**VPN Tunneling**

Chawin Metah

Department of Computer Science, Purdue University Fort Wayne

ACS 54500: Cryptography and Network Security

Lab 6

Dr. Zesheng Chen

March 2, 2022

**Task 1: Network Setup**

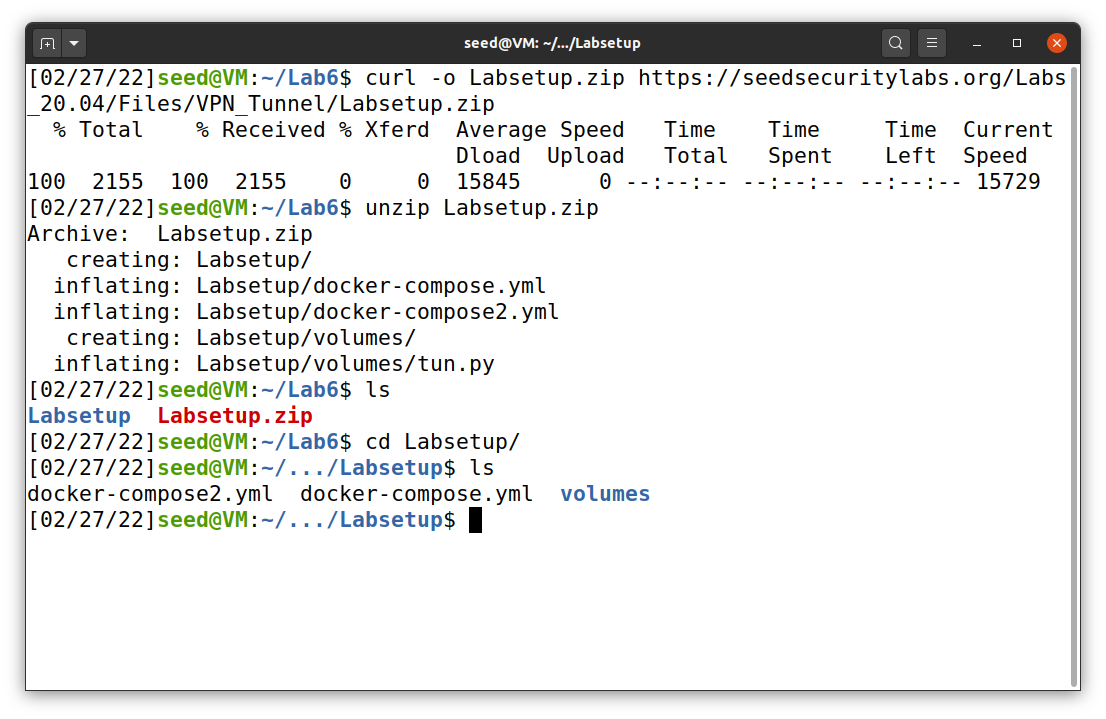
Similar to the previous labs, we download and unzip Labsetup.zip, which contains docker-compose.yml and other directories (Figure 1). It can be downloaded with the following command: curl -o Labsetup.zip https://seedsecuritylabs.org/Labs\_20.04/‌Files/VPN\_Tunnel/Labsetup.zip. I used the unzip command to extract the file. Then, I used the command below to initiate network, as shown in Figure 2. This lab, we have VPN client (10.9.0.5), VPN Server (Router), and other 3 users connected to router (the diagram is shown in Firewall.pdf, available at https://seedsecuritylabs.org/Labs\_20.04/Files/VPN\_Tunnel/‌VPN\_Tunnel.pdf).

dcbuild

dcup

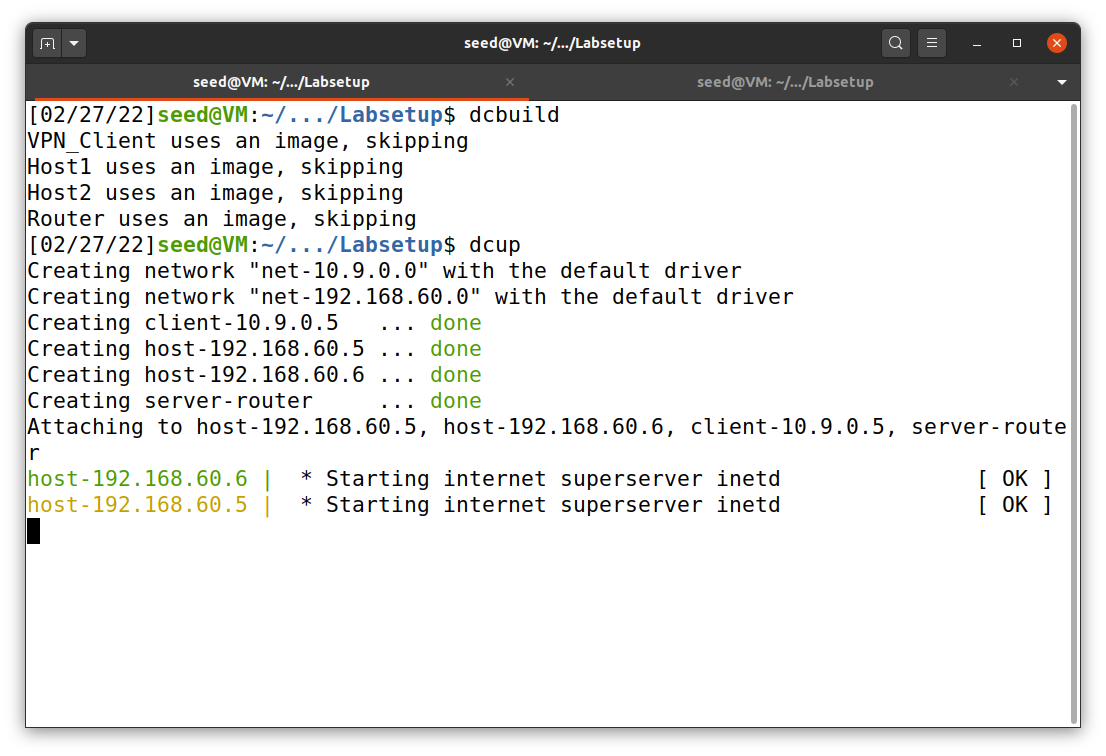
**Figure 1**

*Labsetup.zip*

**

**Figure 2**

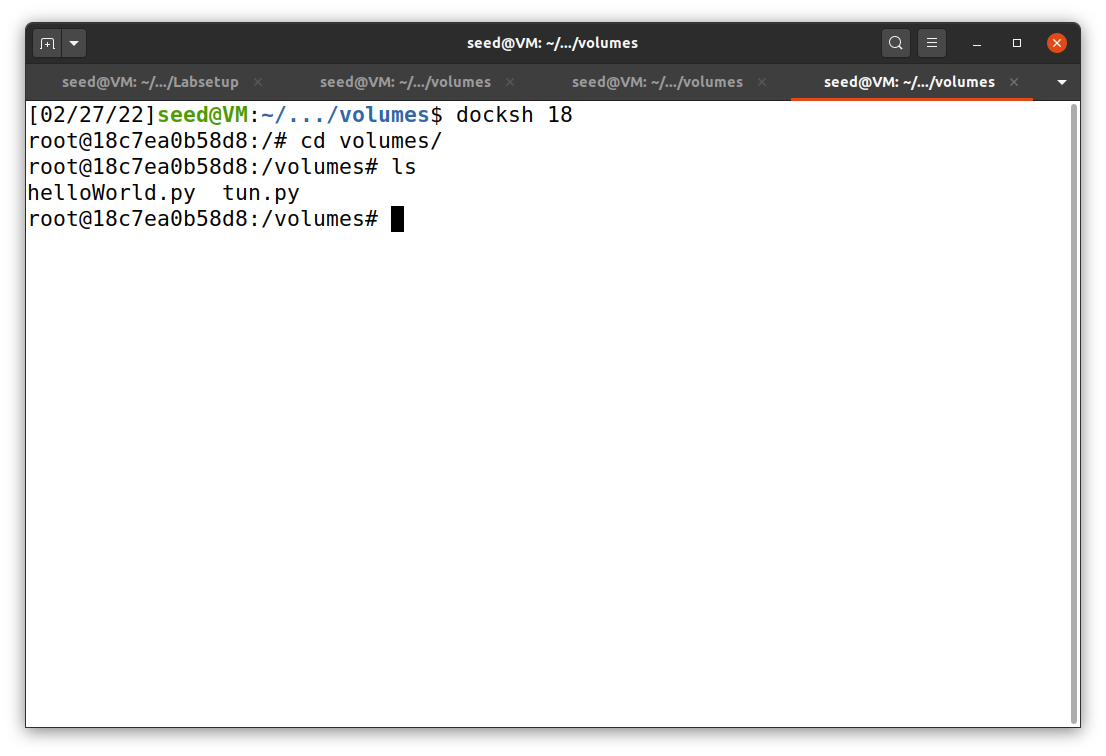
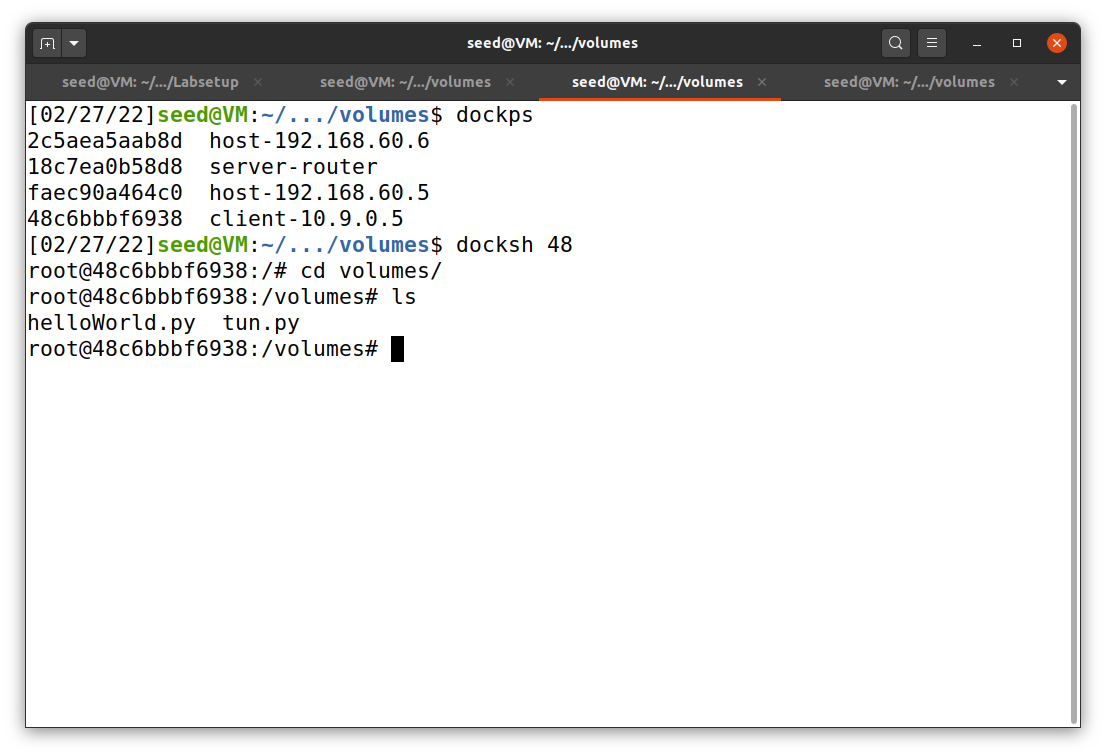
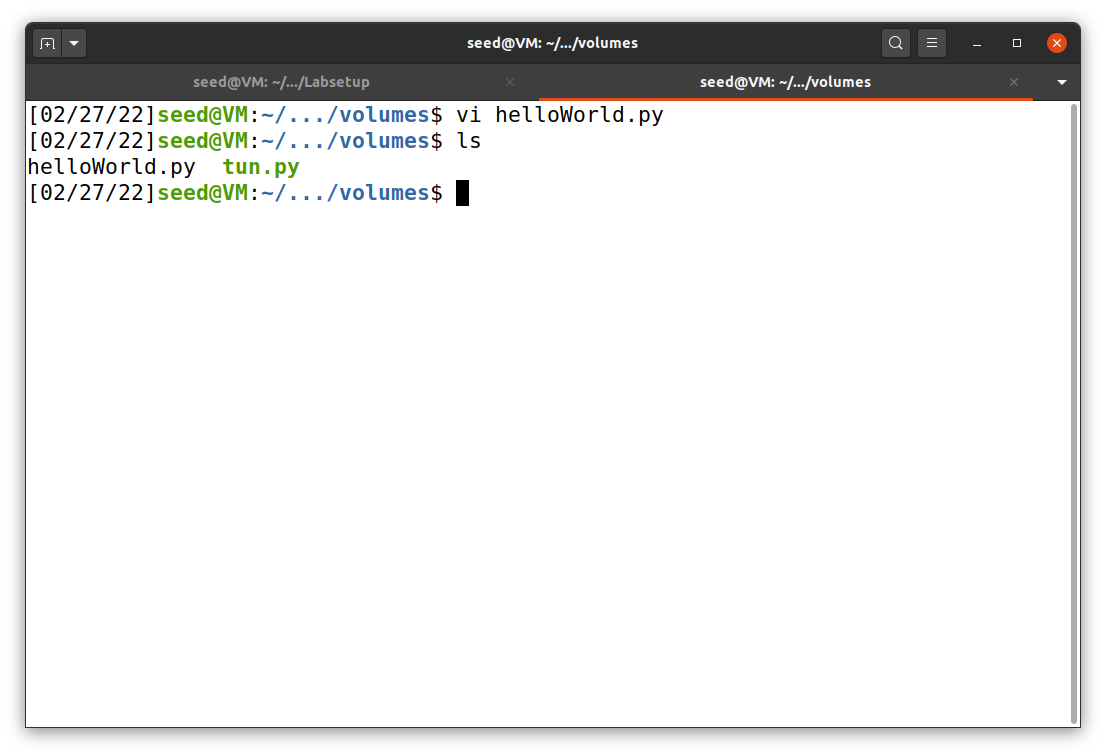
*Network initiation*

**

In this lab’s setup file, it provides the infrastructure for shared folder with the directory “volume.” In other words, the file in volume directory on VM, host U (client 10.9.0.5), and server (VPN) router can be shared between each other. This has been put in place because we do have to use VM to code and share with the hosts. To test this, I created a file “helloWorld.py” on the volume directory and check it on both host U and router (Figure 3).

**Figure 3**

*Shared folder*



Router

host U

VM

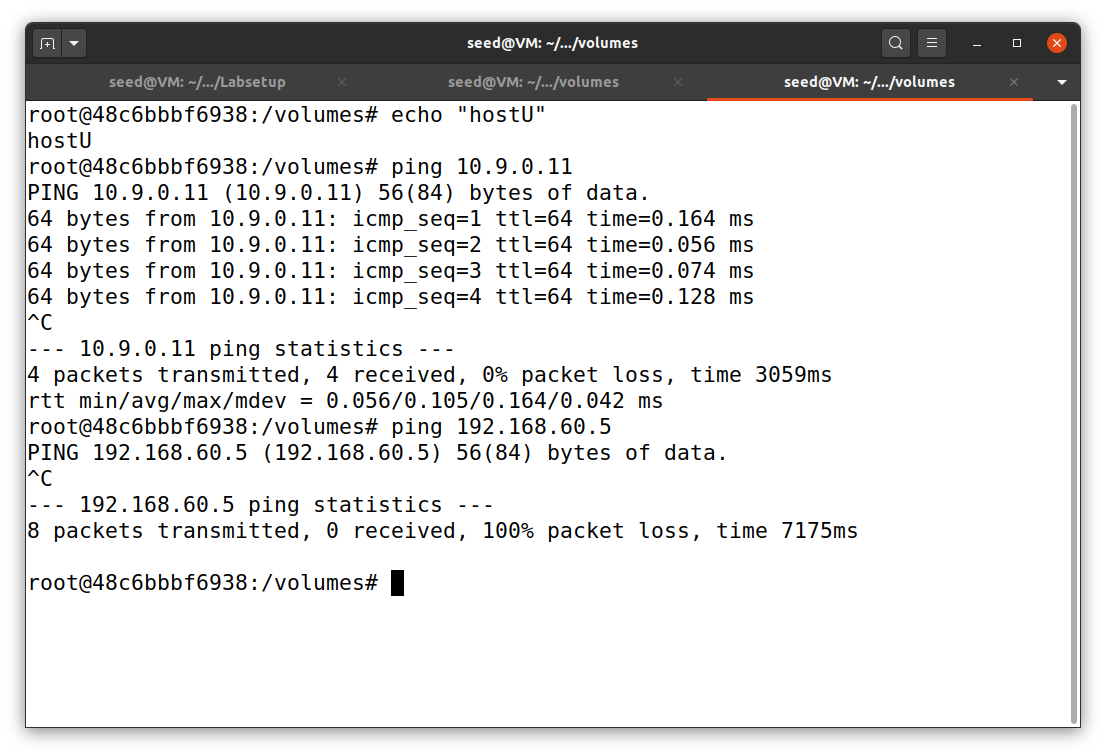
Next, I tested the lab environment to check if it meets with these conditions:

* Host U can communicate with VPN Server.
* VPN Server can communicate with Host V.
* Host U should not be able to communicate with Host V.

First, I docked on host U (10.9.0.5) and try to ping VPN server (router) and Host V (192.168.60.5). The result is shown in Figure 4.

**Figure 4**

*Host U communicates with VPN Server and host V*



Host U can ping router

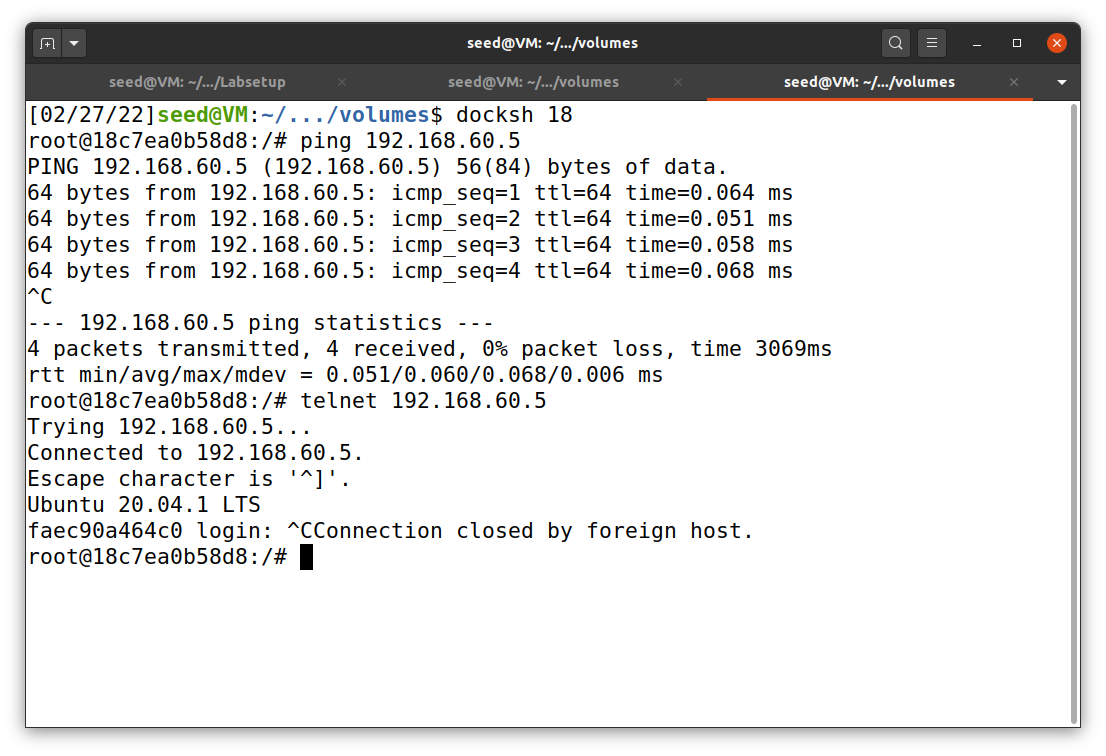
Host U cannot ping host V

host U

Next, I docked on VPN server and try to ping Host V (192.168.60.5), as shown in Figure 5.

**Figure 5**

*VPN Server communicates with host V*



VPN server can telnet host V

VPN server can ping host V

Router

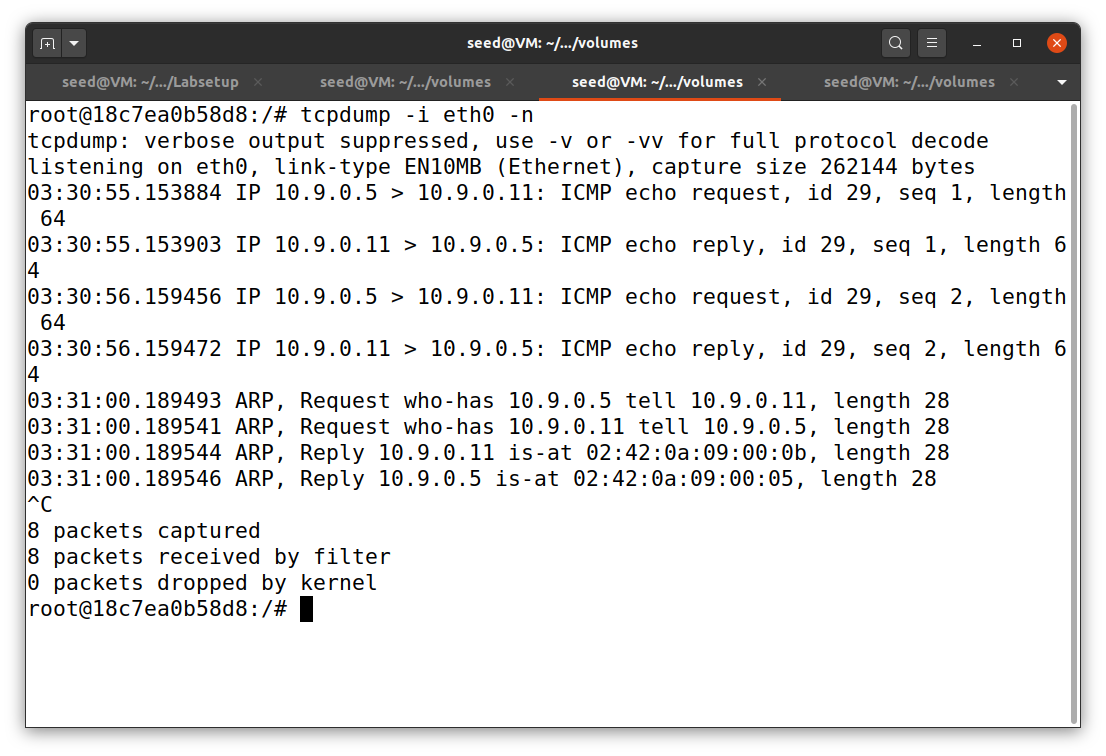
Finally, I docked on VPN server and run tcpdump on the router to sniff the traffic on each of the network., as shown in Figure 6 and 7. The commands below were used to sniff the communication packets on the server that coming from host U and host V.

tcpdump -i eth0 -n

tcpdump -i eth1 -n

**Figure 6**

*Packets sniffing between router and host U*

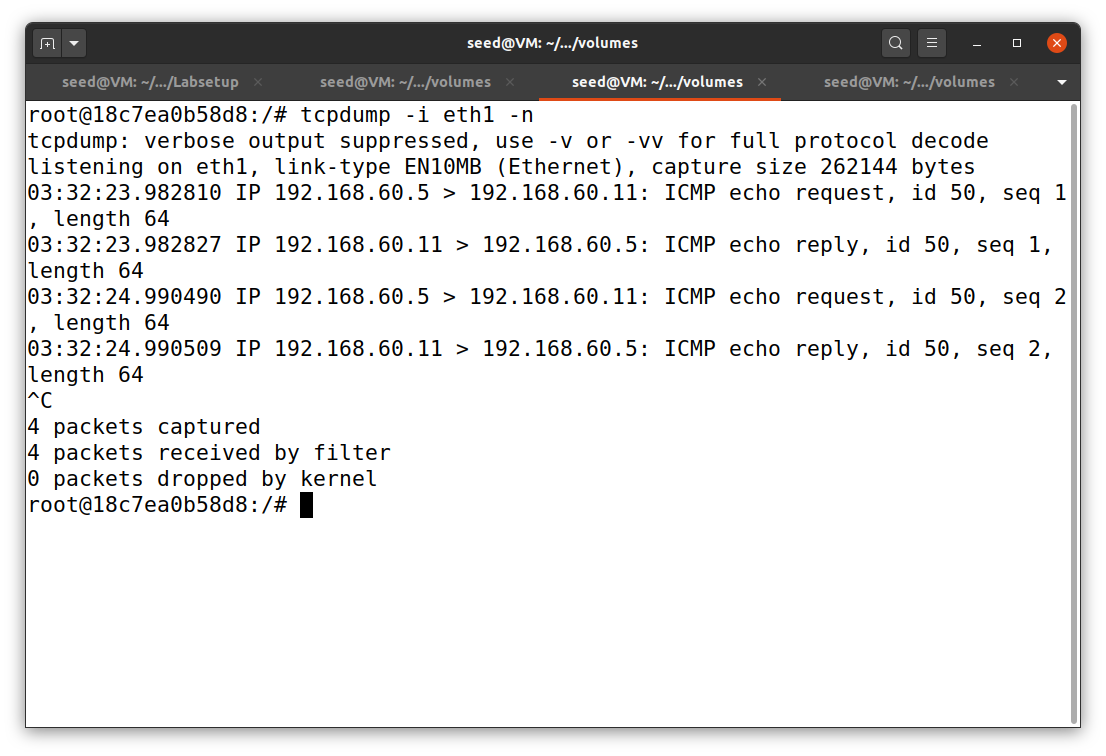
**

Captured ping packets between router and host U

Router

**Figure 7**

*Packets sniffing between router and host V*



Captured ping packets between router and host V

Router

**Task 2: Create and Configure TUN Interface**

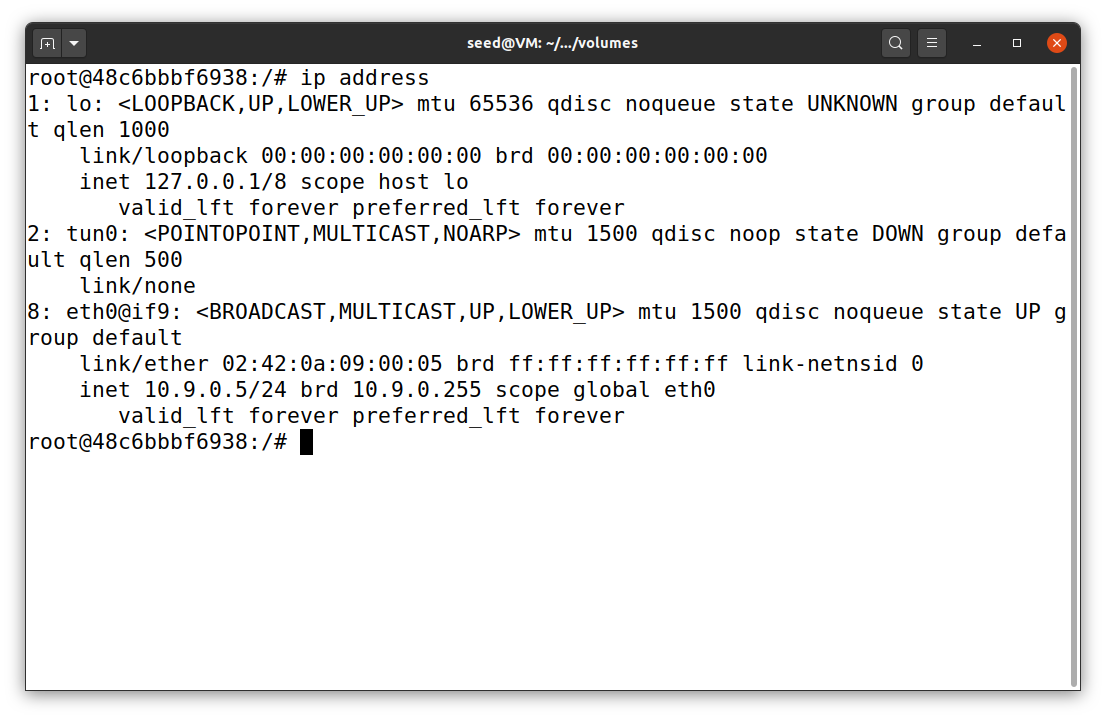
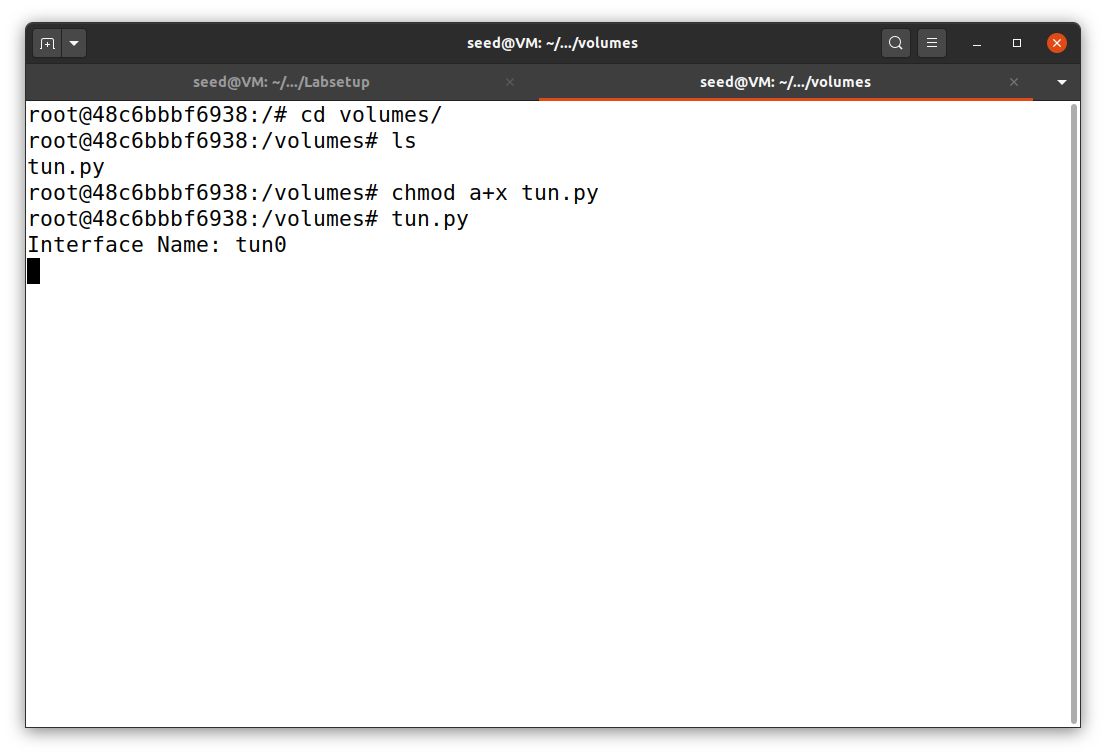
***Task 2.A: Name of the Interface***

In this task, we will get familiarize with TUN interface. First, I docked on host U and run original tun.py file, provided under volume directory in Labsetup.zip. To run it, I have to make it an executable file and access with root privilege with the command below (Figure 8). Then, I used ip address command (on another terminal) to check whether tun interface is up or not (Figure 9).

chmod a+x tun.pytun.py

**Figure 8**

*TUN interface on host U*



host U

terminal 2

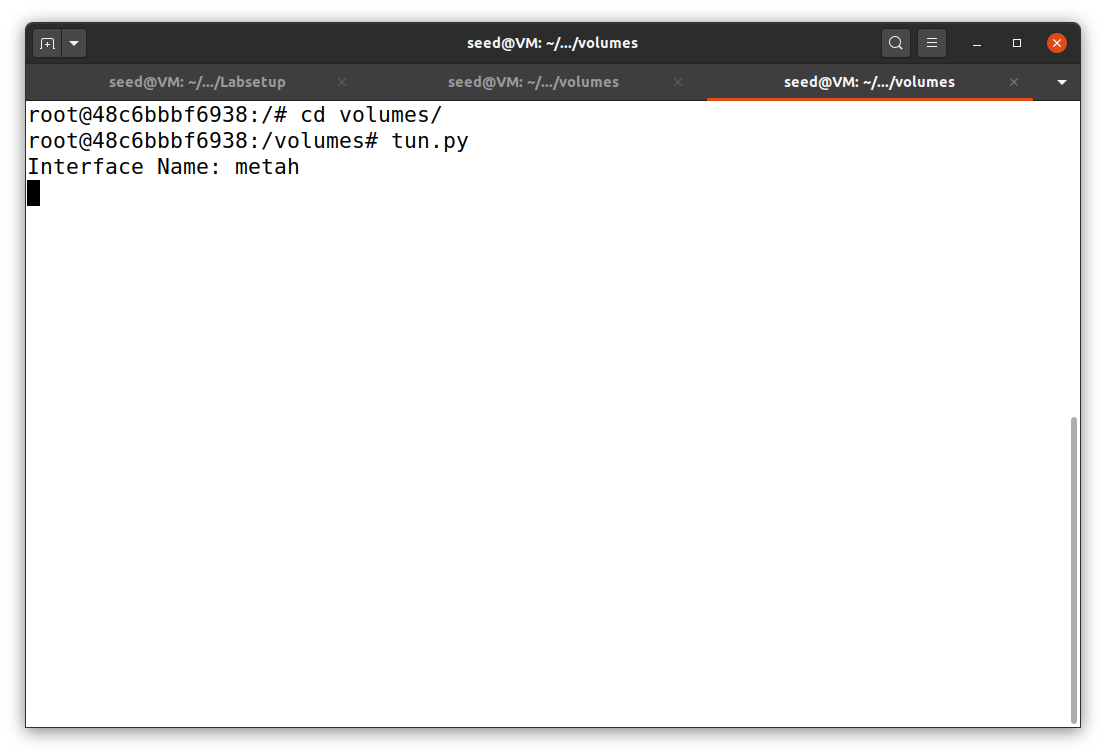
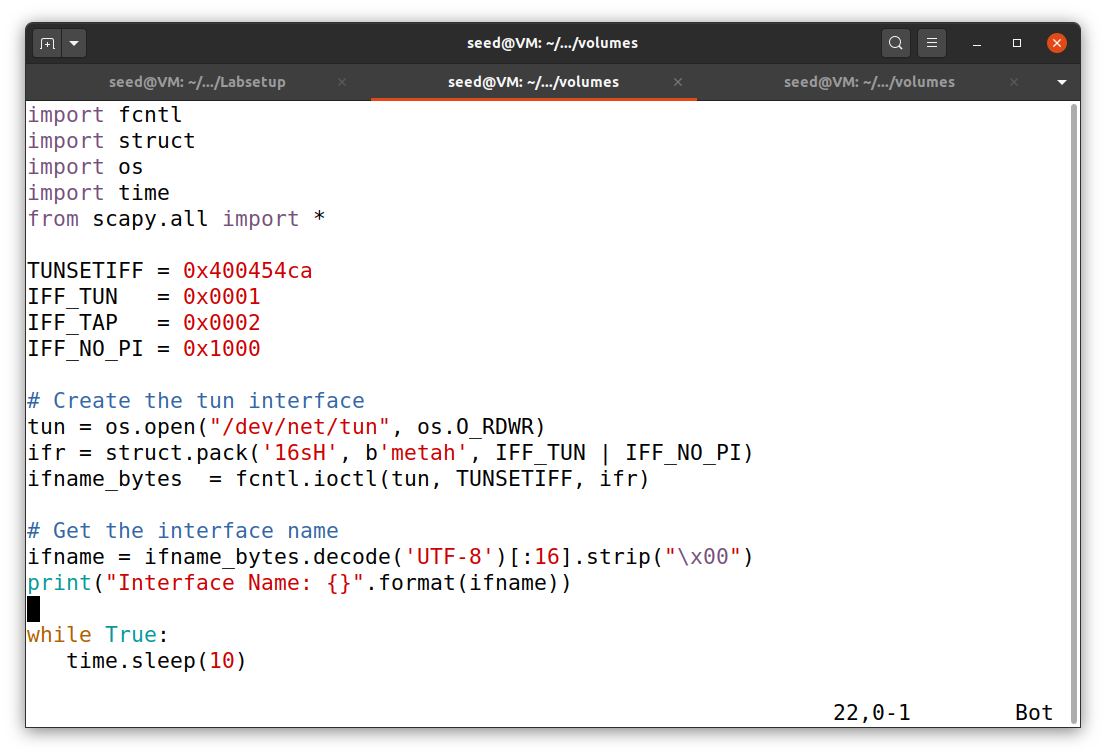
host U

terminal 1

Next, I reconfigured tun.py program to change tun interface name into my last name “metah.” The code is available in the repository on GitHub. The result is shown in Figure 9.

**Figure 9**

*TUN interface “Metah” on host U*



tun.py

host U

terminal 2

host U

terminal 1

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

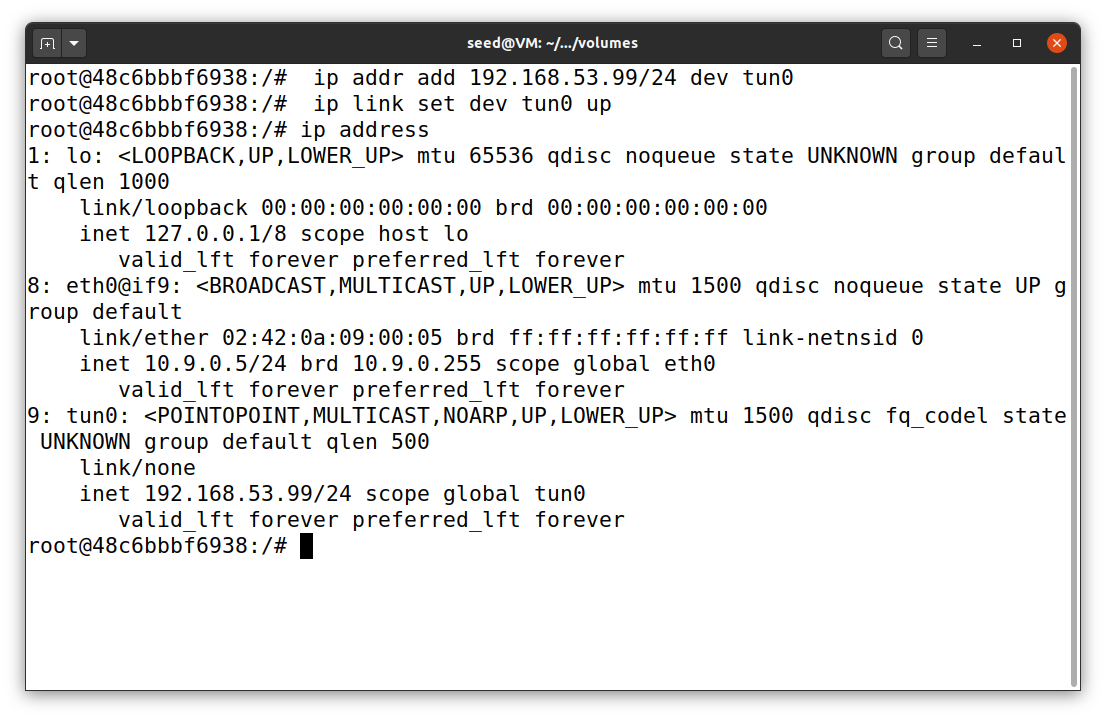
For simplicity, I changed the tun interface name back to tun0 and then run the file tun.py. After TUN interface was initiated, I put two commands below in the second terminal to complete TUN interface set up and used the ip address command to check the result (Figure 10).

ip addr add 192.168.53.99/24 dev tun0

ip link set dev tun0 up

**Figure 10**

*TUN interface complete set up on host U*



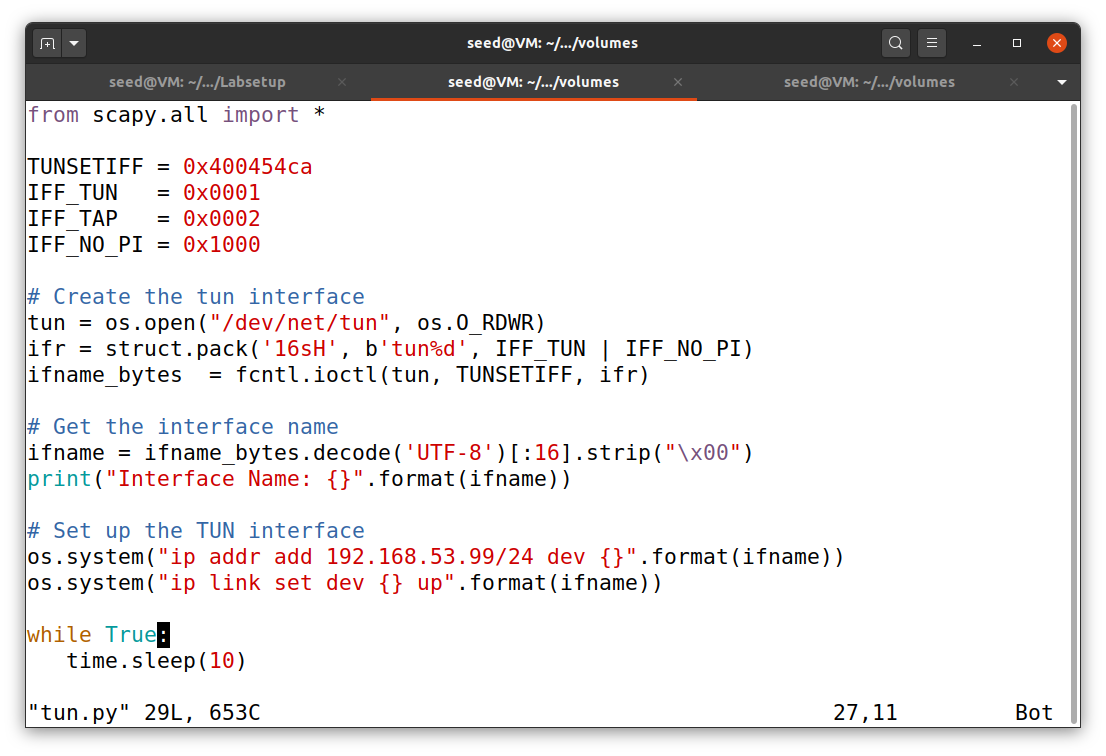
host U

terminal 2

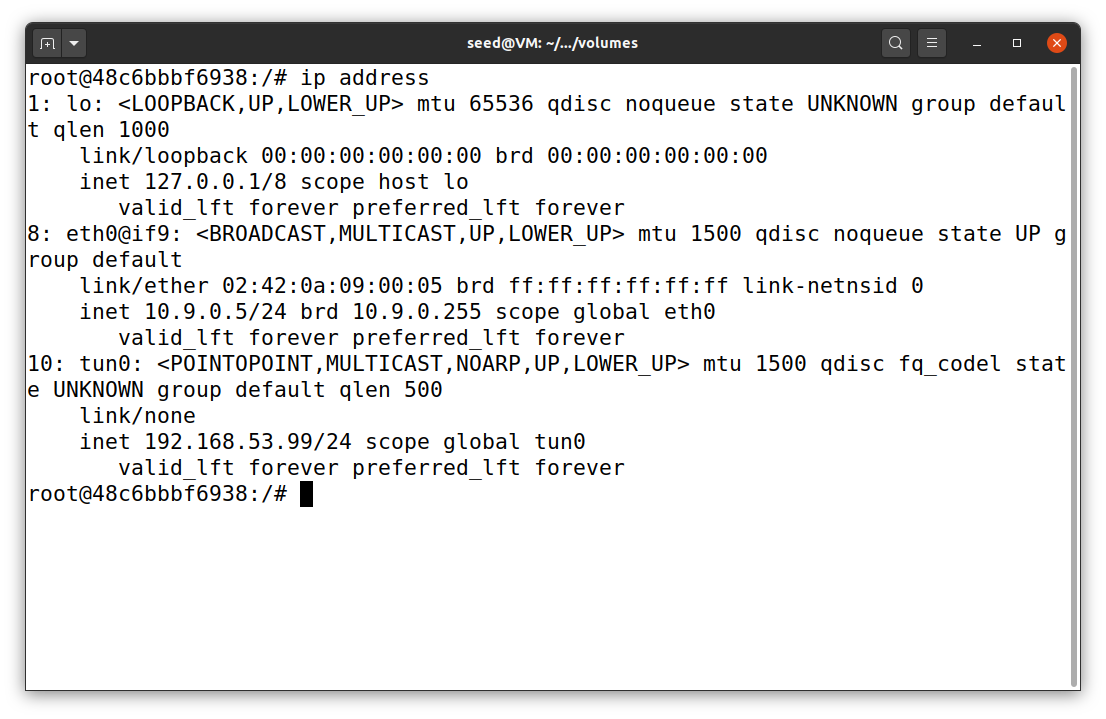
Then, I added the two commands above into the program. So, I do not have to insert the commands every time I run tun.py. The code is available in the repository on GitHub. The added part and result are shown in Figure 11. There is no difference in both ways of setting up TUN interface.

**Figure 11**

*TUN interface complete set up on host U without extra set up commands*



tun.py



host U

terminal 2

***Task 2.C: Read from the TUN Interface***

By following the lab document, I added a few lines of code to tun.py, as shown in Figure 12. Then, I ran the program on host U. While I opened a second terminal, I used a ping command on host U to a host on TUN interface (192.168.53.1). Also, I tried to ping a host in the internal network 192.168.60.0/24 (192.168.60.5). The results are shown in Figure 12.

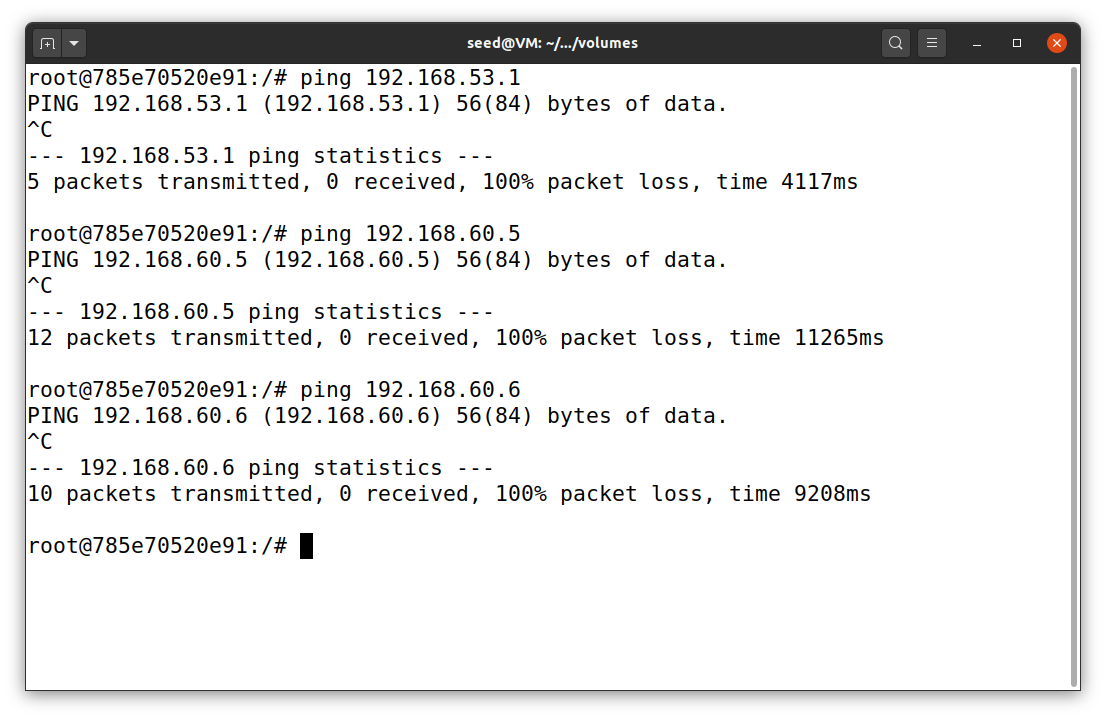
From the results, it can be concluded that tun.py will print out any IP packets on TUN interface. In this case, I used ping, which is ICMP/IP protocol type 8 (echo request), from TUN interface 192.168.53.99 to a host 192.168.53.1. The communication outside of the interface will not be printed out because the code captures only packets that past though TUN interface.

**Figure 12**

*TUN interface complete set up on host U and packets reading*

Text

Description automatically generatedText

Description automatically generated

Only read packets on TUN interface

host U

terminal 1

host U ping two hosts on 192.168.60.0/24

host U ping a host on TUN interface

host U

terminal 2

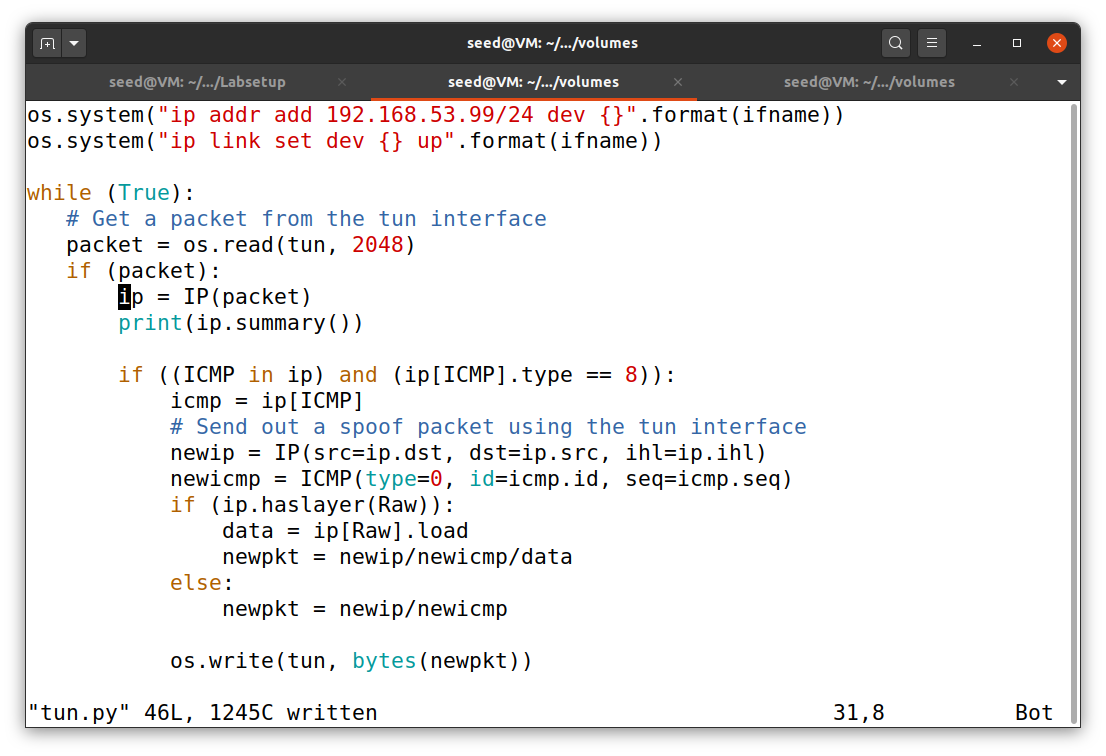
tun.py

***Task 2.D: Write to the TUN Interface***

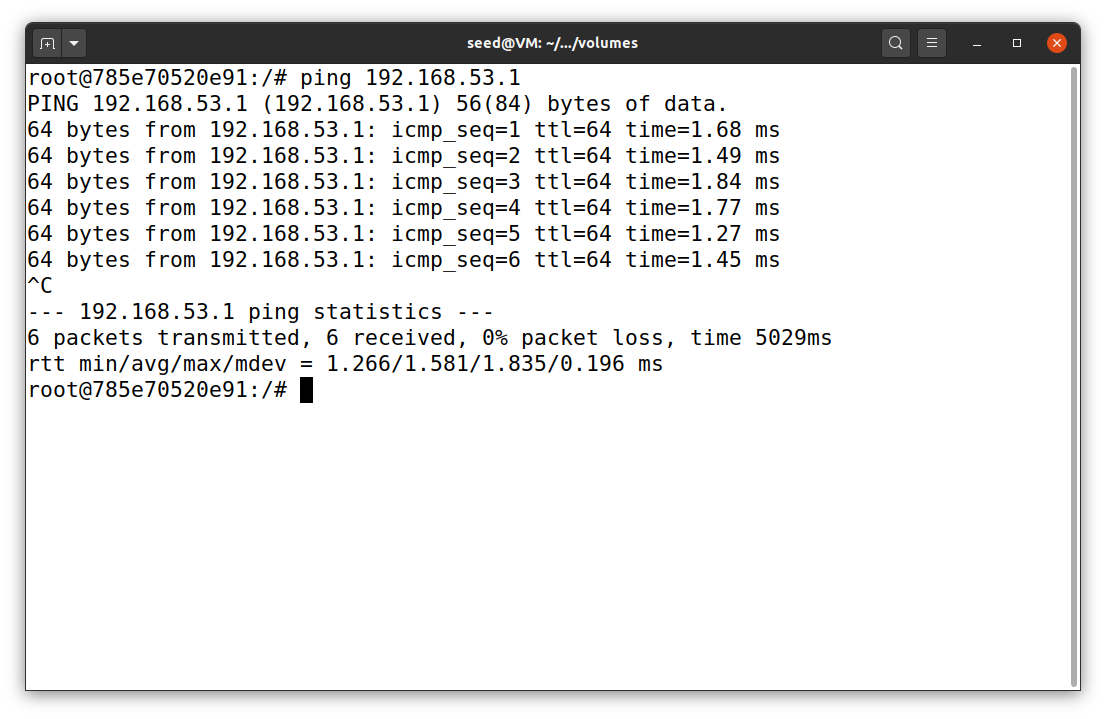
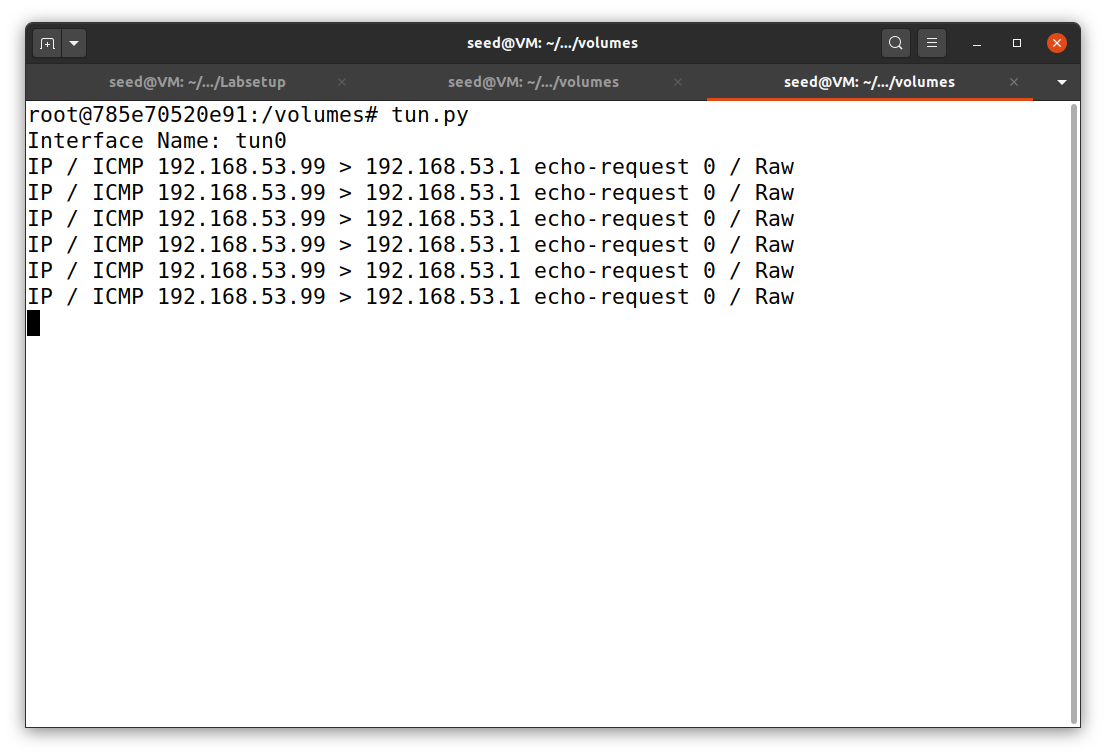
By following the lab document, I added a few lines of code to tun.py, as shown in Figure 13. As shown in the previous task, the program is able to capture any packets on TUN interface. For this added part of the code, if those packets are ICMP type 8 (echo request), the program will now send the spoof ICMP type 0 packets (echo reply) back to the sender (source).

**Figure 13**

*TUN interface with ICMP packets write on host U*



tun.py



Now, it got the echo reply packets from TUN interface

host U

terminal 2

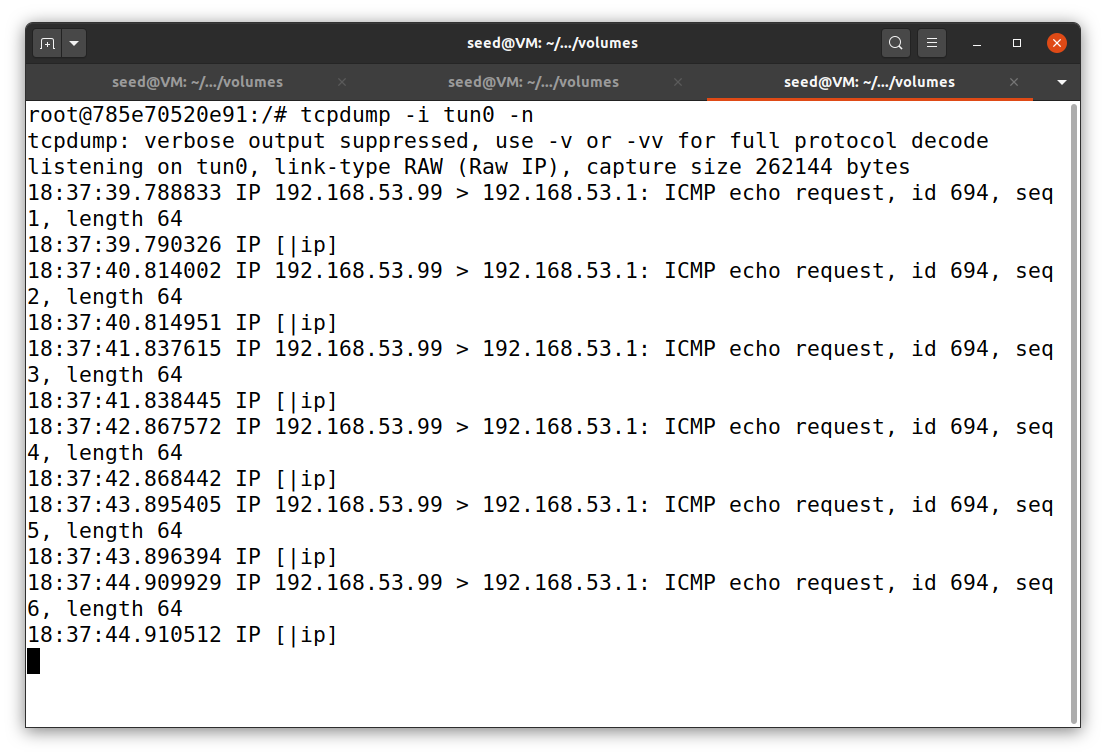
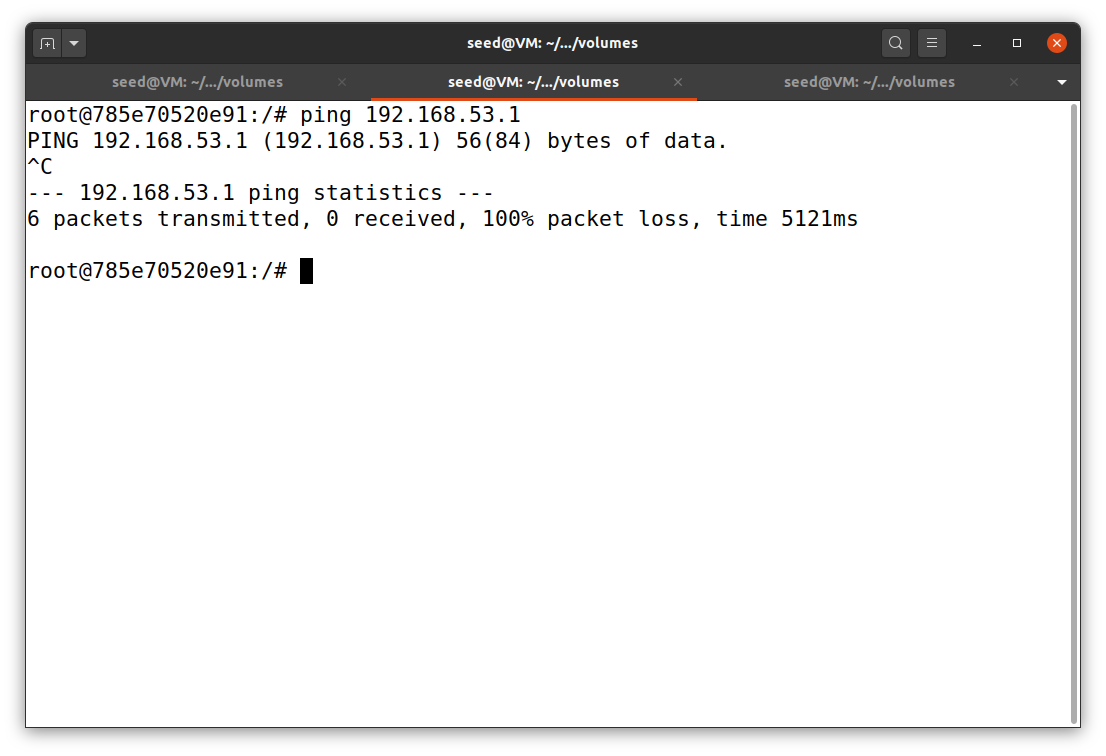
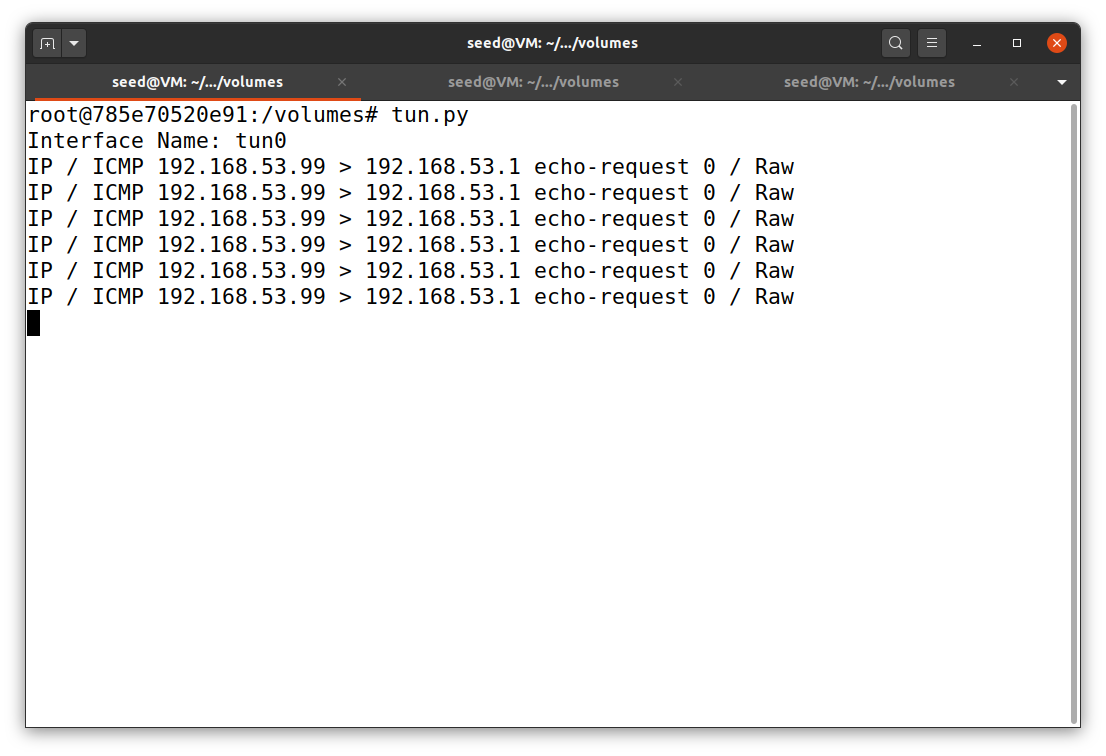
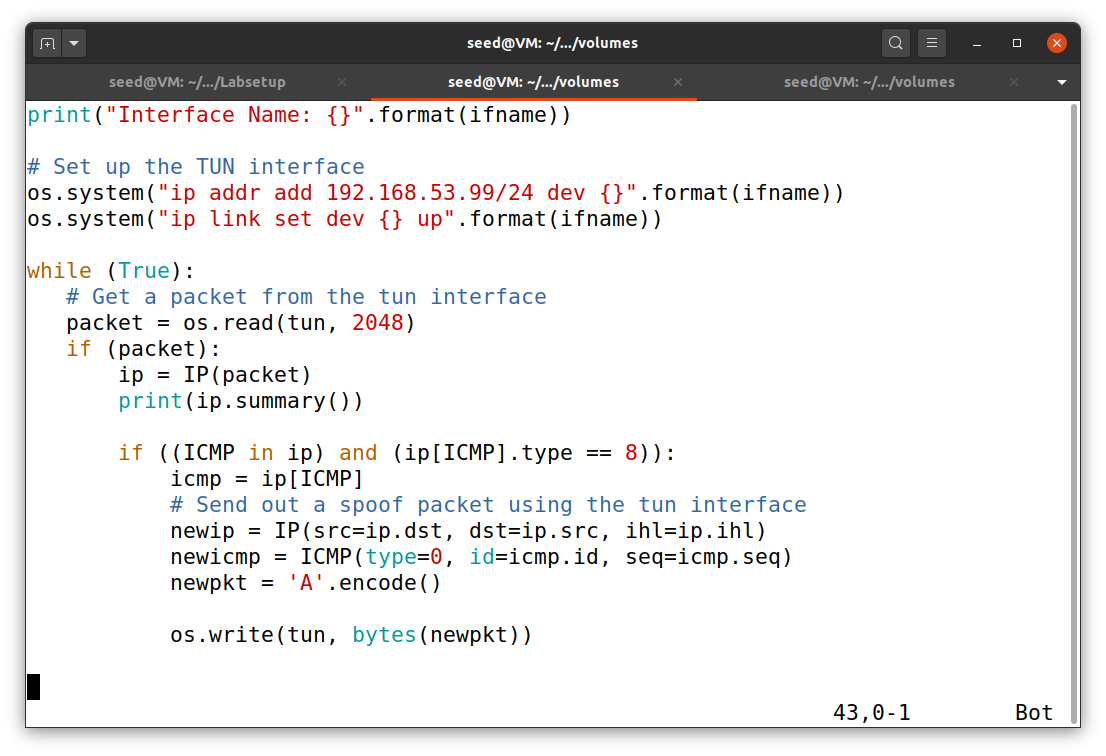
host U

terminal 1

Next, I reconfigured the program again. This time, instead of writing an IP packet to the interface, the program will write some arbitrary data to the interface (Figure 14). To capture packets on TUN interface, I also used tcpdump -I tun0 -n on host U.

**Figure 14**

*TUN interface with arbitrary data write on host U*



Just send a string “A”

host U

terminal 3

host U

terminal 2

host U

terminal 1

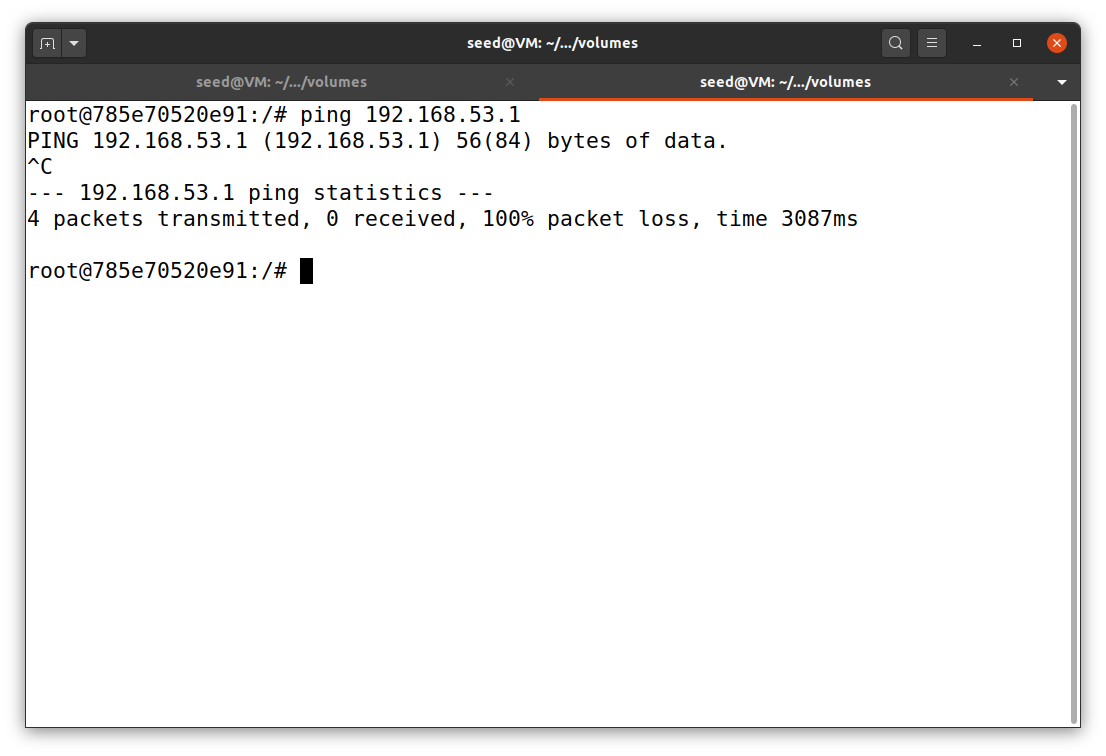
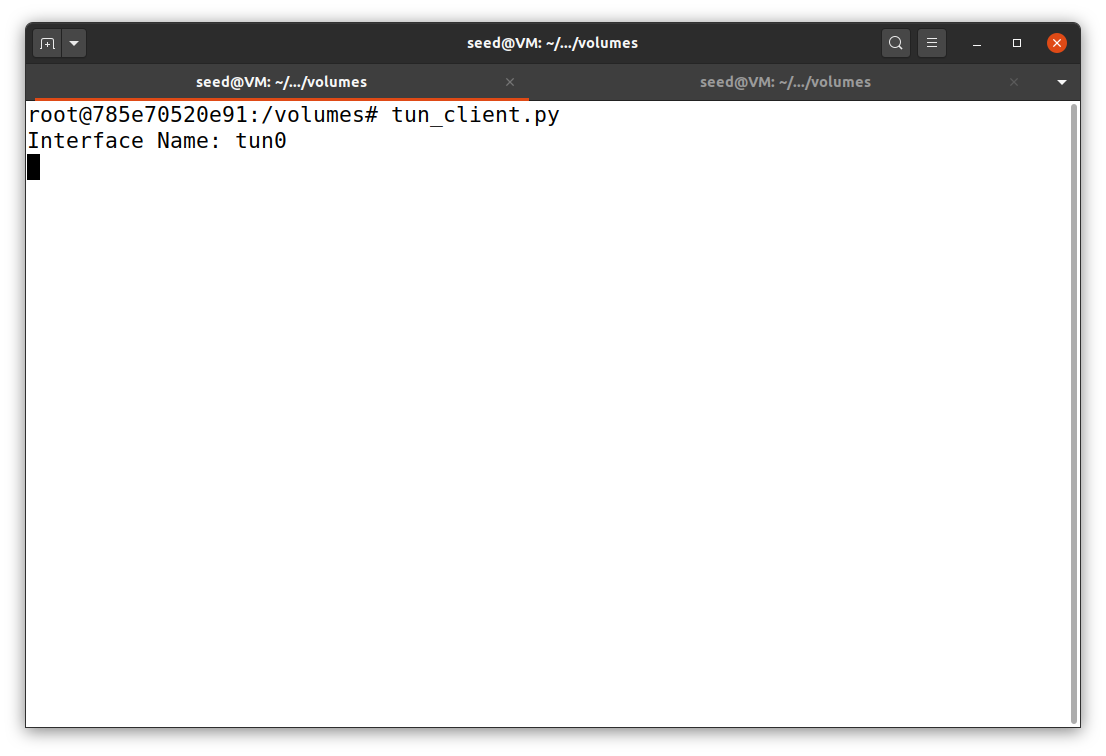
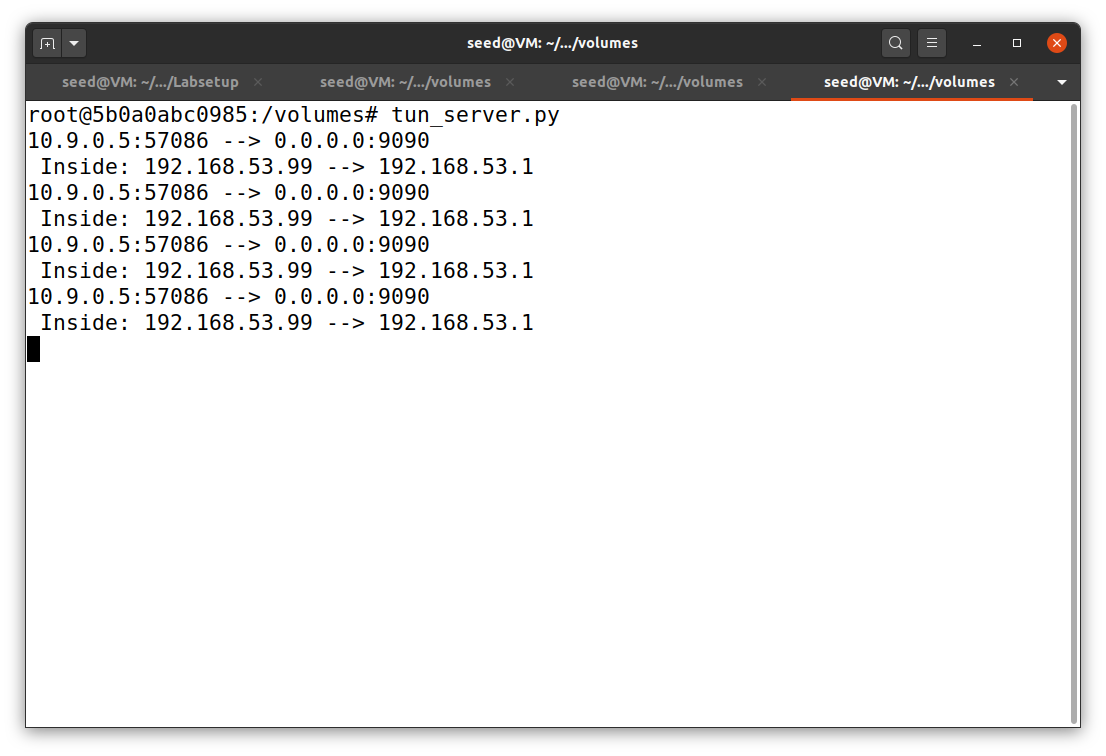
tun.py

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

I created the other two files for this task: tun\_server.py and tun\_client.py. The first part is to use the code given in the lab direction. So, only the second part has been uploaded to my repository. The tun\_server.py program will be run on VPN Server, and then run tun client.py on Host U. Next, I used host U to ping a host on TUN interface (192.168.53.1). The results are shown in Figure 15. As shown below, VPN server print out UDP packets on port 9090 that were sent by 10.9.0.5 (host U) on TUN interface (192.168.53.99) to host 192.168.53.1.

**Figure 15**

*UDP Packets via TUN interface*

****

host U ping a host on TUN interface

VPN server

host U

terminal 2

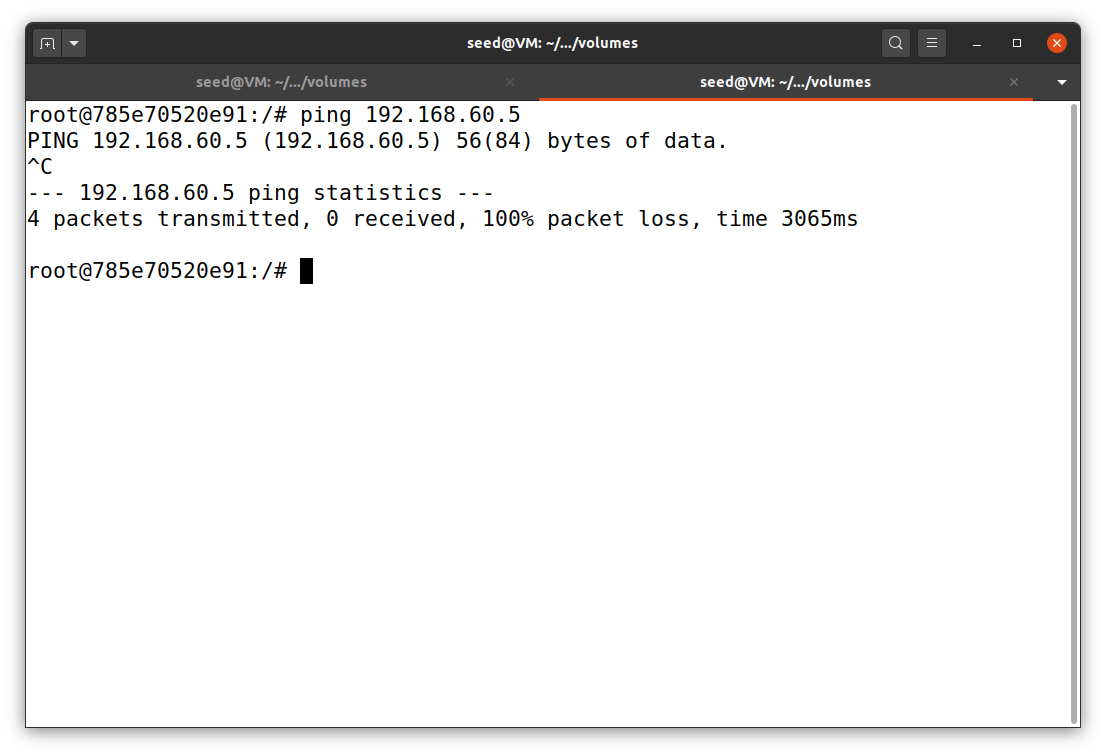
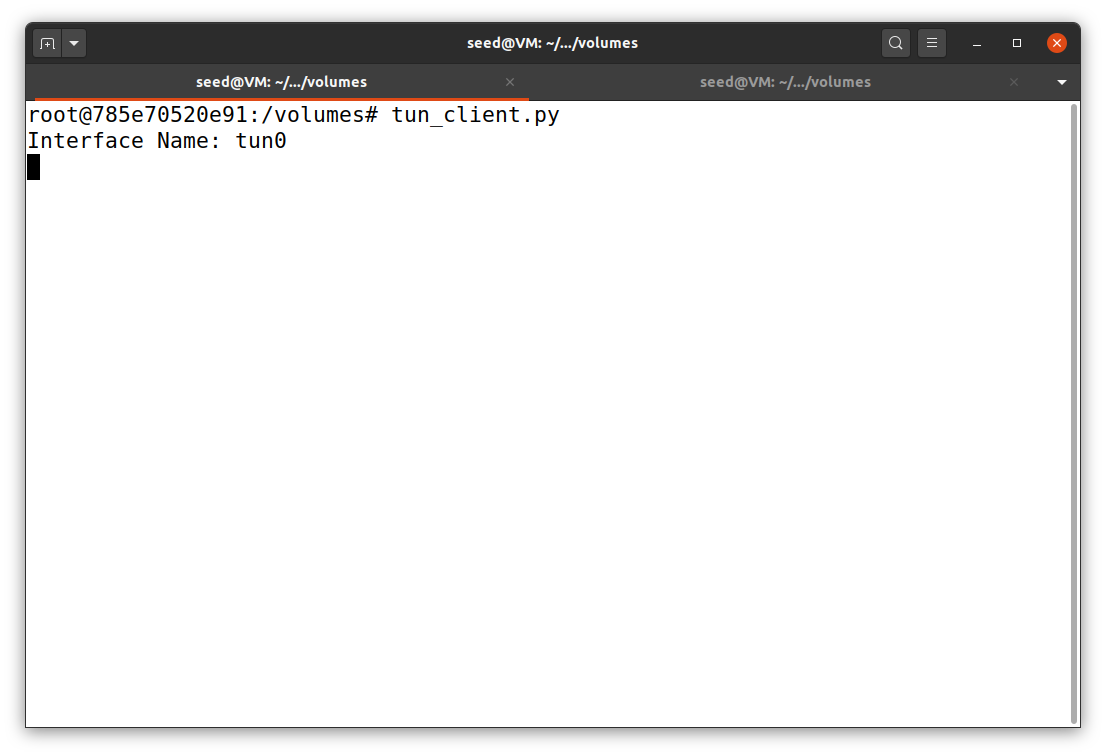
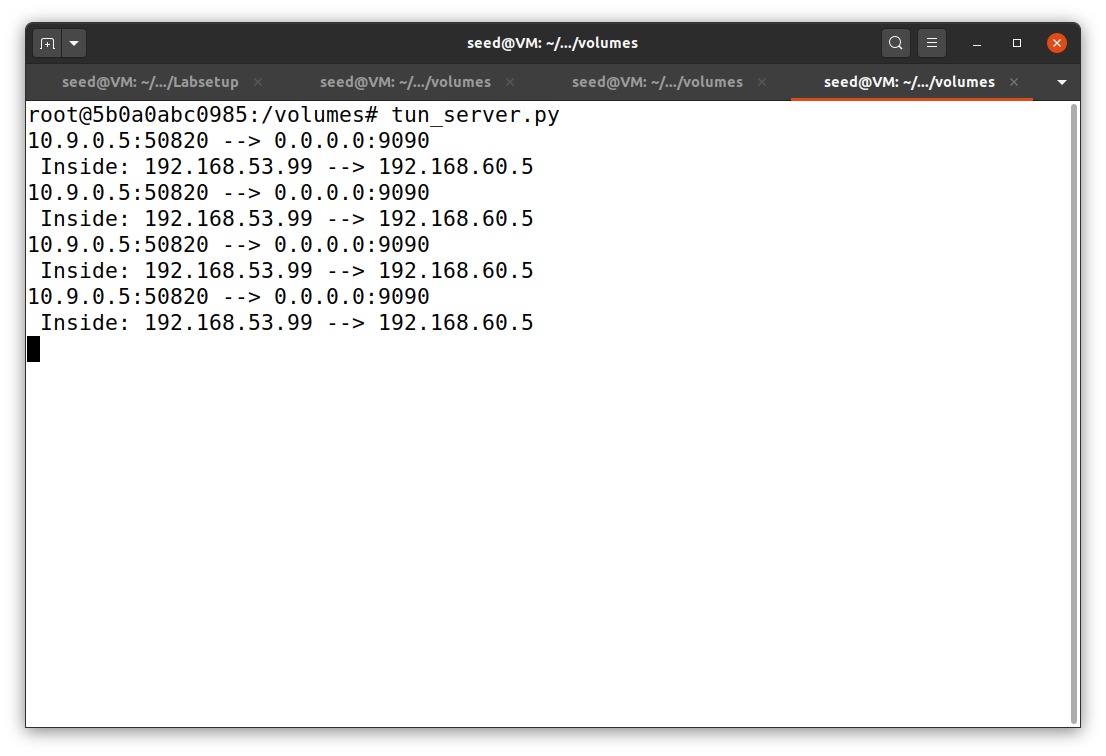
host U

terminal 1

Next, I reconfigured the code to add a route to network 192.168.60.0/24. This will allow host U to ping the internal hosts via TUN interface. The results are shown in Figure 16. The program tun\_server.py clearly shows that it prints out packets on TUN interface to destination of 192.168.60.5, which is the host V inside network 192.168.60.0/24.

**Figure 16**

*Connection tracking on UDP*



Packets go through TUN interface to host V

host U ping host V

VPN server

host U

terminal 2

host U

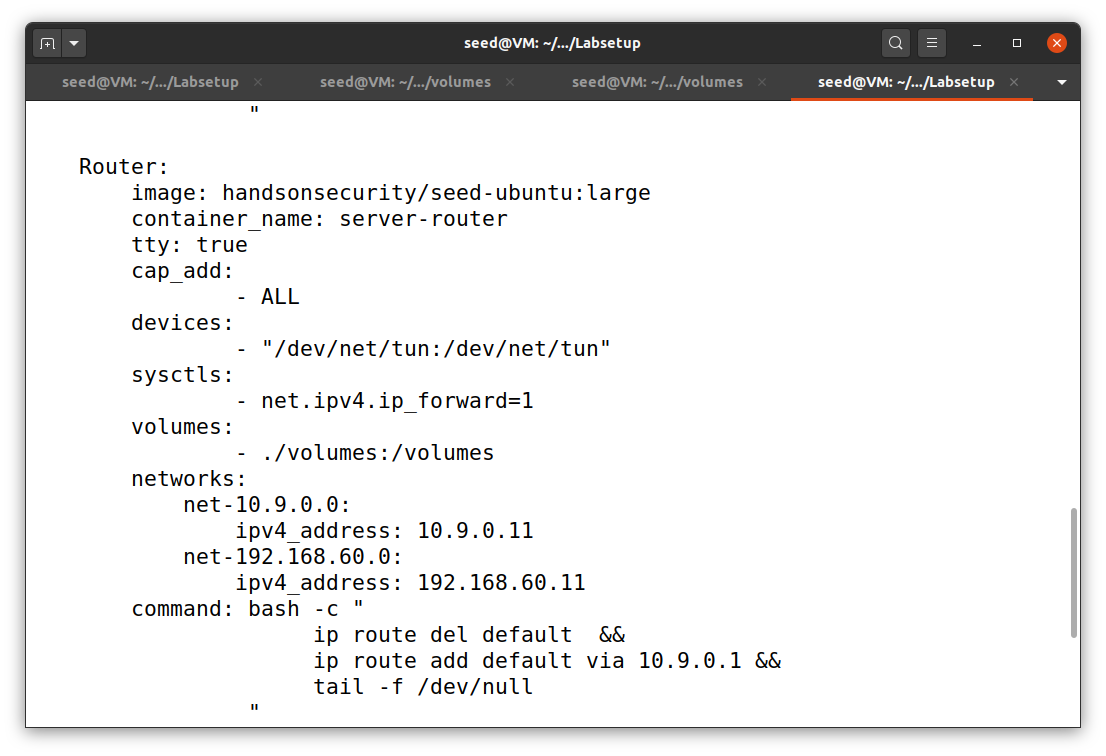
terminal 1

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

Before tackling this task, I just checked the docker-compose.yml to make sure that enable the IP forwarding (which it does), as shown in Figure 17. Then, I edited the tun\_server.py program (available on my GitHub). This time, the tun\_server.py has its own part of TUN interface creation. After getting the data from the socket interface, it will write the packet to the TUN interface. I used the command tcpdump -i tun0 -n on the other terminal of VPN sever to grab any packets that going though TUN interface. The programs tun\_server.py and tun\_client.py are initiated on VPN server and host U, respectively. Finally, on another terminal of host U, I ping host U to host V via tun0. The result indicates that the tcpdump command can capture TUN packets, which are heading to host V (Figure 18).

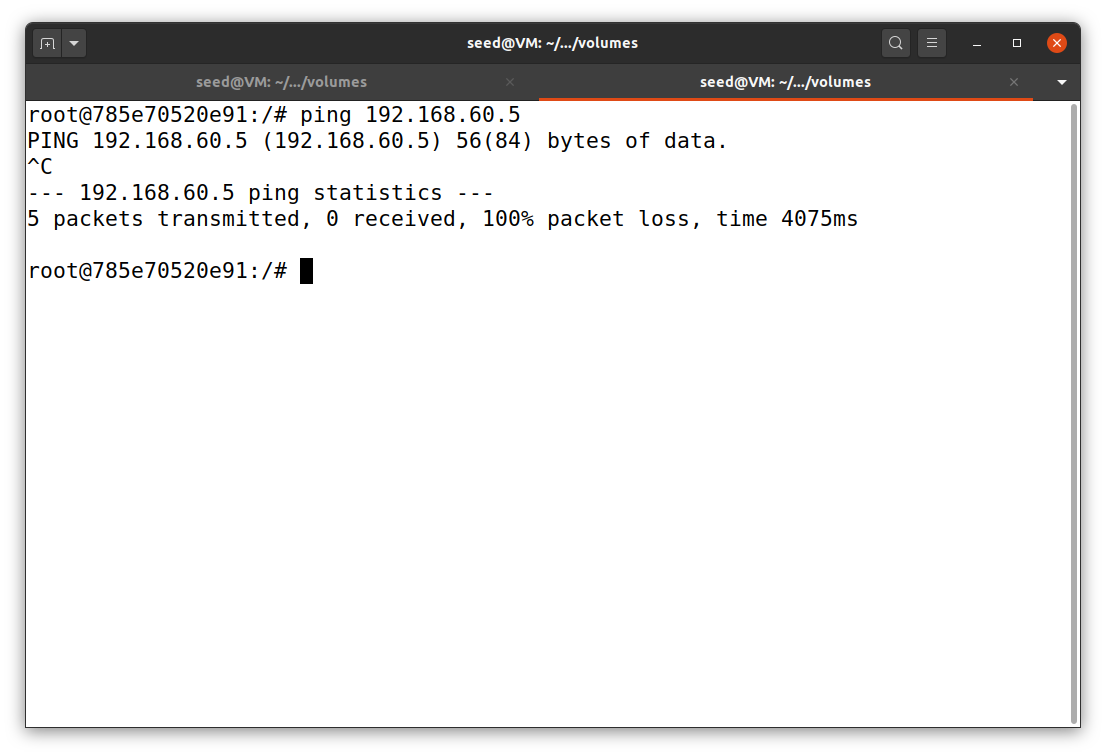
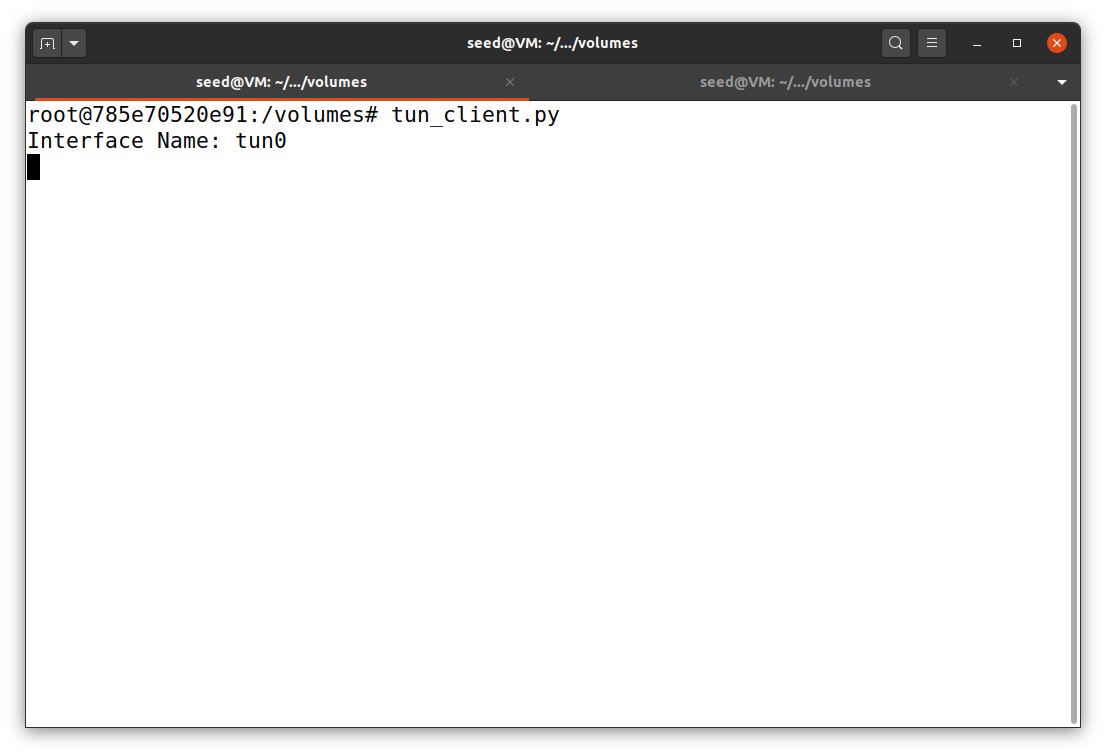
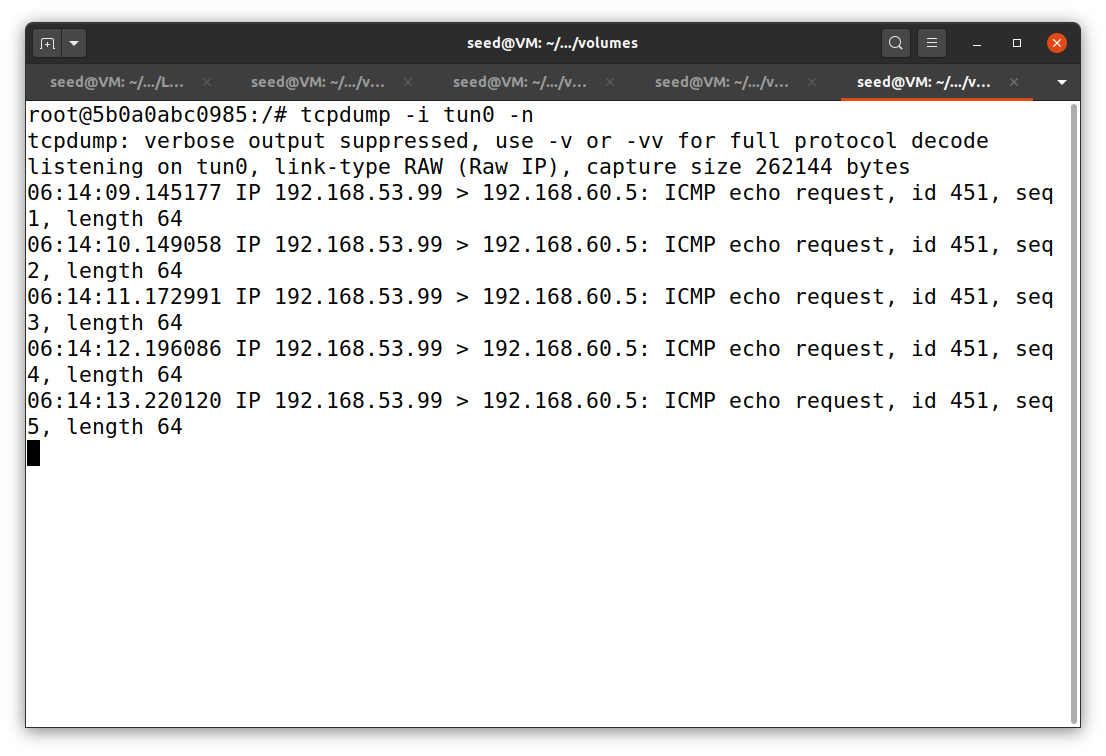
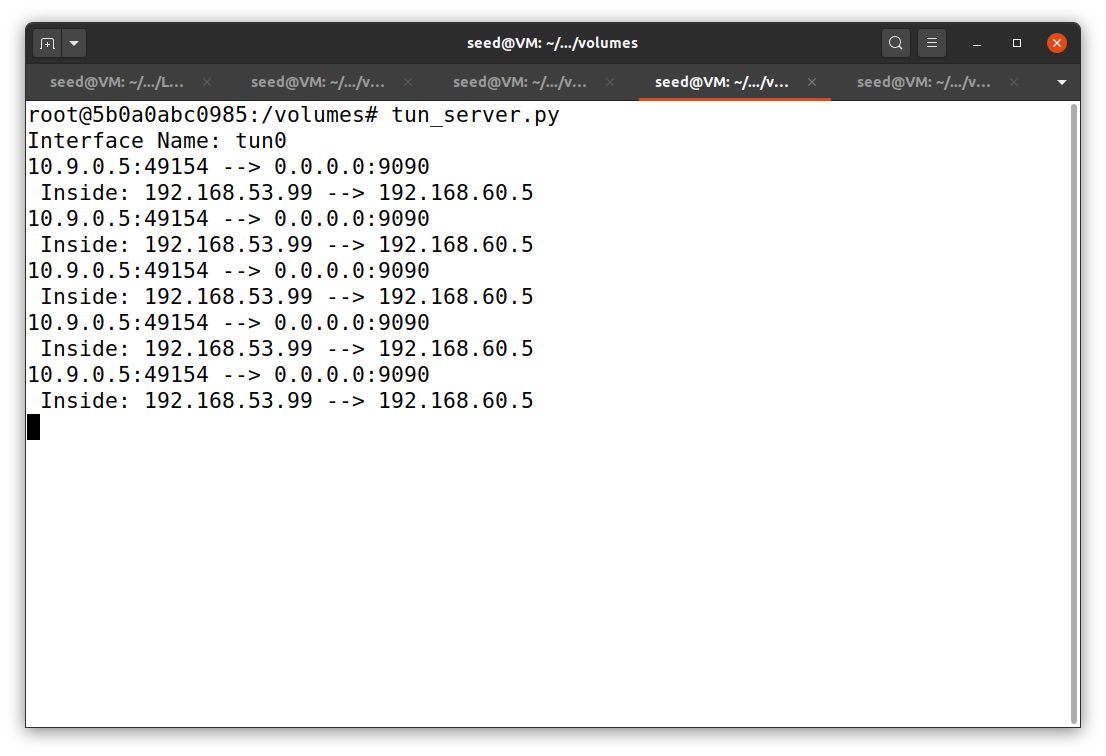
**Figure 17**

*docker-compose.yml*



**Figure 18**

*The results of task 4*



VPN server

terminal 2

VPN server

terminal 1

Tcpdump can captured packets on TUN interface heading to host V

host U ping host V

host U

terminal 1

host U

terminal 2

Packets go through TUN interface to host V

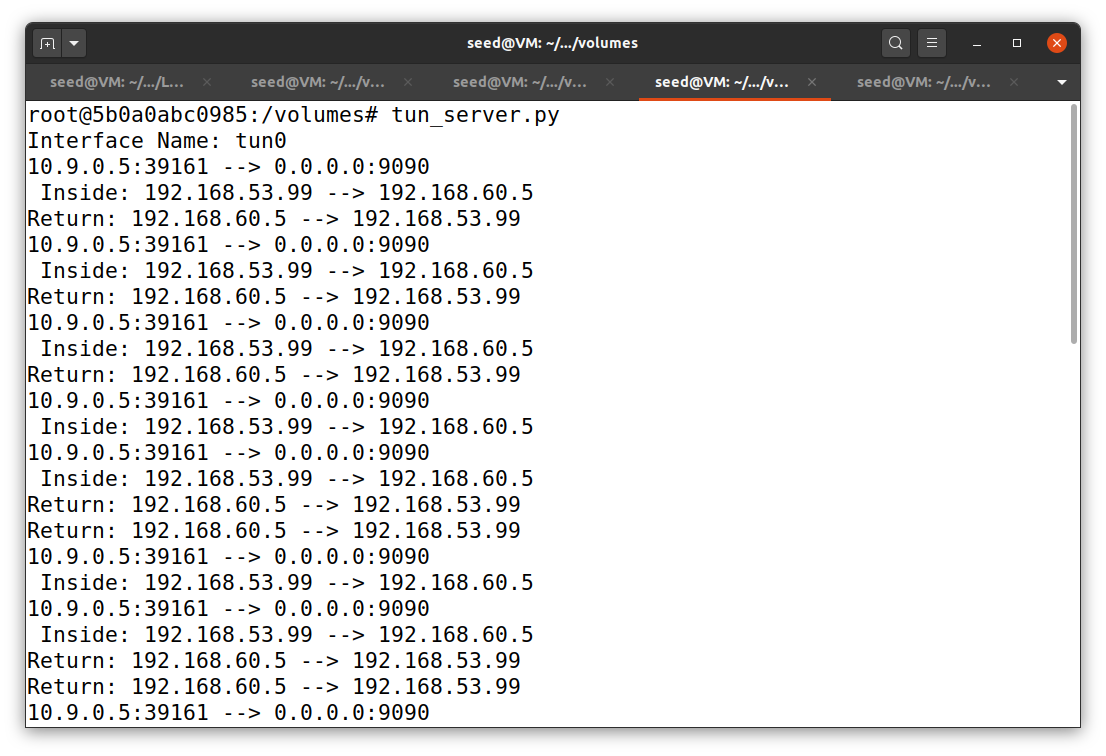
**Task 5: Handling Traffic in Both Directions**

In this task, I modified the two programs to enable host U to connect to host V and vise versa. The codes were uploaded to my GitHub repository. To test this experiment, I ran the tun\_server.py on VPN server and tun\_client.py on host U. Then, I used host U to ping and telnet host V. All traffics were recorded using tcpdump on VPN server with the command below. The results are shown in Figure 19 to 21.

tcpdump -i tun0 -n

**Figure 19**

*VPN server of task 5*

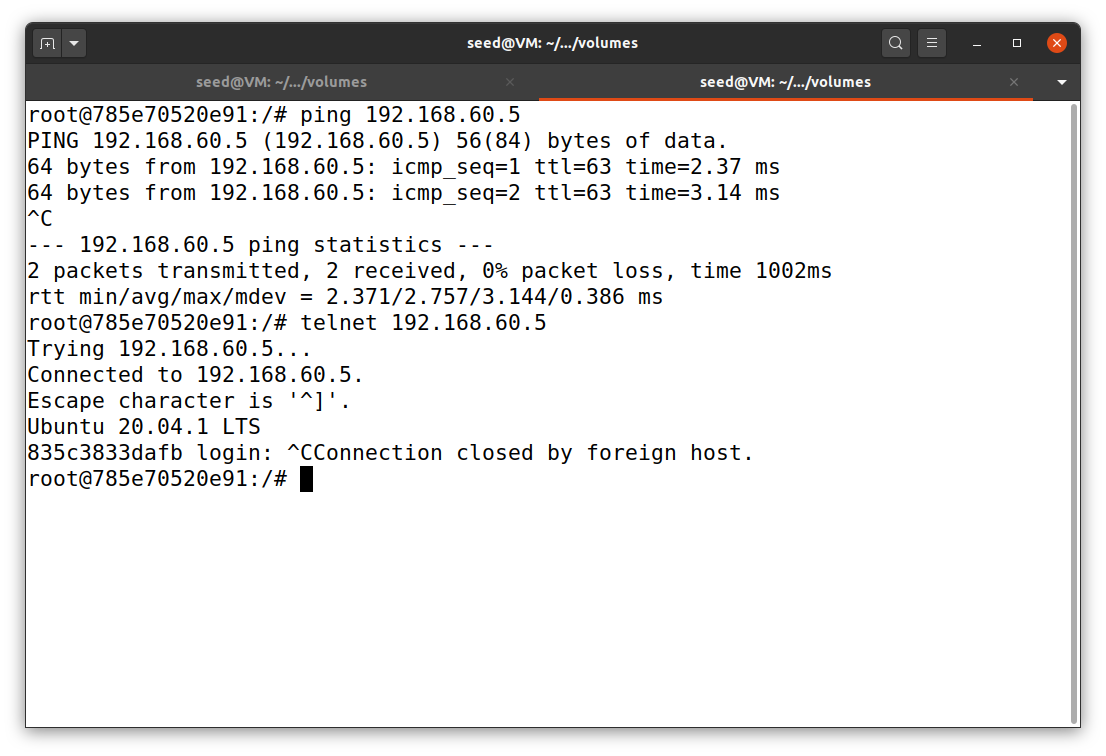
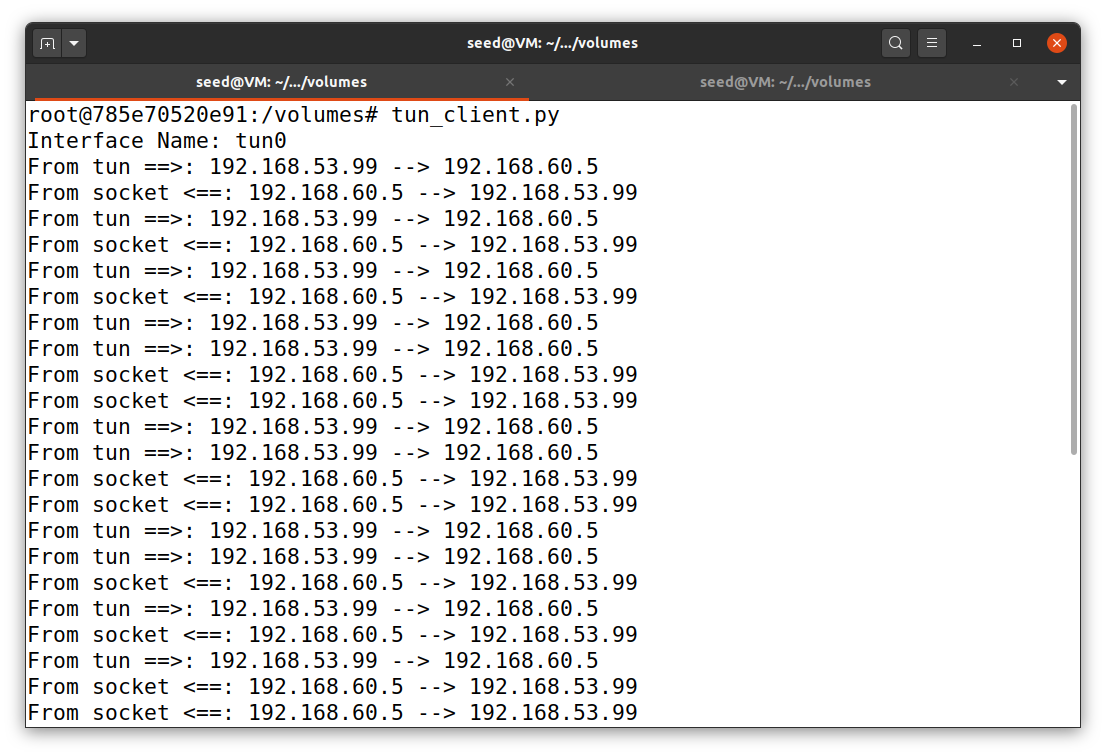


VPN server

terminal 1

**Figure 20**

*Host U of task 5*



host U

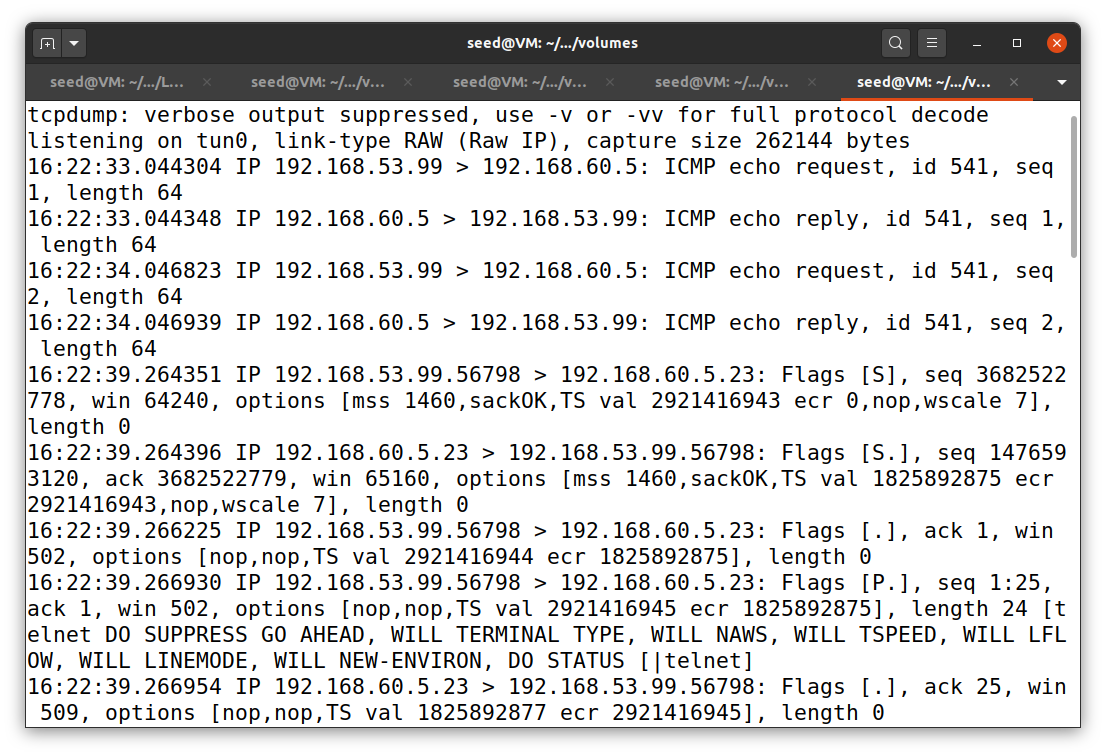
terminal 2

host U

terminal 1

**Figure 21**

*Tcpdump result of task 5*



VPN server

terminal 2

The result in Figure 21 shows that I used two types of connection: ping and telnet. When I ping from host U to host V, tcpdump clearly shows that there are packets coming from TUN interface 192.168.53.99 (from host U) to host V at 192.168.60.5 with ICMP echo request. Then, host V reply to the TUN interface, which goes back to host U, with ICMP echo reply. We can also observe this in Figure 20 (host U terminal 1) that host U got reply packets from host V.

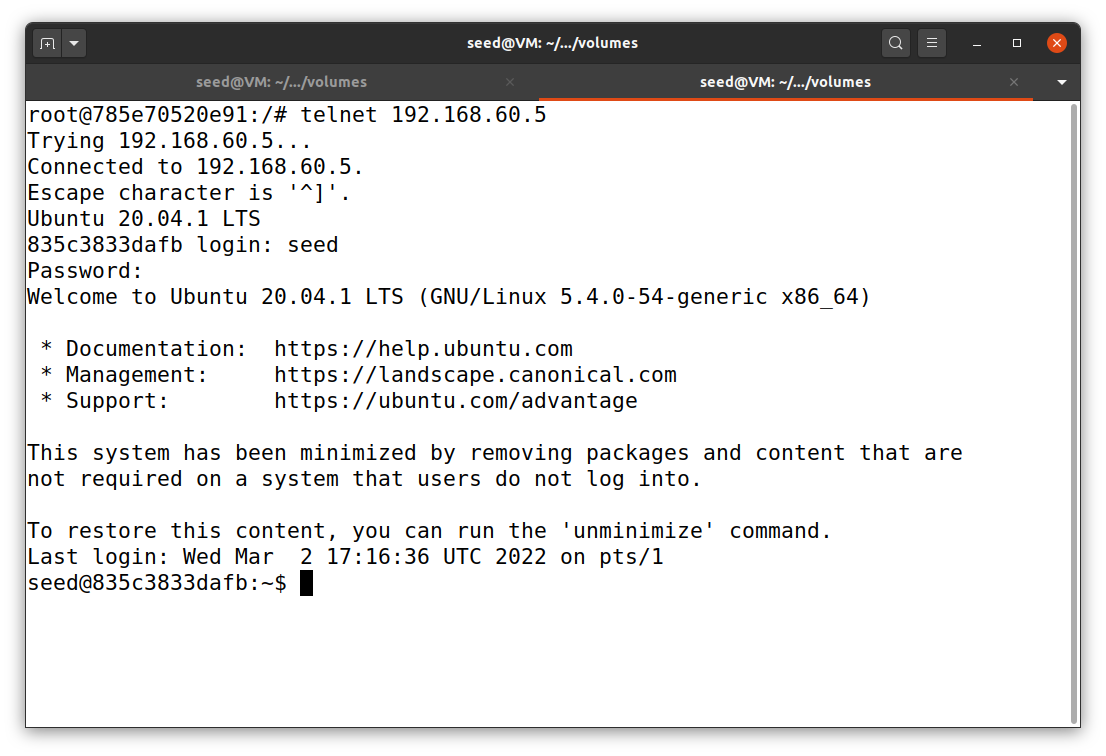
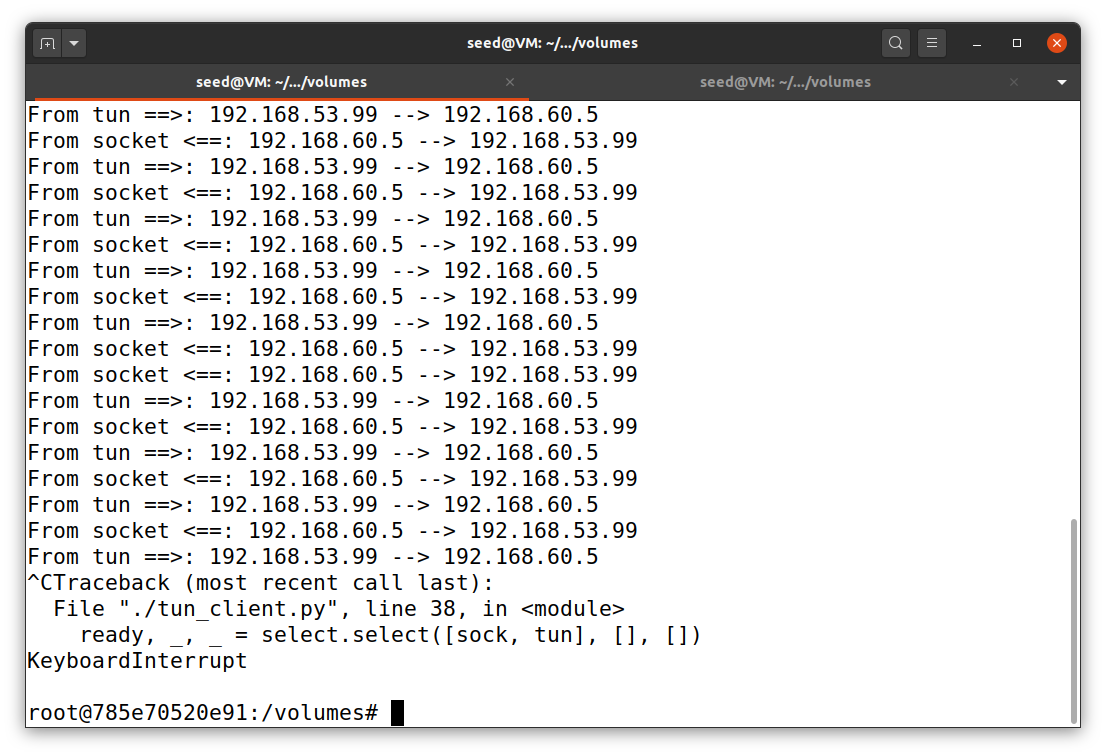
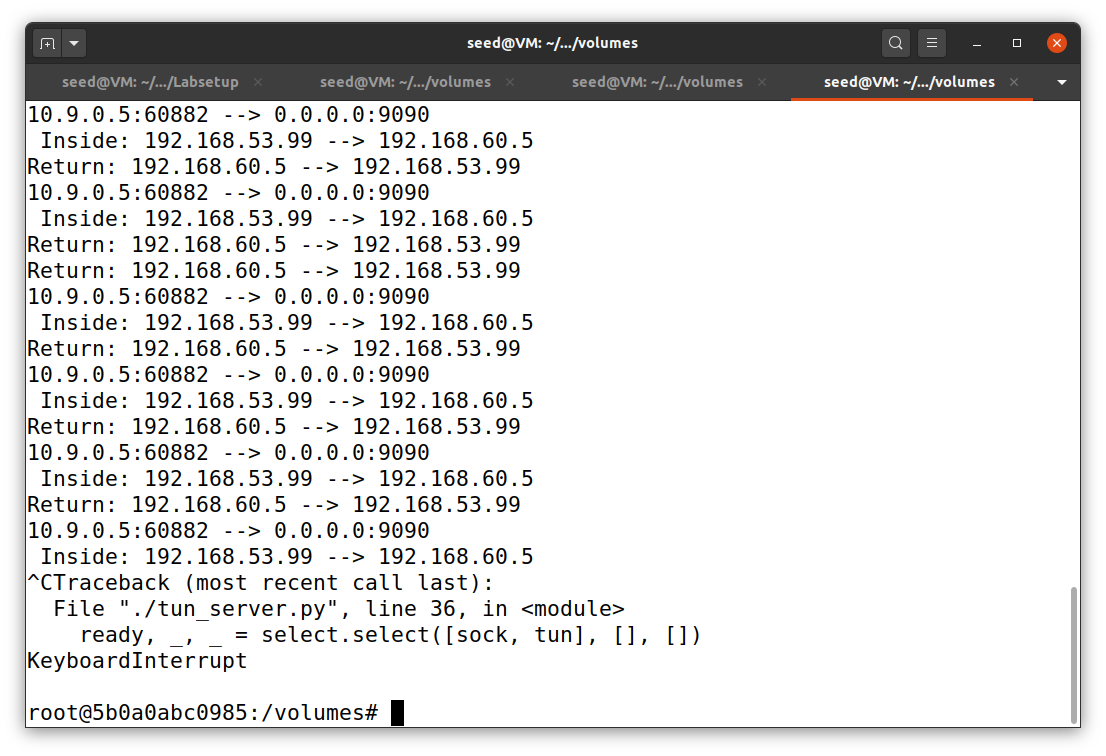
After that, I telnet host V from host U. Again, we can view the result by using tcpdump. Figure 21 shows that there are TCP packets from TUN interface 192.168.53.99 (host U) to host V with SYN flag to start the connection. Host V, then, replied back to host U via TUN interface with SYN flag and so on.

**Task 6: Tunnel-Breaking Experiment**

Similar to previous task, I ran tun\_server.py on VPN server and tun\_client.py on host U. On host U, I used telnet 192.168.60.5 to telnet host V. Next, I stopped the TUN interface on VPN server, host U, or both. In this case, I chose to stop both programs. The result in Figure 22 shows that there is nothing show up when I tried to type in host V that telnet by host U. After, both programs were up again, things that I just typed earlier showed up and telnet was back to normal (Figure 23).

**Figure 22**

*The result when telnet was halt*



Nothing is showing up

host telnet host V

host U

terminal 2

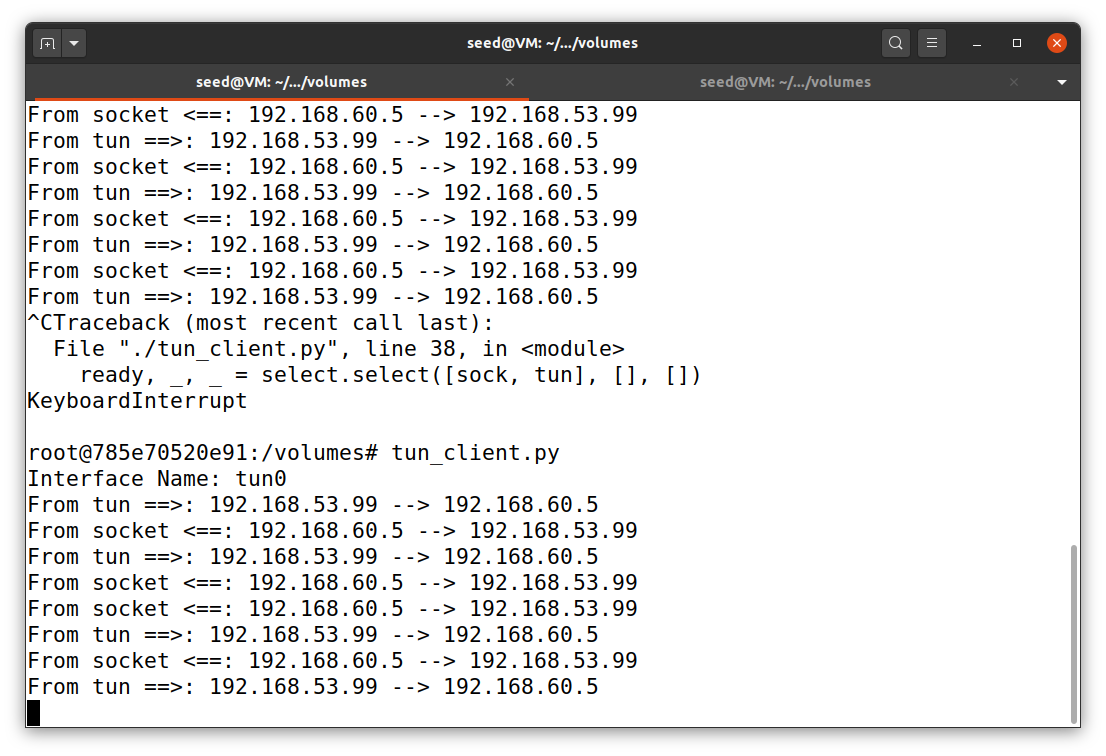
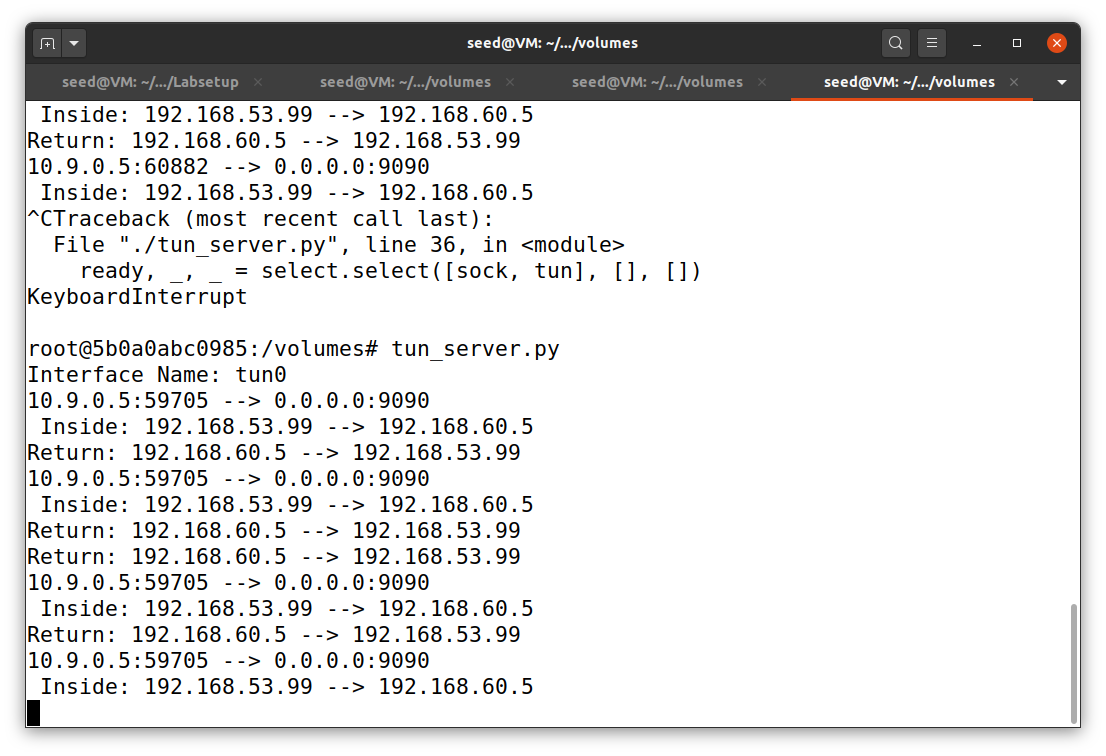
host U

terminal 1

VPN server

**Figure 23**

*The result when telnet resumes*



host U

terminal 1

host U

terminal 2

VPN server

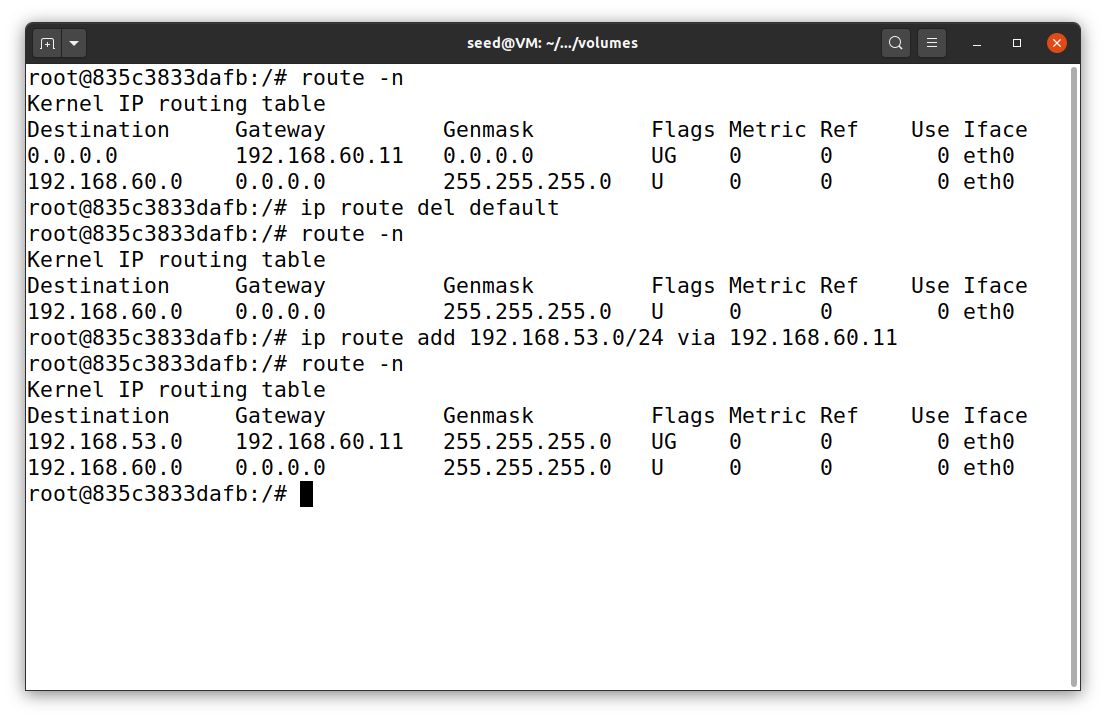
**Task 7: Routing Experiment on Host V**

In this task, I just change routing table of host V with the command below. The results are show below and it appears to be the same as the previous tasks.

* Delete default route
  + ip route del default
* Add TUN interface via VPN server
  + ip route add 192.168.53.0/24 via 192.168.60.11
* Display routing table
  + route -n

**Figure 24**

*Host V’s routing table*



Add TUN interface via VPN router

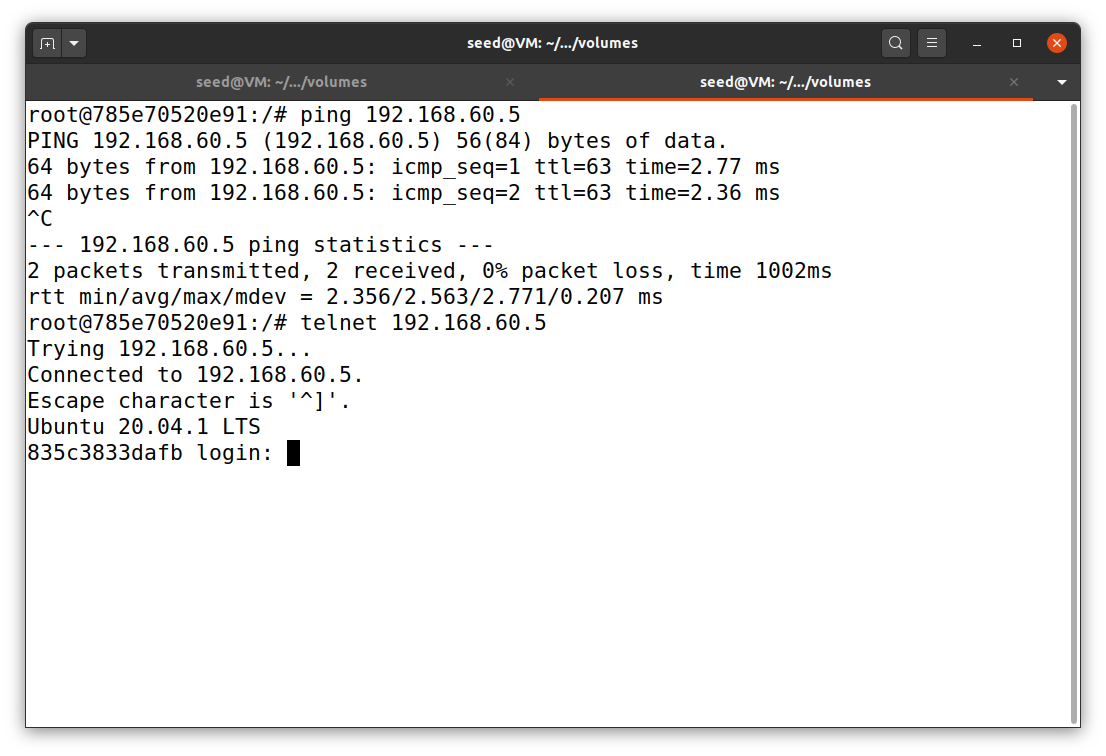
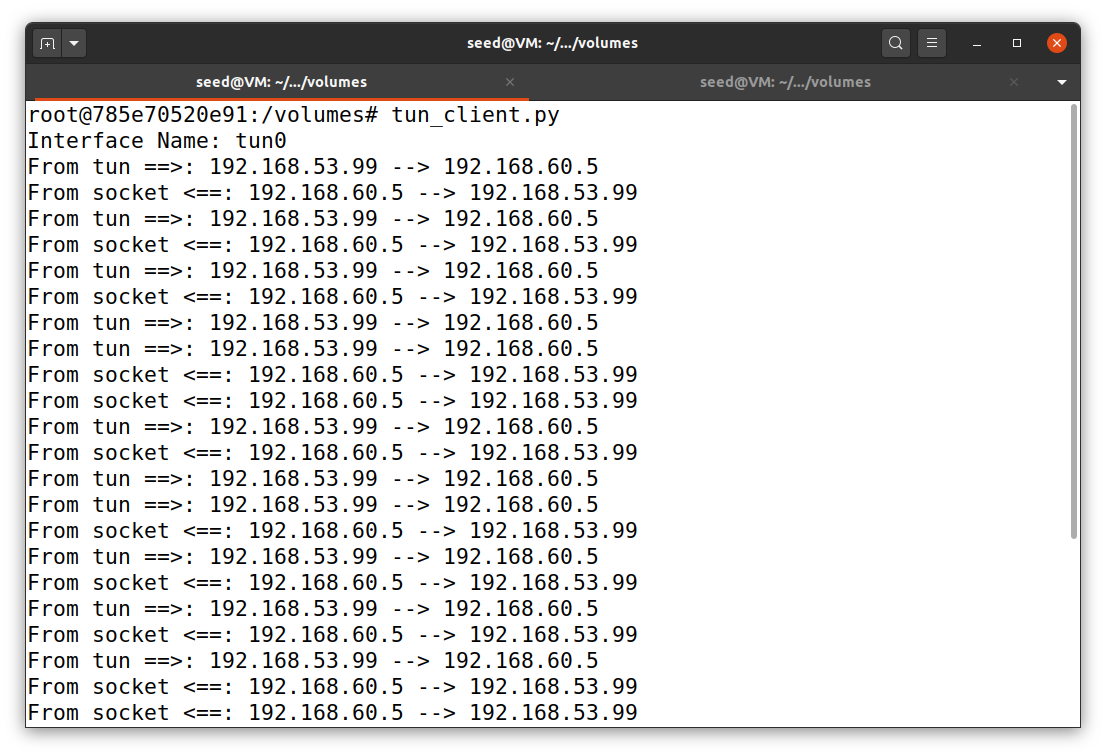
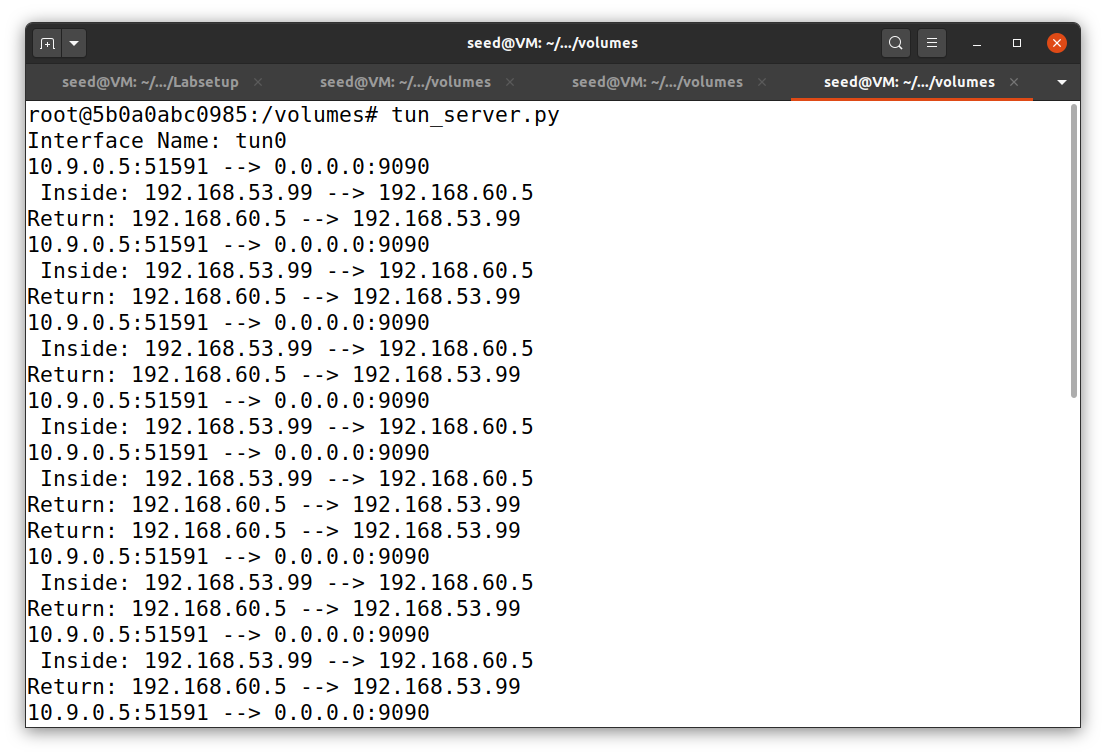
Delete the default route

Display the default settings

host V

**Figure 25**

*The result of task 7*



Host U can ping and telnet host V

host U

terminal 2

VPN server

host U

terminal 1

**References**

Du, W. (2019). Computer & internet security: A hands-on approach (2nd ed.). Independently published.

Du, W. (n.d.). VPN Tunneling Lab. SeedLabs 2.0. https://seedsecuritylabs.org/Labs\_20.04/‌Networking/VPN\_Tunnel/