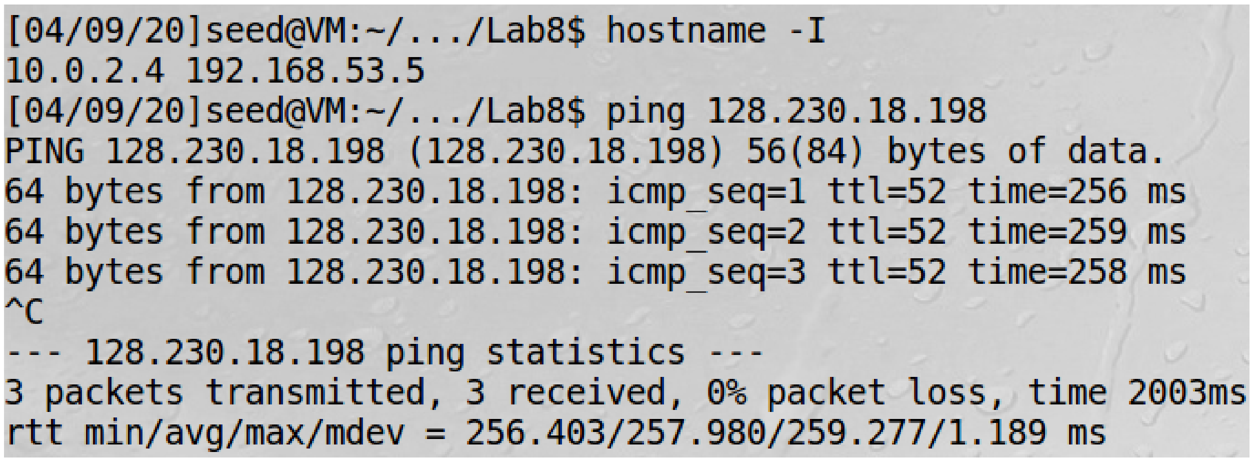
Task 2: Setup Firewall

Machine		**IP Address**
A	VPN Client (inside firewall)	`10.0.2.4`
B	VPN Server (outside firewall)	`10.0.2.5`

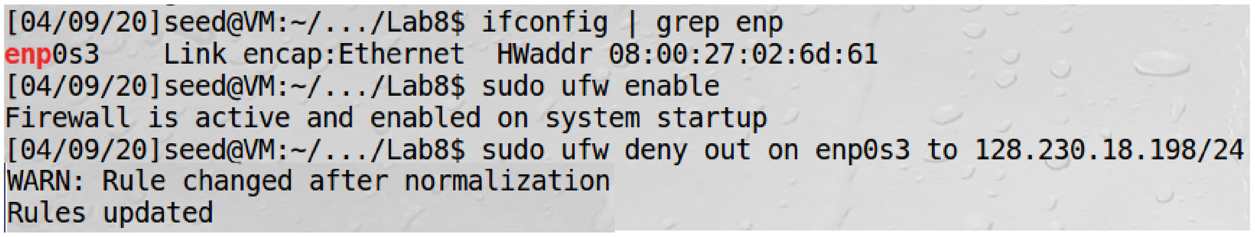
A website that we would like to block is `www.syr.edu` (`128.230.18.198`). As shown in Figure 1 below, we can see that before the firewall is setup, the target IP address is reachable.



```
[04/09/20]seed@VM:~/.../Lab8$ hostname -I
10.0.2.4 192.168.53.5
[04/09/20]seed@VM:~/.../Lab8$ ping 128.230.18.198
PING 128.230.18.198 (128.230.18.198) 56(84) bytes of data.
64 bytes from 128.230.18.198: icmp_seq=1 ttl=52 time=256 ms
64 bytes from 128.230.18.198: icmp_seq=2 ttl=52 time=259 ms
64 bytes from 128.230.18.198: icmp_seq=3 ttl=52 time=258 ms
^C
--- 128.230.18.198 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 256.403/257.980/259.277/1.189 ms
```

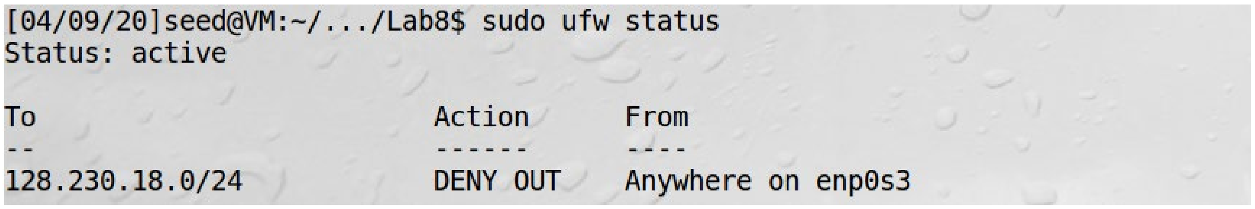
Figure 1: Successful ping to 128.230.18.198 on Client VM (10.0.2.4)

Using the `ufw` program, we set up the firewall rule on the first VM (`10.0.2.4`) by running `sudo ufw enable` and then `sudo ufw deny out on ethxx to 128.230.18.198/xx` as shown in Figure 2. We can see that the firewall rule has been successfully set up as shown in Figure 3.



```
[04/09/20]seed@VM:~/.../Lab8$ ifconfig | grep enp
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:02:6d:61
[04/09/20]seed@VM:~/.../Lab8$ sudo ufw enable
Firewall is active and enabled on system startup
[04/09/20]seed@VM:~/.../Lab8$ sudo ufw deny out on enp0s3 to 128.230.18.198/24
WARN: Rule changed after normalization
Rules updated
```

Figure 2: Set up Firewall Rule

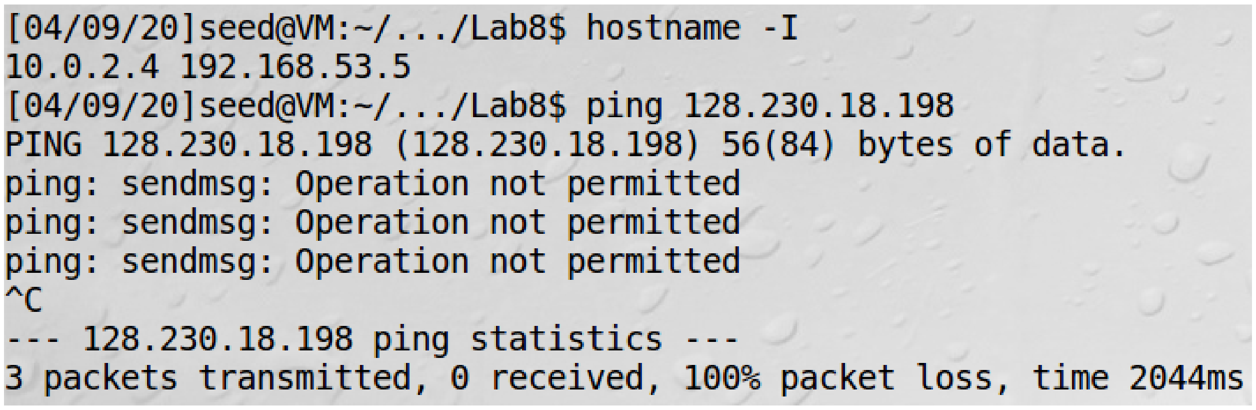


```
[04/09/20]seed@VM:~/.../Lab8$ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
128.230.18.0/24            DENY OUT    Anywhere on enp0s3
```

Figure 3: Successful Firewall Rule Set Up

To test the firewall rule, we try to reach the target IP address again by running `ping 128.230.18.198`, but as shown in Figure 4, the target IP address is no longer reachable. This shows that the firewall rule has been successfully setup.



```
[04/09/20]seed@VM:~/.../Lab8$ hostname -I
10.0.2.4 192.168.53.5
[04/09/20]seed@VM:~/.../Lab8$ ping 128.230.18.198
PING 128.230.18.198 (128.230.18.198) 56(84) bytes of data.
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
^C
--- 128.230.18.198 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2044ms
```

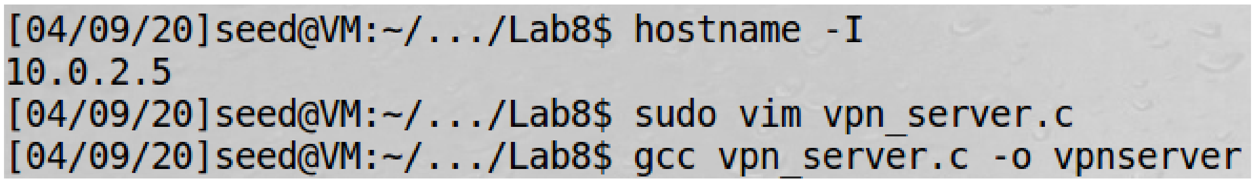
Figure 4: Unuccessful ping to 128.230.18.198 on Client VM (10.0.2.4)

Task 3: Bypassing Firewall using VPN

Machine		**IP Address**
A	VPN Client (inside firewall)	`10.0.2.4`
B	VPN Server (outside firewall)	`10.0.2.5`

Step 1: Run VPN Server (`10.0.2.5`)

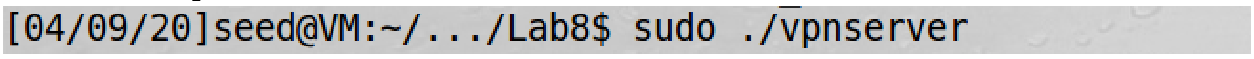
To compile the server program with C, we run `gcc vpn_server.c -o vpnserver` as shown in Figure 5 on the VPN Server (`10.0.2.5`).



```
[04/09/20]seed@VM:~/.../Lab8$ hostname -I
10.0.2.5
[04/09/20]seed@VM:~/.../Lab8$ sudo vim vpn_server.c
[04/09/20]seed@VM:~/.../Lab8$ gcc vpn_server.c -o vpnserver
```

Figure 5: Compiling Server Program with C

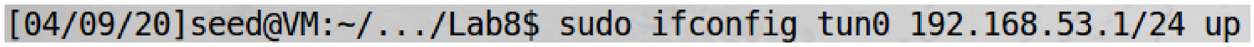
To run the compiled server program, we run `sudo ./vpnserver` on the VPN Server (`10.0.2.5`) as shown in Figure 6.



```
[04/09/20]seed@VM:~/.../Lab8$ sudo ./vpnserver
```

Figure 6: Running Compiled Server Program

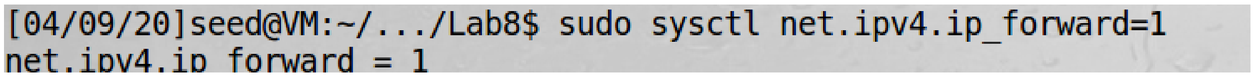
To configure the new interface, we configure it by giving it an IP address such as `192.168.53.1` for this interface. In a second terminal window, we run the command, `sudo ifconfig tun0 192.168.53.1/24 up` as shown in Figure 7 below.



```
[04/09/20]seed@VM:~/.../Lab8$ sudo ifconfig tun0 192.168.53.1/24 up
```

Figure 7: Configuring interface on Server VM (10.0.2.5)

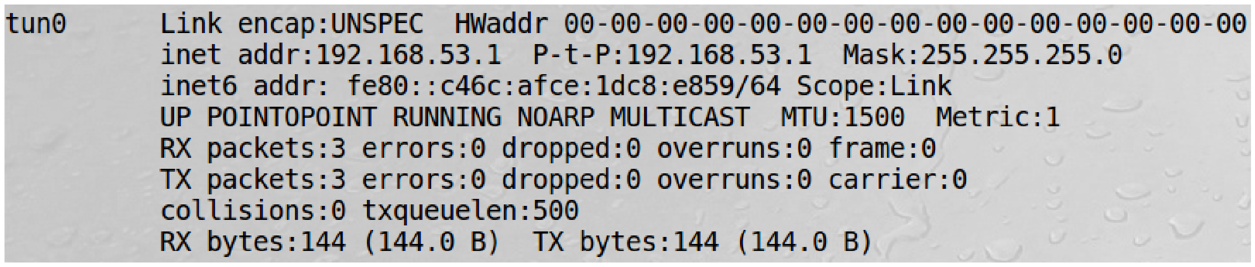
To configure the VPN Server to function as a gateway so that it can forward packets to other destinations, we need to enable IP forwarding. On the VPN Server (`10.0.2.5`), we enable IP forwarding by running the command, `sudo sysctl net.ipv4.ip_forward=1`, as shown in Figure 8.



```
[04/09/20]seed@VM:~/.../Lab8$ sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
```

Figure 8: Enabling IP Forwarding on Server VM (10.0.2.5)

From Figure 9, we can run `ifconfig -a` and see that the TUN interface is successfully setup on the VPN Server.

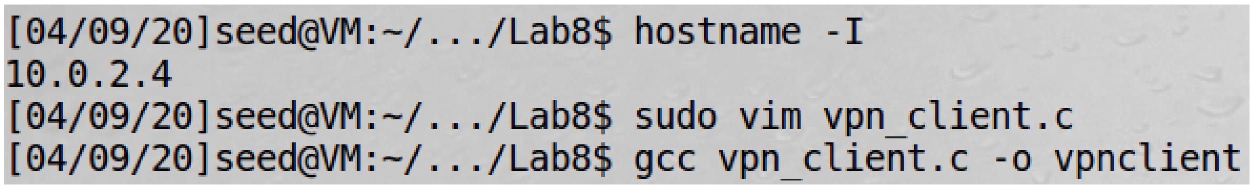


```
tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:192.168.53.1  P-t-P:192.168.53.1  Mask:255.255.255.0
          inet6 addr: fe80::c46c:afce:1dc8:e859/64 Scope:Link
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:3 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:144 (144.0 B)  TX bytes:144 (144.0 B)
```

Figure 9: TUN Interface on Server VM (10.0.2.5)

Step 2: Run VPN Client (`10.0.2.4`)

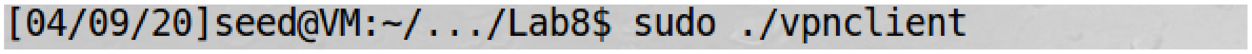
To compile the client program with C, we run `gcc vpn_client.c -o vpnclient` as shown in Figure 10 on the VPN Client (`10.0.2.4`).



```
[04/09/20]seed@VM:~/.../Lab8$ hostname -I
10.0.2.4
[04/09/20]seed@VM:~/.../Lab8$ sudo vim vpn_client.c
[04/09/20]seed@VM:~/.../Lab8$ gcc vpn_client.c -o vpnclient
```

Figure 10: Compiling Client Program with C

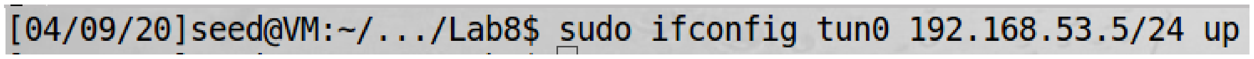
To run the compiled server program, we run `sudo ./vpnclient` on the VPN Client (`10.0.2.4`) as shown in Figure 11.



```
[04/09/20]seed@VM:~/.../Lab8$ sudo ./vpnclient
```

Figure 11: Running Compiled Client Program

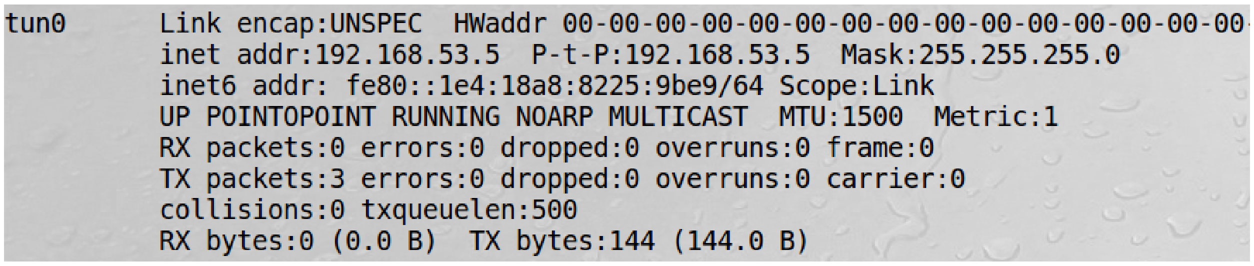
To configure the new interface, we configure it by giving it an IP address such as `192.168.53.5` for this interface. In a second terminal window, we run the command, `sudo ifconfig tun0 192.168.53.5/24 up` as shown in Figure 12 below.



```
[04/09/20]seed@VM:~/.../Lab8$ sudo ifconfig tun0 192.168.53.5/24 up
```

Figure 12: Configuring interface on Client VM (10.0.2.4)

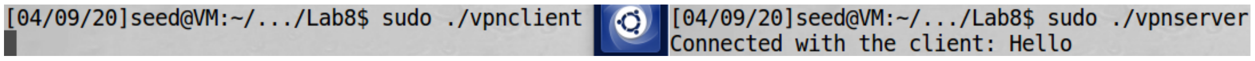
From Figure 13, we can run `ifconfig -a` and see that the TUN interface is successfully setup on the VPN Client.



```
tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:192.168.53.5  P-t-P:192.168.53.5  Mask:255.255.255.0
          inet6 addr: fe80::1e4:18a8:8225:9be9/64 Scope:Link
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:144 (144.0 B)
```

Figure 13: TUN Interface on Client VM (10.0.2.4)

From Figure 14, we can see that the server has successfully connected to the client.



```
[04/09/20]seed@VM:~/.../Lab8$ sudo ./vpnclient
```

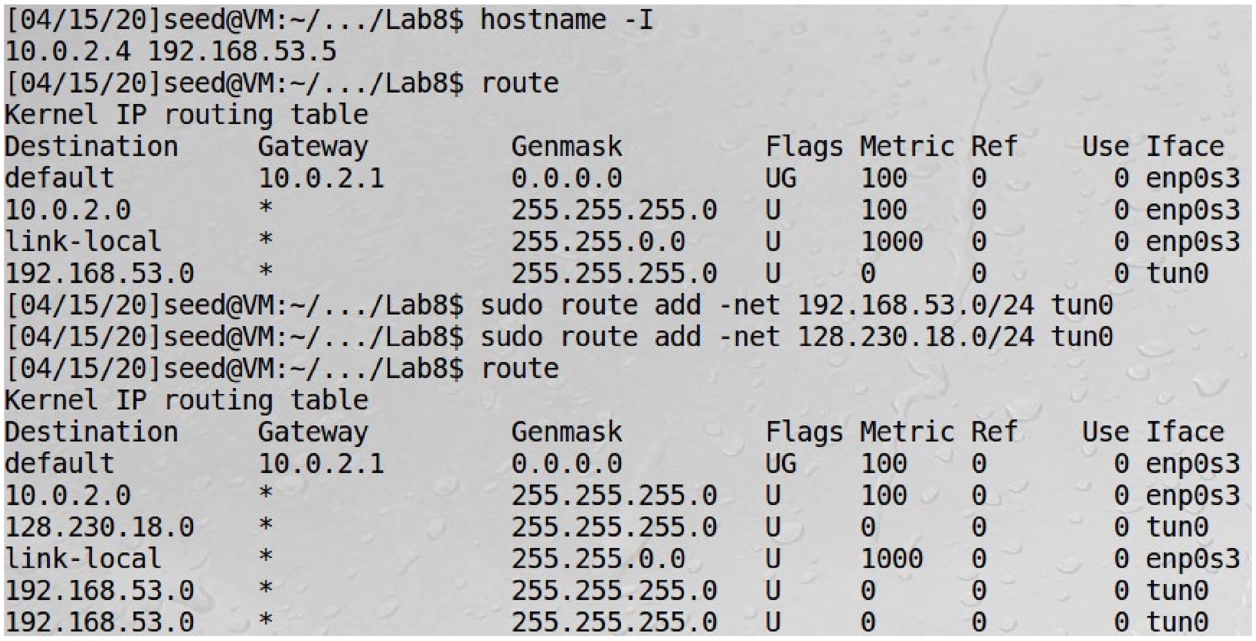
```
[04/09/20]seed@VM:~/.../Lab8$ sudo ./vpnserver
Connected with the client: Hello
```

Figure 14: Successful Server-Client Connection

Step 3: Set Up Routing on Client (`10.0.2.4`) and Server (`10.0.2.5`) VMs

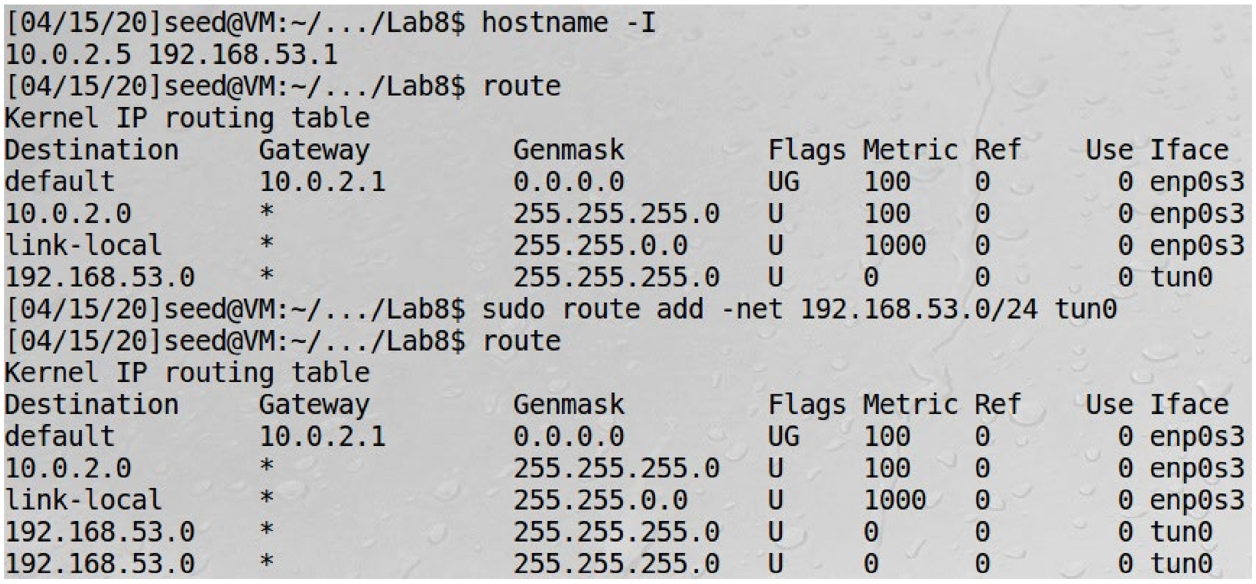
Before we can use the tunnel, we need to set up routing paths on both client and server machines to direct the intended traffic through the tunnel.

On both the client (`10.0.2.4`) and the server (`10.0.2.5`) VMs, the `route` command is used to add a routing entry to route all `192.168.53.0/24`-bound packets to the interface `tun0` by running `sudo route add -net 192.168.53.0/24 tun0` as shown in Figures 15 and 16 respectively. We also route all the `128.230.18.0/24`-bound packets to the interface `tun0` by running `sudo route add -net 128.230.18.0/24 tun0` as shown in Figure 15.



```
[04/15/20]seed@VM:~/.../Lab8$ hostname -I
10.0.2.4 192.168.53.5
[04/15/20]seed@VM:~/.../Lab8$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         10.0.2.1        0.0.0.0         UG    100    0        0 enp0s3
10.0.2.0        *               255.255.255.0   U     100    0        0 enp0s3
link-local      *               255.255.0.0     U     1000   0        0 enp0s3
192.168.53.0    *               255.255.255.0   U     0      0        0 tun0
[04/15/20]seed@VM:~/.../Lab8$ sudo route add -net 192.168.53.0/24 tun0
[04/15/20]seed@VM:~/.../Lab8$ sudo route add -net 128.230.18.0/24 tun0
[04/15/20]seed@VM:~/.../Lab8$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         10.0.2.1        0.0.0.0         UG    100    0        0 enp0s3
10.0.2.0        *               255.255.255.0   U     100    0        0 enp0s3
128.230.18.0    *               255.255.255.0   U     0      0        0 tun0
link-local      *               255.255.0.0     U     1000   0        0 enp0s3
192.168.53.0    *               255.255.255.0   U     0      0        0 tun0
192.168.53.0    *               255.255.255.0   U     0      0        0 tun0
```

Figure 15: Routing on Client VM (10.0.2.4)



```
[04/15/20]seed@VM:~/.../Lab8$ hostname -I
10.0.2.5 192.168.53.1
[04/15/20]seed@VM:~/.../Lab8$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         10.0.2.1        0.0.0.0         UG    100    0        0 enp0s3
10.0.2.0        *               255.255.255.0   U     100    0        0 enp0s3
link-local      *               255.255.0.0     U     1000   0        0 enp0s3
192.168.53.0    *               255.255.255.0   U     0      0        0 tun0
[04/15/20]seed@VM:~/.../Lab8$ sudo route add -net 192.168.53.0/24 tun0
[04/15/20]seed@VM:~/.../Lab8$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         10.0.2.1        0.0.0.0         UG    100    0        0 enp0s3
10.0.2.0        *               255.255.255.0   U     100    0        0 enp0s3
link-local      *               255.255.0.0     U     1000   0        0 enp0s3
192.168.53.0    *               255.255.255.0   U     0      0        0 tun0
192.168.53.0    *               255.255.255.0   U     0      0        0 tun0
```

Figure 16: Routing on Server VM (10.0.2.5)

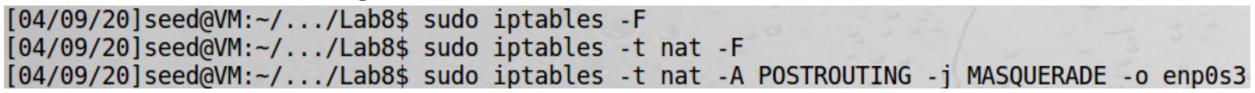
Step 4: Set Up NAT on Server VM (`10.0.2.5`)

When the final destination sends packets back to users, the packet will be sent to the VPN Server first.

The packets are sent by the VPN server over its enp0s3 interface before it goes through VirtualBox's NAT server, where the source IP of the packets (`192.168.53.5`) will be replaced by the IP address of the host computer (`10.0.2.5`). The packet will eventually arrive at the blocked website and the reply packet will return to our host computer (`10.0.2.5`) before it is given to VirtualBox's NAT server where the destination address is translated back to `192.168.53.5`. However, VirtualBox's NAT server does not know anything about the `192.168.53.0/24` network because this was created for the our `TUN` interface. Even if an ARP request is sent out to request for the MAC address of the machine who owns `192.168.53.5`, the private network is virtual and thus `192.168.53.5` will not receive the ARP request. As a result, this packet will be dropped.

In order to counter this issue, instead of returning the packet directly to the host computer (`10.0.2.5`), the packet will be first sent to the VPN Server (`192.168.53.1`). This is done with our own set up of the NAT server on the `enp0s3` interface of the VPN server. As the packets are received from the client's TUN interface (`192.168.53.5`), the source IP address on these packets are replaced by the IP address of our VPN server (`192.168.53.1`), instead of our host computer (`10.0.2.5`). As such, when the packets are sent back to the user, it can be sent to our VPN server (`192.168.53.1`) which is able to use its NAT server to forward the packets back to the client's TUN interface (`192.168.53.5`) and subsequently to the client themselves (`10.0.2.4`).

We need to create another NAT on the Server VM (`10.0.2.5`) so that all packets coming out of the Server VM (`10.0.2.5`) will have this VM's IP address, `10.0.2.5` as their source IP. To enable the NAT on the Server VM (`10.0.2.5`), the following commands, `sudo iptables -F`, `sudo iptables -t nat -F` and `sudo iptables -t nat -A POSTROUTING -j MASQUERADE -o enp0s3` as shown in Figure 17 are run.

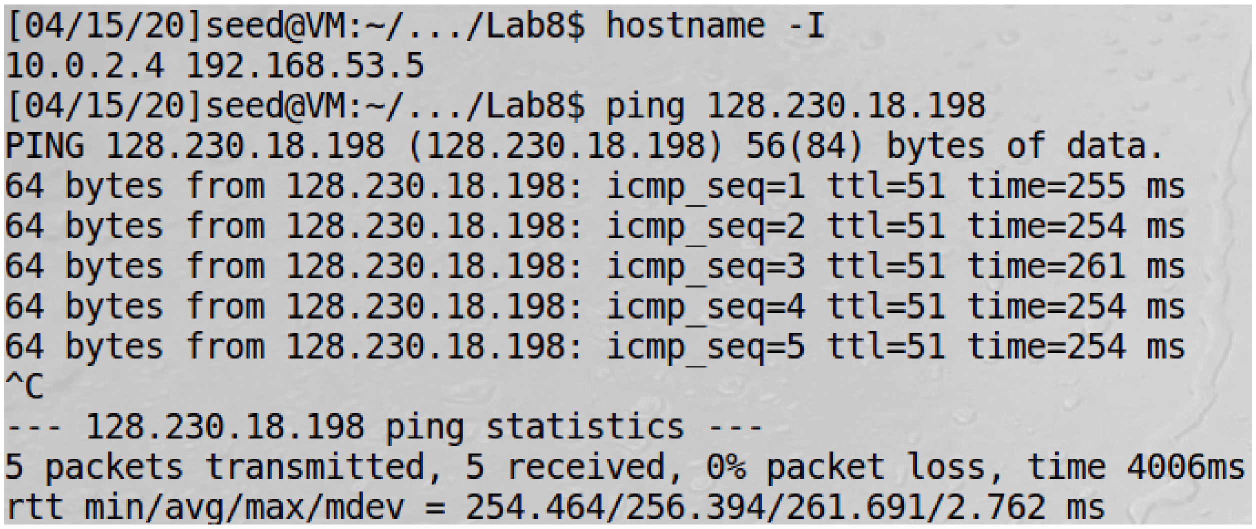


```
[04/09/20]seed@VM:~/.../Lab8$ sudo iptables -F
[04/09/20]seed@VM:~/.../Lab8$ sudo iptables -t nat -F
[04/09/20]seed@VM:~/.../Lab8$ sudo iptables -t nat -A POSTROUTING -j MASQUERADE -o enp0s3
```

Figure 17: Create another NAT on Server VM (10.0.2.5)

	IP Address	**VPN Tunnel IP Address**
VPN Client	`10.0.2.4`	`192.169.53.5`
VPN Server	`10.0.2.5`	`192.169.53.1`

Figure 18 shows the us attempting to and successfully pinging the blocked website, `www.syr.edu` (`128.230.18.198`) from the client VM (`10.0.2.4`).

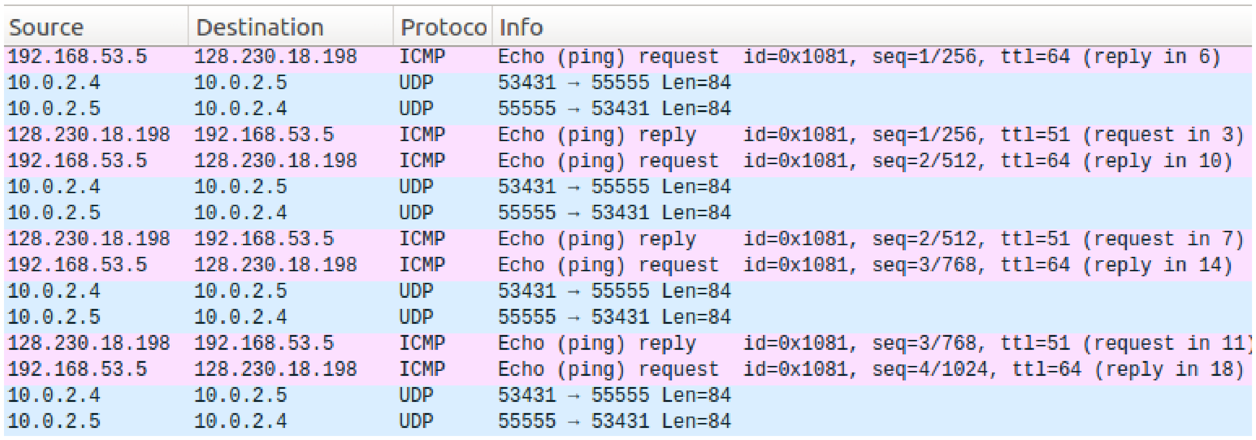


```
[04/15/20]seed@VM:~/.../Lab8$ hostname -I
10.0.2.4 192.168.53.5
[04/15/20]seed@VM:~/.../Lab8$ ping 128.230.18.198
PING 128.230.18.198 (128.230.18.198) 56(84) bytes of data.
64 bytes from 128.230.18.198: icmp_seq=1 ttl=51 time=255 ms
64 bytes from 128.230.18.198: icmp_seq=2 ttl=51 time=254 ms
64 bytes from 128.230.18.198: icmp_seq=3 ttl=51 time=261 ms
64 bytes from 128.230.18.198: icmp_seq=4 ttl=51 time=254 ms
64 bytes from 128.230.18.198: icmp_seq=5 ttl=51 time=254 ms
^C
--- 128.230.18.198 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 254.464/256.394/261.691/2.762 ms
```

Figure 18: Successful ping to blocked website from Client VM (10.0.2.4)

Figure 19 shows the captured network traffic using Wireshark when we ping the blocked website, `www.syr.edu` (`128.230.18.198`) from the client VM (`10.0.2.4`).

In the first line of Figure 19, we can see that the `tun0` is being used to access the blocked website, `www.syr.edu` (`128.230.18.198`) from the `tun0` client interface (`192.168.53.5`). The second and third lines of Figure 19 show how the VPN client and server communicate with each other using UDP, through their respective sockets. Through the tun interface, IP packets are received from the hosting system and sent through the tunnel before they are retrieved and forwarded to the hosting system which forwards the packet to its destination.



Source	Destination	Protoco	Info
192.168.53.5	128.230.18.198	ICMP	Echo (ping) request id=0x1081, seq=1/256, ttl=64 (reply in 6)
10.0.2.4	10.0.2.5	UDP	53431 → 55555 Len=84
10.0.2.5	10.0.2.4	UDP	55555 → 53431 Len=84
128.230.18.198	192.168.53.5	ICMP	Echo (ping) reply id=0x1081, seq=1/256, ttl=51 (request in 3)
192.168.53.5	128.230.18.198	ICMP	Echo (ping) request id=0x1081, seq=2/512, ttl=64 (reply in 10)
10.0.2.4	10.0.2.5	UDP	53431 → 55555 Len=84
10.0.2.5	10.0.2.4	UDP	55555 → 53431 Len=84
128.230.18.198	192.168.53.5	ICMP	Echo (ping) reply id=0x1081, seq=2/512, ttl=51 (request in 7)
192.168.53.5	128.230.18.198	ICMP	Echo (ping) request id=0x1081, seq=3/768, ttl=64 (reply in 14)
10.0.2.4	10.0.2.5	UDP	53431 → 55555 Len=84
10.0.2.5	10.0.2.4	UDP	55555 → 53431 Len=84
128.230.18.198	192.168.53.5	ICMP	Echo (ping) reply id=0x1081, seq=3/768, ttl=51 (request in 11
192.168.53.5	128.230.18.198	ICMP	Echo (ping) request id=0x1081, seq=4/1024, ttl=64 (reply in 18)
10.0.2.4	10.0.2.5	UDP	53431 → 55555 Len=84
10.0.2.5	10.0.2.4	UDP	55555 → 53431 Len=84

Figure 19: PCAP of Client-Server Communication