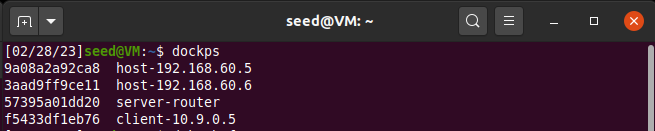
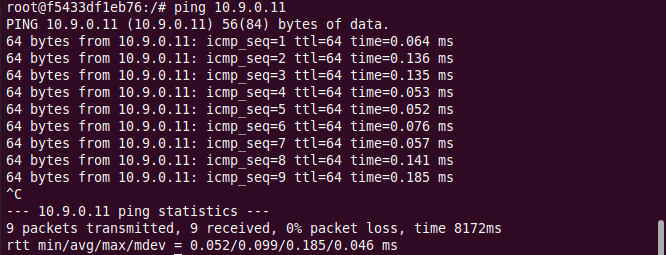
**ACS 54500 Cryptography and Network Security – Lab 6**

**Task 1:**

In this task, we have to setup the lab environment and test a few cases to see if it is running correctly. Here is the lab setup –

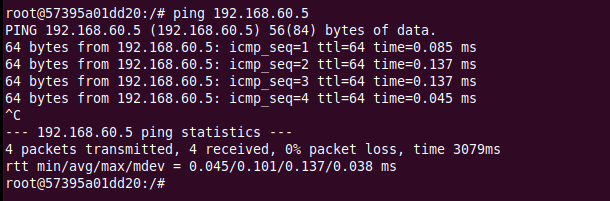


The first test is to see if Host U can communicate with the VPN Server. To test this, we docksh into Host U (10.9.0.5) and ping the server (10.9.0.11) as shown below –



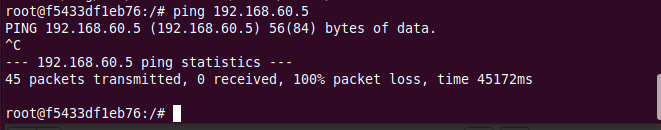
This ping goes through, so that means that Host U can communicate with the VPN Server.

The second test is to see if the VPN Server can communicate with Host V. We docksh into the VPN Server in a new terminal instance and ping Host V (192.168.60.5) –



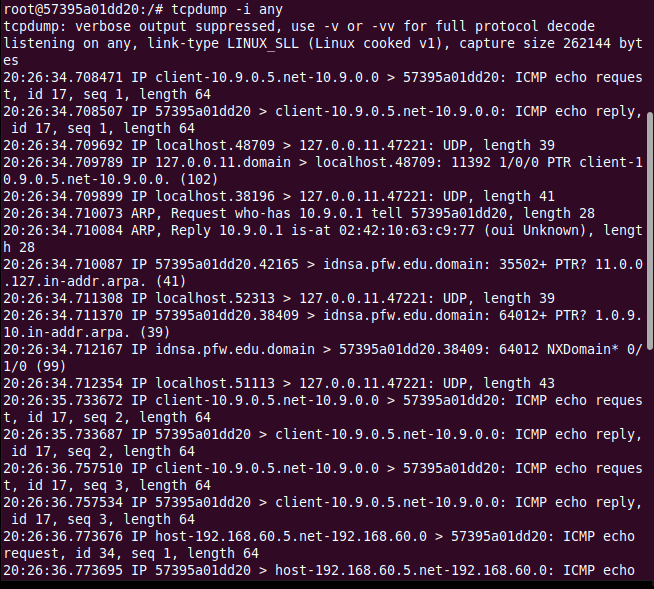
This works too, so that means the VPN Server can communicate with Host V too.

The third test is to see if Host U is unable toc communicate with Host V. So the ping must fail.



This does fail, so U cannot communicate with V.

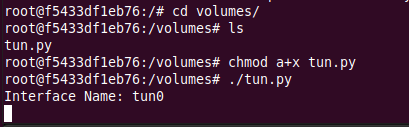
The last test is to run tcpdump on the router, and sniff the traffic on each of the network, showing that we can capture packets. To do this, we ping the server-router from both hosts and run tcpdump –



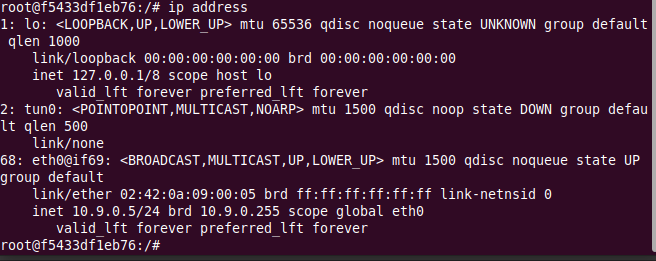
**Task 2:**

**Task 2.A:**

In this sub-task we run tun.py on Host U after giving it executable privileges.



As we can see from the above screenshot, there is now a new interface called tun0. This should show up as one of the interfaces after running the command ‘ip address’ –



There is a new interface running called tun0. Now, the task is to modify the prefix of this interface’s name to show the first 5 letters of my last name, which is ‘ramch’.

Here is the modified code for tun.py –

#!/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'ramch%d', IFF\_TUN | IFF\_NO\_PI)

ifname\_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)

# Get the interface name

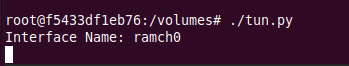
ifname = ifname\_bytes.decode('UTF-8')[:16].strip("\x00")

print("Interface Name: {}".format(ifname))

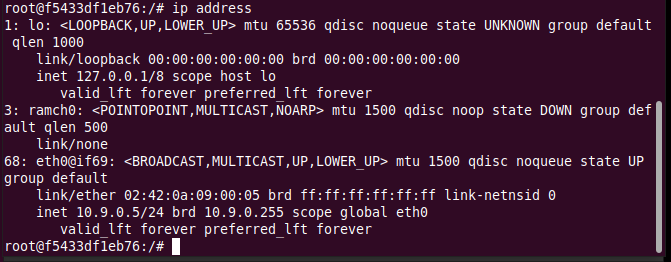
while True:

   time.sleep(10)

Now we run this code again –



There is a new interface called ramch0. We run the ip address command to see if it shows up on the list of interfaces -



It does show up. Thus, we have modified the name.

**Task 2.B:**

We need to set up the TUN interface now. First, we assign an IP address to the interface with the following modified code and run it again –

#!/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))

# Bring up the interface and assign it an IP address

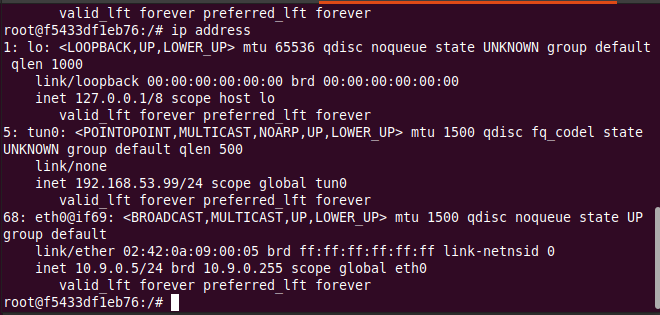
os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))

os.system("ip link set dev {} up".format(ifname))

while True:

   time.sleep(10)

We run this program again, and check the interfaces using ip address -



We can see that the interface is assigned the IP address and the state is now UNKNOWN instead of DOWN.

Task 2.C:

We need to read from the TUN interface. To do this, we add in the given while loop to replace the one already in the tun.py. Here is the modified code -

#!/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))

# Bring up the interface and assign it an IP address

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

os.system("ip link set dev {} up".format(ifname))

while True:

   # Get a packet from the tun interface

   packet = os.read(tun, 2048)

   if packet:

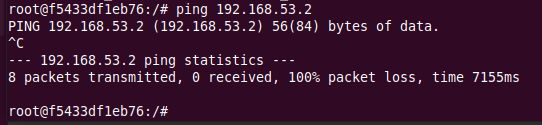
    ip = IP(packet)

    print(ip.summary())

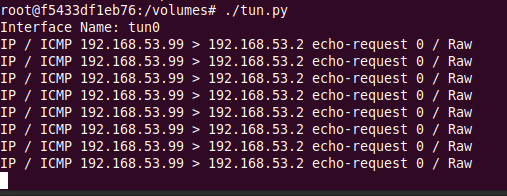
We run this program again.

Now we need to ping any host in the internal network of 192.168.53.0/24. I chose to ping 192.168.53.2.



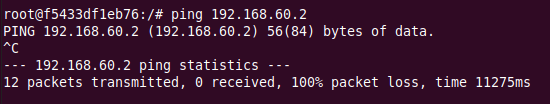


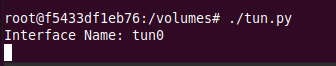
We can see that this is what is printed on the TUN interface -



This is because the TUN interface is setup and assigned the IP addres of 192.168.53.99. Since this is in the network that we just pinged, the scapy program starts to sniff packets.

Now we ping a host in the internal network 192.168.60.0/24. I chose to ping 192.168.60.2.





We can see that the interface doesn’t print anything, because this address is not in the network that it is looking for.

**Task 2.D:**

In this sub-task, we need to write to the TUN interface. When an ICMP echo request packet is received by the interface, it needs to construct an appropriate ICMP echo reply, and some arbitrary data should be written to the interface.

To do this, we need to modify the code for tun.py and add in code that takes in IP request packets, and does two things in two situations – if the packet is an ICMP echo request packet, then it sends back an ICMP echo reply. If the packet is not ICMP but still IP, then it sends back some arbitrary data.

Here is the modified code –

#!/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))

# Bring up the interface and assign it an IP address

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

os.system("ip link set dev {} up".format(ifname))

while True:

   # Get a packet from the tun interface

   packet = os.read(tun, 2048)

   if packet:

      pkt = IP(packet)

      print(pkt.summary())

      if ICMP in pkt and pkt[ICMP].type == 8:

        newip = IP(src = pkt[IP].dst, dst = pkt[IP].src, ihl = pkt[IP].ihl, ttl = 50)

        icmp\_reply = ICMP(src = pkt[IP].dst, sqe = pkt[ICMP].seq, type  = 0)

        data = pkt[Raw].load

        newpkt = newip/icmp\_reply/data

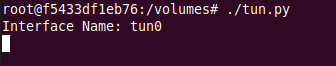
        os.write(tun, bytes(newpkt))

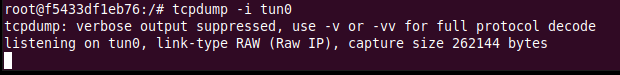
      else:

        data = b"hello"

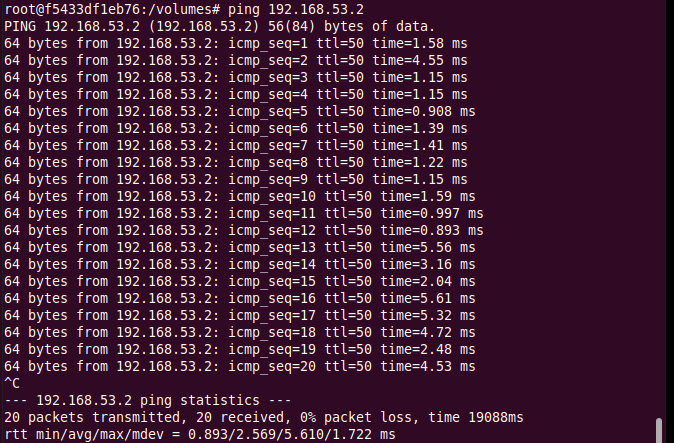
        os.write(tun, bytes(data))

Now, we run this program, and then run tcpdump on the tun0 interface, as shown below –

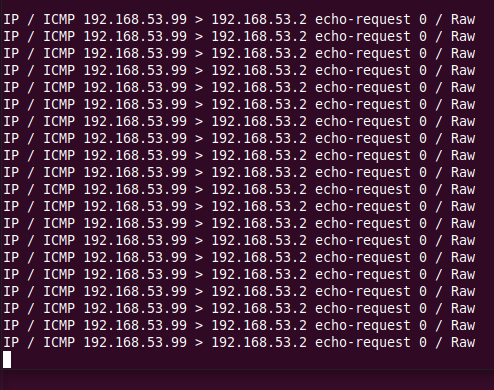




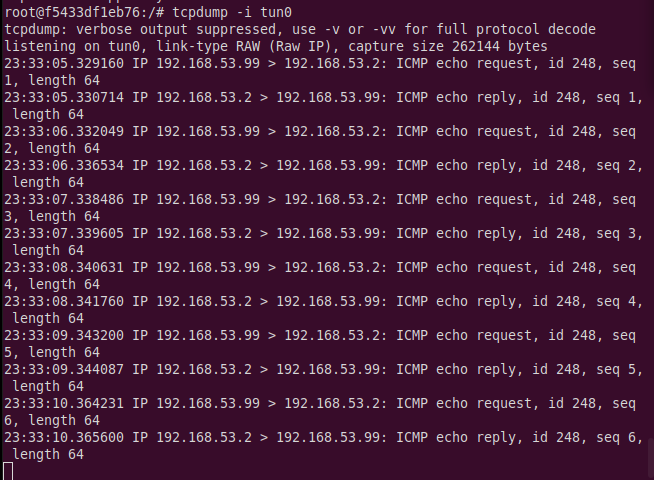
To test this, we ping and telnet to any IP address in the internal network of 192.168.53.0/24. I chose 192.168.53.2. The ping is shown below –



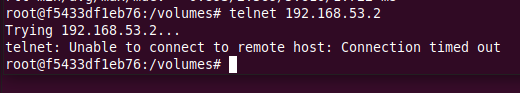
The ping goes through, and now we check the tun0 interface and tcpdump to see what shows up –



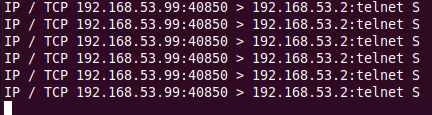
This shows that when an ICMP request is sent from Host U to the network, the tun0 interface captures it because we have linked this in code. Then it sends a reply packet back in response. Tcpdump shows how traffic is captured. This reply packet is written to the interface.



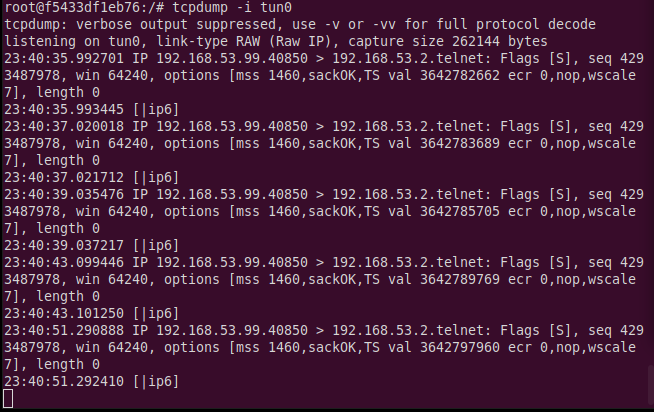
Now, we need to telnet to 192.168.53.2 and test if the arbitrary data that we put in is also written to the interface.



This request does not succeed. Now we check the tun0 interface and tcpdump to see what shows up -



This shows that when an IP request packet other than ICMP is sent from Host U to the network, the tun0 interface captures it because we have linked this in code. Then it writes back the data we wrote in code as an IP packet in response. This can be seen in tcpdump, which shows how traffic is captured. This reply packet is written to the interface.



**Task 3:**

In this task, we have to set up the TUN Server and the TUN Client. We do this by writing and running the following two programs –

#!/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))

tun\_server.py

import socket

#!/usr/bin/env python3

import fcntl

import struct

import os

import time

from scapy.all import \*

SERVER\_IP = "10.9.0.11"

SERVER\_PORT = 9090

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))

# Bring up the interface and assign it an IP address

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 packet:

        # Send the packet via the tunnel

        sock.sendto(packet, (SERVER\_IP, SERVER\_PORT))

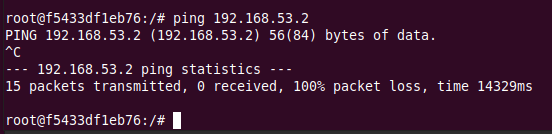
tun\_client.py

Now we run these two programs by running the tun server.py program on the VPN Server, and then running tun client.py on Host U. This is shown as below –

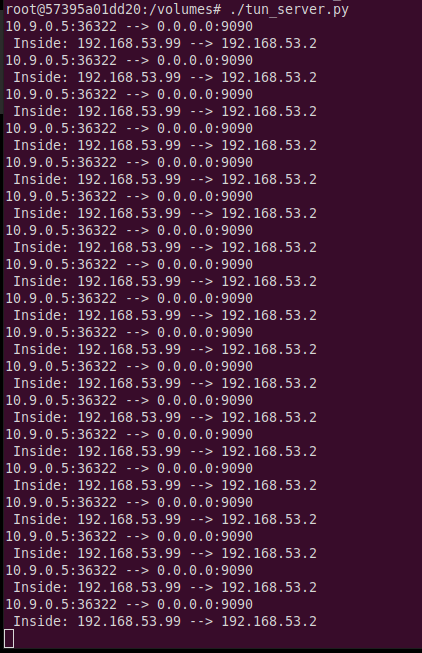




We now ping any host in the internal network of 192.168.53.0/24 from Host U and see what happens. Nothing shows up on the client tun interface but we can see that there is a lot that shows up on the server tun interface.

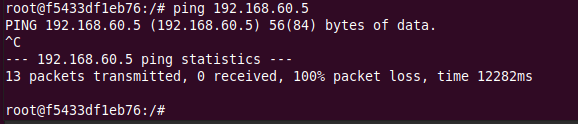


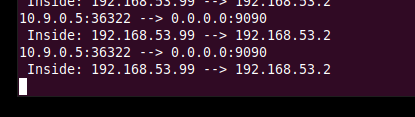




This is because we receive IP packets from host U and print the summary.

Now we try pinging Host V, and see whether the ICMP packet is sent to VPN Server through the tunnel.





Nothing shows up on both the server and client interfaces. So, this means that we still can’t access Host V from U. We need to use routing to add a route from to route packets from this network to the TUN interface and given to the tun client program. This can be done with the ip route command, we add that in code as shown below –

#!/usr/bin/env python3

import fcntl

import struct

import os

import time

from scapy.all import \*

SERVER\_IP = "10.9.0.11"

SERVER\_PORT = 9090

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))

# Bring up the interface and assign it an IP address

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

os.system("ip link set dev {} up".format(ifname))

# Set up routing

os.system("ip route add 192.168.60.0/24 dev {}".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 packet:

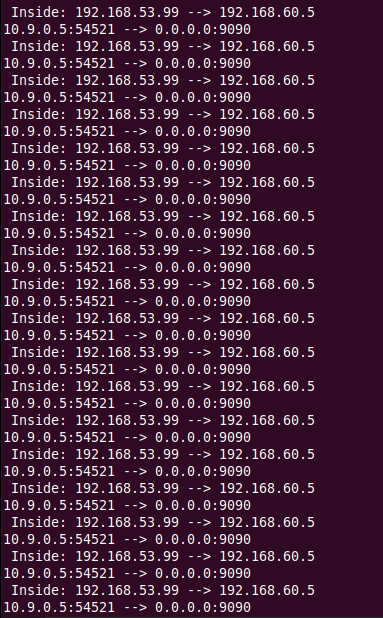
        # Send the packet via the tunnel

        sock.sendto(packet, (SERVER\_IP, SERVER\_PORT))

tun\_client.py

Now we ping Host V from U and observe what the tun\_server interface shows -





So, when we ping an IP address in the 192.168.60.0/24 network, the ICMP packets are received by tun\_server.py through the tunnel.

**Task 4:**

In this sub-task, we need to set up the VPN server. After tun server.py gets a packet from the tunnel, it needs to feed the packet to the kernel, so the kernel can route the packet towards its final destination. This needs to be done through a TUN interface, just like what we did in Task 2.

Here is the modified tun\_server code –

#!/usr/bin/env python3

import fcntl

import struct

import os

import time

from scapy.all import \*

IP\_A = "0.0.0.0"

PORT = 9090

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))

os.system("ip addr add 192.168.53.1/24 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)

    pkt = IP(data)

    print("{}:{} --> {}:{}".format(ip, port, IP\_A, PORT))

    print("    Inside Tunnel: {} --> {}".format(pkt.src, pkt.dst))

    os.write(tun, data)

tun\_server.py

#!/usr/bin/env python3

import fcntl

import struct

import os

import time

from scapy.all import \*

import socket

SERVER\_IP = "10.9.0.11"

SERVER\_PORT = 9090

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))

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

os.system("ip link set dev {} up".format(ifname))

# Set up routing

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

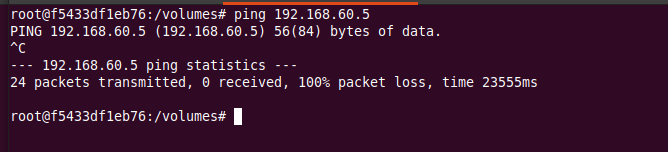
    if packet:

        # Send the packet via the tunnel

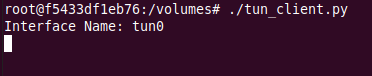
        sock.sendto(packet, (SERVER\_IP, SERVER\_PORT))

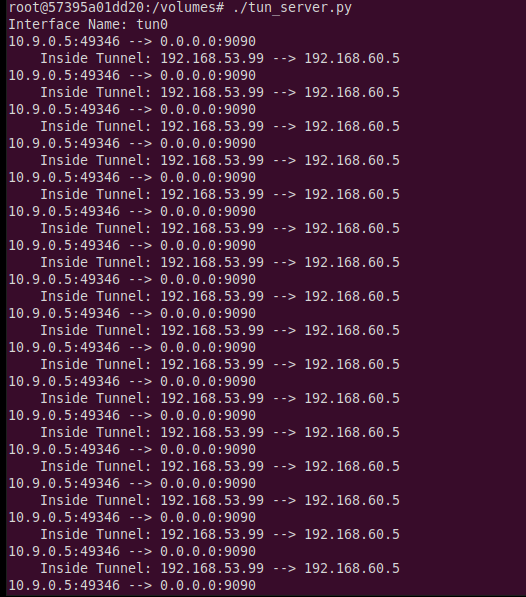
tun\_client.py

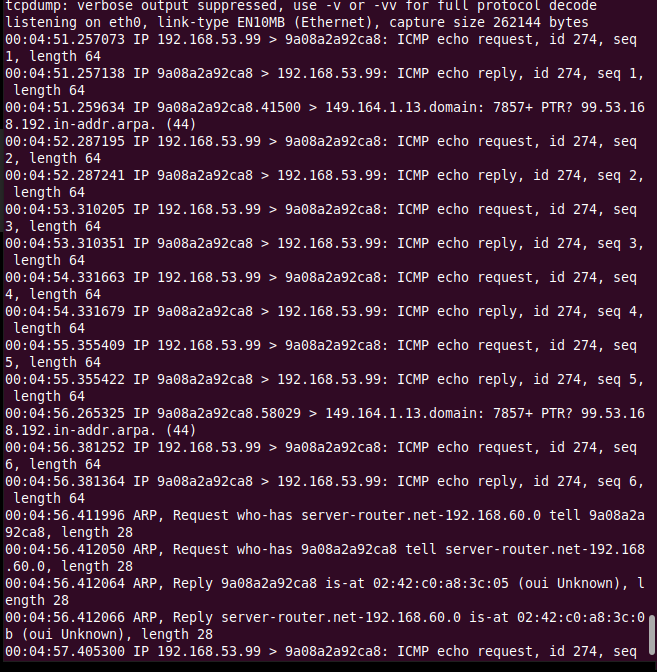
Now we run both files, and run tcpdump. We ping Host V from U to see if our setup works.



This ping fails. We check the interface terminals. Nothing shows up on tun client but we can see that this ping reaches the tun server, as shown in the screenshot, but it is unable to reach Host V as of now.







**Task 5:**

In this task, we need to modify the given code and replace the while loop in both the TUN server and client programs to be able to communicate with Machine V from Machine U, and the VPN tunnel (un-encrypted) should be completed. Here are the modified codes –

#!/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

IP\_A = "0.0.0.0"

PORT = 9090

# Create 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)

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.select([sock, tun], [], [])

    for fd in ready:

        if fd is sock:

            print("sock ...")

            data, (ip, port) = sock.recvfrom(2048)

            pkt = IP(data)

            print("{}:{} --> {}:{}".format(ip, port, IP\_A, PORT))

            print("    Inside Tunnel: {} --> {}".format(pkt.src, pkt.dst))

            os.write(tun, data)

        if fd is tun:

            print("tun ...")

            packet = os.read(tun, 2048)

            pkt = IP(packet)

            print("Return: {} --> {}".format(pkt.src, pkt.dst))

            sock.sendto(packet, (ip, port))

tun\_server.py

#!/usr/bin/env python3

import fcntl

import struct

import os

import time

from scapy.all import \*

SERVER\_IP = "10.9.0.11"

SERVER\_PORT = 9090

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))

# Bring up the interface and assign it an IP address

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

os.system("ip link set dev {} up".format(ifname))

# Set up routing

os.system("ip route add 192.168.60.0/24 dev {}".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 packet:

        # Send the packet via the tunnel

        sock.sendto(packet, (SERVER\_IP, SERVER\_PORT))

tun\_client.py

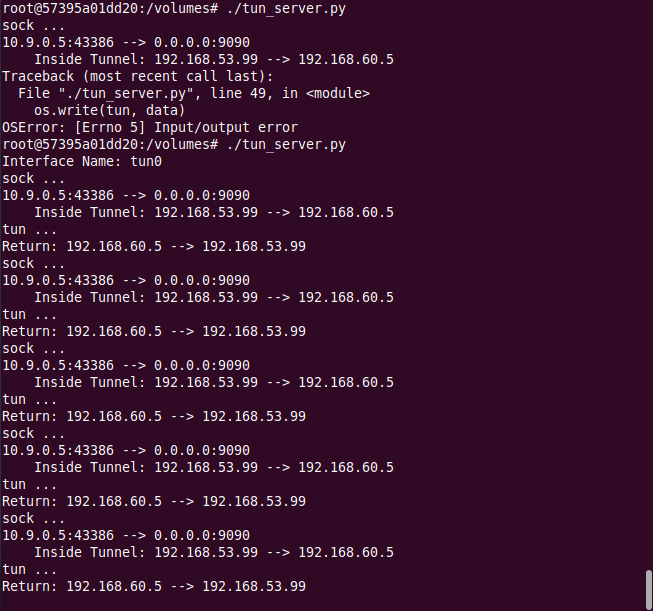
Now, we run both programs.

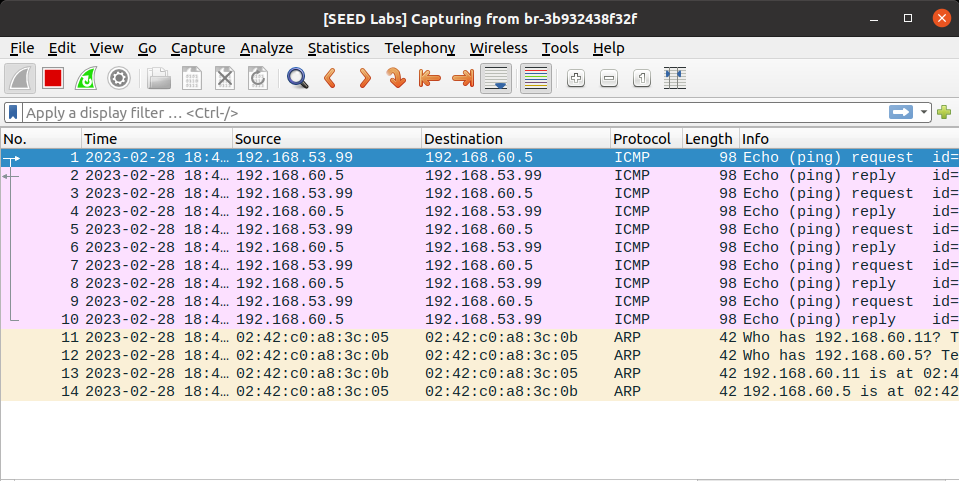




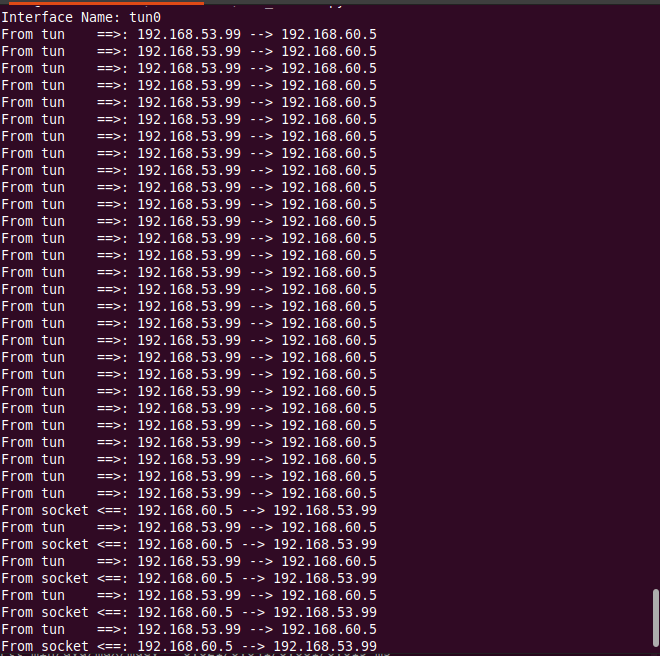
We need to ping and telnet Host V from Host U. First, we ping and record the traffic using wireshark.

This is what we observe on the server interface and wireshark –



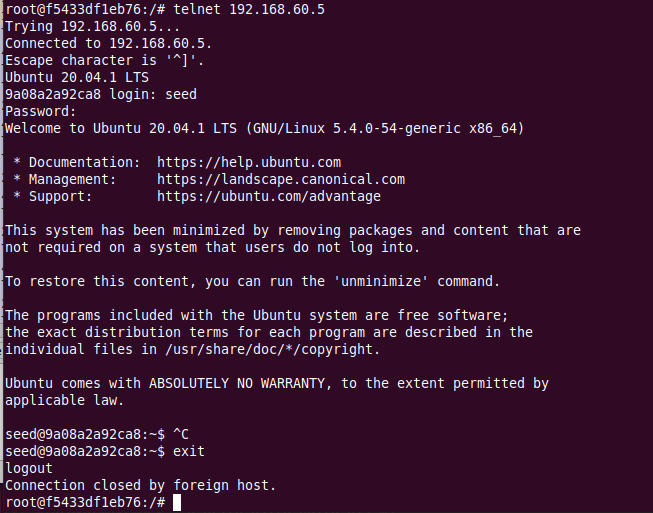


This is what we observe on the tun client interface -

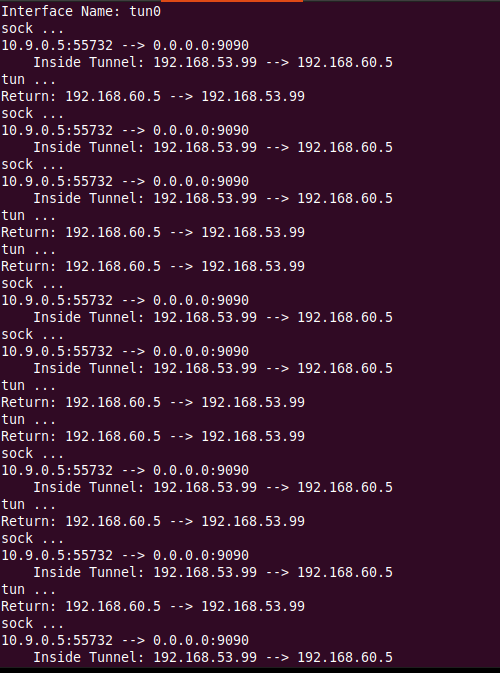


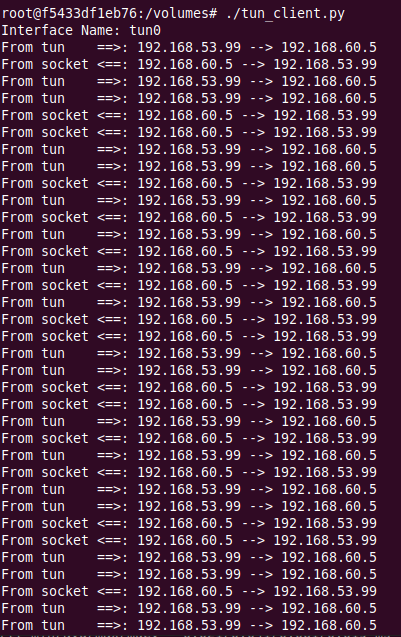
This shows that Host U can communicate with V via ping, because we have set up a connection between V and the server, and another between the server and U.

Now we try telnet –

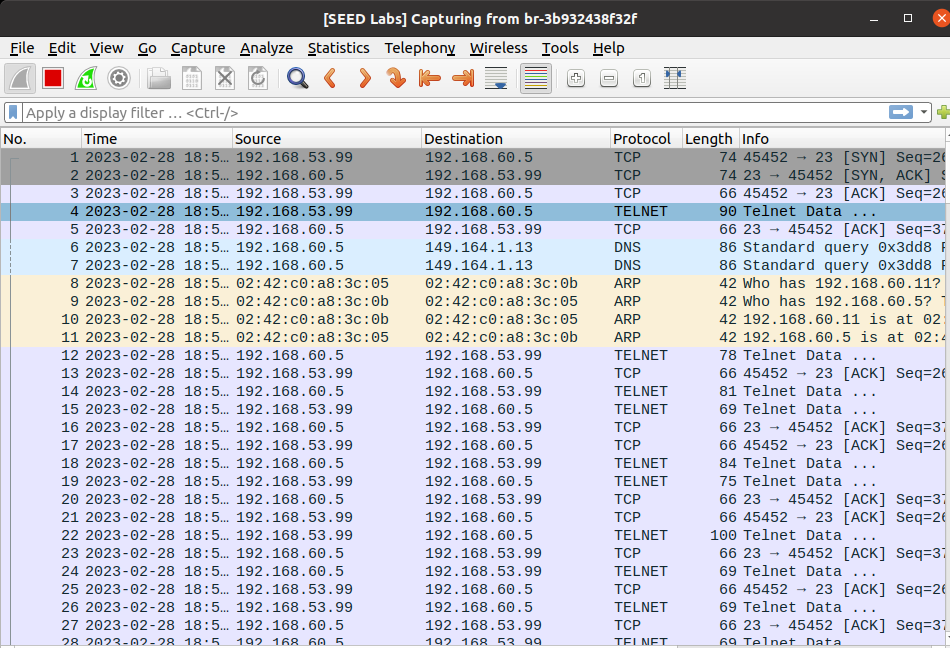


This is what we observe on both of the interfaces –





This is what the wireshark observation looks like –

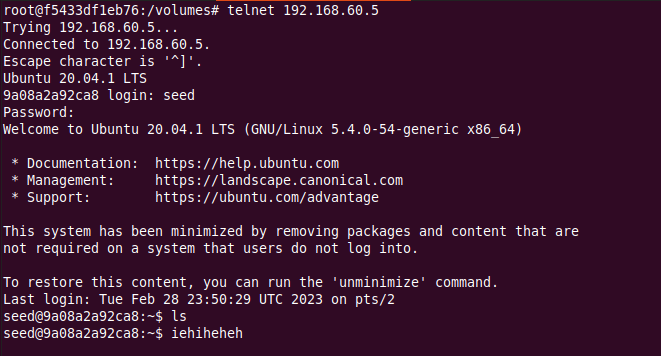


This shows that Host U can also communicate with V via telnet, because again, we have set up a connection between V and the server, and another between the server and U.

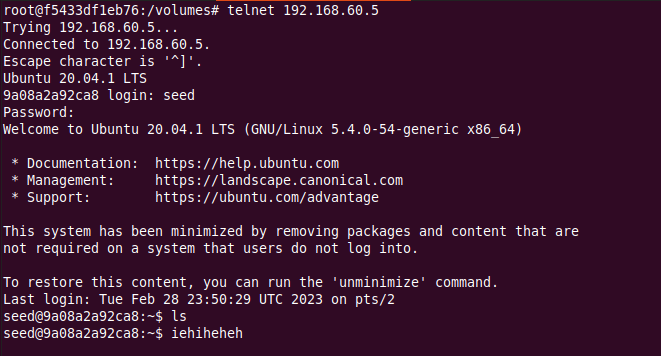
**Task 6:**

In this task, we need to break the tunnel connection established in task 5, observe what happens with telnet and report the results.   
We use the same codes for tun\_server and tun\_client as in task 5.

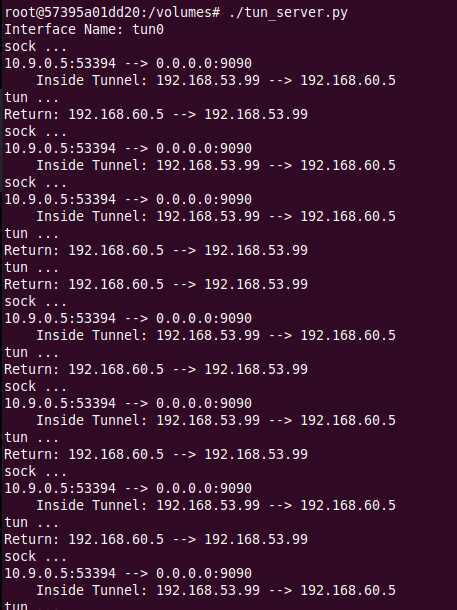
First, we run both the tun server and client programs on the server-router and Host U respectively. Now, we telnet to 192.168.60.5 (Host V) from U. The connection works and is established. This is as shown below -

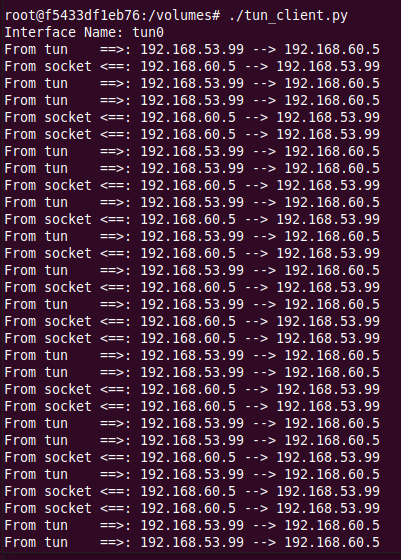


Now, we type a command in. I typed in ls.

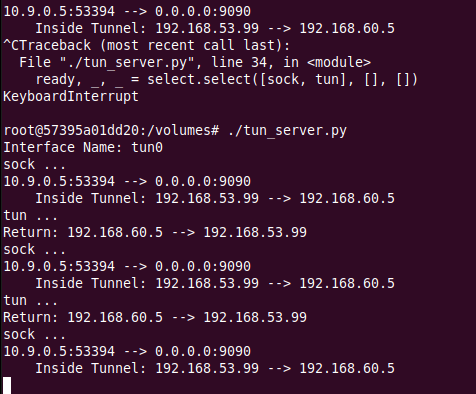


The tun server and client messages are shown below -

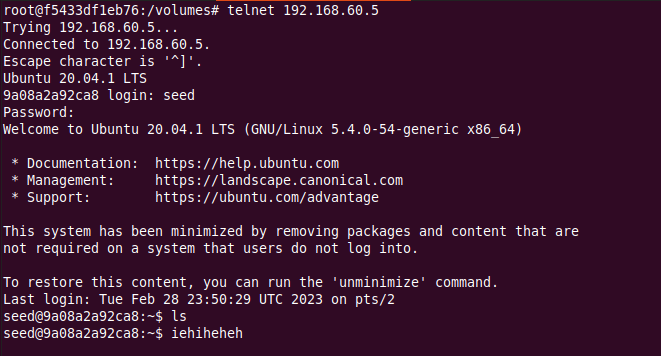




We now break the tunnel by stopping the tun\_server, and try to type something in the telnet window. Nothing shows up. We start the connection again as shown below.



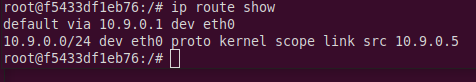
We can see that whatever we had typed earlier now shows up on the telnet window.



Thus, we have successfully broken the tunnel connection.

**Task 7:**

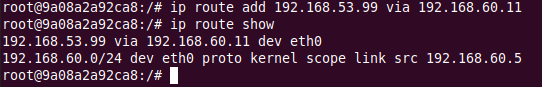
In this task, we have to delete the default route from Host V’s ip routing table, and add a custom route to 192.168.53.99. First, we look at how the default route looks like.



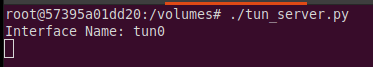
Now we delete the default one –



We can see that the default route has been deleted. The below screenshot shows how we can add the necessary custom route –

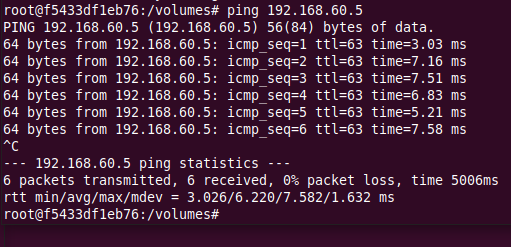


The tun server and client interfaces are already running.





We ping Host V from Host U to check if our route works –



This works, as shown above, meaning that our route works. Here is how our server and client interfaces look like –

