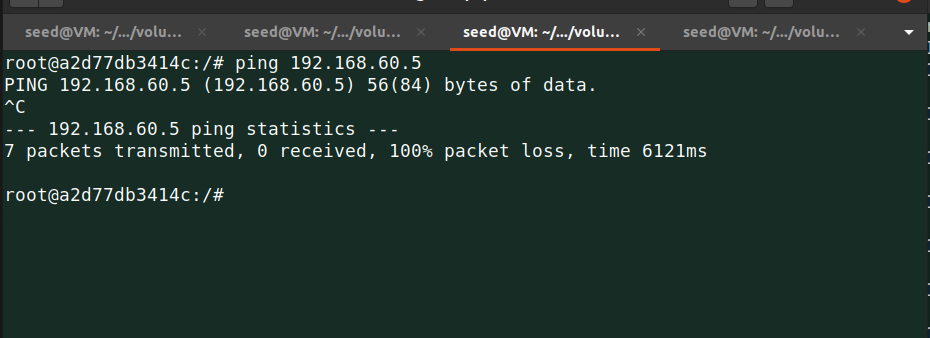
| #!/usr/bin/env python3  import fcntl import struct import os import time from scapy.all import \*  IP\_A = "0.0.0.0" PORT = 9090  TUNSETIFF = 0x400454ca IFF\_TUN = 0x0001 IFF\_TAP = 0x0002 IFF\_NO\_PI = 0x1000  # Create the tun interface tun = os.open("/dev/net/tun", os.O\_RDWR) ifr = struct.pack('16sH', b'tun%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 192.168.53.1/24 dev {}".format(ifname)) os.system("ip link set dev {} up".format(ifname))  sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM) sock.bind((IP\_A, PORT))  while True:  data, (ip, port) = sock.recvfrom(2048)  print("{}:{} --> {}:{}".format(ip, port, IP\_A, PORT))  pkt = IP(data)  print(" Inside: {} --> {}".format(pkt.src, pkt.dst))  os.write(tun, data) |
| --- |

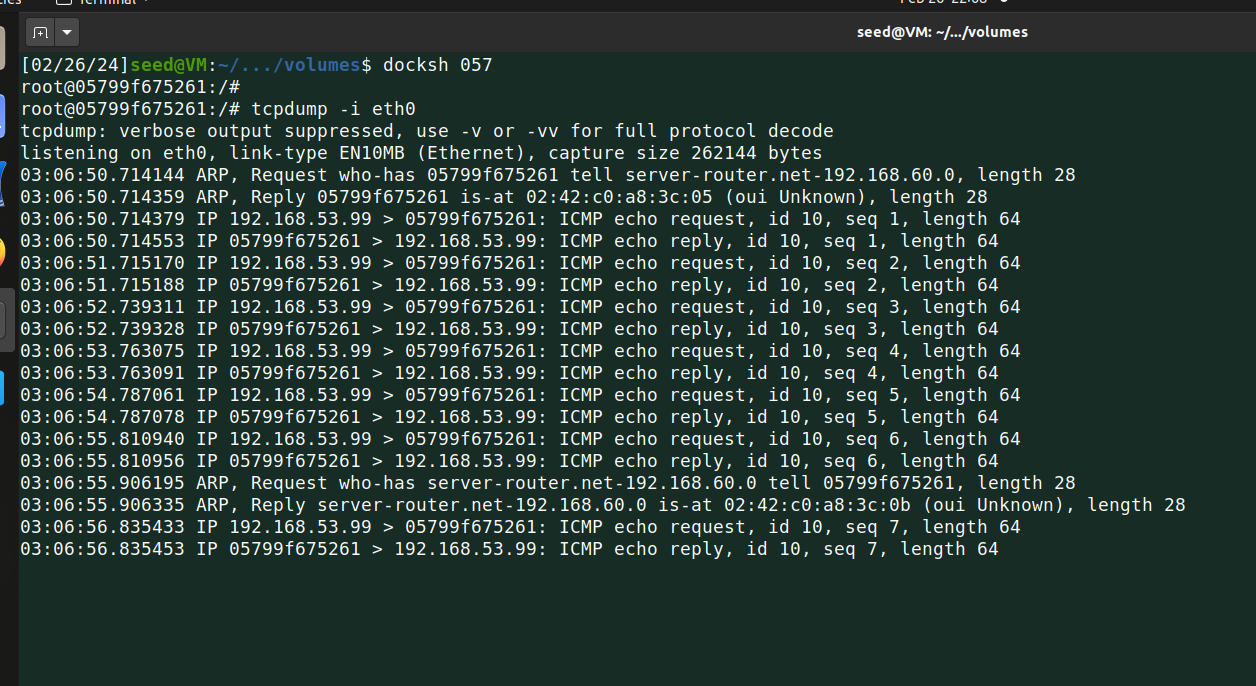
I ran the client code on Host U and the server code on the server side.

Let’s check the details, of what I got.

Unfortunately, the ping failed.

Because still, I have to set up some parts, anyway it is reaching Host V but the reply from Host V is not coming to Host U.





**Task 5: Handling Traffic in both directions.**

Now at this point, one direction of my tunnel is complete which is I am able to send packets from Host U to Host V via the tunnel.

To achieve that my TUN client program and server program need to read data from two interfaces, the TUN interface and the socket interface.

Below are the modified codes.

**Code: tun\_server.py**

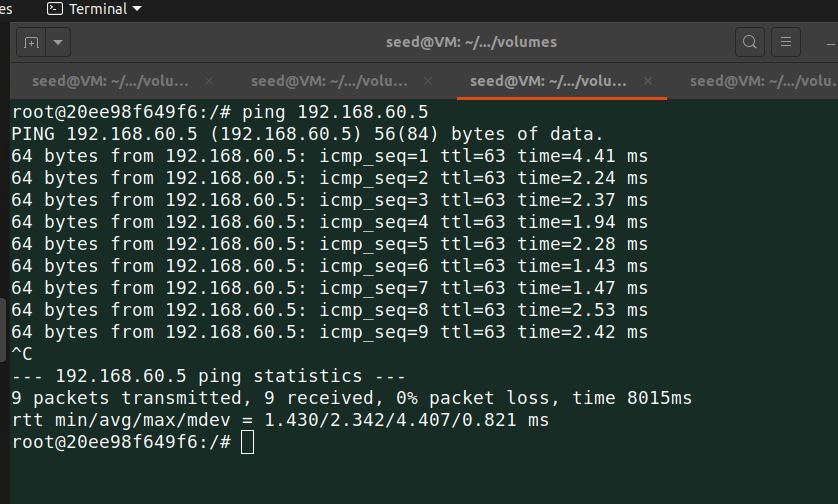
| #!/usr/bin/env python3  import fcntl import struct import os import time from scapy.all import \*  IP\_A = "0.0.0.0" PORT = 9090  TUNSETIFF = 0x400454ca IFF\_TUN = 0x0001 IFF\_TAP = 0x0002 IFF\_NO\_PI = 0x1000  # Create the tun interface tun = os.open("/dev/net/tun", os.O\_RDWR) ifr = struct.pack('16sH', b'tun%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 192.168.53.1/24 dev {}".format(ifname)) os.system("ip link set dev {} up".format(ifname))  sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM) sock.bind((IP\_A, PORT))  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:  print("sock ...")  data, (ip, port) = sock.recvfrom(2048)  pkt = IP(data)  print("{}:{} --> {}:{}".format(ip, port, IP\_A, PORT))  print(" Inside Tunnel: {} --> {}".format(pkt.src, pkt.dst))  os.write(tun, data)   if fd is tun:  print("tun ...")  packet = os.read(tun, 2048)  pkt = IP(packet)  print("Return: {} --> {}".format(pkt.src, pkt.dst))  sock.sendto(packet, (ip, port)) |
| --- |

**Code: tun\_client.py**

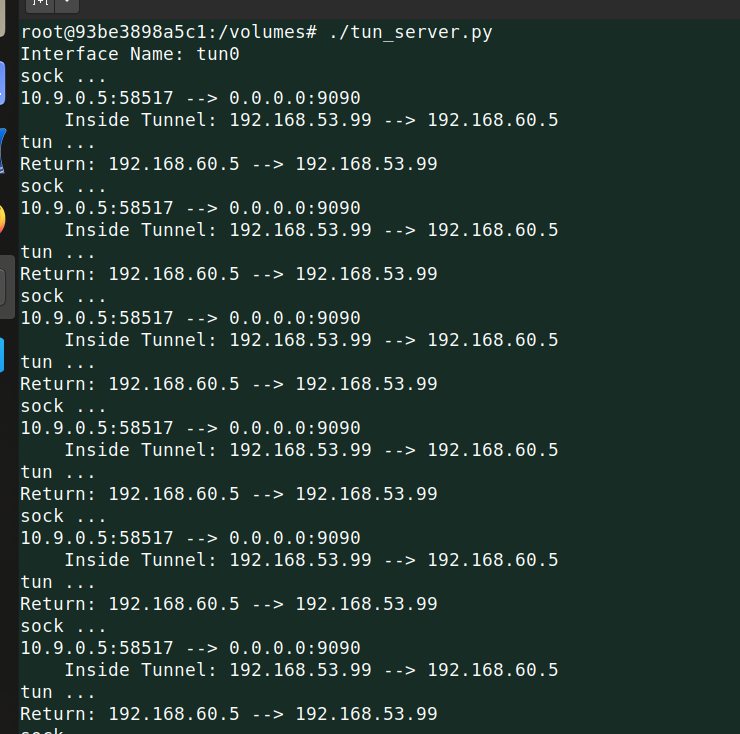
| #!/usr/bin/env python3  import fcntl import struct import os import time from scapy.all import \*  SERVER\_IP = "10.9.0.11" SERVER\_PORT = 9090  TUNSETIFF = 0x400454ca IFF\_TUN = 0x0001 IFF\_TAP = 0x0002 IFF\_NO\_PI = 0x1000  # Create the tun interface tun = os.open("/dev/net/tun", os.O\_RDWR) ifr = struct.pack('16sH', b'tun%d', IFF\_TUN | IFF\_NO\_PI) ifname\_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)  # Get the interface name ifname = ifname\_bytes.decode('UTF-8')[:16].strip("\x00") print("Interface Name: {}".format(ifname))  os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname)) os.system("ip link set dev {} up".format(ifname))  # Set up routing 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)  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, (SERVER\_IP, SERVER\_PORT)) |
| --- |

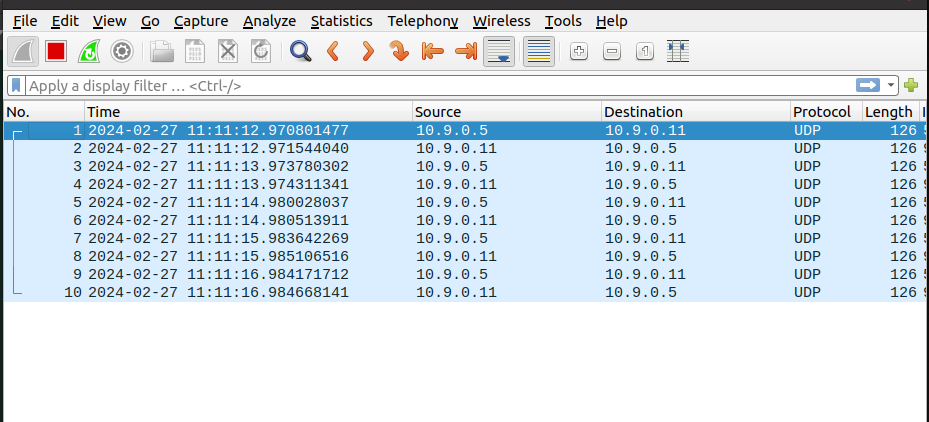
I ran both codes client on Host U and server on the server side.

First I ping Host V and recorded traffic using Wireshark.

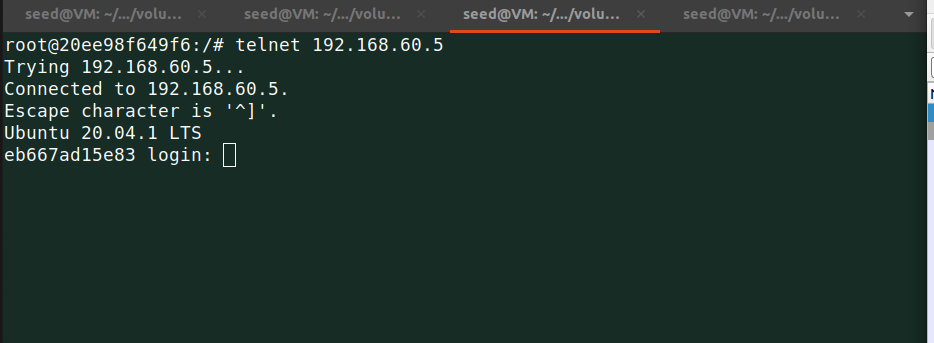


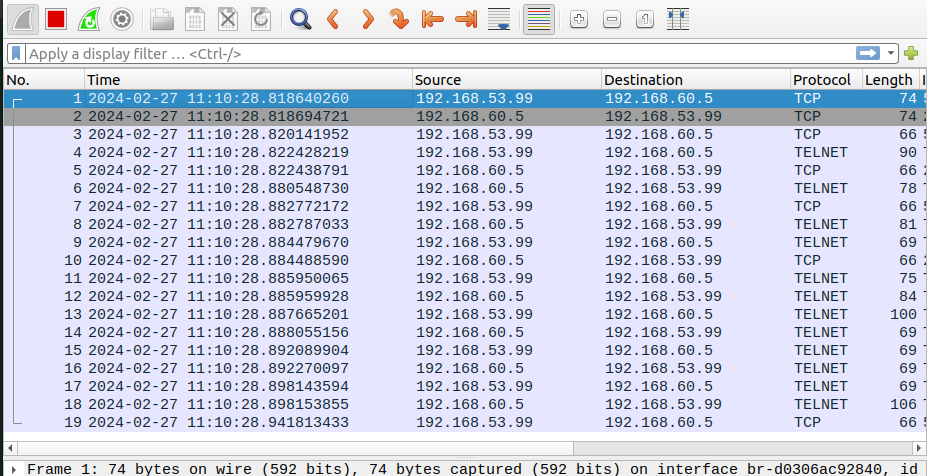
Below is captured on the server-side while ping operation.



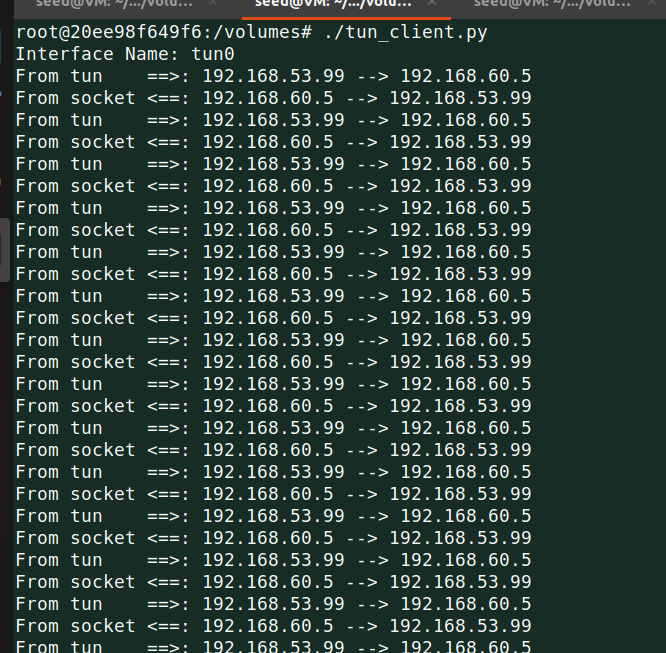


Below are the details of telnet.





Below is captured on the client side.

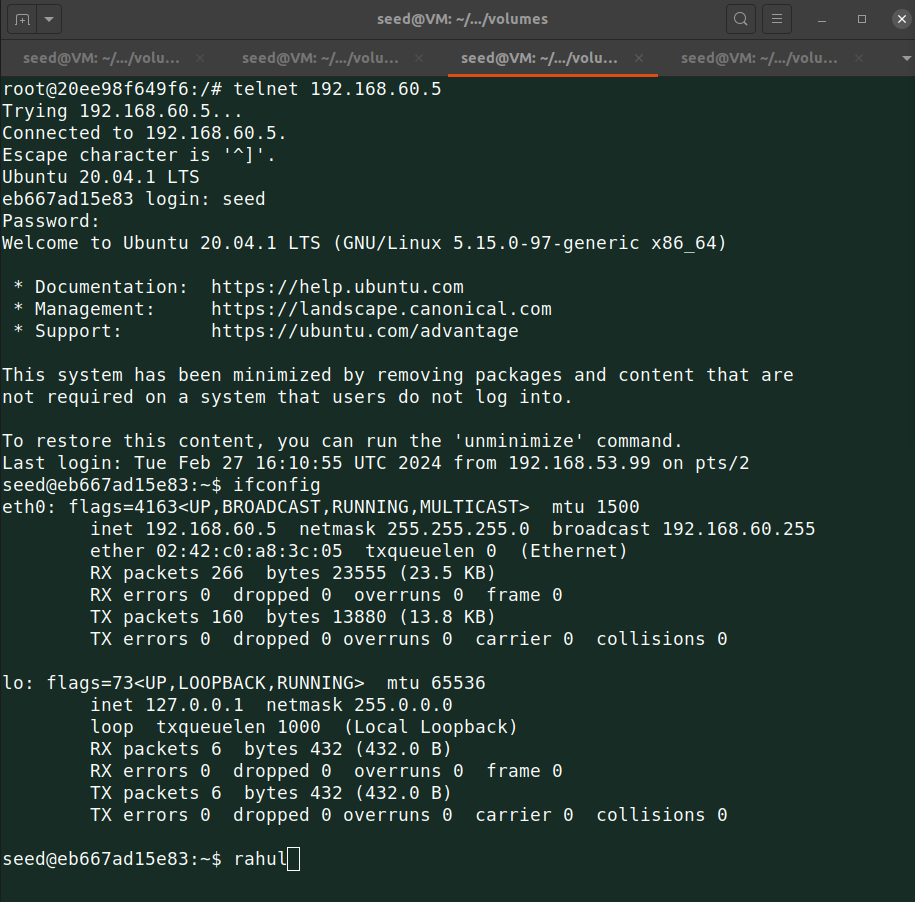


Now, everything is working end to end as we received a packet from Host V to Host U.

**Task 6: Tunnel Breaking Experiment**

In this task, I ran the client and server programs and then from Host U started connection using telnet to Host V and then stopped the server program for some seconds, in that time I typed some commands on Host U which didn’t show on the terminal but as soon as I started server program again my typed command shown on the terminal.

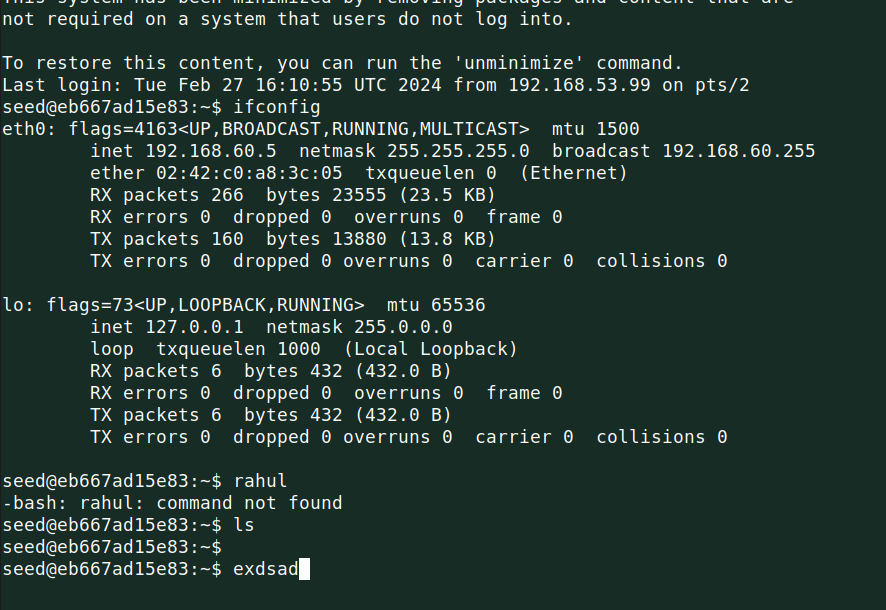
Below shows, that on Host U started the connection for Host V and I typed the command ***ifconfig*** and typed my name.



Then I stopped the server program.

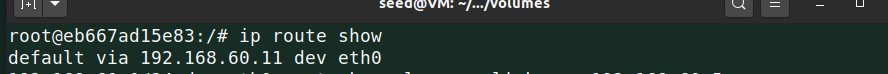


When I stopped the code, I typed random characters on the terminal, and then I ran the server code again and those random characters showed on the terminal.

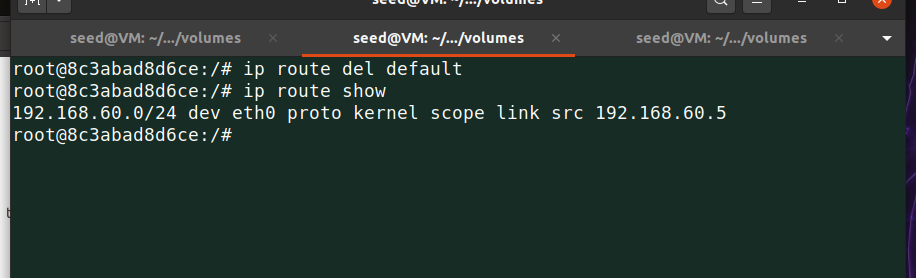


**Task 7: Routing Experiment on Host V**

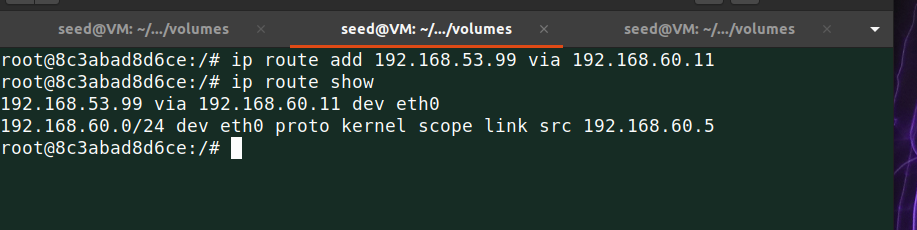
In this task, I deleted the default route from the Host V routing table and added a custom route 192.168.53.99 via 192.168.60.11.



Deleted default route.



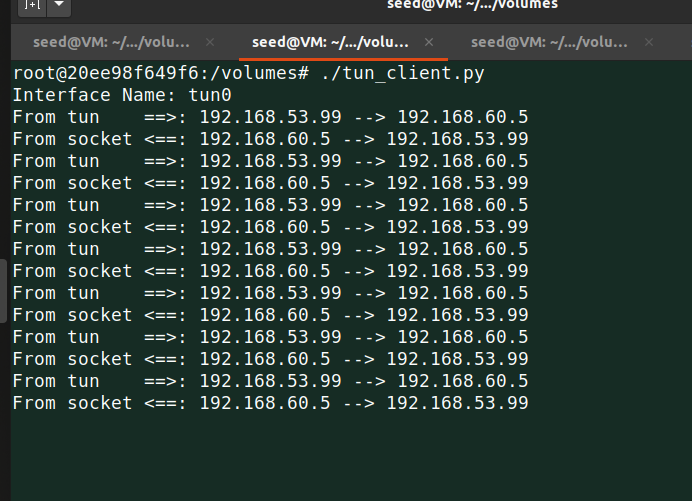
Added new route in the routing table.

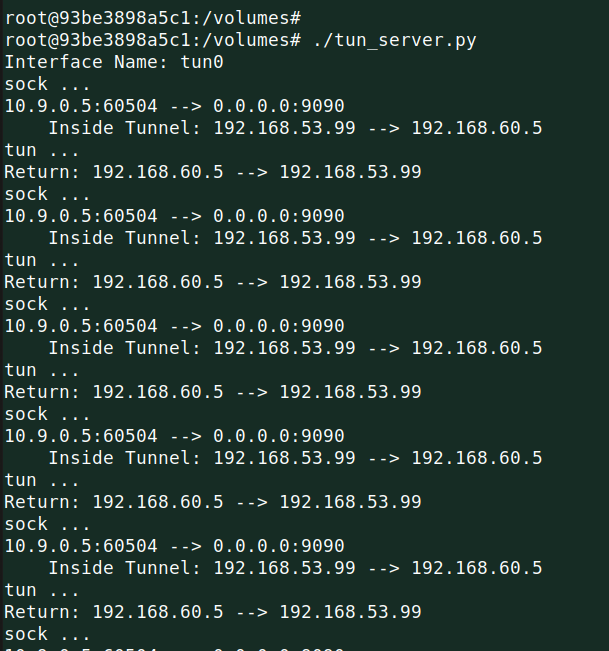


Ran the client and server-side code to check whether it’s working or not.

For this, I ping 192.168.60.5 from Host U.

Below are the details from the client and server side for ping operation.





And thus after deleting default route and then adding new custom one in routing table, I was able to perform ping from Host U to Host V without any issue.