

VPN Tunnelling

Parts Copyright © 2020 Wenliang Du (Syracuse University), All rights reserved.

Free to use for non-commercial educational purposes. Commercial uses of the materials are prohibited. The SEED project was funded by multiple grants from the US National Science Foundation.

Parts Copyright © 2021 Jonathan White (UWE Bristol) All rights reserved.

Contents

| | |
|--|----|
| Contents..... | 1 |
| 1 Aims and Objectives..... | 2 |
| 1.1 Related Text | 2 |
| 2 Lab Tasks | 3 |
| 2.1 Background | 3 |
| 2.2 Task 1: Network Setup | 3 |
| 2.2.1 VPN Server Configuration | 5 |
| 2.2.2 Host V Configuration..... | 7 |
| 2.2.3 Testing the Configuration | 8 |
| 2.3 Task 2: Create and Configure the TUN interface | 9 |
| 2.3.1 Name the TUN interface | 10 |
| 2.3.2 Setup the TUN interface | 11 |
| 2.3.3 Read from the TUN interface | 12 |
| 2.3.4 Write to the TUN interface | 13 |
| 2.4 Task 3: Send the IP Packet to the VPN Server through a Tunnel | 15 |
| 2.4.1 TUN Server program | 15 |
| 2.4.2 Implement the client program <code>tun_client.py</code> | 16 |
| 2.5 Task 4: Setup the VPN Server..... | 18 |
| 2.6 Task 5: Handling Traffic in Both Directions | 19 |
| 2.7 Task 6: Tunnel-breaking experiment | 21 |
| 2.8 Encrypting the Tunnel | 21 |
| 2.9 Task 7: Choosing between an SSL/TLS VPN vs IPsec VPN (Research Task) | 22 |
| 2.9.1 Task Requirements..... | 22 |
| 3 Submission | 22 |
| 4 Plagiarism..... | 22 |
| 5 Marking Criteria | 23 |
| 6 Document Revision History..... | 24 |

1 Aims and Objectives

A Virtual Private Network (VPN) is a private network built on top of a public network, usually the Internet. Computers that are part of a VPN can communicate securely, just like if they were on a real private network that is physically isolated from outside, even though their traffic may go through a public network. VPNs also enables employees to securely access a company's intranet while traveling; it also allows companies to expand their private networks to places across the country and around the world.

The objective of this lab is to help students understand how a VPN works. We focus on a specific type of VPN (the most common type), which is built on top of the transport layer. We will build a very simple VPN from scratch and use the process to illustrate how each piece of the VPN technology works.

A real VPN program has two essential pieces, tunnelling, and encryption. This lab only focuses on the tunnelling aspects, helping students understand the tunnelling technology involved in a VPN. The tunnel in this lab will not be encrypted. Tunnels would normally be encrypted using TLS and you can optionally extend the functionality in the report using the OpenSSL libraries to encrypt the data over the tunnel. Encrypting the tunnel is not required for this submission.

This lab covers the following topics:

- Virtual Private Network
- The TUN/TAP virtual interface
- IP tunnelling
- Routing

1.1 Related Text

Detailed coverage of the TUN/TAP virtual interface and how VPNs work can be found in the following:

- Chapter 16 of the SEED Book, Computer & Internet Security: A Hands-on Approach, 2nd Edition, by Wenliang Du. See details at <https://www.handsonsecurity.net>.

More information regarding the Python library Scapy, which is a powerful packet manipulation program can be found in the following:

- Scapy documentation - <https://scapy.readthedocs.io/en/latest/introduction.html>

2 Lab Tasks

2.1 Background

There are three main types of VPN:

- Host-to-Host
- Host-to-Gateway
- Gateway-to-Gateway

Host to Host VPN connects one desktop or workstation to another station by way of a host-to-host connection. This type of connection uses the network to which each host is connected to create a secure tunnel between the two.

Host-To-Gateway VPN is a user-to-internal network connection via a public or shared network. Many large companies have employees that need to connect to the internal network from the field. These field agents access the internal network by using remote computers and laptops without a static IP address.

Gateway-to-Gateway VPN otherwise known as a *Site-to-Site* VPN connects an entire network (such as a LAN or WAN) to a remote network via a network-to-network connection. A network-to-network connection requires routers on each side of the connecting networks to transparently process and route information from one node on a local LAN to another node on a remote LAN.

2.2 Task 1: Network Setup

For this lab we will be creating a Host-to-Gateway VPN using three VMs:

- **VPN Client**, also serving as *Host U*.
- **VPN Server**, acting as the *Gateway*.
- **Host V**, which is a host in the private network.

If you have not done so already, make two more copies of the UWE Ubuntu VM and open them in VMWare. We suggest you change the background image on the *VPN Server* and *Host V* VMs so that you can identify and distinguish between the three VMs easily. You can do this by right clicking on the desktop background and selecting **Change Desktop Background**. You can then select a new Wallpaper or picture from the *Pictures Folder*.

NOTE: Details on how to duplicate the VM and change the background is detailed in the [Lab Setup Guide](#) provided during Block 0 in sections **2.8 Duplicating the VM** and **2.9 Customising the new VM**

The network setup showing example IP addresses (yours are likely to be different) is depicted in Figure 1.

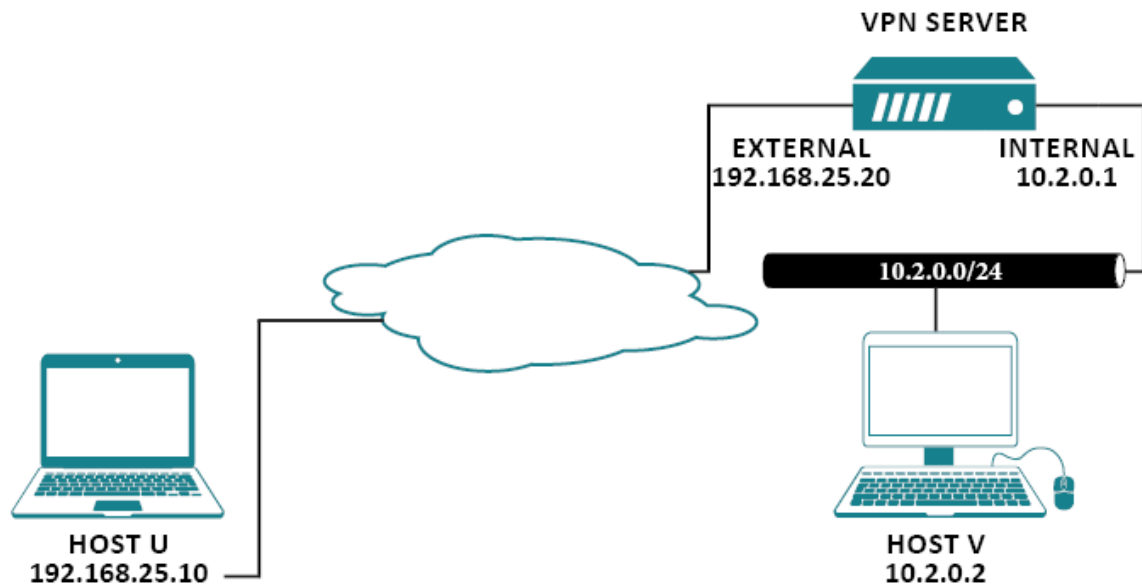


Figure 1: VM setup

By default, the network configuration for the UWE Ubuntu VM has one network adapter on the default NAT network. As you can see from Figure 1, the VPN Server must have a second network connection added which resides on a private network (10.2.0.0/24). Host V, whilst having only one network connection must change its configuration so that its network adapter is only on the private network.

2.2.1 VPN Server Configuration

For the VM that you have designated as the *VPN Server / Gateway* perform the following changes.

- Right click the VM and select **Settings**.
- Add a new *Network Adapter*
- Change the *Network Connection* to be **Custom: Specific virtual network** and set this drop down to **VMNet2**

Once complete, your network settings should look like those shown in Figure 2.

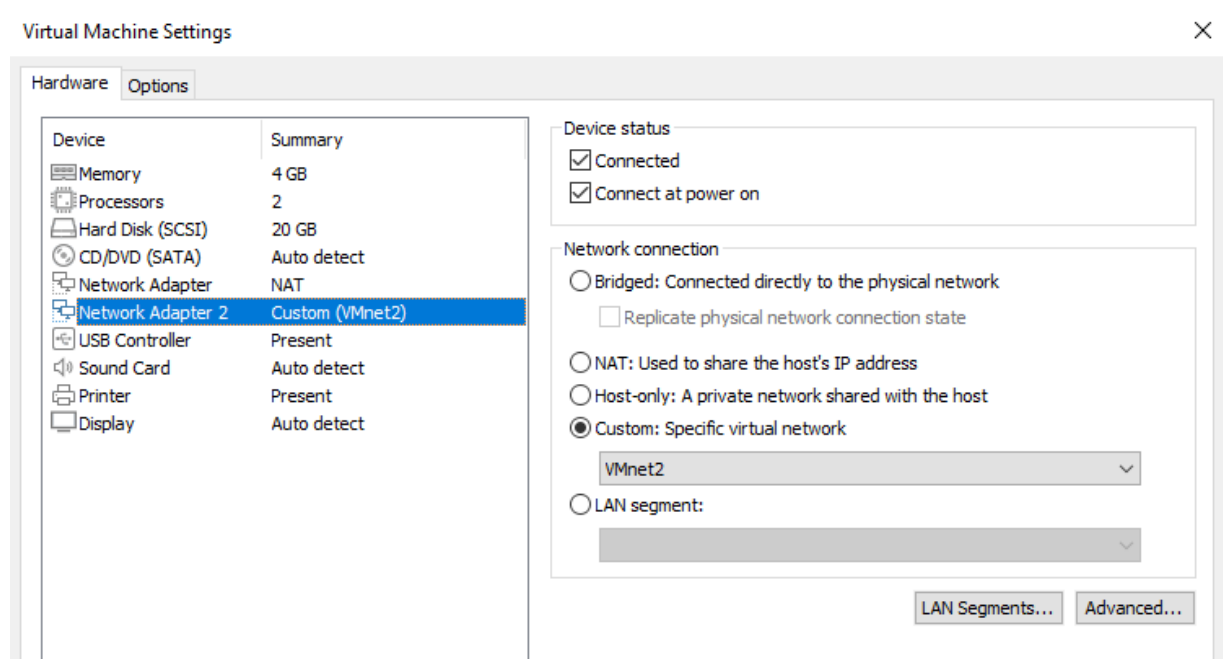


Figure 2: VPN Gateway virtual machine settings

Now that we have defined the network adaptor, we must configure its IP address to be **10.2.0.1**. From a terminal window execute the following command to edit the networking configuration files.

```
$ sudo gedit /etc/network/interfaces
```

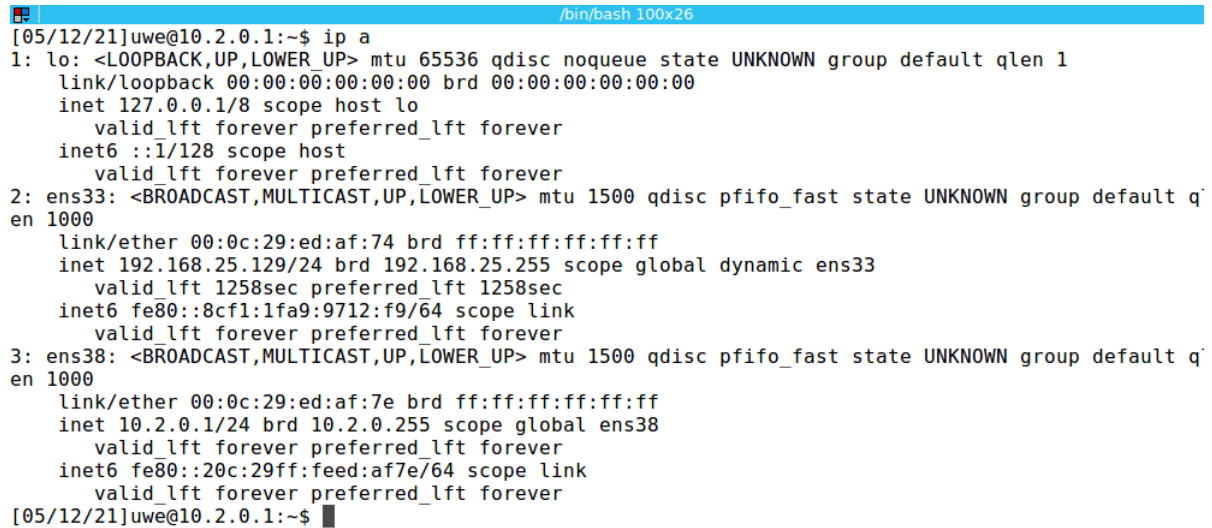
You will need to add the following information to the file to configure the new network adapter interface **ens38**:

```
auto ens38
iface ens38 inet static
    address 10.2.0.1
    netmask 255.255.255.0
    gateway 10.2.0.1
```

Save the file and exit the editor. We now need to restart the networking stack to pick up the changes to the configuration and then verify that the interface has been configured correctly.

Issue the following commands to do this:

```
$ sudo /etc/init.d/networking restart
$ ip address
```



```
[05/12/21]uwe@10.2.0.1:~$ ip a
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 q
en 1000
    link/ether 00:0c:29:ed:af:74 brd ff:ff:ff:ff:ff:ff
    inet 192.168.25.129/24 brd 192.168.25.255 scope global dynamic ens33
        valid_lft 1258sec preferred_lft 1258sec
    inet6 fe80::8cf1:1fa9:9712:f9/64 scope link
        valid_lft forever preferred_lft forever
3: ens38: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN group default q
en 1000
    link/ether 00:0c:29:ed:af:7e brd ff:ff:ff:ff:ff:ff
    inet 10.2.0.1/24 brd 10.2.0.255 scope global ens38
        valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:feed:af7e/64 scope link
        valid_lft forever preferred_lft forever
[05/12/21]uwe@10.2.0.1:~$
```

Figure 3: Verify VPN Gateway IP configuration

As shown in Figure 3, the *ens38* interface has the IP address *10.2.0.1*.

NOTE: You may receive an indication in the command line that the networking restart command has failed. Verify using the “*ip address*” command that the new interface is present, has the correct IP address assigned and the interface is UP.

If this has not worked, reboot your VM and check again. If you are still experiencing issues after a reboot, verify your configuration changes and finally contact your tutor for support.

2.2.2 Host V Configuration

For *Host V* we need to change the existing network adaptor to use **VMNet2** instead of **NAT**.

- Right click the VM and select **Settings**.
- Select the **Network Adapter**
- Change the **Network Connection** to be **Custom: Specific virtual network** and set this drop down to **VMNet2**

Once complete your network settings should look like those in Figure 4.

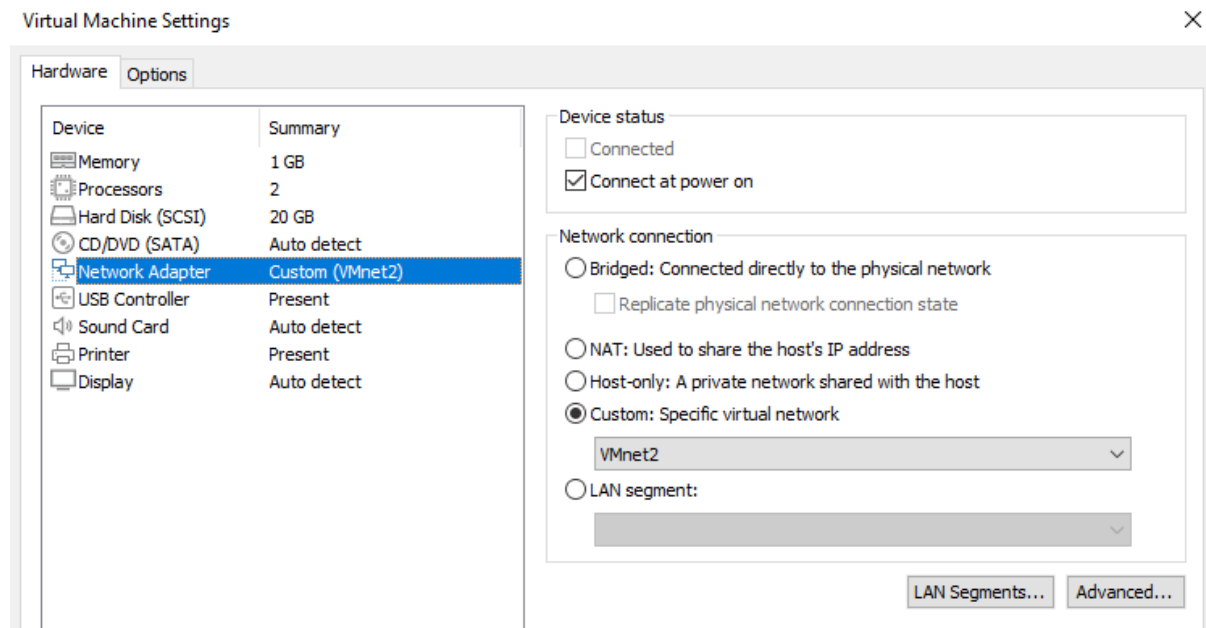


Figure 4: Host V Network Settings

Now that we have defined the network adaptor, we must configure its IP address to be **10.2.0.2**. From a terminal window execute the following command to edit the networking configuration files.

```
$ sudo gedit /etc/network/interfaces
```

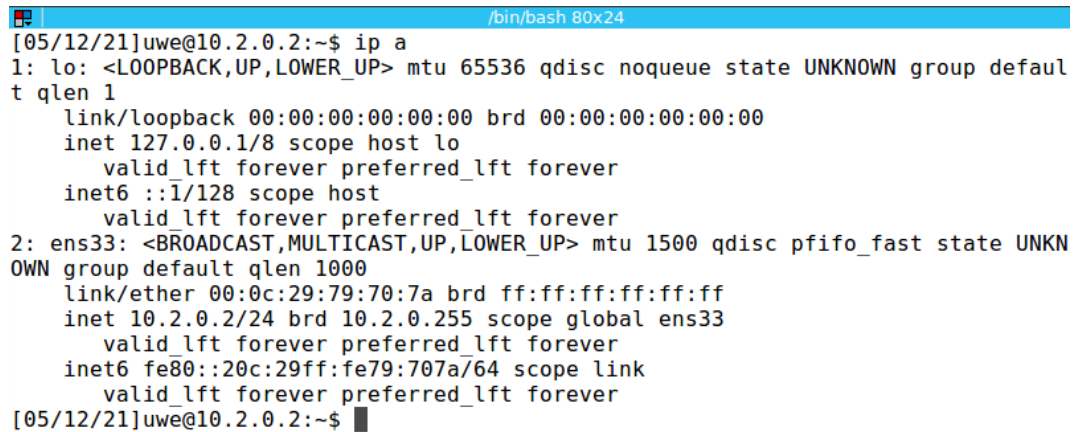
This time we need to modify the existing network interface *ens33*. Note the different interface name. You will need to add the following information to the file to configure the existing network adapter.

```
auto ens33
iface ens33 inet static
    address 10.2.0.2
    netmask 255.255.255.0
    gateway 10.2.0.1
```

Save the file and exit the editor. We now need to restart the networking stack to pick up the changes to the configuration and then verify that the interface has been configured correctly. Issue the following commands to do this:

```
$ sudo /etc/init.d/networking restart
$ ip address
```

Once complete your IP configuration should look the same as Figure 5.



```
[05/12/21]uwe@10.2.0.2:~$ ip a
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:29:79:70:7a brd ff:ff:ff:ff:ff:ff
    inet 10.2.0.2/24 brd 10.2.0.255 scope global ens33
        valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe79:707a/64 scope link
        valid_lft forever preferred_lft forever
[05/12/21]uwe@10.2.0.2:~$
```

Figure 5: Verify Host V IP configuration

2.2.3 Testing the Configuration

Please conduct the following tests to ensure the network setup has been performed correctly:

- The VPN Client can ping the VPN Sever IP address over the NAT network.
- The VPN Server can ping Host V over the private network.
- The VPN Client is not able to pin Host V.

Demonstrate the result of your tests.

2.3 Task 2: Create and Configure the TUN interface

The VPN tunnel that we are going to build is based on the TUN/TAP technologies. TUN and TAP are virtual network kernel drivers; they implement network device that are supported entirely in software. TAP (as in network tap) simulates an Ethernet device, and it operates with layer-2 packets such as Ethernet frames; TUN (as in network TUNnel) simulates a network layer device, and it operates with layer-3 packets such as IP packets. With TUN/TAP, we can create virtual network interfaces.

As the VPN Client/Server application sits in the user space, it makes it difficult to pass full IP packets to and from the application as it lays outside the network stack. One solution to this is to use the virtual kernel network devices TUN/TAP. A TUN device simulates a network layer device (Layer-3) and creates a virtual interface.

A user space application can attach itself to the TUN/TAP interface. Any packets sent by the application to the TUN device will be sent to the kernel and will appear as if they were received by an external network. Packets sent by the kernel to the virtual device will be delivered to the user-space application.

To the operating system, it appears that the packets come from an external source through the virtual network interface. Figure 6 shows an illustration of the packet flow through the TUN device.

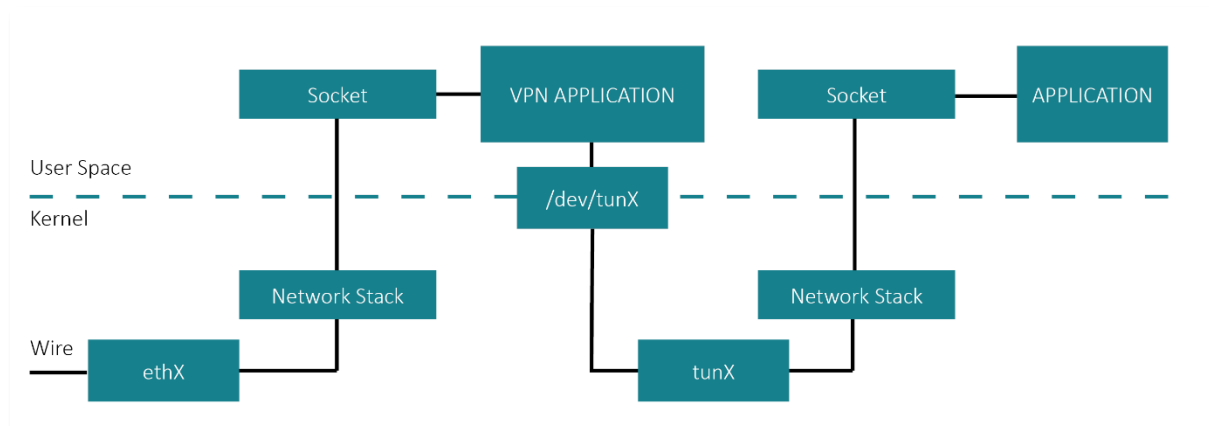


Figure 6: Packet flow through the TUN device

The objective of this task is to get familiar with the TUN/TAP technology. We will conduct several experiments to learn the technical details of the TUN/TAP interface. We will use the following Python program as shown in Listing 1 as the basis for the experiments. We will modify this base code throughout this lab. You can download [tun.py](#) from Blackboard or from the link provided here.

```
#!/usr/bin/env python3

import fcntl
import struct
import os
import time
from scapy.all import *

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

# Create the 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)
```

Listing 1: Creating a TUN interface in Python3 (tun.py)

2.3.1 Name the TUN interface

We will run the *tun.py* program on the *VPN Client*. Make the *tun.py* executable and run it using root privileges using the following commands:

```
// Make the Python program executable
$ chmod +x tun.py

// Run the program using the root privilege
$ sudo tun.py
```

Once the program is executed, it will block the CLI. You can go to another terminal window and get a new shell prompt.

NOTE: If you have not previously completed any other labs that require the Scapy Python package on this VM, you may receive an error saying that the Scapy module is not installed when executing the . If that is the case, you should ensure that Scapy is installed on all three VMs. Install it with the following command:

```
$ sudo pip3 install scapy
```

Print out all the interfaces on the machine.

```
$ ip address
```

You should be able to find an interface called `tun0`. For this task you must modify the code in `tun.py` so that instead of using `tun` as the prefix of the interface name, you should use your last name as the prefix. For example, if your last name is **Smith**, you should use `smith` as the prefix. If you have a long surname, you can use the first five characters of your last name.

Show your results.

2.3.2 Setup the TUN interface

At this point, the TUN interface is not usable, because it has not been configured yet. There are two things that we need to do before the interface can be used.

1. We need to assign an IP address to it.
2. We need to bring up the interface because the interface is created in the down state.

Once we have configured the TUN devices on both the VPN Client and Server, we will have a logical tunnel as shown in Figure 7.

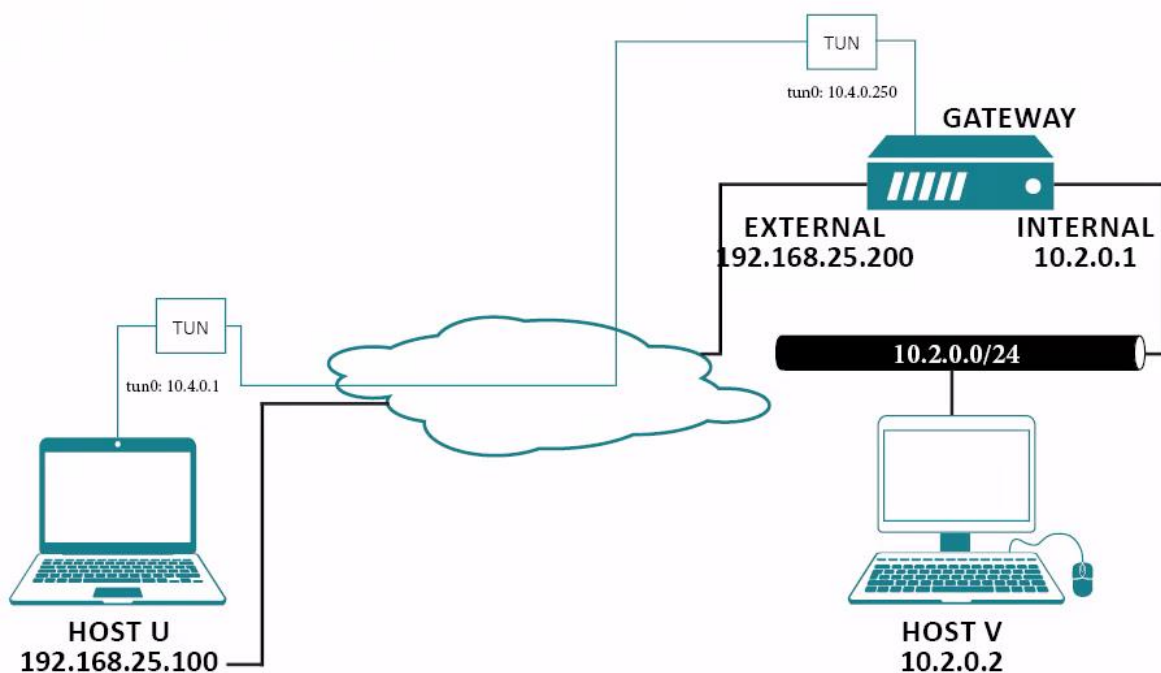


Figure 7: Network diagram including TUN IP addresses

We can use the following two commands to achieve this configuration: (Remember to change the name `tun0` to the name you have used in the previous task)

```
// Assign IP address to the interface
$ sudo ip addr add 10.4.0.1/24 dev tun0

// Bring up the interface
$ sudo ip link set dev tun0 up
```

If you exit from the *tun.py* program and start it again, you will have to manually configure the IP address and interface state again as this configuration would be lost when the interface is deleted. To make it easier, you can modify the *tun.py* program to automatically perform these commands for you. Python can execute system commands using the *os.system()* command. Add the following two lines to *tun.py* after the TUN device has been created and the interface name determined, but before the *while()* loop.

```
os.system("ip addr add 10.4.0.1/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
```

Execute the the “ip address” command again and report your observations on the output for the TUN device. How is it different from that before running the configuration command?

2.3.3 Read from the TUN interface

In this task we will add the functionality to read data from the TUN interface. The data received from the TUN interface is an IP packet. We can use the Python library Scapy to cast the received data into a Scapy IP object so that we can print out each field of the IP packet.

Replace the *while()* loop in *tun.py* with the following code:

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

    if packet:
        pkt = IP(packet)
        print(pkt.summary())
```

Run the revised *tun.py* program on the *VPN Client/Host U* and conduct the following experiments describing your observations.

Ping another host in the 10.4.0.0/24 network (not your own TUN IP address). What do you observe? What is printed out by *tun.py*, why?

Ping another host on your NAT network (EG, from the network diagram the **VPN Sever** is at 192.168.25.200 – Your IP addresses will differ). Does *tun.py* print out anything? Why?

2.3.4 Write to the TUN interface

In this task we will write data to the TUN interface. Since this is a virtual network interface, whatever is written to the interface by the application will appear in the kernel as an IP packet. We will modify the *tun.py* program, so that after getting an ICMP echo-request packet from the TUN interface, we construct a new ICMP echo-reply packet based on the received packet. We then write the new packet to the TUN interface.

Add the following code to the *while()* loop in *tun.py* to spoof an ICMP reply after a packet is received.

```
# Send out a spoof packet using the tun interface
if ICMP in pkt:
    # Reverse the IP addresses in the reply
    newip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)

    # Set the Time-To-Live field
    newip.ttl = 99

    # Create the ICMP layer, setting the ID and sequence numbers.
    newicmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)

    # Copy over any payload, if present and combine the layers.
    if pkt.haslayer(Raw):
        data = pkt[Raw].load
        newpkt = newip/newicmp/data
    else:
        newpkt = newip/newicmp

    # Write the combined packet to the TUN device.
    os.write(tun, bytes(newpkt))
```

REMEMBER: Python is strict on indentation. The code must be indented correctly to be executed in the correct *if()* statement in the file.

Ping the same host in the 10.4.0.0/24 network again. What has changed and why?

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

In this task, we will put the IP packet received from the TUN interface into the UDP payload field of a new IP packet, and send this new packet to the destination sever. Namely, the original packet is encapsulated inside a new packet. This is called IP Tunnelling. In a full VPN, the original IP packet would be encrypted before encapsulation, as shown in Figure 8. In our simple implementation, we will not be encrypting the packets, but they will still be encapsulated and tunnelled.

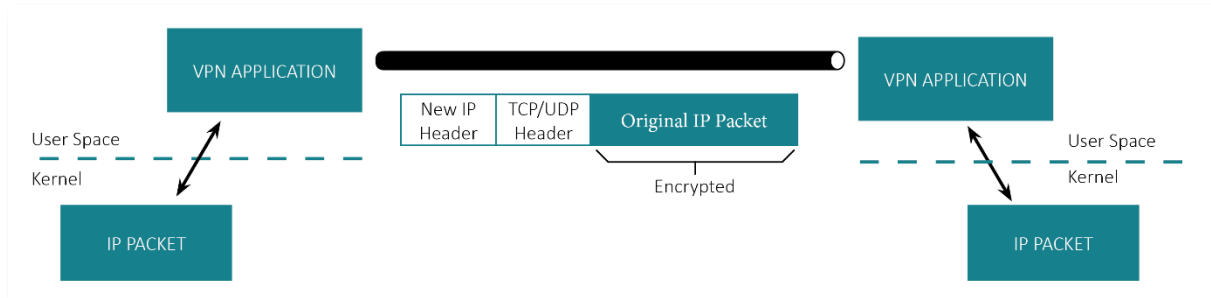


Figure 8: TLS/SSL Tunnelling

Tunnelling is standard client/server programming and can be built on top of either the UDP or TCP protocols. In this task, we will use UDP.

2.4.1 TUN Server program

The `tun_server.py` program will be run on the **VPN Server** VM. This program is a standard UDP server program. It listens on port 9090 and prints out whatever is received. The socket is bound to the IP Address `0.0.0.0`, which is a special IP address that means “listen on all interfaces”.

This server program assumes that the data in the UDP payload field is an IP packet so casts the payload to a Scapy IP object and prints out the source and destination IP addresses of the enclosed IP packet.

```
#!/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))
```

Listing 2: `tun_server.py`

Either download [tun_server.py](#) from Blackboard or the link provided, or enter the code from Listing 2, and run it on the **VPN Server** VM with root privileges.

2.4.2 Implement the client program `tun_client.py`

We will need to modify the `tun.py` program that we have been using on the **VPN Client/Host U machine**. Copy `tun.py` to a new file named `tun_client.py`.

We will modify `tun_client.py` to send data to another computer using standard UDP socket programming. Replace the `while()` loop in `tun_client.py` with the following code. Replace `SERVER_IP` and `SERVER_PORT` with the actual IP address of the **VPN Sever** VM, and the port that the `tun_server.py` program is listening on.

```
# 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:
        pkt = IP(packet)
        print(pkt.summary())

        # Send the packet via the tunnel
        sock.sendto(packet, (SERVER_IP, SERVER_PORT))
```

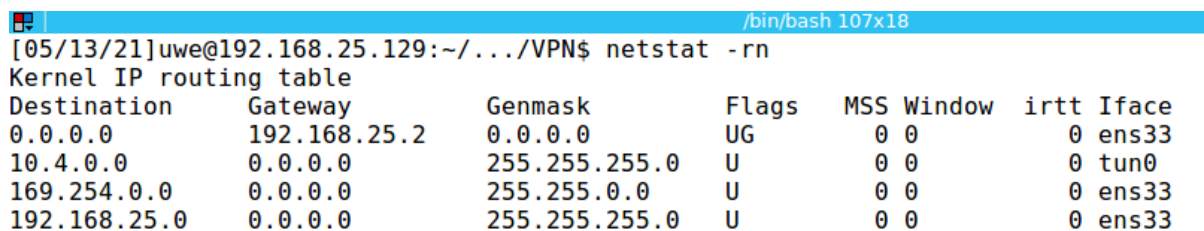
Run the `tun_server.py` program on the **VPN Sever** VM and run `tun_client.py` on the **VPN Client/Host U VM**. To test the tunnel, ping any IP address on the 10.4.0.0/24 network (but not your own TUN IP address) from the **VPN Client**.

What is printed out on the VPN Server? Why?

Our ultimate goal is to reach the hosts inside the private network 10.2.0.0/24 from Host U using the tunnel.

Ping Host V from the **VPN Client/Host U** and verify if the ICMP packet is sent to the VPN server through the tunnel. You should see that the ping to **Host V** does not traverse the tunnel and is not replied to.

The reason for this is due to the IP routing tables. The command “`netstat -rn`” will display the current routing table. Figure 9 shows an example output of a routing table from a VPN Client.



```
[05/13/21]uwe@192.168.25.129:~/.../VPN$ netstat -rn
Kernel IP routing table
Destination    Gateway         Genmask         Flags   MSS Window  irtt  Iface
0.0.0.0        192.168.25.2   0.0.0.0         UG      0  0        0     ens33
10.4.0.0       0.0.0.0        255.255.255.0   U        0  0        0     tun0
169.254.0.0    0.0.0.0        255.255.0.0     U        0  0        0     ens33
192.168.25.0   0.0.0.0        255.255.255.0   U        0  0        0     ens33
```

Figure 9: Example routing table

The figure shows that packets destined for the `10.4.0.0/24` network would be routed via the `tun0` device. Packets for the `169.254.0.0/16` network and `192.168.25.0/24` network would be routed via the `ens33` network interface. The remaining entry for destination `0.0.0.0` is a default route, which means that if a packet does match any of the specific routes, then it will be sent out the default route, in this case, via `ens33`.

As the ICMP packet destined for `10.2.0.0/24` does not match a specific route, it would fall into the default route and be sent over the `ens33` network interface rather than being routed over the tunnel using the `tun0` interface.

The following command shows how to add an entry to the routing table:

```
$ sudo ip route add <network> dev <interface>
```

Modify the routing table to send packets destined for the private network `10.2.0.0/24` which **Host V** resides on through the tunnel. You can add this command to the `tun_client.py` program to automatically configure the route using the Python `os.system()` command.

Note: After making the routing change, you will still not receive a response to the ICMP ping request as we have not yet configured the server to send the response packets back through the tunnel.

Demonstrate that when you pin an IP address in the `10.2.0.0/24` network, the ICMP packets are received by the `tun_server.py` program on the **VPN Server** VM.

2.5 Task 4: Setup the VPN Server

When the *tun_server.py* gets a packet from the tunnel, it needs to pass the packet to the kernel so that the kernel can route the packet towards its final destination. This needs to be done through a TUN interface, like we did for the client in Task 2.

Modify the *tun_server.py* file so it can do the following:

- Create a TUN interface and configure it with the IP address 10.4.0.250.
- Read data from the socket interface; treat the received data as an IP packet.
- Write the packet to the TUN interface.

Before running the modified *tun_server.py* program, we need to enable IP forwarding on the **VPN Server**. Unless specifically configured to do so, a computer will only act as a host and not as a gateway. This means a computer will only act on packets where it is the destination. In our situation, the received packet has an ultimate destination that is another computer. Therefore the **VPN Server** needs to forward the packets between the tunnel and the private network, so it needs to function as a gateway.

We can enable the IP forwarding for a computer to behave like a gateway using the following command:

```
$ sudo sysctl net.ipv4.ip_forward=1
```

Again, you may find this easier to add this to your *tun_server.py* program using the *os.system()* command so that you do not forget to enable this.

Demonstrate that when you ping from the **VPN Client/Host U** to **Host V** that the ICMP echo request packets arrive at **Host V** through the tunnel.

Note: Although **Host V** receives the ICMP echo request packets and responds, the reply will not get back to the **VPN Client/Host U** yet as we have not completely set everything up yet. For the purposes of this task, it is sufficient to show using *Wireshark* or *tcpdump* that the ICMP packets have arrived at **Host V**.

2.6 Task 5: Handling Traffic in Both Directions

At this point, the functionality for supporting traffic in one direction is complete. i.e. we can send packets from **Host U** to **Host V**. We can also see that **Host V** has replied, but the response is dropped somewhere. This is because our tunnel is only one directional; we need to support traffic in the opposite direction so that packets can be tunnelled back to the **VPN Client/Host U**.

To achieve this, our TUN client and server programs must read data from two interfaces; the TUN interface and socket interface, then pass the received data to the opposite interface.

Internally to the Operating System, interfaces are represented by *File Descriptors (FD)*. We can monitor the file descriptors to see if there is data present from them. There are multiple ways to do this.

Polling – We can poll the file descriptors to see if there is any data available to be read. The performance of this approach is undesirable as the process will need to keep running in an idle loop when there is no data present. This can have a negative effect on the rest of the system.

Read – When we read from an interface, if there is no data present then the process is suspended (blocked) until data becomes available. When data is available, the process is unblocked, and its execution will resume. The benefit of this approach is that it does not waste CPU time when no data is available. However, this mechanism does not work well when there is more than one interface. If we have blocked reading one interface and data becomes available on a second interface, the program will remain blocked until data arrives on the first interface.

Select() – Linux has a system call called *select()* which allows a program to block and monitor multiple file descriptors simultaneously. To use *select()*, we store all the file descriptors we wish to monitor in a set, and pass the set to *select()*. When data becomes available on any of the file descriptors in the set, the program will resume, and we can check which file descriptor has received data and process it accordingly.

Use the code below to replace the *while()* loop in both the *tun_client.py* and *tun_server.py*. The code is incomplete. You will need to complete the code in order to write the data out to the correct interface.

```
# We assume that sock and tun file descriptors have already been created.

# Default IP and Port. These will be set correctly to the Client IP
# data on reception of the first packet from the socket
ip = '9.9.9.9'
port = 9999

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, (ip, port) = sock.recvfrom(2048)
            pkt = IP(data)
            print("From socket <==: {} --> {}".format(pkt.src, pkt.dst))
            ... (code needs to be added by students) ...

        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) ...
```

Demonstrate that you can now communicate with **Host V** from **Host U**, and that the unencrypted VPN tunnel is now complete. Show *Wireshark* proof for a *ping* command. Discuss how the packets flow.

2.7 Task 6: Tunnel-breaking experiment

From **Host U**, *telnet* to **Host V** over the VPN Tunnel. Now we will break the tunnel by stopping either the *tun_client.py* or *tun_server.py* program. Continue to type in the *telnet* window.

Now reconnect the VPN tunnel by restarting the program that you stopped (Do not take too long to do this). Once the tunnel is re-established, what happens to the *telnet* window?

Describe and explain your observations. Ensure you show the successful *telnet* traffic in *Wireshark*.

2.8 Encrypting the Tunnel

At this point we have created an IP tunnel, but the tunnel is not protected. Only after the tunnel is secure can we really call it a VPN. We will not cover encrypting the tunnel in this lab as it is a significant amount of additional work to achieve this.

To secure the tunnel we need to achieve two goals: *Confidentiality* and *Integrity*. Confidentiality is achieved by encrypting the packets that pass through the tunnel. Integrity ensures that nobody can tamper with the traffic in the tunnel or launch a replay attack. Integrity can be achieved by using Message Authentication Codes (MAC). Both goals can be achieved by implementing Transport Layer Security (TLS).

TLS is typically built on top of TCP, although a version of TLS called DTLS (Datagram Transport Layer Security) exists that can run on top of UDP.

We will not cover encrypting the tunnel in this lab as it is a significant amount of further work to replace the UDP connection with a TCP connection, authenticate the VPN Server using public-key certificates, authenticating the VPN client using either a password/username or client certificates, and finally supporting multiple client connections to the server. If you wished to explore encrypting your tunnel further, then you can look at the [OpenSSL](#) libraries and incorporate setting up a TLS connection in the client and server programs. This is not required for this lab.

2.9 Task 7: Choosing between an SSL/TLS VPN vs IPsec VPN (Research Task)

VPNs come in many types and protocols. An SSL/TLS VPN is a VPN that applies TLS encryption to the tunnel that we created during this lab. Once the tunnel is established, a TLS connection is made through the tunnel and all further communication over the tunnel is then encrypted using the TLS connection.

A common alternative VPN technology is IPsec. IPsec is a group of protocols that are used together to encrypt connections between devices. The two VPNs work in fundamentally different ways to achieve the same result of encrypting communication between two points.

2.9.1 Task Requirements

Write a short report (1000 words max) discussing the two alternative approaches to VPNs (SSL/TLS and IPsec) and how they are implemented. Your report should include relevant references from the literature and your discussion should cover the advantages and disadvantages of the alternate approaches.

3 Submission

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. The format of the lab report is up to you. You can copy the questions from this worksheet into a new document and answer them in the separate report. The report should be of a professional standard.

You need to provide explanation to the observations that are interesting or surprising. Please also list any important code snippets you have written followed by explanation. Simply attaching code or screenshots without any explanation will not receive credits. The report must demonstrate your understanding of the subject and material and not just be a log of your actions.

All screenshots in the report must have your student number and date stamp in the user prompt. Failure to include these details in the screenshots will invalidate the report and receive a mark of zero.

4 Plagiarism

This is an assessed lab sheet and while it is acceptable to discuss your assignment with your peers as per the university rules, this assignment is intended as an individual assignment. Submissions that are substantially similar will be subject to investigation according to university regulations and any proven cases will be dealt with according to the regulations. More details can be found here <http://www1.uwe.ac.uk/students/academicadvice/assessments/assessmentoffences.aspx>

5 Marking Criteria

| | 0-29% | 30-39% | 40-49% | 50-59% | 60-69% | 70-84% | 85-100% |
|----------------------------------|---|--|--|--|--|---|---|
| Tasks 1-6 (65%) | Little or no effort made to complete the tasks detailed / Serious gaps or errors in understanding the topic | Some tasks complete with major omission / Some evidence of understanding the topic with major errors or gaps | Most tasks complete but with minor omissions / Evidence of understanding the topic but with minor errors or gaps | All tasks complete in full. Evidence incomplete or unclear in places / Adequate understanding of topic | All tasks complete in full. Evidence of a good standard to detail tasks. / Clear understanding of topic | All tasks complete in full. Excellent use of evidence to detail tasks. / Thorough and comprehensive understanding of topic | All tasks complete in full. Highly reflective use of evidence to develop argument / Impressive and original depth of understanding of topic |
| Task 7 (25%) | Little to no discussion | Poor analysis. Demonstrates little or no insight into the problem | Below-average analysis. Analysis is lacking in most aspects | Adequate analysis of relevant works, but lacks depth; demonstrates some insight into the problem | Good analysis of relevant works but could be more critical; demonstrates good insight into the problem. Good use of source | Very good analysis of relevant works; demonstrates excellent insight into the problem. Good use of sources and all sources are appropriately referenced | Outstanding analysis of relevant works; demonstrates outstanding insight into the problem and fully covers all aspects. Excellent use of sources and all sources are appropriately referenced |
| Report Presentation (10%) | Very poor presentation | Weak presentation | Has not followed required conventions; poor proof-reading | Usually follows required practices; some issues to be addressed e.g., typos, punctuation | Follows required presentational practices; a few typos/errors in punctuation or grammar | Excellent presentation: typos/errors in punctuation etc. are rare | Excellent presentation |

6 Document Revision History

| Version | Date | Changes |
|---------|---------------------------|---|
| 1.0 | 13 th May 2021 | Initial Release |
| 1.1 | 27 th May 2021 | Updated Network setup instructions to highlight possible error when restarting the networking services. Added additional instruction on how to install Scapy if it is not already installed on the VM. |