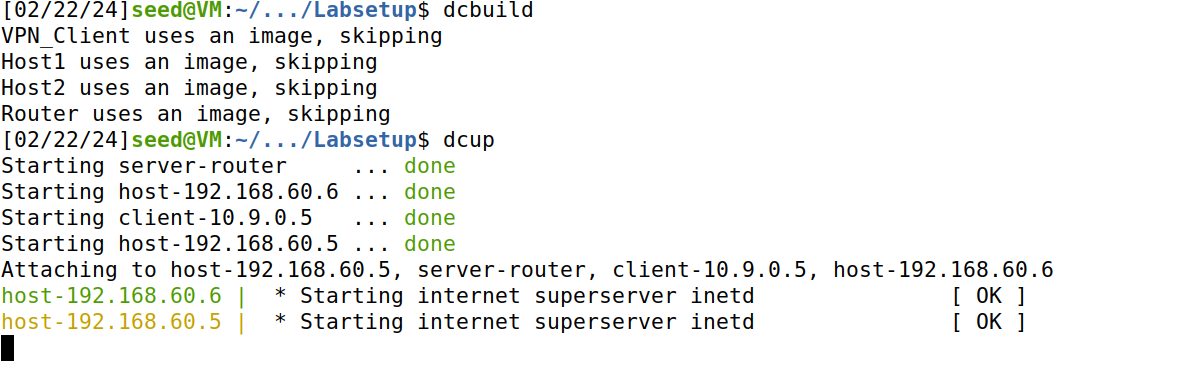
ACS 54500 Cryptography and Network Security

Lab 6: VPN Lab: The Container Version

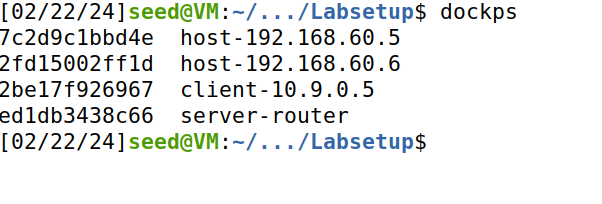
Name: Vijayagiridharan Subramanian

**Task 1 (Network Setup) – 10 pts**

We initialize the network setup by downloading the files and going inside the Lab Setup, We build docker containers, and start the container.

****

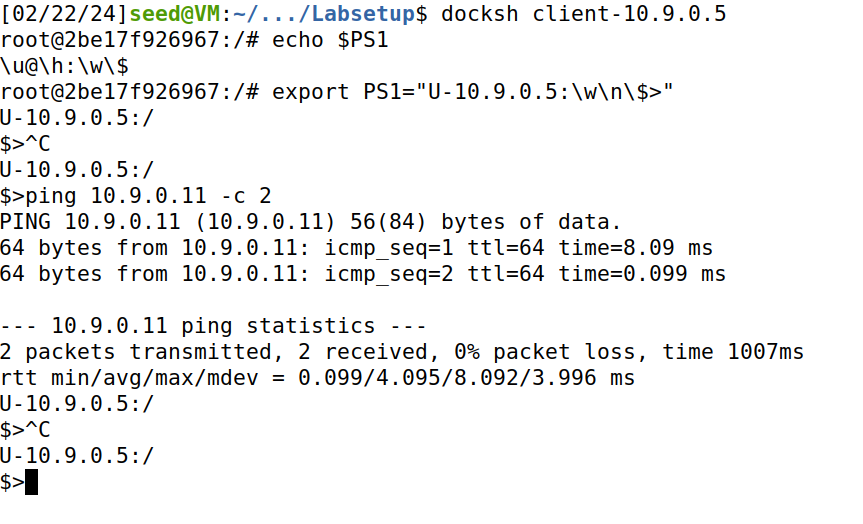
We run the dockps command which shows 2 hosts container, 1 client and 1 server-router containers

****

**-Host U can communicate with VPN Server.**

In the Client, we add export **PS1=”U-10.9.0.5:\w\n\$>”U-10.9.0.5:/** to show the host U clearly like **U-10.9.0.5. In** Host U, we **ping 10.9.0.11 -c 2** which is a VPN server.The -c 2 flag specifies that you want to send only 2 ICMP echo requests.

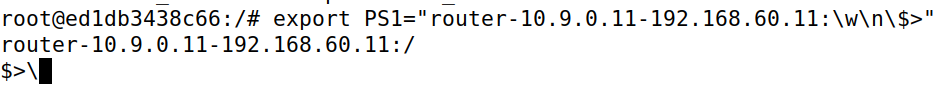
This shows 2 packets transmitted and 2 packets received.



In host V,we add export **PS1=”V-192.168.60.5:\w\n\$>”V-192.168.60.5:/** to show the host V clearly like **V-192.168.60.5.**



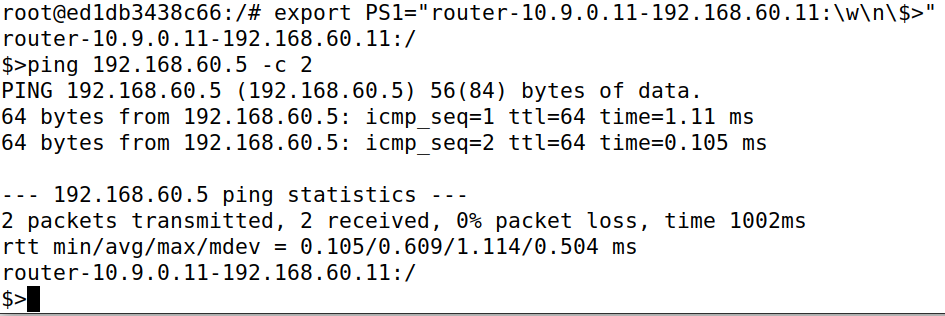
In server-router, we add export **PS1=”router-10.9.0.11-192.168.60.11:\w\n\$>router-10.9.0.11-192.168.60.11”:/** to show the router**.**



**• VPN Server can communicate with Host V.**

In the router, we **ping 192.168.60.5 -c 2** which is host V.The -c 2 flag specifies that you want to send only 2 ICMP echo requests.

This shows 2 packets transmitted and 2 packets received.

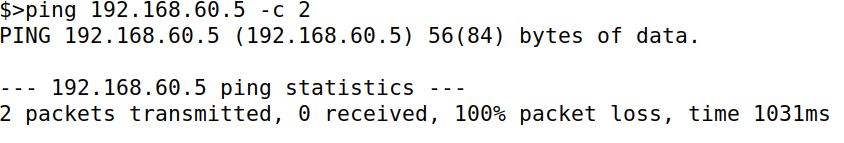


**• Host U should not be able to communicate with Host V.**

In host U , we **ping 192.168.60.5 -c 2** which is host V.The -c 2 flag specifies that you want to send only 2 ICMP echo requests.

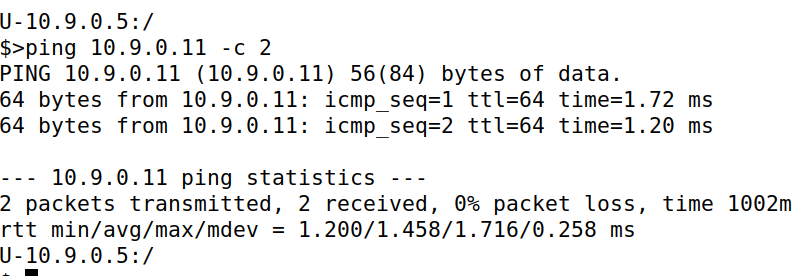
This shows 2 packets transmitted, 0 packets received and 100% packet loss.

This shows host U not able to communicate with Host V.

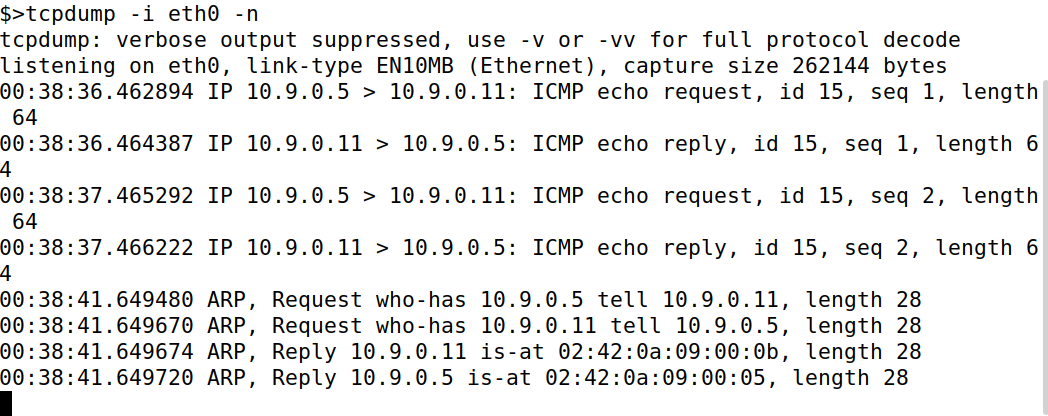


**• Run tcpdump on the router, and sniff the traffic on each of the network. Show that you can capture packets**.

In Host U, we ping router with -c 2 flag

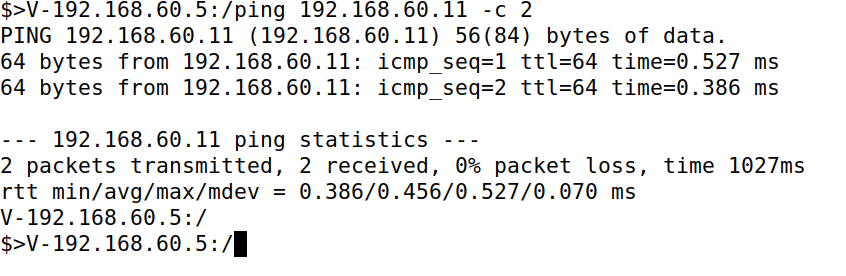


In the router, we run **tcpdump -i eth0 -n,** simultaneously if we ping the router from **host U,** we receive 2 ICMP echo requests from **10.9.0.5 to 10.9.0.11.** Similarly, we receive 2 ICMP echo replies from **10.9.0.11 to 10.9.0.5.**

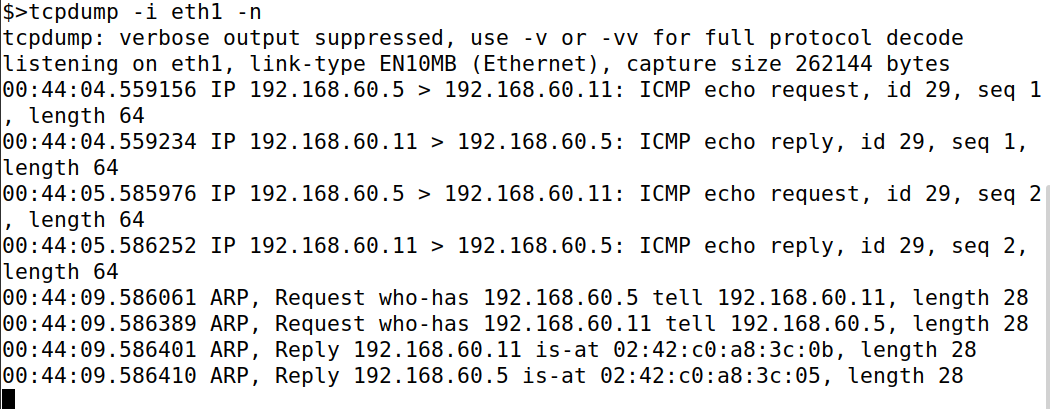


Similarly like above ones,

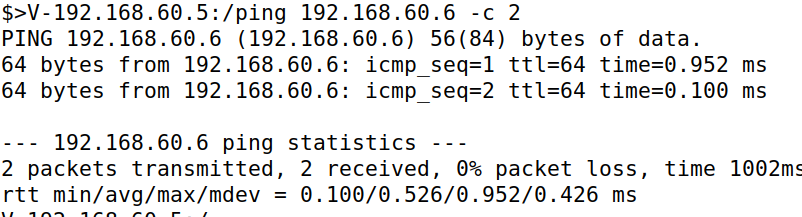
In Host V, we ping the router with -c 2 flag.



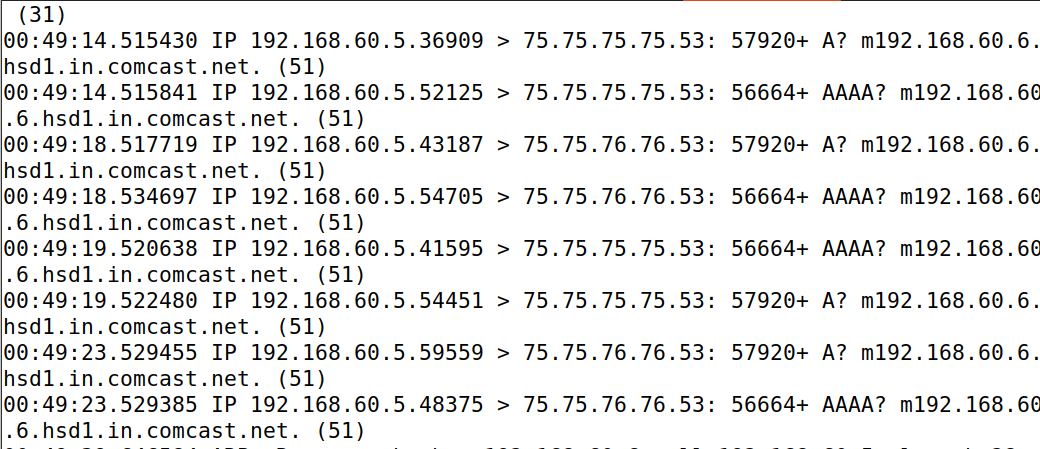
In the router, we run **tcpdump -i eth1 -n,** simultaneously if we ping the router from **host V,** we receive 2 ICMP echo requests from **192.168.60.5 (hostV) to 192.168.60.11(router).** Similarly, we receive 2 ICMP echo replies from **192.168.60.11 to 192.168.60.5.**



In host V, we ping other host U(192.168.60.6), but we cannot find these on the router, because these packets do not go to the router.



This shows output received in the server-client.



This shows that running tcpdump on the router, and sniffing the traffic on each of the network.

**Task 2 (Create and Configure TUN Interface) – 30 pts**

**Task 2.a (Name of the Interface) – 5 pts**

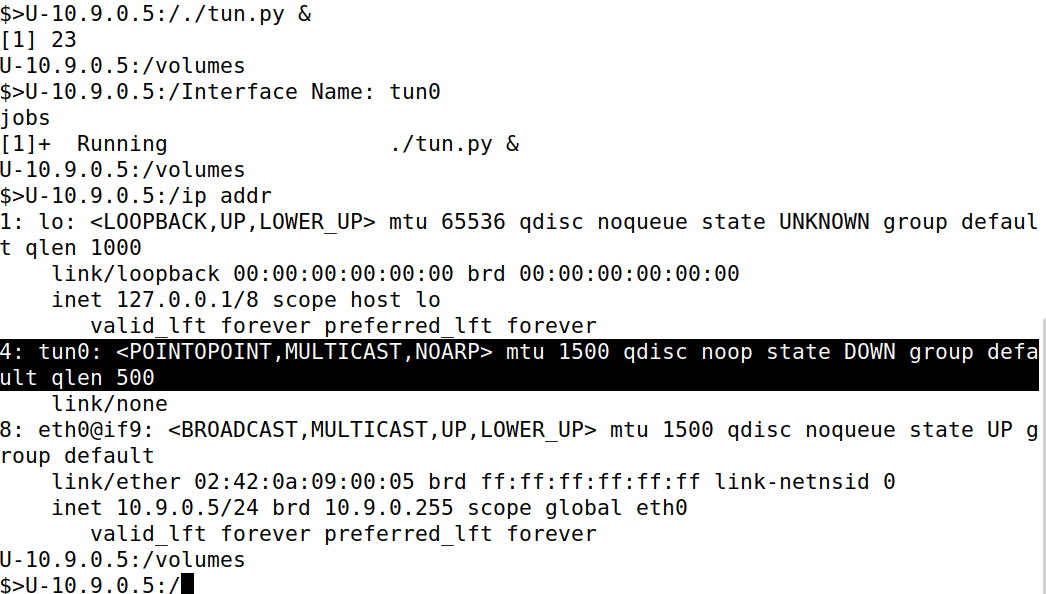
**Code for 2.a and 2.b:**

**tun.py:**

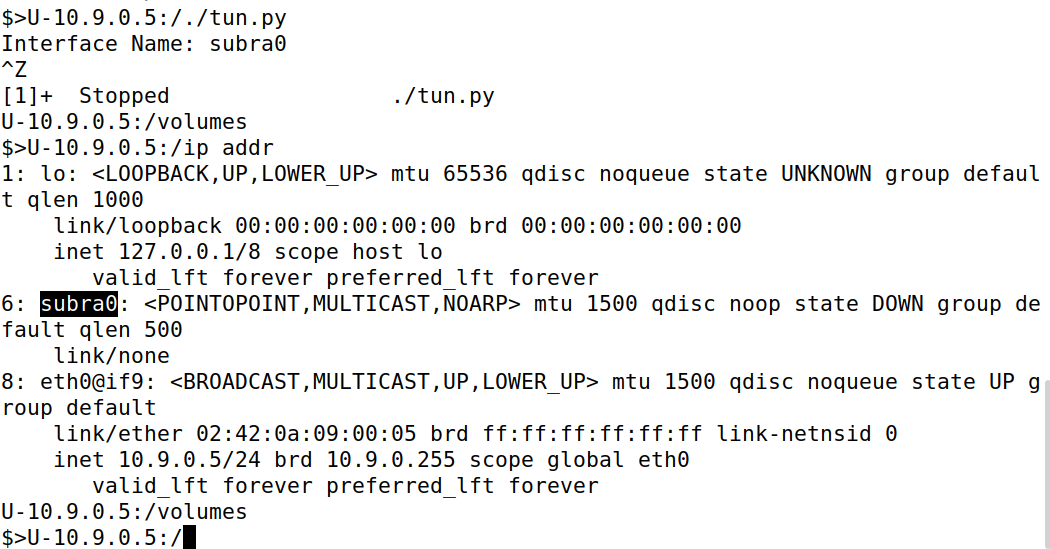
****

In Host U, the command `**chmod a+x tun.py**` makes the file `tun.py` executable by all users. Then we run **tun.py,** highted interface **tun0** output after running **ip addr.**





My last name is Subramanian so my interface name is **subra0.** So I am modifying it in the code**.** You can see my highlighted interface name below once I run **tun.py.**



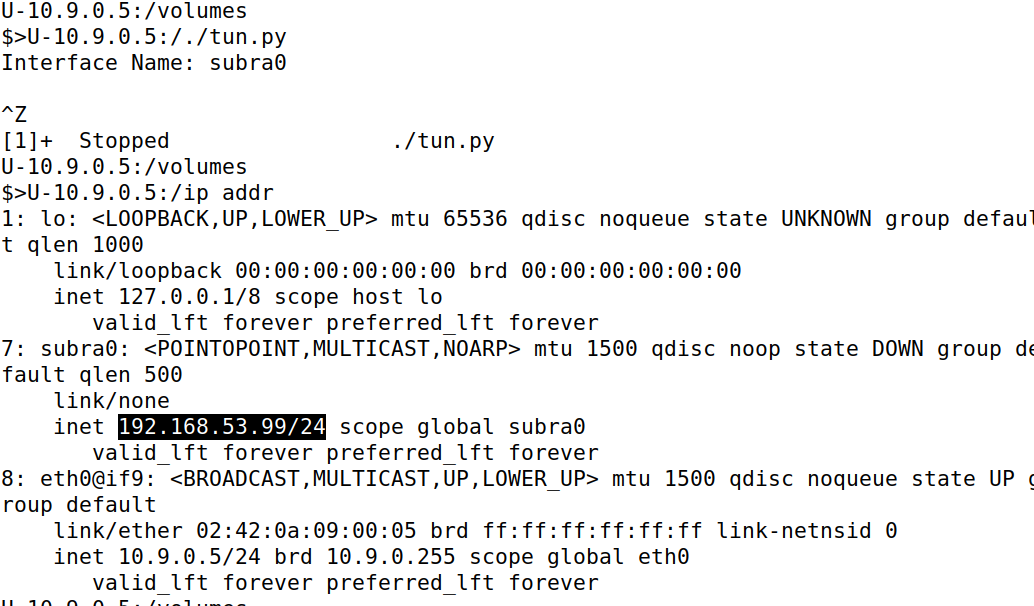
**Task 2.b (Set up the TUN Interface) – 5 pts**

We add these lines

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

**os.system("ip link set dev {} up".format(ifname))** in our code**.**

Now we run **tun.py,** then you can see my changed interface name. Then I ran **ip addr,** and we got ip address in inet **192.168.53.99/24.** When we compare this with our old one, we gotinet **192.168.53.99/24 scope global subra0** in our new one which is not there in the old one**.**



**Task 2.c (Read from the TUN Interface) – 10 pts**

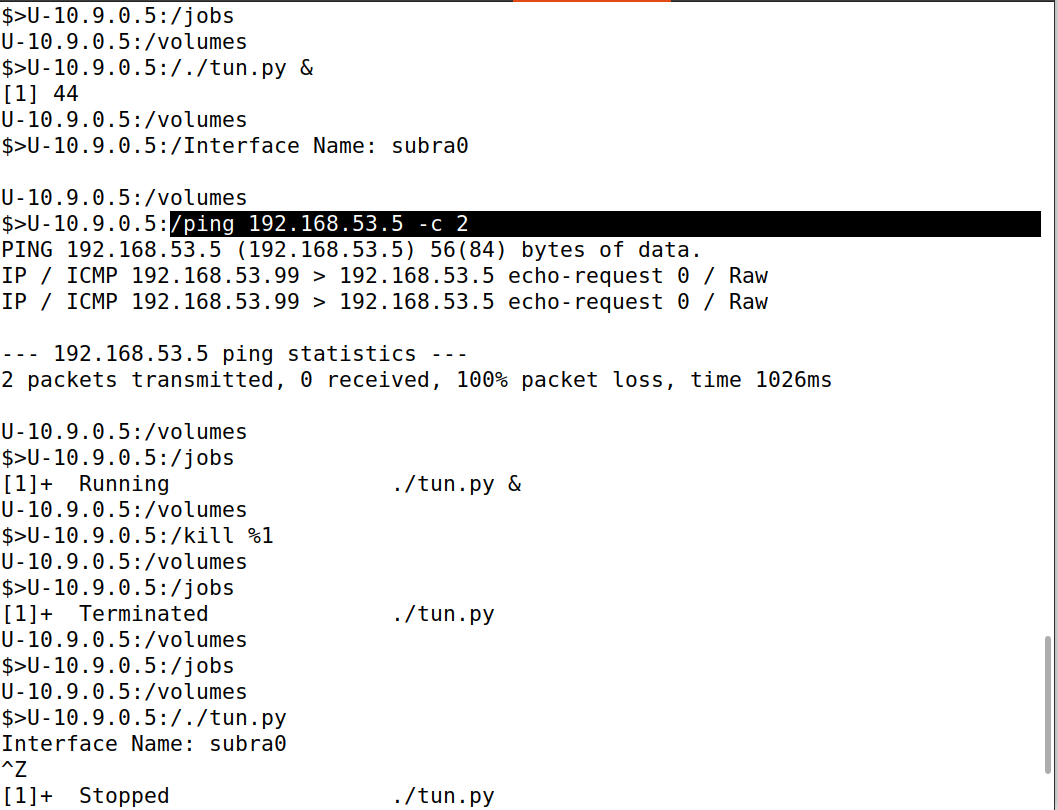
**Code:**

**(updated code with the requirements)**



• **On Host U, ping a host in the 192.168.53.0/24 network. What is printed out by the tun.py program? What has happened? Why?**

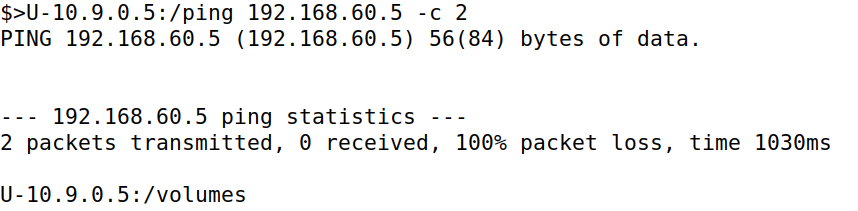
Now we run our updated code, **tun.py** and **ping 192.168.53.5 -c 2** in the host U. We find 2 echo-requests are printed, which shows 2 packets are transmitted, but we are not able to receive the reply.



**• On Host U, ping a host in the internal network 192.168.60.0/24, Does tun.py print out anything? Why?**

Now in host U, we ping host V, packets are transmitted but not received and there is a

packet loss.



**Task 2.d (Write to the TUN Interface) – 10 pts**

**CODE:**

**#!/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'subra%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))**

**#Configure the interface**

**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:**

**pkt = IP(packet)**

**print("{}:".format(ifname), pkt.summary())**

**# Send out a spoof packet using the tun interface**

**# sniff and print out icmp echo request packet**

**if ICMP in pkt and pkt[ICMP].type == 8:**

**print("Original Packet.........")**

**print("Source IP : ", pkt[IP].src)**

**print("Destination IP :", pkt[IP].dst)**

**# spoof an icmp echo reply packet**

**# swap srcip and dstip**

**ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)**

**icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)**

**data = pkt[Raw].load**

**newpkt = ip/icmp/data**

**print("Spoofed Packet.........")**

**print("Source IP : ", newpkt[IP].src)**

**print("Destination IP :", newpkt[IP].dst)**

**arbdata= b'Instead of writing an IP packet to the interface, write some arbitrary data to the interface, and report your observation.'**

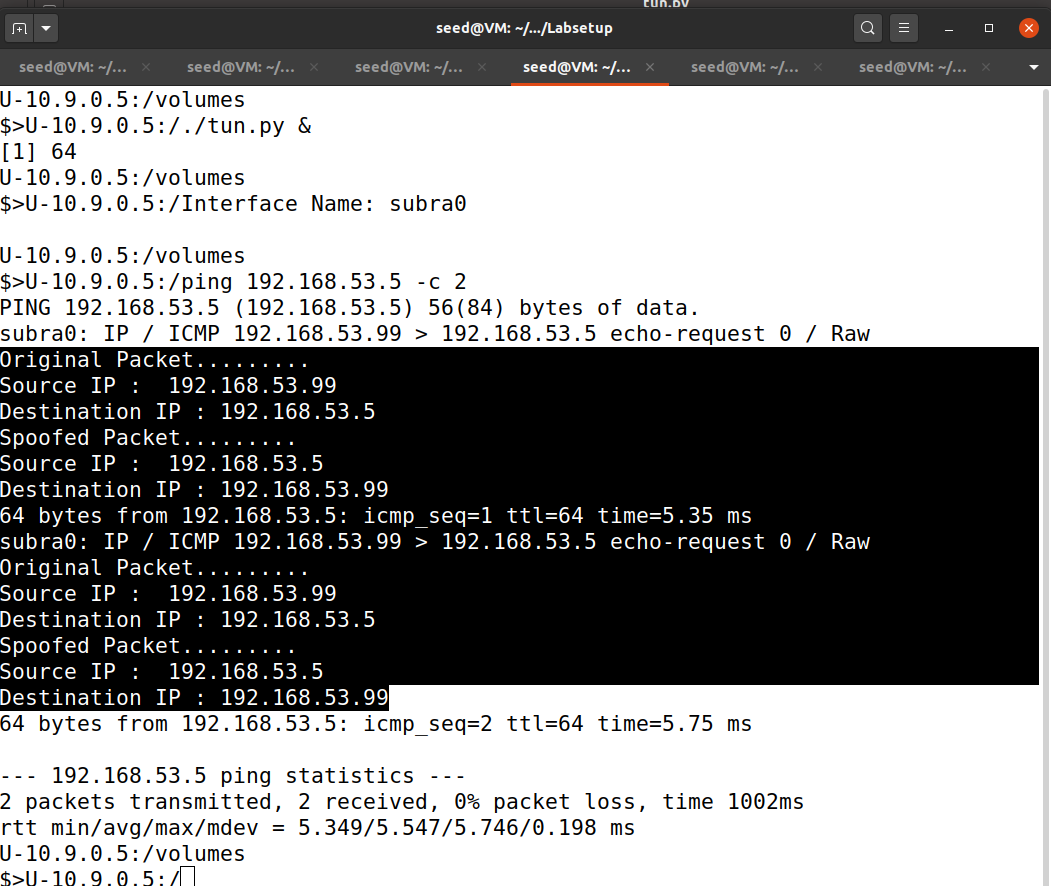
**os.write(tun, arbdata)**

**• After getting a packet from the TUN interface, if this packet is an ICMP echo request packet, construct a corresponding echo reply packet and write it to the TUN interface. Please provide evidence to show that the code works as expected.**

Now, we are running **tun.py** and pinging **192.168.53.5 -c 2,** we can see 2 original packets, 2 spoofed packets with source and destination IP.

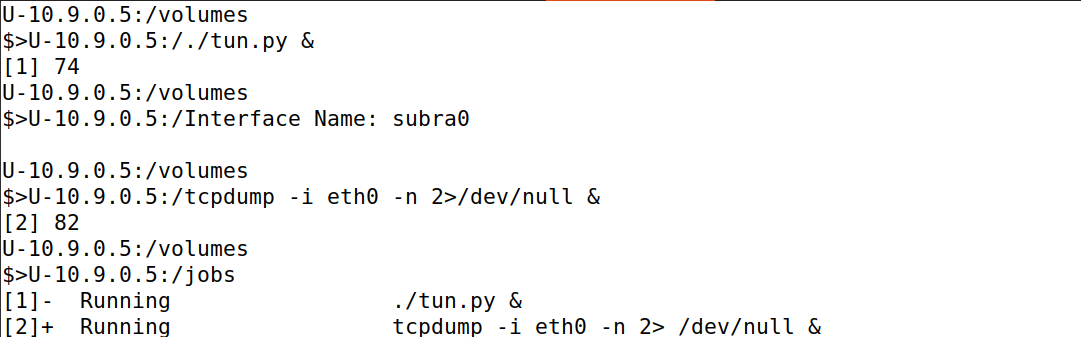
Spoofed packet is reply packet.

2 packets transmitted and 2 packets received.

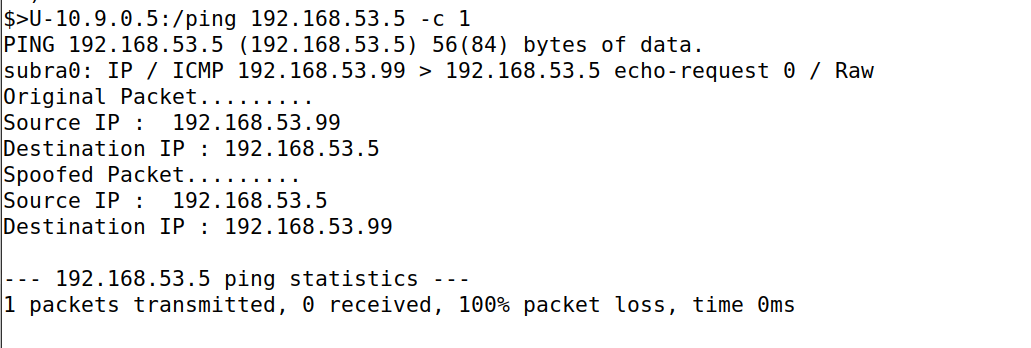
****

**• Instead of writing an IP packet to the interface, write some arbitrary data to the interface, and report your observation.**

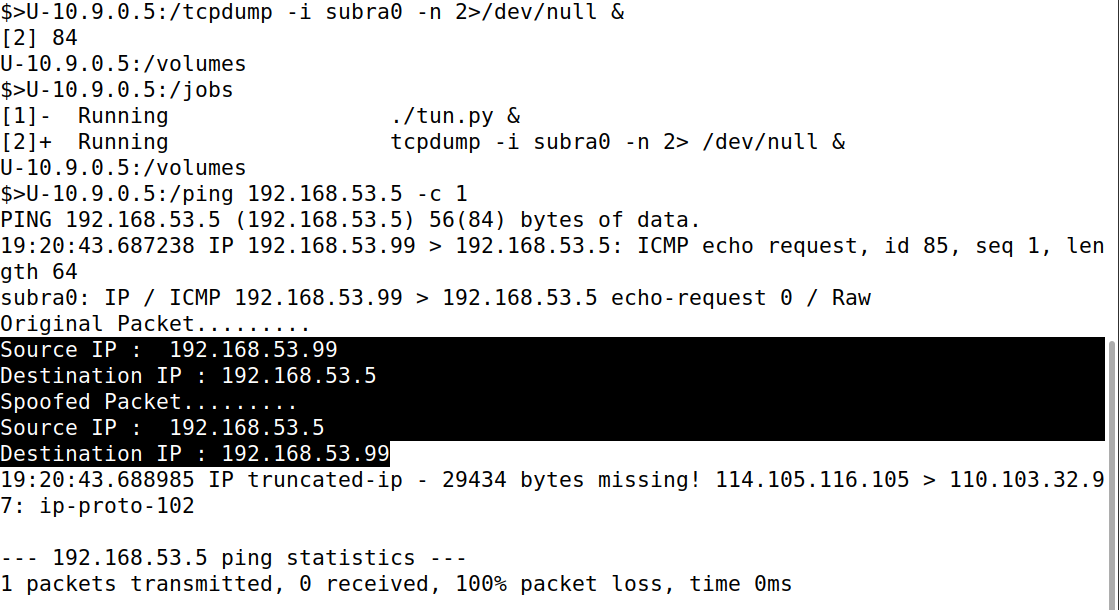
Now we run the code tun.py then we run **tcpdump -i eth0 -n 2>/dev/null &**



Now I ping **192.168.53.5 -c 1**, so that I can find 1 original packet and 1 spoofed packet but 1 packet is transmitted , 0 received.

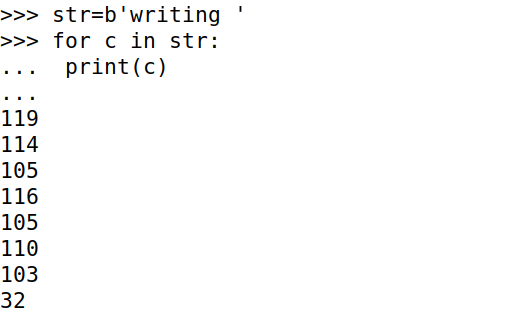


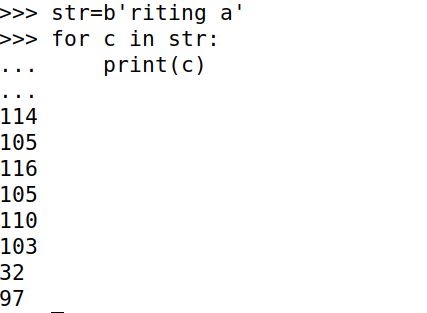
Now I run **tcpdump -i subra0 -n 2>/dev/null & ,** then we are pinging **192.168.53.5 -c 1** one more time. Now we can find IP truncated IP - 29434 bytes missing.



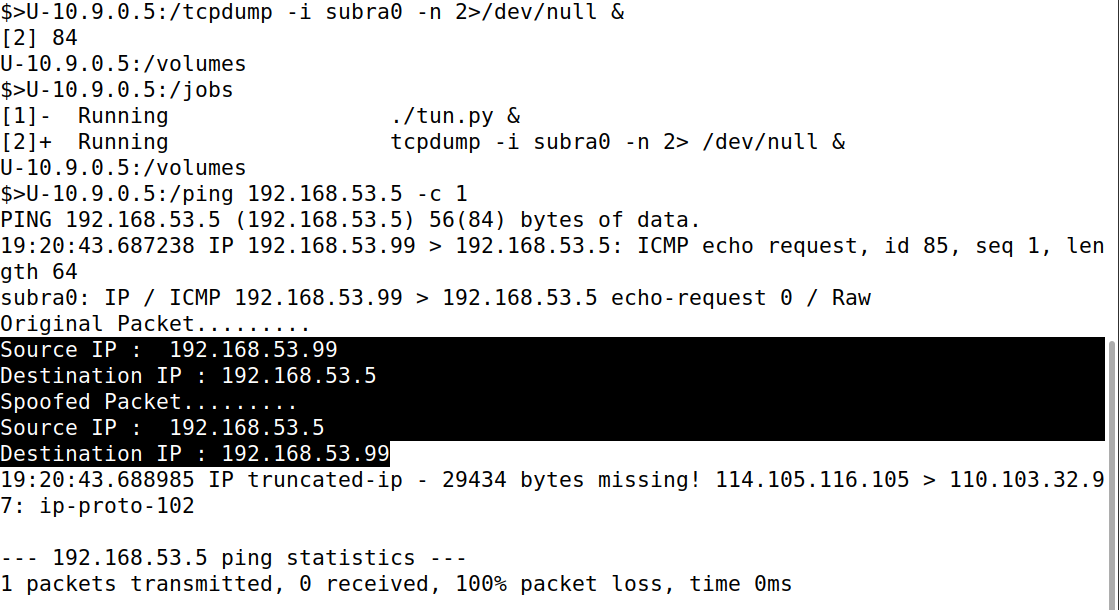
We run python3 in a lab setup, then we enter str=b’writing ‘ for c in str: print(c).

Numbers are displayed source and destination IP address mentioned above in the **IP truncated-ip.** This writing text is available in my code in the line of **arbdata.**





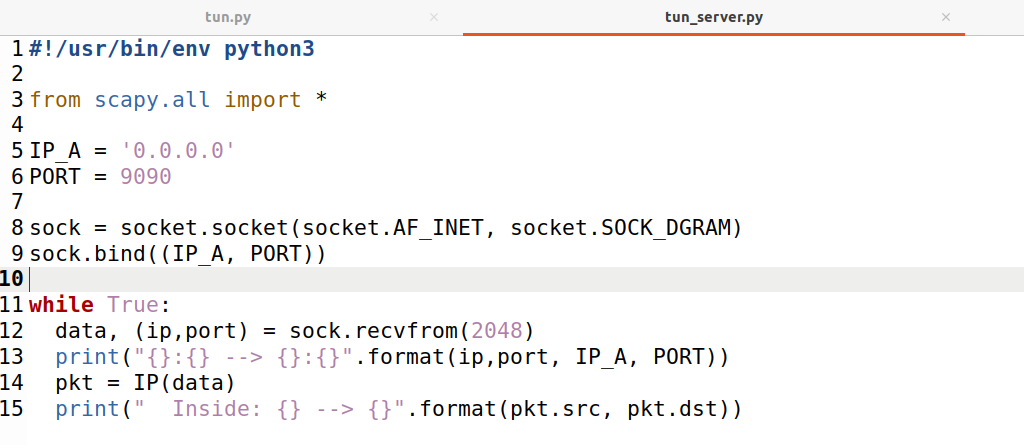
Numbers are from this, in **IP truncated-ip**



**Task 3 (Send the IP Packet to VPN Server Through a Tunnel) – 10 pts**

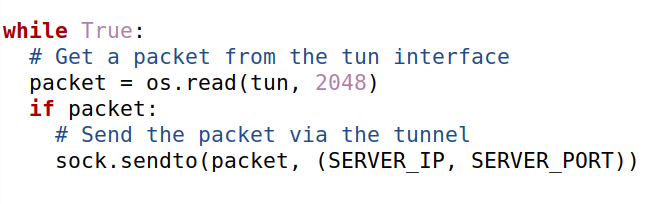
**Code:**

**tun\_server.py:**



**tun\_client.py:**

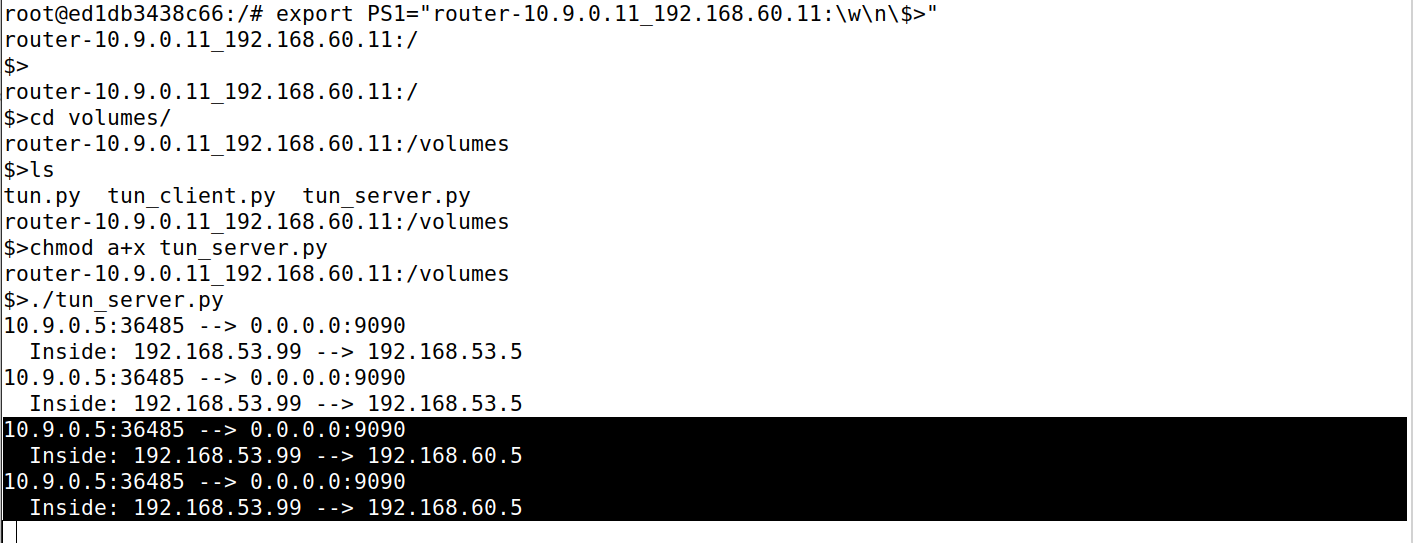




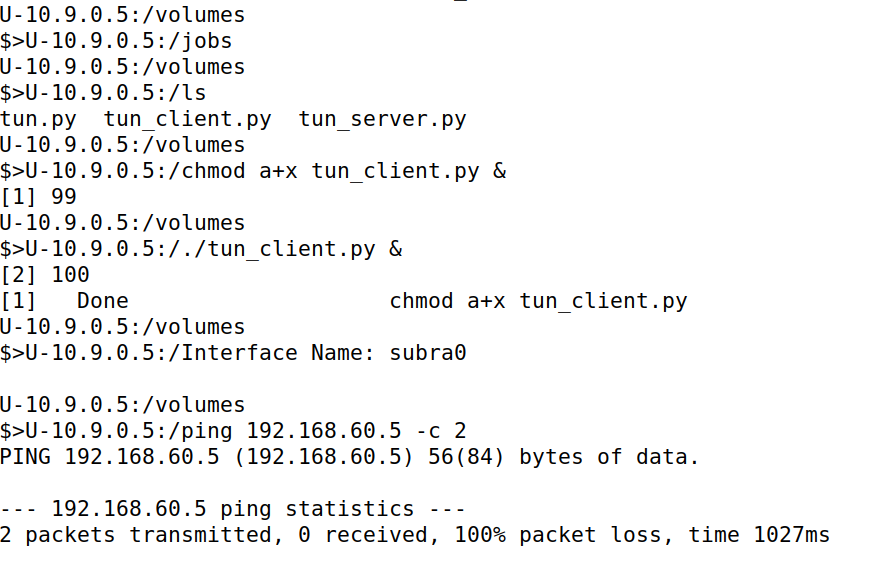
Now we are running tun\_server.py on the router.

We can find **10.9.0.5:36485(random port number)** to **0.0.0.0:9090,**which is a UDP packet from **client** to **VPN server.** And inside we can find **192.168.53.99 to 192.168.53.5** which is a UDP packet **.**

We received it one more time. VPN tunnel works as an IP packet is a port inside UDP payload and received by the server.



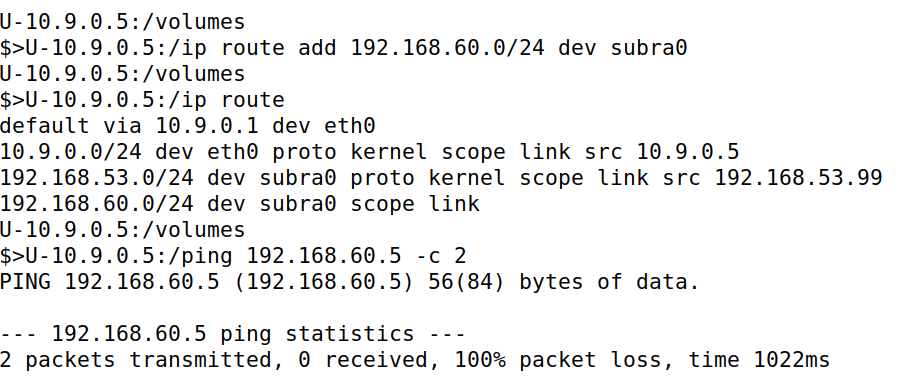
In Host U we run **tun\_client.py &** in background and while pinging **192.168.60.5 -c 2**, 2 packets are transmitted and 0 received with 100% packet loss.



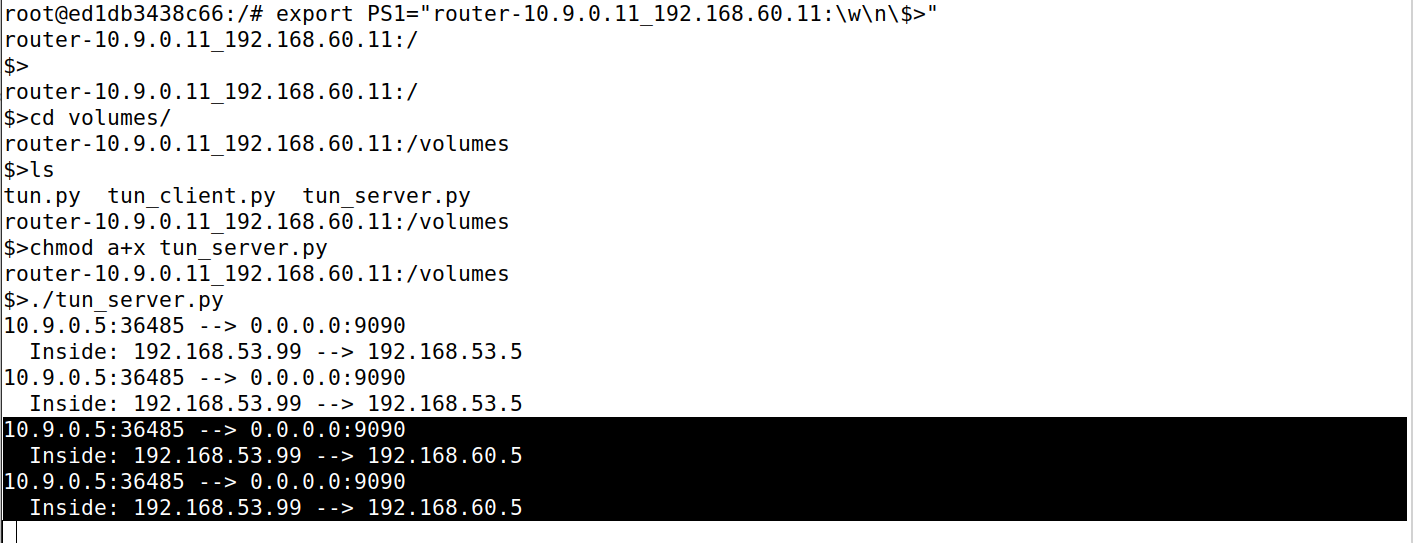
Now in host U, I run **ip route add 192.168.60.0/24 dev subra0,**add a route to the 192.168.60.0/24 network via the subra0 interface.

Then I use ip route, to show the routes.

Then while pinging **192.168.60.5 -c 2**, 2 packets are transmitted and 0 received with 100% packet loss.

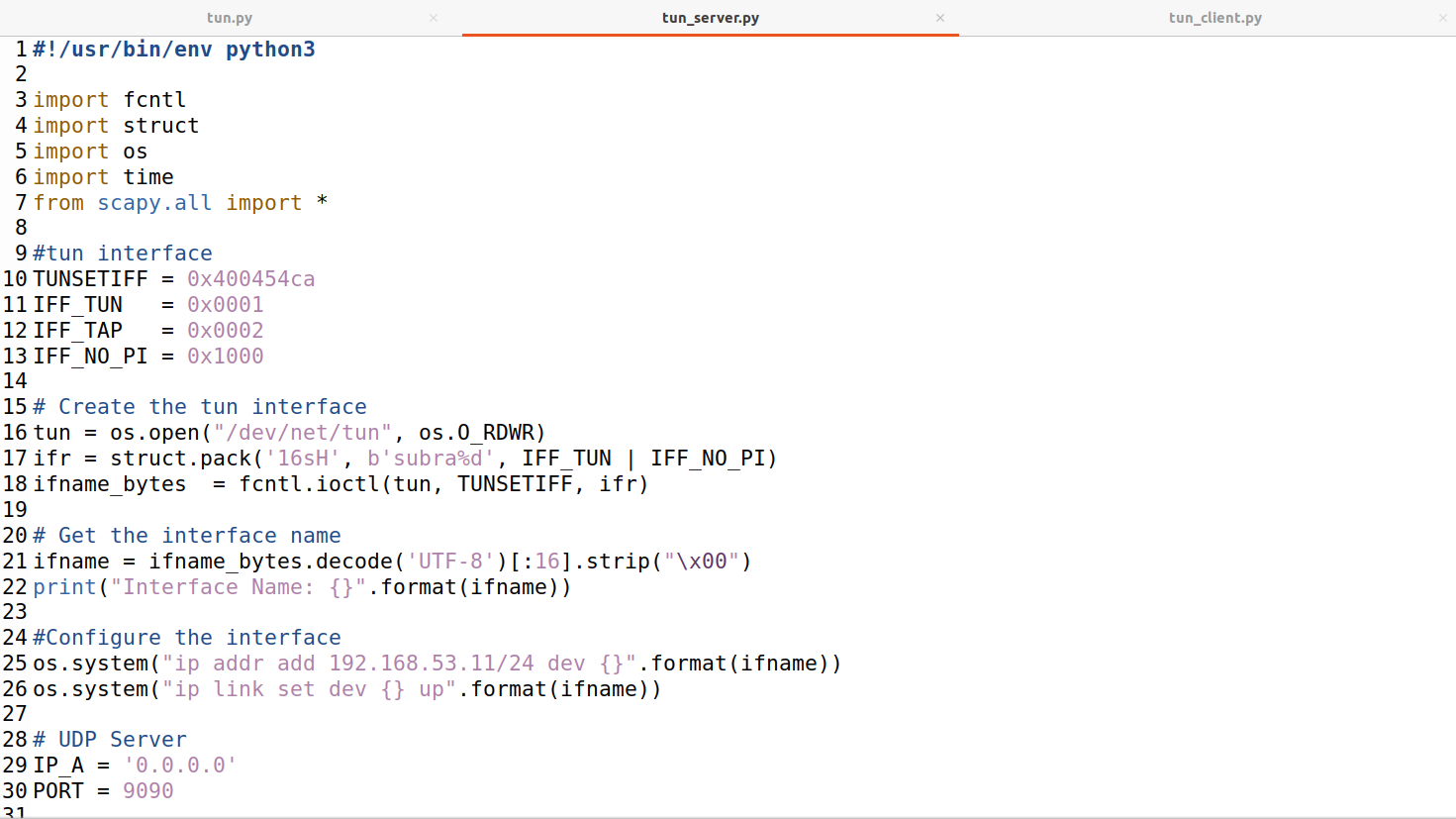


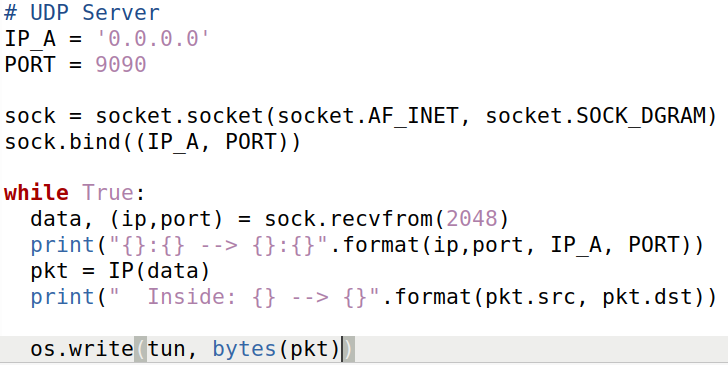
We can find newly added in highlighted one, **10.9.0.5:36485(random port number)** to **0.0.0.0:9090,**which is a UDP packet from **client** to **VPN server.** And inside we can find **192.168.53.99 to 192.168.60.5(host V)**which is a UDP packet **.**



**Task 4 (Set Up the VPN Server) – 10 pts**

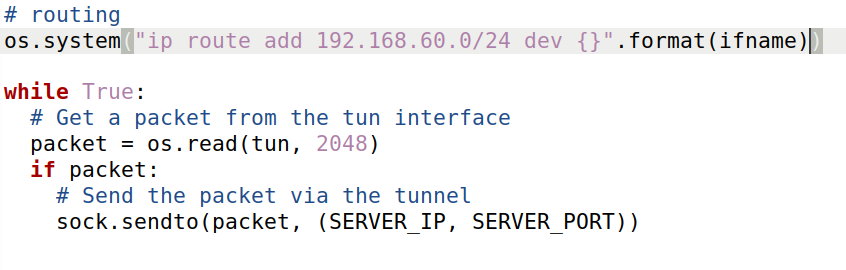
**Code: Modified tun\_server.py:**

****

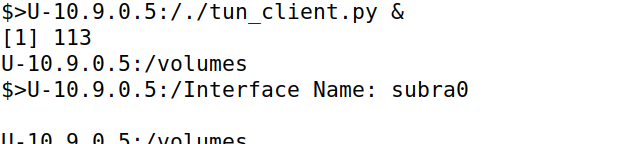
****

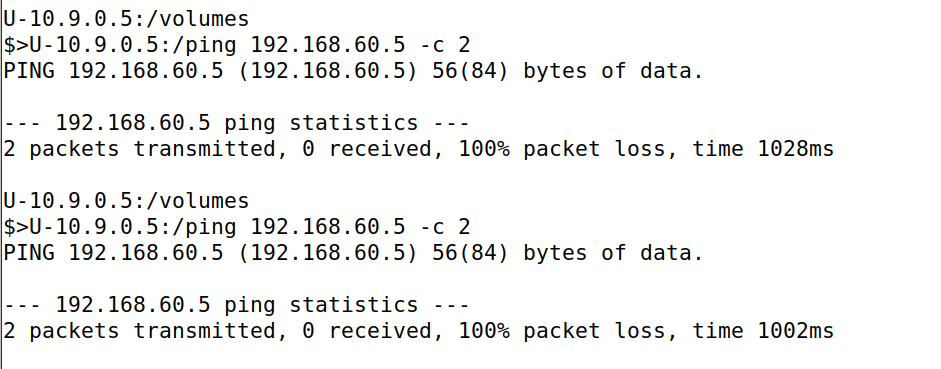
**Modified tun\_client.py:**

****

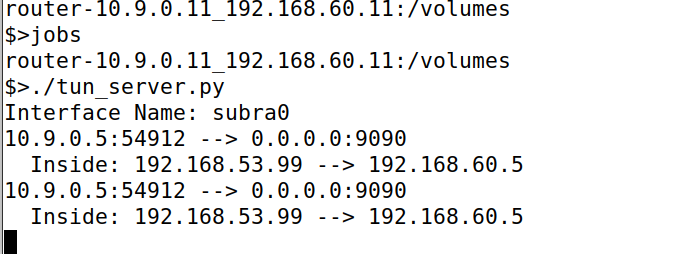
****

In Host U, we run **tun\_client.py &** in background and ping 192.168.60.5 -c 2

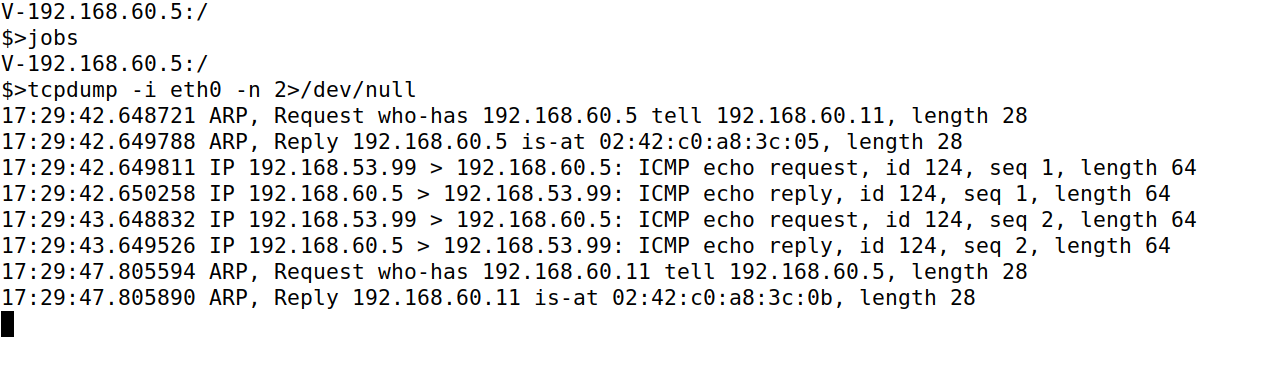
****

****

In the router, we are running tun\_server.py, we will be simultaneously ping 192.168.60.5 -c 2 in host U, we find 2 received ping requests,from client to server, Inside IP request to hostV.



In hostV, we run tcpdump -i eth0 -n 2>/dev/null, showing it is sniffing. We can find 2 ICMP echo requests and 2 replies. This shows that ICMP packets arrived at host V.



**Task 5 (Handling Traffic in Both Directions) – 20 pts**

**Code:**

**tun\_client.py:**

**#!/usr/bin/env python3**

**import fcntl**

**import struct**

**import os**

**import time**

**from scapy.all import \***

**# Create UDP socket**

**sock = socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM)**

**SERVER\_IP, SERVER\_PORT = "10.9.0.11", 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'subra%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))**

**#Configure the interface**

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

**os.system("ip link set dev {} up".format(ifname))**

**# routing**

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

**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)**

**print("From UDP {}:{} --> {}".format(ip, port, "10.9.0.5"))**

**pkt = IP(data)**

**print("From socket(IP) ==>: {} --> {}".format(pkt.src, pkt.dst))**

**os.write(tun, bytes(pkt))**

**if fd is tun:**

**packet = os.read(tun,2048)**

**pkt = IP(packet)**

**print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))**

**# Send the packet via the tunnel**

**sock.sendto(packet, (SERVER\_IP, SERVER\_PORT))**

**tun\_server.py:**

**#!/usr/bin/env python3**

**import fcntl**

**import struct**

**import os**

**import time**

**from scapy.all import \***

**#tun interface**

**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'subra%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))**

**#Configure the interface**

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

**os.system("ip link set dev {} up".format(ifname))**

**# UDP Server**

**IP\_A = '0.0.0.0'**

**PORT = 9090**

**ip,port = '10.9.0.5', 12345**

**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:**

**data, (ip,port) = sock.recvfrom(2048)**

**print("From UDP {}:{} --> {}:{}".format(ip,port,IP\_A,PORT))**

**pkt = IP(data)**

**print("From socket(IP)==>: {} --> {}".format(pkt.src, pkt.dst))**

**os.write(tun, bytes(pkt))**

**if fd is tun:**

**packet = os.read(tun,2048)**

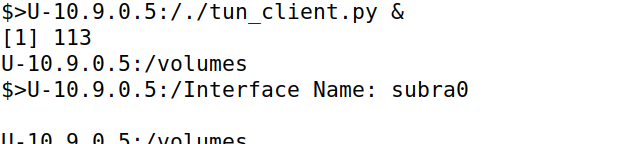
**pkt = IP(packet)**

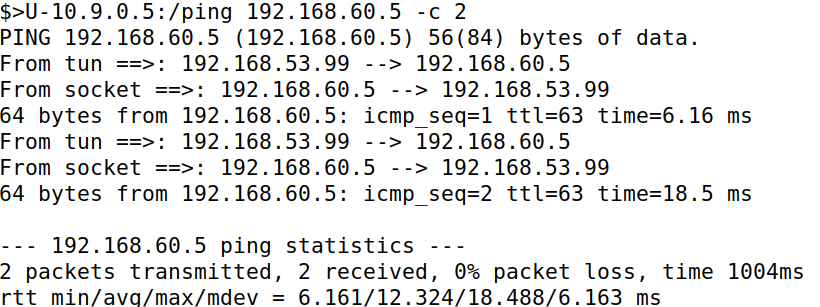
**print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))**

**sock.sendto(packet, (ip,port))**

First we run tun\_server.py in the seed\_router.

In Host U, run tun\_client.py & in the background and we are pinging (host V) **192.168.60.5 -c 2,** we received 2 packets transmitted and 2 received.and in server we can see output after pinging, which shows the tunnel works as expected.

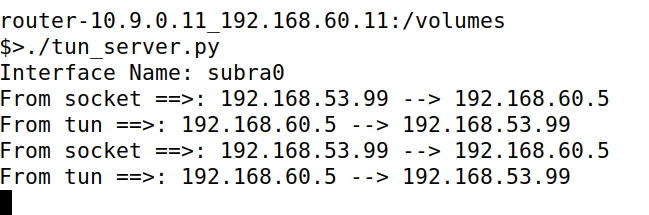
****



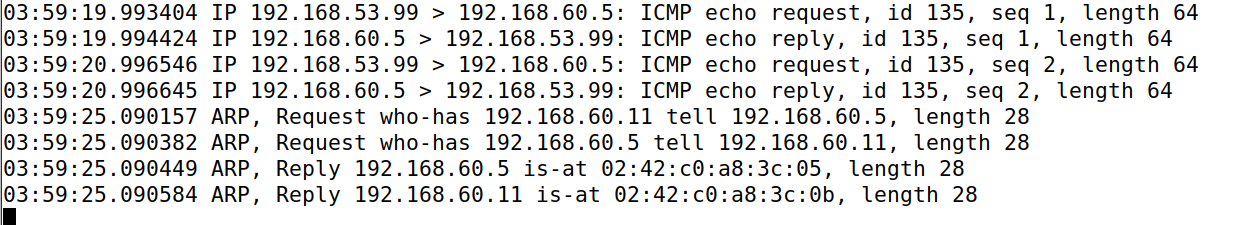
We will receive this output, after pinging

Output from UDP socket to VPN client(host V). Similarly from client to socket.

Below from socket, from host U(which is 192.168.53.99 because of the tunnel) to host V in the tunnel network. And in the tunnel from host V to host U.



**In Host-V,** we can see these 2 ICMP echo requests and ICMP echo reply from host U to host V and host V to host U respectively.This shows pinging is successful.

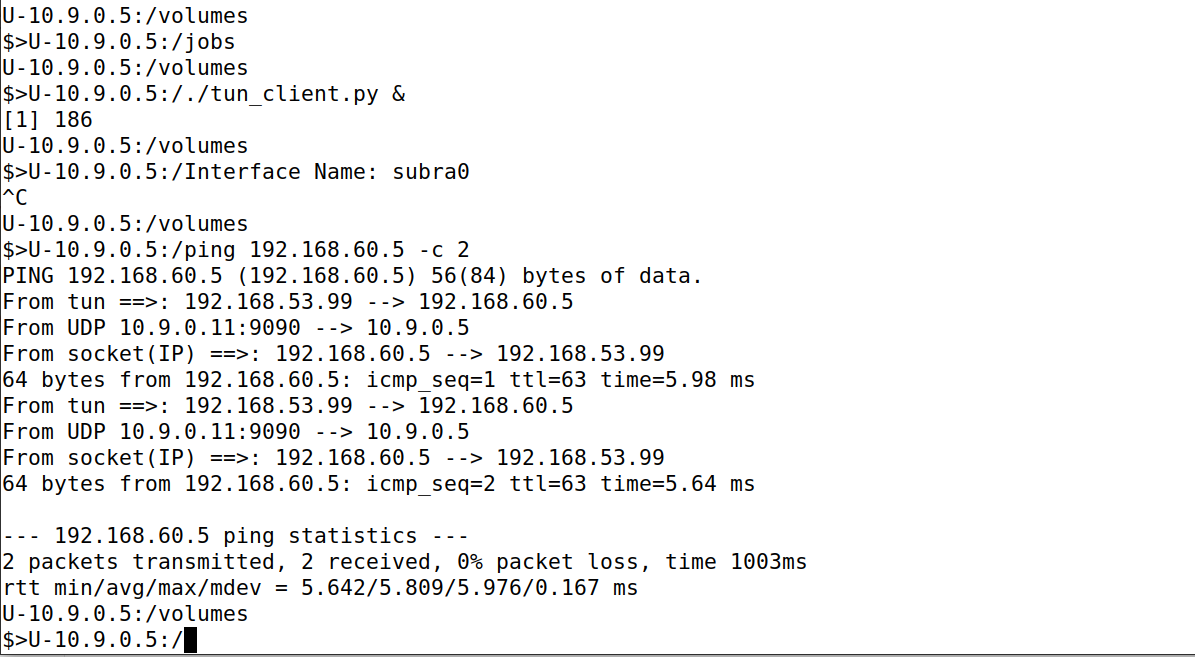


Now we updated code to show Ip inside UDP payload

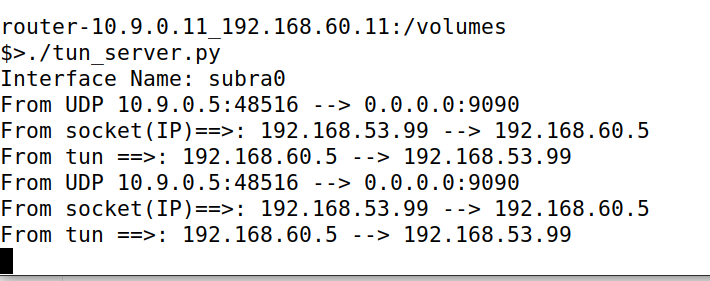
In host U, run tun\_client.py & in the background and we are pinging (host V) **192.168.60.5 -c 2,** we received 2 packets transmitted and 2 received.and in server we can see output after pinging.

In below socket(IP) is encapsulated IP inside UDP tunnel.

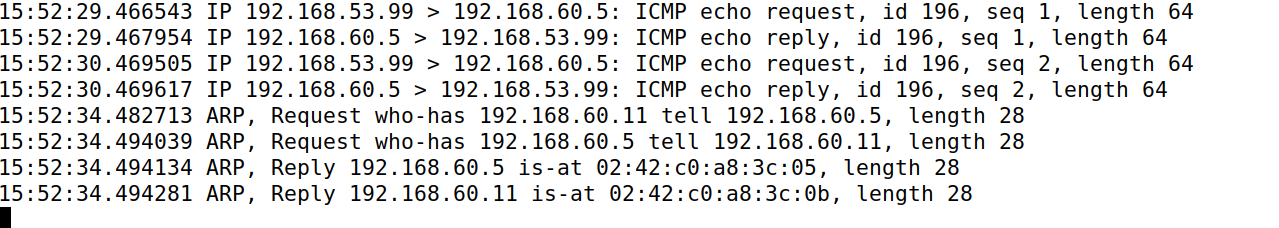
**From tun** which is host U to host V**, from UDP (10.9.0.11:9090)** which is router to client **and from socket(IP)** which is actually reply from hostV to host U.



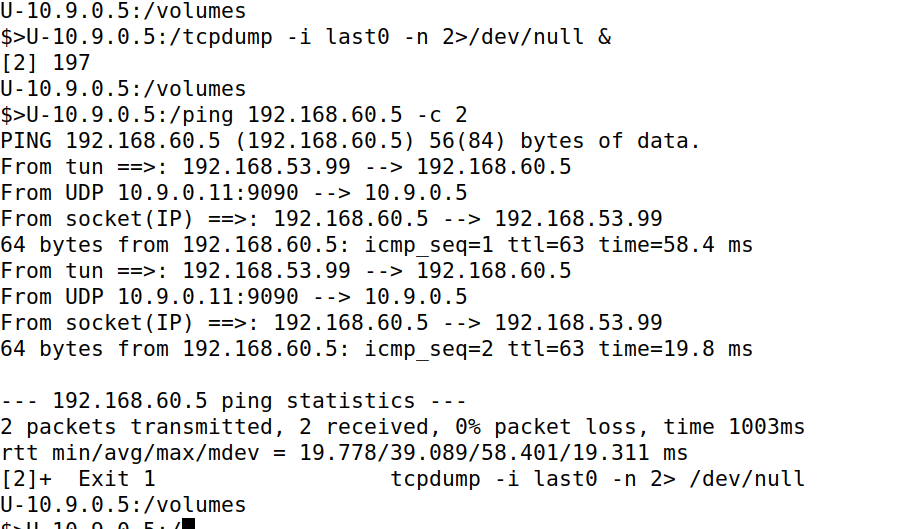
In router, we can see the same things happened above here



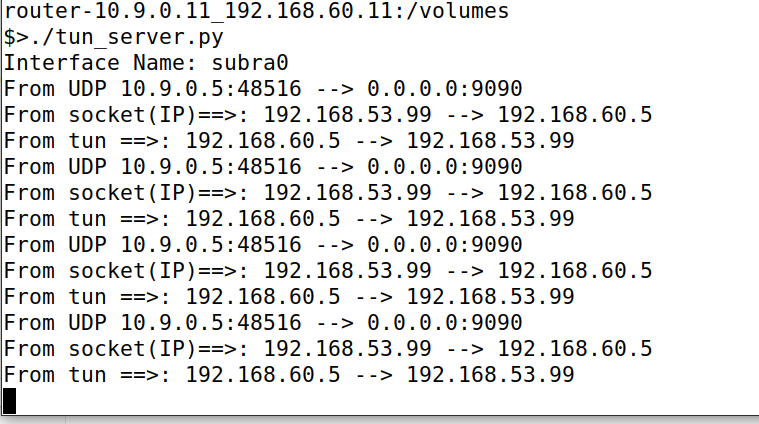
In host V, we can see the same things here also .



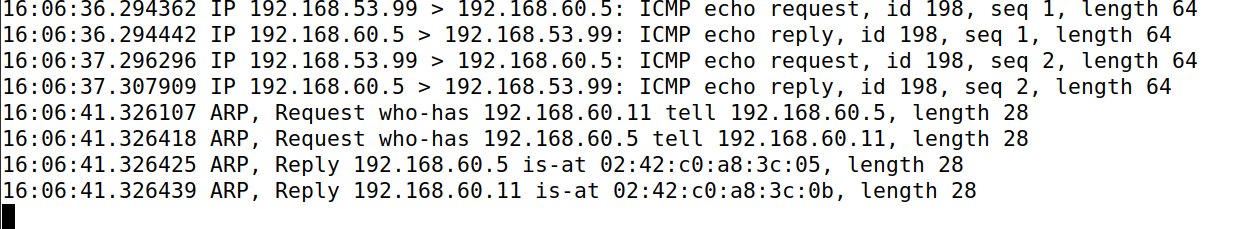
On host U, we run **tcpdump -i last0 -n 2>/dev/null** which sniff on last0 and subtract the error output and run in the background.Then we ping host V, we can see packets captured by tcpdump.



Similarly we can see the above things on the server.



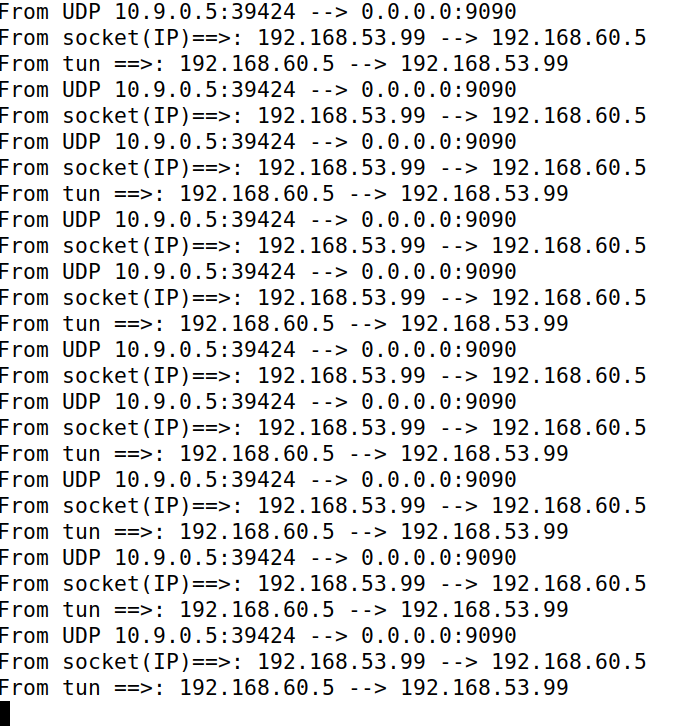
Similarly we see above things in host V.



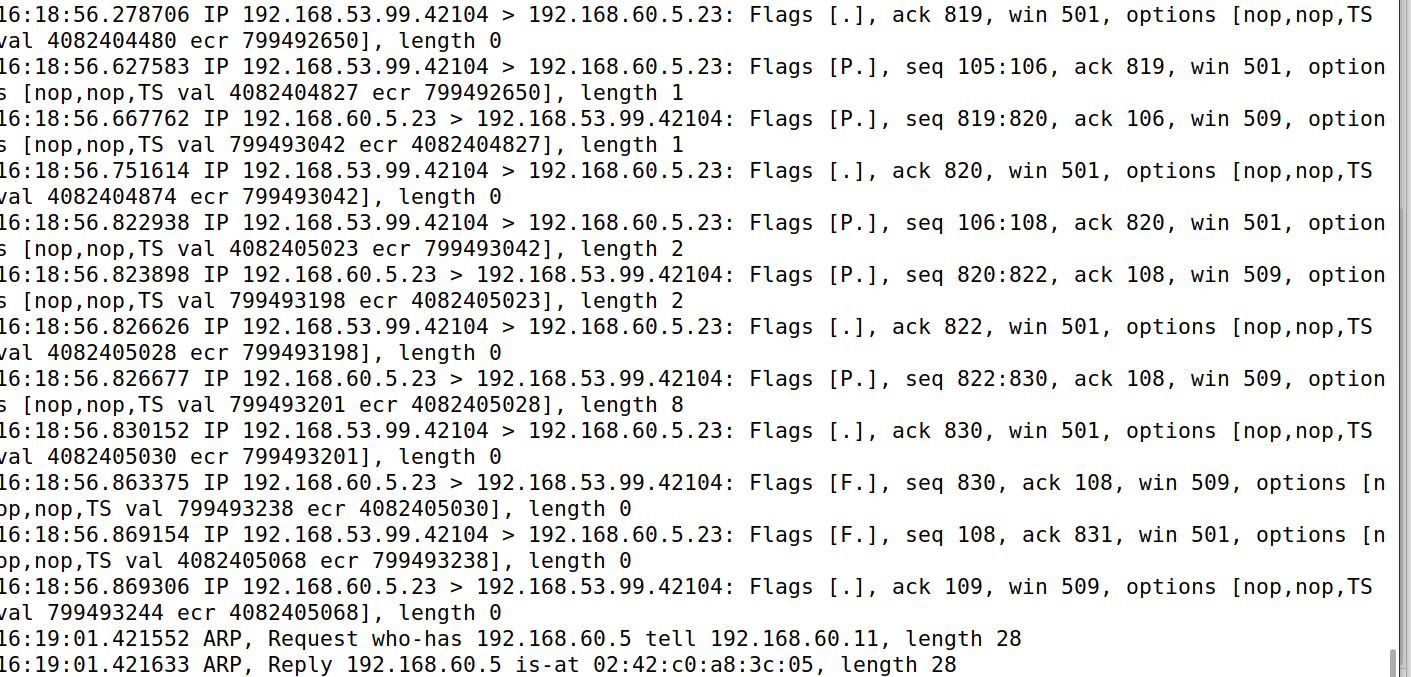
Now we do telnet to host V in host U, we enter login and password.we logged in host V through vpn tunnel.



While logging in to host V, we can see a list of packet flow in th**e server**.

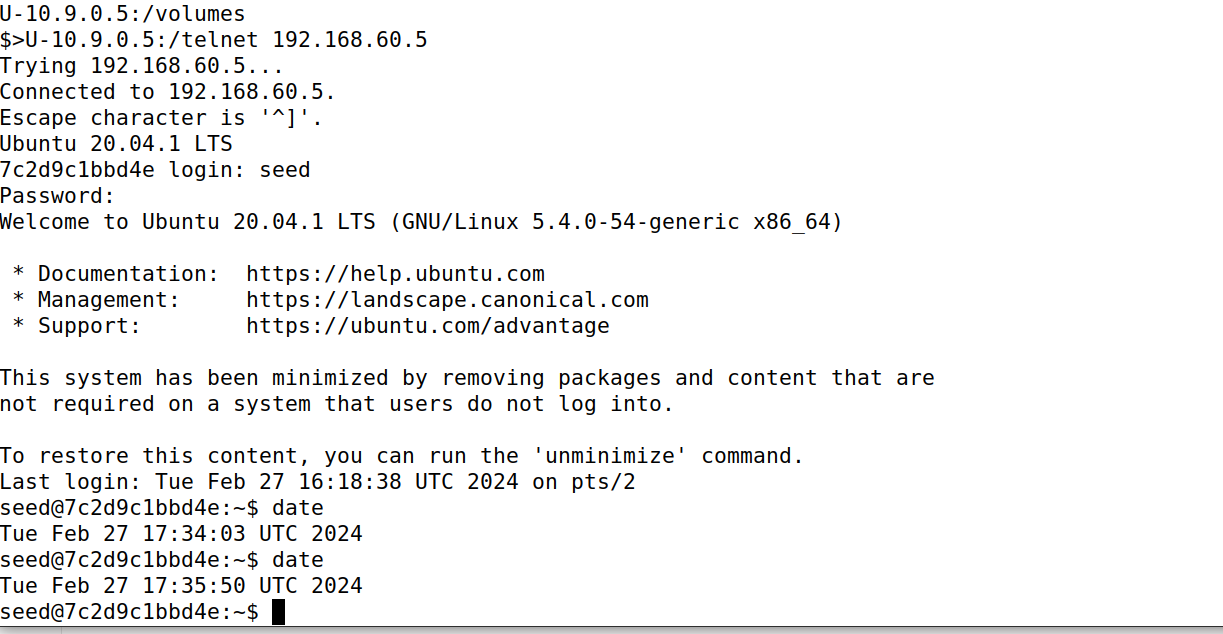


**We can see the following output after logging in to host V. This shows tcp dump captured a lot of packets. And telnet succeeded.**

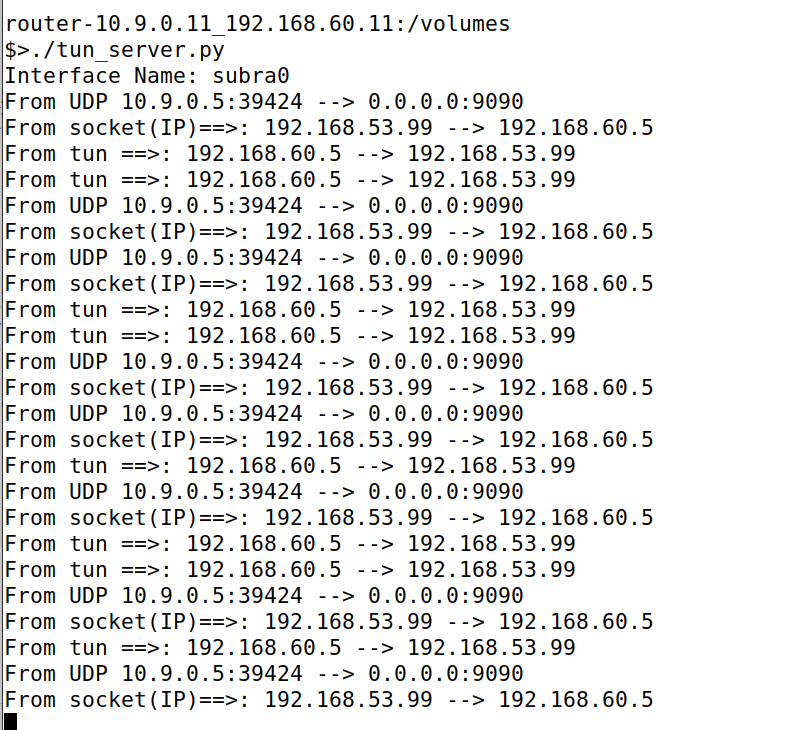


**Task 6 (Tunnel-Breaking Experiment) – 10 pts ( same code as above one)**

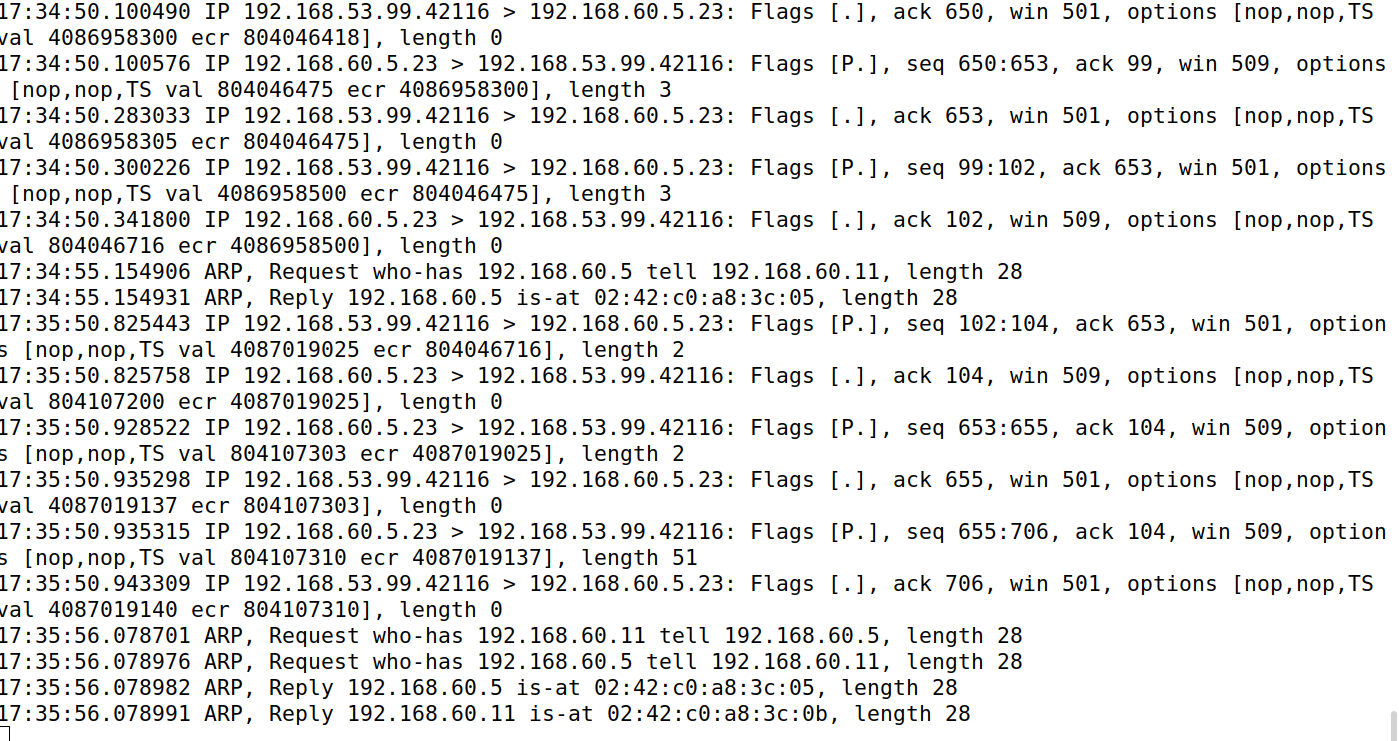
In host U, we telnet to host V,and it is successful we go inside seed and type date, date along with time is displayed. If we stop running code in the router, then we cannot type anything inside the seed. Now after reconnecting in the router, it works.



While running tun-server.py in the router, we can see the following TCP requests.



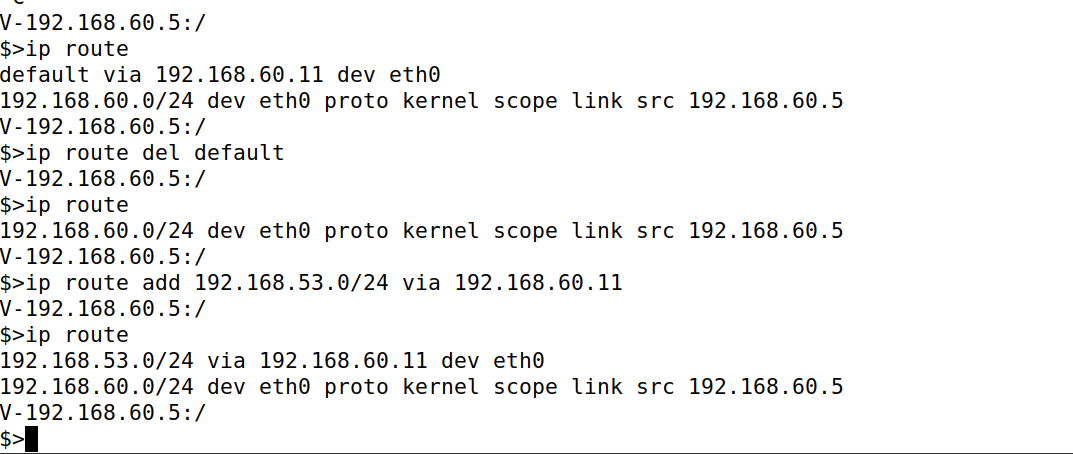
This is the output we got in hostV while running server and telnet.



TCP doesn’t receive acknowledgement, and keeps resending packets. TCP connection is reliable, so connection is not broken.

**Task 7 (Routing Experiment on Host V) – 10 pts**

We are doing this experiment in host V, we are running an ip route which shows entries. And we type ip route del default to to delete that entry and we use ip route add 192.168.53.0/24 via 192.168.60.11 to add the entry.

****