Setup

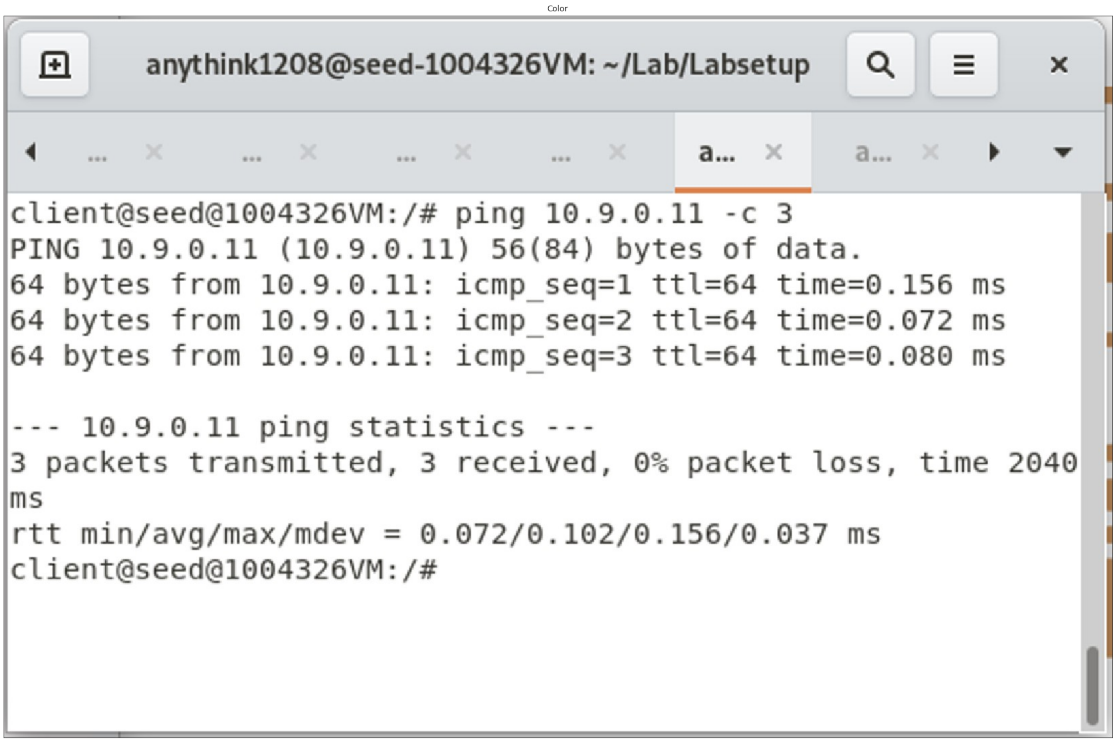
```
import cv2
from matplotlib import pyplot as plt

# This is a bit of magic to make matplotlib figures appear inline in
the notebook
# rather than in a new window.
%matplotlib inline
plt.rcParams['figure.figsize'] = (100.0, 80.0) # set default size of
plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'


def show_img(img):
    img = cv2.imread(img,-1)
    plt.subplot(131),plt.imshow(img),
    plt.title('Color'),plt.xticks([]), plt.yticks([])
    plt.show()
```

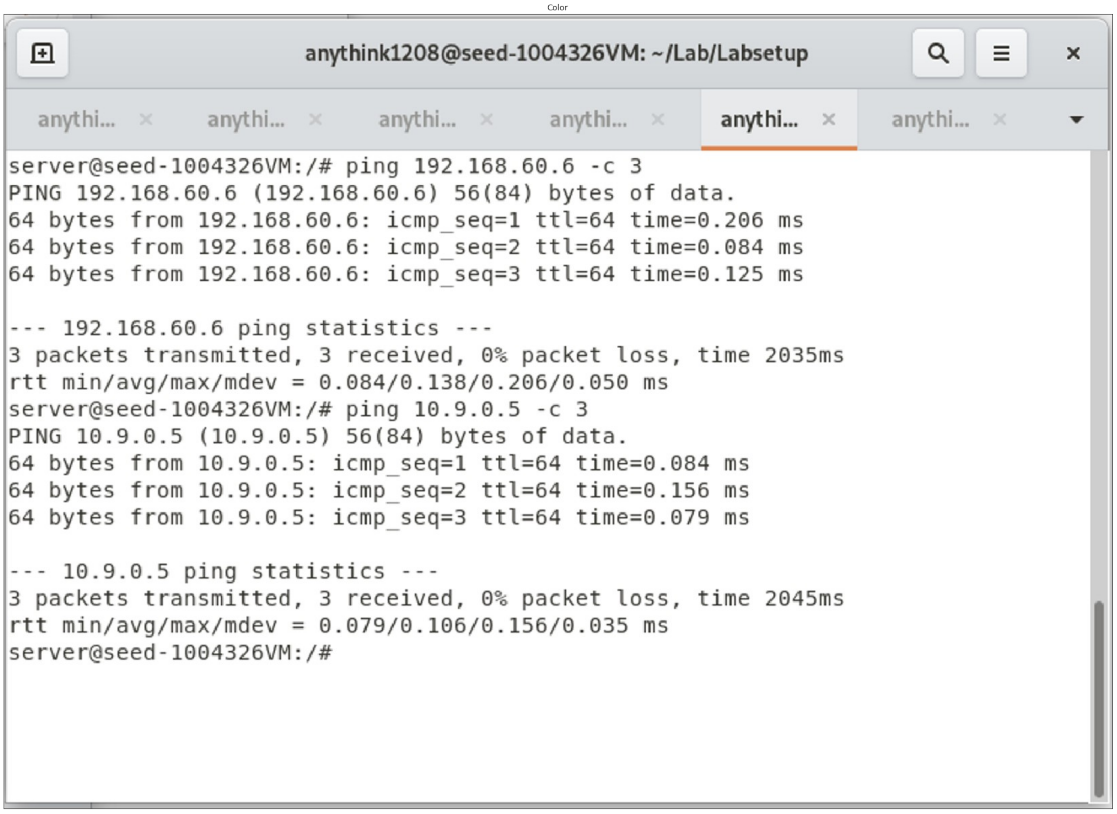
U (10.9.0.5) can communicate with VPN Server (10.9.0.11)

```
show_img('setup/client_server_comm.png')
```



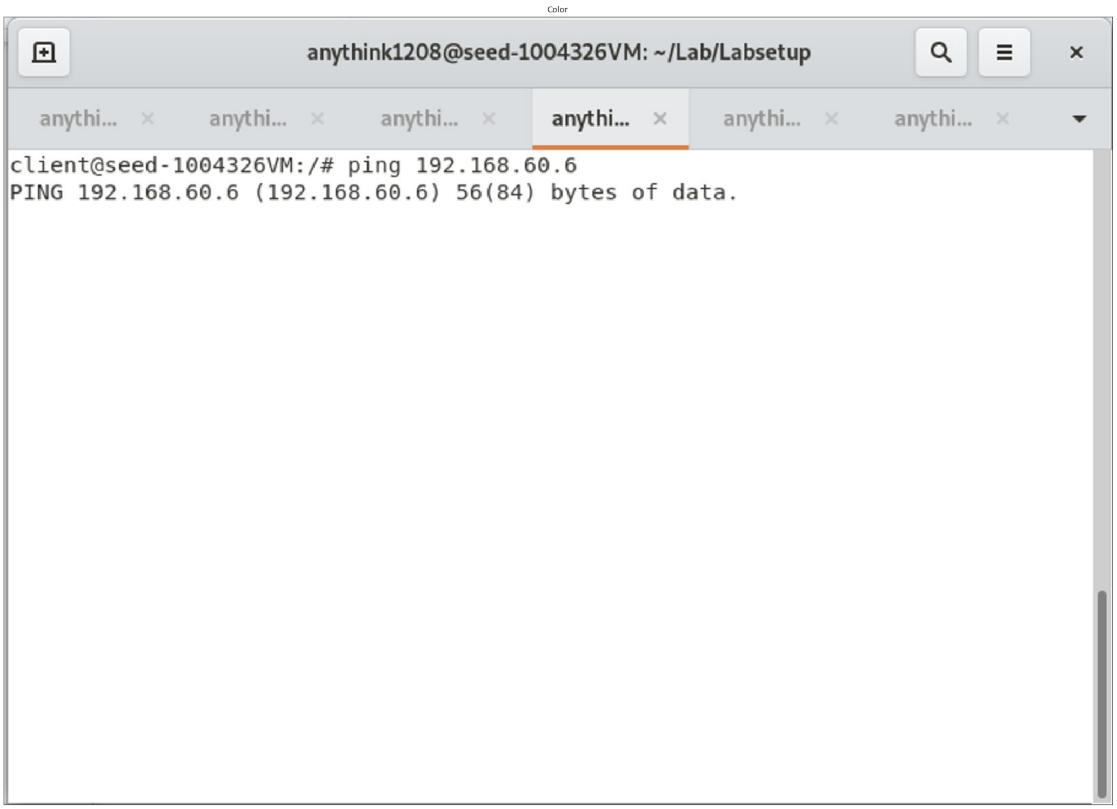
VPN Server (192.168.60.11) is able to communicate with V (192.168.60.6)

```
show_img('setup/server_host_comm.png')
```



U (10.9.0.5) is unable to ping V (192.168.60.6)

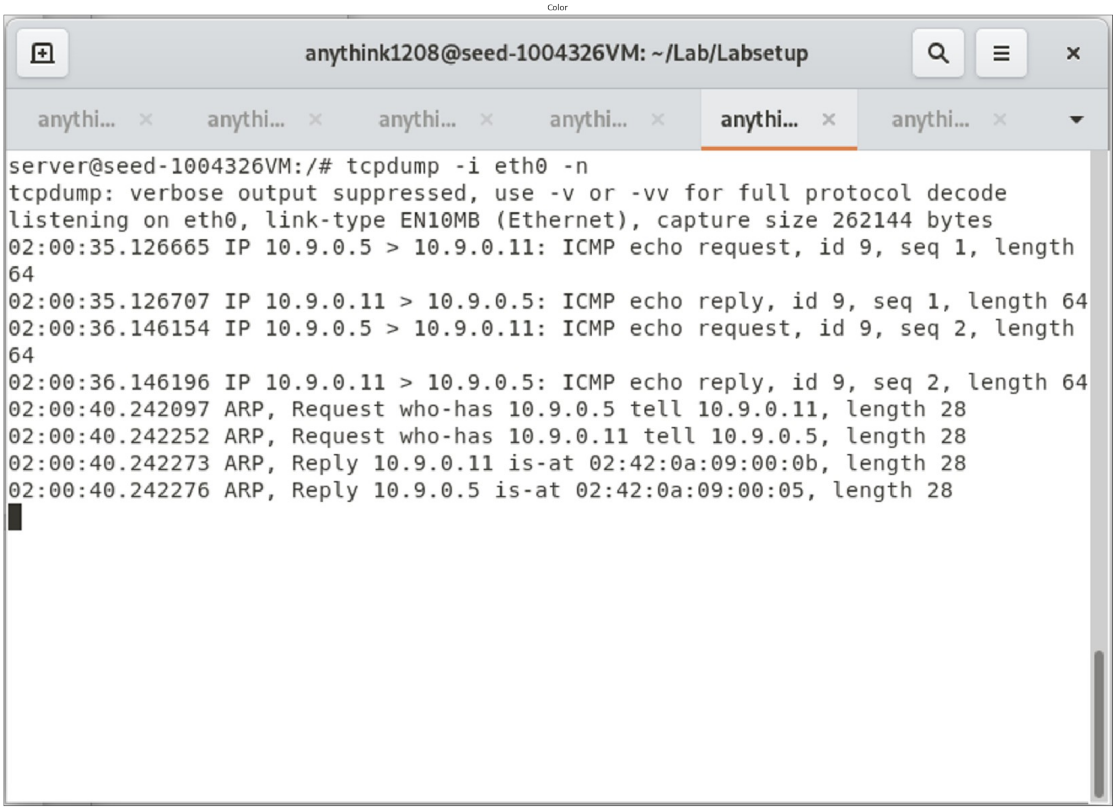
```
show_img('setup/client_host_comm.png')
```



Run tcpdump on the router, and sniff the packet on each network. Capture the packet

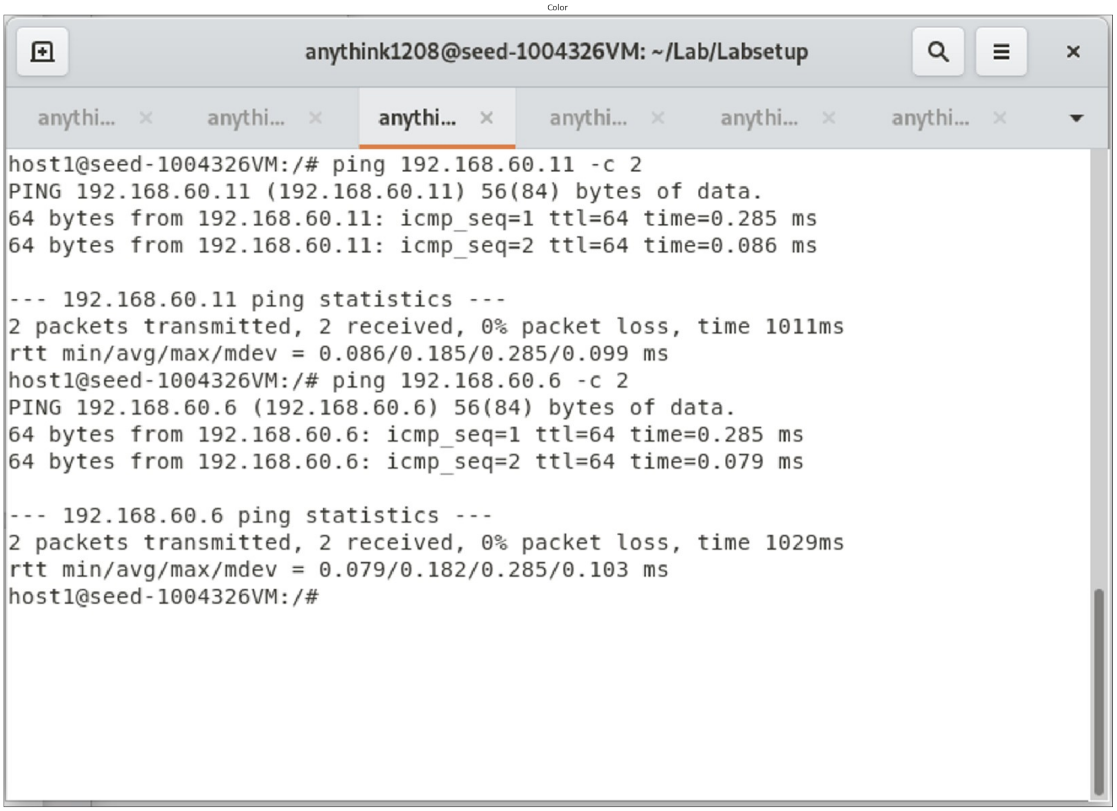
Testing eth0

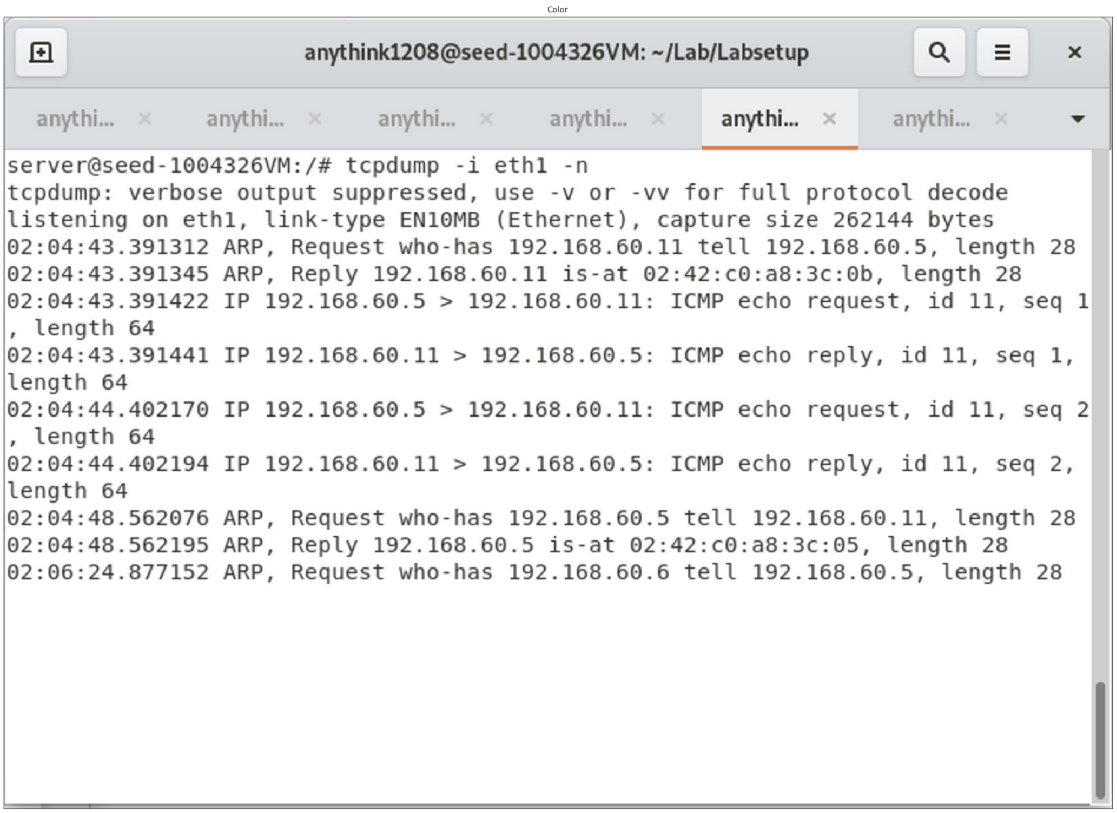
```
show_img('setup/eth0_test_ping.png')
show_img('setup/tcpdump_eth0.png')
```



Testing eth1

```
show_img('setup/eth1_test_ping.png')
show_img('setup/tcpdump_eth1.png')
```





Task 2.a: Name of the Interface

```
show_img('Task 2a/check_ip_addr.png')
```

Color

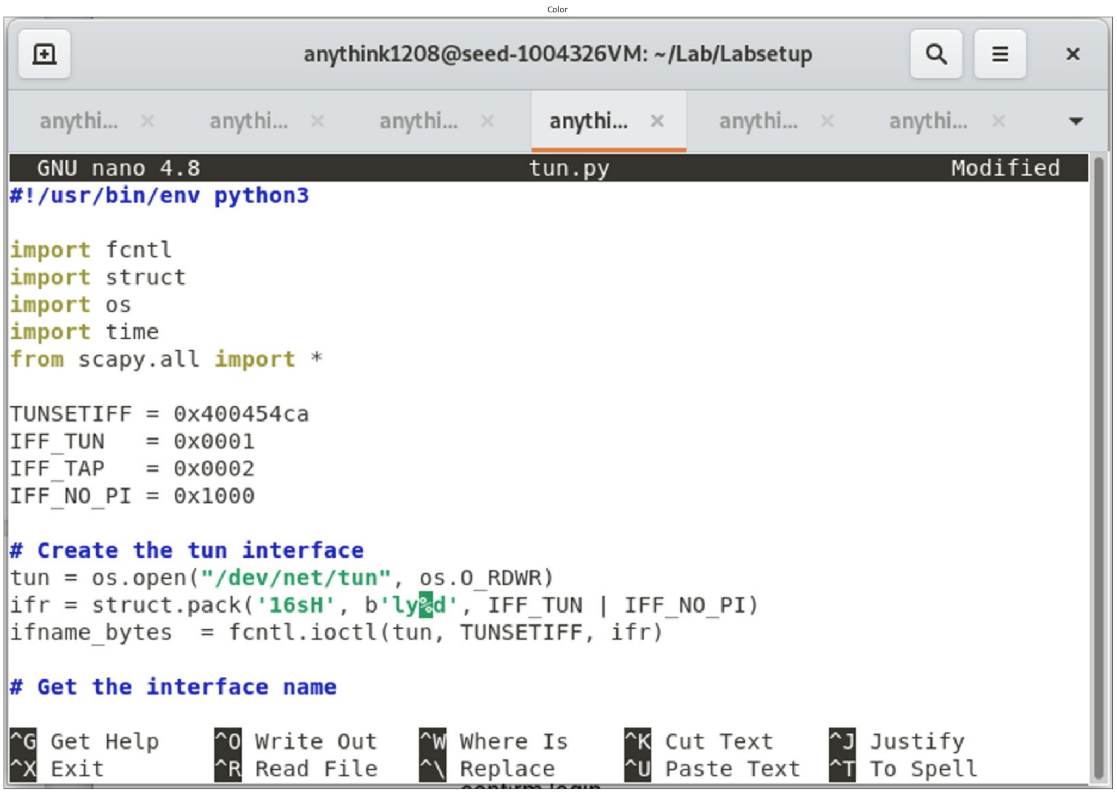
```
client@seed-1004326VM:/volumes# ./tun.py &
[1] 53
client@seed-1004326VM:/volumes# Interface Name: tun0

client@seed-1004326VM:/volumes# jobs
[1]+  Running                 ./tun.py &
client@seed-1004326VM:/volumes# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.5  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:05  txqueuelen 0  (Ethernet)
        RX packets 149  bytes 16726 (16.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 52  bytes 4480 (4.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

client@seed-1004326VM:/volumes# ip addr
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
4: tun0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group default qlen 500
    link/none
8: eth0@if9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:0a:09:00:05 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.9.0.5/24 brd 10.9.0.255 scope global eth0
       valid_lft forever preferred_lft forever
client@seed-1004326VM:/volumes#
```

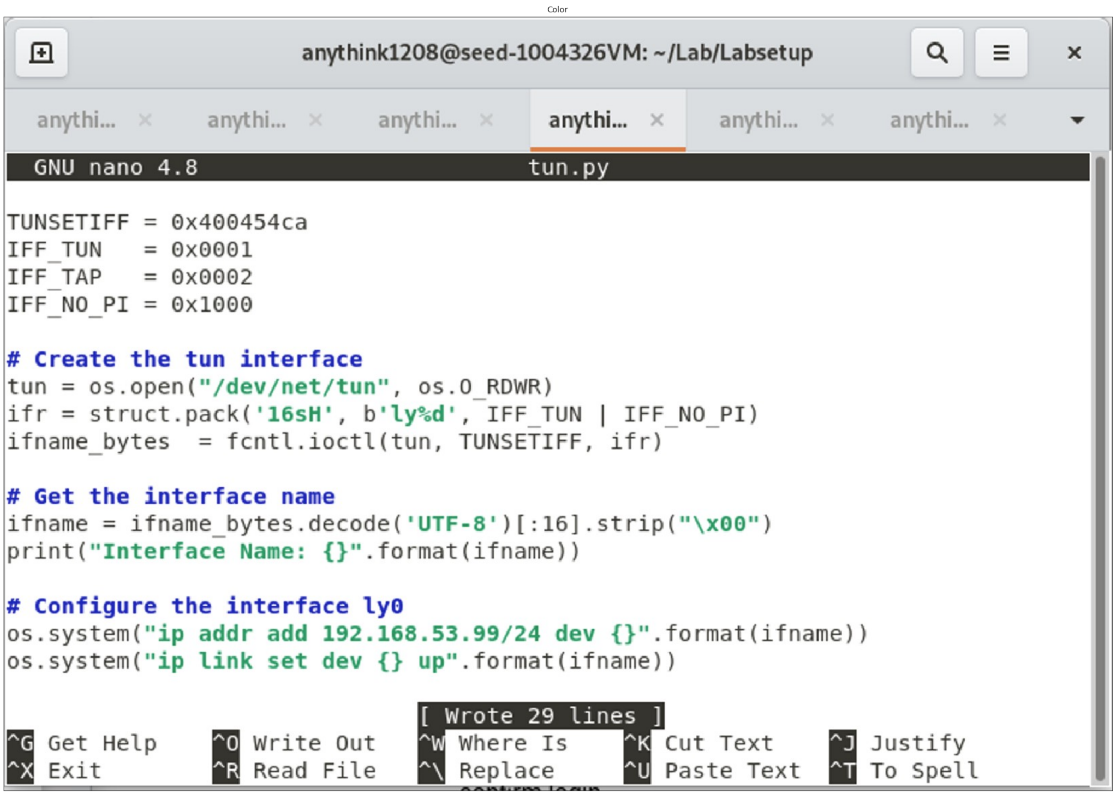
```
show_img('Task 2a/rename_tun.png')
```



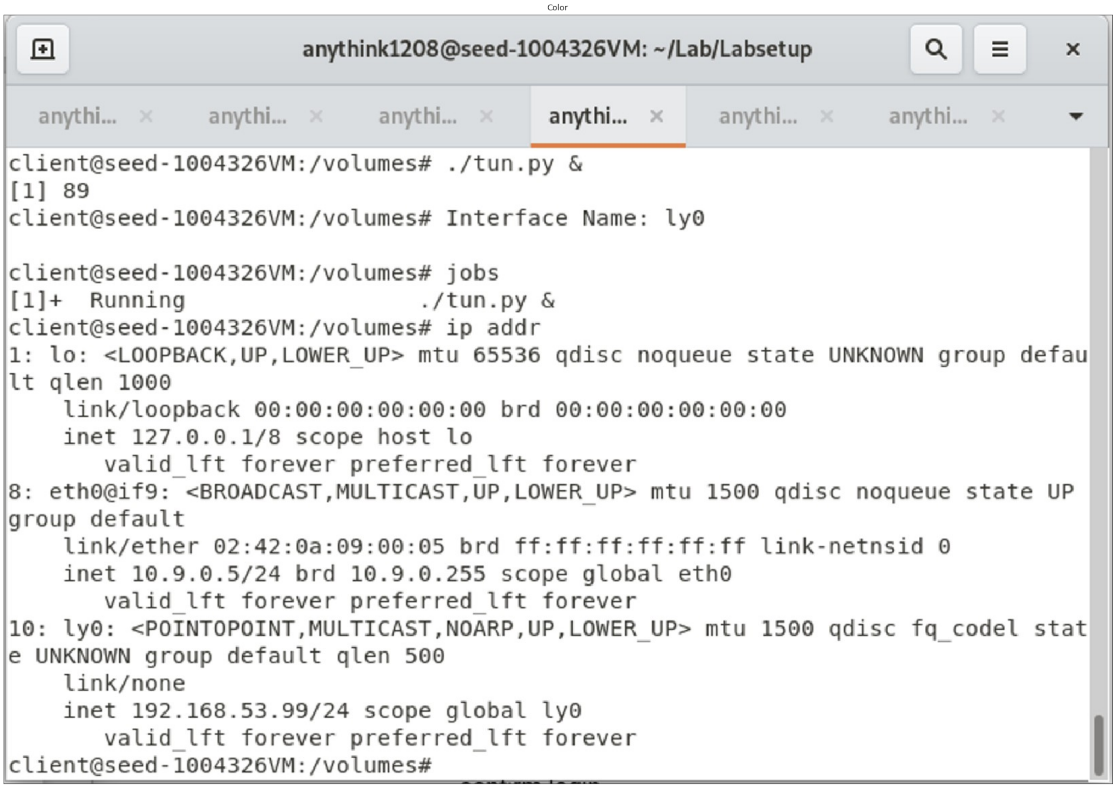
Task 2.b: Set up the TUN Interface

Instead of running `sudo ip addr add 192.168.53.99/24 dev tun0 && sudo ip link set dev tun0 up`, I added it in the tun.py script, configuration is automatically performed by the program

```
show_img('Task 2b/set_ly0_tun.png')
```



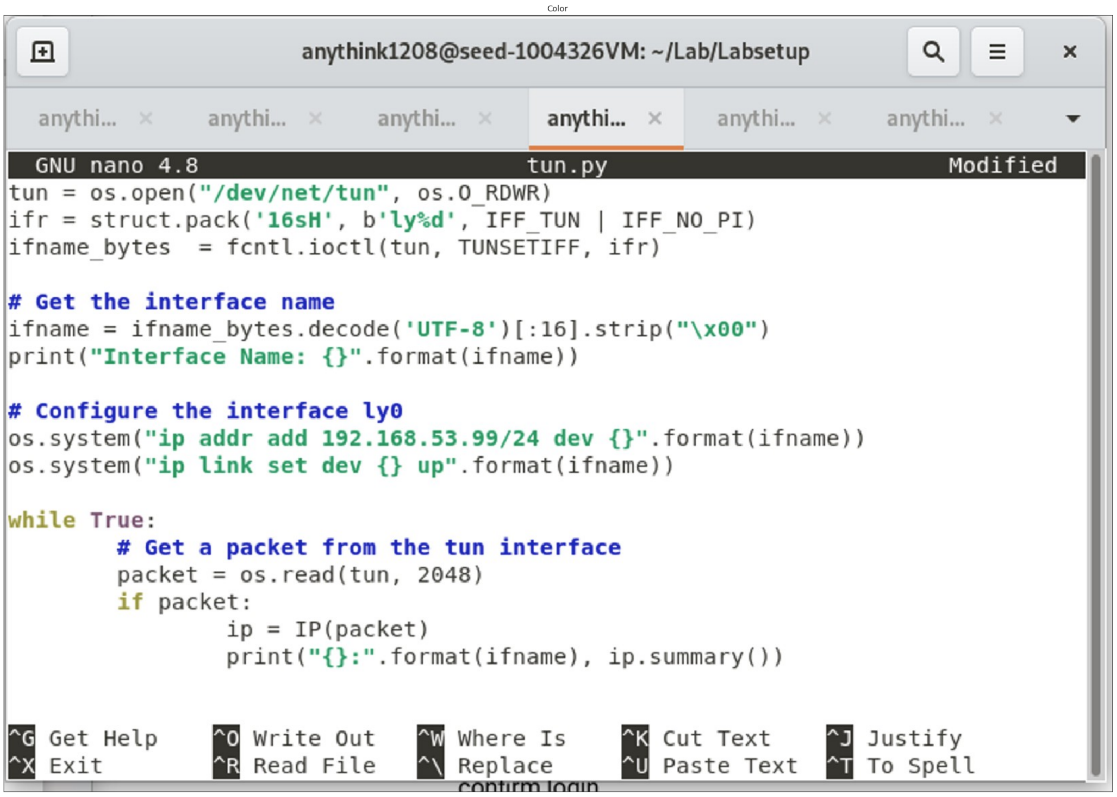

```
show_img('Task 2b/new_ip_addr_out.png')
```



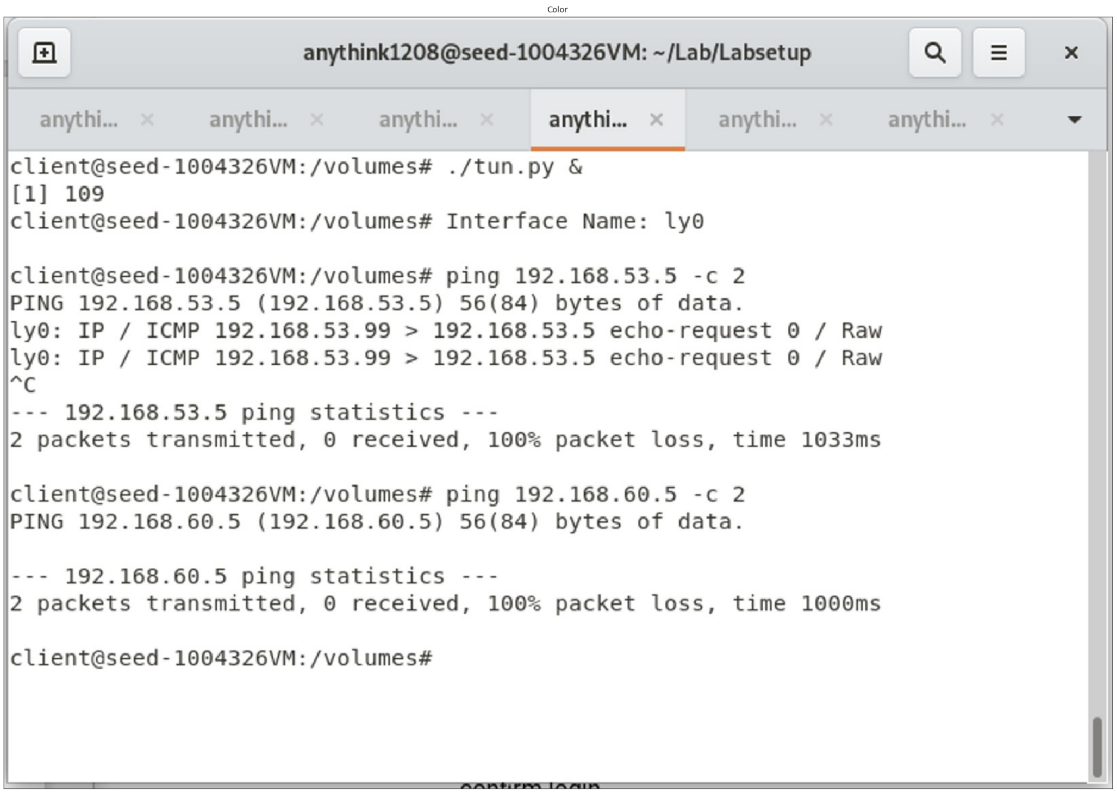
At line 10, ly0 has a newly added IP address 192.168.53.99 and an UNKNOWN state as compared to line 4 tun0 that has no IP address and a DOWN state

Task 2.c: Read from the TUN Interface

```
show_img('Task 2c/print_ip_scapy.png')
```




```
show_img('Task 2c/ping_net.png')
```

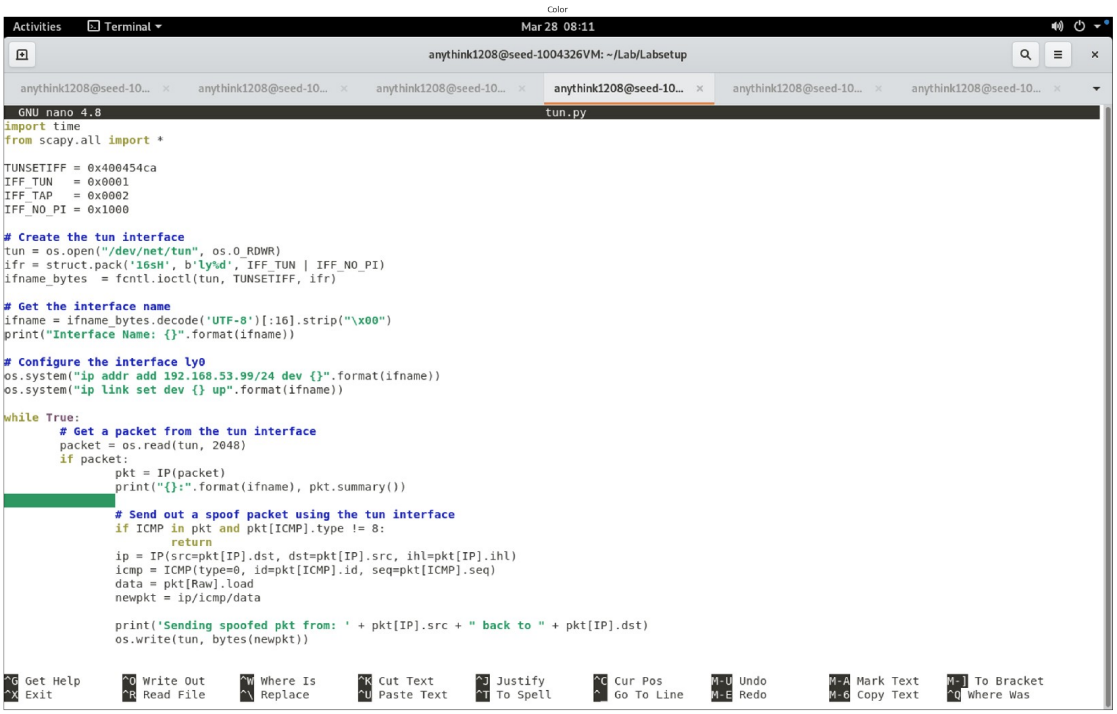


When pinging 192.168.53.5, the packet is directed to the ly0 interface (192.168.53.99) which then captures the packet for inspection. However, 192.168.53.5 does not exist, so there is 100% packet loss.

When pinging 192.168.60.5, there is no interface located in the 192.168.60.0/24 network, so the packet is not captured for inspection. 192.168.60.5 does not exist as well, so a 100% packet loss is expected.

Task 2.d: Write to the TUN Interface

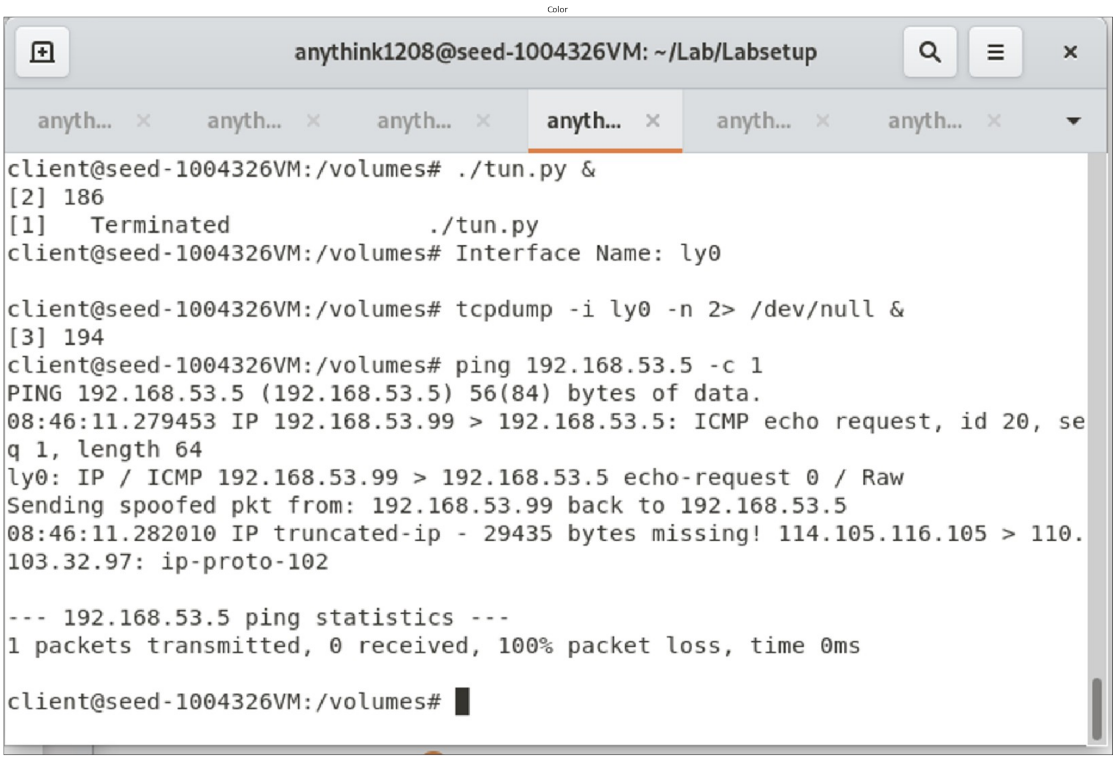
```
show_img('Task 2d/spoof_icmp.png')
```



Check if ICMP packet type is 8 (Echo request), and construct an echo reply and write it to the TUN interface. The code was taken from the sniff_and_spoof.py code from Week 1 Lab.

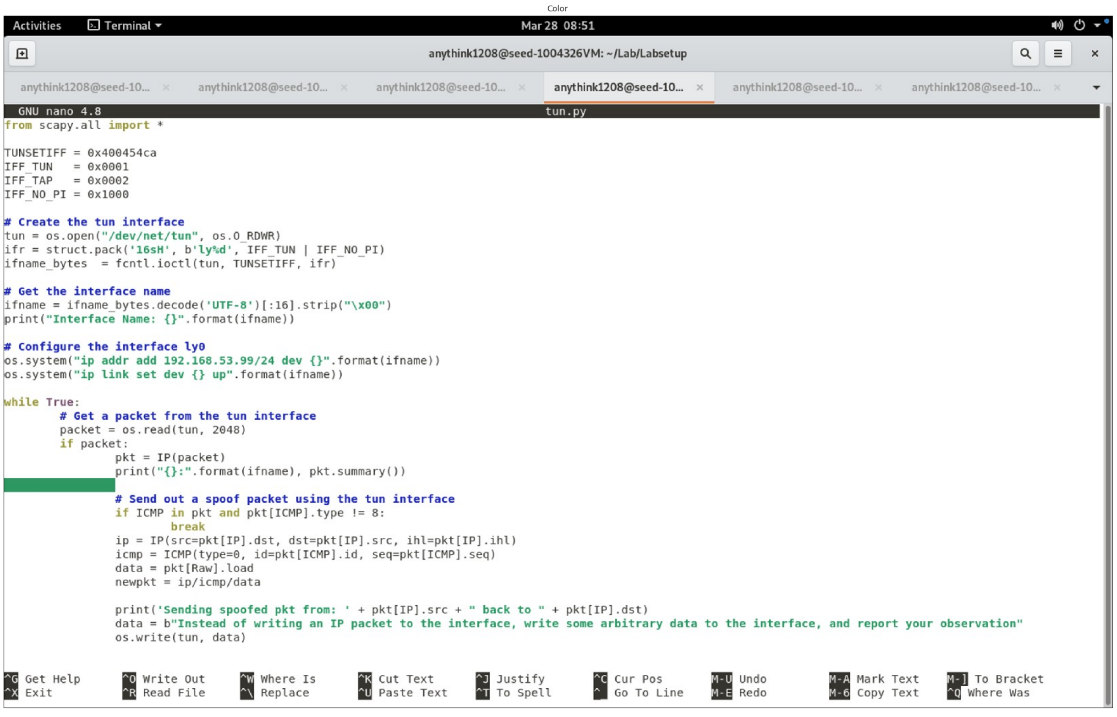
The new packet that was created has its source and destination swapped with a random data being added

```
show_img('Task 2d/spoofed_icmp_out.png')
```

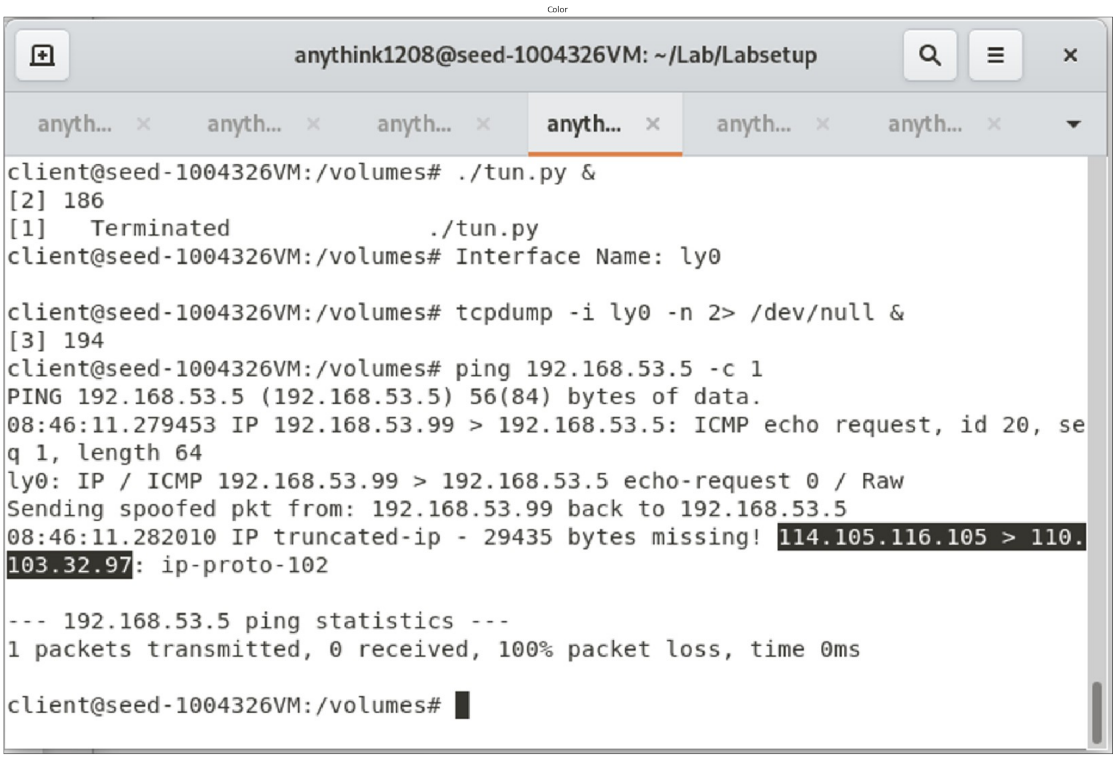


Part 2: write some arbitrary data to the interface

```
show_img('Task 2d/data_changed.png')
```

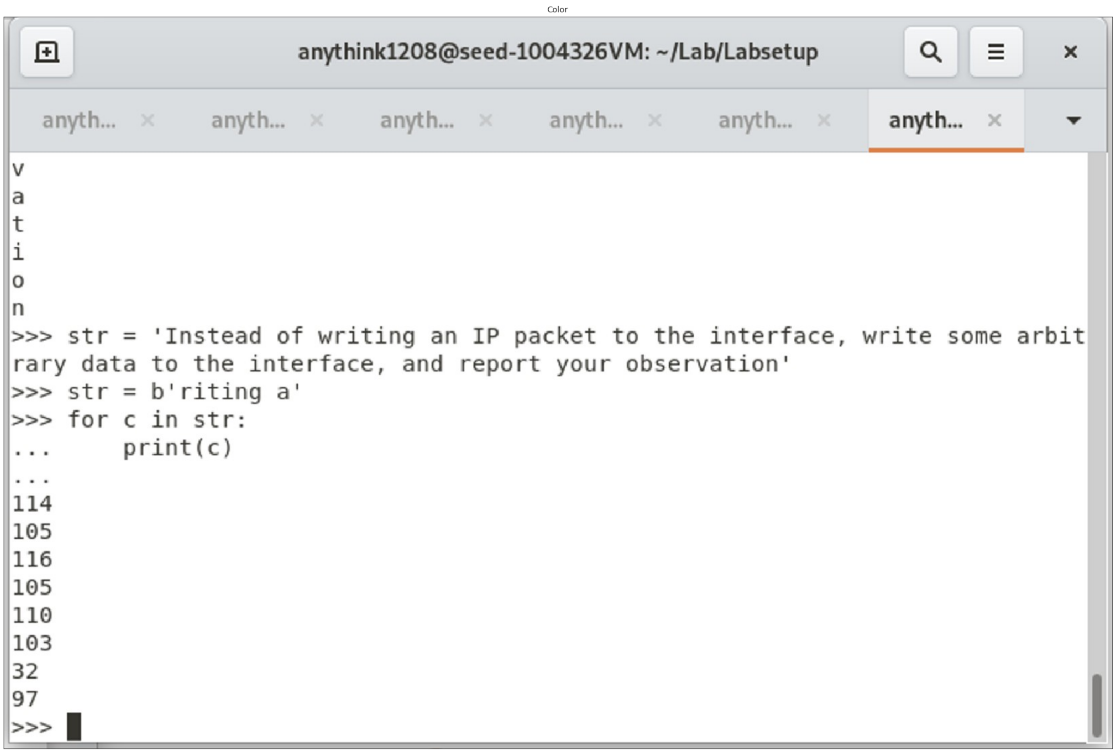



```
show_img('Task 2d/dump_out.png')
```



After changing the data to a binary data, with random text, some random IP address was provided. The IP address provided is actually the binary values of 'riting a', which is seen below:

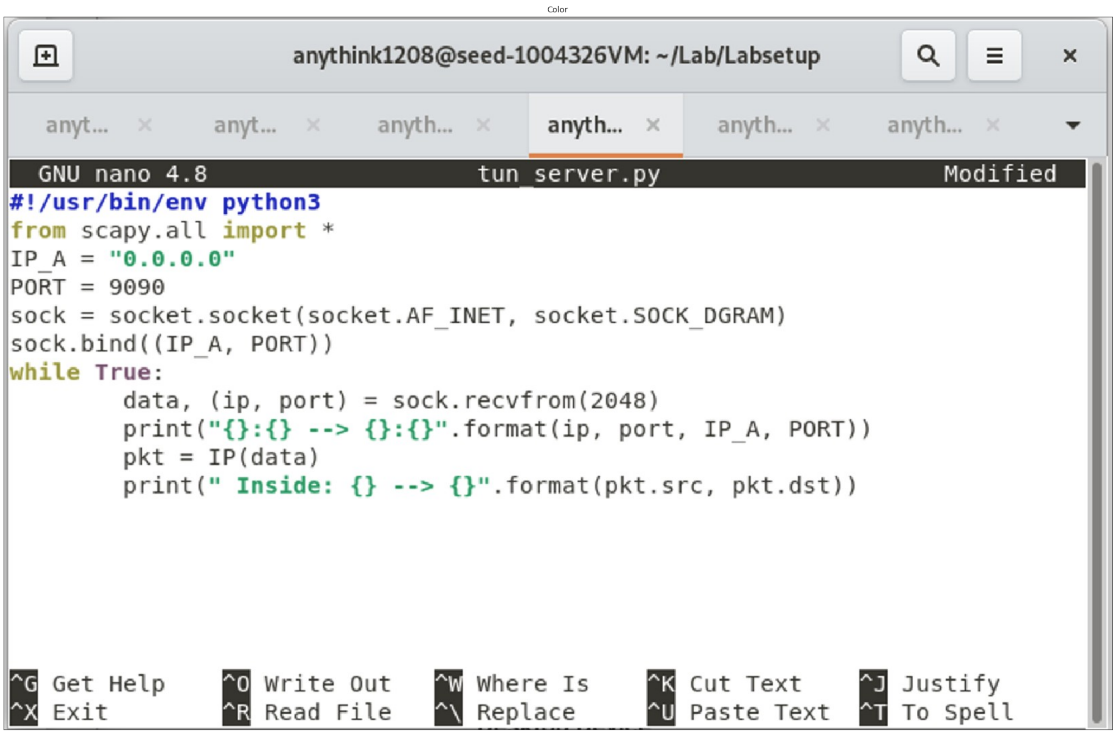
```
show_img('Task 2d/bin_val.png')
```



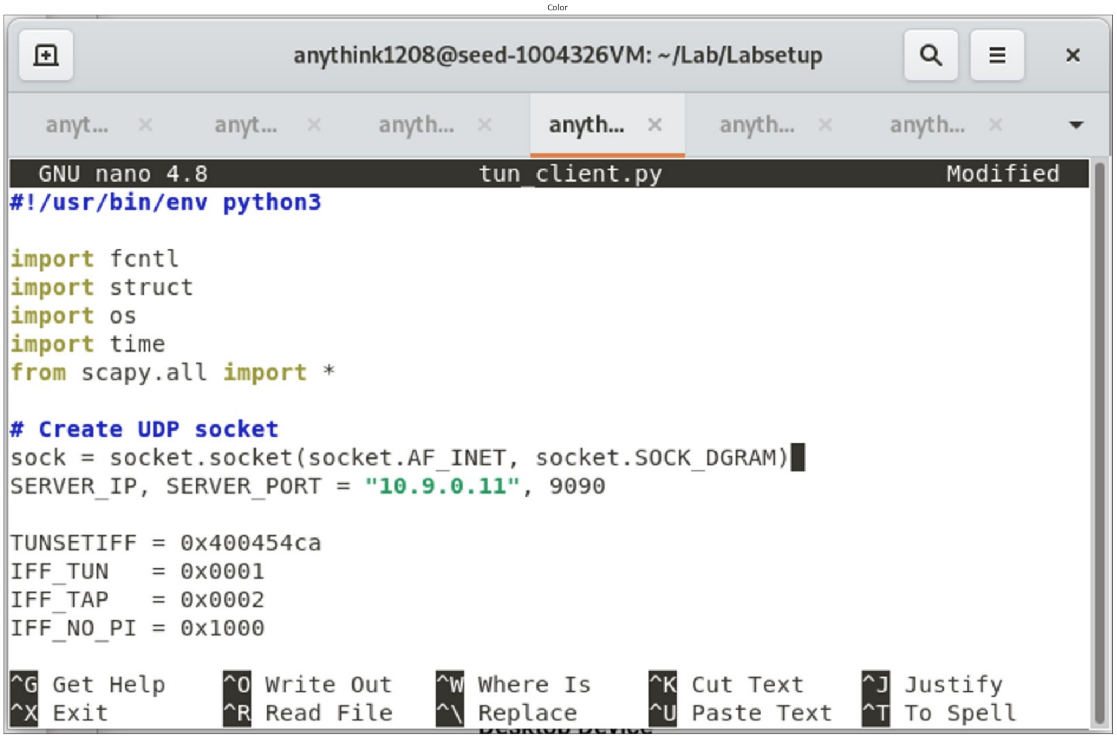
The source data starts from the 13th byte, inclusive of spacing. The 4 bytes (13-16) refers to the source IP and after the next corresponding 4 bytes (17-20) refers to the destination IP

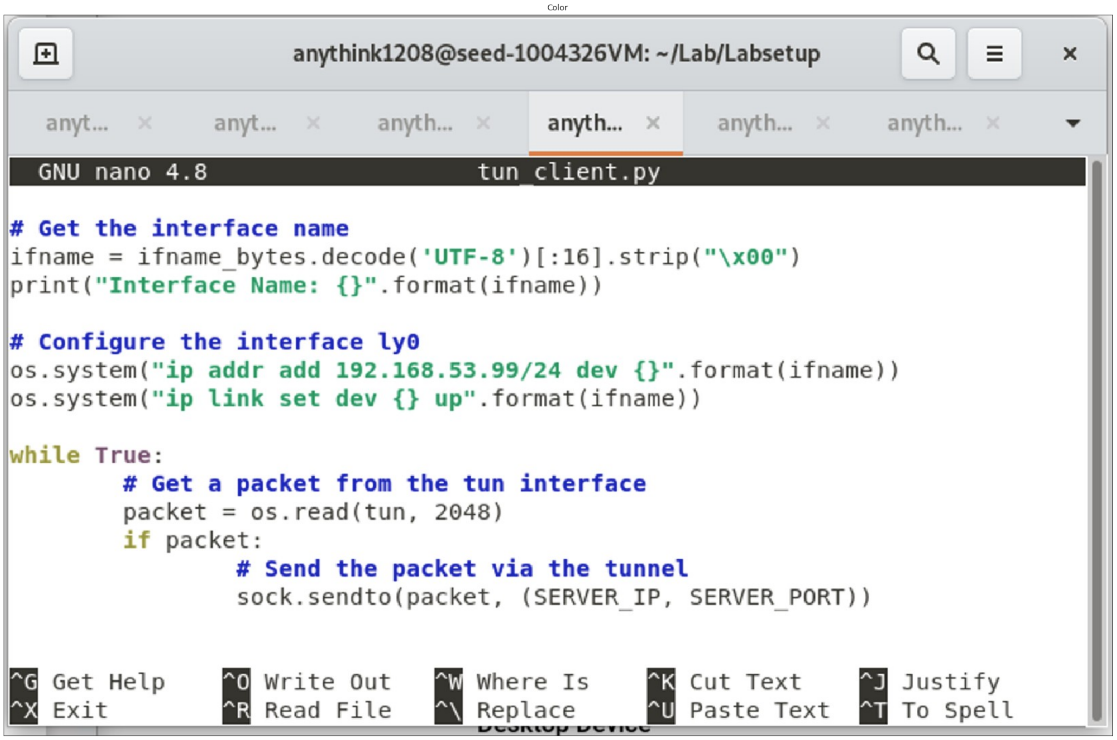
Task 3: Send the IP Packet to VPN Server Through a Tunnel

```
show_img('Task 3/tun_server.png')
```

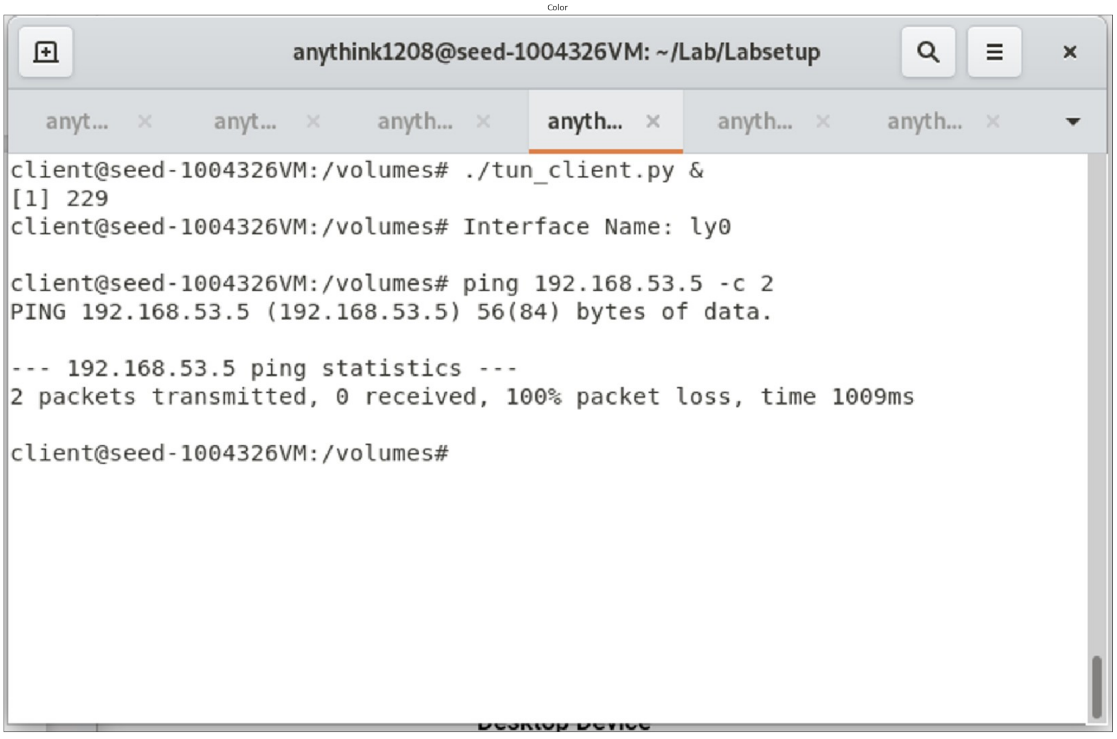



```
show_img('Task 3/tun_client_1.png')
show_img('Task 3/tun_client_2.png')
```





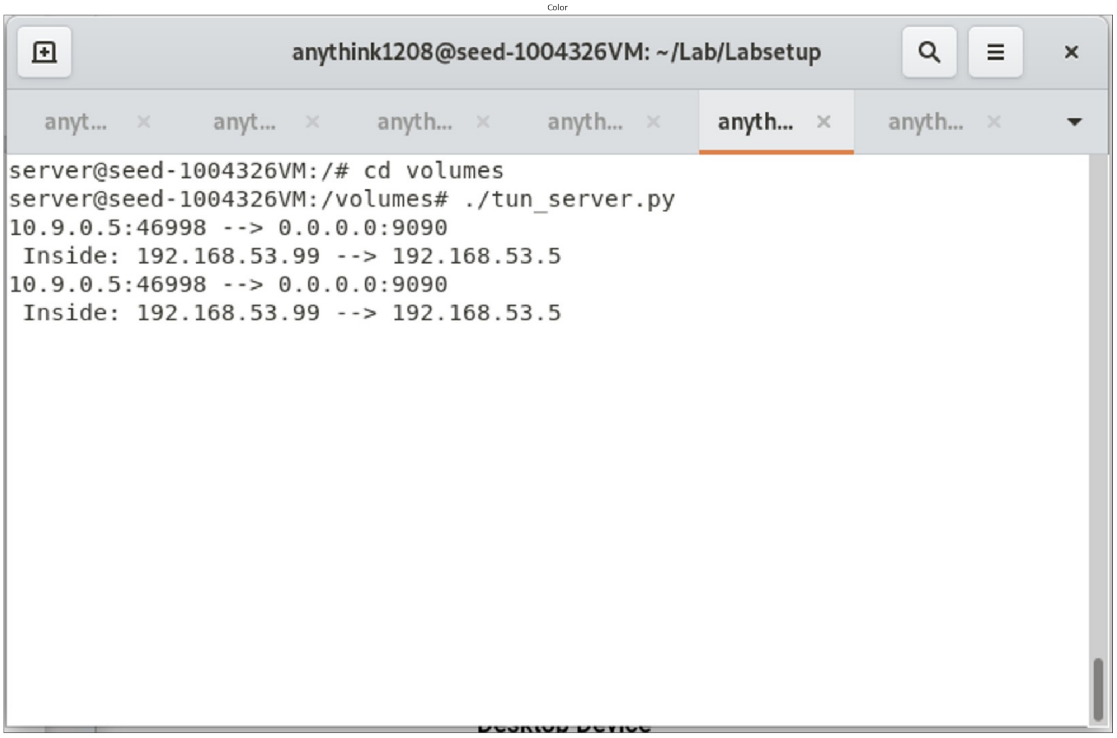

```
show_img('Task 3/ping_net.png')
```



There is no reply to the ping echo request, but the packet is routed to the tun interface ly0 (192.168.53.99), so the tun server is able to capture the packet. The UDP packet is sent

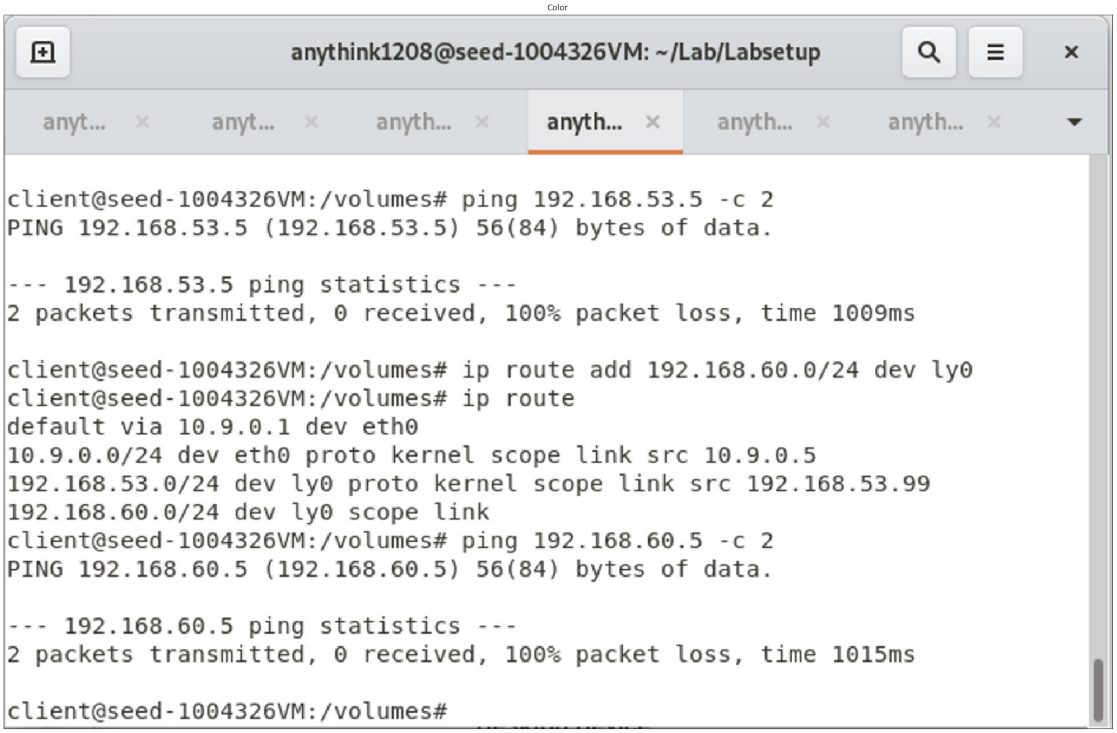
from the client to the server at port 9090. Inside the UDP packet, it shows that the packet is received from ly0 IP address to a host that is non-existent (192.168.53.5)

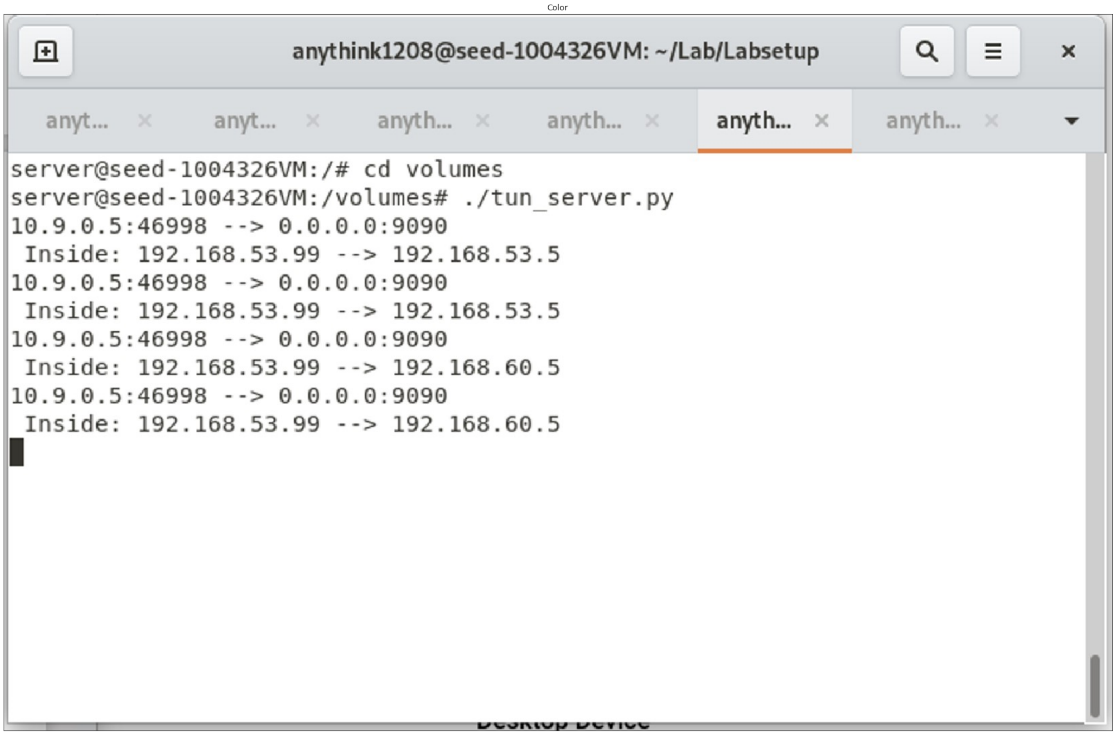
```
show_img('Task 3/tun_server_out.png')
```



After an IP route is added to 192.168.60.0/24 network, the virtual address is able to connect to hosts in the network, so the ping to 192.168.60.5 is routed to the ly0 interface.

```
show_img('Task 3/ip_route.png')
show_img('Task 3/tun_server_out_2.png')
```





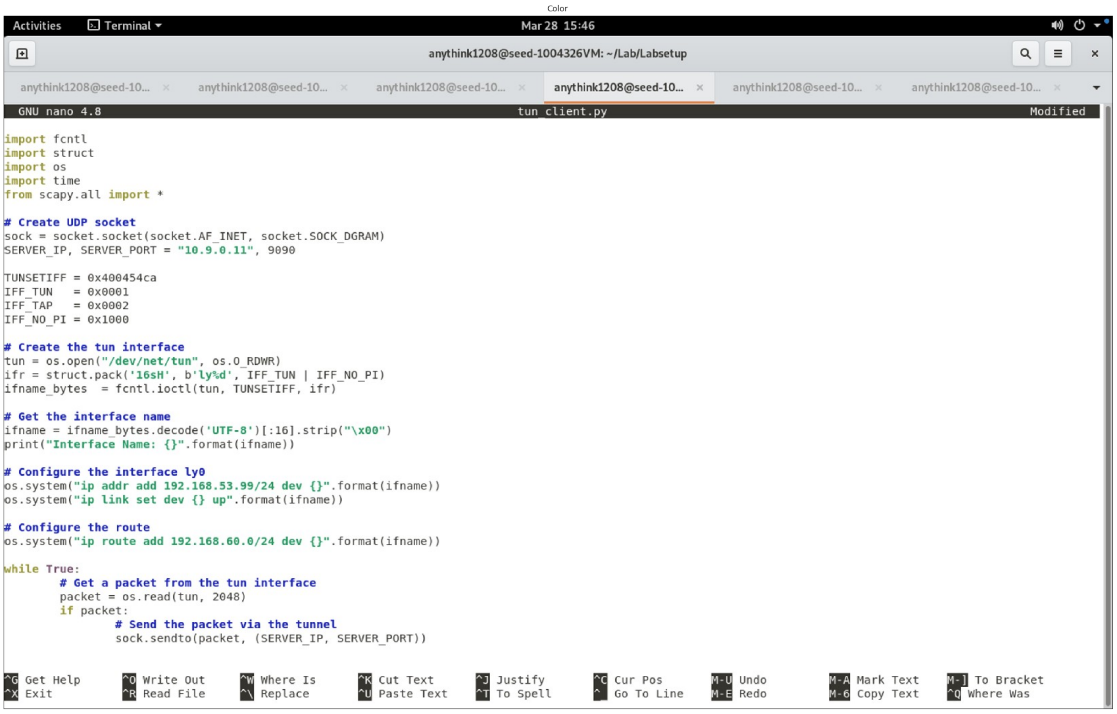
Task 4: Set Up the VPN Server

- Create a TUN interface and configure it
- Get the data from the socket interface; treat the received data as an IP packet

- Write the packet to the TUN interface

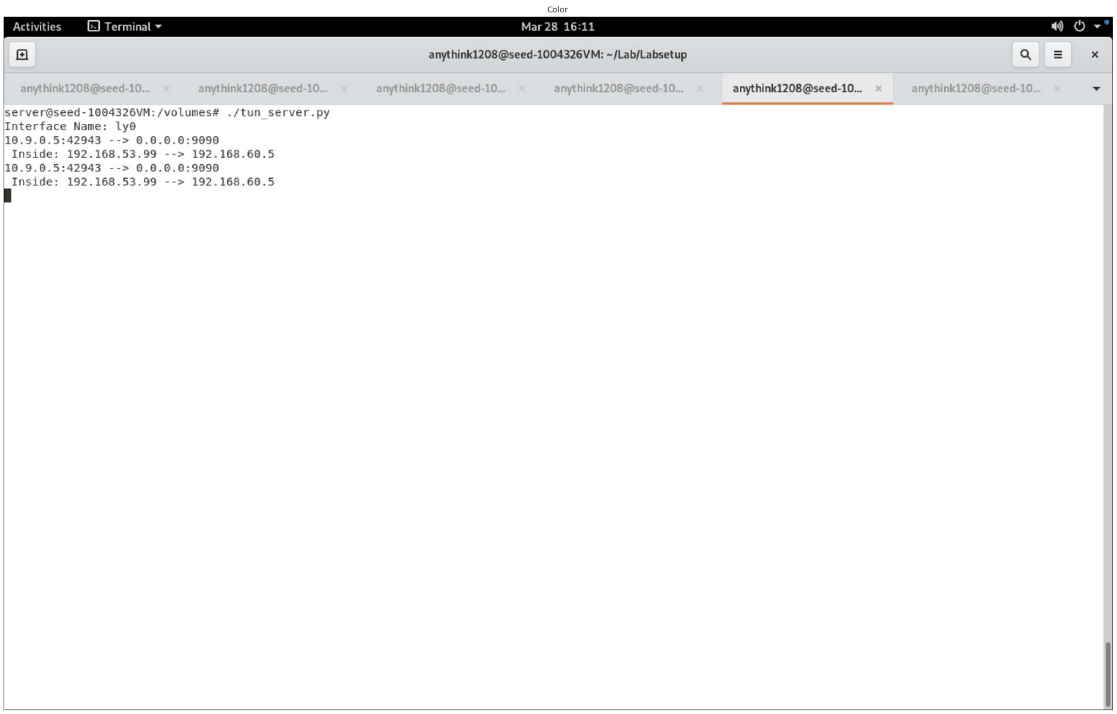
```
show_img('Task 4/config_tun_server.png')
```

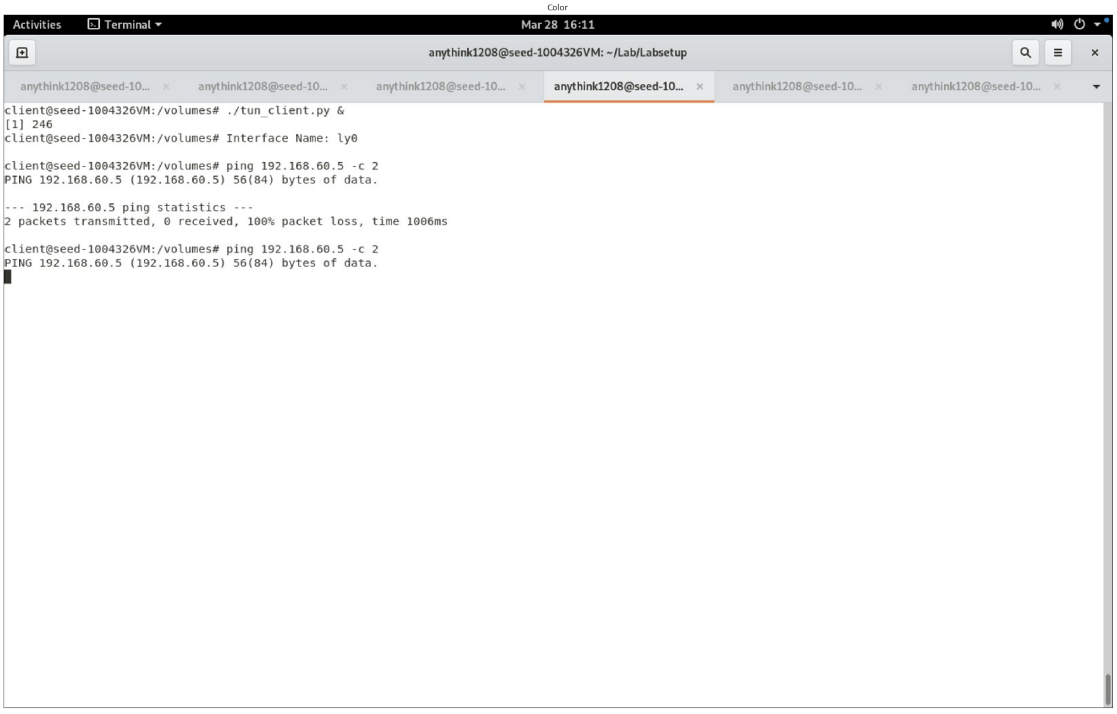

```
show_img('Task 4/config_tun_client.png')
```



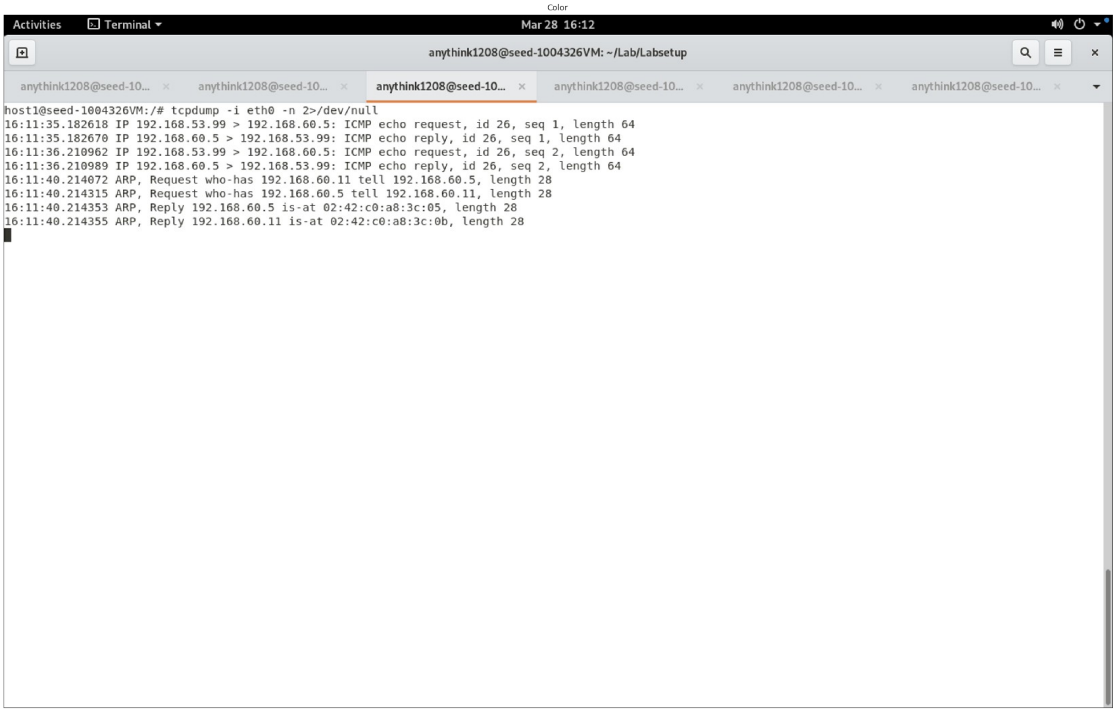

```
show_img('Task 4/tun_server_out.png')
show_img('Task 4/tun_client_out.png')
```

```
show_img('Task 4/dump_out.png')
show_img('Task 4/wireshark_out.png')
```

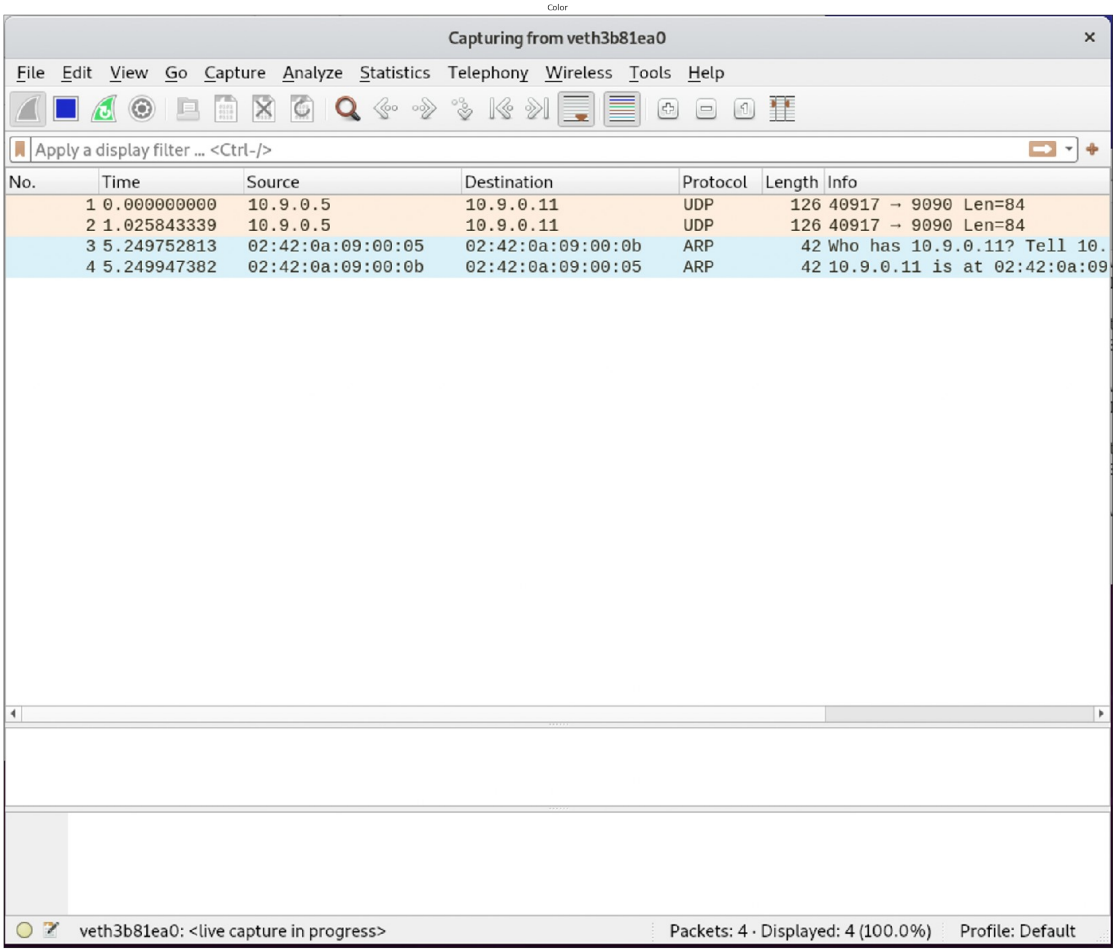




Color


Activities
Terminal
Mar 28 16:12
anythink1208@seed-1004326VM: ~/Lab/Labsetup
anythink1208@seed-10...
anythink1208@seed-10...
anythink1208@seed-10...
anythink1208@seed-10...
anythink1208@seed-10...
anythink1208@seed-10...
host1@seed-1004326VM:/# tcpdump -i eth0 -n 2>/dev/null
16:11:35.182618 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 26, seq 1, length 64
16:11:35.182670 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 26, seq 1, length 64
16:11:36.210962 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 26, seq 2, length 64
16:11:36.210989 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 26, seq 2, length 64
16:11:40.214072 ARP, Request who-has 192.168.60.11 tell 192.168.60.5, length 28
16:11:40.214315 ARP, Request who-has 192.168.60.5 tell 192.168.60.11, length 28
16:11:40.214353 ARP, Reply 192.168.60.5 is-at 02:42:c0:a8:3c:05, length 28
16:11:40.214355 ARP, Reply 192.168.60.11 is-at 02:42:c0:a8:3c:0b, length 28

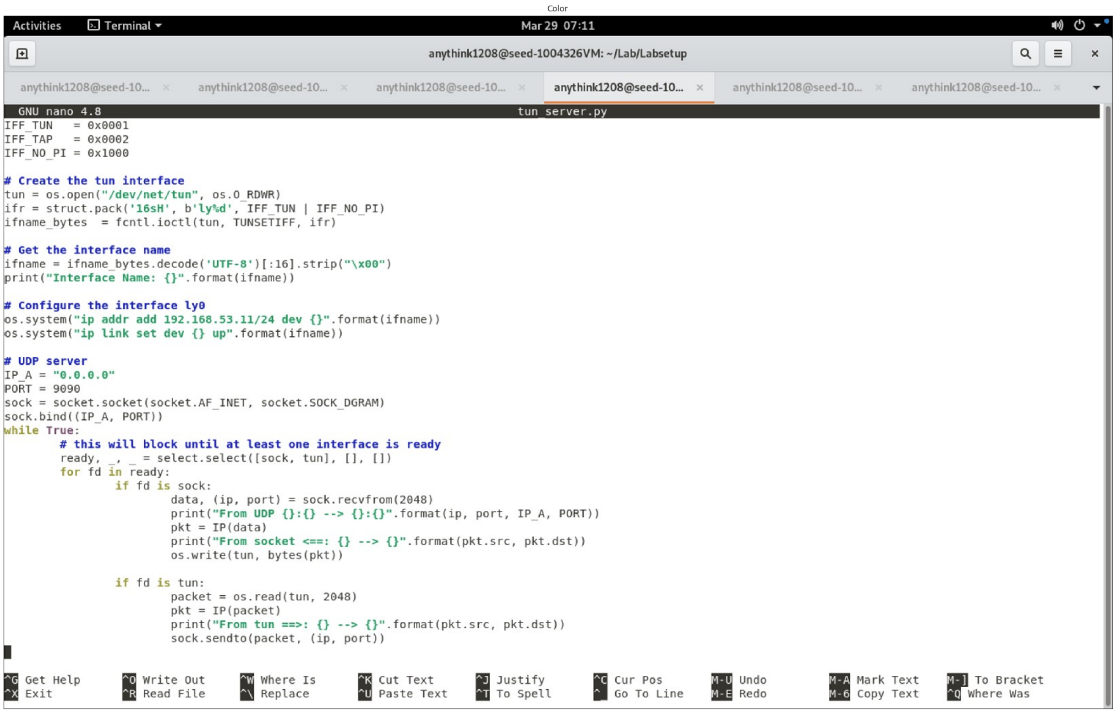
Color


Capturing from veth3b81ea0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
1 0.000000000 10.9.0.5 10.9.0.11 UDP 126 40917 → 9090 Len=84
2 1.025843339 10.9.0.5 10.9.0.11 UDP 126 40917 → 9090 Len=84
3 5.249752813 02:42:0a:09:00:05 02:42:0a:09:00:0b ARP 42 Who has 10.9.0.11? Tell 10.
4 5.249947382 02:42:0a:09:00:0b 02:42:0a:09:00:05 ARP 42 10.9.0.11 is at 02:42:0a:09
veth3b81ea0: <live capture in progress>
Packets: 4 · Displayed: 4 (100.0%)
Profile: Default

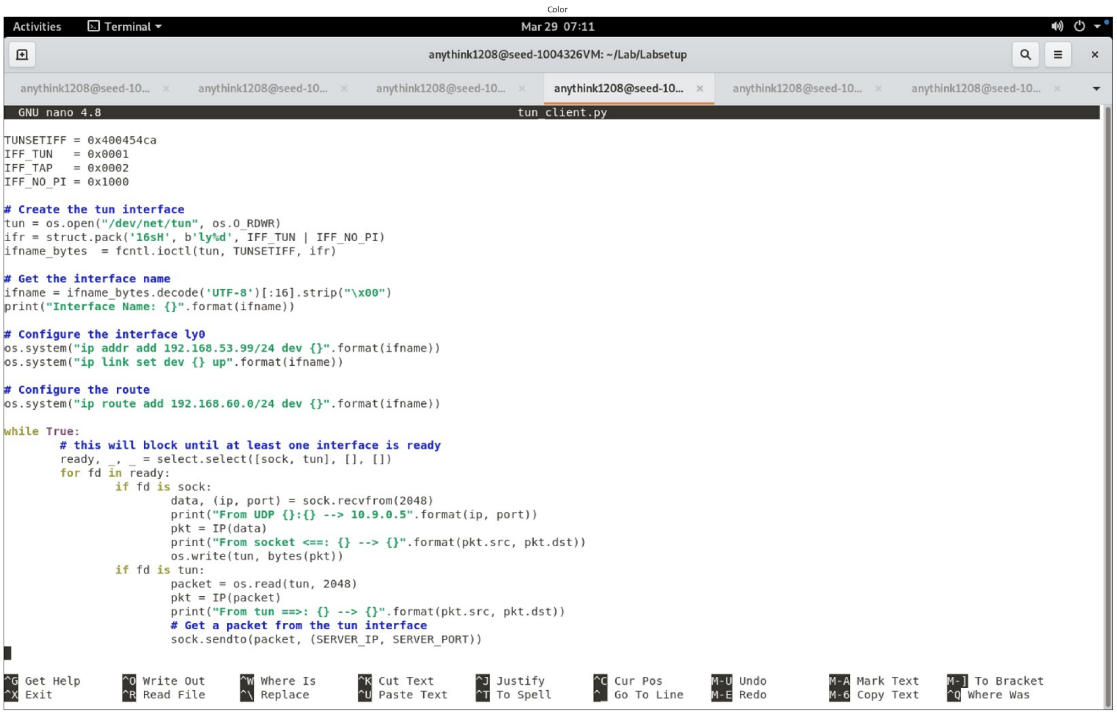
In the tcpdump and wireshark above, we can see that 2 ICMP echo request and replies are made, as expected from the 2 ping packets. The client has successfully sent a packet to the server and host V 192.168.60.5 and sent back.

Task 5: Handling Traffic in Both Directions

```
show_img('Task 5/config_tun_server.png')
```

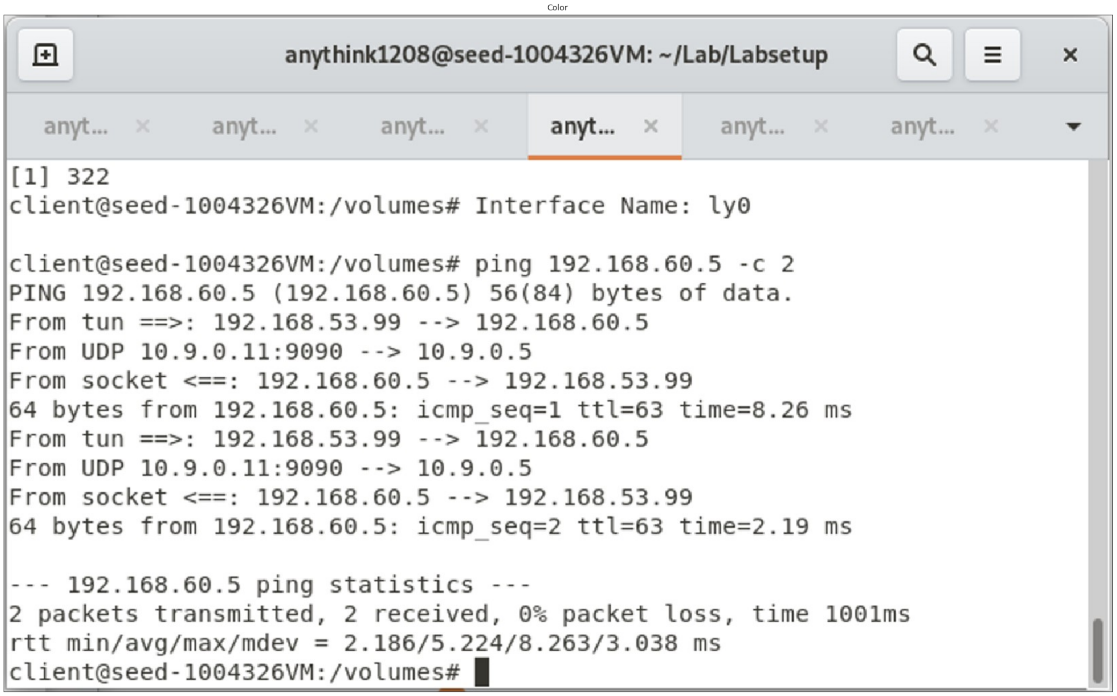


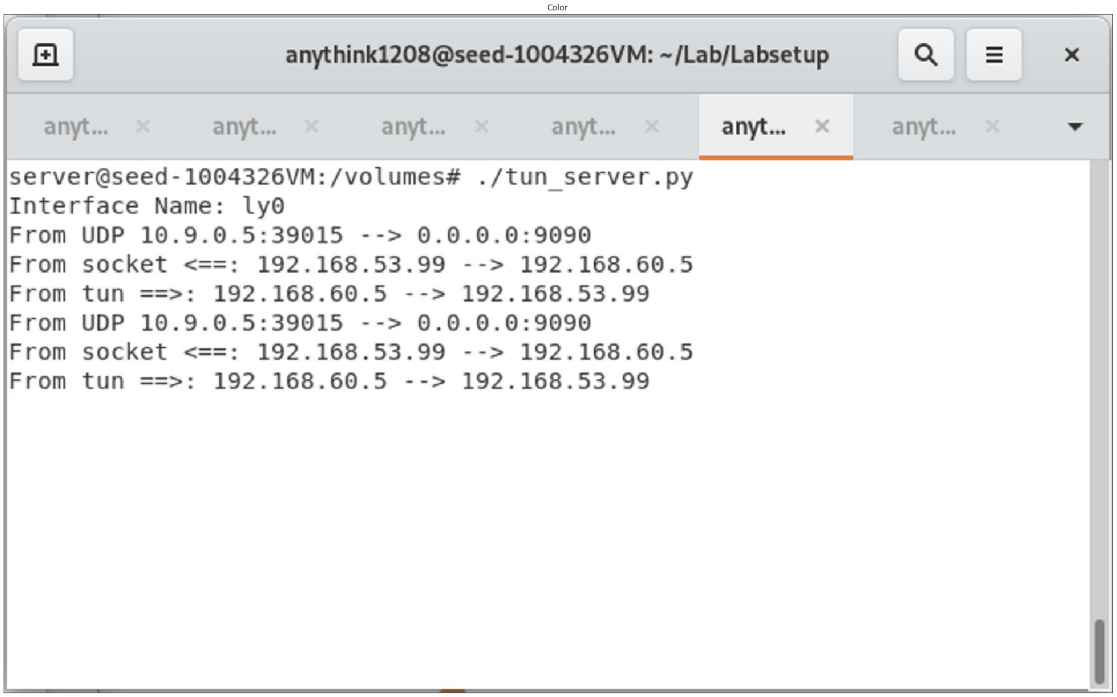

```
show_img('Task 5/config_tun_client.png')
```

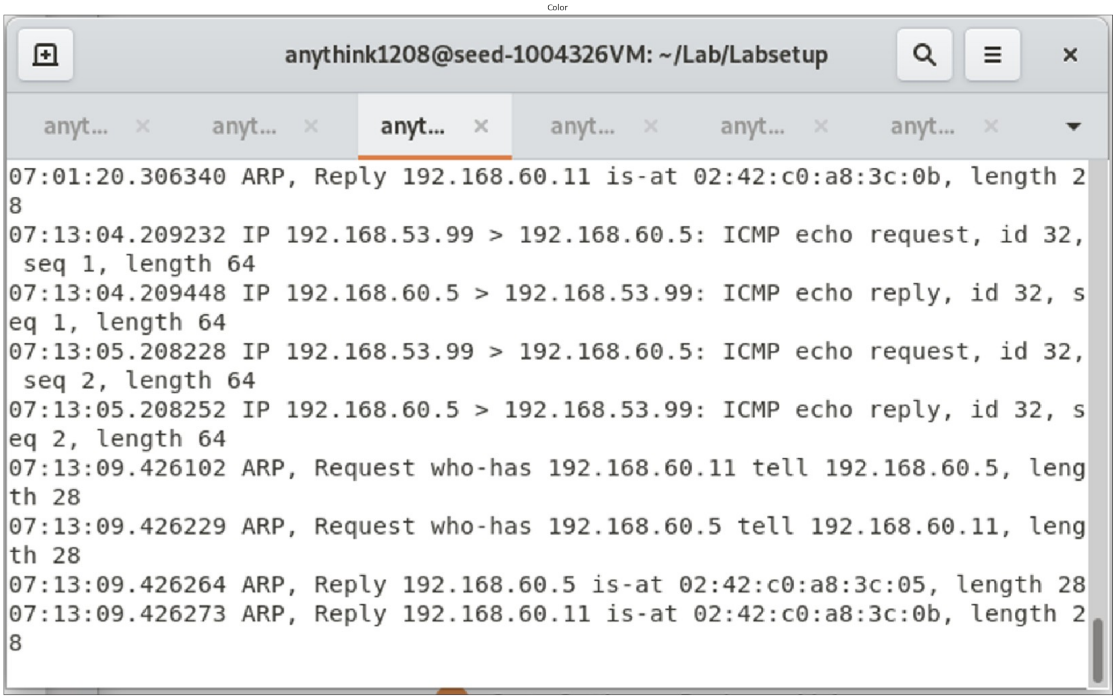


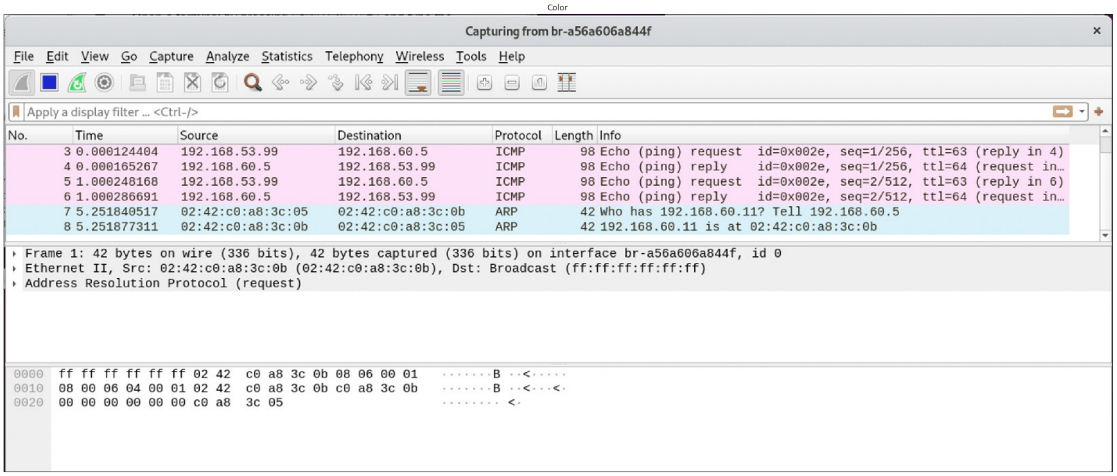
When client or server receives data in the tun interface, it will forward the UDP socket, and when the data is received from the UDP socket, it sends to the tun interface.

```
show_img('Task 5/ping/ping_net.png')
show_img('Task 5/ping/tun_server_out.png')
show_img('Task 5/ping/dump_out.png')
show_img('Task 5/ping/wireshark_out.png')
```



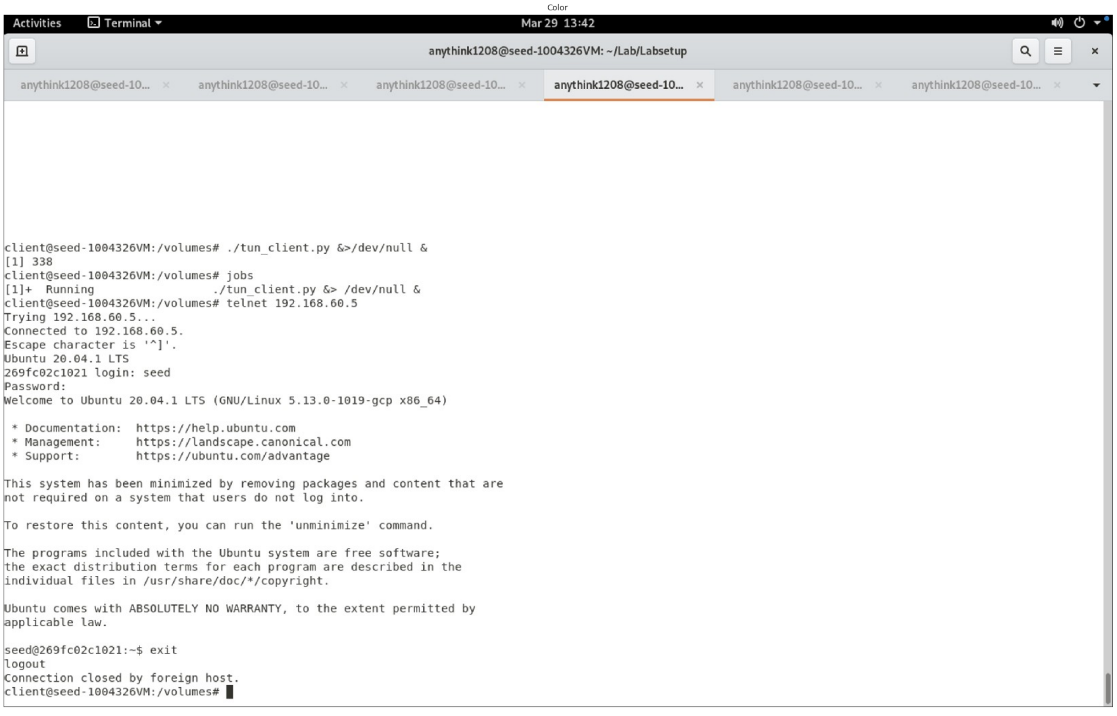

Color
anythink1208@seed-1004326VM: ~/Lab/Labsetup
anyt...
anyt...
anyt...
anyt...
anyt...
anyt...
server@seed-1004326VM:/volumes# ./tun_server.py
Interface Name: ly0
From UDP 10.9.0.5:39015 --> 0.0.0.0:9090
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99
From UDP 10.9.0.5:39015 --> 0.0.0.0:9090
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99


Color
anythink1208@seed-1004326VM: ~/Lab/Labsetup
anyt...
anyt...
anyt...
anyt...
anyt...
anyt...
07:01:20.306340 ARP, Reply 192.168.60.11 is-at 02:42:c0:a8:3c:0b, length 2
8
07:13:04.209232 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 32,
seq 1, length 64
07:13:04.209448 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 32, s
eq 1, length 64
07:13:05.208228 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 32,
seq 2, length 64
07:13:05.208252 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 32, s
eq 2, length 64
07:13:09.426102 ARP, Request who-has 192.168.60.11 tell 192.168.60.5, leng
th 28
07:13:09.426229 ARP, Request who-has 192.168.60.5 tell 192.168.60.11, leng
th 28
07:13:09.426264 ARP, Reply 192.168.60.5 is-at 02:42:c0:a8:3c:05, length 28
07:13:09.426273 ARP, Reply 192.168.60.11 is-at 02:42:c0:a8:3c:0b, length 2
8

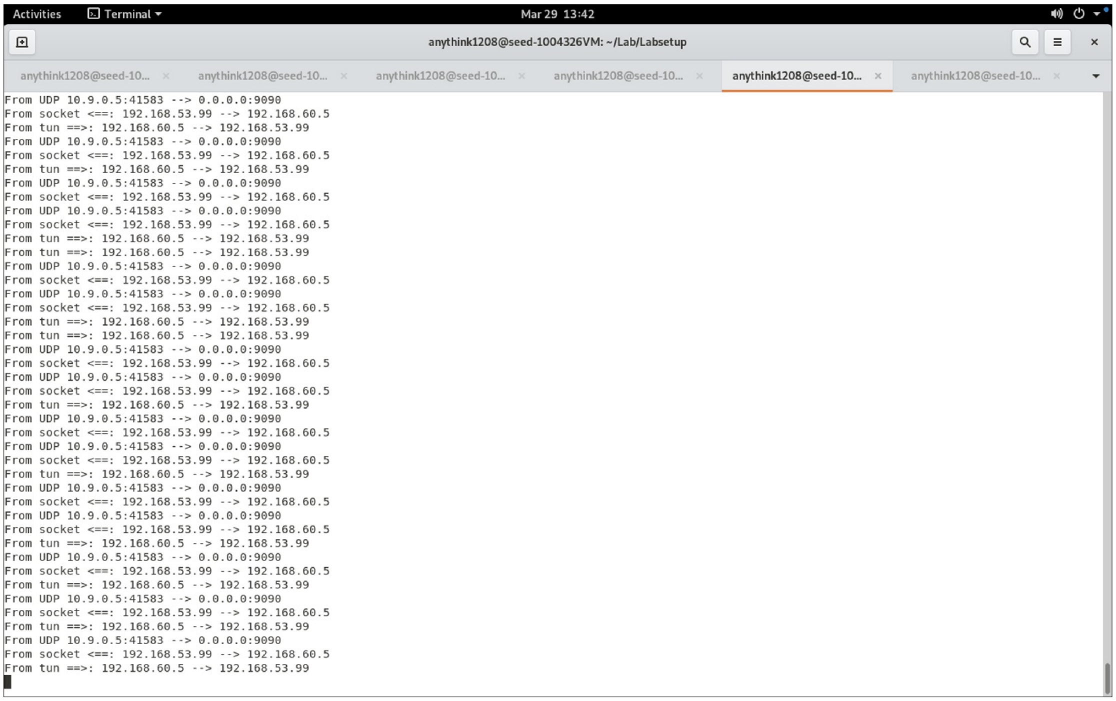


When ping is sent from 10.9.0.5 to the server at 10.9.0.11, port 9090, the server forwards it to 192.168.60.5 from address 192.168.53.99.

```
show_img('Task 5/telnet/telnet_net.png')
show_img('Task 5/telnet/tun_server_out.png')
show_img('Task 5/telnet/dump_out.png')
show_img('Task 5/telnet/wireshark_out.png')
```

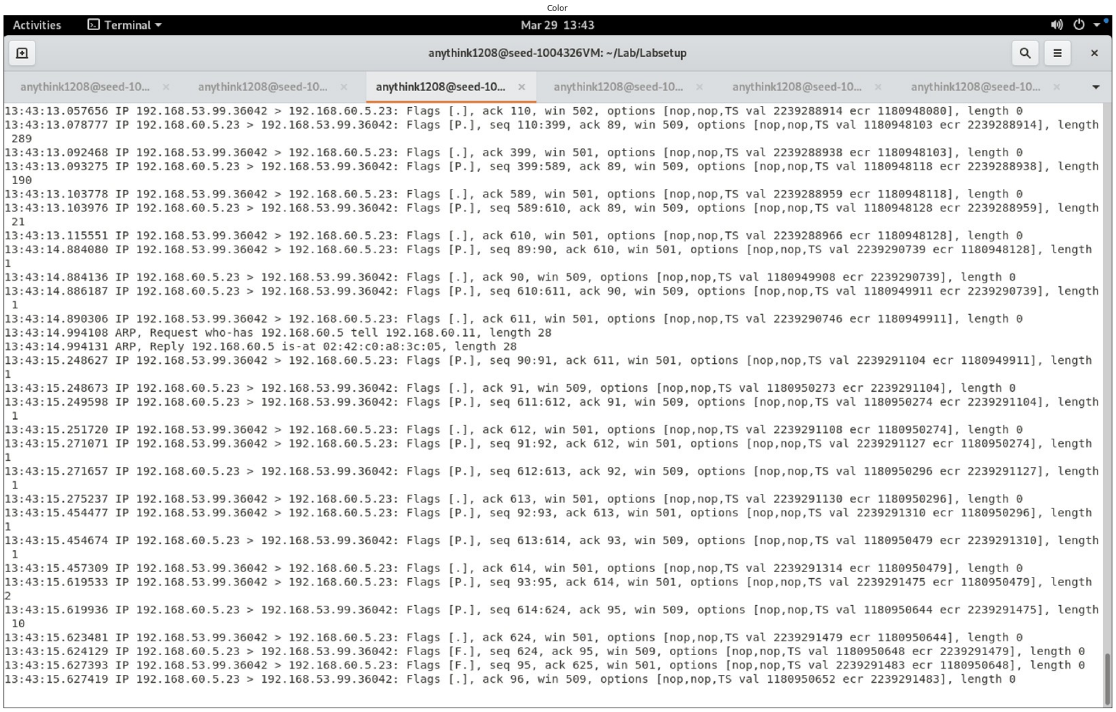


Color

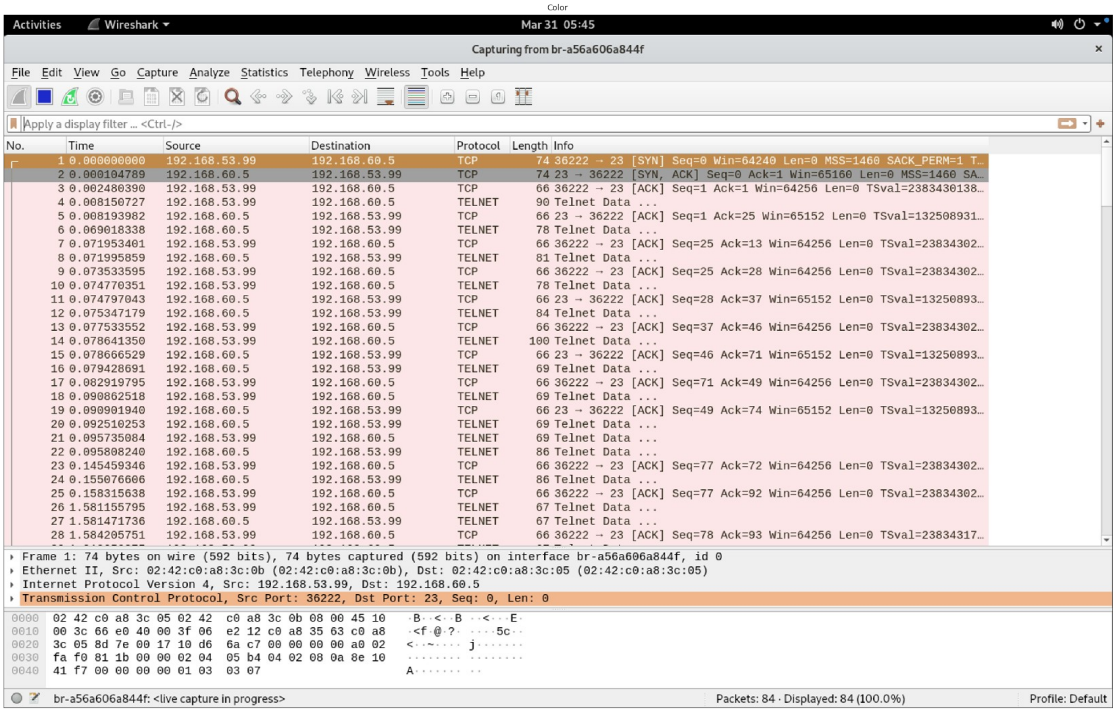


```
Activities   Terminal ▾   Mar 29 13:42
anythink1208@seed-1004326VM: ~/Lab/Labsetup
From UDP 10.9.0.5:41583 --> 0.0.0.0:9090
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99
From UDP 10.9.0.5:41583 --> 0.0.0.0:9090
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99
From UDP 10.9.0.5:41583 --> 0.0.0.0:9090
From socket <==: 192.168.53.99 --> 192.168.60.5
From UDP 10.9.0.5:41583 --> 0.0.0.0:9090
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99
From tun ==>: 192.168.60.5 --> 192.168.53.99
From UDP 10.9.0.5:41583 --> 0.0.0.0:9090
From socket <==: 192.168.53.99 --> 192.168.60.5
From UDP 10.9.0.5:41583 --> 0.0.0.0:9090
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99
From tun ==>: 192.168.60.5 --> 192.168.53.99
From UDP 10.9.0.5:41583 --> 0.0.0.0:9090
From socket <==: 192.168.53.99 --> 192.168.60.5
From UDP 10.9.0.5:41583 --> 0.0.0.0:9090
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99
From UDP 10.9.0.5:41583 --> 0.0.0.0:9090
From socket <==: 192.168.53.99 --> 192.168.60.5
From UDP 10.9.0.5:41583 --> 0.0.0.0:9090
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99
From UDP 10.9.0.5:41583 --> 0.0.0.0:9090
From socket <==: 192.168.53.99 --> 192.168.60.5
From UDP 10.9.0.5:41583 --> 0.0.0.0:9090
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99
From UDP 10.9.0.5:41583 --> 0.0.0.0:9090
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99
From UDP 10.9.0.5:41583 --> 0.0.0.0:9090
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99
From UDP 10.9.0.5:41583 --> 0.0.0.0:9090
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99
```

Color



```
Activities   Terminal ▾   Mar 29 13:43
anythink1208@seed-1004326VM: ~/Lab/Labsetup
13:43:13.057656 IP 192.168.53.99.36042 > 192.168.60.5.23: Flags [.], ack 110, win 502, options [nop,nop,TS val 2239288914 ecr 1180948080], length 0
13:43:13.078777 IP 192.168.60.5.23 > 192.168.53.99.36042: Flags [P.], seq 110:399, ack 89, win 509, options [nop,nop,TS val 1180948103 ecr 2239288914], length
 289
13:43:13.092468 IP 192.168.53.99.36042 > 192.168.60.5.23: Flags [.], ack 399, win 501, options [nop,nop,TS val 2239288938 ecr 1180948103], length 0
13:43:13.093275 IP 192.168.60.5.23 > 192.168.53.99.36042: Flags [P.], seq 399:589, ack 89, win 509, options [nop,nop,TS val 1180948118 ecr 2239288938], length
 190
13:43:13.103778 IP 192.168.53.99.36042 > 192.168.60.5.23: Flags [.], ack 589, win 501, options [nop,nop,TS val 2239288959 ecr 1180948118], length 0
13:43:13.103976 IP 192.168.60.5.23 > 192.168.53.99.36042: Flags [P.], seq 589:610, ack 89, win 509, options [nop,nop,TS val 1180948128 ecr 2239288959], length
 21
13:43:13.115551 IP 192.168.53.99.36042 > 192.168.60.5.23: Flags [.], ack 610, win 501, options [nop,nop,TS val 2239288966 ecr 1180948128], length 0
13:43:14.884080 IP 192.168.53.99.36042 > 192.168.60.5.23: Flags [P.], seq 89:90, ack 610, win 501, options [nop,nop,TS val 2239290739 ecr 1180948128], length
1
13:43:14.884136 IP 192.168.60.5.23 > 192.168.53.99.36042: Flags [.], ack 90, win 509, options [nop,nop,TS val 1180949908 ecr 2239290739], length 0
13:43:14.886187 IP 192.168.60.5.23 > 192.168.53.99.36042: Flags [P.], seq 610:611, ack 90, win 509, options [nop,nop,TS val 1180949911 ecr 2239290739], length
 1
13:43:14.890306 IP 192.168.53.99.36042 > 192.168.60.5.23: Flags [.], ack 611, win 501, options [nop,nop,TS val 2239290746 ecr 1180949911], length 0
13:43:14.994108 ARP, Request who-has 192.168.60.5 tell 192.168.60.11, length 28
13:43:14.994131 ARP, Reply 192.168.60.5 is-at 02:42:c0:a8:3c:05, length 28
13:43:15.248627 IP 192.168.53.99.36042 > 192.168.60.5.23: Flags [P.], seq 90:91, ack 611, win 501, options [nop,nop,TS val 2239291104 ecr 1180949911], length
1
13:43:15.248673 IP 192.168.60.5.23 > 192.168.53.99.36042: Flags [.], ack 91, win 509, options [nop,nop,TS val 1180950273 ecr 2239291104], length 0
13:43:15.249598 IP 192.168.60.5.23 > 192.168.53.99.36042: Flags [P.], seq 611:612, ack 91, win 509, options [nop,nop,TS val 1180950274 ecr 2239291104], length
 1
13:43:15.251720 IP 192.168.53.99.36042 > 192.168.60.5.23: Flags [.], ack 612, win 501, options [nop,nop,TS val 2239291108 ecr 1180950274], length 0
13:43:15.271071 IP 192.168.53.99.36042 > 192.168.60.5.23: Flags [P.], seq 91:92, ack 612, win 501, options [nop,nop,TS val 2239291127 ecr 1180950274], length
1
13:43:15.271657 IP 192.168.60.5.23 > 192.168.53.99.36042: Flags [P.], seq 612:613, ack 92, win 509, options [nop,nop,TS val 1180950296 ecr 2239291127], length
 1
13:43:15.275237 IP 192.168.53.99.36042 > 192.168.60.5.23: Flags [.], ack 613, win 501, options [nop,nop,TS val 2239291130 ecr 1180950296], length 0
13:43:15.454477 IP 192.168.53.99.36042 > 192.168.60.5.23: Flags [P.], seq 92:93, ack 613, win 501, options [nop,nop,TS val 2239291310 ecr 1180950296], length
1
13:43:15.454674 IP 192.168.60.5.23 > 192.168.53.99.36042: Flags [P.], seq 613:614, ack 93, win 509, options [nop,nop,TS val 1180950479 ecr 2239291310], length
 1
13:43:15.457309 IP 192.168.53.99.36042 > 192.168.60.5.23: Flags [.], ack 614, win 501, options [nop,nop,TS val 2239291314 ecr 1180950479], length 0
13:43:15.619533 IP 192.168.53.99.36042 > 192.168.60.5.23: Flags [P.], seq 93:95, ack 614, win 501, options [nop,nop,TS val 2239291475 ecr 1180950479], length
2
13:43:15.619936 IP 192.168.60.5.23 > 192.168.53.99.36042: Flags [P.], seq 614:624, ack 95, win 509, options [nop,nop,TS val 1180950644 ecr 2239291475], length
 10
13:43:15.623481 IP 192.168.53.99.36042 > 192.168.60.5.23: Flags [.], ack 624, win 501, options [nop,nop,TS val 2239291479 ecr 1180950644], length 0
13:43:15.624129 IP 192.168.60.5.23 > 192.168.53.99.36042: Flags [F.], seq 624, ack 95, win 509, options [nop,nop,TS val 1180950648 ecr 2239291479], length 0
13:43:15.627393 IP 192.168.53.99.36042 > 192.168.60.5.23: Flags [F.], seq 95, ack 625, win 501, options [nop,nop,TS val 2239291483 ecr 1180950648], length 0
13:43:15.627419 IP 192.168.60.5.23 > 192.168.53.99.36042: Flags [.], ack 96, win 509, options [nop,nop,TS val 1180950652 ecr 2239291483], length 0
```

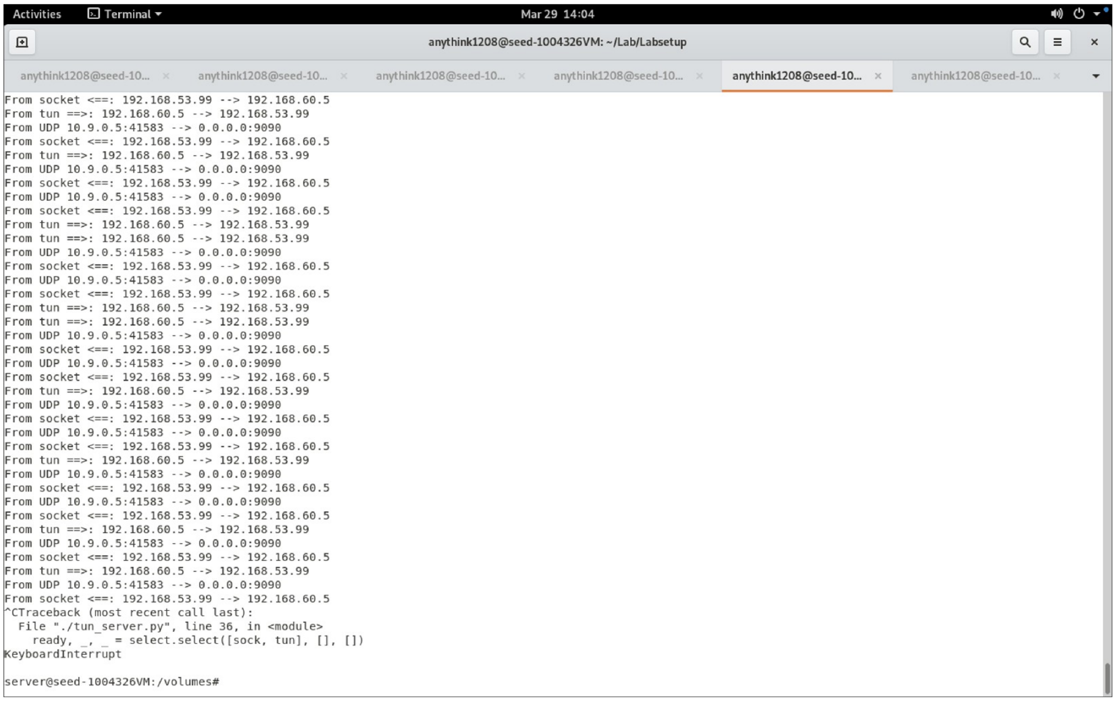


When telnet is sent from 10.9.0.5 to the server at 10.9.0.11, port 36042, the server forwards it to 192.168.60.5 from address 192.168.53.99.

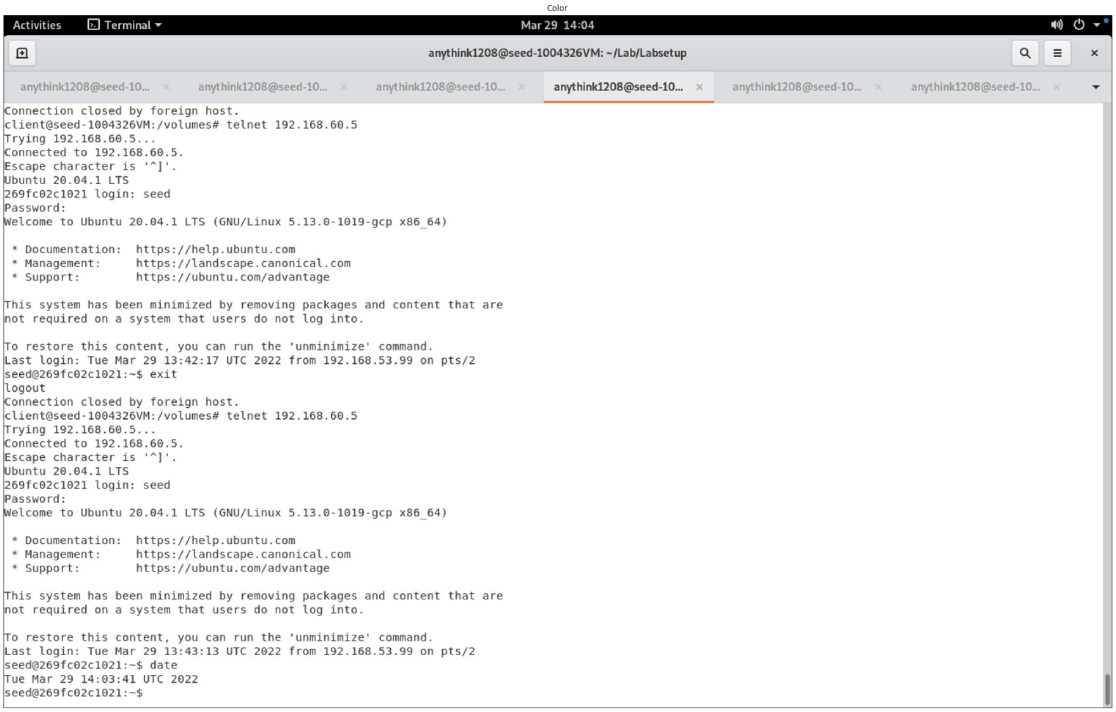
Task 6: Tunnel-Breaking Experiment

```
show_img('Task 6/break_server.png')
show_img('Task 6/telnet_hang.png')
```

Color

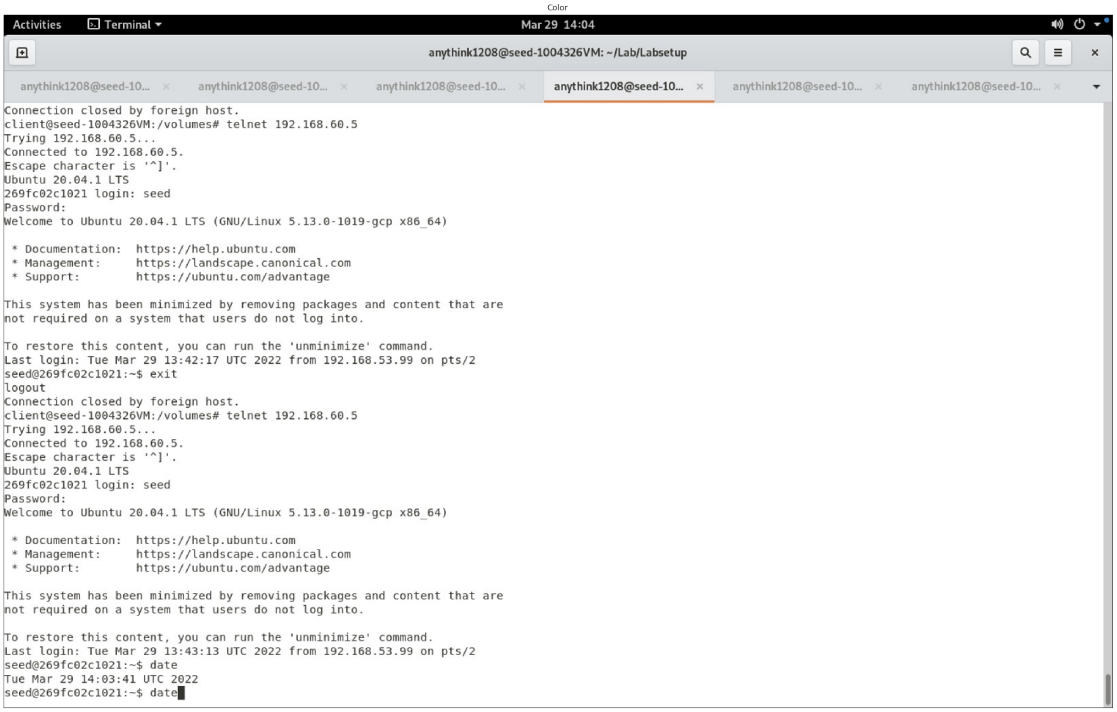


Color



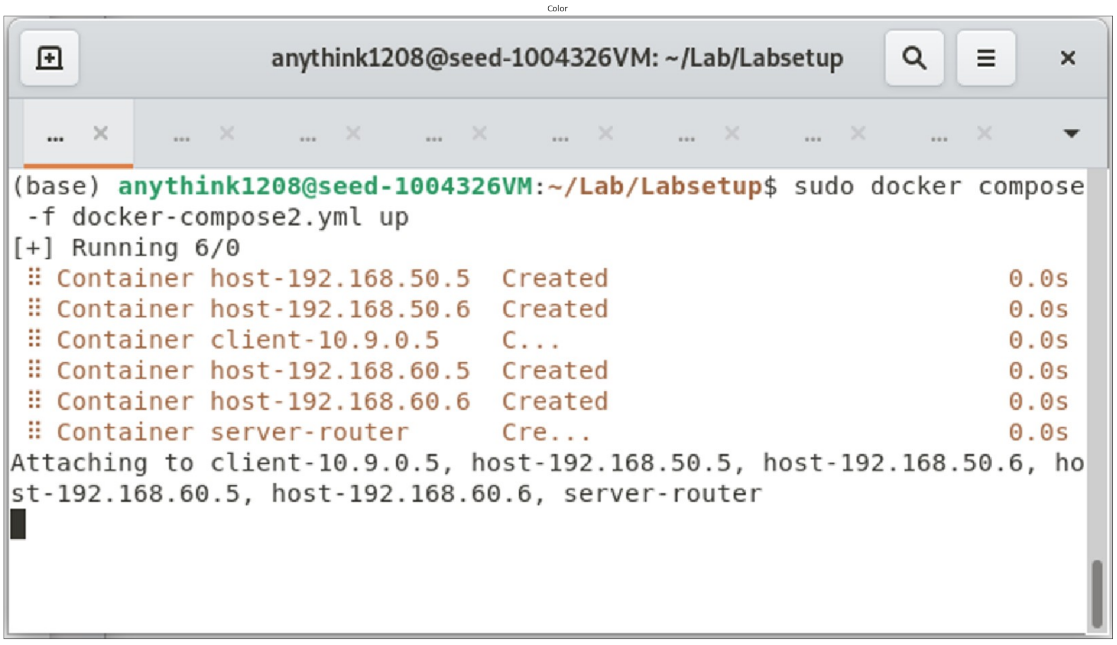
The shell was not responsive when the server is stopped. However, when the server resumes, the shell reconnects and becomes responsive

```
show_img('Task 6/telnet_resume.png')
```

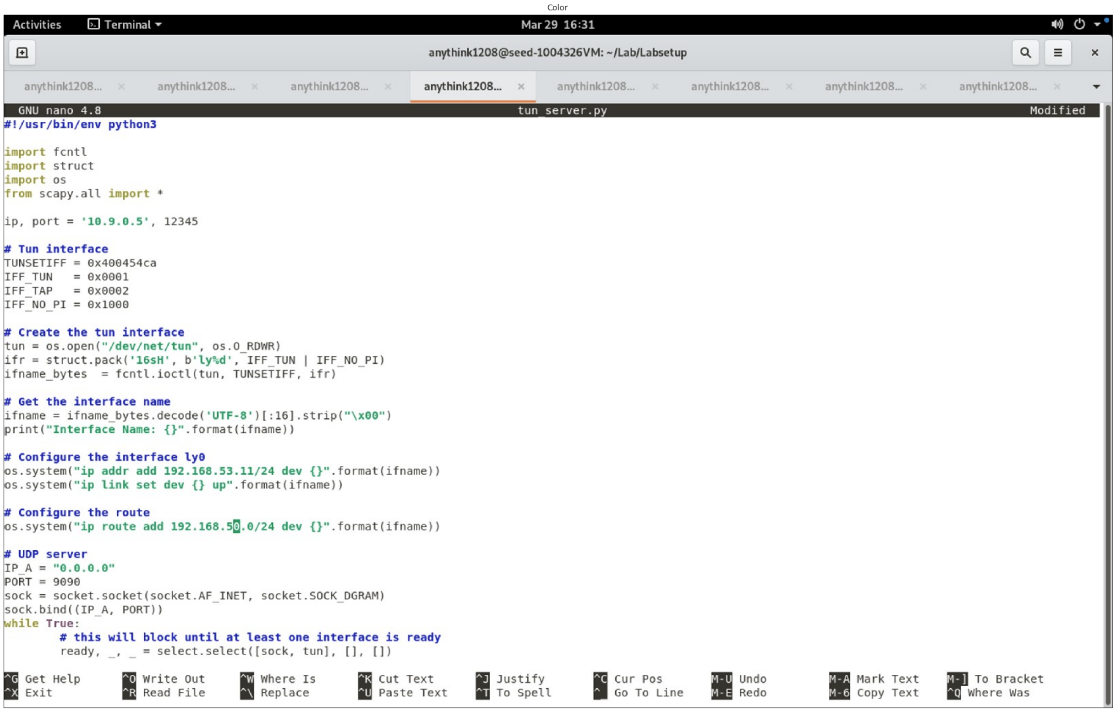


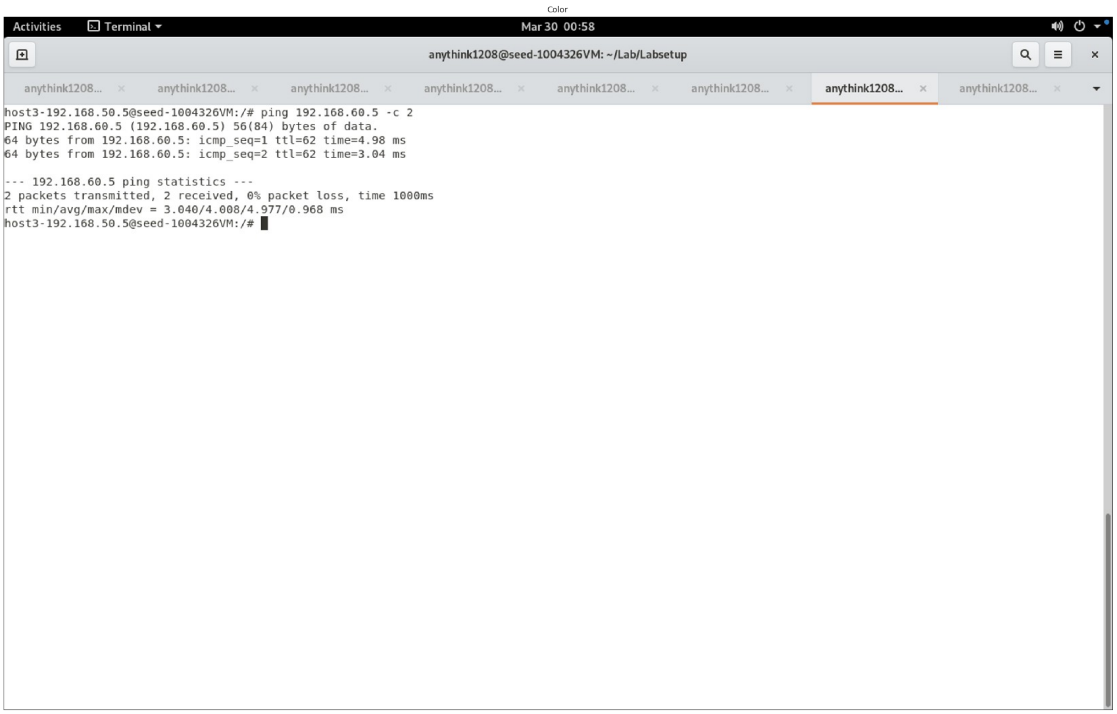
Task 7: Routing Experiment on Host V

```
show_img('setup/docker_2_setup.png')
```




```
show_img('Task 7/add_route.png')
show_img('Task 7/ping_net.png')
```

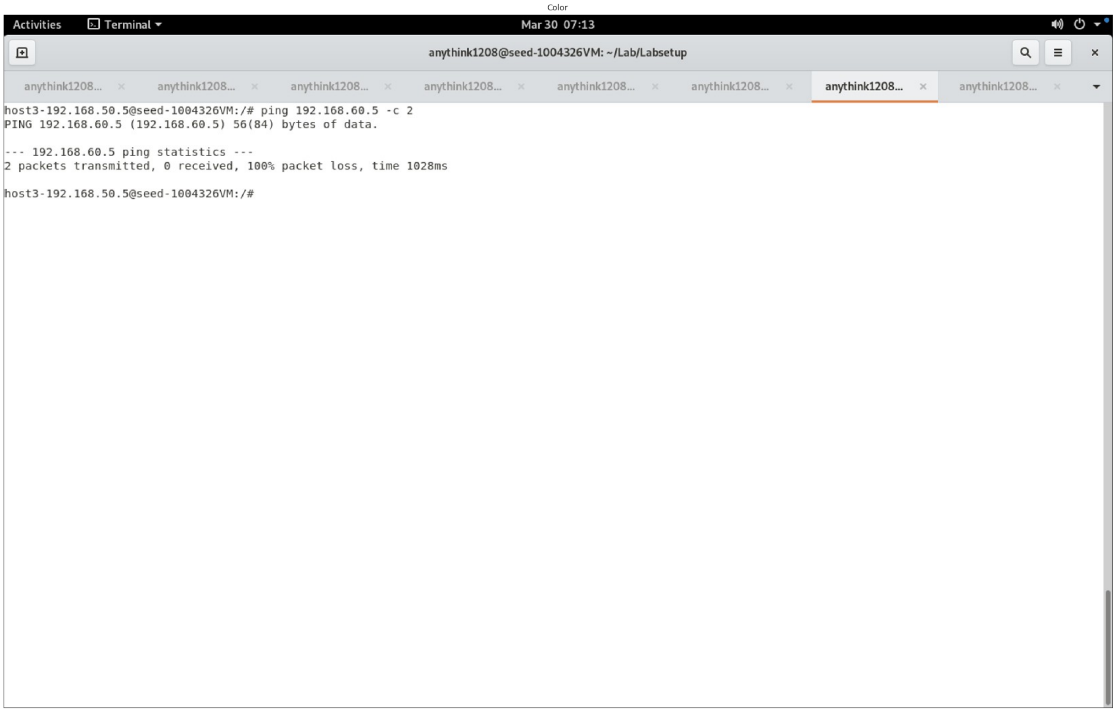


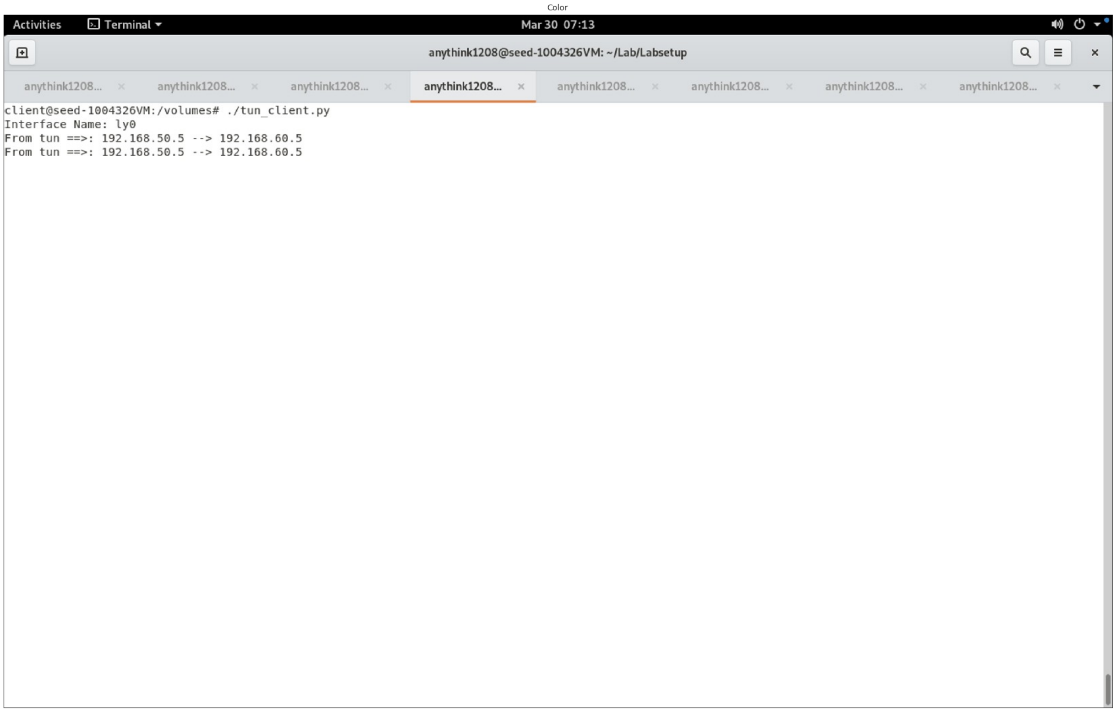


Task 8: Experiment with the TUN IP Address

On Host U, change its TUN interface's IP address to 192.168.30.99

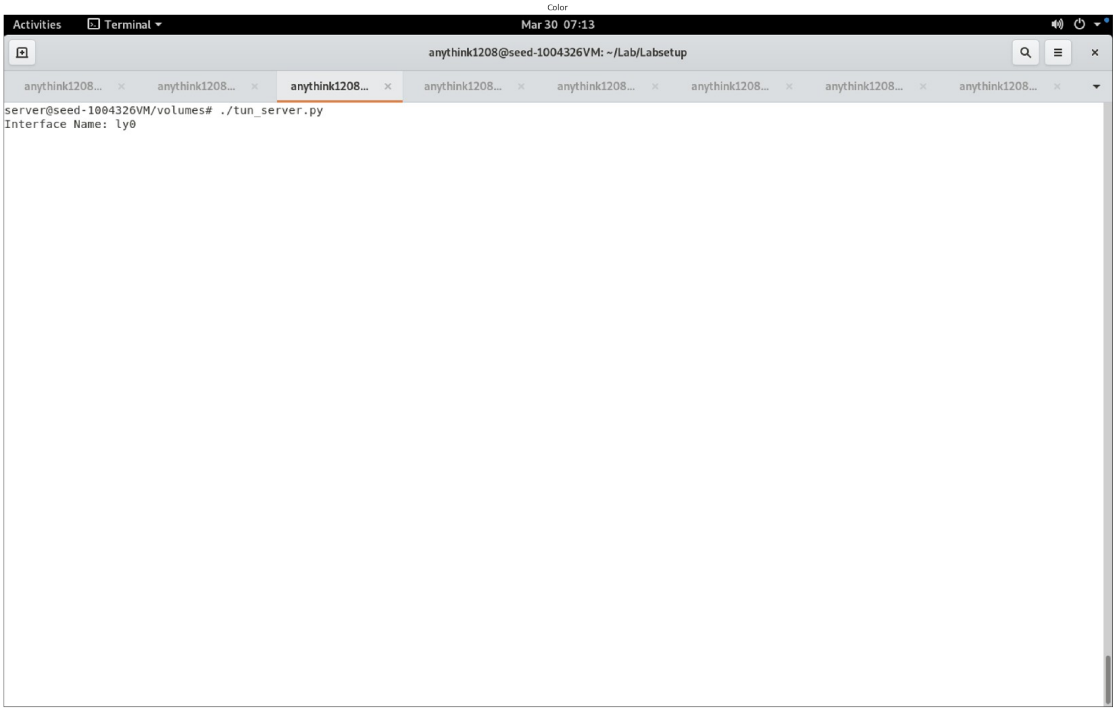
```
show_img('Task 8/change_ip.png')
show_img('Task 8/ping_net.png')
show_img('Task 8/tun_client_out.png')
```





The packets are dropped at 192.168.30.0 as there is no corresponding interface and route at 192.168.30.0/24 network. Due to the lack of time, I used tcpdump to check the packet trace instead.

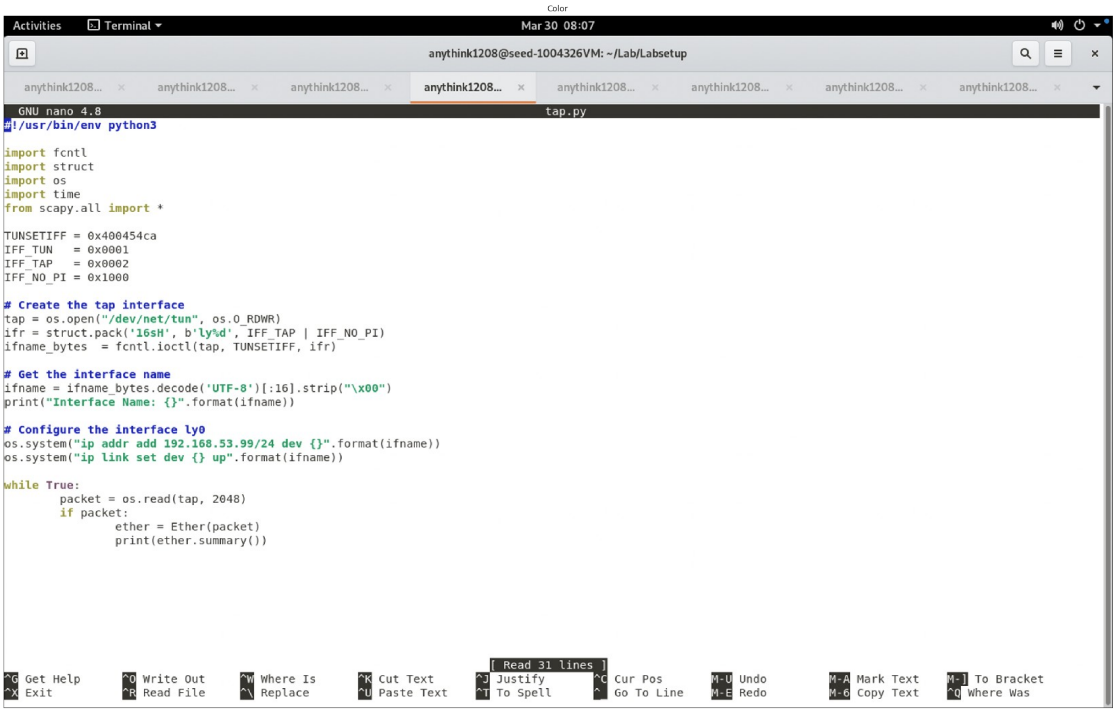
```
show_img('Task 8/tun_server_out.png')
```

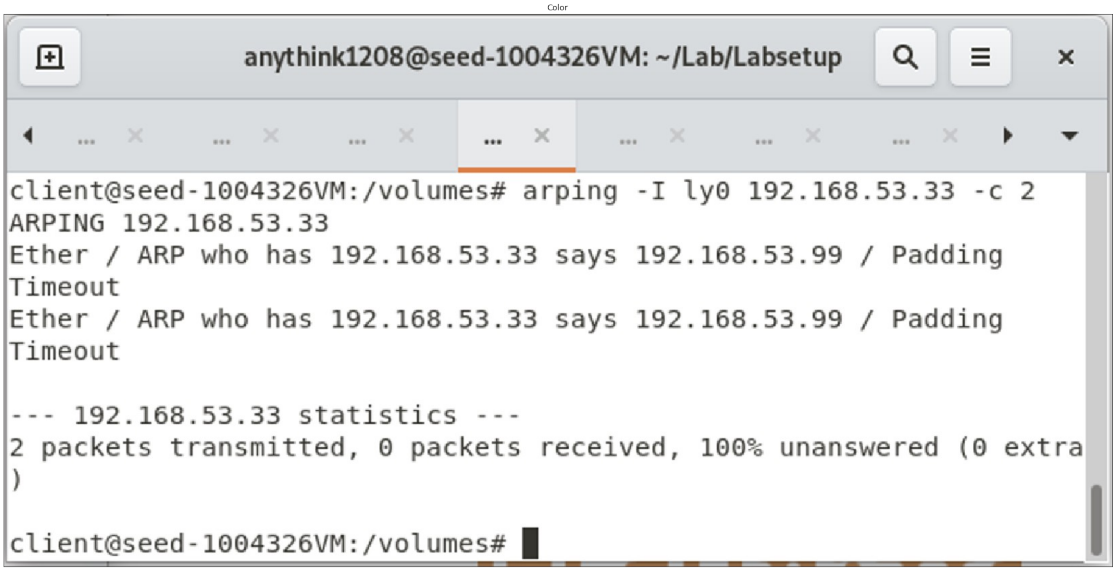


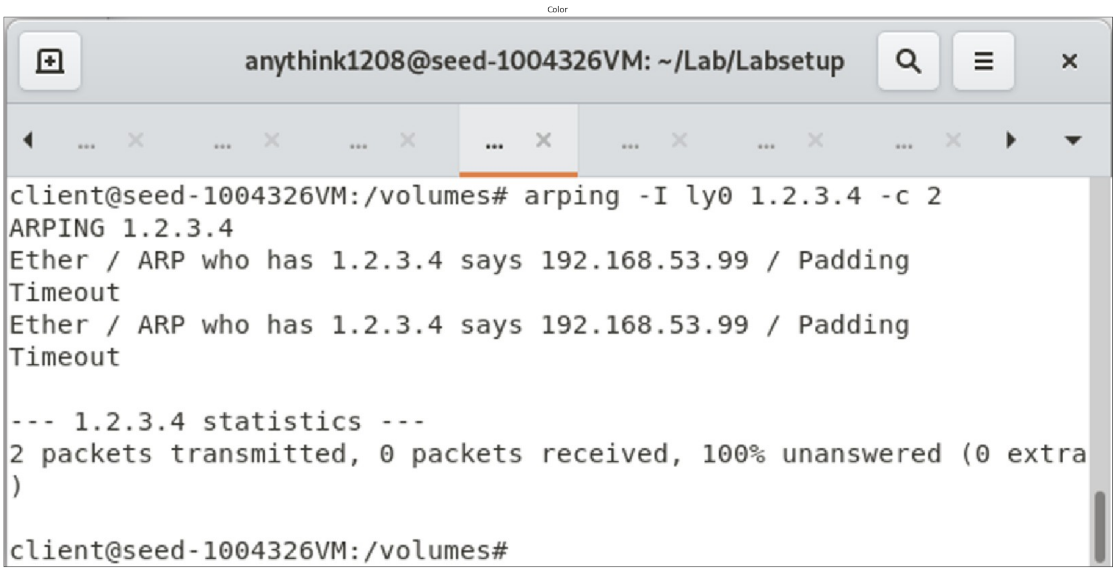
Since the tun_server is located 192.168.53.99/24, the server does not receive the packets

Task 9: Experiment with the TAP Interface

```
show_img('Task 9/create_tap.png')
show_img('Task 9/arping_without_spoof.png')
show_img('Task 9/arping_without_spoof_2.png')
```







When initiating a ping to 192.168.53.5, it starts off with sending an ARP packet to find MAC address of the host with the corresponding IP. Afterwards, it sends 2 ping requests to 192.168.53.5. Arping gives 100% unanswered as 192.168.53.5 or 1.2.3.4 is non-existent in the networks.

Spoofed ARP reply TAP

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

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

# Configure the interface ly0
os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
```

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

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

while True:
     packet = os.read(tap, 2048)
     if packet:
          print("--------------------------------")
          ether = Ether(packet)
          print(ether.summary())
          # Send a spoofed ARP response
          FAKE_MAC = "aa:bb:cc:dd:ee:ff"
          if ARP in ether and ether[ARP].op == 1 :
               arp = ether[ARP]
               newether = Ether(dst=ether.src, src=FAKE_MAC)
               newarp = ARP(psrc=arp.pdst, hwsrc=FAKE_MAC,
               pdst=arp.psrc, hwdst=ether.src, op=2)
               newpkt = newether/newarp
               print("***** Fake response:
{}".format(newpkt.summary()))
               os.write(tap, bytes(newpkt))

show_img('Task 9/ping_net.png')
show_img('Task 9/arping_with_spoof.png')
```

Color

```
client@seed-1004326VM:/volumes# ./tap.py &
[1] 163
client@seed-1004326VM:/volumes# Interface Name: ly0

client@seed-1004326VM:/volumes# ping 192.168.53.5 -c 2
PING 192.168.53.5 (192.168.53.5) 56(84) bytes of data.
--------------------------------
Ether / ARP who has 192.168.53.5 says 192.168.53.99
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 192.168.53.5
--------------------------------
Ether / IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
--------------------------------
Ether / IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw

--- 192.168.53.5 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1021ms
```

```
client@seed-1004326VM:/volumes# arping -I ly0 192.168.53.33 -c 2
ARPING 192.168.53.33
--------------------------------
Ether / ARP who has 192.168.53.33 says 192.168.53.99 / Padding
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 192.168.53.33
42 bytes from aa:bb:cc:dd:ee:ff (192.168.53.33): index=0 time=10.650 usec
--------------------------------
Ether / ARP who has 192.168.53.33 says 192.168.53.99 / Padding
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 192.168.53.33
42 bytes from aa:bb:cc:dd:ee:ff (192.168.53.33): index=1 time=7.008 usec

--- 192.168.53.33 statistics ---
2 packets transmitted, 2 packets received,   0% unanswered (0 extra)
rtt min/avg/max/std-dev = 0.007/0.009/0.011/0.002 ms
client@seed-1004326VM:/volumes# arping -I ly0 1.2.3.4 -c 2
ARPING 1.2.3.4
--------------------------------
Ether / ARP who has 1.2.3.4 says 192.168.53.99 / Padding
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 1.2.3.4
42 bytes from aa:bb:cc:dd:ee:ff (1.2.3.4): index=0 time=10.221 usec
--------------------------------
Ether / ARP who has 1.2.3.4 says 192.168.53.99 / Padding
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 1.2.3.4
42 bytes from aa:bb:cc:dd:ee:ff (1.2.3.4): index=1 time=1.516 msec

--- 1.2.3.4 statistics ---
2 packets transmitted, 2 packets received,   0% unanswered (0 extra)
rtt min/avg/max/std-dev = 0.010/0.763/1.516/0.753 ms
```

With a successful ARP spoofing, a faked MAC address of aa:bb:cc:dd:ee:ff is received for an arping request of 192.168.53.33 or 1.2.3.4.