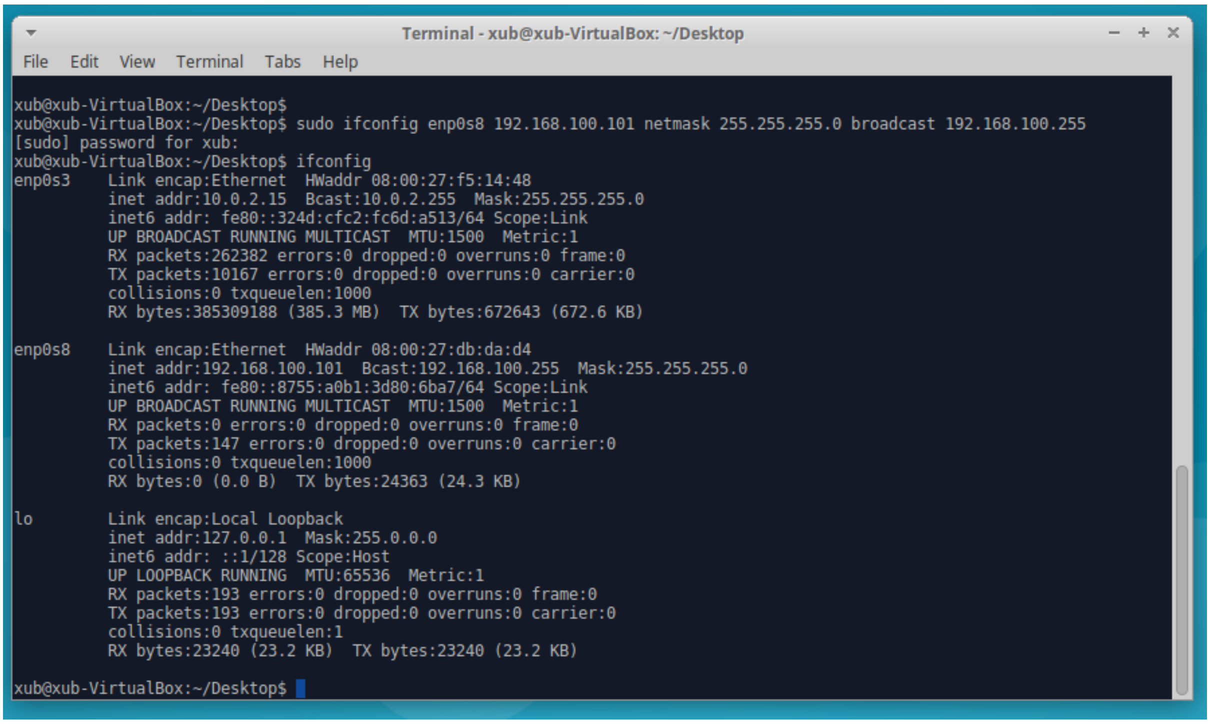
CSC3064 Practical 2
Tunneling

This practical covers a series of exercises for you to familiarize yourself with tunnelling and VPNs. Following the exercise, there are some short questions for you to answer. Make a note of your answers and we will discuss them in class next week.

Test Network Configuration

The exercises in this practical use 2 Linux virtual machines (VMs) as a simple test network.

1. From the Windows Desktop, launch Virtualbox.
2. Start the 2 xubuntu VMs. Username: xub, Password: CSC3064_2018
3. Check "ifconfig" on each machine to see the networking information.
4. To enable the two VMs to communicate on an internal network, allocate private IP addresses to each VM.
   a. On VM1, allocate an IP e.g. 192.168.101.11 and check the new IP address settings using "ifconfig"
   b. On VM2, allocate a different IP e.g. 192.168.101.12 and check the new IP address settings using "ifconfig"


Terminal - xub@xub-VirtualBox: ~/Desktop
File Edit View Terminal Tabs Help

```
xub@xub-VirtualBox:~/Desktop$
xub@xub-VirtualBox:~/Desktop$ sudo ifconfig enp0s8 192.168.100.101 netmask 255.255.255.0 broadcast 192.168.100.255
[sudo] password for xub:
xub@xub-VirtualBox:~/Desktop$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:f5:14:48
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::324d:cfc2:fc6d:a513/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:262382 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10167 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:385309188 (385.3 MB)  TX bytes:672643 (672.6 KB)

enp0s8    Link encap:Ethernet  HWaddr 08:00:27:db:da:d4
          inet addr:192.168.100.101  Bcast:192.168.100.255  Mask:255.255.255.0
          inet6 addr: fe80::8755:a0b1:3d80:6ba7/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:147 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:24363 (24.3 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:193 errors:0 dropped:0 overruns:0 frame:0
          TX packets:193 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:23240 (23.2 KB)  TX bytes:23240 (23.2 KB)

xub@xub-VirtualBox:~/Desktop$
```

Figure 1: Allocate a private IP address to each VM

5. Check that you can communicate between the two machines using *ping*

In this lab, we will set up a host-to-host tunnel between our two virtual machines, as shown in Figure 2.

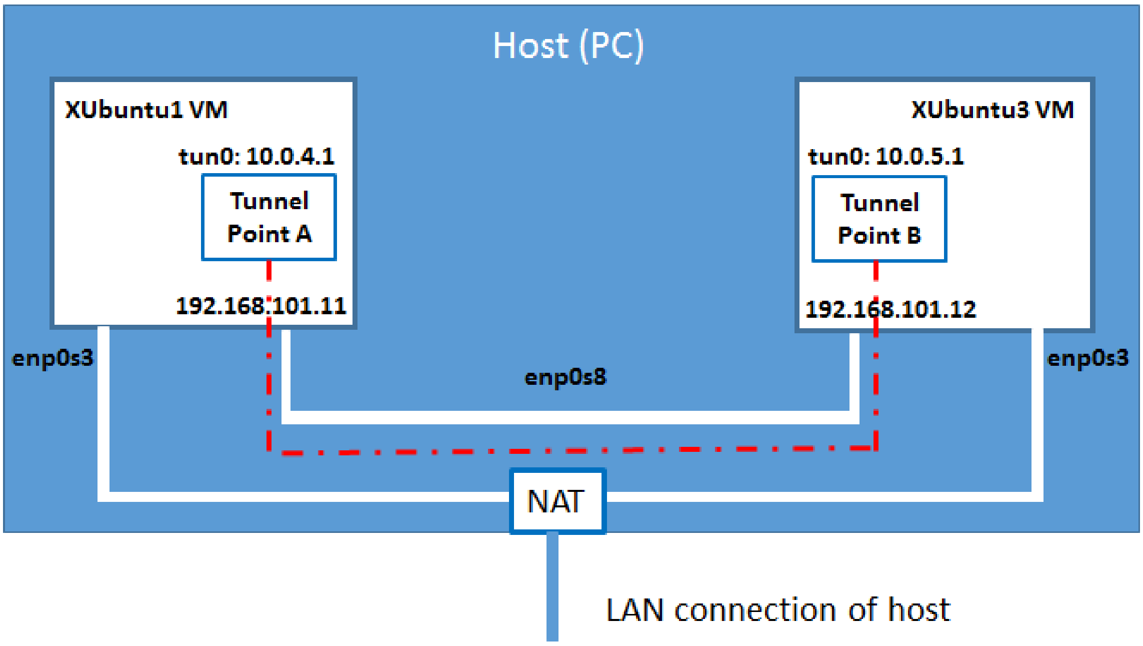


Figure 2: Host-to-Host Tunnel

Exercise: Create a Host-to-Host Tunnel using TUN/TAP

The enabling technology for the TLS/SSL VPNs is TUN/TAP, which is now widely implemented in modern operating systems. TUN and TAP are virtual network kernel drivers; they implement network devices 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.

A user-space program is usually attached to the TUN/TAP virtual network interface. Packets sent by an operating system via a TUN/TAP network interface are delivered to the user-space program. On the other hand, packets sent by the program via a TUN/TAP network interface are injected into the operating system network stack; to the operating system, it appears that the packets come from an external source through the virtual network interface.

When a program is attached to a TUN/TAP interface, the IP packets that the computer sends to this interface will be piped into the program; on the other hand, the IP packets that the program sends to the interface will be piped into the computer, as if they came from the outside through this virtual network interface. The program can use the standard read() and write() system calls to receive packets from or send packets to the virtual interface.

In this lab we use the program **simpletun**, which connects two computers using the TUN tunneling technique. The program has been written by Davide Brini and is described at the following link: http://backreference.org/2010/03/26/tuntap-interface-tutorial

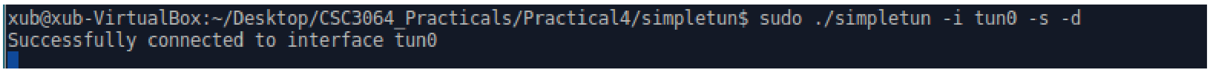
The simpletun program can run as both a client and a server. When it is running with the -s flag, it acts as a server; when it is running with the -c flag, it acts as a client.

Step 1: On VM1, navigate to the simpletun folder ~/Desktop/CSC3064_Practicals/Practical4/simpletun

Step 2: We use Tunnel Point A as the server side of the tunnel. Point A is on machine 192.168.101.11 (see Figure 2). Note: the client/server concept is only meaningful when *establishing* the connection between the two ends. Once the tunnel is established, there is no difference between client and server.

On VM1, set the tunnel endpoint, '-i' identifies the interface, '-s' indicates in server mode, and '-d' sets the program to print out debug information.

sudo ./simpletun –i tun0 –s -d



```
xub@xub-VirtualBox:~/Desktop/CSC3064_Practicals/Practical4/simpletun$ sudo ./simpletun -i tun0 -s -d
Successfully connected to interface tun0
```

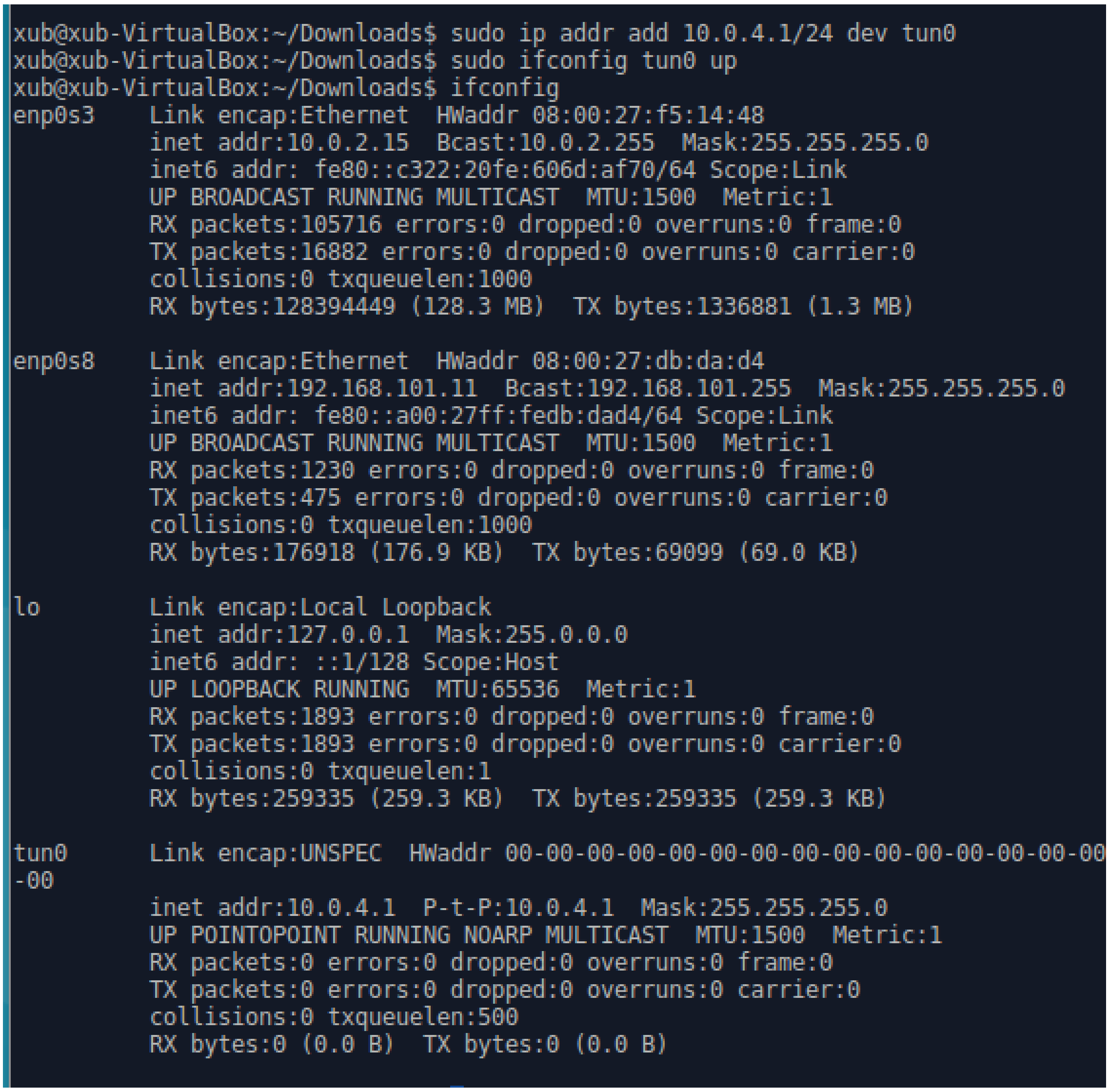
Figure 3: simpletun on VM1 waiting for connections

The above command has set up an additional virtual network interface called tun0. However, this interface is not yet configured. We'll configure it by assigning an IP address. Note: As you can see from Figure 3, simpletun on VM1 is waiting for connections. Open a second terminal window on VM1 to configure the tun0 interface.

sudo ip addr add 10.0.4.1/24 dev tun0

sudo ifconfig tun0 up

ifconfig



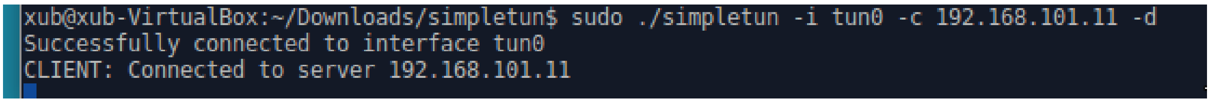
```
xub@xub-VirtualBox:~/Downloads$ sudo ip addr add 10.0.4.1/24 dev tun0
xub@xub-VirtualBox:~/Downloads$ sudo ifconfig tun0 up
xub@xub-VirtualBox:~/Downloads$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:f5:14:48
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::c322:20fe:606d:af70/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:105716 errors:0 dropped:0 overruns:0 frame:0
          TX packets:16882 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:128394449 (128.3 MB)  TX bytes:1336881 (1.3 MB)

enp0s8    Link encap:Ethernet  HWaddr 08:00:27:db:da:d4
          inet addr:192.168.101.11  Bcast:192.168.101.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fedb:dad4/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1230 errors:0 dropped:0 overruns:0 frame:0
          TX packets:475 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:176918 (176.9 KB)  TX bytes:69099 (69.0 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:1893 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1893 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:259335 (259.3 KB)  TX bytes:259335 (259.3 KB)

tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
-00
          inet addr:10.0.4.1  P-t-P:10.0.4.1  Mask:255.255.255.0
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Figure 4: interface configuration on VM1

Step 3: We use Tunnel Point B as the client side of the tunnel. Point B is on machine 192.168.101.12 (see Figure 2). First, we connect to the server program running on 192.168.101.11, which is the machine that runs the Tunnel Point A.

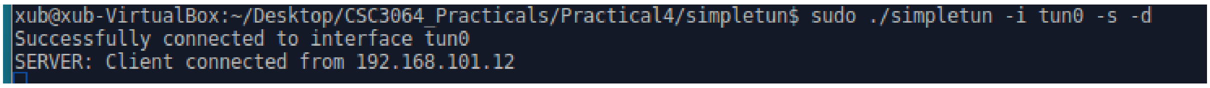
On VM2, navigate to the simpletun folder
~/Desktop/CSC3064_Practicals/Practical4/simpletun

sudo ./simpletun –i tun0 –c 192.168.101.11 -d



```
xub@xub-VirtualBox:~/Downloads/simpletun$ sudo ./simpletun -i tun0 -c 192.168.101.11 -d
Successfully connected to interface tun0
CLIENT: Connected to server 192.168.101.11
```

Figure 5: simpletun on VM2 connected to VM1 via tunnel



```
xub@xub-VirtualBox:~/Desktop/CSC3064_Practicals/Practical4/simpletun$ sudo ./simpletun -i tun0 -s -d
Successfully connected to interface tun0
SERVER: Client connected from 192.168.101.12
```

Figure 6: simpletun on VM1 showing tunnel connection to VM2

Note, as shown in Figure 5, this command also blocks, so open a second terminal window on VM2 to configure the tun0 interface.

sudo ip addr add 10.0.5.1/24 dev tun0

sudo ifconfig tun0 up

ifconfig

Step 4: The tunnel is now established, as shown in Figures 5 and 6. Before we can use the tunnel, we need to set up the routing path on both machines to direct the intended traffic through the tunnel.

On VM1, direct the packets to the 10.0.5.0/24 network through the interface tun0.

sudo route add –net 10.0.5.0 netmask 255.255.255.0 dev tun0

On VM2, direct the packets to the 10.0.4.0/24 network through the interface tun0.

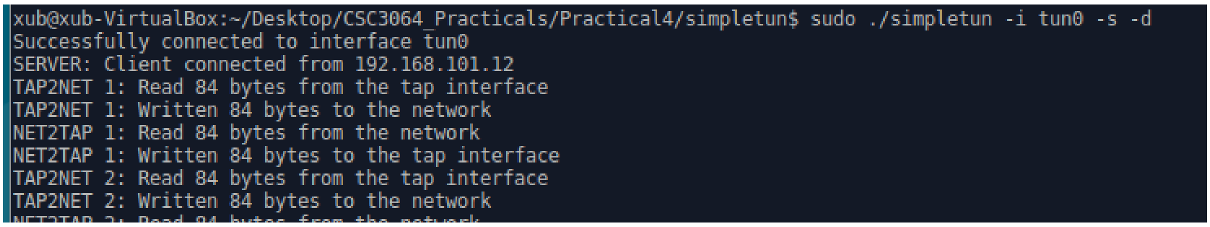
sudo route add –net 10.0.4.0 netmask 255.255.255.0 dev tun0

Step 5: We can now access 10.0.5.1 from 192.168.101.12 (and similarly access 10.0.4.1 from 192.168.101.12). Test the tunnel using ping.

On VM1, ping 10.0.5.1

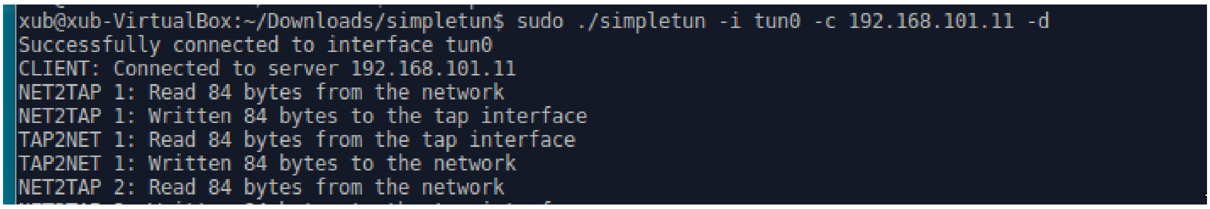
On VM2, ping 10.0.4.1

You should see something similar to that shown in Figures 7 and 8.



```
xub@xub-VirtualBox:~/Desktop/CSC3064_Practicals/Practical4/simpletun$ sudo ./simpletun -i tun0 -s -d
Successfully connected to interface tun0
SERVER: Client connected from 192.168.101.12
TAP2NET 1: Read 84 bytes from the tap interface
TAP2NET 1: Written 84 bytes to the network
NET2TAP 1: Read 84 bytes from the network
NET2TAP 1: Written 84 bytes to the tap interface
TAP2NET 2: Read 84 bytes from the tap interface
TAP2NET 2: Written 84 bytes to the network
NET2TAP 2: Read 84 bytes from the network
```

Figure 7: Testing simpletun using ping – VM1



```
xub@xub-VirtualBox:~/Downloads/simpletun$ sudo ./simpletun -i tun0 -c 192.168.101.11 -d
Successfully connected to interface tun0
CLIENT: Connected to server 192.168.101.11
NET2TAP 1: Read 84 bytes from the network
NET2TAP 1: Written 84 bytes to the tap interface
TAP2NET 1: Read 84 bytes from the tap interface
TAP2NET 1: Written 84 bytes to the network
NET2TAP 2: Read 84 bytes from the network
```

Figure 8: Testing simpletun using ping – VM2

Step 6: Run wireshark on VM1. Observe the difference between ping packets between the VMs using the tunnel and ping packets with no tunnel.

Now answer questions 1 to 3.

Lab4 Q1: Describe the difference(s) in the wireshark packet information between a ping via the tunnel and with no tunnel.

Lab4 Q2: The connection used in the simpletun program is a TCP connection. Why would it be better to use UDP in the tunnel, instead of TCP?

Lab4 Q3: What is the average RTT for the ping via the tunnel? Note: RTT is round trip time. Comment on the difference in RTT with and without the tunnel.