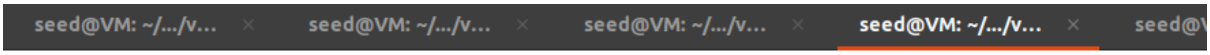
Name: Dhanashree Reddy Srinivasa Reddy

SUID: 393473169

Course: Internet Security

Task1:

- Host U can communicate with VPN Server.




```
server:volumes$>ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=64 time=0.155 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=64 time=0.094 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=64 time=0.086 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=64 time=0.070 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=64 time=0.084 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=64 time=0.110 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=64 time=2.09 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=64 time=1.17 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=64 time=0.187 ms
64 bytes from 192.168.60.5: icmp_seq=10 ttl=64 time=0.071 ms
64 bytes from 192.168.60.5: icmp_seq=11 ttl=64 time=0.345 ms
64 bytes from 192.168.60.5: icmp_seq=12 ttl=64 time=0.069 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=64 time=0.077 ms
^C
--- 192.168.60.5 ping statistics ---
13 packets transmitted, 13 received, 0% packet loss, time 12290ms
rtt min/avg/max/mdev = 0.069/0.354/2.093/0.578 ms
```

Server successfully pinged Host U

• VPN Server can communicate with Host V.

```
server:volumes$>ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.272 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.081 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=64 time=0.434 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=64 time=0.238 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=64 time=0.094 ms
^C
--- 10.9.0.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4111ms
rtt min/avg/max/mdev = 0.081/0.223/0.434/0.129 ms
```

Server successfully pinged Host V

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

```
HostU:volumes$>ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
```

Host U not able to ping Host V

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

```
server:volumes$>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
04:47:47.244836 IP 10.9.0.1.mdns > 224.0.0.251.mdns: 0 [9q] PTR (QM)? _nfs._tcp.local. PTR (QM)? _ip
p._tcp.local. PTR (QM)? _ipps._tcp.local. PTR (QM)? _ftp._tcp.local. PTR (QM)? _webdav._tcp.local. P
TR (QM)? _webdavs._tcp.local. PTR (QM)? _sftp-ssh._tcp.local. PTR (QM)? _smb._tcp.local. PTR (QM)? _
afpovertcp._tcp.local. (141)
04:47:47.245886 IP6 fe80::42:9eff:fe4a:57b4.mdns > ff02::fb.mdns: 0 [9q] PTR (QM)? _nfs._tcp.local.
PTR (QM)? _ipp._tcp.local. PTR (QM)? _ipps._tcp.local. PTR (QM)? _ftp._tcp.local. PTR (QM)? _webdav.
_tcp.local. PTR (QM)? _webdavs._tcp.local. PTR (QM)? _sftp-ssh._tcp.local. PTR (QM)? _smb._tcp.local
. PTR (QM)? _afpovertcp._tcp.local. (141)
04:47:47.328391 ARP, Request who-has 10.9.0.1 tell 20b9f2e06236, length 28
04:47:47.328467 ARP, Reply 10.9.0.1 is-at 02:42:9e:4a:57:b4 (oui Unknown), length 28
04:47:47.328479 IP 20b9f2e06236.53223 > SAC2V1A.lan.domain: 21060+ PTR? 251.0.0.224.in-addr.arpa. (4
2)
04:47:47.455596 IP SAC2V1A.lan.domain > 20b9f2e06236.53223: 21060 NXDomain 0/1/1 (110)
04:47:47.475519 IP 20b9f2e06236.58949 > SAC2V1A.lan.domain: 32300+ PTR? 1.0.9.10.in-addr.arpa. (39)
04:47:47.480756 IP SAC2V1A.lan.domain > 20b9f2e06236.58949: 32300 NXDomain* 0/0/0 (39)
04:47:47.482389 IP 20b9f2e06236.44247 > SAC2V1A.lan.domain: 26308+ PTR? b.f.0.0.0.0.0.0.0.0.0.0.0.
0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.f.f.ip6.arpa. (90)
```

Task2a. Name of the Interface

```
  GNU nano 4.8                                          tun.py
import os
import time
from scapy.all import *

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

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)
ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)

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

while True:
   time.sleep(10)
```

Run the python code on Host U

```
root@f8a15b1e2340:/volumes# tun.py
Interface Name: tun0
```

Get all the interfaces on the machine: Can see tun0

```
root@f8a15b1e2340:/# 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
4: tun0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group default qlen 500
    link/none
49: eth0@if50: <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
```

In the python program, change the name of the interface to last name:

```
from scapy.all import *

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

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'srini', IFF_TUN | IFF_NO_PI)
ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)

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

while True:
   time.sleep(10)
```

New interface name on Host U: srini

```
root@f8a15b1e2340:/volumes# tun.py
Interface Name: srini
```

Interface tun0 is replaced with srini

```
root@f8a15b1e2340:/volumes# 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
5: srini: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group default qlen 500
    link/none
49: eth0@if50: <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
```

Task 2b: Set up the TUN Interface

In the above program we just set up the name of the interface, but IP address the state is DOWN.

```
  GNU nano 4.8                                          tun2b.py
TUNSETIFF = 0x400454ca
IFF_TUN   = 0x0001
IFF_TAP   = 0x0002
IFF_NO_PI = 0x1000

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'srini', IFF_TUN | IFF_NO_PI)
ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)

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

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

while True:
   time.sleep(10)
```

Add the ip address to the ‘srini’ interface. And bring UP the interface using the below configuration.

```
root@f8a15b1e2340:/#  ip addr add 192.168.53.99/24 dev srini
root@f8a15b1e2340:/#  ip link set dev tun0 up
Cannot find device "tun0"
root@f8a15b1e2340:/#  ip link set dev srini up
root@f8a15b1e2340:/# 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
7: srini: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group default
qlen 500
    link/none
    inet 192.168.53.99/24 scope global srini
       valid_lft forever preferred_lft forever
49: eth0@if50: <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
```

2c: Read from the TUN Interface:

The code below reads the packet from the TUN interface and converts it into a ip packet and prints each field in the packet.

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

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

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

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

Try to ping 192.168.53.1 on the Host U, packets are transmitted, tun can receive data packets, but will not return any content.

```
root@f8a15b1e2340:/volumes# ping 192.168.53.1
PING 192.168.53.1 (192.168.53.1) 56(84) bytes of data.

--- 192.168.53.1 ping statistics ---
44 packets transmitted, 0 received, 100% packet loss, time 44106ms
```

The packets are received by tun2c.py program. The IP address of the srini0 interface is 192.168.53.99(source address), so this ip address captures the packets sent out by 192.168.53.1

```
root@f8a15b1e2340:/volumes# tun2c.py
Interface Name: srini0
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
^CTraceback (most recent call last):
  File "./tun2c.py", line 28, in <module>
    packet = os.read(tun, 2048)
KeyboardInterrupt
```

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

```
root@f8a15b1e2340:/volumes# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^O
^C
--- 192.168.60.5 ping statistics ---
33 packets transmitted, 0 received, 100% packet loss, time 32759ms

root@f8a15b1e2340:/volumes#
root@f8a15b1e2340:/volumes#
root@f8a15b1e2340:/volumes#
root@f8a15b1e2340:/volumes# ping 192.168.53.1
PING 192.168.53.1 (192.168.53.1) 56(84) bytes of data.
^C
--- 192.168.53.1 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3075ms

root@f8a15b1e2340:/volumes#
root@f8a15b1e2340:/volumes#
root@f8a15b1e2340:/volumes#
root@f8a15b1e2340:/volumes# tun2c.py
Interface Name: srini0
^O
```

We pinged another internal network, and found that the ping was successful, but it did not go through the tun we just set.

Task 2d: Write to the TUN Interface

The above program only reads the packet, add a write() call to the while loop so that the program will construct the ICMP reply packet and write it to the TUN interface.

```
  GNU nano 4.8                                        tun2d.py

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

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

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

                # Send out a spoof packet using the tun interface
                newip = IP(src='192.168.53.3', dst=ip.src)
                newpkt = newip/ip.payload
                os.write(tun, bytes(newpkt))
```

```
root@f8a15b1e2340:/volumes# tun2d.py
Interface Name: srini0
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-reply 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-reply 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-reply 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-reply 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-reply 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-reply 0 / Raw
^CTraceback (most recent call last):
  File "./tun2d.py", line 28, in <module>
    packet = os.read(tun, 2048)
KeyboardInterrupt
```

The other end of the TUN interface receives the packets and reply packets written to the interface and reach the ping program.

```
root@f8a15b1e2340:/volumes# ping 192.168.53.3
PING 192.168.53.3 (192.168.53.3) 56(84) bytes of data.
64 bytes from 192.168.53.3: icmp_seq=1 ttl=64 time=11.2 ms
64 bytes from 192.168.53.3: icmp_seq=2 ttl=64 time=7.05 ms
64 bytes from 192.168.53.3: icmp_seq=3 ttl=64 time=8.08 ms
64 bytes from 192.168.53.3: icmp_seq=4 ttl=64 time=6.72 ms
64 bytes from 192.168.53.3: icmp_seq=5 ttl=64 time=6.57 ms
64 bytes from 192.168.53.3: icmp_seq=6 ttl=64 time=6.32 ms
^C
--- 192.168.53.3 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5012ms
rtt min/avg/max/mdev = 6.317/7.647/11.159/1.667 ms
```

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

```
  GNU nano 4.8                                  tun2d.py
# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface Name: {}".format(ifname))

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

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

                # Send out a spoof packet using the tun interface
                newip = IP(src= '192.168.53.3', dst=ip.src)
                newpkt = newip/ip.payload
                arb_data = b'Any arbitary data'
                os.write(tun, arb_data)
```

```
root@f8a15b1e2340:/# ping 192.168.53.3
PING 192.168.53.3 (192.168.53.3) 56(84) bytes of data.
^C
--- 192.168.53.3 ping statistics ---
22 packets transmitted, 0 received, 100% packet loss, time 21502ms
```

When some arbitrary data is written to the TUN interface then the ping does not receive any packet.

```
root@f8a15b1e2340:/volumes# tun2d.py
Interface Name: srini0
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.3 echo-request 0 / Raw
^CTraceback (most recent call last):
  File "./tun2d.py", line 28, in <module>
    packet = os.read(tun, 2048)
KeyboardInterrupt
```

```
root@f8a15b1e2340:/# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on srini0, link-type RAW (Raw IP), capture size 262144 bytes
01:43:41.718330 IP 192.168.53.99 > 192.168.53.3: ICMP echo request, id 40, seq 13, length 64
01:43:41.719808 IP [|ip]
01:43:42.742111 IP 192.168.53.99 > 192.168.53.3: ICMP echo request, id 40, seq 14, length 64
01:43:42.744209 IP [|ip]
01:43:43.766680 IP 192.168.53.99 > 192.168.53.3: ICMP echo request, id 40, seq 15, length 64
01:43:43.769301 IP [|ip]
01:43:44.791012 IP 192.168.53.99 > 192.168.53.3: ICMP echo request, id 40, seq 16, length 64
01:43:44.792937 IP [|ip]
01:43:45.815101 IP 192.168.53.99 > 192.168.53.3: ICMP echo request, id 40, seq 17, length 64
01:43:45.817000 IP [|ip]
^C
10 packets captured
10 packets received by filter
0 packets dropped by kernel
```

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

UDP socket program

```
seed@VM: ~/.../volumes   seed@VM: ~/.../volumes   seed@VM: ~/.../volumes   seed@VM: ~/.../
  GNU nano 4.8                                tun_client.py
ifr = struct.pack('16sH', b'srini%d', IFF_TUN | IFF_NO_PI)
ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)

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

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

SERVER_PORT = 9090
SERVER_IP = "10.9.0.11"

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
while True:
        # Get a packet from the tun interface
        packet = os.read(tun, 2048)
        if packet:
                sock.sendto(packet, (SERVER_IP, SERVER_PORT))
```

UDP packet is created with port number 9090. The program thinks the data in the UDP payload field is an IP packet.

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

from scapy.all import *

IP_A = "0.0.0.0"
PORT = 9090

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))
```

Run tun_client.py on Host U container

```
root@f8a15b1e2340:/volumes# chmod a+x tun_client.py
root@f8a15b1e2340:/volumes# tun_client.py
Interface Name: srini0
^CTraceback (most recent call last):
  File "./tun_client.py", line 32, in <module>
    packet = os.read(tun, 2048)
KeyboardInterrupt
```

Run tun_server.py on Server container

```
root@20b9f2e06236:/volumes# chmod a+x tun_server.py
root@20b9f2e06236:/volumes# tun_server.py
10.9.0.5:42208 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.3
10.9.0.5:42208 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.3
10.9.0.5:42208 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.3
10.9.0.5:42208 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.3
10.9.0.5:42208 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.3
10.9.0.5:42208 --> 0.0.0.0:9090
```

By executing the above code, a tunnel is created.

Packet reaches the tun_client.py program from the TUN interface. The packet is then put inside an UDP packet and sent to VPN server.

```
root@f8a15b1e2340:/volumes# ping 192.168.53.3
PING 192.168.53.3 (192.168.53.3) 56(84) bytes of data.
^C
--- 192.168.53.3 ping statistics ---
24 packets transmitted, 0 received, 100% packet loss, time 23503ms
```

Configure the routing table for Host V, and ping the host V from Host U.

Let us ping Host V and see whether the ICMP packet is sent to VPN Server through the tunnel. so the ping packet can be sent through the tunnel. This is done through routing, i.e., packets going to the 192.168.60.0/24 network should be routed to the TUN interface and be given to the tun client.py program. The following command shows how to add an entry to the routing table:

```
root@f8a15b1e2340:/# ip route add 192.168.60.5 dev srini0
root@f8a15b1e2340:/# p route
bash: p: command not found
root@f8a15b1e2340:/# ip route
default via 10.9.0.1 dev eth0
10.9.0.0/24 dev eth0 proto kernel scope link src 10.9.0.5
192.168.53.0/24 dev srini0 proto kernel scope link src 192.168.53.99
192.168.60.5 dev srini0 scope link
root@f8a15b1e2340:/#
root@f8a15b1e2340:/#
root@f8a15b1e2340:/# 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 ---
11 packets transmitted, 0 received, 100% packet loss, time 10228ms
```

```
root@f8a15b1e2340:/volumes# tun_client.py
Interface Name: srini0
^CTraceback (most recent call last):
  File "./tun_client.py", line 32, in <module>
    packet = os.read(tun, 2048)
KeyboardInterrupt
```

```
root@20b9f2e06236:/volumes# tun_server.py
10.9.0.5:43194 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:43194 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:43194 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:43194 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:43194 --> 0.0.0.0:9090
```

It can be seen that the server has successfully received and ready to forward.

Task 4: Set up the VPN server:

```
 GNU nano 4.8                          tun_client.py

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

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

SERVER_PORT = 9090
SERVER_IP = "10.9.0.11"

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

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
while True:
        # Get a packet from the tun interface
        packet = os.read(tun, 2048)
        if packet:
                sock.sendto(packet, (SERVER_IP, SERVER_PORT))
```

Set up the server program, which creates a TUN interface , get the ip packet and and write the packet to the TUN interface.

```
 GNU nano 4.8                          tun_server.py
ifr = struct.pack('16sH', b'srini%d', IFF_TUN | IFF_NO_PI)
ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)

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

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

IP_A = "0.0.0.0"
PORT = 9090
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))
```

```
root@595373dfc9f1:/# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
03:48:03.104987 IP6 fe80::42:a7ff:feba:9bda.mdns > ff02::fb.mdns: 0 [2q] PTR (QM
)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
03:48:03.105067 IP6 fe80::909f:cdff:fef4:d5c7.mdns > ff02::fb.mdns: 0 [2q] PTR (
QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
03:48:03.106660 ARP, Request who-has server-router.net-192.168.60.0 tell 595373d
fc9f1, length 28
03:48:03.106706 ARP, Reply server-router.net-192.168.60.0 is-at 02:42:c0:a8:3c:0
b (oui Unknown), length 28
03:48:03.106709 IP 595373dfc9f1.53510 > 192.168.1.1.domain: 42020+ PTR? b.f.0.0.
0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.f.f.ip6.arpa. (90)
03:48:08.112609 IP 595373dfc9f1.36498 > 192.168.1.1.domain: 42020+ PTR? b.f.0.0.
0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.f.f.ip6.arpa. (90)
03:48:13.113328 IP 595373dfc9f1.51705 > 192.168.1.1.domain: 39359+ PTR? a.d.b.9.
a.b.e.f.f.f.7.a.2.4.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.e.f.ip6.arpa. (90)
03:48:18.118142 IP 595373dfc9f1.58658 > 192.168.1.1.domain: 39359+ PTR? a.d.b.9.
```

```
root@f8a15b1e2340:/volumes# tun_client.py
Interface Name: srini0
^CTraceback (most recent call last):
  File "./tun_client.py", line 34, in <module>
    packet = os.read(tun, 2048)
KeyboardInterrupt
```

```
root@20b9f2e06236:/volumes# tun_server.py
Interface Name: srini0
10.9.0.5:53528 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:53528 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:53528 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:53528 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:53528 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:53528 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:53528 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:53528 --> 0.0.0.0:9090
```

Ping Host V from Host U, packets have been received at HOST V.

```
root@f8a15b1e2340:/volumes# 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 ---
41 packets transmitted, 0 received, 100% packet loss, time 40962ms
```

Task 5: Handling Traffic in Both Directions

tun_server.py

```
IP_A = "0.0.0.0"
PORT = 9090

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))
SERVER_IP="10.9.0.5"
SERVER_PORT= 9090
```

```
while True:
        # this will block until at least one interface is ready
        ready, _, _ = select.select([sock, tun], [], [])

        for fd in ready:
                if fd is sock:
                        data, (SERVER_IP, SERVER_PORT) = sock.recvfrom(2048)
                        pkt = IP(data)
                        print("From socket <==: {} --> {}".format(pkt.src, pkt.dst))
                        #... (code needs to be added by students) ...
                        os.write(tun, bytes(pkt))

                if fd is tun:
                        packet = os.read(tun, 2048)
                        pkt = IP(packet)
                        print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))
                        #... (code needs to be added by students) ...
                        sock.sendto(packet, (SERVER_IP,SERVER_PORT))
```

tun_client.py

```
os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

SERVER_PORT = 9090
SERVER_IP = "10.9.0.11"

os.system("ip route add 192.168.60.0/24 dev {}".format(ifname))
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

```
while True:
        # this will block until at least one interface is ready
        ready, _, _ = select.select([sock, tun], [], [])
        for fd in ready:
                if fd is sock:
                        data, (SERVER_IP, SERVER_PORT) = sock.recvfrom(2048)
                        pkt = IP(data)
                        print("From socket <==: {} --> {}".format(pkt.src, pkt.dst))
                        os.write(tun, bytes(pkt))

                if fd is tun:
                        packet = os.read(tun, 2048)
                        pkt = IP(packet)
                        print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))
                        #... (code needs to be added by students) ...
                        sock.sendto(packet, (SERVER_IP, SERVER_PORT))
```

Host U ping Host V

```
root@f8a15b1e2340:/# ping 192.168.60.5 -c 2
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=5.45 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=3.20 ms

--- 192.168.60.5 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
```

Tunnel is created

```
root@f8a15b1e2340:/volumes# tun_client5.py
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.5
```

```
root@20b9f2e06236:/volumes# tun_server5.py
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99
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 socket <==: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99
```

Telnet connection between Host U and Host V

```
root@f8a15b1e2340:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
595373dfc9f1 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Mon Mar  6 19:29:58 UTC 2023 on pts/1
```

- HOST-U send to SERVER
- SERVER Send a ping request to HOST-V
- HOST-V Reply SERVER to a ping request for
- ERVER send a reply HOST-U

TASK 6: Tunnel-Breaking Experiment

Telnet to Host V from Host U, and after that disconnect the connection on tun_server.py program. The telnet program becomes unresponsive.

Anything that we type does not show up, and after that when we try to recreate the server program and on the telnet program the typings are displayed back.

```
Ubuntu 20.04.1 LTS
595373dfc9f1 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Mon Mar  6 19:43:17 UTC 2023 on pts/3
seed@595373dfc9f1:~$
seed@595373dfc9f1:~$
seed@595373dfc9f1:~$
seed@595373dfc9f1:~$
seed@595373dfc9f1:~$ test
seed@595373dfc9f1:~$ ???
-bash: ???: command not found
seed@595373dfc9f1:~$ test1
-bash: test1: command not found
```