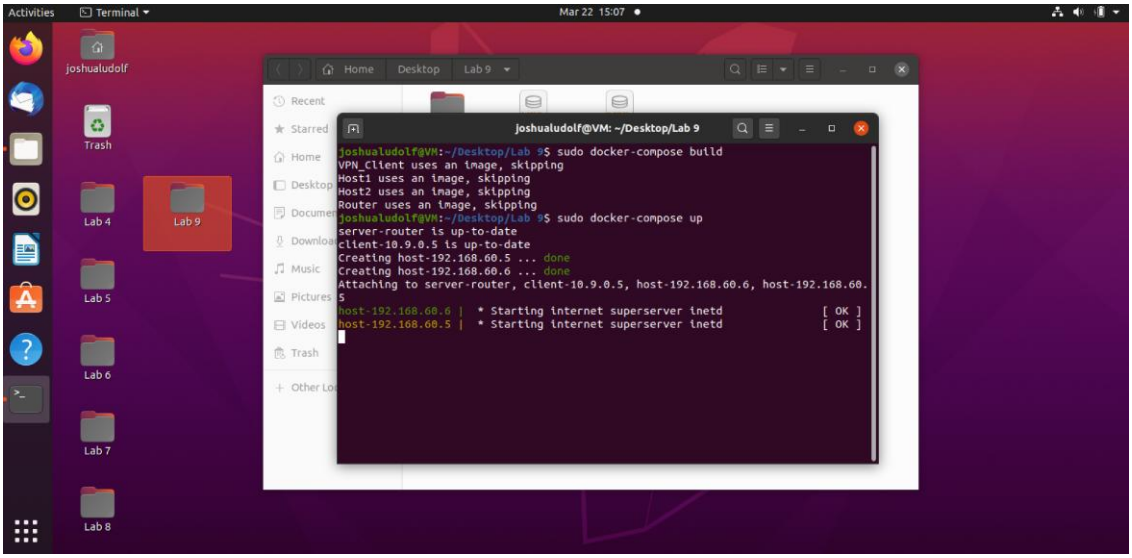
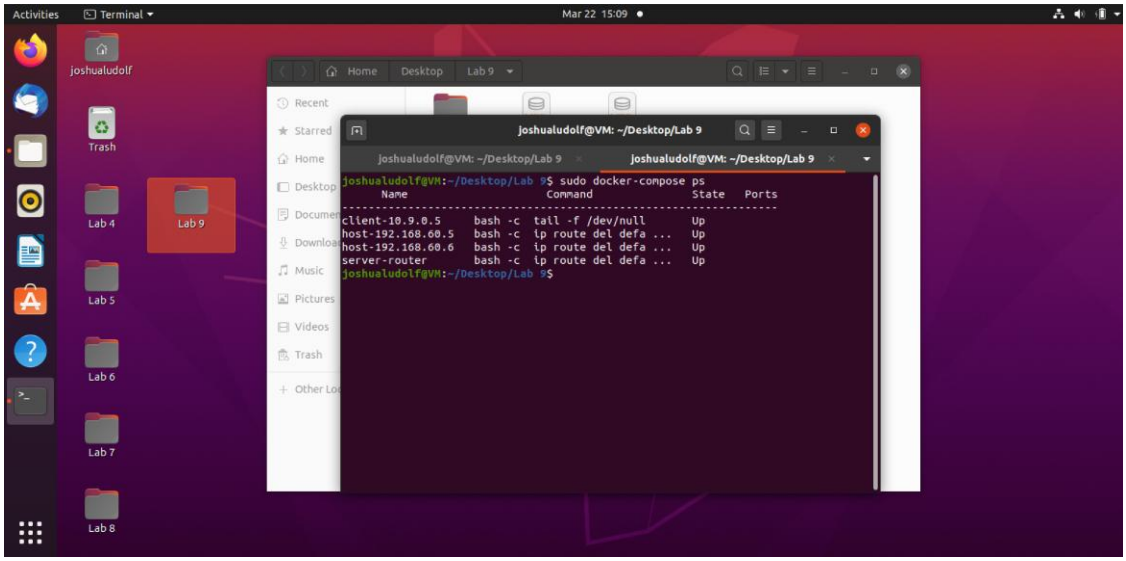
Lab 9: VPN Lab - The Container Version

Joshua Ludolf
CSCI 4321
Computer Security

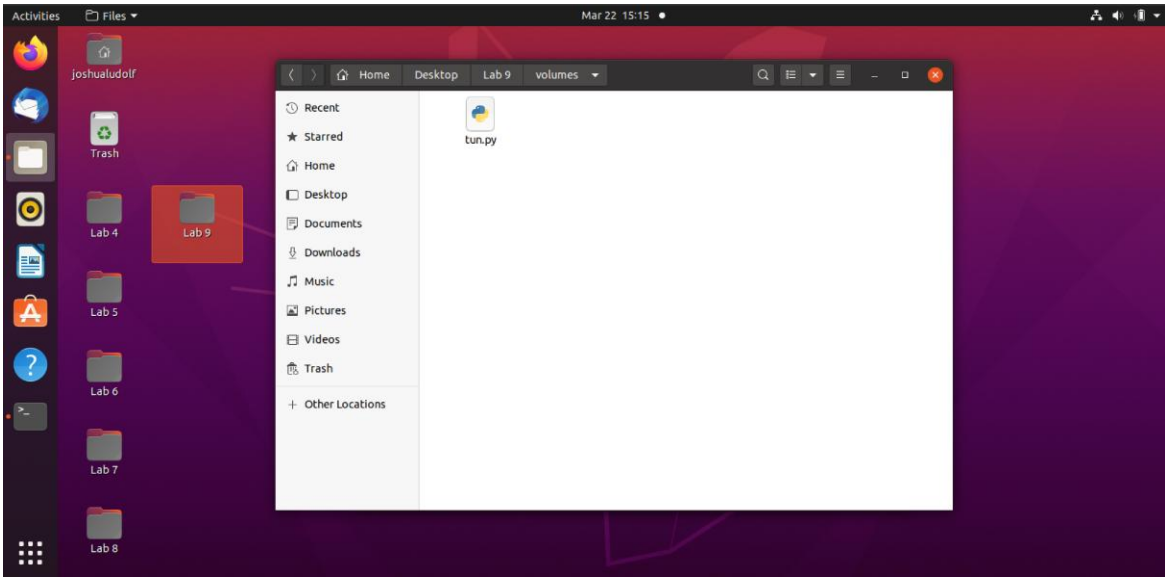
- **I additionally did task 6-8 for extra credit.**

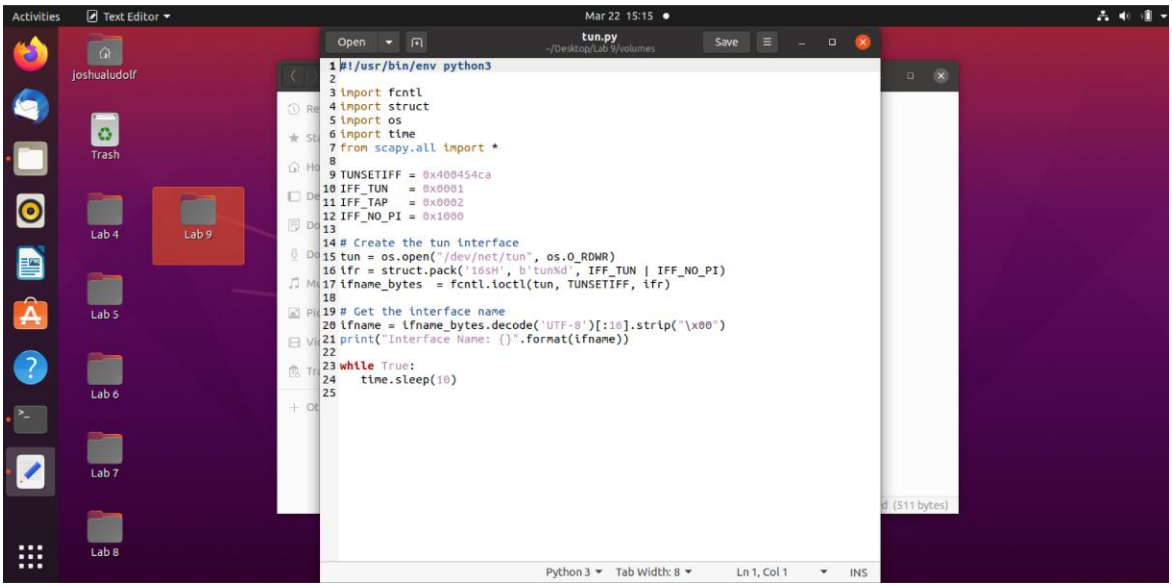
- **Firstly, I had to build the docker container and turn it on using following commands– sudo docker-compose build and sudo docker-compose up (Additionally had to execute sudo docker rm 192.168.60.5 and same for 192.168.60.6):**

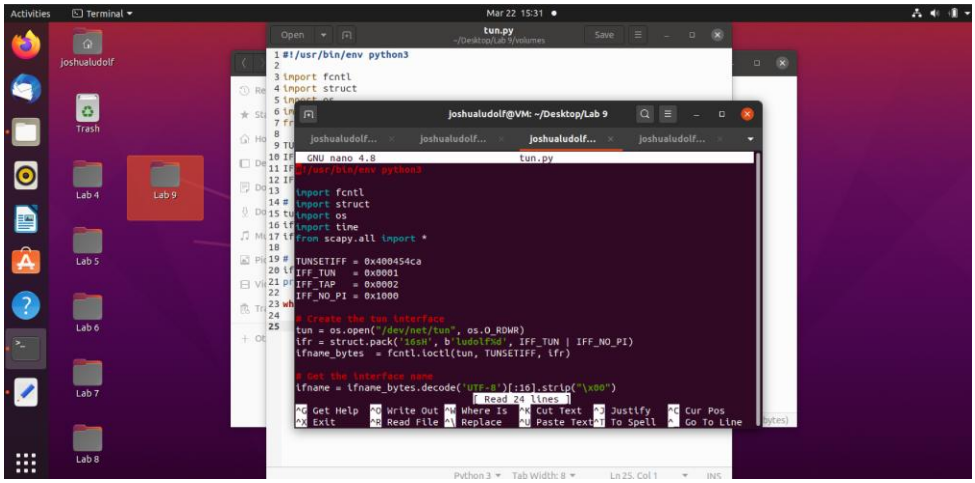


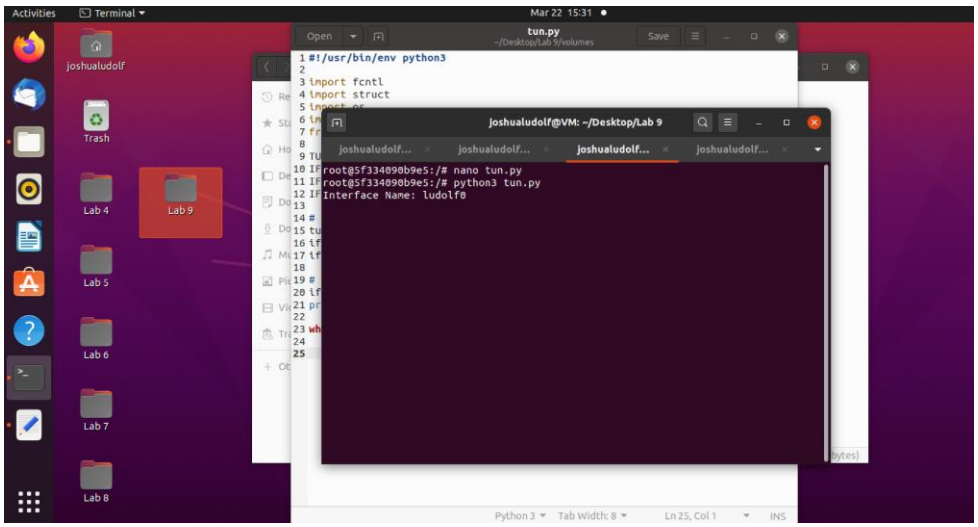


❖ **Next, I created and configured TUN Interface using the tun.py file in the volumes directory on Host U (which is the user side according to the topology):**

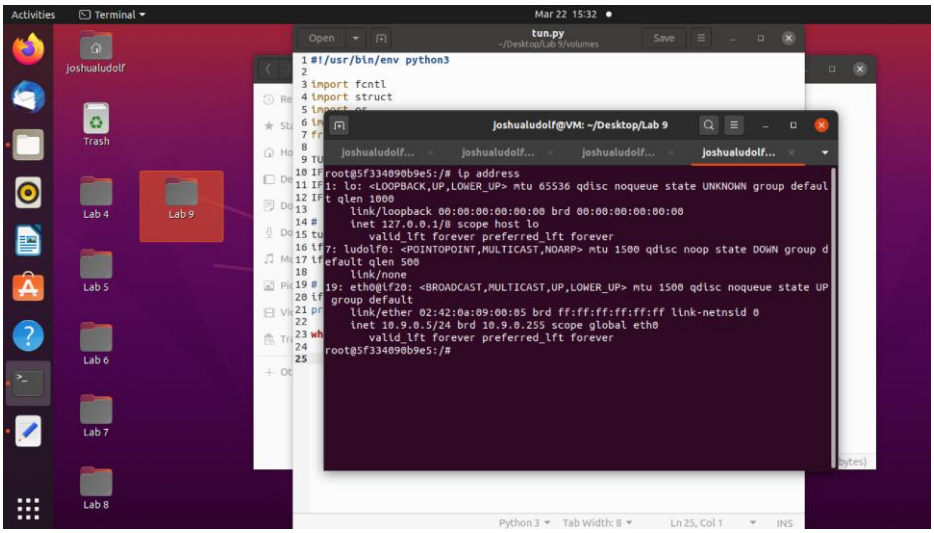




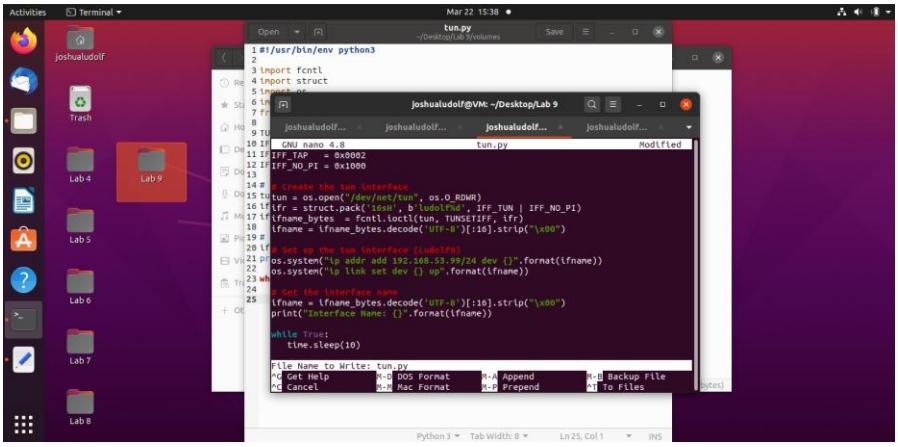


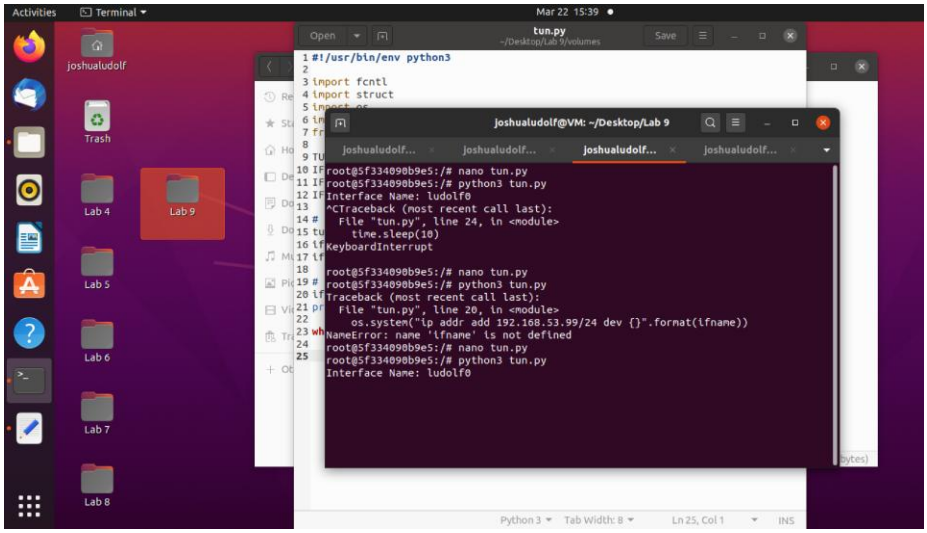


❖ **I observed that it would add an interface name tun0 to Host U (10.9.0.5):**

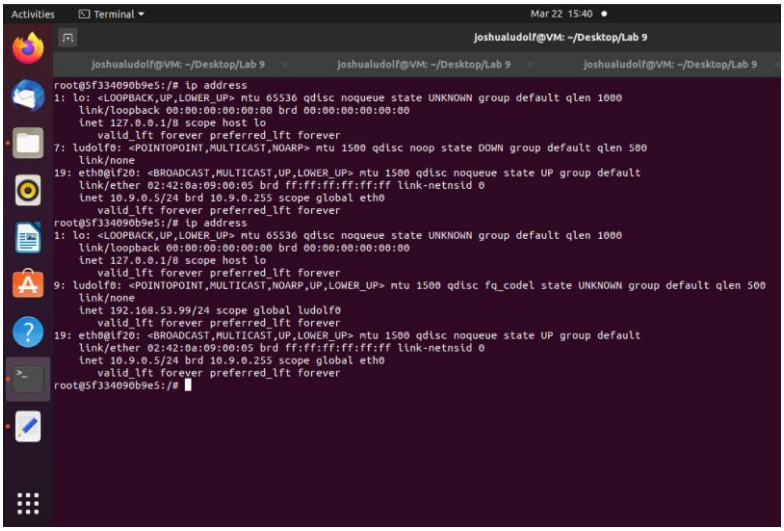


❖ **In the previous step, the TUN interface is not usuable as it wasn't configured, here I edited the tun.py to configure the interface:**

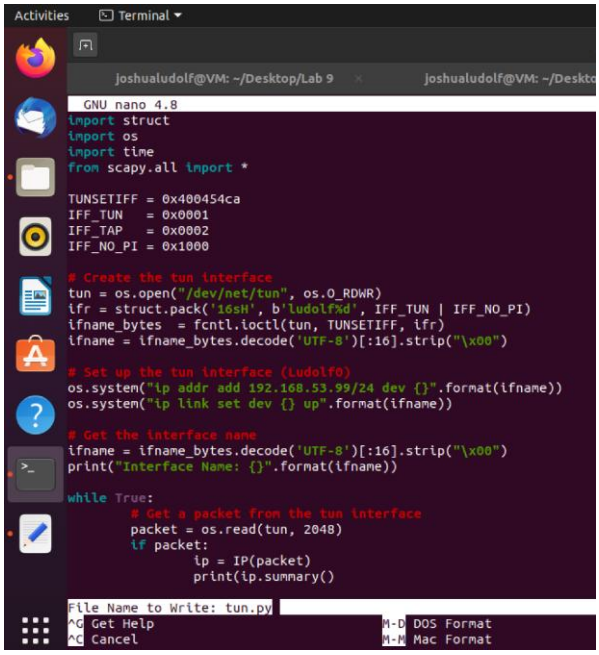


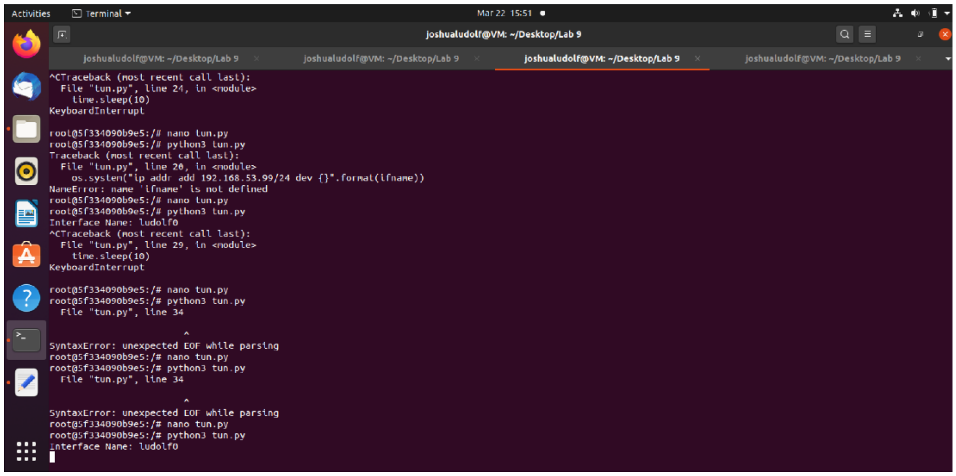


❖ **From that I observed that the ip address 192.168.53.99/24 was attached to the ludolf0 interface:**

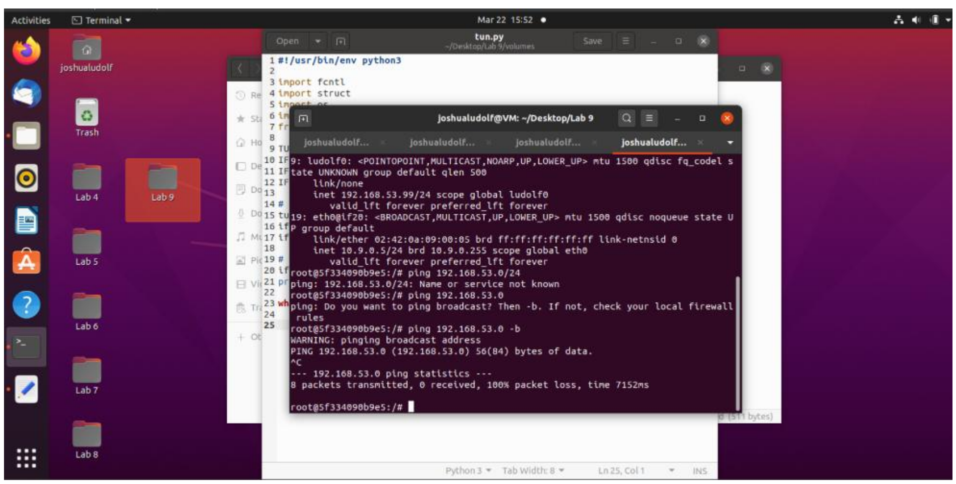


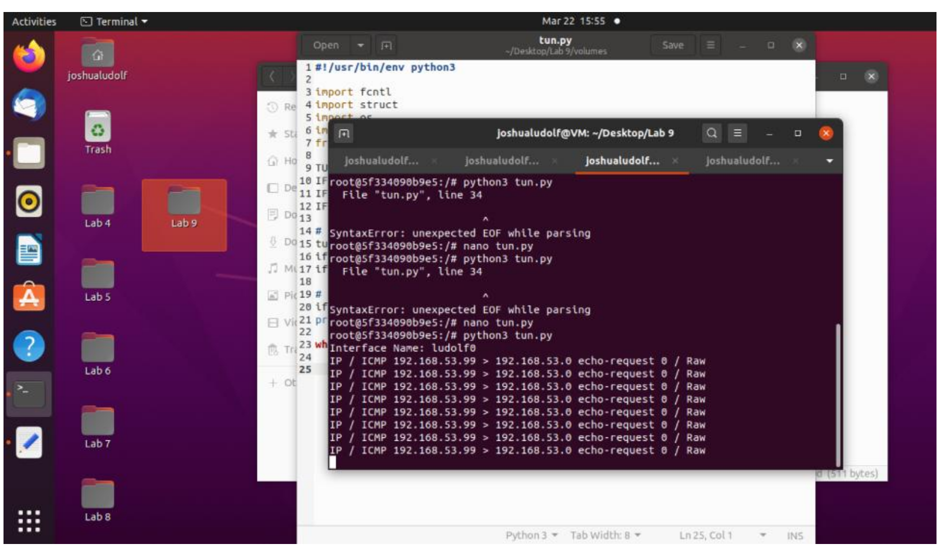
❖ **After that, I wanted to read from the TUN interface as I wanted to see the IP packet:**



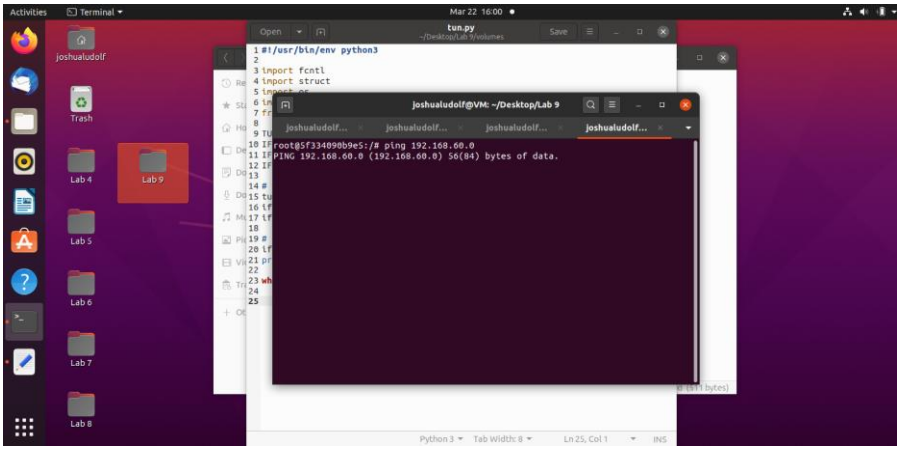


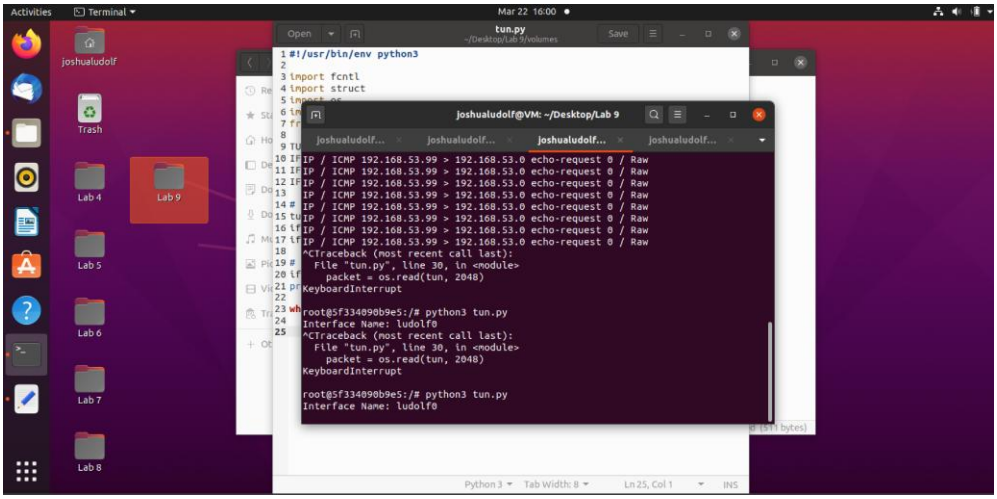
❖ **On Host U when you ping 192.168.53.0/24 it displays an echo request as it is trying to break through the firewall (being successful), of course this is what I saw with print statements from tun.py.:**



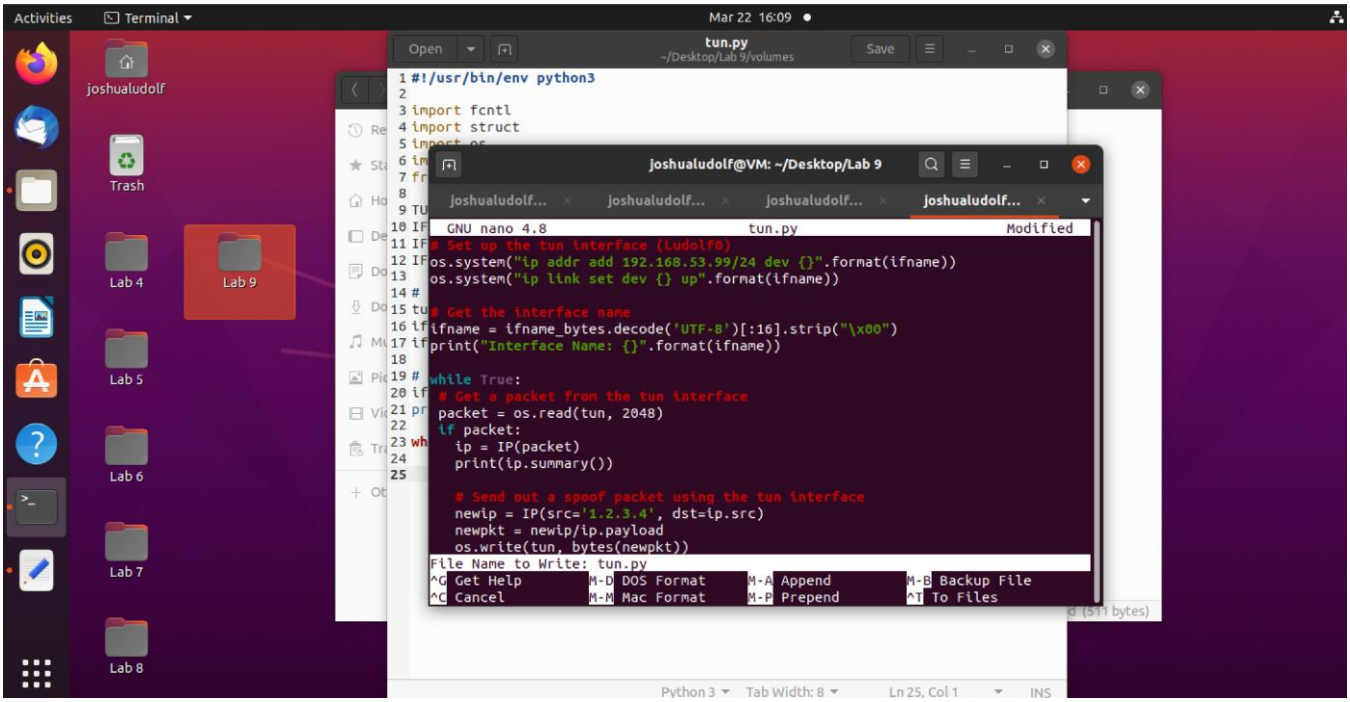


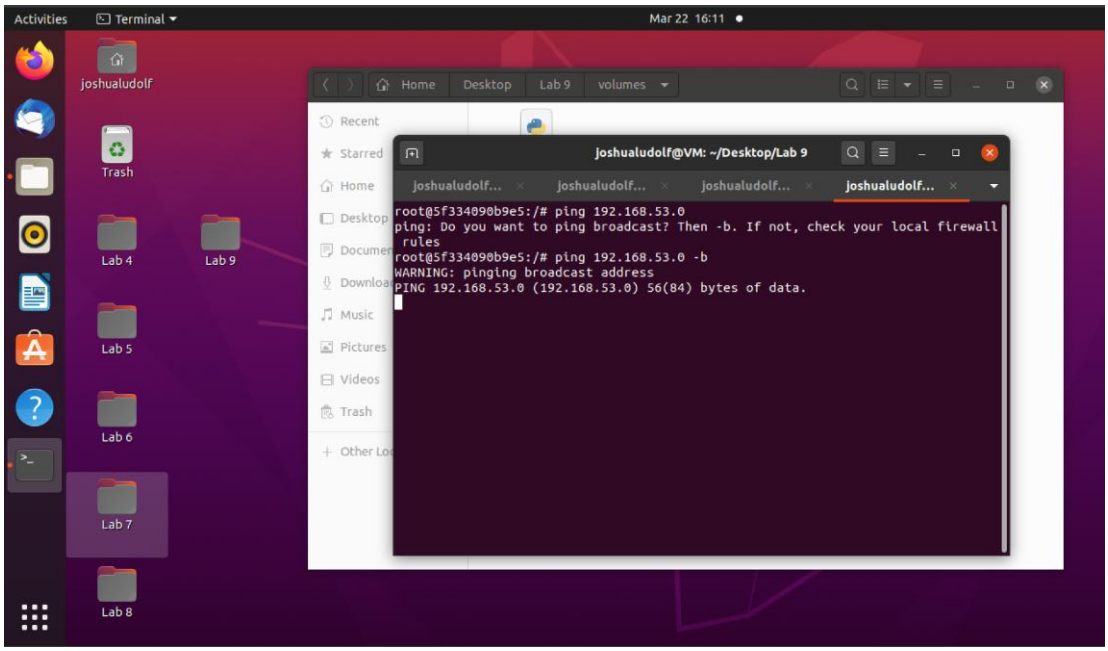
- **However, for pinging 192.168.60.0/24, the network doesn't print out anything as it is unsuccessful since we essentially routed ludolf0 interface outside of the internal network:**

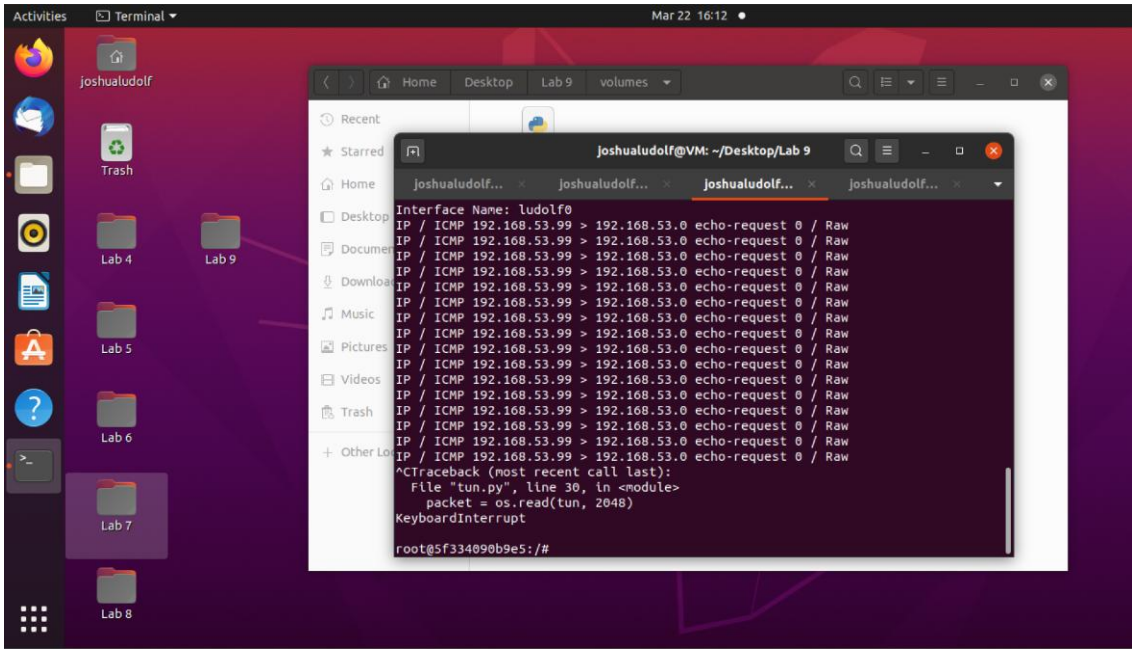




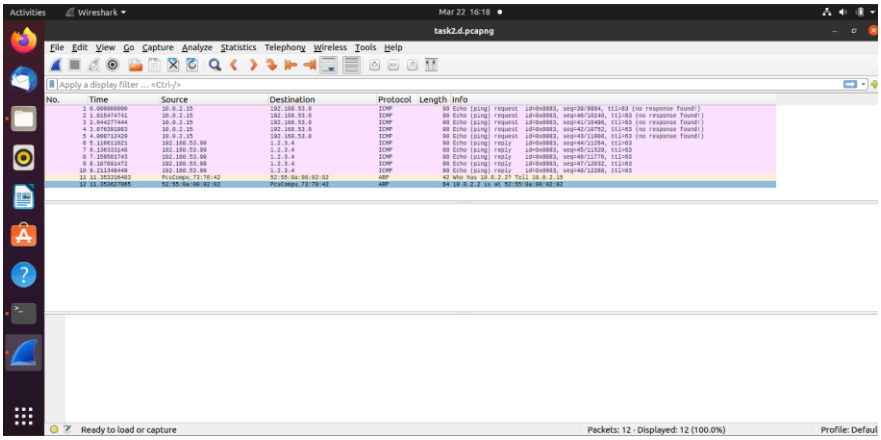
- **Now it was time to write to the TUN interface, additionally we note that whatever is written on the interface by the application will appear in the kernel as an IP packet. I modified the tun.py program, so after getting a packet from the TUN interface, we construct a new packet based on the received packet. I then write the new packet to the TUN interface.**


Activities
Terminal
Mar 22 16:09
joshualudolf
Trash
Lab 4
Lab 9
Lab 5
Lab 6
Lab 7
Lab 8
Open
tun.py
~/Desktop/Lab 9/volumes
Save
joshualudolf@VM: ~/Desktop/Lab 9
GNU nano 4.8
tun.py
Modified
File Name to Write: tun.py
Python 3
Tab Width: 8
Ln 25, Col 1
INS

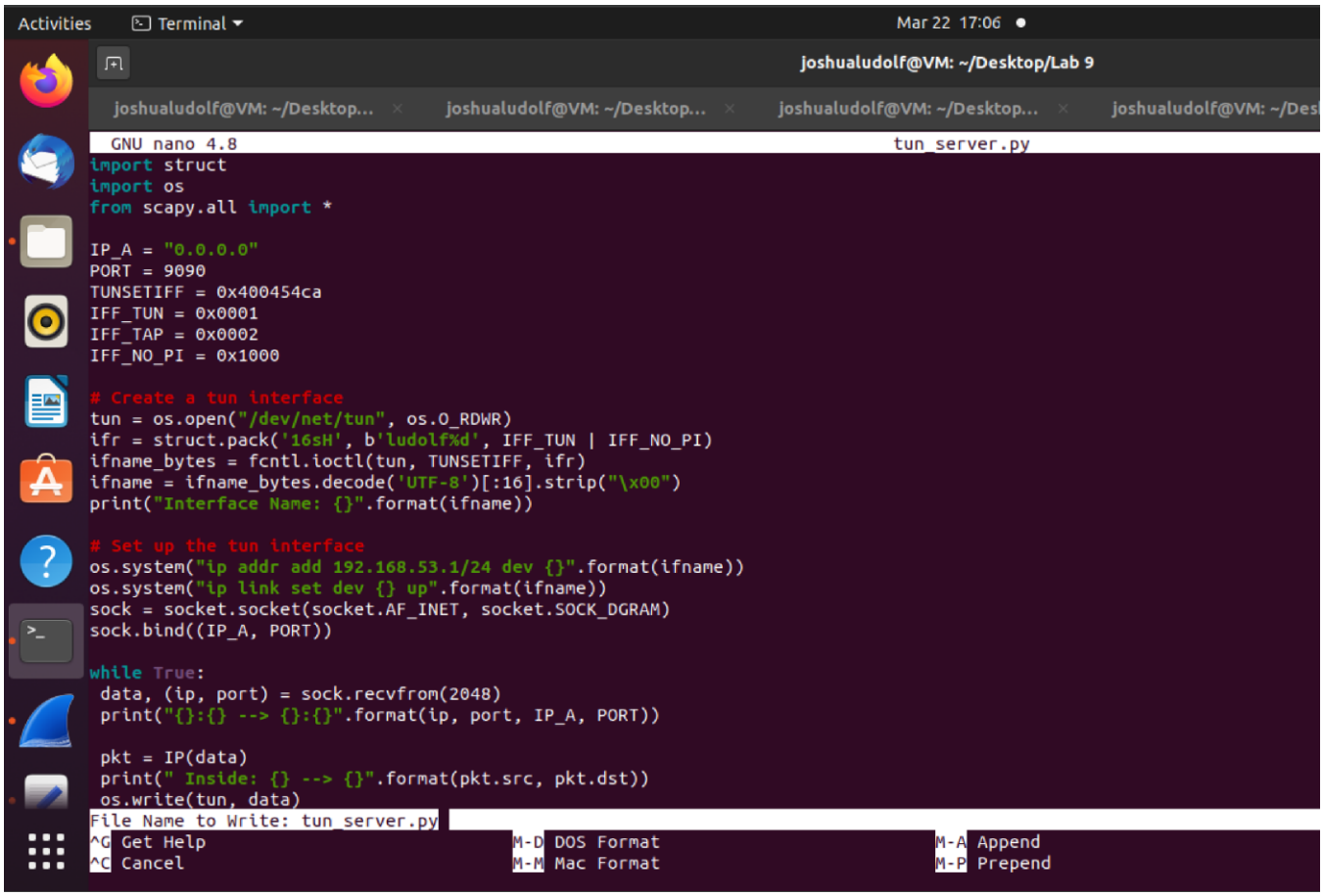

Activities
Terminal
Mar 22 16:11
joshualudolf
Trash
Lab 4
Lab 9
Lab 5
Lab 6
Lab 7
Lab 8
Home
Desktop
Lab 9
volumes
Recent
Starred
Home
Desktop
Music
Pictures
Videos
Trash
joshualudolf@VM: ~/Desktop/Lab 9
root@5f334090b9e5:/# ping 192.168.53.0
ping: Do you want to ping broadcast? Then -b. If not, check your local firewall rules
root@5f334090b9e5:/# ping 192.168.53.0 -b
WARNING: pinging broadcast address
PING 192.168.53.0 (192.168.53.0) 56(84) bytes of data.

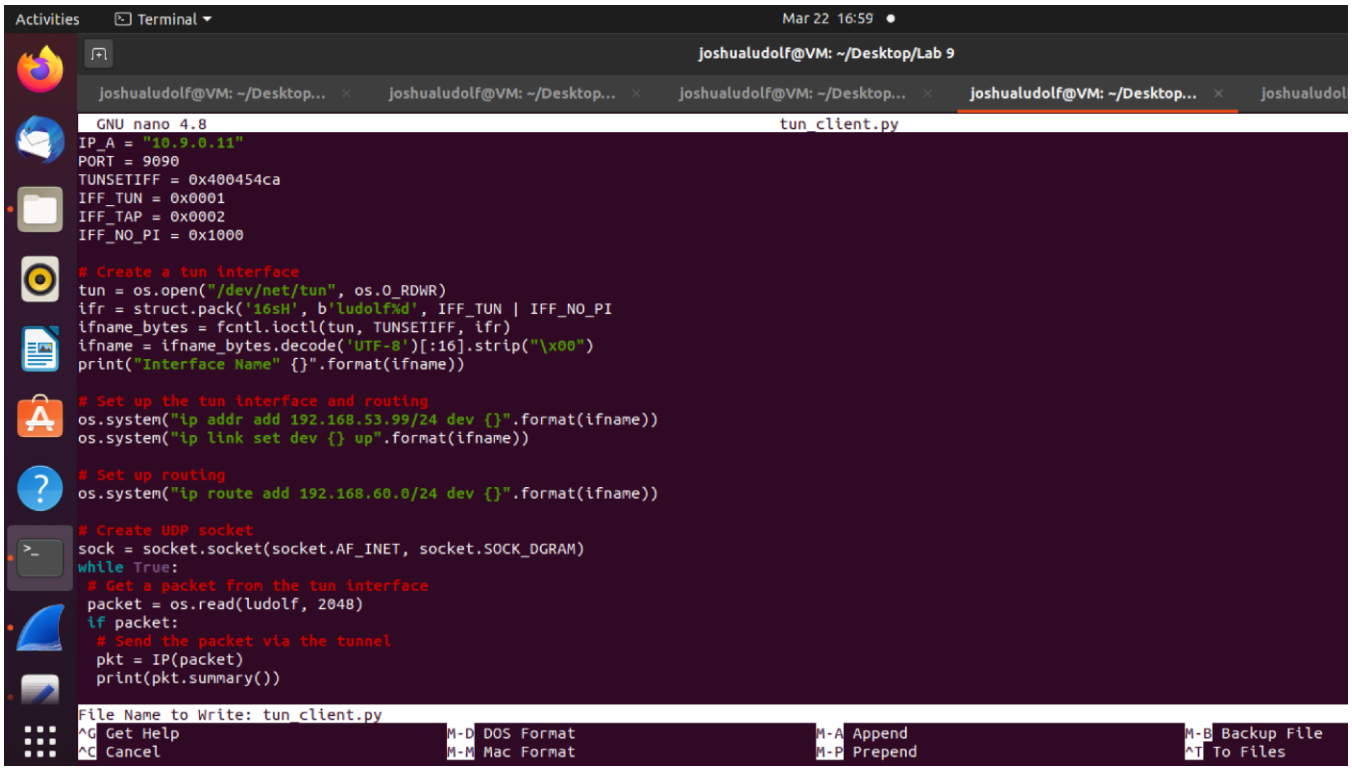


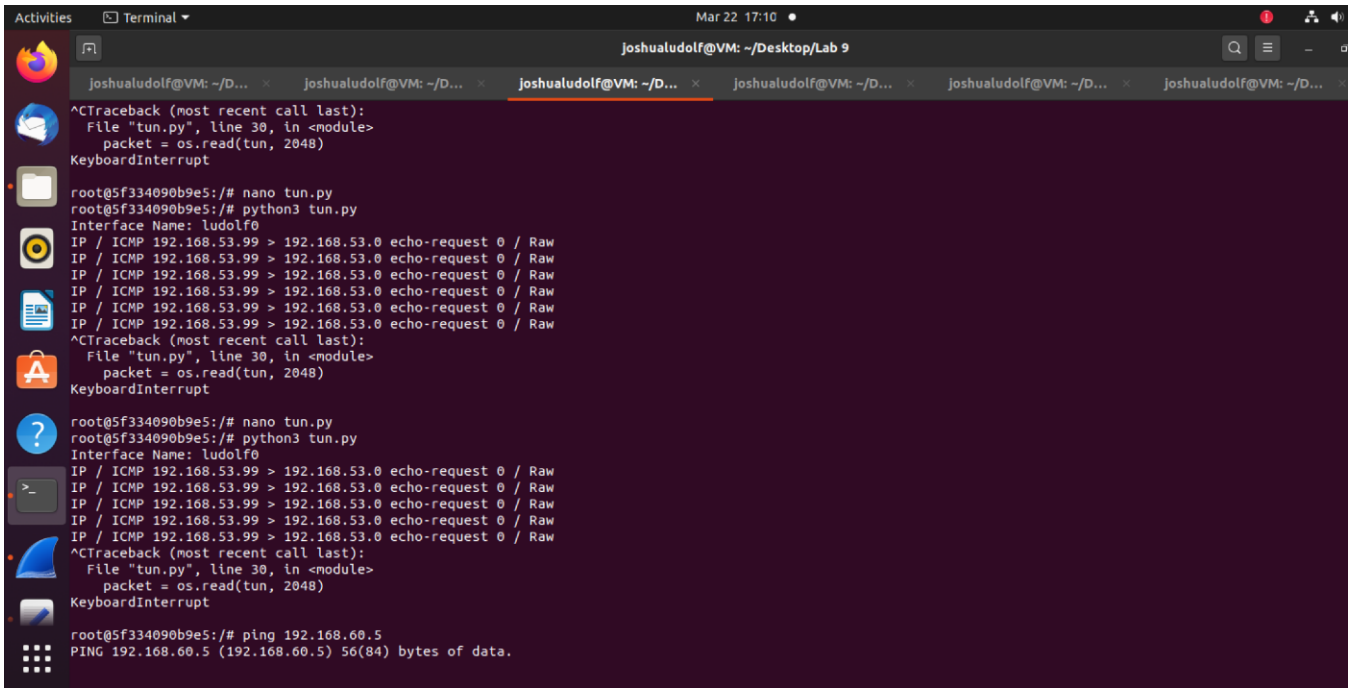
- **I observed in wireshark that if we ping before running the code frames 1 – 5, I get a request ping with no response (this is before executing tun.py). In frames 6 – 10 we get replies once I executed tun.py:**

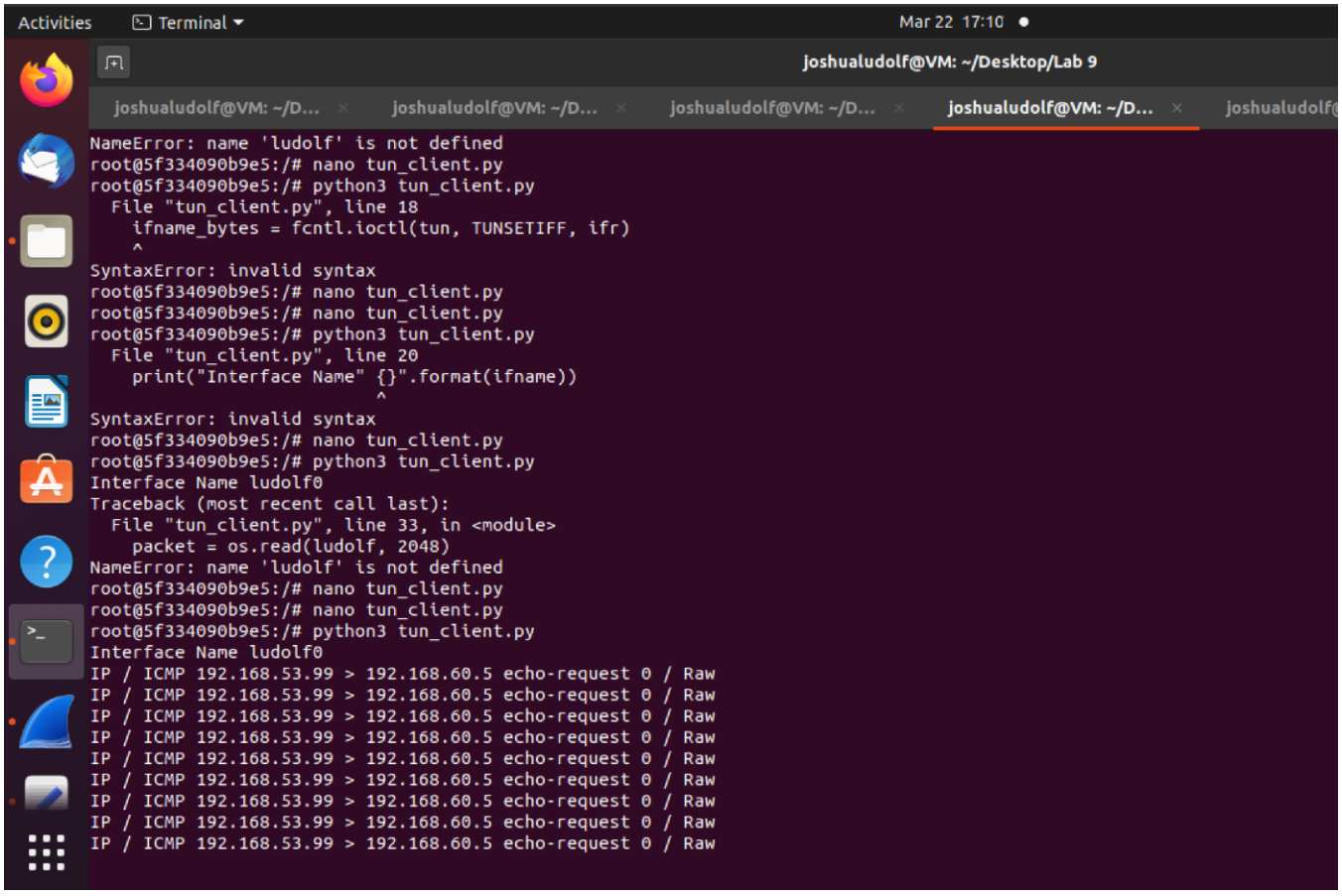


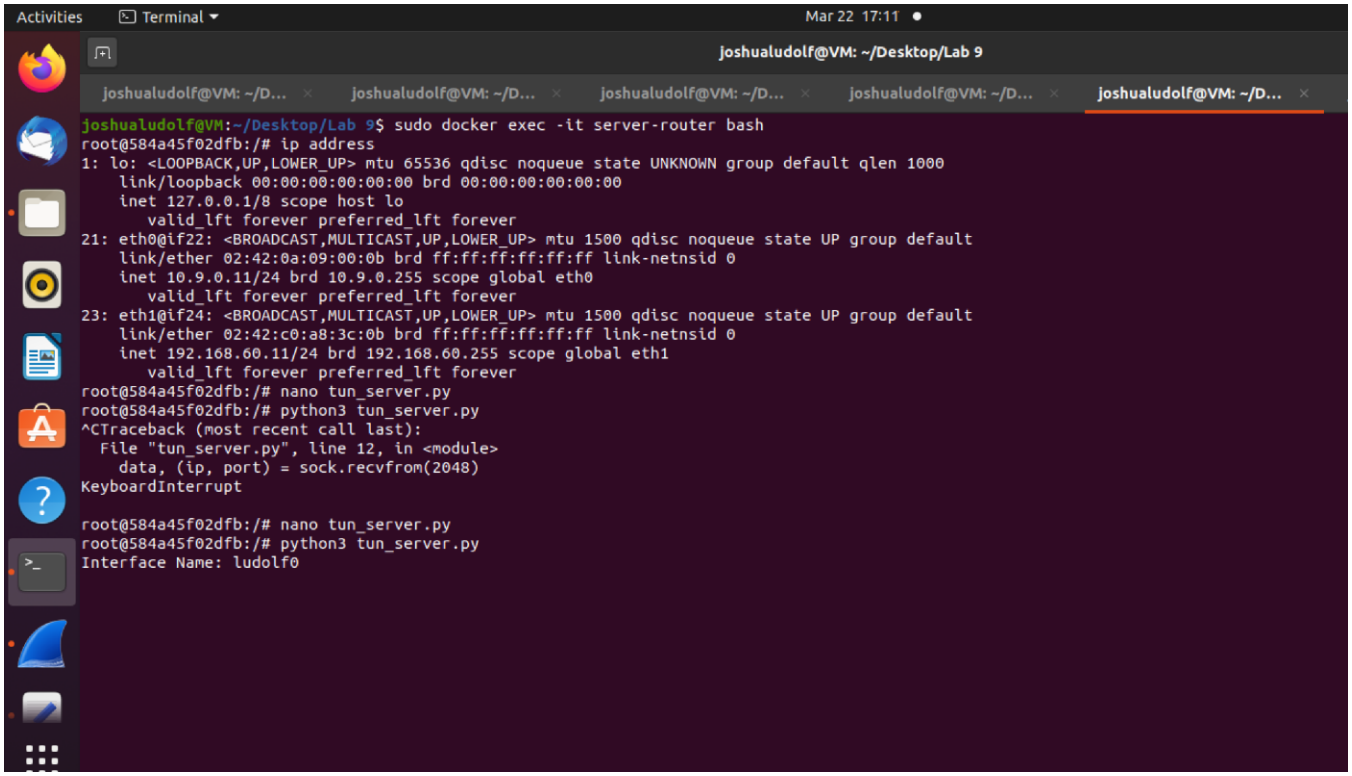
- **Now it was time to try my first attempt of sending IP packet to VPN Server Through a Tunnel, on the VPN server I create tun_server.py and tun_client.py in Host U:**


Activities
Terminal
Mar 22 17:06
joshualudolf@VM: ~/Desktop/Lab 9
joshualudolf@VM: ~/Desktop...
joshualudolf@VM: ~/Desktop...
joshualudolf@VM: ~/Desktop...
joshualudolf@VM: ~/Desktop...
GNU nano 4.8
tun_server.py
import struct
import os
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 a tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'ludolf%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface Name: {}".format(ifname))
# Set up the tun interface
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)
File Name to Write: tun_server.py
^G Get Help
M-D DOS Format
M-A Append
^C Cancel
M-M Mac Format
M-P Prepend

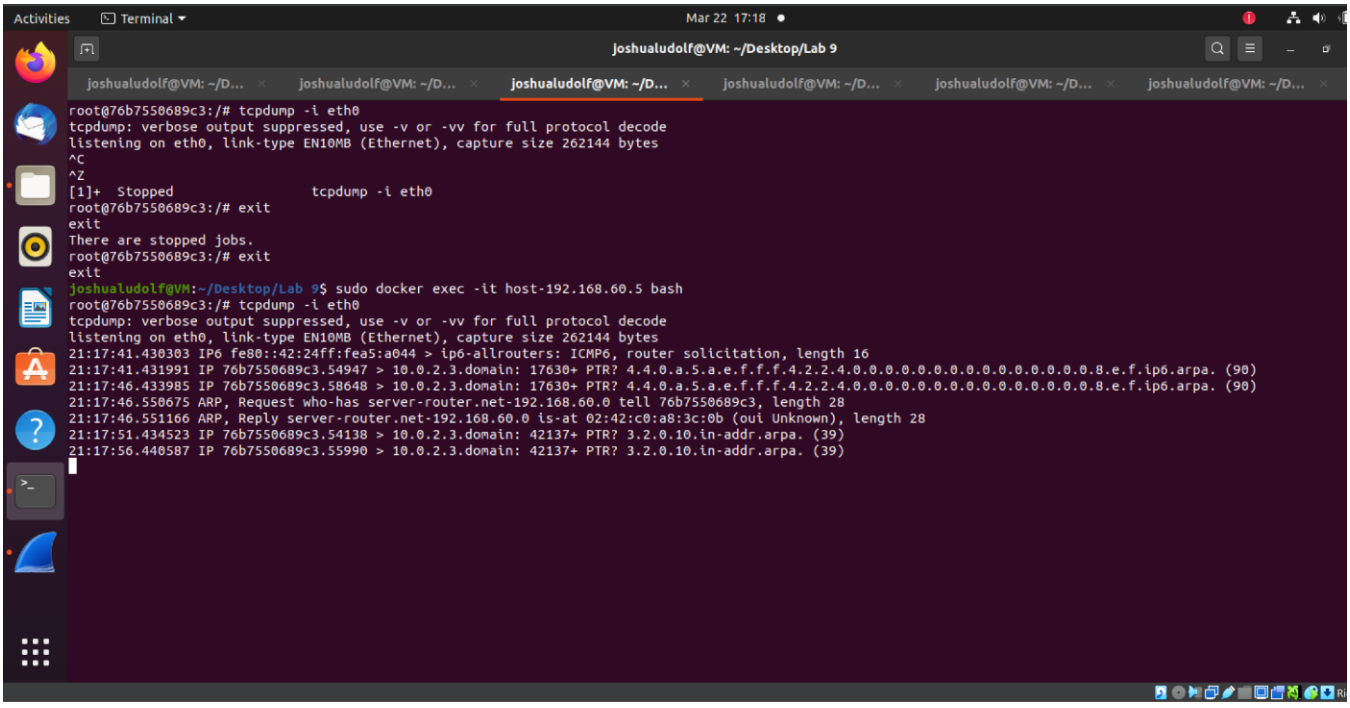

Activities
Terminal
Mar 22 16:59
joshualudolf@VM: ~/Desktop/Lab 9
joshualudolf@VM: ~/Desktop...
joshualudolf@VM: ~/Desktop...
joshualudolf@VM: ~/Desktop...
joshualudolf@VM: ~/Desktop...
GNU nano 4.8
tun_client.py
IP_A = "10.9.0.11"
PORT = 9090
TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# Create a tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'ludolf%d', IFF_TUN | IFF_NO_PI
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface Name" {}".format(ifname))
# Set up the tun interface and routing
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:
# Get a packet from the tun interface
packet = os.read(ludolf, 2048)
if packet:
# Send the packet via the tunnel
pkt = IP(packet)
print(pkt.summary())
File Name to Write: tun_client.py
^G Get Help
M-D DOS Format
M-A Append
M-B Backup File
^C Cancel
M-M Mac Format
M-P Prepend
^T To Files


Activities
Terminal
Mar 22 17:10
joshualudolf@VM: ~/Desktop/Lab 9
joshualudolf@VM: ~/D...
joshualudolf@VM: ~/D...
joshualudolf@VM: ~/D...
joshualudolf@VM: ~/D...
joshualudolf@VM: ~/D...
joshualudolf@VM: ~/D...
^CTraceback (most recent call last):
File "tun.py", line 30, in <module>
packet = os.read(tun, 2048)
KeyboardInterrupt
root@5f334090b9e5:/# nano tun.py
root@5f334090b9e5:/# python3 tun.py
Interface Name: ludolf0
IP / ICMP 192.168.53.99 > 192.168.53.0 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.0 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.0 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.0 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.0 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.0 echo-request 0 / Raw
^CTraceback (most recent call last):
File "tun.py", line 30, in <module>
packet = os.read(tun, 2048)
KeyboardInterrupt
root@5f334090b9e5:/# nano tun.py
root@5f334090b9e5:/# python3 tun.py
Interface Name: ludolf0
IP / ICMP 192.168.53.99 > 192.168.53.0 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.0 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.0 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.0 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.0 echo-request 0 / Raw
^CTraceback (most recent call last):
File "tun.py", line 30, in <module>
packet = os.read(tun, 2048)
KeyboardInterrupt
root@5f334090b9e5:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.


Activities
Terminal
Mar 22 17:10
joshualudolf@VM: ~/Desktop/Lab 9
joshualudolf@VM: ~/D...
joshualudolf@VM: ~/D...
joshualudolf@VM: ~/D...
joshualudolf@VM: ~/D...
NameError: name 'ludolf' is not defined
root@5f334090b9e5:/# nano tun_client.py
root@5f334090b9e5:/# python3 tun_client.py
File "tun_client.py", line 18
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
^
SyntaxError: invalid syntax
root@5f334090b9e5:/# nano tun_client.py
root@5f334090b9e5:/# nano tun_client.py
root@5f334090b9e5:/# python3 tun_client.py
File "tun_client.py", line 20
print("Interface Name" {}".format(ifname))
^
SyntaxError: invalid syntax
root@5f334090b9e5:/# nano tun_client.py
root@5f334090b9e5:/# python3 tun_client.py
Interface Name ludolf0
Traceback (most recent call last):
File "tun_client.py", line 33, in <module>
packet = os.read(ludolf, 2048)
NameError: name 'ludolf' is not defined
root@5f334090b9e5:/# nano tun_client.py
root@5f334090b9e5:/# nano tun_client.py
root@5f334090b9e5:/# python3 tun_client.py
Interface Name ludolf0
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw

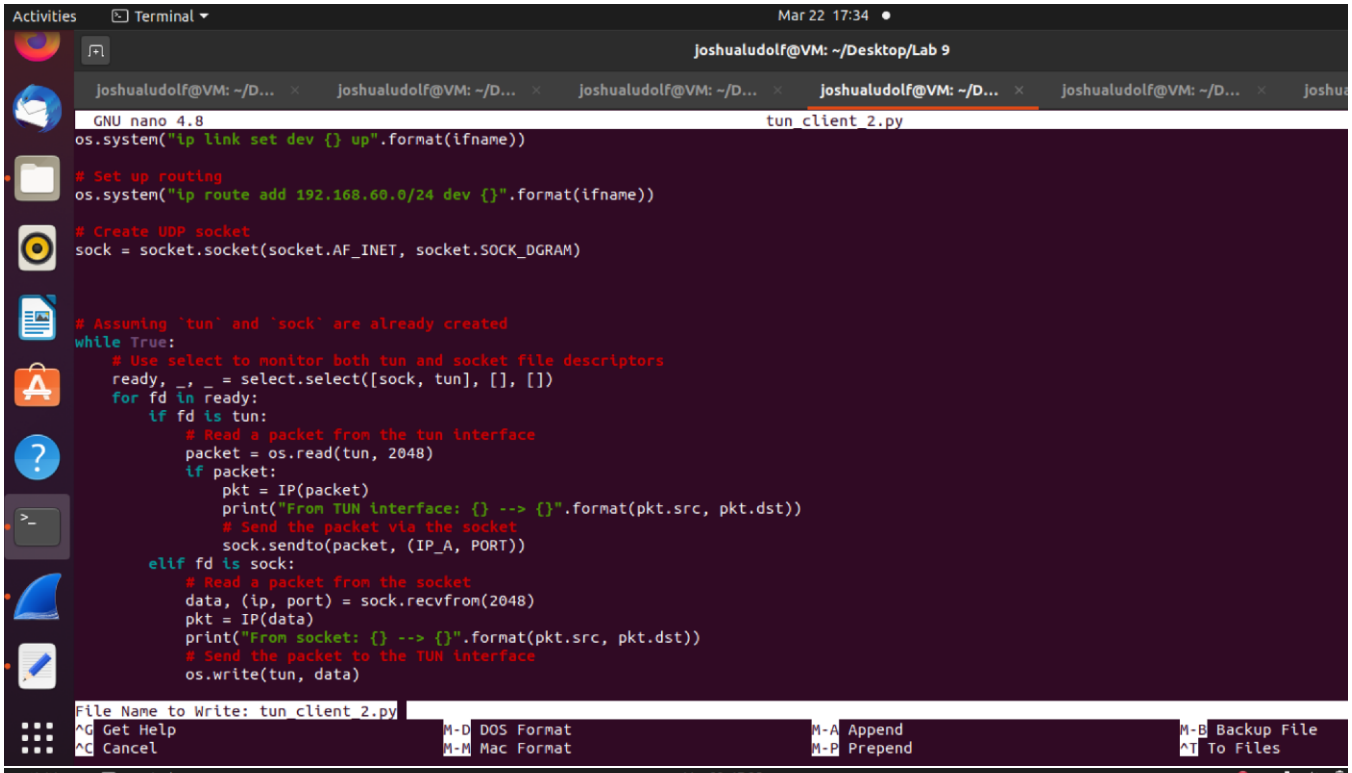


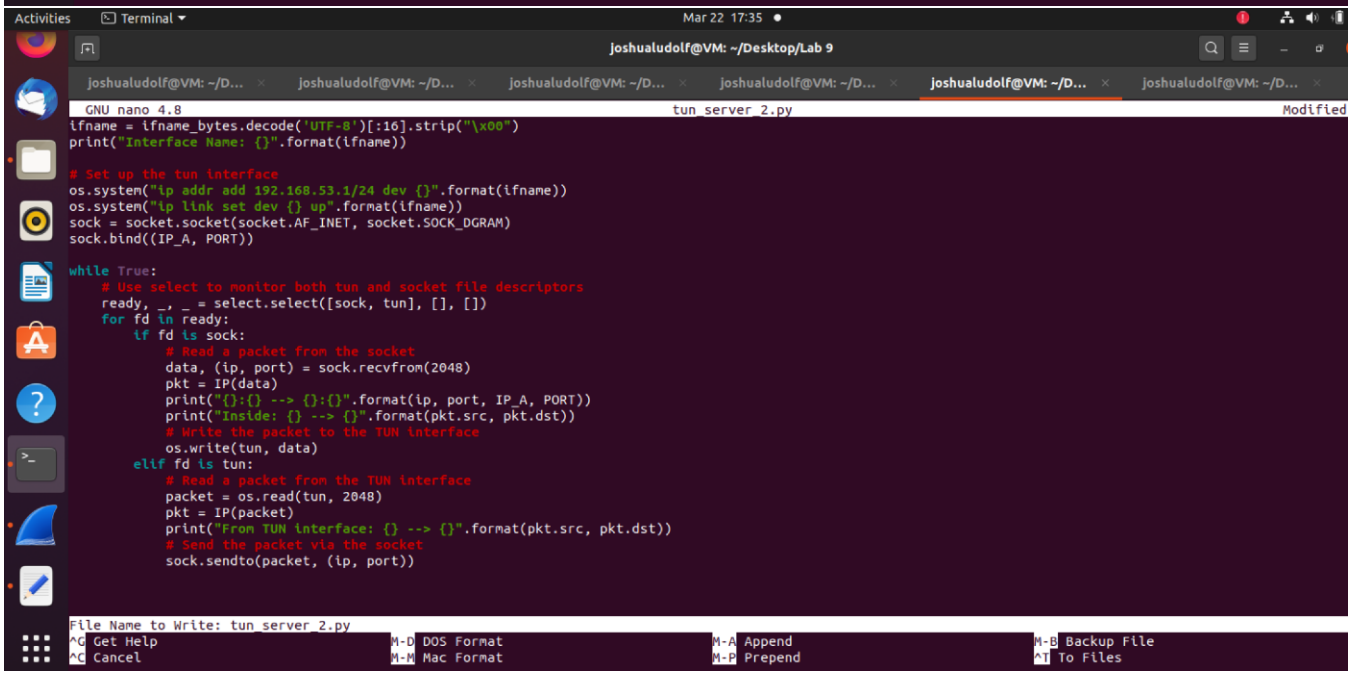
- **And then, I ping 192.168.60.5 but unfortunately didn't get same results as shown in professors exapme:**

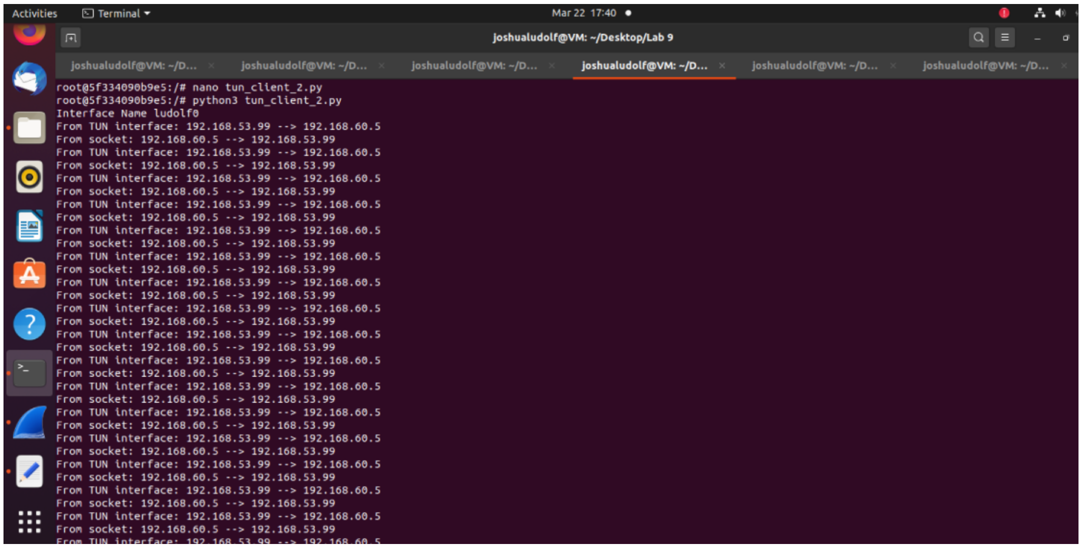


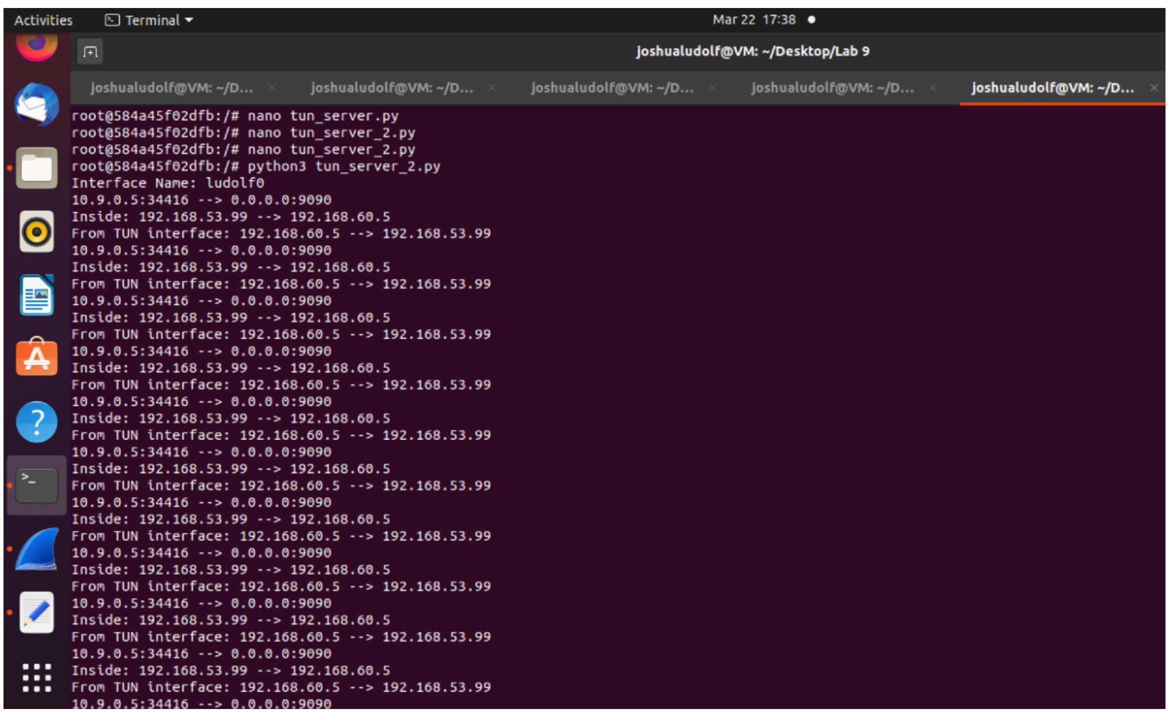
- **After getting to this point, one of my tunnel is complete, that is I can send packets from Host U to Host V via tunnel. The packets get dropped somewhere as there is no response, I**

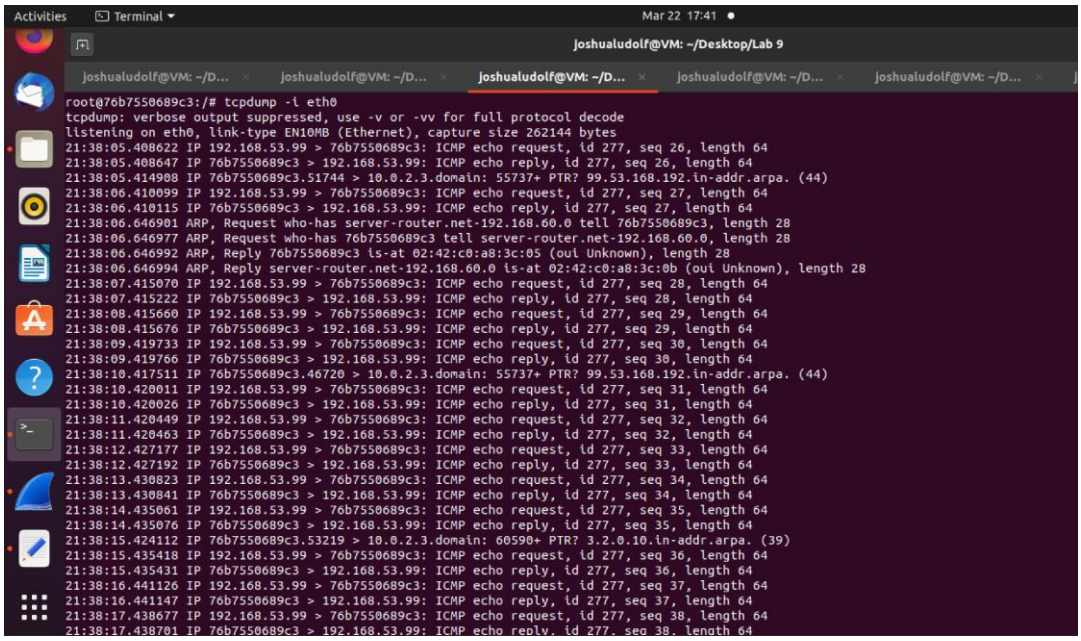
set up the direction Host V to Host U (took a bit, but finally after couple minutes it worked 😍):



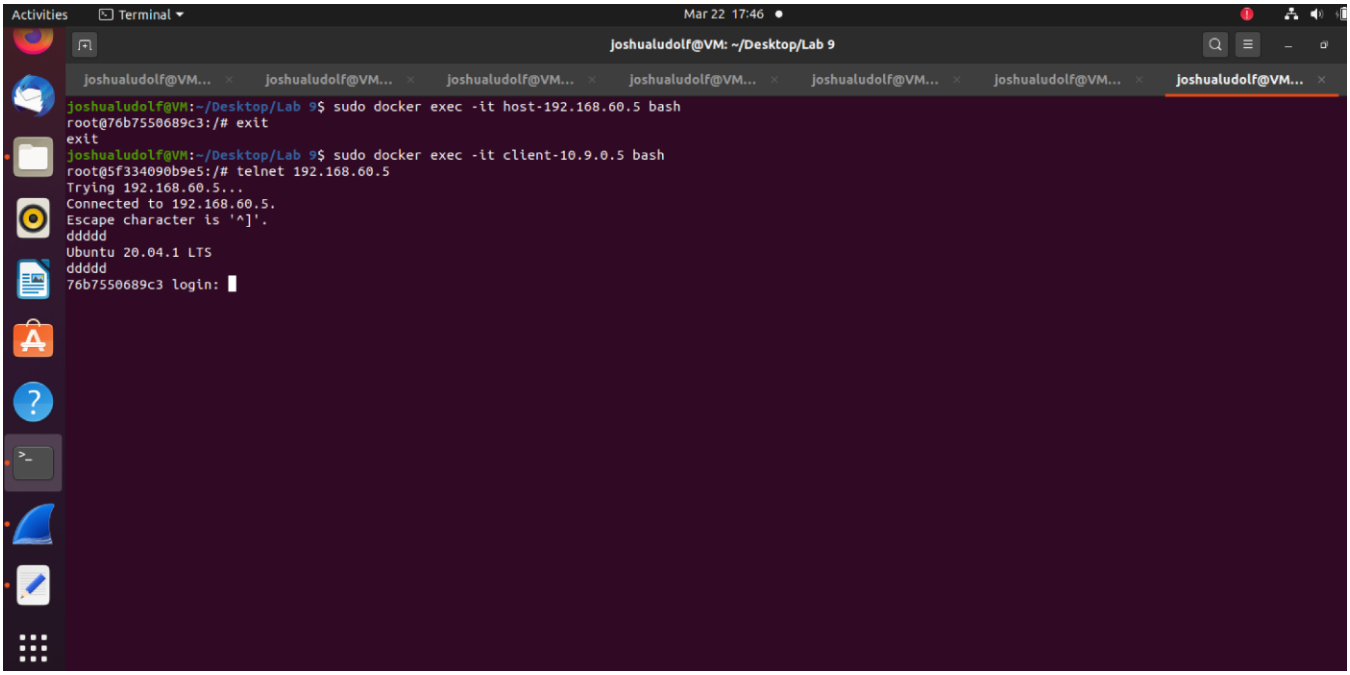



Activities
Terminal
Mar 22 17:40
joshualudolf@VM: ~/Desktop/Lab 9
root@5f334090b9e5:/# nano tun_client_2.py
root@5f334090b9e5:/# python3 tun_client_2.py
Interface Name ludolf0
From TUN interface: 192.168.53.99 --> 192.168.60.5
From socket: 192.168.60.5 --> 192.168.53.99
From TUN interface: 192.168.53.99 --> 192.168.60.5
From socket: 192.168.60.5 --> 192.168.53.99
From TUN interface: 192.168.53.99 --> 192.168.60.5
From socket: 192.168.60.5 --> 192.168.53.99
From TUN interface: 192.168.53.99 --> 192.168.60.5
From socket: 192.168.60.5 --> 192.168.53.99
From TUN interface: 192.168.53.99 --> 192.168.60.5
From socket: 192.168.60.5 --> 192.168.53.99
From TUN interface: 192.168.53.99 --> 192.168.60.5
From socket: 192.168.60.5 --> 192.168.53.99
From TUN interface: 192.168.53.99 --> 192.168.60.5
From socket: 192.168.60.5 --> 192.168.53.99
From TUN interface: 192.168.53.99 --> 192.168.60.5
From socket: 192.168.60.5 --> 192.168.53.99
From TUN interface: 192.168.53.99 --> 192.168.60.5
From socket: 192.168.60.5 --> 192.168.53.99
From TUN interface: 192.168.53.99 --> 192.168.60.5
From socket: 192.168.60.5 --> 192.168.53.99
From TUN interface: 192.168.53.99 --> 192.168.60.5
From socket: 192.168.60.5 --> 192.168.53.99
From TUN interface: 192.168.53.99 --> 192.168.60.5
From socket: 192.168.60.5 --> 192.168.53.99
From TUN interface: 192.168.53.99 --> 192.168.60.5
From socket: 192.168.60.5 --> 192.168.53.99
From TUN interface: 192.168.53.99 --> 192.168.60.5
From socket: 192.168.60.5 --> 192.168.53.99
From TUN interface: 192.168.53.99 --> 192.168.60.5
From socket: 192.168.60.5 --> 192.168.53.99
From TUN interface: 192.168.53.99 --> 192.168.60.5
From socket: 192.168.60.5 --> 192.168.53.99

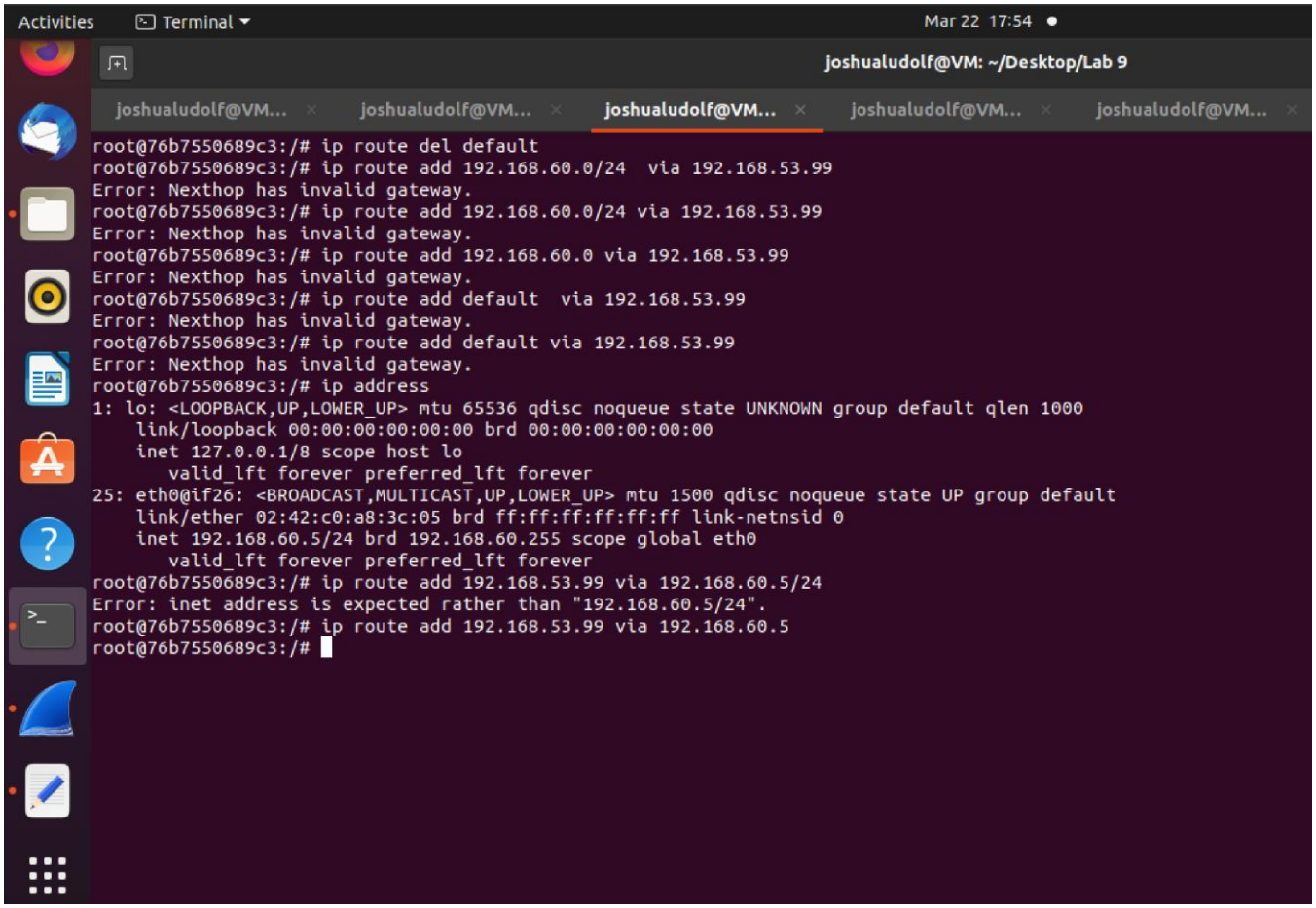

Activities
Terminal
Mar 22 17:38
joshualudolf@VM: ~/Desktop/Lab 9
root@584a45f02dfb:/# nano tun_server.py
root@584a45f02dfb:/# nano tun_server_2.py
root@584a45f02dfb:/# nano tun_server_2.py
root@584a45f02dfb:/# python3 tun_server_2.py
Interface Name: ludolf0
10.9.0.5:34416 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
From TUN interface: 192.168.60.5 --> 192.168.53.99
10.9.0.5:34416 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
From TUN interface: 192.168.60.5 --> 192.168.53.99
10.9.0.5:34416 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
From TUN interface: 192.168.60.5 --> 192.168.53.99
10.9.0.5:34416 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
From TUN interface: 192.168.60.5 --> 192.168.53.99
10.9.0.5:34416 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
From TUN interface: 192.168.60.5 --> 192.168.53.99
10.9.0.5:34416 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
From TUN interface: 192.168.60.5 --> 192.168.53.99
10.9.0.5:34416 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
From TUN interface: 192.168.60.5 --> 192.168.53.99
10.9.0.5:34416 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
From TUN interface: 192.168.60.5 --> 192.168.53.99
10.9.0.5:34416 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
From TUN interface: 192.168.60.5 --> 192.168.53.99
10.9.0.5:34416 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
From TUN interface: 192.168.60.5 --> 192.168.53.99
10.9.0.5:34416 --> 0.0.0.0:9090

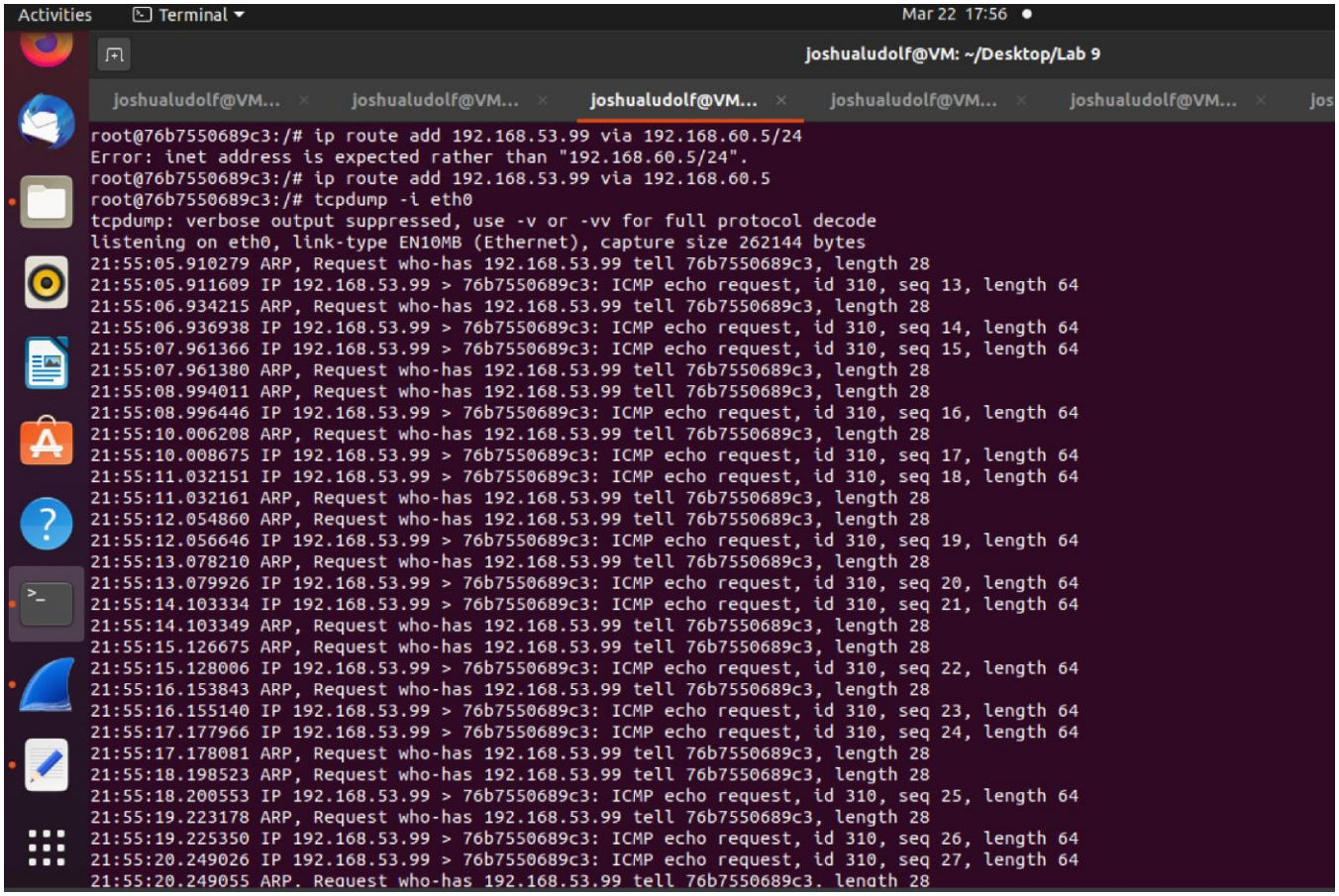


- **For task 6, it was time to see if I could break this tunnel by executing the telenet command on Host V, which didn't break the tunnel but rather asked me to login after I typed something and hit enter:**



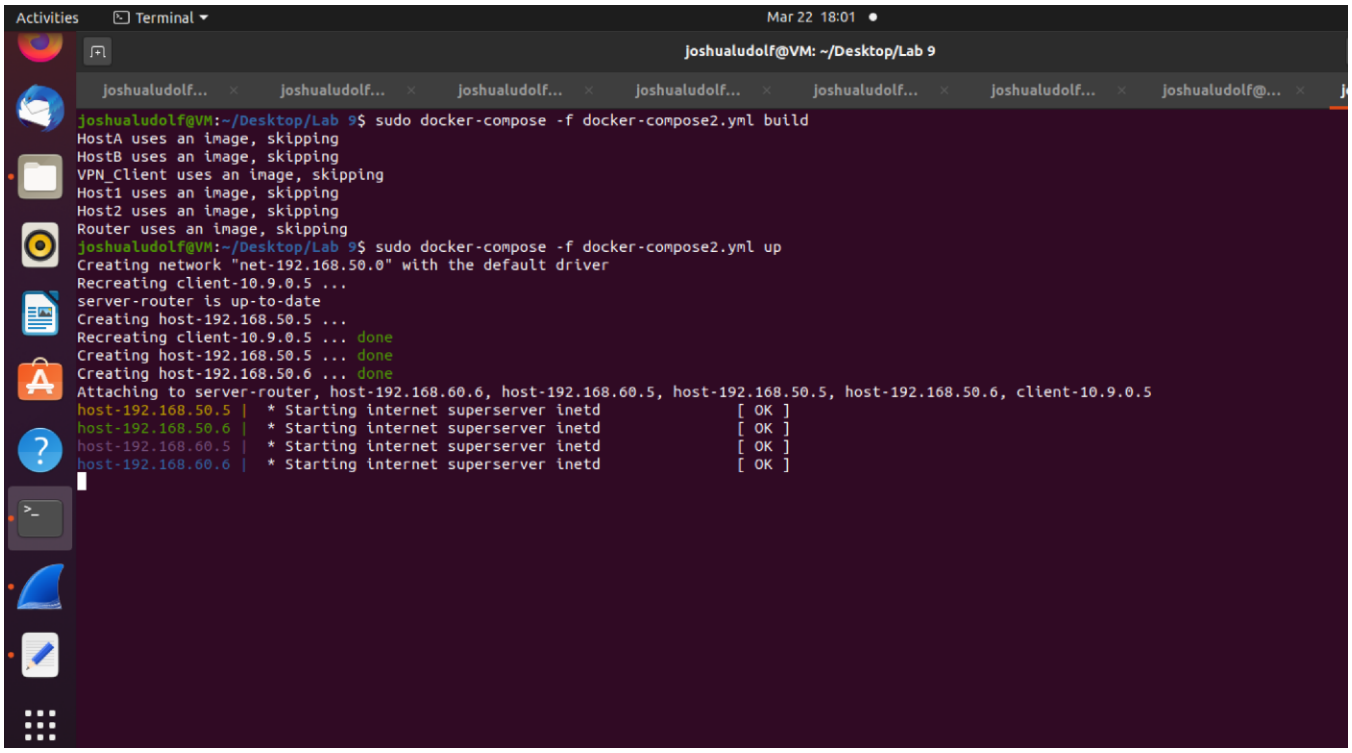
- **For task 7, Rounting experiment on Host V, which was really cool cause now it sends it to 76b7550689c3 (this is leaving the python scripts as they were, only executed terminal commands):**


Activities
Terminal
Mar 22 17:54
joshualudolf@VM: ~/Desktop/Lab 9
joshualudolf@VM...
joshualudolf@VM...
joshualudolf@VM...
joshualudolf@VM...
joshualudolf@VM...
root@76b7550689c3:/# ip route del default
root@76b7550689c3:/# ip route add 192.168.60.0/24 via 192.168.53.99
Error: Nexthop has invalid gateway.
root@76b7550689c3:/# ip route add 192.168.60.0/24 via 192.168.53.99
Error: Nexthop has invalid gateway.
root@76b7550689c3:/# ip route add 192.168.60.0 via 192.168.53.99
Error: Nexthop has invalid gateway.
root@76b7550689c3:/# ip route add default via 192.168.53.99
Error: Nexthop has invalid gateway.
root@76b7550689c3:/# ip route add default via 192.168.53.99
Error: Nexthop has invalid gateway.
root@76b7550689c3:/# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
25: eth0@if26: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
link/ether 02:42:c0:a8:3c:05 brd ff:ff:ff:ff:ff:ff link-netnsid 0
inet 192.168.60.5/24 brd 192.168.60.255 scope global eth0
valid_lft forever preferred_lft forever
root@76b7550689c3:/# ip route add 192.168.53.99 via 192.168.60.5/24
Error: inet address is expected rather than "192.168.60.5/24".
root@76b7550689c3:/# ip route add 192.168.53.99 via 192.168.60.5
root@76b7550689c3:/#

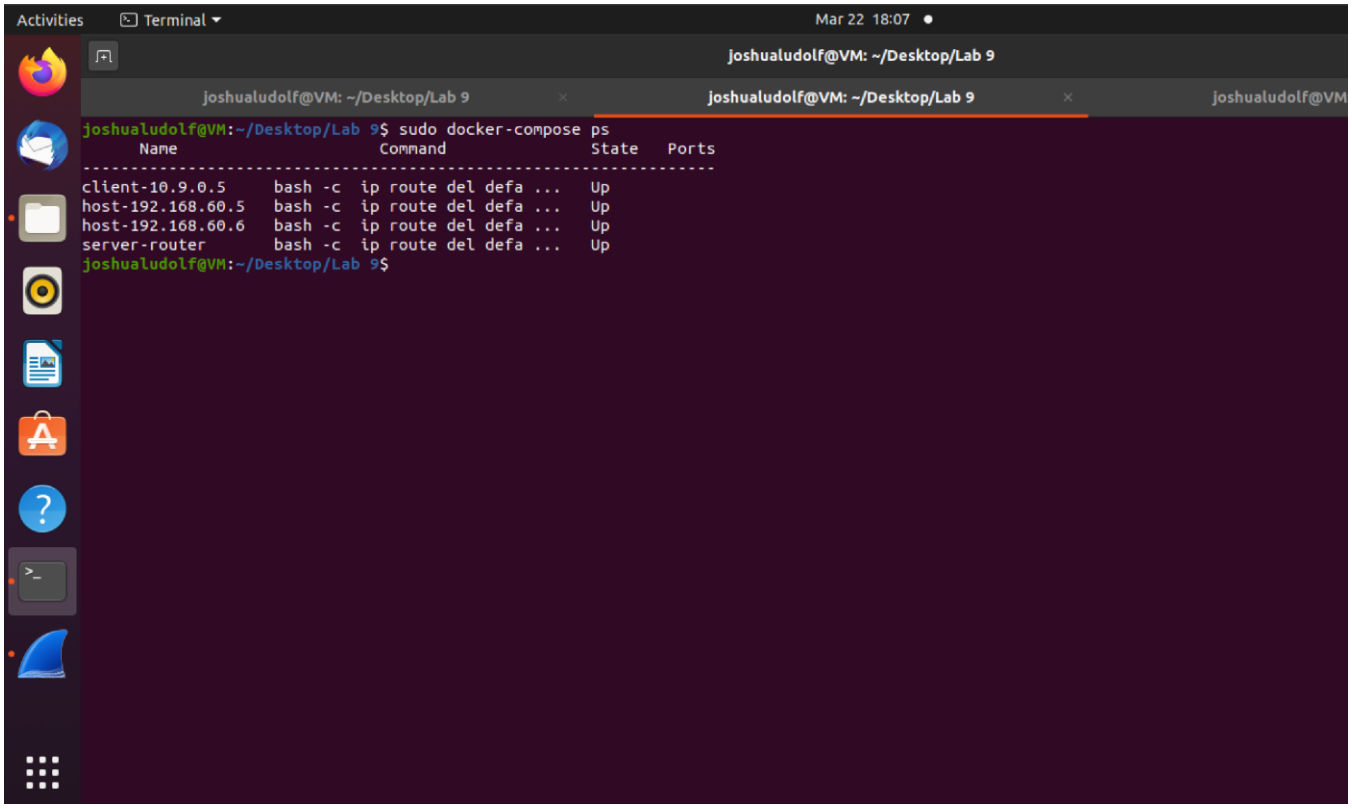


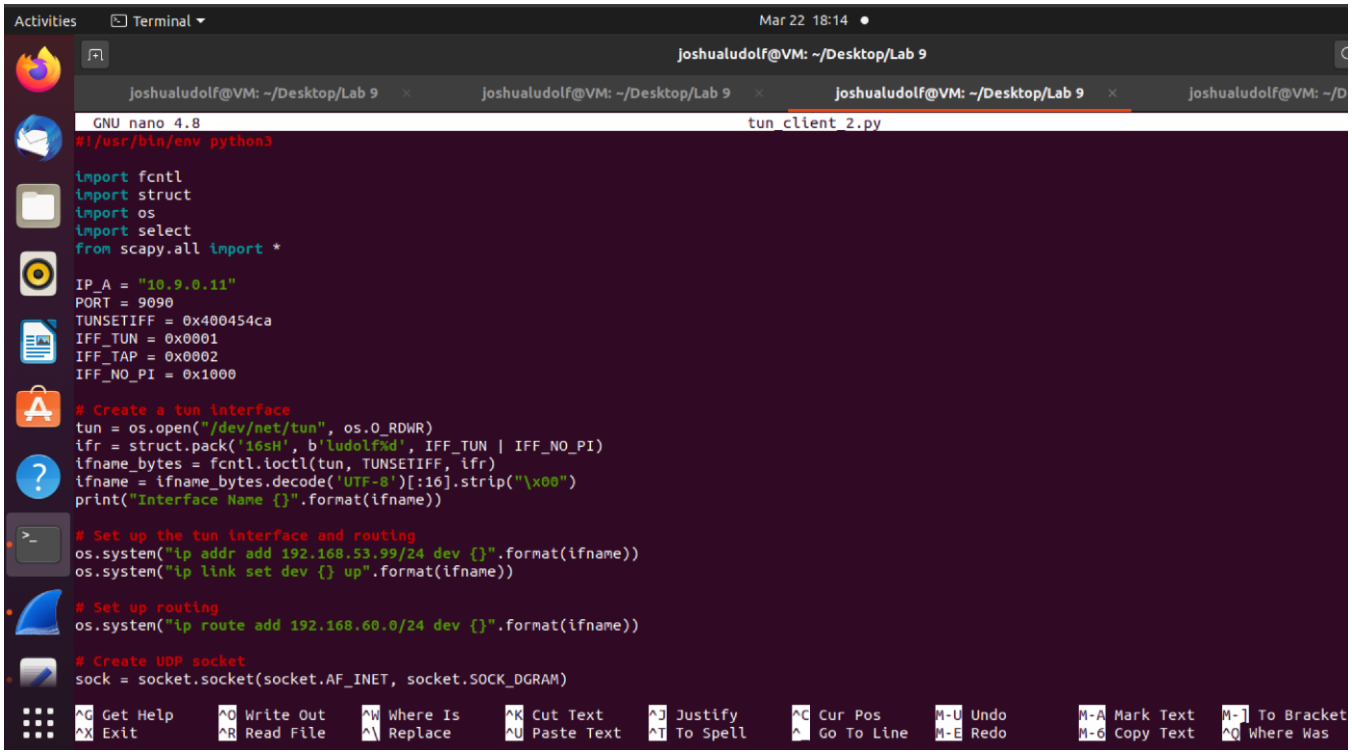

```
21:57:10.970117 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 310, seq 135, length 64
21:57:11.991943 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 310, seq 136, length 64
```

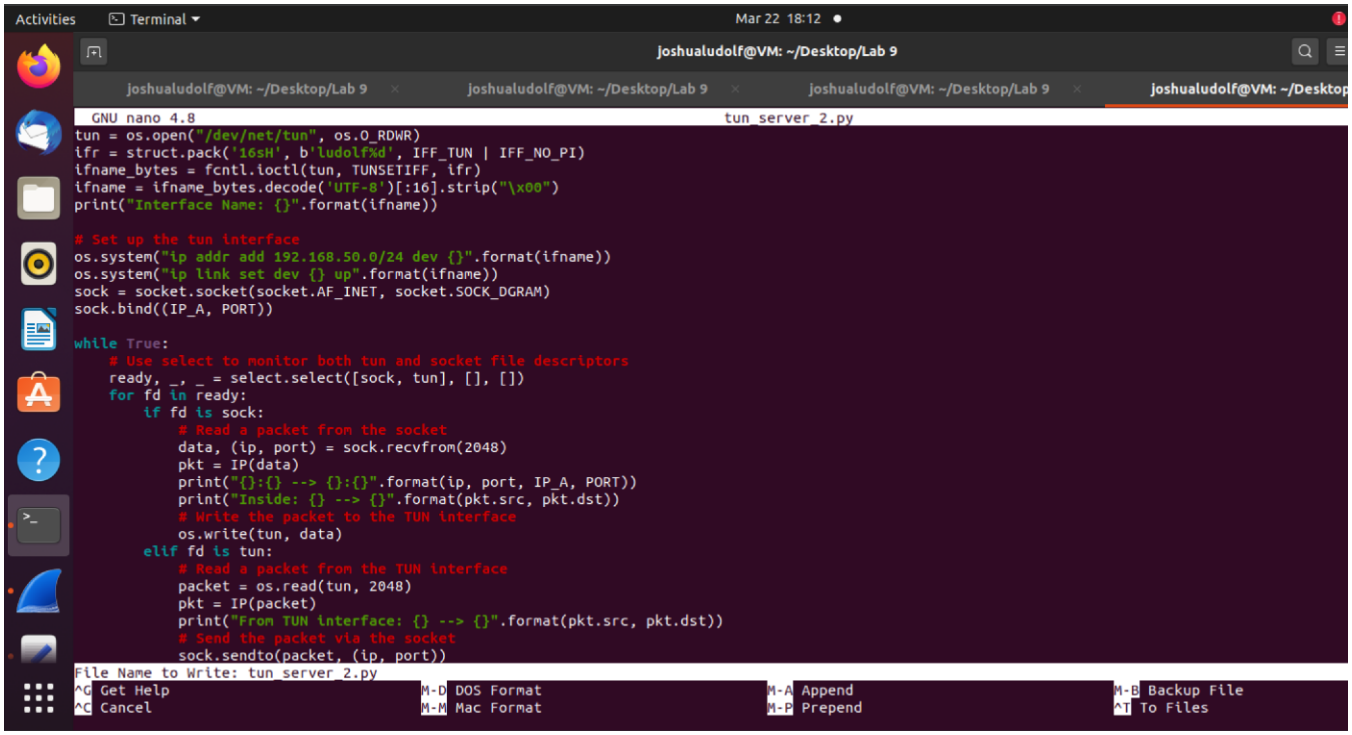
- **For task 8, I started with building and starting the docker-compose2.yml file:**

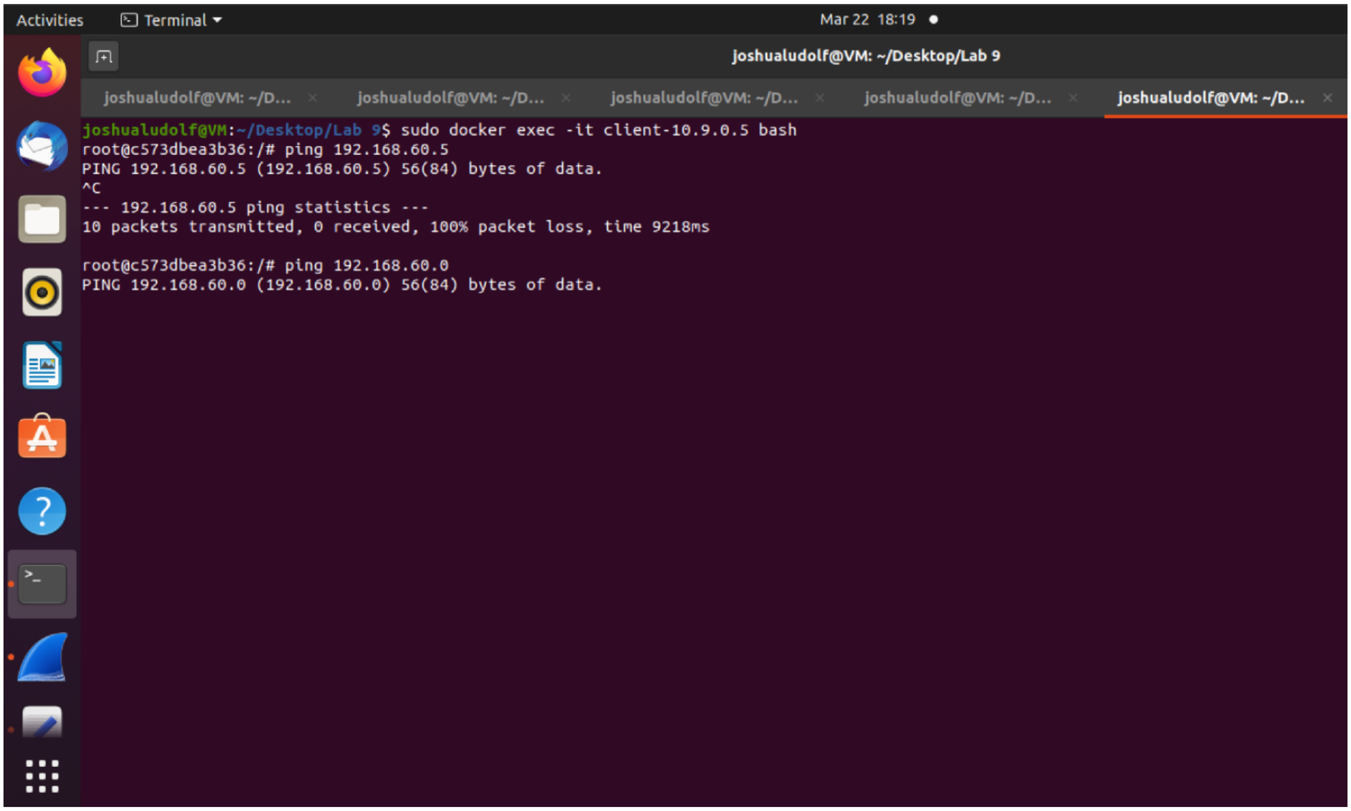


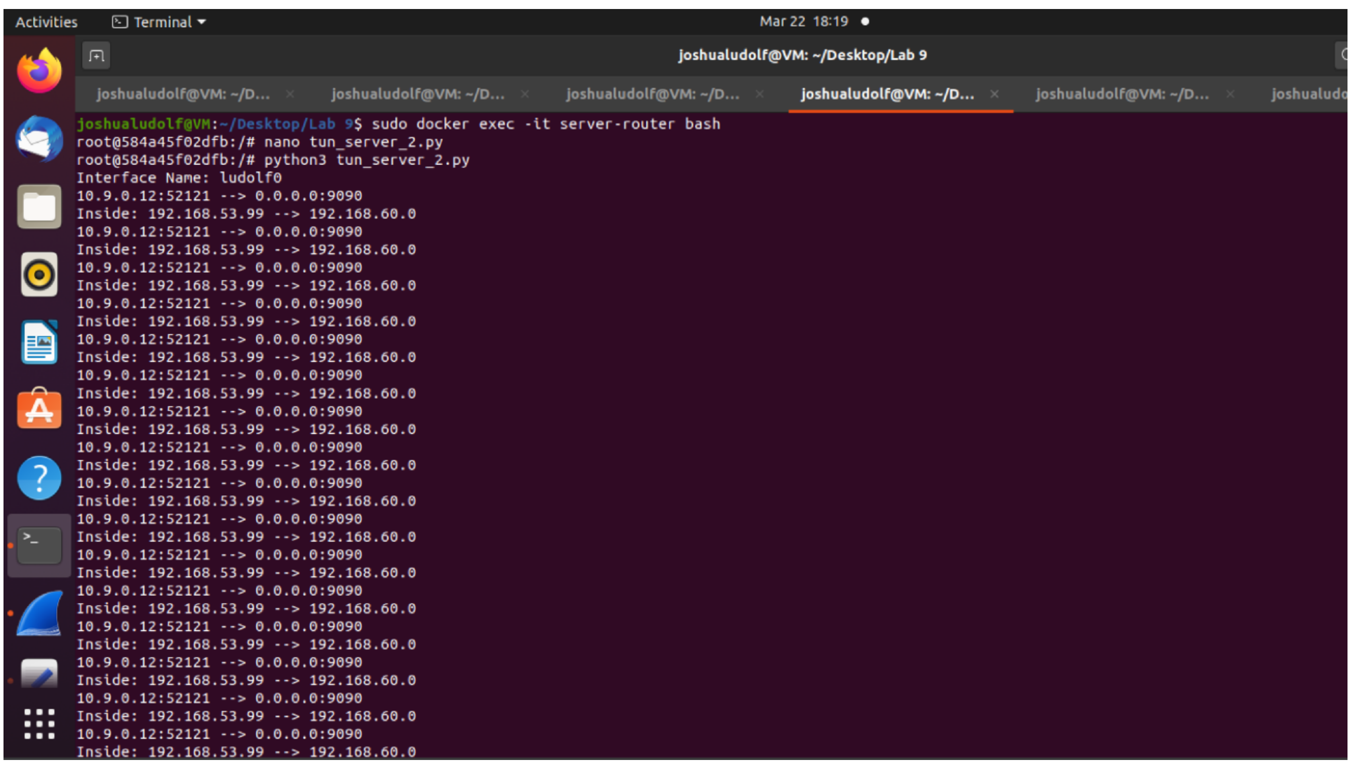
- **I added routes to each respective host to allow for communication through the vpn (so I reused my tun_client_2.py and tun_server_2.py with slight modification for ip routes added, not with my code):**

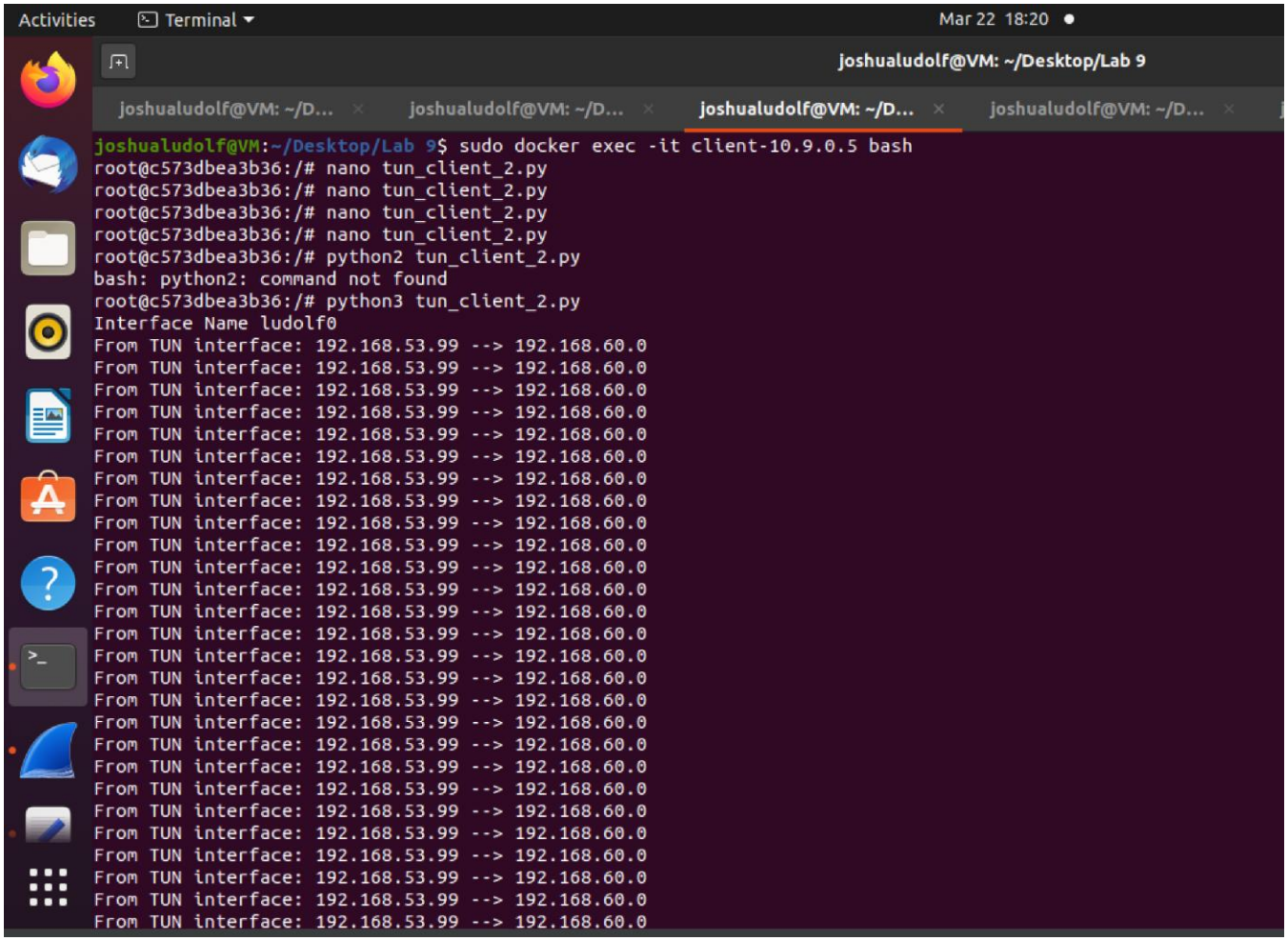


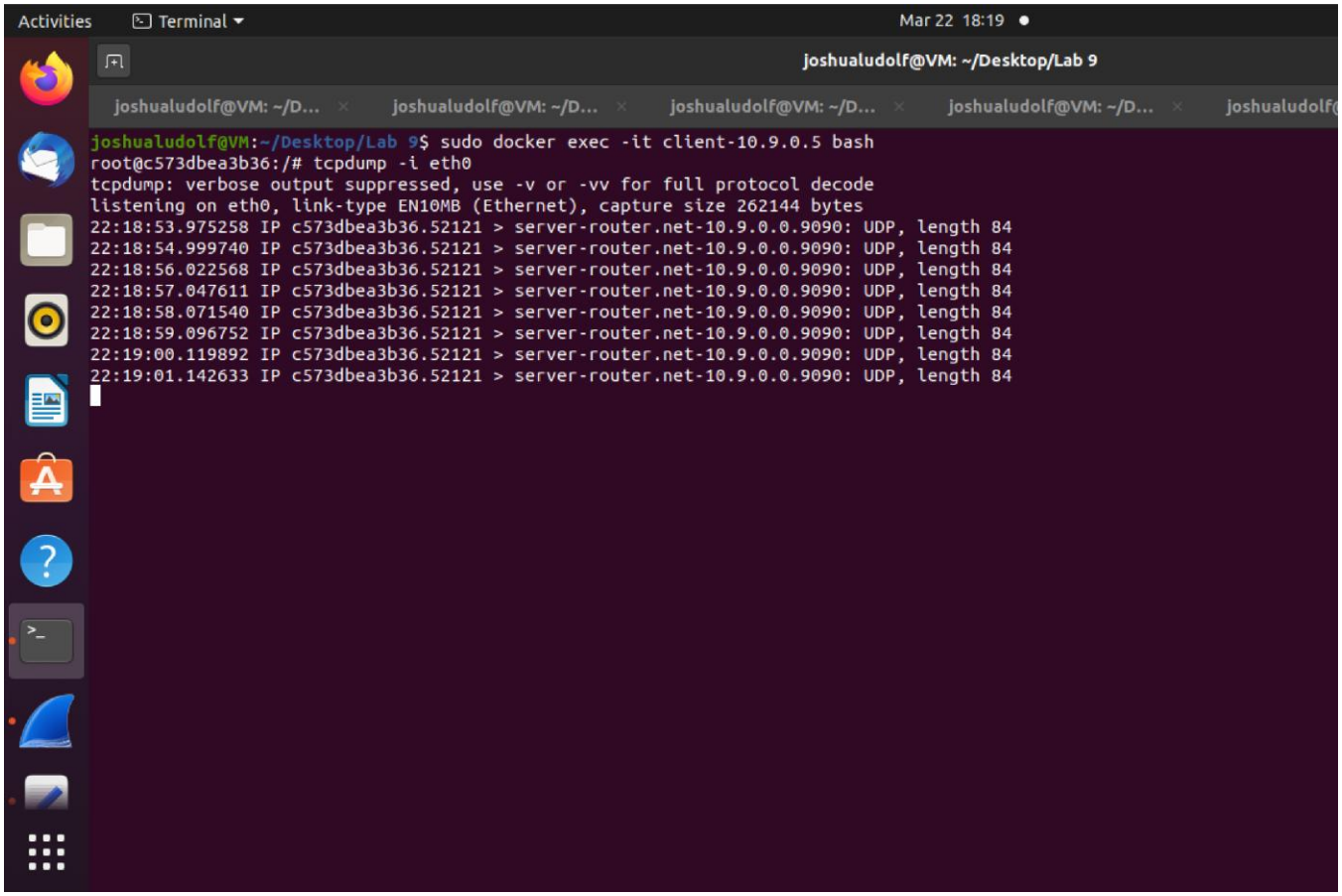

Activities
Terminal
Mar 22 18:14
joshualudolf@VM: ~/Desktop/Lab 9
joshualudolf@VM: ~/Desktop/Lab 9
joshualudolf@VM: ~/Desktop/Lab 9
joshualudolf@VM: ~/Desktop/Lab 9
joshualudolf@VM: ~/D
GNU nano 4.8
tun_client_2.py
#!/usr/bin/env python3
import fcntl
import struct
import os
import select
from scapy.all import *
IP_A = "10.9.0.11"
PORT = 9090
TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# Create a tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'ludolf%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface Name {}".format(ifname))
# Set up the tun interface and routing
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)
^G Get Help
^O Write Out
^W Where Is
^K Cut Text
^J Justify
^C Cur Pos
M-U Undo
M-A Mark Text
M-] To Bracket
^X Exit
^R Read File
^\ Replace
^U Paste Text
^T To Spell
^_ Go To Line
M-E Redo
M-6 Copy Text
^Q Where Was


Activities
Terminal
Mar 22 18:12
joshualudolf@VM: ~/Desktop/Lab 9
joshualudolf@VM: ~/Desktop/Lab 9
joshualudolf@VM: ~/Desktop/Lab 9
joshualudolf@VM: ~/Desktop/Lab 9
joshualudolf@VM: ~/Desktop
GNU nano 4.8
tun_server_2.py
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'ludolf%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface Name: {}".format(ifname))
# Set up the tun interface
os.system("ip addr add 192.168.50.0/24 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:
# Use select to monitor both tun and socket file descriptors
ready, _, _ = select.select([sock, tun], [], [])
for fd in ready:
if fd is sock:
# Read a packet from the socket
data, (ip, port) = sock.recvfrom(2048)
pkt = IP(data)
print("{}:{} --> {}:{}".format(ip, port, IP_A, PORT))
print("Inside: {} --> {}".format(pkt.src, pkt.dst))
# Write the packet to the TUN interface
os.write(tun, data)
elif fd is tun:
# Read a packet from the TUN interface
packet = os.read(tun, 2048)
pkt = IP(packet)
print("From TUN interface: {} --> {}".format(pkt.src, pkt.dst))
# Send the packet via the socket
sock.sendto(packet, (ip, port))
File Name to Write: tun_server_2.py
^G Get Help
M-D DOS Format
M-A Append
M-B Backup File
^C Cancel
M-M Mac Format
M-P Prepend
^T To Files

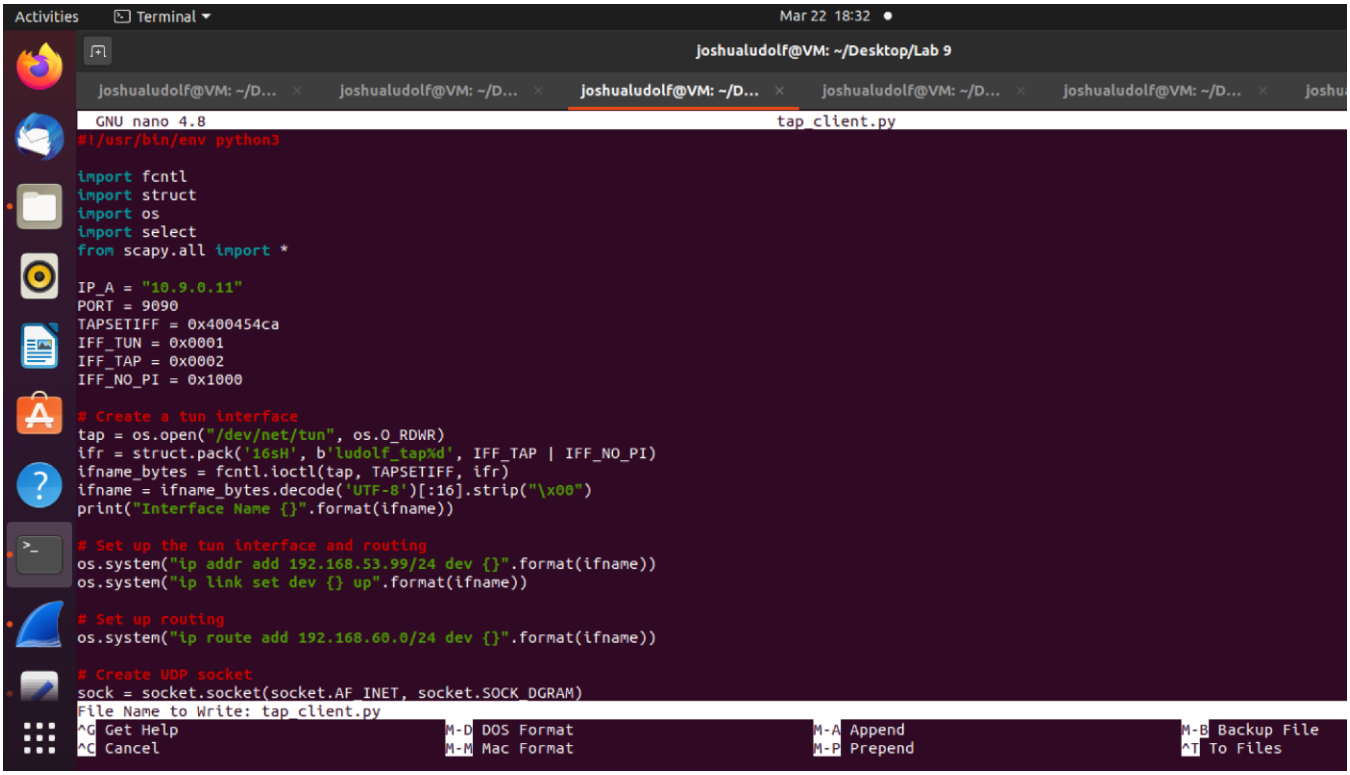

Activities
Terminal
Mar 22 18:19
joshualudolf@VM: ~/Desktop/Lab 9
joshualudolf@VM:~/Desktop/Lab 9$ sudo docker exec -it client-10.9.0.5 bash
root@c573dbea3b36:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
10 packets transmitted, 0 received, 100% packet loss, time 9218ms
root@c573dbea3b36:/# ping 192.168.60.0
PING 192.168.60.0 (192.168.60.0) 56(84) bytes of data.

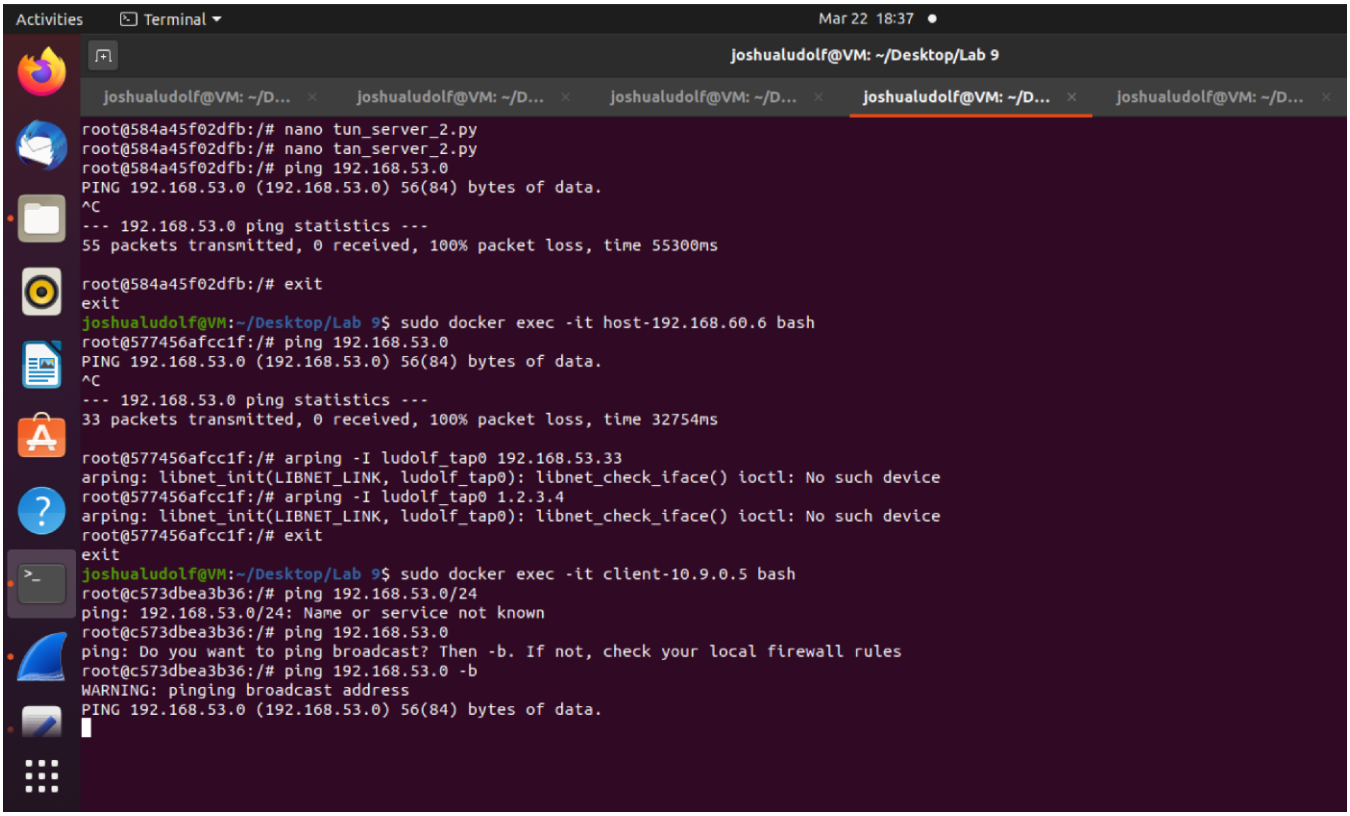

Activities
Terminal
Mar 22 18:19
joshualudolf@VM: ~/Desktop/Lab 9
joshualudolf@VM:~/Desktop/Lab 9$ sudo docker exec -it server-router bash
root@584a45f02dfb:/# nano tun_server_2.py
root@584a45f02dfb:/# python3 tun_server_2.py
Interface Name: ludolf0
10.9.0.12:52121 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.0
10.9.0.12:52121 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.0
10.9.0.12:52121 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.0
10.9.0.12:52121 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.0
10.9.0.12:52121 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.0
10.9.0.12:52121 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.0
10.9.0.12:52121 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.0
10.9.0.12:52121 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.0
10.9.0.12:52121 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.0
10.9.0.12:52121 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.0
10.9.0.12:52121 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.0
10.9.0.12:52121 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.0
10.9.0.12:52121 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.0
10.9.0.12:52121 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.0
10.9.0.12:52121 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.0
10.9.0.12:52121 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.0

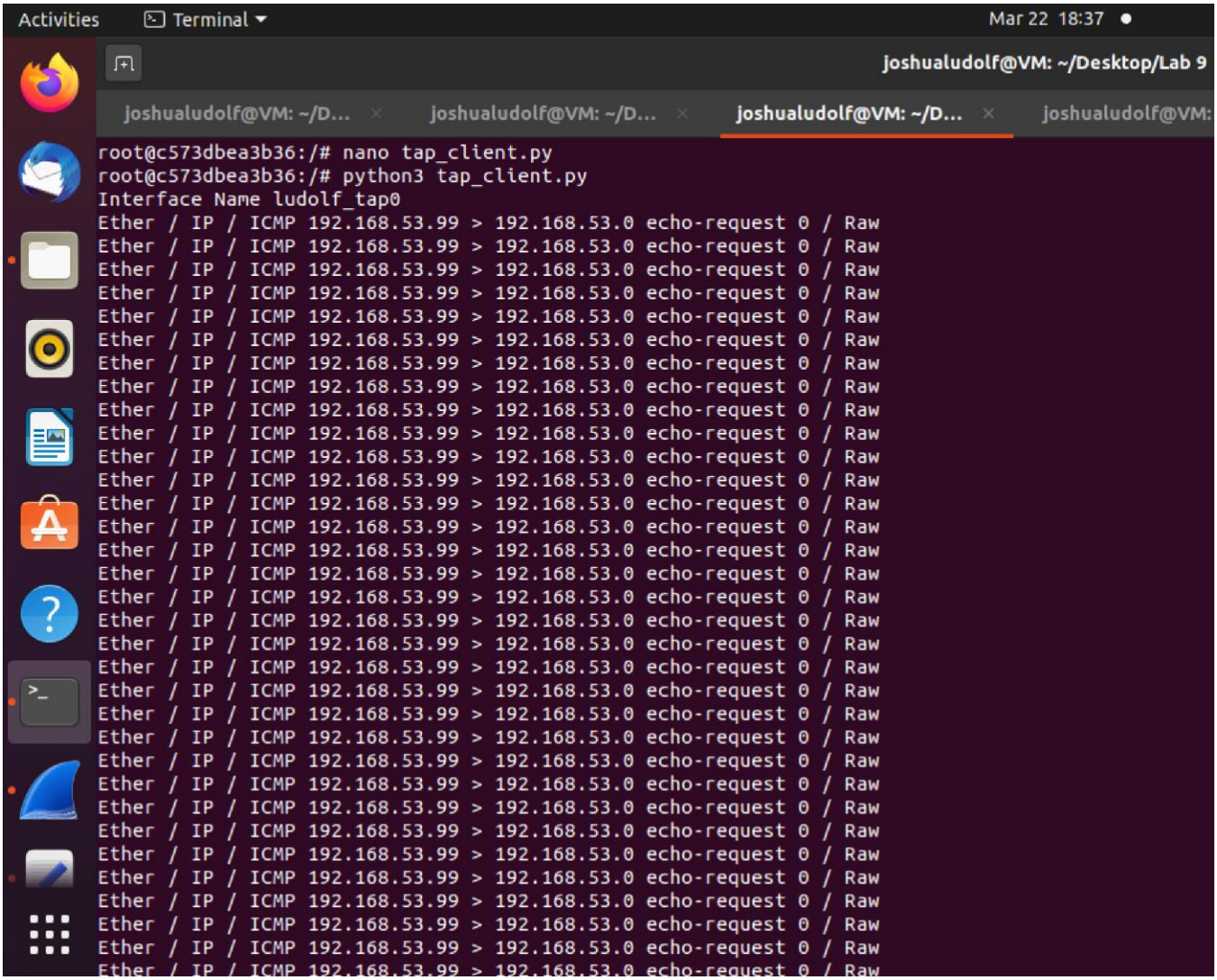

Activities
Terminal
Mar 22 18:20
joshualudolf@VM: ~/Desktop/Lab 9
joshualudolf@VM: ~/D...
joshualudolf@VM: ~/D...
joshualudolf@VM: ~/D...
joshualudolf@VM: ~/D...
joshualudolf@VM:~/Desktop/Lab 9$ sudo docker exec -it client-10.9.0.5 bash
root@c573dbea3b36:/# nano tun_client_2.py
root@c573dbea3b36:/# nano tun_client_2.py
root@c573dbea3b36:/# nano tun_client_2.py
root@c573dbea3b36:/# nano tun_client_2.py
root@c573dbea3b36:/# python2 tun_client_2.py
bash: python2: command not found
root@c573dbea3b36:/# python3 tun_client_2.py
Interface Name ludolf0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0
From TUN interface: 192.168.53.99 --> 192.168.60.0



- **Finally, in the final task I experimented with the TAP Interface as it is like TUN Interface, the difference is that the kernel end of the TUN interface is hooked to the IP layer, while the kernel end of the TAP interface is hooked to the MAC layer. Therefore, the packet going through the TAP interface includes the MAC header, while the packet going through the TUN interface only includes the IP header. Other than getting the frames containing IP packets, using the TAP interface, applications can also get other types of frames, such as ARP frames. (for the first part it only worked with host on same network, I didn't do second part after that, didn't want to do anymore for now):**


Activities
Terminal
Mar 22 18:32
joshualudolf@VM: ~/Desktop/Lab 9
joshualudolf@VM: ~/D...
joshualudolf@VM: ~/D...
joshualudolf@VM: ~/D...
joshualudolf@VM: ~/D...
joshualudolf@VM: ~/D...
GNU nano 4.8
tap_client.py
#!/usr/bin/env python3
import fcntl
import struct
import os
import select
from scapy.all import *
IP_A = "10.9.0.11"
PORT = 9090
TAPSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# Create a tun interface
tap = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'ludolf_tap%d', IFF_TAP | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tap, TAPSETIFF, ifr)
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface Name {}".format(ifname))
# Set up the tun interface and routing
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)
File Name to Write: tap_client.py
^G Get Help
M-D DOS Format
M-A Append
M-B Backup File
^C Cancel
M-M Mac Format
M-P Prepend
^T To Files


Activities
Terminal
Mar 22 18:37
joshualudolf@VM: ~/Desktop/Lab 9
joshualudolf@VM: ~/D...
joshualudolf@VM: ~/D...
joshualudolf@VM: ~/D...
joshualudolf@VM: ~/D...
joshualudolf@VM: ~/D...
root@584a45f02dfb:/# nano tun_server_2.py
root@584a45f02dfb:/# nano tan_server_2.py
root@584a45f02dfb:/# ping 192.168.53.0
PING 192.168.53.0 (192.168.53.0) 56(84) bytes of data.
^C
--- 192.168.53.0 ping statistics ---
55 packets transmitted, 0 received, 100% packet loss, time 55300ms
root@584a45f02dfb:/# exit
exit
joshualudolf@VM:~/Desktop/Lab 9$ sudo docker exec -it host-192.168.60.6 bash
root@577456afcc1f:/# ping 192.168.53.0
PING 192.168.53.0 (192.168.53.0) 56(84) bytes of data.
^C
--- 192.168.53.0 ping statistics ---
33 packets transmitted, 0 received, 100% packet loss, time 32754ms
root@577456afcc1f:/# arping -I ludolf_tap0 192.168.53.33
arping: libnet_init(LIBNET_LINK, ludolf_tap0): libnet_check_iface() ioctl: No such device
root@577456afcc1f:/# arping -I ludolf_tap0 1.2.3.4
arping: libnet_init(LIBNET_LINK, ludolf_tap0): libnet_check_iface() ioctl: No such device
root@577456afcc1f:/# exit
exit
joshualudolf@VM:~/Desktop/Lab 9$ sudo docker exec -it client-10.9.0.5 bash
root@c573dbea3b36:/# ping 192.168.53.0/24
ping: 192.168.53.0/24: Name or service not known
root@c573dbea3b36:/# ping 192.168.53.0
ping: Do you want to ping broadcast? Then -b. If not, check your local firewall rules
root@c573dbea3b36:/# ping 192.168.53.0 -b
WARNING: pinging broadcast address
PING 192.168.53.0 (192.168.53.0) 56(84) bytes of data.



❖ **What I learned from this lab:**

n this lab, I learned the fundamental concepts behind building a VPN tunnel between two private networks. I began by understanding the idea of a VPN tunnel—how it securely encapsulates IP packets, allowing two separate networks to communicate over an untrusted medium like the Internet. Although encryption was not the focus of this exercise, I grasped the underlying mechanisms used in real-world VPNs where traffic is both encapsulated and, eventually, encrypted for confidentiality and integrity.

Working with Linux's TUN/TAP interfaces was a major hands-on experience. I created a virtual TUN interface using and learned how to set it up by configuring the appropriate flags with system calls like . This process showed me how low-level network interfaces can be manipulated to intercept and inject packets, providing the basis for tunneling traffic between networks.

I also delved into socket programming in Python, using UDP sockets to send and receive encapsulated packets through the tunnel. Integrating the socket with the TUN interface not only bridged the virtual and physical network realms but also demonstrated how traffic could be rerouted effectively. Using the Python system call was particularly enlightening, as it allowed me to efficiently monitor multiple file descriptors—ensuring that the process responded only when activity was detected on either the tunnel or the socket. This was a key step in preventing unnecessary CPU usage and achieving smooth bidirectional communication.

Furthermore, I gained a deeper understanding of advanced routing techniques. I had to modify the routing tables on both sides to ensure that packets destined for the remote network were correctly routed through the VPN tunnel. This involved deleting default routes and adding specific entries to direct traffic explicitly to the VPN endpoints, highlighting the complexities of managing return traffic in multi-hop, real-world scenarios.

Overall, this lab not only solidified my grasp of VPN tunnel fundamentals and low-level network interface configuration but also equipped me with practical skills in efficient I/O handling, socket programming, and routing management. These insights have laid a strong foundation for exploring more advanced topics like encrypted VPN communication and dynamic routing protocols in the future.