

Name: Kwa Li Ying  
Student ID: 1003833

## 50.020 Network Security Lab 6: VPN Tunneling

### Task 1: Network Setup

#### IP Addresses Setup

IP Address of Host U / VPN client: 10.0.2.128 (ens33)

IP Addresses of VPN server: 10.0.2.129 (ens33)  
192.168.60.128 (ens38)

IP Address of Host V: 192.168.60.129 (ens33)

#### NAT Network Adaptor

The NAT Network Adaptor (VMnet2) is used for the VPN client and VPN server. Their network configurations are as shown:

Device	Summary
Memory	4 GB
Processors	2
Hard Disk (SCSI)	20 GB
CD/DVD (SATA)	Auto detect
Network Adapter	Custom (VMnet2)
USB Controller	Present
Sound Card	Auto detect
Printer	Present
Display	Auto detect

Device status

☒ Connected

☒ Connect at power on

Network connection

☐ Bridged: Connected directly to the physical network  
☐ Replicate physical network connection state

☐ NAT: Used to share the host's IP address

☐ Host-only: A private network shared with the host

☒ Custom: Specific virtual network

VMnet2 (NAT)

☐ LAN segment:

This LAN within the NAT Network simulates the Internet. The VPN client and server are given the IP addresses 10.0.2.128 (interface ens33) and 10.0.2.129 (interface ens33) respectively.

To test their connectivity, we ping the VPN server from the VPN client:

```
[03/28/21]seed@VM:~$ ping 10.0.2.129
PING 10.0.2.129 (10.0.2.129) 56(84) bytes of data:
64 bytes from 10.0.2.129: icmp_seq=1 ttl=64 time=1.59 ms
64 bytes from 10.0.2.129: icmp_seq=2 ttl=64 time=1.60 ms
64 bytes from 10.0.2.129: icmp_seq=3 ttl=64 time=1.46 ms
64 bytes from 10.0.2.129: icmp_seq=4 ttl=64 time=0.758 ms
64 bytes from 10.0.2.129: icmp_seq=5 ttl=64 time=1.53 ms
^C
--- 10.0.2.129 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4008ms
rtt min/avg/max/mdev = 0.758/1.389/1.607/0.321 ms
```

Name: Kwa Li Ying  
Student ID: 1003833

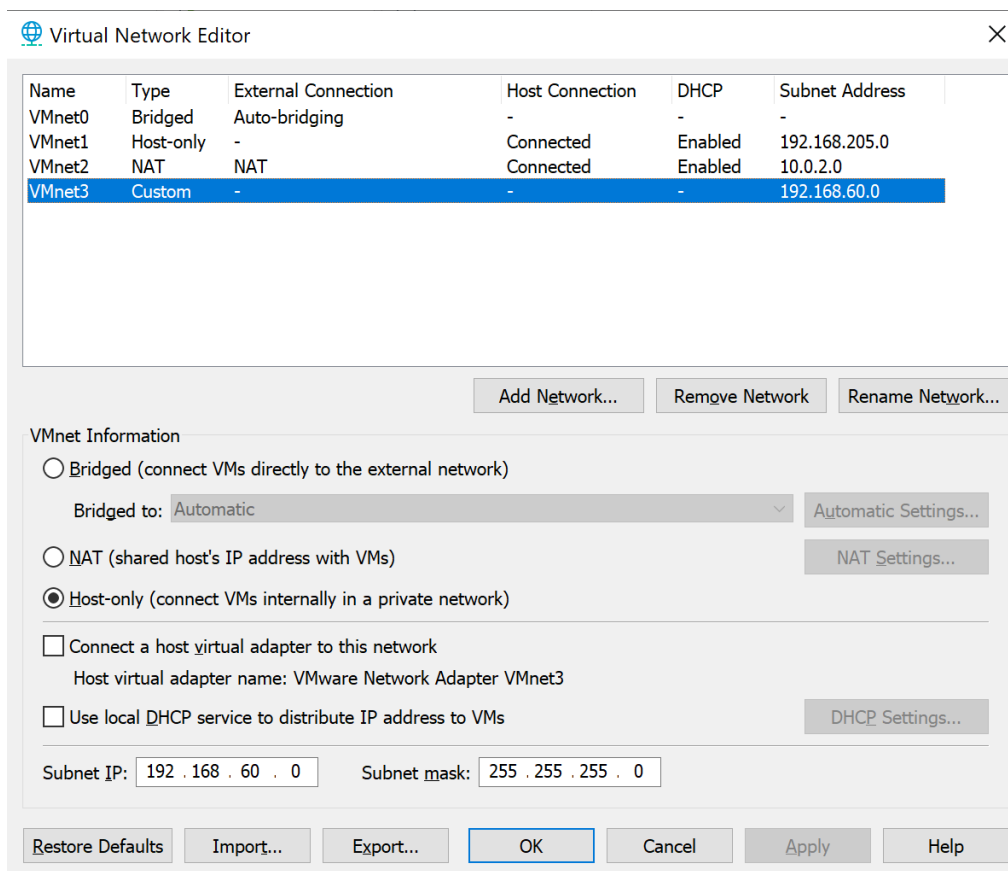
and vice versa:

```
[03/28/21]seed@VM:~$ ping 10.0.2.128
PING 10.0.2.128 (10.0.2.128) 56(84) bytes of data.
64 bytes from 10.0.2.128: icmp_seq=1 ttl=64 time=0.485 ms
64 bytes from 10.0.2.128: icmp_seq=2 ttl=64 time=1.54 ms
64 bytes from 10.0.2.128: icmp_seq=3 ttl=64 time=1.60 ms
64 bytes from 10.0.2.128: icmp_seq=4 ttl=64 time=1.07 ms
64 bytes from 10.0.2.128: icmp_seq=5 ttl=64 time=1.53 ms
^C
--- 10.0.2.128 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4018ms
rtt min/avg/max/mdev = 0.485/1.249/1.607/0.427 ms
```

## Internal Network

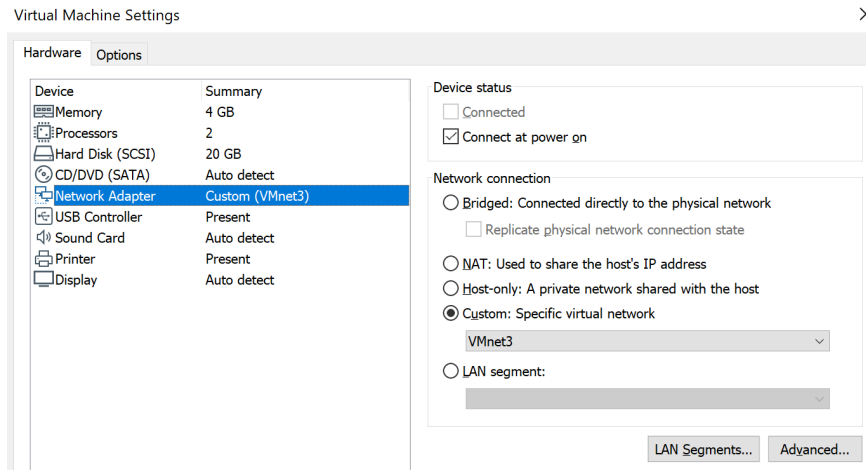
Unfortunately for the hypervisor that I am using (VMWare Workstation), it does not have an “Internal Network” option for network adaptors like there is in VirtualBox. However, there is a way to configure an Internal Network using the options given. This can be done by either creating a custom Virtual Interface or by creating a LAN Segment (reference: <https://techgenix.com/understanding-virtual-networking-vmware-workstation-9/>). The former method will be used here.

The Virtual Network Editor is opened and a new virtual network is created (VMnet3). It is configured to be a Custom network (Host-only is selected but the host virtual adapter option and DHCP option are unselected, removing the host connectivity) and the subnet 192.168.60.0/24 is assigned to it:

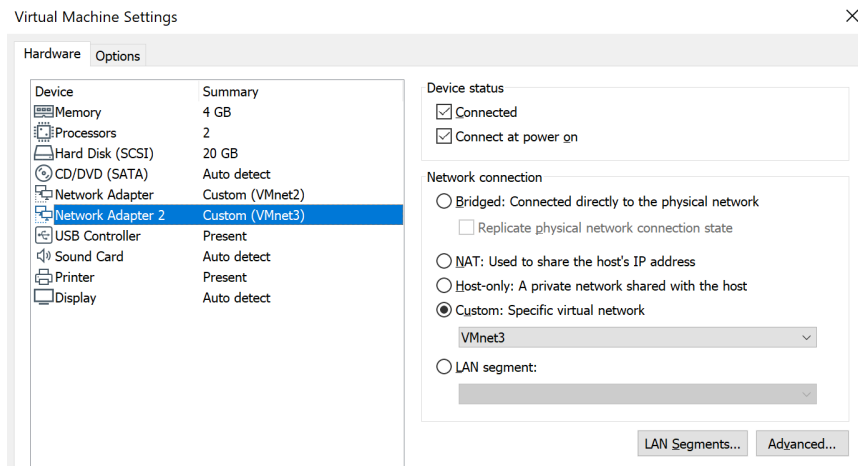


Name: Kwa Li Ying  
Student ID: 1003833

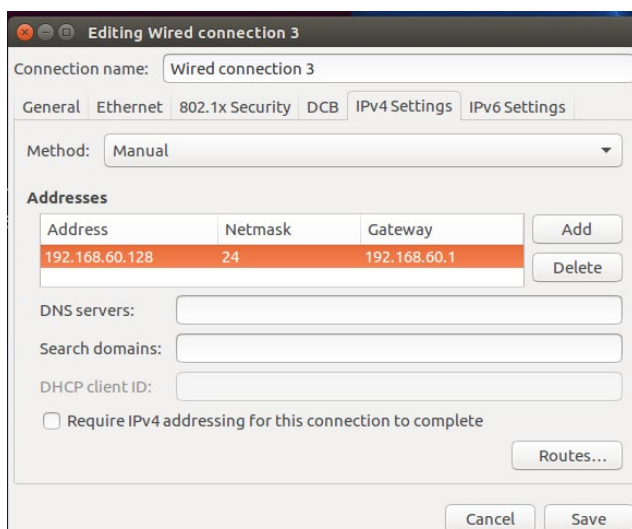
VMnet3 is then specified as the Network Adapter for Host V:



As well as the SECOND Network Adapter for the VPN server:

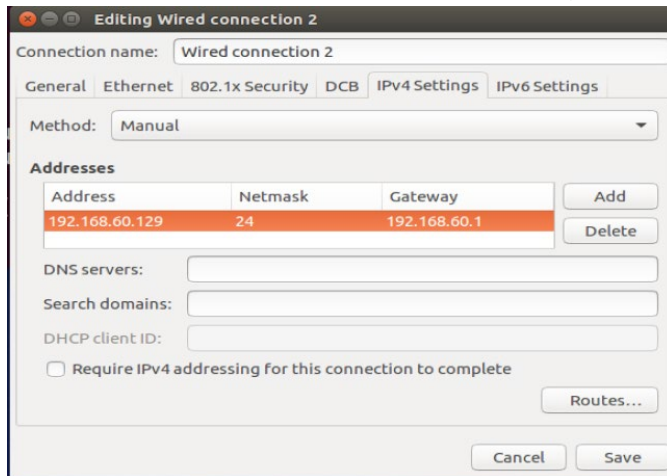


The IP address is set to be 192.168.60.128 (interface ens38) for the VPN server:



Name: Kwa Li Ying  
Student ID: 1003833

And the IP address is set to be 192.168.60.129 (interface ens33) for Host V:



To test their connectivity, we ping Host V from the VPN server:

```
[03/28/21]seed@VM:~$ ping 192.168.60.129
PING 192.168.60.129 (192.168.60.129) 56(84) bytes of data.
64 bytes from 192.168.60.129: icmp_seq=1 ttl=64 time=0.731 ms
64 bytes from 192.168.60.129: icmp_seq=2 ttl=64 time=0.899 ms
64 bytes from 192.168.60.129: icmp_seq=3 ttl=64 time=0.951 ms
64 bytes from 192.168.60.129: icmp_seq=4 ttl=64 time=1.07 ms
64 bytes from 192.168.60.129: icmp_seq=5 ttl=64 time=1.70 ms
^C
--- 192.168.60.129 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4027ms
rtt min/avg/max/mdev = 0.731/1.072/1.707/0.335 ms
```

and vice versa:

```
[03/28/21]seed@VM:~$ ping 192.168.60.128
PING 192.168.60.128 (192.168.60.128) 56(84) bytes of data.
64 bytes from 192.168.60.128: icmp_seq=1 ttl=64 time=0.686 ms
64 bytes from 192.168.60.128: icmp_seq=2 ttl=64 time=0.977 ms
64 bytes from 192.168.60.128: icmp_seq=3 ttl=64 time=1.48 ms
64 bytes from 192.168.60.128: icmp_seq=4 ttl=64 time=1.08 ms
64 bytes from 192.168.60.128: icmp_seq=5 ttl=64 time=1.64 ms
^C
--- 192.168.60.128 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4018ms
rtt min/avg/max/mdev = 0.686/1.176/1.645/0.348 ms
```

To confirm that Host U cannot communicate with Host V, we ping Host V from Host U:

```
[03/28/21]seed@VM:~$ ping 192.168.60.129
PING 192.168.60.129 (192.168.60.129) 56(84) bytes of data.
^C
--- 192.168.60.129 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4075ms
```

and vice versa:

```
[03/28/21]seed@VM:~$ ping 10.0.2.128
PING 10.0.2.128 (10.0.2.128) 56(84) bytes of data.
From 192.168.60.129 icmp_seq=1 Destination Host Unreachable
From 192.168.60.129 icmp_seq=2 Destination Host Unreachable
From 192.168.60.129 icmp_seq=3 Destination Host Unreachable
^C
--- 10.0.2.128 ping statistics ---
6 packets transmitted, 0 received, +3 errors, 100% packet loss, time 5100ms
pipe 4
```

Name: Kwa Li Ying  
Student ID: 1003833

## Task 2: Create and Configure TUN Interface

### Task 2a: Name of the Interface

The tun.py code is copied from the template and edited to give `task2a.py` as shown:

```
#!/usr/bin/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 tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'kwa%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)
```

The code is edited to have my last name 'kwa' as the prefix of the interface name instead of 'tun'.

The code is run on Host U with root privileges:

```
[03/28/21]seed@VM:~/.../task2a$ sudo ./task2a.py
Interface Name: kwa0
```

Another terminal is opened and all the interfaces are printed using the command 'ip address':

```
[03/28/21]seed@VM:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    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
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN group default qlen 1000
    link/ether 00:0c:29:49:f7:51 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.128/24 brd 10.0.2.255 scope global dynamic ens33
        valid_lft 997sec preferred_lft 997sec
    inet6 fe80::235b:c8be:e42f:630f/64 scope link
        valid_lft forever preferred_lft forever
4: kwa0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group default qlen 500
    link/none
```

We have successfully edited the name of the interface to be kwa0.

Name: Kwa Li Ying  
Student ID: 1003833

## Task 2b: Set up the TUN Interface

The following lines are added to task2a.py to produce **task2b.py** before the infinite loop so that the configuration of the TUN interface can be automatically performed by the program:

```
# Configure and set up the interface
os.system("ip addr add 192.168.53.99/24 dev {}".format(iframe))
os.system("ip link set dev {} up".format(iframe))
```

The program is run on Host U with root privileges:

```
[03/28/21]seed@VM:~/../task2b$ sudo ./task2b.py
Interface Name: kwa0
```

Another terminal is opened and the interfaces are printed:

```
[03/28/21]seed@VM:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    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
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN group default qlen 1000
    link/ether 00:0c:29:49:f7:51 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.128/24 brd 10.0.2.255 scope global dynamic ens33
        valid_lft 1784sec preferred_lft 1784sec
    inet6 fe80::235b:c8be:e42f:630f/64 scope link
        valid_lft forever preferred_lft forever
6: kwa0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN group default qlen 500
    link/none
    inet 192.168.53.99/24 scope global kwa0
        valid_lft forever preferred_lft forever
    inet6 fe80::57e1:4524:55c7:2fc/64 scope link flags 800
        valid_lft forever preferred_lft forever
```

This time, the kwa0 interface reflects the assigned IP address 192.168.53.99/24.

Name: Kwa Li Ying  
Student ID: 1003833

## Task 2c: Read from the TUN Interface

The while loop in task2b.py is replaced with the following code to produce task2c.py:

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

The program is run on Host U as shown:

```
[03/28/21]seed@VM:~/../task2c$ sudo ./task2c.py
Interface Name: kwa0
0.0.0.0 > 88.45.131.0 128 frag:6911 / Padding
0.0.0.0 > 88.45.131.0 128 frag:6911 / Padding
0.0.0.0 > 88.45.131.0 128 frag:6911 / Padding
```

## Experiment 1

On Host U, we ping a host in the 192.168.53.0/24 network. The host chosen is 192.168.53.1:

```
[03/28/21]seed@VM:~$ 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 ---
5 packets transmitted, 0 received, 100% packet loss, time 4097ms
```

The following output is seen on the program:

```
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
```

This is because any packet bound for the 192.168.42.0/24 network would be routed to the kwa0 interface, as seen from the routing table:

```
[03/28/21]seed@VM:~$ route
Kernel IP routing table
Destination    Gateway         Genmask         Flags Metric Ref    Use Iface
default        10.0.2.2        0.0.0.0         UG    100    0      0 ens33
10.0.2.0       *               255.255.255.0   U     100    0      0 ens33
link-local     *               255.255.0.0     U     1000   0      0 ens33
192.168.53.0   *               255.255.255.0   U     0      0      0 kwa0
```

As the program reads packets from the TUN interface, these ICMP ping packets are printed as program output.

Name: Kwa Li Ying  
Student ID: 1003833

## Experiment 2

On Host U, we ping a host in the 192.168.60.0/24 network. The host chosen is 192.168.60.1:

```
[03/28/21]seed@VM:~$ ping 192.168.60.1
PING 192.168.60.1 (192.168.60.1) 56(84) bytes of data.
^C
--- 192.168.60.1 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5111ms
```

The program does not print anything.

This is because packets bound for any destination outside the 192.168.42.0/24 network and the 10.0.2.0/24 network would be routed to the gateway 10.0.2.2, as seen from the routing table:

```
[03/28/21]seed@VM:~$ route
Kernel IP routing table
Destination    Gateway      Genmask      Flags Metric Ref    Use Iface
default        10.0.2.2     0.0.0.0      UG    100    0      0 ens33
10.0.2.0       *           255.255.255.0 U    100    0      0 ens33
link-local     *           255.255.0.0  U    1000   0      0 ens33
192.168.53.0   *           255.255.255.0 U    0      0      0 kwa0
```

As the program only reads packets from the TUN interface, these ICMP ping packets bound for the 192.168.60.0/24 network would not be captured and printed as program output.



Name: Kwa Li Ying  
Student ID: 1003833

## Task 2d: Write to the TUN Interface

### Construct an echo reply packet

The task2c.py program is modified to include the following code in **task2dpart1.py**:

```
while True:
    # Get a packet from the tun interface
    packet = os.read(tun, 2048)
    if True:
        ip = IP(packet)
        icmp = ip/ICMP()
        # If packet is ICMP echo request, write echo reply
        if ip.proto==1 and icmp.type==8:
            newip = IP(src=ip.dst, dst=ip.src) / ICMP(type=0)
            newpkt = newip/ip.payload
            os.write(tun, bytes(newpkt))
```

This code constructs a corresponding echo reply packet and writes it to the TUN interface in the event that the packet read from the TUN interface is an ICMP echo request packet. The conditions for the if-statement are determined by the protocol numbers (stated here:

<https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml>) and the ICMP type numbers (stated here: <https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml>). The protocol number for ICMP is 1 and the ICMP type number for echo (request) is 8. The constructed IP/ICMP packet is also based on the ICMP type numbers, as the specified type is 0 which is echo reply.

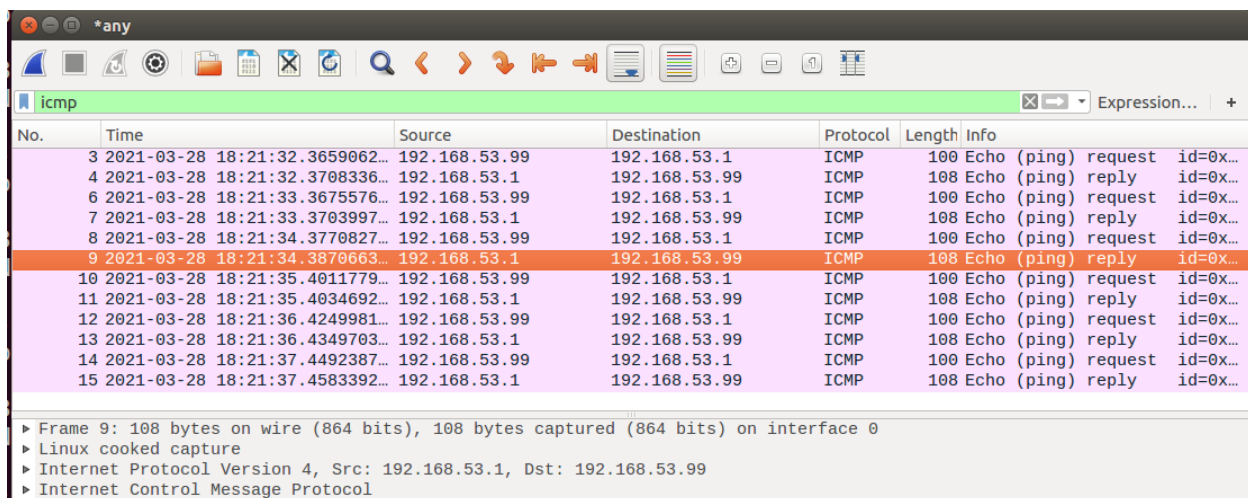
The program is run on Host U as shown:

```
[03/28/21]seed@VM:~/.../task2d$ sudo ./task2dpart1.py
Interface Name: kwa0
```

Another terminal is opened to execute the ping command to 192.168.53.1:

```
[03/28/21]seed@VM:~$ ping 192.168.53.1
PING 192.168.53.1 (192.168.53.1) 56(84) bytes of data.
```

Wireshark is opened and the following packets are captured:



No.	Time	Source	Destination	Protocol	Length	Info
3	2021-03-28 18:21:32.3659062...	192.168.53.99	192.168.53.1	ICMP	100	Echo (ping) request id=0x...
4	2021-03-28 18:21:32.3708336...	192.168.53.1	192.168.53.99	ICMP	108	Echo (ping) reply id=0x...
6	2021-03-28 18:21:33.3675576...	192.168.53.99	192.168.53.1	ICMP	100	Echo (ping) request id=0x...
7	2021-03-28 18:21:33.3703997...	192.168.53.1	192.168.53.99	ICMP	108	Echo (ping) reply id=0x...
8	2021-03-28 18:21:34.3770827...	192.168.53.99	192.168.53.1	ICMP	100	Echo (ping) request id=0x...
9	2021-03-28 18:21:34.3870663...	192.168.53.1	192.168.53.99	ICMP	108	Echo (ping) reply id=0x...
10	2021-03-28 18:21:35.4011779...	192.168.53.99	192.168.53.1	ICMP	100	Echo (ping) request id=0x...
11	2021-03-28 18:21:35.4034692...	192.168.53.1	192.168.53.99	ICMP	108	Echo (ping) reply id=0x...
12	2021-03-28 18:21:36.4249981...	192.168.53.99	192.168.53.1	ICMP	100	Echo (ping) request id=0x...
13	2021-03-28 18:21:36.4349703...	192.168.53.1	192.168.53.99	ICMP	108	Echo (ping) reply id=0x...
14	2021-03-28 18:21:37.4492387...	192.168.53.99	192.168.53.1	ICMP	100	Echo (ping) request id=0x...
15	2021-03-28 18:21:37.4583392...	192.168.53.1	192.168.53.99	ICMP	108	Echo (ping) reply id=0x...

▶ Frame 9: 108 bytes on wire (864 bits), 108 bytes captured (864 bits) on interface 0  
▶ Linux cooked capture  
▶ Internet Protocol Version 4, Src: 192.168.53.1, Dst: 192.168.53.99  
▶ Internet Control Message Protocol

*Name: Kwa Li Ying*

*Student ID: 1003833*

The packet capture reflects the ICMP echo request from the interface (192.168.53.99 to 192.168.53.1) and the ICMP echo reply to the interface (192.168.53.1 to 192.168.53.99). This shows that the code works as expected.

Name: Kwa Li Ying  
Student ID: 1003833

### Write some arbitrary data to the interface

The task2dpart1.py program is modified to include the following code in **task2dpart2.py**:

```
while True:
    # Get a packet from the tun interface
    packet = os.read(tun, 2048)
    if True:
        ip = IP(packet)
        icmp = ip/ICMP()
        # If packet is ICMP echo request, write echo reply
        if ip.proto==1 and icmp.type==8:
            os.write(tun, b'arbitraryDataKwa')
```

The code sends 'arbitraryDataKwa' in bytes to the interface.

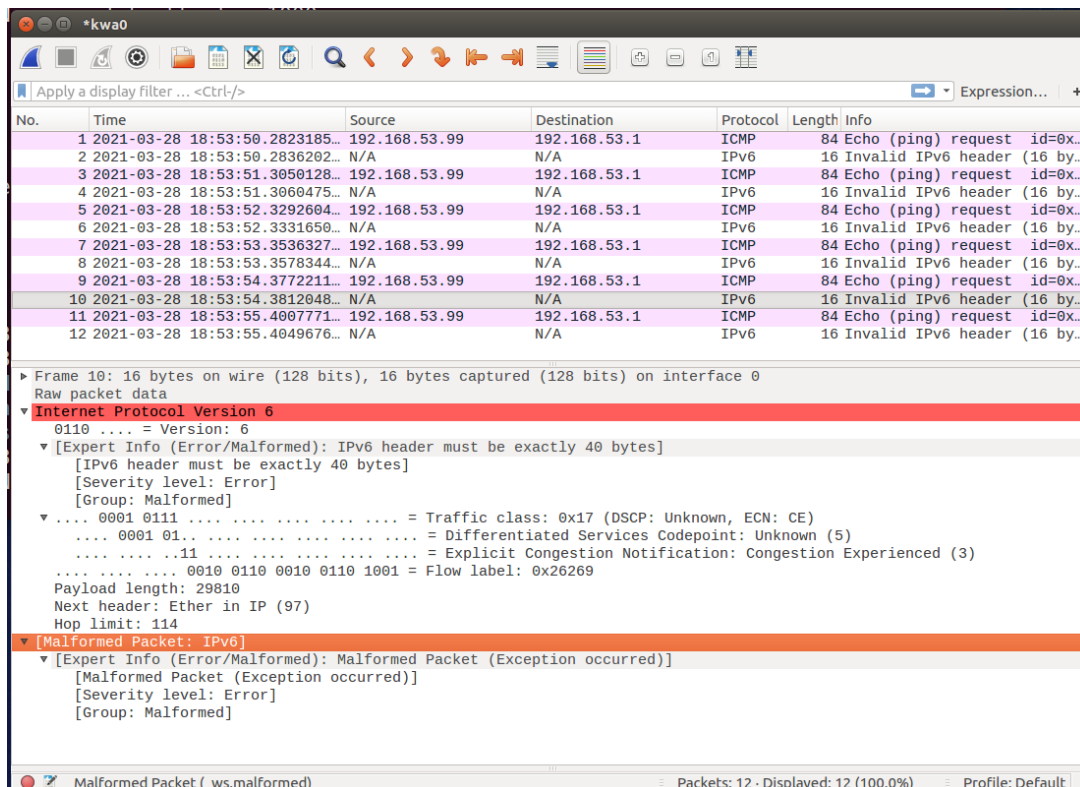
The program is run as shown:

```
[03/28/21]seed@VM:~/.../task2d$ sudo ./task2dpart1.py
Interface Name: kwa0
```

Another terminal is opened to execute the ping command to 192.168.53.1:

```
[03/28/21]seed@VM:~$ ping 192.168.53.1
PING 192.168.53.1 (192.168.53.1) 56(84) bytes of data.
```

Wireshark is opened and the following packets are captured:



The packet capture shows malformed packets replying to the ICMP echo request packets.

Name: Kwa Li Ying  
Student ID: 1003833

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

#### The server program

The following code is saved as `tun_server.py`:

```
#!/usr/bin/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))
```

#### Implement the client program

The TUN program is modified to give the following code as `tun_client.py`:

```
#!/usr/bin/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

SERVER_IP = "10.0.2.129"
SERVER_PORT = 9090

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'kwa%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))

# Configure and set up the interface
os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".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(tun, 2048)
    if True:
        # Send the packet via the tunnel
        sock.sendto(packet, (SERVER_IP, SERVER_PORT))
```

Name: Kwa Li Ying  
Student ID: 1003833

### Testing

The `tun_server.py` program is run on the VPN server:

```
[03/28/21]seed@VM:~/.../task3$ sudo ./tun_server.py
```

The `tun_client.py` program is run on Host U:

```
[03/28/21]seed@VM:~/.../task3$ sudo ./tun_client.py  
Interface Name: kwa0
```

On Host U, the address 192.169.53.1 (belongs to the 192.168.53.0/24 network) is pinged:

```
[03/28/21]seed@VM:~$ ping 192.168.53.1  
PING 192.168.53.1 (192.168.53.1) 56(84) bytes of data.
```

The following output is printed on the VPN Server:

```
[03/28/21]seed@VM:~/.../task3$ sudo ./tun_server.py  
10.0.2.128:36904 --> 0.0.0.0:9090  
  Inside: 192.168.53.99 --> 192.168.53.1  
10.0.2.128:36904 --> 0.0.0.0:9090  
  Inside: 192.168.53.99 --> 192.168.53.1  
10.0.2.128:36904 --> 0.0.0.0:9090  
  Inside: 192.168.53.99 --> 192.168.53.1  
10.0.2.128:36904 --> 0.0.0.0:9090  
  Inside: 192.168.53.99 --> 192.168.53.1  
10.0.2.128:36904 --> 0.0.0.0:9090  
  Inside: 192.168.53.99 --> 192.168.53.1  
10.0.2.128:36904 --> 0.0.0.0:9090
```

The first line of output (10.0.2.128:36904 --> 0.0.0.0:9090) shows that the packet received at the VPN Server at port 9090 has a source of IP address 10.0.2.128 (Host U) and port 36904. This is to be expected as the packet read from the TUN interface at Host U is encapsulated and then forwarded to port 9090 on the VPN server. Therefore the outer packet reflects the IP addresses of Host U and the VPN Server.

The second line of output (Inside: 192.168.53.99 --> 192.168.53.1) shows that the encapsulated payload of the received IP packet is an IP packet itself that has a source IP address 192.168.53.99 (the TUN interface of Host A) and destination IP address 192.168.53.1.

Name: Kwa Li Ying  
Student ID: 1003833

On Host U, we try to ping Host V:

```
[03/28/21]seed@VM:~$ ping 192.168.60.129
PING 192.168.60.129 (192.168.60.129) 56(84) bytes of data.
```

The ICMP packet is not sent to the VPN Server as there is no output shown from the program running on the VPN Server:

```
[03/28/21]seed@VM:~/.../task3$ sudo ./tun_server.py
```

The problem is that only packets bound for the 192.168.53.0/24 network will be sent to the TUN interface, but this packet is bound for 192.168.60.129, which is not part of the network. The following command is run on Host U so that packets going to the 192.168.60.0/24 network will be routed to the TUN interface and be given to the tun\_client.py program:

```
[03/28/21]seed@VM:~$ sudo ip route add 192.168.60.0/24 dev kwa0 via 192.168.53.99
```

To check that the routing was configured properly, the following command is executed on Host U to view the routing table:

```
[03/28/21]seed@VM:~$ route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default 10.0.2.2 0.0.0.0 UG 100 0 0 ens33
10.0.2.0 * 255.255.255.0 U 100 0 0 ens33
link-local * 255.255.0.0 U 1000 0 0 ens33
192.168.53.0 * 255.255.255.0 U 0 0 0 kwa0
192.168.60.0 192.168.53.99 255.255.255.0 UG 0 0 0 kwa0
```

Following this, we try to ping Host V from Host U again:

```
[03/28/21]seed@VM:~$ ping 192.168.60.129
PING 192.168.60.129 (192.168.60.129) 56(84) bytes of data.
```

This time, the program on the VPN server prints the following output:

```
[03/28/21]seed@VM:~/.../task3$ sudo ./tun_server.py
10.0.2.128:36904 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.129
10.0.2.128:36904 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.129
10.0.2.128:36904 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.129
10.0.2.128:36904 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.129
```

This shows that the ICMP packets bound for the 192.168.60.0/24 network are not received by the tun\_server.py program through the tunnel.

Name: Kwa Li Ying  
Student ID: 1003833

## Task 4: Set Up the VPN Server

The tun\_server.py file is modified and the new tun\_server.py code can be found in the task4 folder. The new tun\_server.py has the following contents:

```
#!/usr/bin/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

IP_A = "0.0.0.0"
PORT = 9090

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'kwa%d' % 0, 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))

# Configure and set up the interface
os.system("ip addr add 192.168.53.100/24 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))
    # Send packet to tunnel interface
    os.write(tun, bytes(pkt))
```

The modified tun\_server.py code, similar to the tun\_client.py code, carries out the following extra steps:

- Creates a TUN interface called kwa0
- Gets the data from the socket interface and treats the received data as an IP packet
- Writes this IP packet to the TUN interface

Note that this new TUN interface has an IP address of 192.168.53.100, which is different from that of Host U's TUN interface

IP forwarding is enabled on the VPN Server to forward packets between the private network and the tunnel, so that it behaves like a gateway:

```
[03/28/21]seed@VM:~/.../task4$ sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
```

Name: Kwa Li Ying  
Student ID: 1003833

## Testing

The `tun_server.py` program is run on the VPN server:

```
[03/28/21]seed@VM:~/.../task4$ sudo ./tun_server.py
Interface Name: kwa0
```

The `tun_client.py` program is run on Host U:

```
[03/28/21]seed@VM:~/.../task3$ sudo ./tun_client.py
Interface Name: kwa0
```

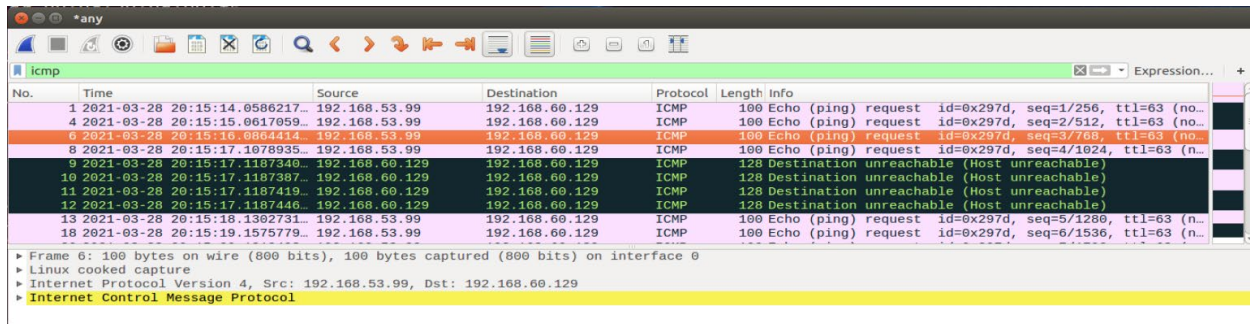
On Host U, the following command is executed to set up the proper routing for packets bound to the 192.168.60.0/24 network:

```
[03/28/21]seed@VM:~$ sudo ip route add 192.168.60.0/24 dev kwa0 via 192.168.53.99
[03/28/21]seed@VM:~$ route
Kernel IP routing table
Destination    Gateway         Genmask         Flags Metric Ref    Use Iface
default        10.0.2.2        0.0.0.0         UG    100    0      0 ens33
10.0.2.0       *               255.255.255.0   U     100    0      0 ens33
link-local     *               255.255.0.0     U     1000   0      0 ens33
192.168.53.0   *               255.255.255.0   U     0      0      0 kwa0
192.168.60.0   192.168.53.99   255.255.255.0   UG    0      0      0 kwa0
```

On Host U, the address 192.169.53.1 (belongs to the 192.168.53.0/24 network) is pinged:

```
[03/28/21]seed@VM:~$ ping 192.168.53.1
PING 192.168.53.1 (192.168.53.1) 56(84) bytes of data.
```

On Host V, Wireshark is opened and the following packets are captured:



No.	Time	Source	Destination	Protocol	Length	Info
1	2021-03-28 20:15:14.0586217...	192.168.53.99	192.168.60.129	ICMP	100	Echo (ping) request id=0x297d, seq=1/256, ttl=63 (no...
4	2021-03-28 20:15:15.0617059...	192.168.53.99	192.168.60.129	ICMP	100	Echo (ping) request id=0x297d, seq=2/512, ttl=63 (no...
6	2021-03-28 20:15:16.0864414...	192.168.53.99	192.168.60.129	ICMP	100	Echo (ping) request id=0x297d, seq=3/768, ttl=63 (no...
8	2021-03-28 20:15:17.1078935...	192.168.53.99	192.168.60.129	ICMP	100	Echo (ping) request id=0x297d, seq=4/1024, ttl=63 (n...
9	2021-03-28 20:15:17.1187340...	192.168.60.129	192.168.60.129	ICMP	128	Destination unreachable (Host unreachable)
10	2021-03-28 20:15:17.1187387...	192.168.60.129	192.168.60.129	ICMP	128	Destination unreachable (Host unreachable)
11	2021-03-28 20:15:17.1187419...	192.168.60.129	192.168.60.129	ICMP	128	Destination unreachable (Host unreachable)
12	2021-03-28 20:15:17.1187446...	192.168.60.129	192.168.60.129	ICMP	128	Destination unreachable (Host unreachable)
13	2021-03-28 20:15:18.1302731...	192.168.53.99	192.168.60.129	ICMP	100	Echo (ping) request id=0x297d, seq=5/1280, ttl=63 (n...
18	2021-03-28 20:15:19.1575779...	192.168.53.99	192.168.60.129	ICMP	100	Echo (ping) request id=0x297d, seq=6/1536, ttl=63 (n...

Frame 6: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface 0  
Linux cooked capture  
Internet Protocol Version 4, Src: 192.168.53.99, Dst: 192.168.60.129  
Internet Control Message Protocol

This packet capture shows that the ICMP packets have arrived at Host V.



Name: Kwa Li Ying  
Student ID: 1003833

## Task 5: Handling Traffic in Both Directions

### Modifying the TUN client program

The new `tun_client.py` program is in the folder task5 and is as shown:

```
#!/usr/bin/python3
import fcntl
import struct
import os
import time
from scapy.all import *
from select import select

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

IP_A = "10.0.2.128"
PORT = 8080

SERVER_IP = "10.0.2.129"
SERVER_PORT = 9090

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'kwa%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))

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

# Create UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))

while True:

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

    for fd in ready:

        if fd is sock:
            data, (ip, port) = sock.recvfrom(2048)
            pkt = IP(data)
            print("From socket <==: {} --> {}".format(pkt.src, pkt.dst))
            # Send packet to tunnel interface
            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))
            # Send the packet via the tunnel
            sock.sendto(packet, (SERVER_IP, SERVER_PORT))
```

Name: Kwa Li Ying  
Student ID: 1003833

### Modifying the TUN server program

The new `tun_server.py` program is in the folder task5 and is as shown:

```
#!/usr/bin/python3
import fcntl
import struct
import os
import time
from scapy.all import *
from select import select

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

IP_A = "10.0.2.129"
PORT = 9090

CLIENT_IP = "10.0.2.128"
CLIENT_PORT = 8080

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'kwa%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))

# Configure and set up the interface
os.system("ip addr add 192.168.53.100/24 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:
    # this will block until at least one interface is ready
    ready, _, _ = select([sock, tun], [], [])

    for fd in ready:
        if fd is sock:
            data, (ip, port) = sock.recvfrom(2048)
            pkt = IP(data)
            print("From socket <==: {} --> {}".format(pkt.src, pkt.dst))
            # Send packet to tunnel interface
            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))
            # Send the packet via the tunnel
            sock.sendto(packet, (CLIENT_IP, CLIENT_PORT))
```

Name: Kwa Li Ying  
Student ID: 1003833

## Testing

The `tun_client.py` program is run on Host U:

```
[03/28/21]seed@VM:~/.../task5$ sudo ./tun_client.py
Interface Name: kwa0
From tun ==>: 0.0.0.0 --> 158.60.35.41
From tun ==>: 0.0.0.0 --> 158.60.35.41
From tun ==>: 0.0.0.0 --> 158.60.35.41
```

The `tun_server.py` program is run on the VPN Server:

```
[03/28/21]seed@VM:~/.../task5$ sudo ./tun_server.py
Interface Name: kwa0
From tun ==>: 0.0.0.0 --> 181.141.117.129
From tun ==>: 0.0.0.0 --> 181.141.117.129
From tun ==>: 0.0.0.0 --> 181.141.117.129
```

On Host U, the following command is executed to set up the proper routing for packets bound to the 192.168.60.0/24 network:

```
[03/28/21]seed@VM:~$ sudo ip route add 192.168.60.0/24 dev kwa0 via 192.168.53.99
[03/28/21]seed@VM:~$ route
Kernel IP routing table
Destination    Gateway         Genmask         Flags Metric Ref    Use Iface
default        10.0.2.2        0.0.0.0         UG    100    0      0 ens33
10.0.2.0        *               255.255.255.0   U     100    0      0 ens33
link-local      *               255.255.0.0     U     1000   0      0 ens33
192.168.53.0    *               255.255.255.0   U     0      0      0 kwa0
192.168.60.0    192.168.53.99  255.255.255.0   UG    0      0      0 kwa0
```

On Host V, the following command is executed to set up the proper routing for packets bound to the 192.168.53.0/24 network:

```
[03/28/21]seed@VM:~$ sudo route add -net 192.168.53.0/24 gw 192.168.60.128 ens33
[03/28/21]seed@VM:~$ route
Kernel IP routing table
Destination    Gateway         Genmask         Flags Metric Ref    Use Iface
default        192.168.60.1    0.0.0.0         UG    100    0      0 ens33
link-local      *               255.255.0.0     U     1000   0      0 ens33
192.168.53.0    192.168.60.128 255.255.255.0   UG    0      0      0 ens33
192.168.60.0    *               255.255.255.0   U     100    0      0 ens33
```

On the VPN Server, IP forwarding is enabled to forward packets between the private network and the tunnel, so that it behaves like a gateway:

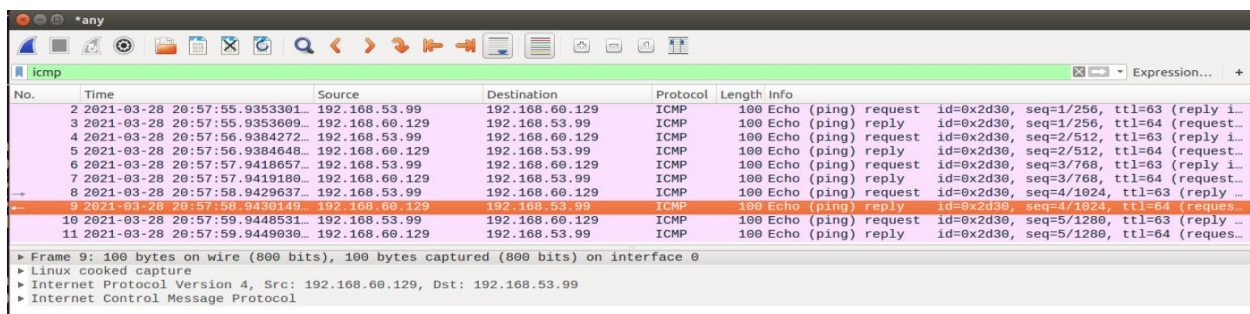
```
[03/28/21]seed@VM:~/.../task4$ sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
```

Name: Kwa Li Ying  
Student ID: 1003833

Finally, we ping Host V from Host U:

```
[03/28/21]seed@VM:~$ ping 192.168.60.129
PING 192.168.60.129 (192.168.60.129) 56(84) bytes of data.
64 bytes from 192.168.60.129: icmp_seq=1 ttl=63 time=4.56 ms
64 bytes from 192.168.60.129: icmp_seq=2 ttl=63 time=6.28 ms
64 bytes from 192.168.60.129: icmp_seq=3 ttl=63 time=8.21 ms
64 bytes from 192.168.60.129: icmp_seq=4 ttl=63 time=9.36 ms
64 bytes from 192.168.60.129: icmp_seq=5 ttl=63 time=6.88 ms
^C
--- 192.168.60.129 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4008ms
rtt min/avg/max/mdev = 4.563/7.062/9.363/1.644 ms
```

The Wireshark packet capture on Host V also shows that Host V successfully responds to Host U with echo reply packets:



No.	Time	Source	Destination	Protocol	Length	Info
2	2021-03-28 20:57:55.9353361	192.168.53.99	192.168.60.129	ICMP	100	Echo (ping) request id=0x2d30, seq=1/256, ttl=63 (reply i...
3	2021-03-28 20:57:55.9353609	192.168.60.129	192.168.53.99	ICMP	100	Echo (ping) reply id=0x2d30, seq=1/256, ttl=64 (request...
4	2021-03-28 20:57:56.9384272	192.168.53.99	192.168.60.129	ICMP	100	Echo (ping) request id=0x2d30, seq=2/512, ttl=63 (reply i...
5	2021-03-28 20:57:56.9384648	192.168.60.129	192.168.53.99	ICMP	100	Echo (ping) reply id=0x2d30, seq=2/512, ttl=64 (request...
6	2021-03-28 20:57:57.9418657	192.168.53.99	192.168.60.129	ICMP	100	Echo (ping) request id=0x2d30, seq=3/768, ttl=63 (reply i...
7	2021-03-28 20:57:57.9419180	192.168.60.129	192.168.53.99	ICMP	100	Echo (ping) reply id=0x2d30, seq=3/768, ttl=64 (request...
8	2021-03-28 20:57:58.9429637	192.168.53.99	192.168.60.129	ICMP	100	Echo (ping) request id=0x2d30, seq=4/1024, ttl=63 (reply ...
9	2021-03-28 20:57:58.9430149	192.168.60.129	192.168.53.99	ICMP	100	Echo (ping) reply id=0x2d30, seq=4/1024, ttl=64 (reques...
10	2021-03-28 20:57:59.9448531	192.168.53.99	192.168.60.129	ICMP	100	Echo (ping) request id=0x2d30, seq=5/1280, ttl=63 (reply ...
11	2021-03-28 20:57:59.9449030	192.168.60.129	192.168.53.99	ICMP	100	Echo (ping) reply id=0x2d30, seq=5/1280, ttl=64 (reques...

Frame 9: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface 0  
Linux cooked capture  
Internet Protocol Version 4, Src: 192.168.60.129, Dst: 192.168.53.99  
Internet Control Message Protocol

We telnet from Host U into Host V:

```
[03/28/21]seed@VM:~$ telnet 192.168.60.129
Trying 192.168.60.129...
Connected to 192.168.60.129.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Mon Mar 15 17:49:37 EDT 2021 from 10.0.2.129 on pts/18
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

1 package can be updated.
0 updates are security updates.
```

Since the telnet and ping from Host U to Host V receives echo responses, the VPN is successfully set up.

(Next page for packet flow)

*Name: Kwa Li Ying*  
*Student ID: 1003833*

Packet flow (Host U to Host V):

- Host U ping/telnet application program to Host U TUN interface kwa0 (192.168.53.99)
- Host U TUN interface kwa0 to Host U TUN application socket (10.0.2.128:8080)
- Host U TUN application socket to VPN Server TUN application socket (10.0.2.129:9090)
- VPN Server TUN application socket to VPN Server TUN interface kwa0 (192.168.53.100)
- VPN Server TUN interface kwa0 to VPN Server Internal Network Adapter ens38 (192.168.60.128)
- VPN Server Internal Network Adapter ens38 to Host V Internal Network Adapter ens33 (192.168.60.129)
- Host V Internal Network Adapter ens33 to Host V ping/telnet application program

Packet flow (Host V to Host U):

- Host V ping/telnet application program to Host V Internal Network Adapter ens33 (192.168.60.129)
- Host V Internal Network Adapter ens33 to VPN Server Internal Network Adapter ens38 (192.168.60.128)
- VPN Server Internal Network Adapter ens38 to VPN Server TUN interface kwa0 (192.168.53.100)
- VPN Server TUN interface kwa0 to VPN Server TUN application socket (10.0.2.129:9090)
- VPN Server TUN application socket to Host U TUN application socket (10.0.2.128:8080)
- Host U TUN application socket to Host U TUN interface kwa0 (192.168.53.99)
- Host U TUN interface kwa0 to Host U ping/telnet application program

Name: Kwa Li Ying  
Student ID: 1003833

## Task 6: Tunnel-Breaking Experiment

We telnet from Host U to Host V as shown:

```
[03/28/21]seed@VM:~$ telnet 192.168.60.129
Trying 192.168.60.129...
Connected to 192.168.60.129.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Sun Mar 28 21:02:35 EDT 2021 from 192.168.53.99 on pts/4
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

1 package can be updated.
0 updates are security updates.

[03/28/21]seed@VM:~$ g
```

On the VPN Server, the `tun_server.py` program is stopped by sending the SIG\_INT command (Ctrl+C):

```
From tun ==>: 192.168.60.129 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.129
^C
Traceback (most recent call last):
  File "./tun_server.py", line 40, in <module>
    ready, _, _ = select([sock, tun], [], [])
KeyboardInterrupt
```

We type some keys in the telnet window but nothing is shown. The telnet window appears to freeze.

The telnet connection is not broken. The TCP connection is still there and Host U keeps resending packets, but they cannot be delivered because the tunnel is broken. Whatever we type in the telnet window will be buffered by TCP, but we are not able to see anything.

We now reconnect the VPN tunnel by starting the `tun_server.py` program:

```
[03/28/21]seed@VM:~/.../task5$ sudo ./tun_server.py
Interface Name: kwa0
```

The characters that were typed in the telnet window but were not reflected earlier show up now:

```
[03/28/21]seed@VM:~$ telnet 192.168.60.129
Trying 192.168.60.129...
Connected to 192.168.60.129.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Sun Mar 28 21:02:35 EDT 2021 from 192.168.53.99 on pts/4
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

1 package can be updated.
0 updates are security updates.

[03/28/21]seed@VM:~$ gasd
```

Once we reconnect the tunnel, the lost packets that are resent will finally reach Host V. The telnet connection continues to function from here on.

*Name: Kwa Li Ying*  
*Student ID: 1003833*

Task 7: Routing Experiment on Host V  
<do if there is still time before submission>

Task 8: Experiment with the TUN IP Address  
<do if there is still time before submission>

Task 9: Experiment with the TAP Interface  
<do if there is still time before submission>