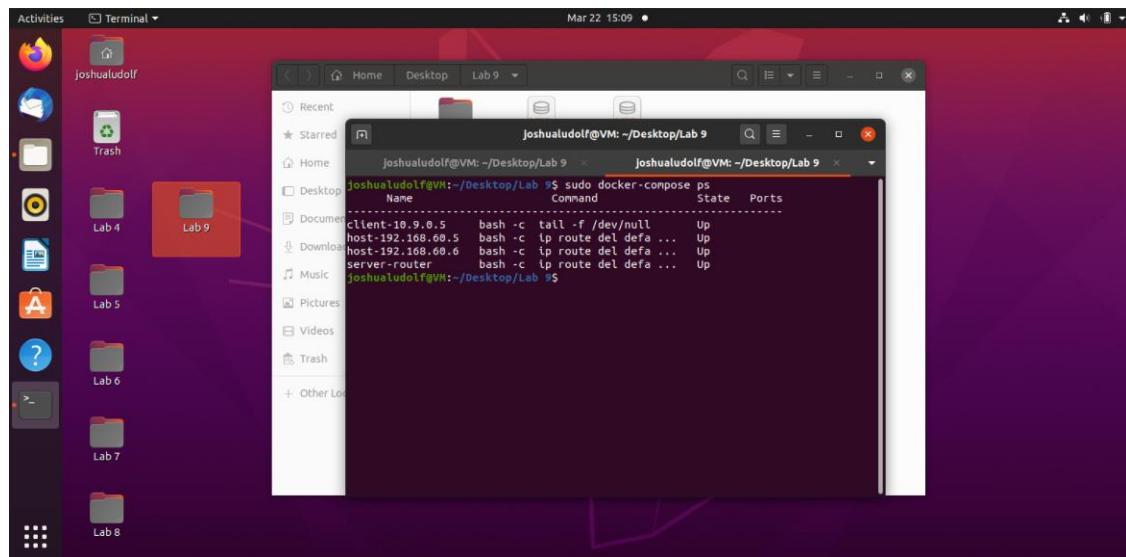
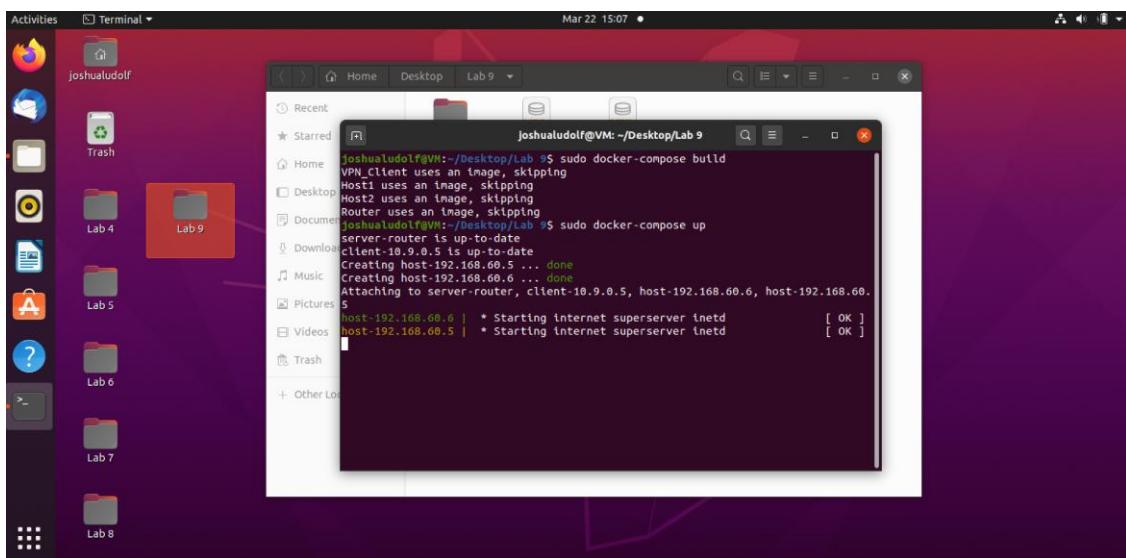


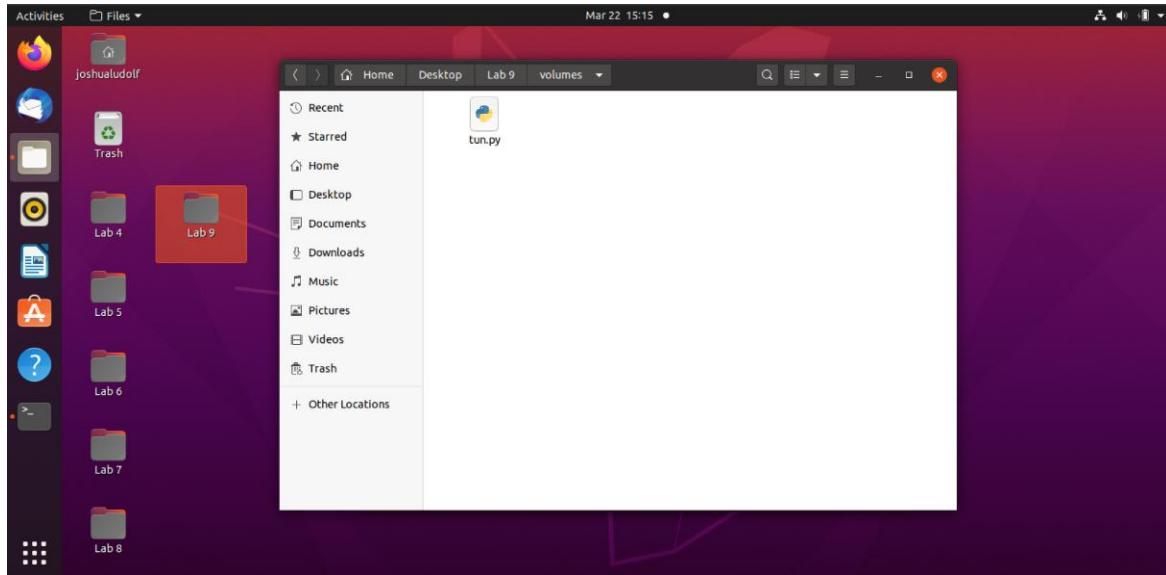
## Lab 9: VPN Lab - The Container Version

**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):



Activities    Text Editor

Mar 22 15:15 •

tun.py

```

1 #!/usr/bin/env python
2
3 import fcntl
4 import struct
5 import os
6 import time
7 from scapy.all import *
8
9 TUNSETIFF = 0x400454ca
10 IFF_TUN   = 0x0001
11 IFF_TAP   = 0x0002
12 IFF_NO_PI = 0x1000
13
14 # Create the tun interface
15 tun = os.open('/dev/net/tun', os.O_RDWR)
16 ifr = struct.pack('16s', b'tun0', IFF_TUN | IFF_NO_PI)
17 ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
18
19 # Get the interface name
20 ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
21 print("Interface Name: {}".format(ifname))
22
23 while True:
24     time.sleep(10)
25

```

Python 3 Tab Width: 8 Ln 1, Col 1 INS

A screenshot of a Linux desktop environment. On the left is a dock with icons for various applications like a browser, file manager, and terminal. A folder named 'Lab 9' is highlighted with a red box. In the center is a terminal window titled 'tun.py' with the following code:

```
1 #!/usr/bin/env python3
2
3 import fcntl
4 import struct
5 import os
6
7 tun = None
8
9 TU = 0x00000001
10 IFNAMSIZ = 16
11 IFF_TUN = 0x00000001
12 IFF_NO_PI = 0x00000004
13 IFF_UP = 0x00000002
14 IFF_RUNNING = 0x00000008
15 IFF_LOOPBACK = 0x00000010
16 IFF_LOWER_UP = 0x00000020
17 IFF_UP = 0x00000040
18 IFF_NOARP = 0x00000080
19 IFF_POINTPOINT = 0x00000100
20 IFF_BROADCAST = 0x00000200
21 IFF_MULTICAST = 0x00000400
22 IFF_UP = 0x00000800
23 IFF_UP = 0x00001000
24
25 def create_tun_interface():
26     tun = os.open("/dev/net/tun", os.O_RDWR)
27     ifr = struct.pack('isbb', 'ludolfFw', IFF_TUN | IFF_NO_PI)
28     ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
29
30     ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
31
32     print("Created interface name: " + ifname)
```

A screenshot of a Linux desktop environment, similar to the previous one. The 'Lab 9' folder is again highlighted with a red box. In the center is a terminal window titled 'tun.py' showing the output of the script:

```
1 #!/usr/bin/env python3
2
3 import fcntl
4 import struct
5 import os
6
7 tun = None
8
9 TU = 0x00000001
10 IFF_TUN = 0x00000001
11 IFF_NO_PI = 0x00000004
12 IFF_UP = 0x00000002
13 IFF_RUNNING = 0x00000008
14 IFF_LOOPBACK = 0x00000010
15 IFF_LOWER_UP = 0x00000020
16 IFF_UP = 0x00000040
17 IFF_NOARP = 0x00000080
18 IFF_POINTPOINT = 0x00000100
19 IFF_BROADCAST = 0x00000200
20 IFF_MULTICAST = 0x00000400
21 IFF_UP = 0x00000800
22 IFF_UP = 0x00001000
23
24
25 def create_tun_interface():
26     tun = os.open("/dev/net/tun", os.O_RDWR)
27     ifr = struct.pack('isbb', 'ludolfFw', IFF_TUN | IFF_NO_PI)
28     ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
29
30     ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
31
32     print("Created interface name: " + ifname)
```

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

A screenshot of a Linux desktop environment. The 'Lab 9' folder is highlighted with a red box. In the center is a terminal window titled 'tun.py' showing the output of the 'ip address' command:

```
1 #!/usr/bin/env python3
2
3 import fcntl
4 import struct
5 import os
6
7 tun = None
8
9 TU = 0x00000001
10 IFF_TUN = 0x00000001
11 IFF_NO_PI = 0x00000004
12 IFF_UP = 0x00000002
13 IFF_RUNNING = 0x00000008
14 IFF_LOOPBACK = 0x00000010
15 IFF_LOWER_UP = 0x00000020
16 IFF_UP = 0x00000040
17 IFF_NOARP = 0x00000080
18 IFF_POINTPOINT = 0x00000100
19 IFF_BROADCAST = 0x00000200
20 IFF_MULTICAST = 0x00000400
21 IFF_UP = 0x00000800
22 IFF_UP = 0x00001000
23
24
25 def create_tun_interface():
26     tun = os.open("/dev/net/tun", os.O_RDWR)
27     ifr = struct.pack('isbb', 'ludolfFw', IFF_TUN | IFF_NO_PI)
28     ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
29
30     ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
31
32     print("Created interface name: " + ifname)
```

The terminal output shows the creation of the 'ludolfFw' interface and its configuration:

```
root@sf34090b9e5:~# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    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
2: tun0: <NOARP,BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noop state DOWN group default
    link/none
3: eth0: <NOARP,BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:0a:09:00:05 brd ff:ff:ff:ff:ff:ff
    inet 10.9.0.5/24 brd 10.9.0.255 scope global eth0
        valid_lft forever preferred_lft forever
root@sf34090b9e5:~#
```

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

The screenshot shows a Linux desktop environment with a terminal window open. The terminal window title is "tun.py" and it displays the following Python script:

```
#!/usr/bin/python3
# ...
# Create the tun interface
# ...
# Set up the tun interface (load TIRF)
# ...
# Get the interface name
# ...
while True:
    time.sleep(10)
```

The terminal window has tabs for "joshualudolf@VM: ~/Desktop/Lab 9", "joshualudolf...", "joshualudolf...", and "joshualudolf...". The status bar at the bottom of the terminal window shows "Python 3 Tab Width: 8 Ln 25, Col 1 IN5".

The screenshot shows a Linux desktop environment with a purple Unity-style interface. A terminal window is open in the foreground, displaying a Python script named `tun.py`. The script uses the `fcntl`, `struct`, and `os` modules to create a network interface named `ludo1f6` with IP address `192.168.53.99`. The terminal window has tabs for multiple sessions, and the status bar at the bottom indicates Python 3, Tab Width: 8, Ln 25, Col 1, and INS mode.

```
1 #!/usr/bin/env python
2
3 import fcntl
4 import struct
5 import os
6 import time
7
8
9 TU... joshualudolf... joshualudolf... joshualudolf...
10 Iroot@f334099bb9e5:~# nano tun.py
11 Iroot@f334099bb9e5:~# python3 tun.py
12 IInterface Name: ludo1f6
13 I  ATraceback (most recent call last):
14 I    File "tun.py", line 24, in <module>
15 I        time.sleep(10)
16 I    KeyboardInterrupt
17 I    File "tun.py", line 18, in <module>
18 I        root@f334099bb9e5:~# nano tun.py
19 I    File "tun.py", line 20, in <module>
20 I        ITraceback (most recent call last):
21 I    File "tun.py", line 20, in <module>
22 I        os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
23 I    NameError: name 'ifname' is not defined
24 I    root@f334099bb9e5:~# nano tun.py
25 I    root@f334099bb9e5:~# python3 tun.py
IInterface Name: ludo1f6
```

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

```

Activities Terminal Mar 22 15:40
joshualudolf@VM: ~/Desktop/Lab 9 joshualudolf@VM: ~/Desktop/Lab 9 joshualudolf@VM: ~/Desktop/Lab 9
root@f334090b9e5:~# ip address
1: lo: <LOOPBACK,NOQUEUE,UP,BROADCAST> 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 brd 127.0.0.1 scope host lo
        valid_lft forever preferred_lft forever
2: tun0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group default qlen 500
    link/none
3: eth0:0: <NOQUEUE,BROADCAST,MULTICAST> 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
4: eth0:1: <NOQUEUE,BROADCAST,MULTICAST> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:0a:09:00:06 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.9.0.6/24 brd 10.9.0.255 scope global eth0
        valid_lft forever preferred_lft forever
5: eth0:2: <NOQUEUE,BROADCAST,MULTICAST> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:0a:09:00:07 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.9.0.7/24 brd 10.9.0.255 scope global eth0
        valid_lft forever preferred_lft forever
root@f334090b9e5:~#

```

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

```

Activities Terminal
joshualudolf@VM: ~/Desktop/Lab 9 joshualudolf@VM: ~/Desktop/Lab 9
GNU nano 4.8
import struct
import os
import time
from scapy.all import *

TUNSETIFF = 0x400454ca
IFF_TUN   = 0x0001
IFF_TAP   = 0x0002
IFF_NO_PI = 0x1000

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16s', b'ludolf\x00', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
ifname = ifname_bytes.decode('UTF-8')[16:].strip("\x00")

# Set up the tun interface (Ludolf)
os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

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

while True:
    # Get a packet from the tun interface
    packet = os.read(tun, 2048)
    if packet:
        ip = IP(packet)
        print(ip.summary())

File Name to Write: tun.py
^D Get Help          M-D DOS Format
^C Cancel           M-M Mac Format

```

```
Activities Terminal Mar 22 15:51
joshualudolf@VM: ~/Desktop/Lab 9 joshualudolf@VM: ~/Desktop/Lab 9 joshualudolf@VM: ~/Desktop/Lab 9 joshualudolf@VM: ~/Desktop/Lab 9

^CTraceback (most recent call last):
  File "tun.py", line 24, in <module>
    time.sleep(10)
KeyboardInterrupt

root@f334d90b9e5:/# nano tun.py
root@f334d90b9e5:/# python3 tun.py
Traceback (most recent call last):
  File "tun.py", line 28, in <module>
    os.system("ip addr add 192.168.33.99/24 dev {}".format(ifname))
NameError: name 'ifname' is not defined
root@f334d90b9e5:/# nano tun.py
root@f334d90b9e5:/# python3 tun.py
Interface Name: ludo7a
^CTraceback (most recent call last):
  File "tun.py", line 29, in <module>
    time.sleep(10)
KeyboardInterrupt

A ? root@f334d90b9e5:/# nano tun.py
root@f334d90b9e5:/# python3 tun.py
File "tun.py", line 34

> ^ SyntaxError: unexpected EOF while parsing
root@f334d90b9e5:/# nano tun.py
root@f334d90b9e5:/# python3 tun.py
File "tun.py", line 34

> ^
SyntaxError: unexpected EOF while parsing
root@f334d90b9e5:/# nano tun.py
root@f334d90b9e5:/# python3 tun.py
Interface Name: ludo7a
```

- ❖ 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:

The screenshot shows a Linux desktop environment with a dark theme. On the left, there is a vertical dock containing icons for various applications and files, including 'Activities', 'Terminal', 'joshualudolf', 'Trash', 'Lab 4', 'Lab 9' (which is highlighted with a red box), 'Lab 5', 'Lab 6', 'Lab 7', and 'Lab 8'. The main window is a terminal application titled 'tun.py' with the path '/Desktop/Lab 9/volumes'. The terminal window has a dark background and displays the following Python script and its execution:

```
1 #!/usr/bin/env python3
2
3 import fcntl
4 import struct
5 import os
6 import time
7
8 joshualudolf@VM: ~/Desktop/Lab 9
9 TU:~ joshualudolf... joshualudolf... joshualudolf... joshualudolf...
10 If root@f33499b9e5:/# ping 192.168.60.0
11 PING 192.168.60.0 (192.168.60.0) 56(84) bytes of data.
12
13 I=1
14 F=0x0
15 T=0x0
16 T=0x0
17 T=0x0
18
19 R=0
20 B=0
21 pr
22
23 wh
24
25
```

The terminal window also shows the command 'ping 192.168.60.0' being run, and the output indicates a successful ping. The status bar at the bottom of the terminal window shows 'Python 3' as the interpreter, 'Tab Width: 8', 'Ln 25, Col 1', and 'INS'.

The screenshot shows a Linux desktop environment with a terminal window open. The terminal window title is "tun.py" and it contains the following Python script:

```
1 #!/usr/bin/env python
2
3 import fcntl
4 import struct
5 import os
6 import time
7
8 tun = os.open('tun0', os.O_RDWR)
9
10 IP = '192.168.53.99'
11 ICMP = '192.168.53.99'
12
13 for i in range(10):
14     raw = tun.read(2048)
15     if raw[0] == 8:
16         if raw[1] == 1:
17             print(raw)
18
19 # CTraceback (most recent call last):
#   File "tun.py", line 30, in <module>
#     packet = os.read(tun, 2048)
20 L[KeyboardInterrupt]
21 V[KeyboardInterrupt]
22
23 w[roo
t@5f334090b9e5:~/Desktop/Lab 9]$ ./tun.py
24 Interface Name: ludu0
25
26 CTraceback (most recent call last):
#   File "tun.py", line 30, in <module>
#     packet = os.read(tun, 2048)
KeyboardInterrupt

root@5f334090b9e5:~/Desktop/Lab 9$ ./tun.py
Interface Name: ludu0
```

- ❖ 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

The terminal window shows the following Python script named `tun.py`:

```
#!/usr/bin/env python
import fcntl
import struct
import os
TUNSETIFF = 0x40005490
IFF_TUN = 0x2000
IFF_NO_PI = 0x1000
ifname = "tun"
os.system("ip link add %s type tun %s < /dev/null" % (ifname, IFF_NO_PI | IFF_TUN))
os.system("ip link set dev %s up" % ifname)
# Get the interface name
ifname_bytes = os.read(ifname, 16).strip("\x00")
print("Interface Name: %s" % ifname_bytes.decode('UTF-8'))
while True:
    # Get a packet from the tun interface
    packet = os.read(ifname, 2048)
    if packet:
        ip = IP(packet)
        print(ip.summary())
    # Send out a spoofed packet using the tun interface
    newip = IP(src='1.2.3.4', dst=ip.src)
    newpkt = newip/ip.payload
    os.write(ifname, bytes(newpkt))
```

File Name to Write: tun.py

File operations menu:

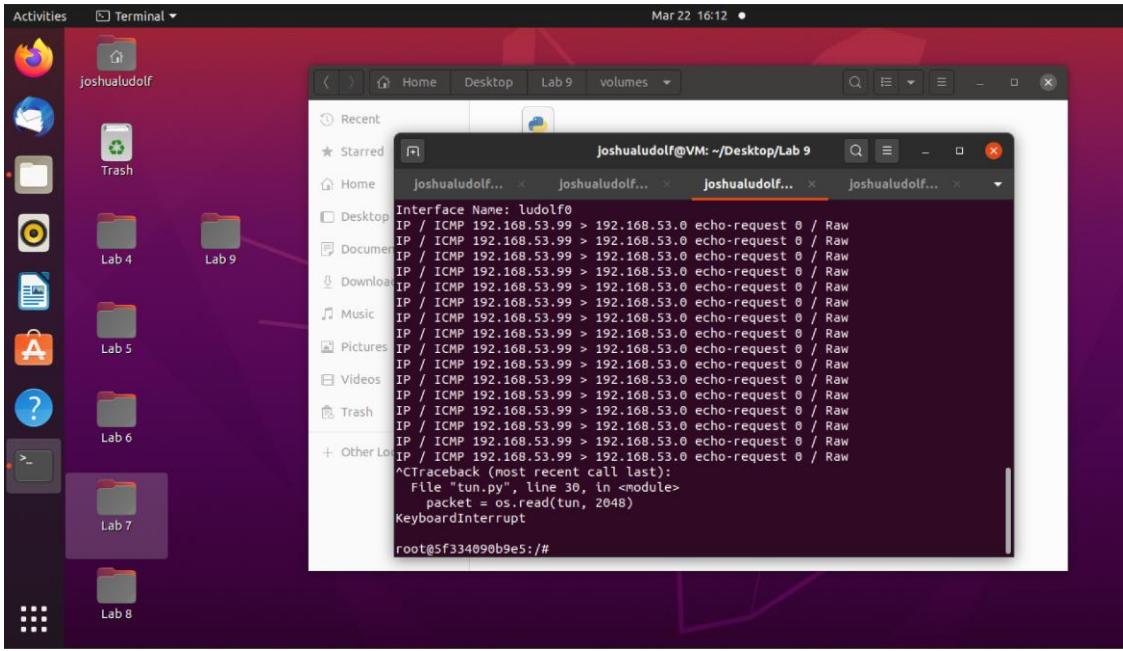
- Get Help
- DOS Format
- Append
- Backup File
- Cancel
- Mac Format
- Prepend
- To Files

Python 3 Tab Width: 8 Ln 25, Col 1 INS

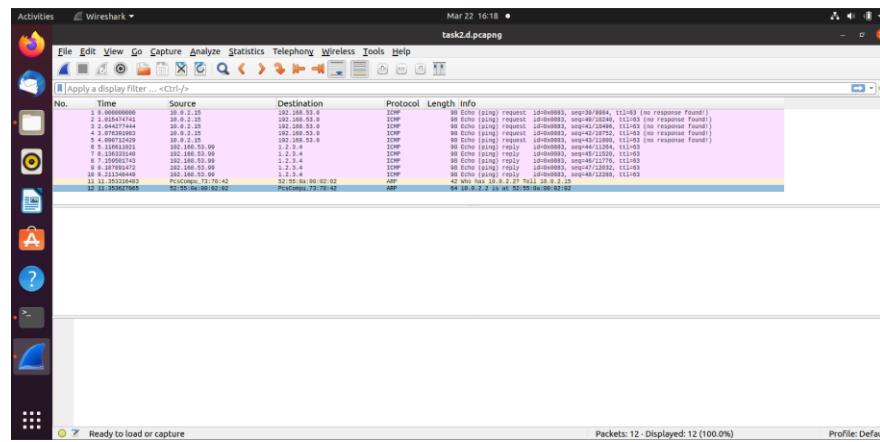
Activities Terminal Mar 22 16:11

The terminal window shows the following root shell session:

```
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

```
GNU nano 4.8
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\x0d', 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

```
GNU nano 4.8
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\x0d', 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

^CTraceback (most recent call last):
  File "tun.py", line 38, 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
IP / ICMP 192.168.53.99 > 192.168.53.0 echo-request 0 / Raw
^CTraceback (most recent call last):
  File "tun.py", line 38, 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
IP / ICMP 192.168.53.99 > 192.168.53.0 echo-request 0 / Raw
^CTraceback (most recent call last):
  File "tun.py", line 38, 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 17:11

joshualudolf@VM: ~/Desktop/Lab 9\$ sudo docker exec -it server-router bash

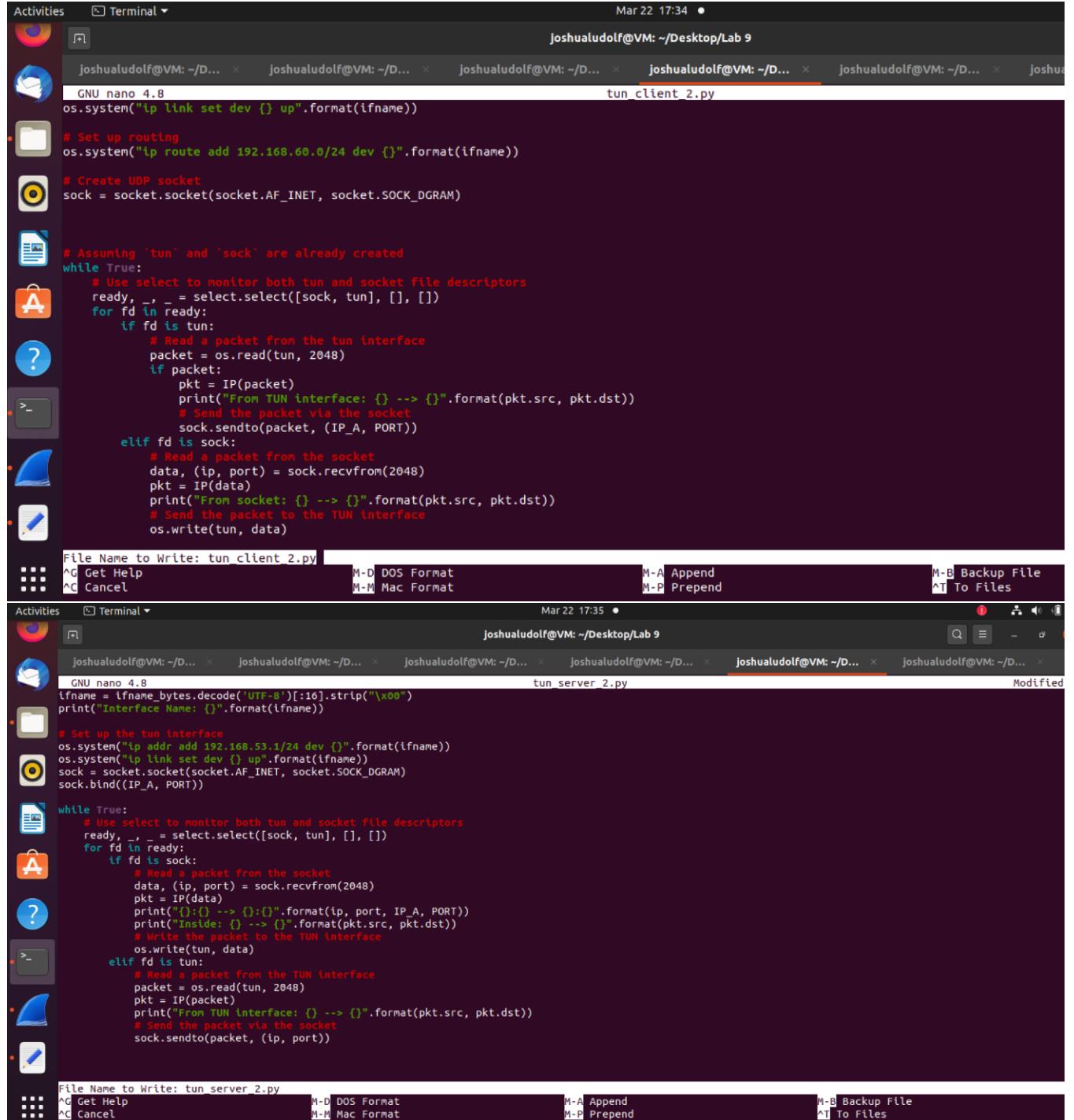
```
root@584a45f02dfb:/# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
21: eth0@if22: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:0a:09:00:0b brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.9.0.11/24 brd 10.9.0.255 scope global eth0
        valid_lft forever preferred_lft forever
23: eth1@if24: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:c0:a8:3c:0b brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.60.11/24 brd 192.168.60.255 scope global eth1
        valid_lft forever preferred_lft forever
root@584a45f02dfb:/# nano tun_server.py
root@584a45f02dfb:/# python3 tun_server.py
^CTraceback (most recent call last):
  File "tun_server.py", line 12, in <module>
    data, (ip, port) = sock.recvfrom(2048)
KeyboardInterrupt

root@584a45f02dfb:/# nano tun_server.py
root@584a45f02dfb:/# python3 tun_server.py
Interface Name: ludolf0
```

- ❖ 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 17:34 joshualudolf@VM: ~/Desktop/Lab 9
joshualudolf@VM: ~/D... joshualudolf@VM: ~/D... joshualudolf@VM: ~/D... joshualudolf@VM: ~/D... joshualudolf@VM: ~/D... joshualudolf@VM: ~/D...
GNU nano 4.8
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)

# Assuming 'tun' and 'sock' are already created
while True:
    # Use select to monitor both tun and socket file descriptors
    ready, _, _ = select.select([sock, tun], [], [])
    for fd in ready:
        if fd is tun:
            # Read a packet from the tun interface
            packet = os.read(tun, 2048)
            if packet:
                pkt = IP(packet)
                print("From TUN interface: {} --> {}".format(pkt.src, pkt.dst))
                # Send the packet via the socket
                sock.sendto(packet, (IP_A, PORT))
        elif fd is sock:
            # Read a packet from the socket
            data, (ip, port) = sock.recvfrom(2048)
            pkt = IP(data)
            print("From socket: {} --> {}".format(pkt.src, pkt.dst))
            # Send the packet to the TUN interface
            os.write(tun, data)

File Name to Write: tun_client_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 17:35 joshualudolf@VM: ~/Desktop/Lab 9
joshualudolf@VM: ~/D... joshualudolf@VM: ~/D... joshualudolf@VM: ~/D... joshualudolf@VM: ~/D... joshualudolf@VM: ~/D... joshualudolf@VM: ~/D...
GNU nano 4.8
ifname = fname_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:
    # 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
```



```

root@76b7550689c3:~# tcpdump -l eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
21:38:05.408647 IP 76b7550689c3 > 192.168.53.99: ICMP echo request, id 277, seq 26, length 64
21:38:05.419988 IP 76b7550689c3.51744 > 192.168.53.99: ICMP echo reply, id 277, seq 26, length 64
21:38:06.410099 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 277, seq 27, length 64
21:38:06.410115 IP 76b7550689c3 > 192.168.53.99: ICMP echo reply, id 277, seq 27, length 64
21:38:06.646981 ARP: Request who-has server-router.net-192.168.60.0 tell 76b7550689c3, length 28
21:38:06.646992 ARP: Reply 76b7550689c3 ls-at 02:42:c0:a8:3c:0b (oui Unknown), length 28
21:38:06.646994 ARP, Reply server-router.net-192.168.60.0 ls-at 02:42:c0:a8:3c:0b (oui Unknown), length 28
21:38:07.415070 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 277, seq 28, length 64
21:38:07.415222 IP 76b7550689c3 > 192.168.53.99: ICMP echo reply, id 277, seq 28, length 64
21:38:08.415660 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 277, seq 29, length 64
21:38:08.415676 IP 76b7550689c3 > 192.168.53.99: ICMP echo reply, id 277, seq 29, length 64
21:38:09.419733 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 277, seq 30, length 64
21:38:09.419766 IP 76b7550689c3 > 192.168.53.99: ICMP echo reply, id 277, seq 30, length 64
21:38:10.417511 IP 76b7550689c3.46720 > 192.168.53.99: ICMP echo request, id 277, seq 31, length 64
21:38:10.420011 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 277, seq 31, length 64
21:38:10.420026 IP 76b7550689c3 > 192.168.53.99: ICMP echo reply, id 277, seq 31, length 64
21:38:11.420449 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 277, seq 32, length 64
21:38:11.420463 IP 76b7550689c3 > 192.168.53.99: ICMP echo reply, id 277, seq 32, length 64
21:38:12.427177 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 277, seq 33, length 64
21:38:12.427192 IP 76b7550689c3 > 192.168.53.99: ICMP echo reply, id 277, seq 33, length 64
21:38:13.430823 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 277, seq 34, length 64
21:38:13.430841 IP 76b7550689c3 > 192.168.53.99: ICMP echo reply, id 277, seq 34, length 64
21:38:14.435061 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 277, seq 35, length 64
21:38:14.435076 IP 76b7550689c3 > 192.168.53.99: ICMP echo reply, id 277, seq 35, length 64
21:38:15.424112 IP 76b7550689c3.53219 > 192.168.53.99: ICMP echo request, id 277, seq 36, length 64
21:38:15.424112 IP 76b7550689c3 > 192.168.53.99: ICMP echo request, id 277, seq 36, length 64
21:38:15.435431 IP 192.168.53.99 > 76b7550689c3: ICMP echo reply, id 277, seq 36, length 64
21:38:16.441266 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 277, seq 37, length 64
21:38:16.441147 IP 76b7550689c3 > 192.168.53.99: ICMP echo reply, id 277, seq 37, length 64
21:38:17.438677 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 277, seq 38, length 64
21:38:17.438701 IP 76b7550689c3 > 192.168.53.99: ICMP echo reply, id 277, seq 38, length 64

```

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

```

joshualudolf@VM...:~/Desktop/Lab 9$ sudo docker exec -it host-192.168.60.5 bash
root@76b7550689c3:~# exit
joshualudolf@VM...:~/Desktop/Lab 9$ sudo docker exec -it client-10.9.0.5 bash
root@5f334090b9e5:/# telnet 192.168.60.5...
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^'.
ddddd
Ubuntu 20.04.1 LTS
ddddd
76b7550689c3 login: 

```

- ❖ 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... 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:/#
```

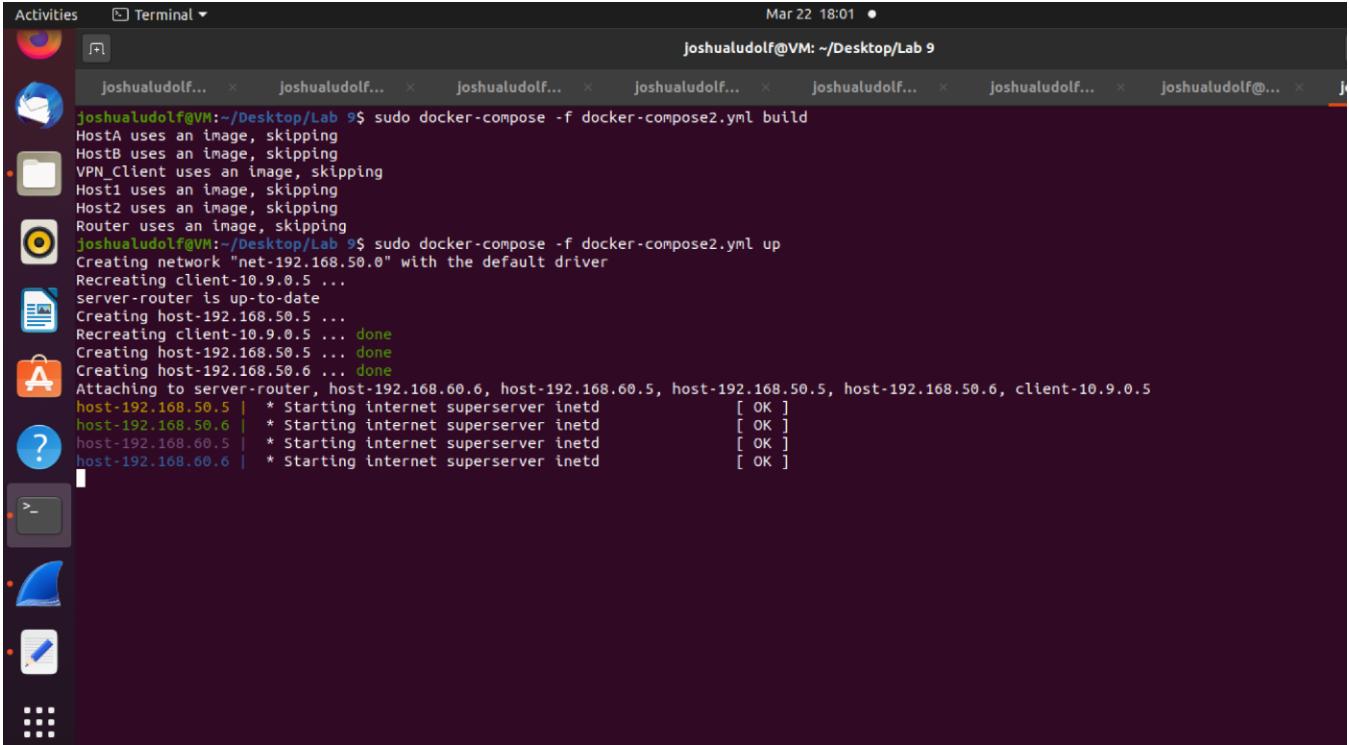
Activities Terminal Mar 22 17:56

```
joshualudolf@VM: ~/Desktop/Lab 9
```

```
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:/# tcpdump -i eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
21:55:05.910279 ARP, Request who-has 192.168.53.99 tell 76b7550689c3, length 28
21:55:05.911609 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 310, seq 13, length 64
21:55:06.934215 ARP, Request who-has 192.168.53.99 tell 76b7550689c3, length 28
21:55:06.936938 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 310, seq 14, length 64
21:55:07.961366 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 310, seq 15, length 64
21:55:07.961380 ARP, Request who-has 192.168.53.99 tell 76b7550689c3, length 28
21:55:08.994011 ARP, Request who-has 192.168.53.99 tell 76b7550689c3, length 28
21:55:08.996446 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 310, seq 16, length 64
21:55:10.006208 ARP, Request who-has 192.168.53.99 tell 76b7550689c3, length 28
21:55:10.008675 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 310, seq 17, length 64
21:55:11.032151 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 310, seq 18, length 64
21:55:11.032161 ARP, Request who-has 192.168.53.99 tell 76b7550689c3, length 28
21:55:12.054860 ARP, Request who-has 192.168.53.99 tell 76b7550689c3, length 28
21:55:12.056646 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 310, seq 19, length 64
21:55:13.078210 ARP, Request who-has 192.168.53.99 tell 76b7550689c3, length 28
21:55:13.079926 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 310, seq 20, length 64
21:55:14.103334 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 310, seq 21, length 64
21:55:14.103349 ARP, Request who-has 192.168.53.99 tell 76b7550689c3, length 28
21:55:15.126675 ARP, Request who-has 192.168.53.99 tell 76b7550689c3, length 28
21:55:15.128006 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 310, seq 22, length 64
21:55:16.153843 ARP, Request who-has 192.168.53.99 tell 76b7550689c3, length 28
21:55:16.155140 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 310, seq 23, length 64
21:55:17.177966 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 310, seq 24, length 64
21:55:17.178081 ARP, Request who-has 192.168.53.99 tell 76b7550689c3, length 28
21:55:18.198523 ARP, Request who-has 192.168.53.99 tell 76b7550689c3, length 28
21:55:18.200553 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 310, seq 25, length 64
21:55:19.223178 ARP, Request who-has 192.168.53.99 tell 76b7550689c3, length 28
21:55:19.225350 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 310, seq 26, length 64
21:55:20.249026 IP 192.168.53.99 > 76b7550689c3: ICMP echo request, id 310, seq 27, length 64
21:55:20.249055 ARP. Reauest who-has 192.168.53.99 tell 76b7550689c3. lenath 28

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
```

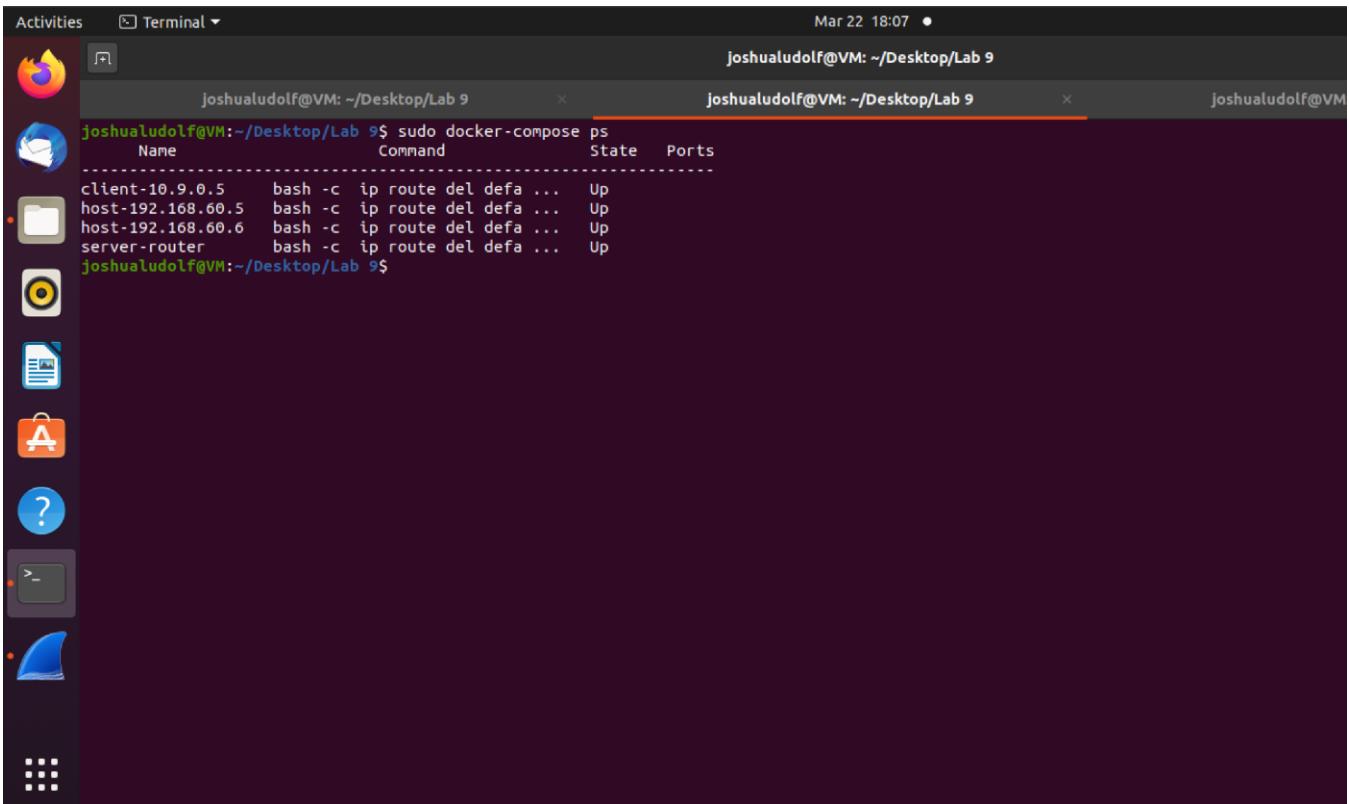
❖ For task 8, I started with building and starting the docker-compose2.yml file:



A screenshot of a Linux desktop environment, likely Ubuntu, showing a terminal window titled "Terminal". The terminal window has a dark background and contains the following command-line session:

```
joshualudolf@VM:~/Desktop/Lab 9$ sudo docker-compose -f docker-compose2.yml build
HostA uses an image, skipping
HostB uses an image, skipping
VPN_Client uses an image, skipping
Host1 uses an image, skipping
Host2 uses an image, skipping
Router uses an image, skipping
joshualudolf@VM:~/Desktop/Lab 9$ sudo docker-compose -f docker-compose2.yml up
Creating network "net-192.168.50.0" with the default driver
Recreating client-10.9.0.5 ...
server-router is up-to-date
Creating host-192.168.50.5 ...
Recreating client-10.9.0.5 ... done
Creating host-192.168.50.5 ... done
Creating host-192.168.50.6 ... done
Attaching to server-router, host-192.168.60.6, host-192.168.60.5, host-192.168.50.5, host-192.168.50.6, client-10.9.0.5
host-192.168.50.5 | * Starting internet superserver inetd [ OK ]
host-192.168.50.6 | * Starting internet superserver inetd [ OK ]
host-192.168.60.5 | * Starting internet superserver inetd [ OK ]
host-192.168.60.6 | * Starting internet superserver inetd [ OK ]
```

- ❖ 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):



A screenshot of a Linux desktop environment, likely Ubuntu, showing three terminal windows. The first terminal window is active and shows the command "sudo docker-compose ps". The output is as follows:

Name	Command	State	Ports
client-10.9.0.5	bash -c ip route del defa ...	Up	
host-192.168.60.5	bash -c ip route del defa ...	Up	
host-192.168.60.6	bash -c ip route del defa ...	Up	
server-router	bash -c ip route del defa ...	Up	

The second terminal window is visible in the background, and the third terminal window is also visible.

Activities Terminal Mar 22 18:14

joshualudolf@VM: ~/Desktop/Lab 9 x joshualudolf@VM: ~/Desktop/Lab 9 x joshualudolf@VM: ~/Desktop/Lab 9 x 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\x0d', 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
^X Exit ^R Read File ^A Replace ^U Paste Text ^H To Spell ^G Go To Line M-E Redo
M-A Mark Text M-C Copy Text M-B To Bracket
^Q Where Was
```

Activities Terminal Mar 22 18:12

joshualudolf@VM: ~/Desktop/Lab 9 x joshualudolf@VM: ~/Desktop/Lab 9 x joshualudolf@VM: ~/Desktop/Lab 9 x joshualudolf@VM: ~/D

```
GNU nano 4.8 tun_server 2.py
#!/usr/bin/env python3

tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'ludolf\x0d', 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))

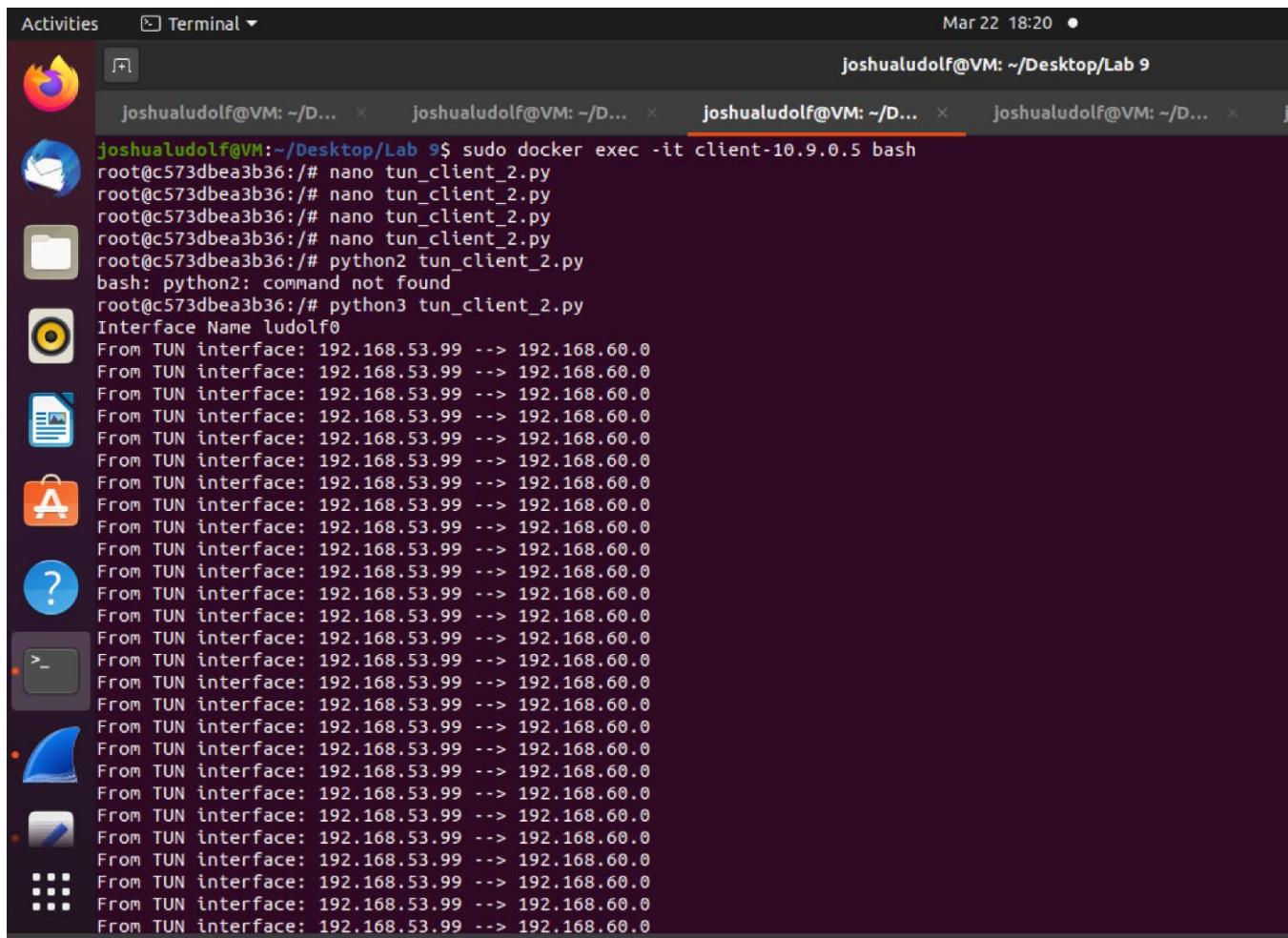
File Name to Write: tun_server 2.py
^G Get Help M-D DOS Format M-A Append M-B Backup File
^C Cancel M-M Mac Format M-P Prepend ^T To Files
```

Activities Terminal Mar 22 18:19

joshualudolf@VM: ~/Desktop/Lab 9\$ 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: ~/D... x joshualudolf@VM: ~/D... x
```

```
joshualudolf@VM:~/Desktop/Lab 9$ sudo docker exec -it client-10.9.0.5 bash
root@c573dbea3b36:/# tcpdump -i eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
22:18:53.975258 IP c573dbea3b36.52121 > server-router.net-10.9.0.0.9090: UDP, length 84
22:18:54.999740 IP c573dbea3b36.52121 > server-router.net-10.9.0.0.9090: UDP, length 84
22:18:56.022568 IP c573dbea3b36.52121 > server-router.net-10.9.0.0.9090: UDP, length 84
22:18:57.047611 IP c573dbea3b36.52121 > server-router.net-10.9.0.0.9090: UDP, length 84
22:18:58.071540 IP c573dbea3b36.52121 > server-router.net-10.9.0.0.9090: UDP, length 84
22:18:59.096752 IP c573dbea3b36.52121 > server-router.net-10.9.0.0.9090: UDP, length 84
22:19:00.119892 IP c573dbea3b36.52121 > server-router.net-10.9.0.0.9090: UDP, length 84
22:19:01.142633 IP c573dbea3b36.52121 > server-router.net-10.9.0.0.9090: UDP, length 84
```

```
22:19:56.630475 IP c573dbea3b36.52121 > server-router.net-10.9.0.0.9090: UDP, length 84
22:19:56.725809 ARP, Request who-has server-router.net-10.9.0.0 tell c573dbea3b36, length 28
22:19:56.726128 ARP, Reply server-router.net-10.9.0.0 is-at 02:42:0a:09:00:0b (oui Unknown), length 28
```

- ❖ 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

```
GNU nano 4.8
#!/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_tap0', 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
^C 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: ~/Desktop/Lab 9$ nano tun_server_2.py
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:

In 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 `ip` and learned how to set it up by configuring the appropriate flags with system calls like `socket`. 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.