Name: Bharath Karumudi
Lab: Virtual Private Network (VPN)

Task 1: VM Setup

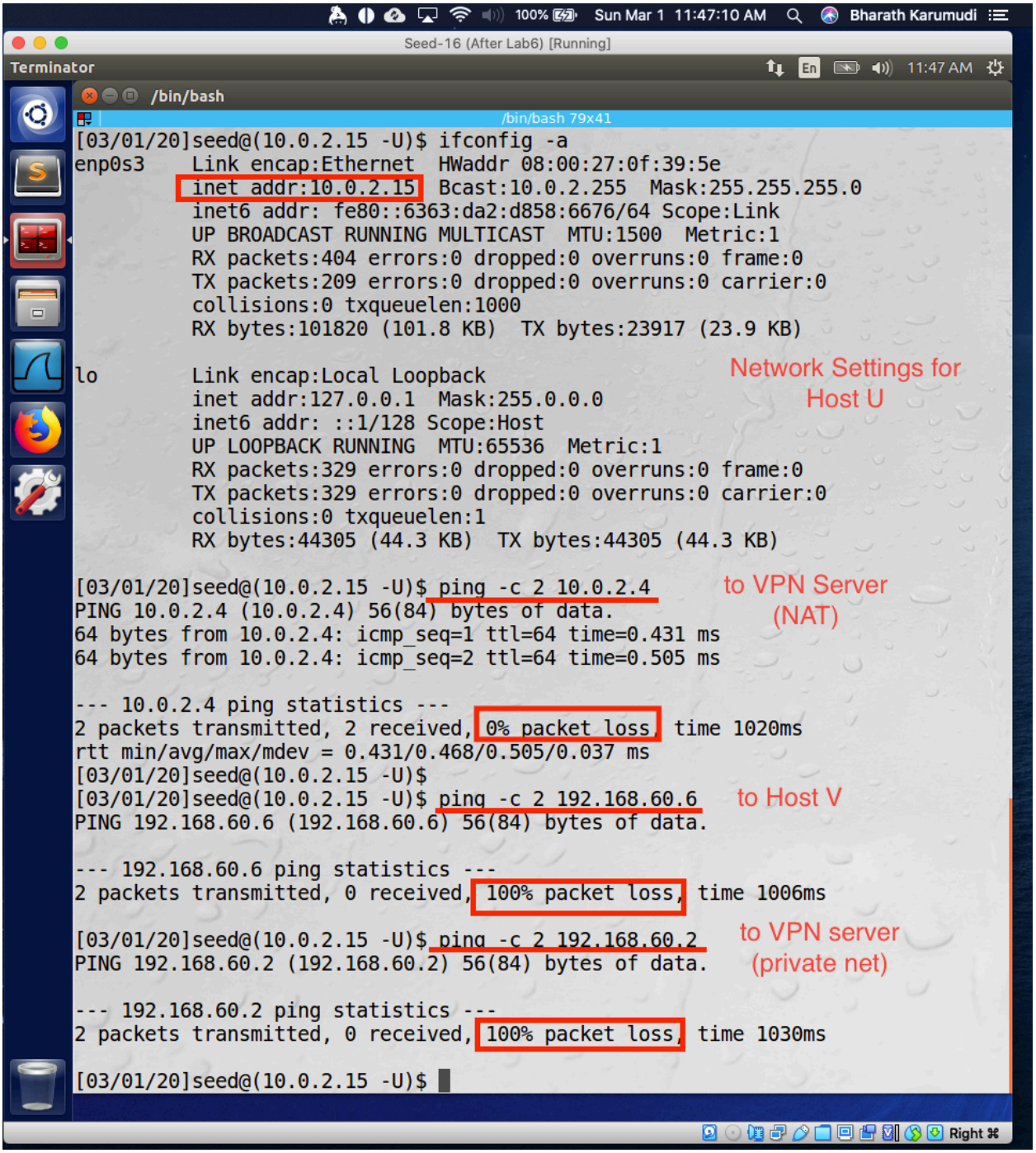

Fig1: Network Connections and Validations of Host U (VPN Client)

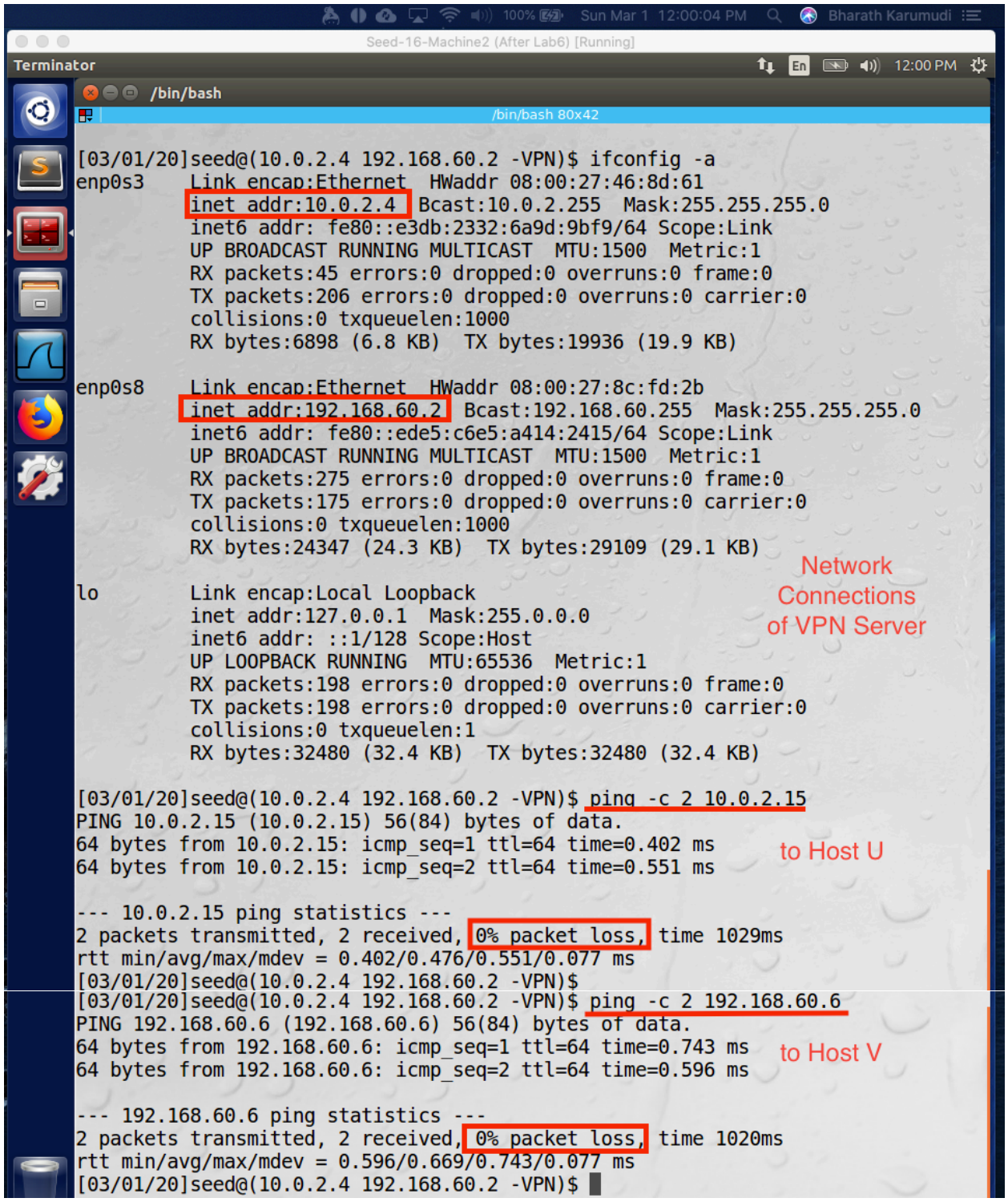

Fig2: Network Connections and Validations of VPN Server

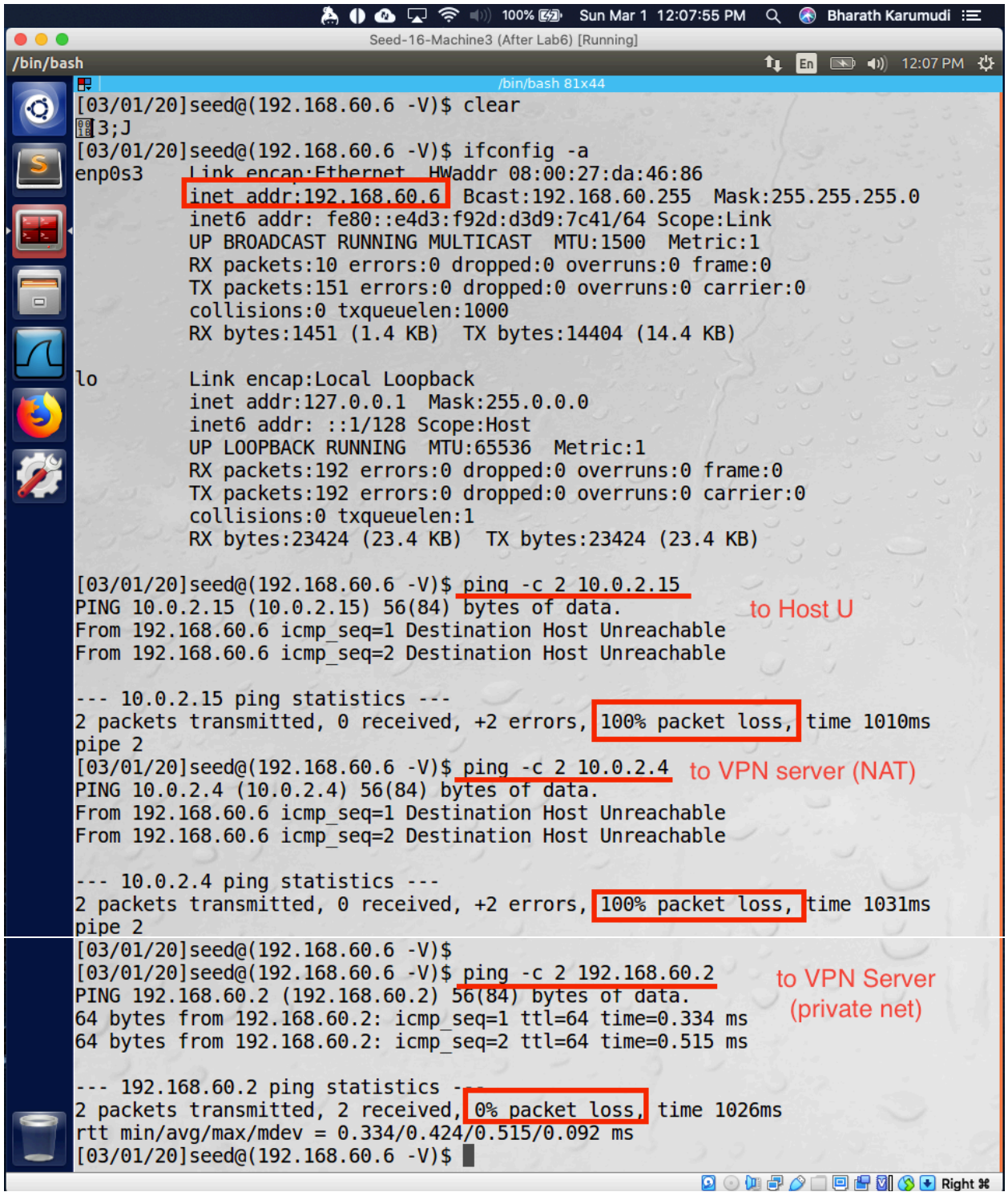

Fig3: Network Connections and Validations of Host V (Private network host)

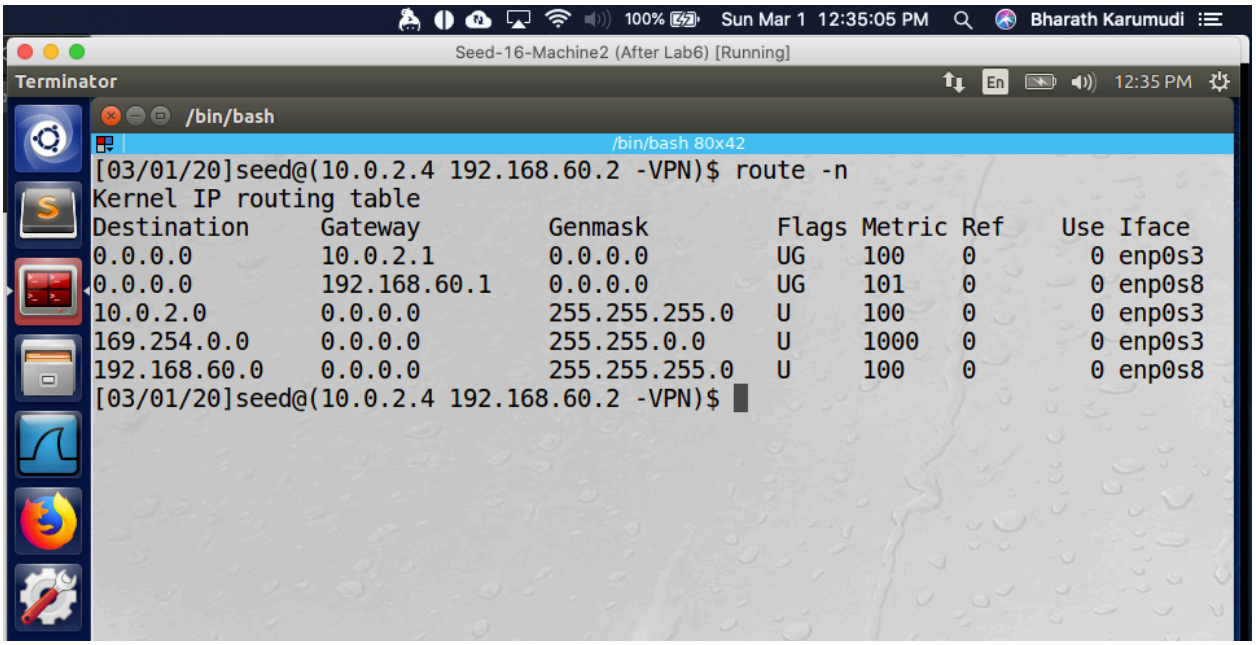

Fig 4: Routing table of VPN server

Observation: Set up the network connections for VPN server, VPN Client (Host U) and Private network host (Host V). After the setup, the Host U is able to communicate with VPN server on NAT (simulated Internet) but not to Host V. VPN server is able to communicate with both Host U and Host V via corresponding networks. Host V is able to communicate with VPN server via internal network but not to Host U. So, the only way that Host U and Host V will be communicated is via VPN server.

Explanation:

1. Created a NAT network of 10.0.2.0/24
2. Create a private network (privnet) of 192.168.60.0/24
3. The Host U is placed in NAT network and assigned IP as 10.0.2.15 (Fig1).
4. The Host V is place in private network and assigned IP as 192.168.60.6 (Fig 3).
5. The VPN server is placed on both the networks by having two network adapters and allocated the IP as 10.0.2.4 (on NAT) and 192.168.60.2 (on privnet) as shown in Fig 2 and routing table is as shown in Fig 4.

Task 2: Creating a VPN Tunnel using TUN/TAP

Step1: Run VPN Server

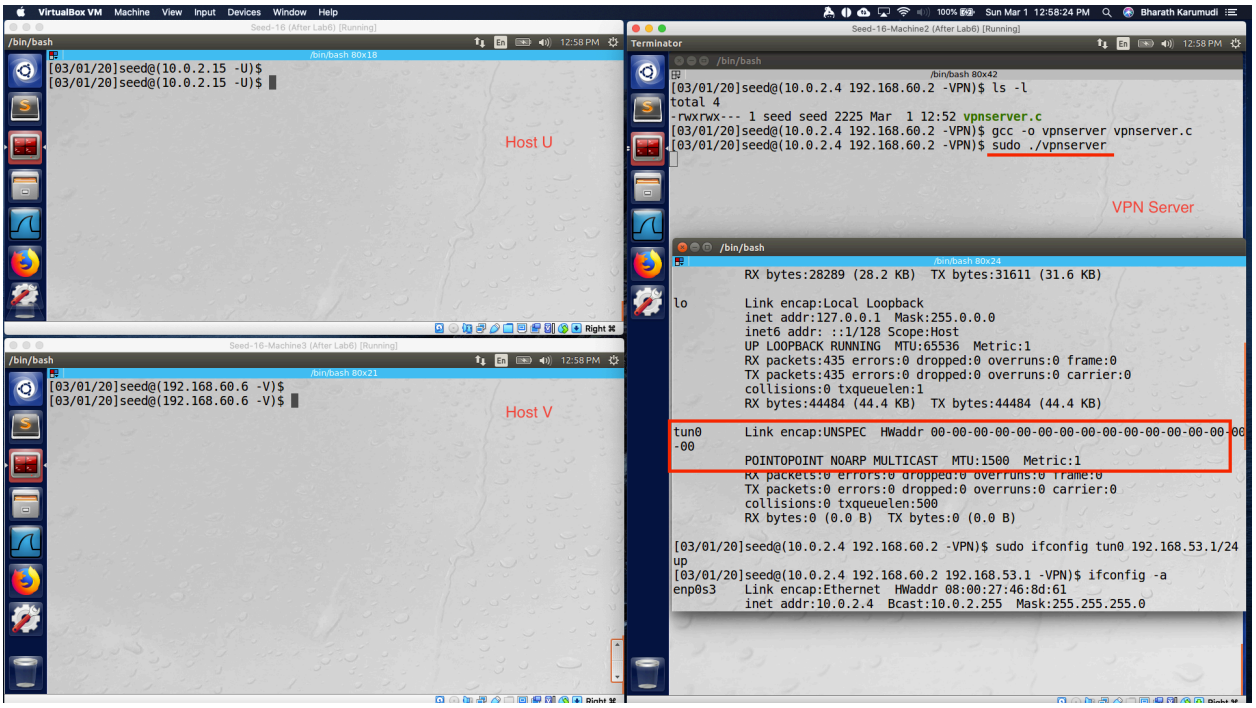

Fig1: Creating the TUN interface (tun0) on VPN server

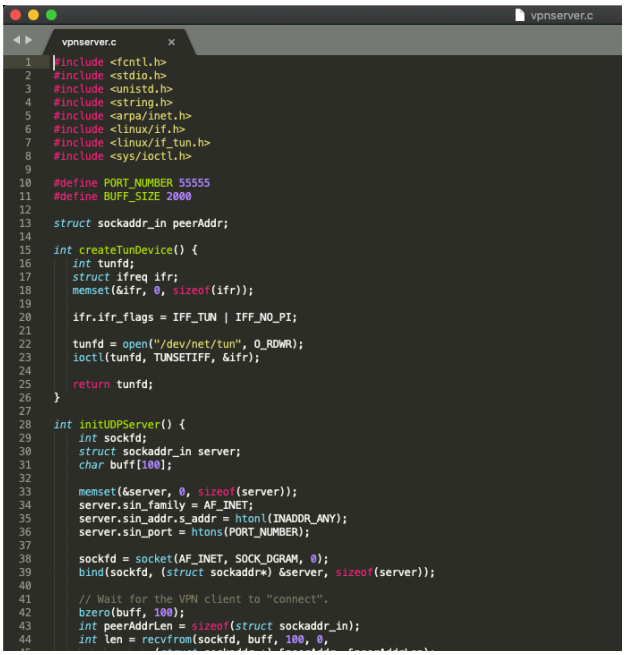

Fig1a: vpnserver.c program

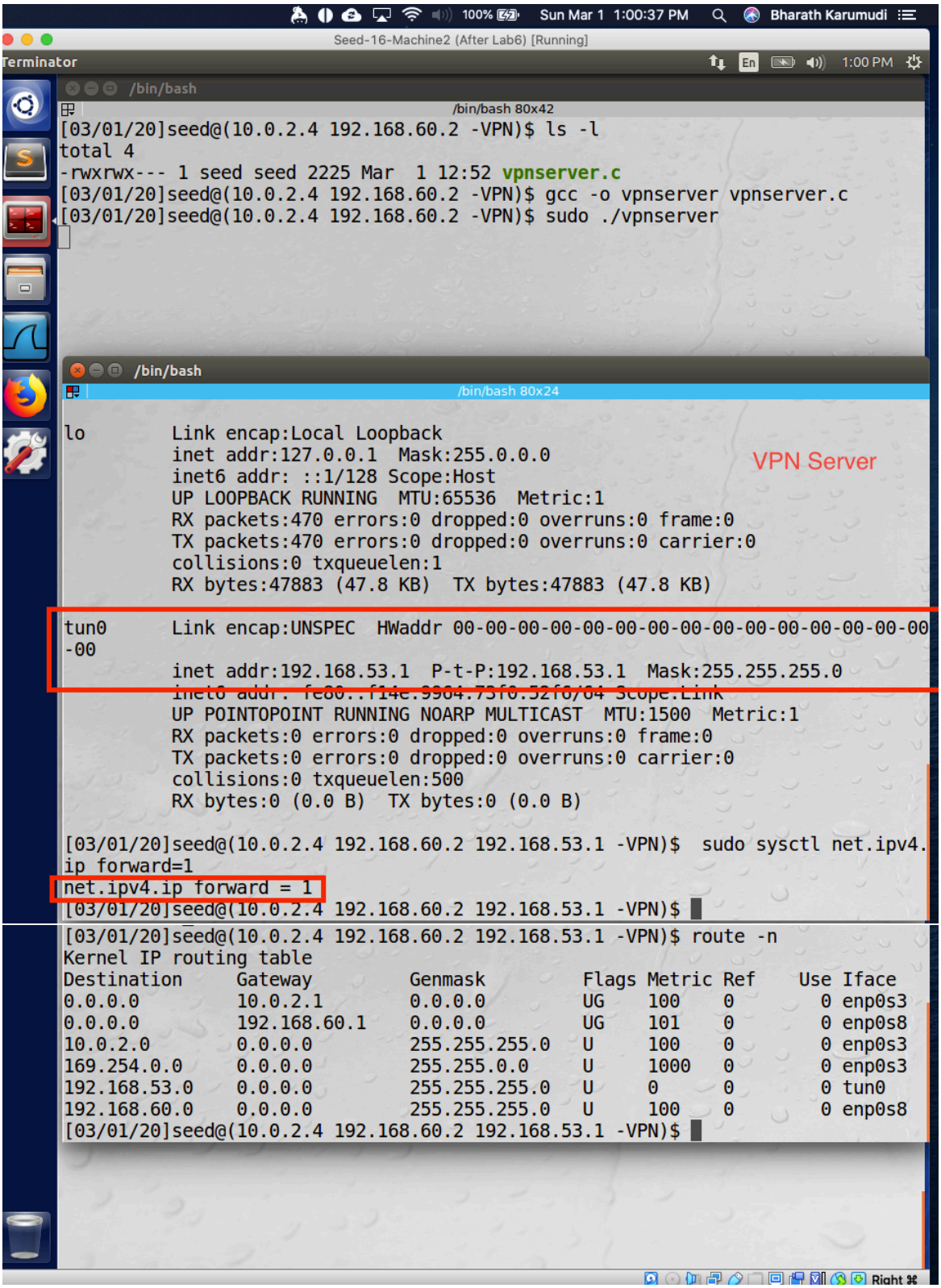

Fig2: Assigned IP to tun0 and VPN server routing table.

Observation: When compiled and executed the vpnserver.c program, the TUN interface tun0 was created and assigned the IP address as 192.168.53.1 for tun0 and made it up. Also made the host as gateway.

Explanation: The vpnserver.c was executed with sudo to create the tun0 interface as shown in Fig 1 and using the ifconfig assigned the 192.168.53.1 IP address to the tun0 interface and using sysctl made the VPN host as gateway so it forward the packets (Fig2).
sudo ifconfig tun0 192.168.53.1/24 up

Step2: Run VPN Client

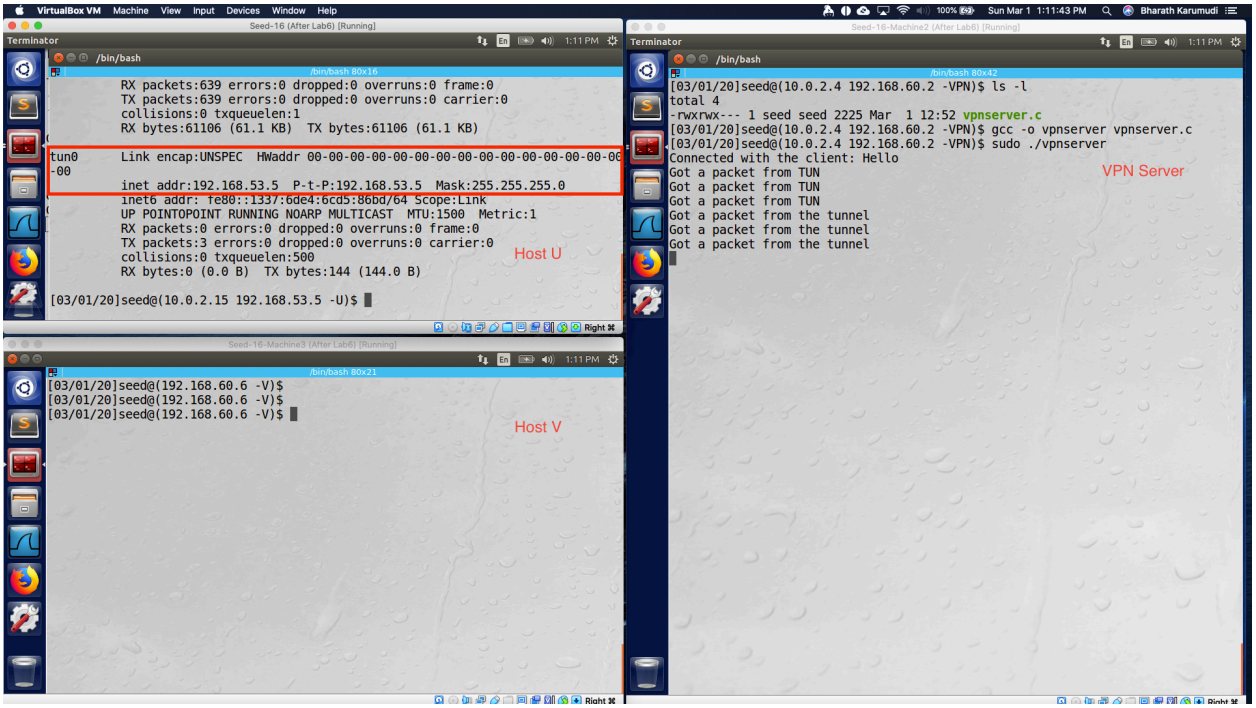

Fig3: Established TUN interface on VPN Client and is up and running

```c
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <linux/if.h>
#include <linux/if_tun.h>
#include <sys/ioctl.h>

#define BUFF_SIZE 2000
#define PORT_NUMBER 55555
#define SERVER_IP "10.0.2.4"
struct sockaddr_in peerAddr;

int createTunDevice() {
   int tunfd;
   struct ifreq ifr;
   memset(&ifr, 0, sizeof(ifr));

   ifr.ifr_flags = IFF_TUN | IFF_NO_PI;

   tunfd = open("/dev/net/tun", O_RDWR);
   ioctl(tunfd, TUNSETIFF, &ifr);

   return tunfd;
}

int connectToUDPServer(){
    int sockfd;
    char *hello="Hello";

    memset(&peerAddr, 0, sizeof(peerAddr));
    peerAddr.sin_family = AF_INET;
    peerAddr.sin_port = htons(PORT_NUMBER);
    peerAddr.sin_addr.s_addr = inet_addr(SERVER_IP);

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);

    // Send a hello message to "connect" with the VPN server
    sendto(sockfd, hello, strlen(hello), 0,
                (struct sockaddr *) &peerAddr, sizeof(peerAddr));

    return sockfd;
}
```

Fig 3a: vpnclient.c program

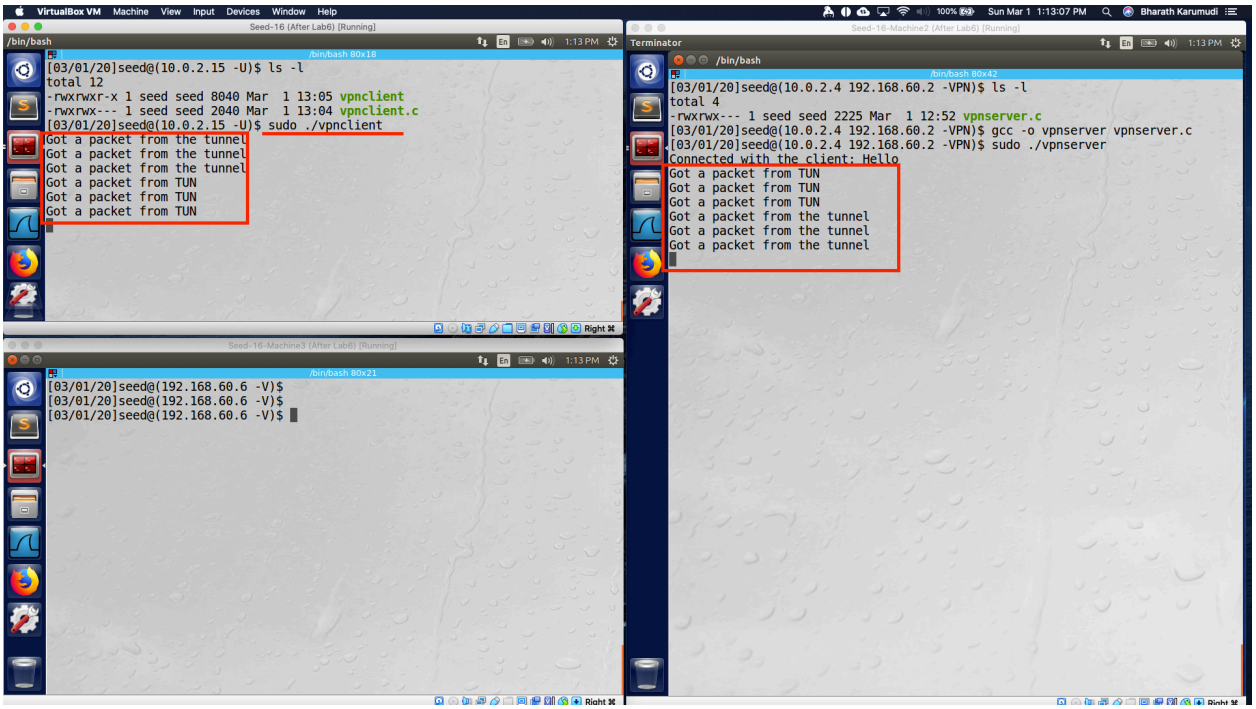

Fig4: Running the VPNClient program

Observation: Executed the VPNClient.c and created a tun0 interface on Host U but down and no IP allocated. Made the interface up and allocated the IP to interface as 192.168.53.5 and the client connected to VPN server.

Explanation: When executed the VPNClient.c on the client (Host U) first the TUN interface was created as tun0 but no IP allocated and also down (Fig 3). So, using the ifconfig allocated the IP as 192.168.53.5 and made the interface up *sudo ifconfig tun0 192.168.53.5/24 up*
and the client connected to VPN server (Fig 4).

Step 3: Set Up Routing on Client and Server VMs

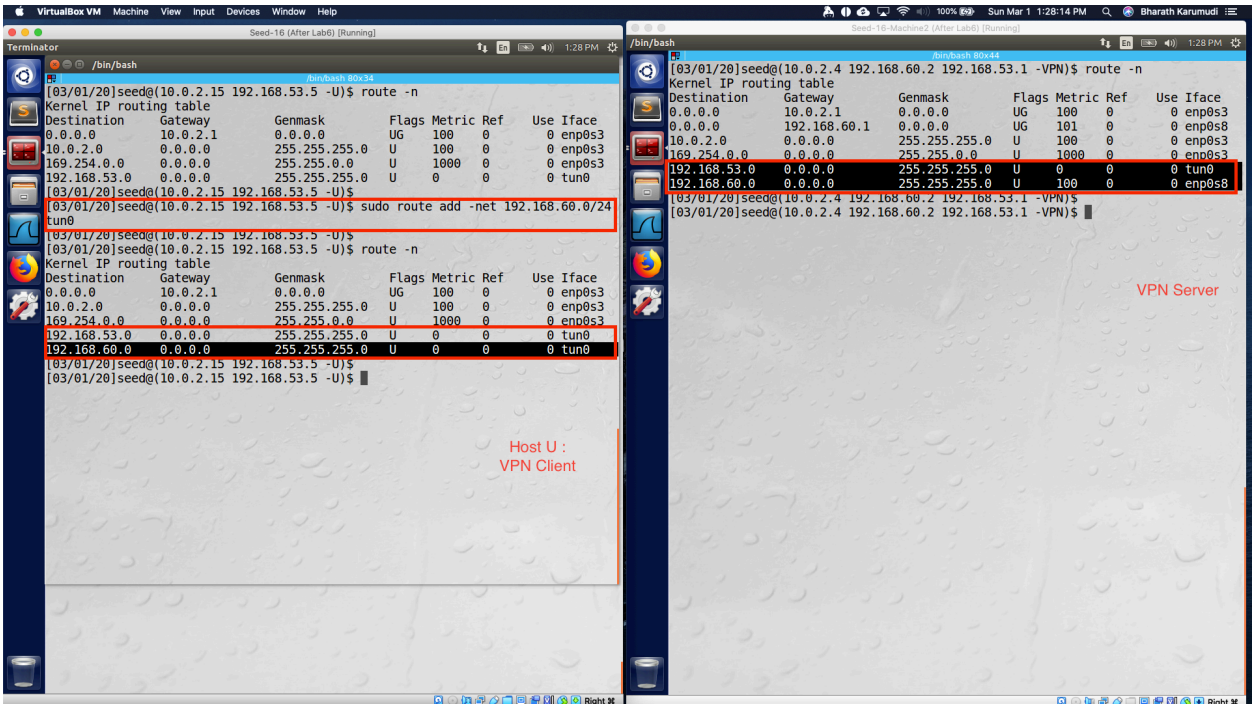

Fig5: Routing setup on VPN Client (Left) and VPN Server (Right)

Observation: The tun0 route was added by default on client for 192.168.53.0/24 net but to communicate with the private network (192.168.60.0/24) the route was not there, so added the route. On the VPN server the entry was added automatically on enp0s3 interface.

Explanation:
On Host U added the route 192.168.60.0/24 on tun0 using:
sudo route add -net 192.168.60.0/24 tun0 – this added the entry to the route table and any packet that has a destination of this network will be routed to tun0 network interface, which is the tunnel between Host U and VPN Server. Once the VPN server receives it, it will forward the packet on enp0S3 interface where the private network is.

Step 4: Set Up Routing on Host V

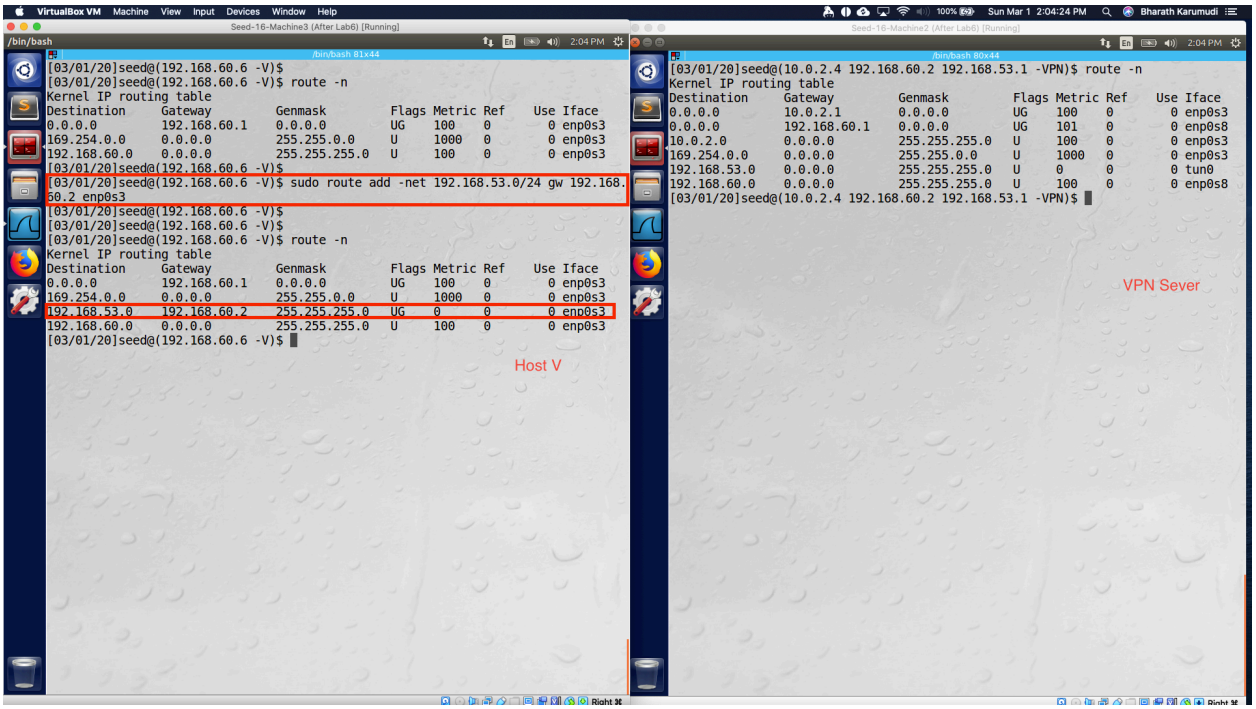

Fig6: Routing setup on Private network host (Host V)

Observation: There were no routes on Host V to respond to Host U, so added a route on Host V.

Explanation: When the packet arrives from Host U to Host V via VPN server, the Host V should respond in the same private channel. To achieve that we need a route to be added on the Host V and with: *sudo route add -net 192.168.53.0/24 gw 192.169.50.2 enp0s3* a new route was added (Fig 6). With the above command, any reply packet that has a destination IP as 192.168.53.0/24 will be sent to enp0s3 interface and uses the VPN server 192.169.50.2 as gateway. So, the packet from Host V will go to Host U via VPN server.

Step 5: Test the VPN Tunnel:

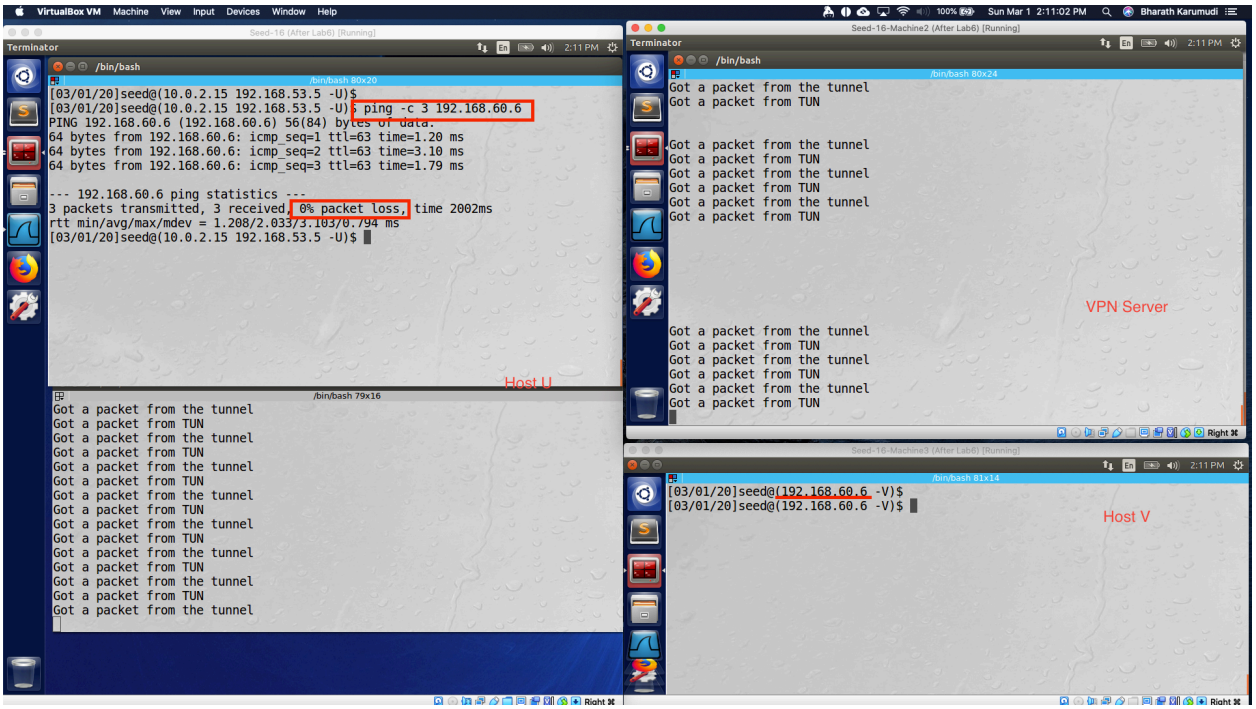

Fig7: Ping test from Host U to Host V

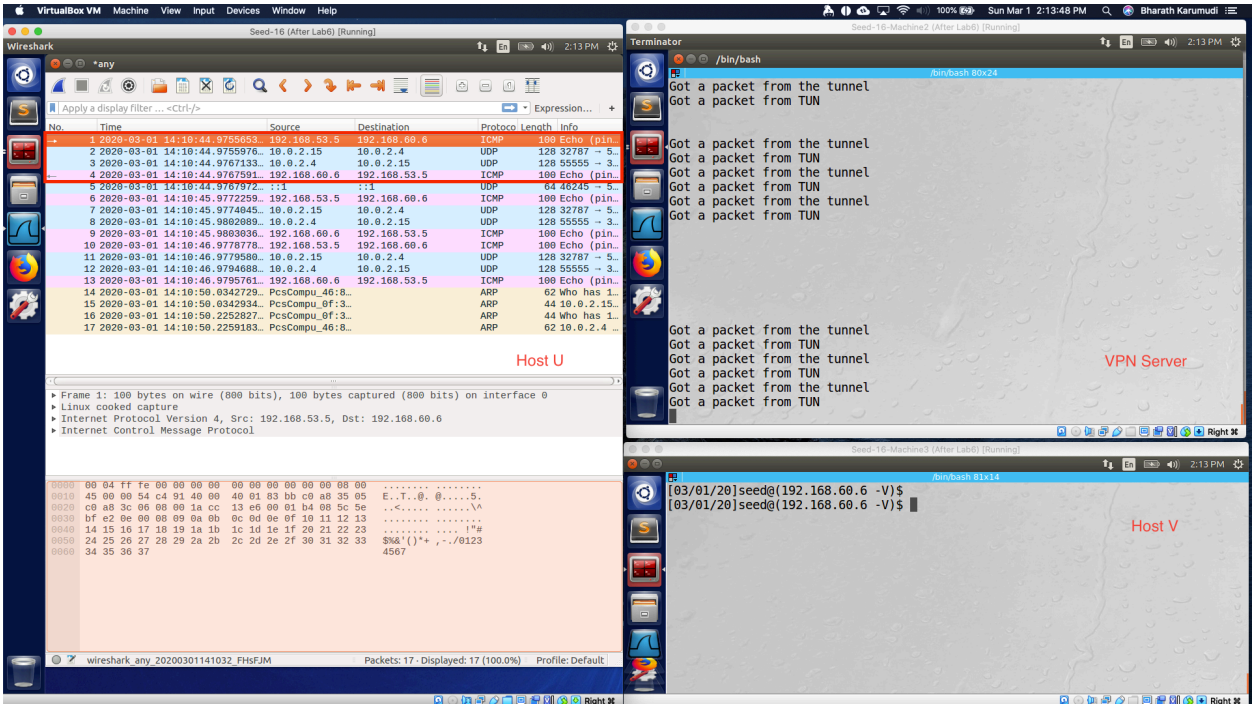

Fig8: Wireshark observation on Host U for ICMP traffic

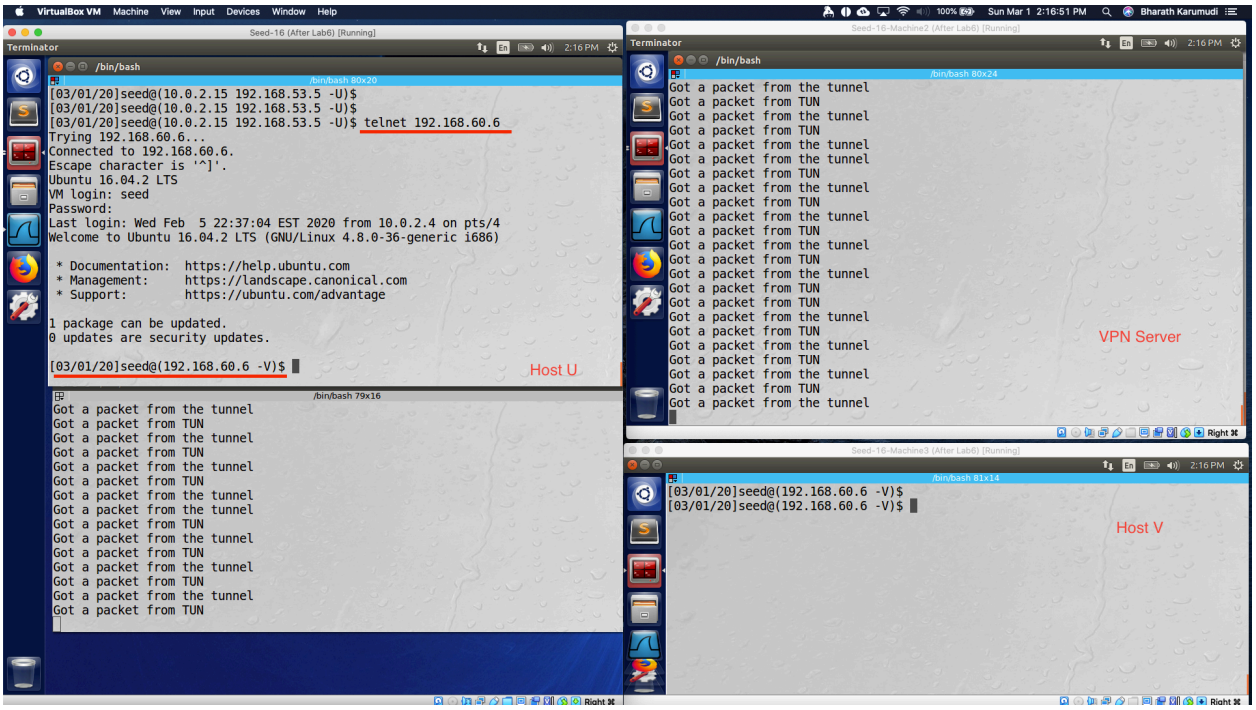

Fig9: Telnet test from Host U to Host V

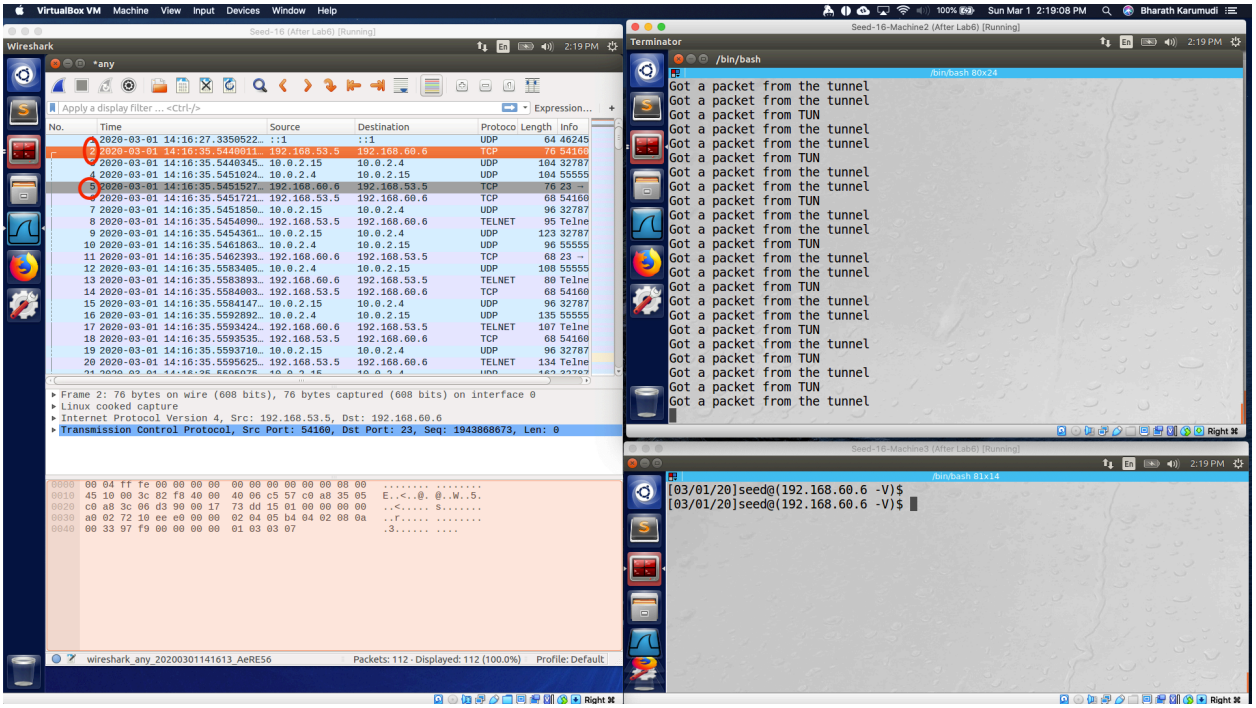

Fig10: Wireshark observation for telnet connection

Observation: Once all the network setup was done, from Host U sent a ping to Host V and received the ping response. Also connected to Host V over telnet from Host U.

Explanation:

1. Sent a ping from Host U to Host V and received all the packets (Fig 7).
2. Observed the traffic on Wireshark (Fig 8) and can see the ICMP traffic went through tunnel and the UDP went on regular network (NAT in this case) – these are physical connection between VPN Client and VPN server.
3. Connected from Host U to Host V over the telnet (Fig 9).
4. The network traffic was observed over the Wireshark (Fig 9), all the telnet and TCP went through tunnel and the physical connection went from non-tunnel.

Step 5: Tunnel-Breaking Test.

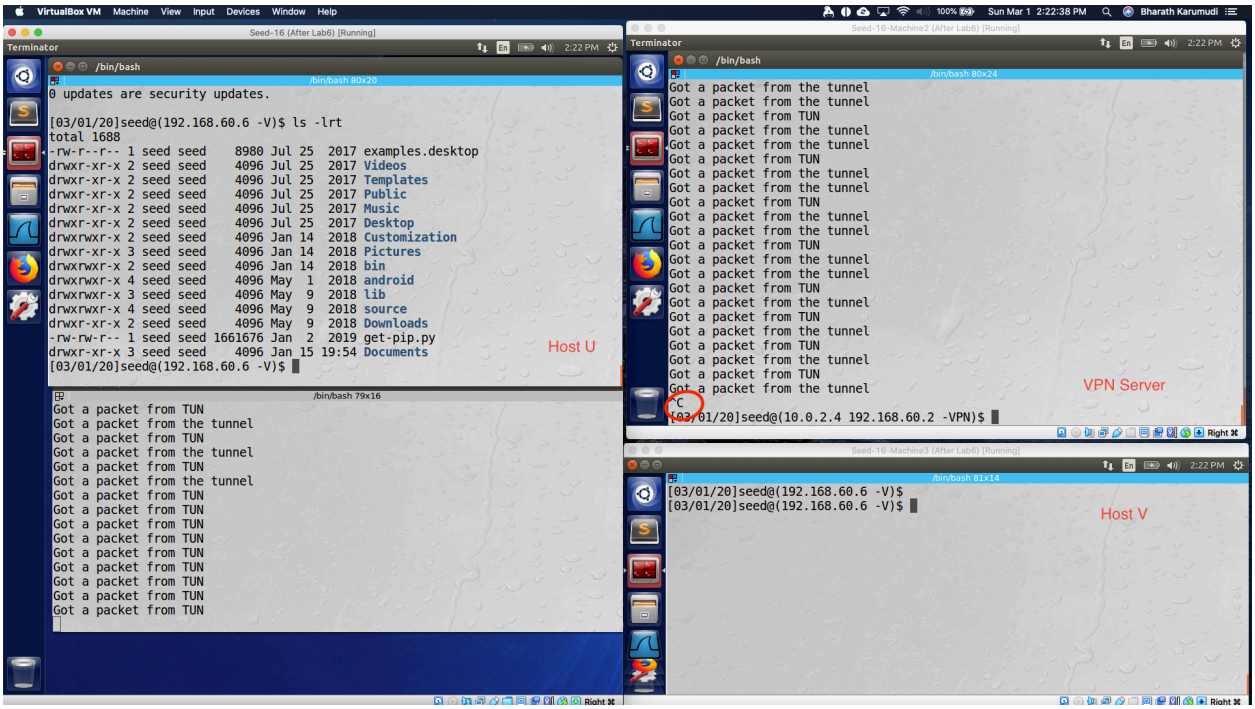

Fig11: Terminated the tunnel

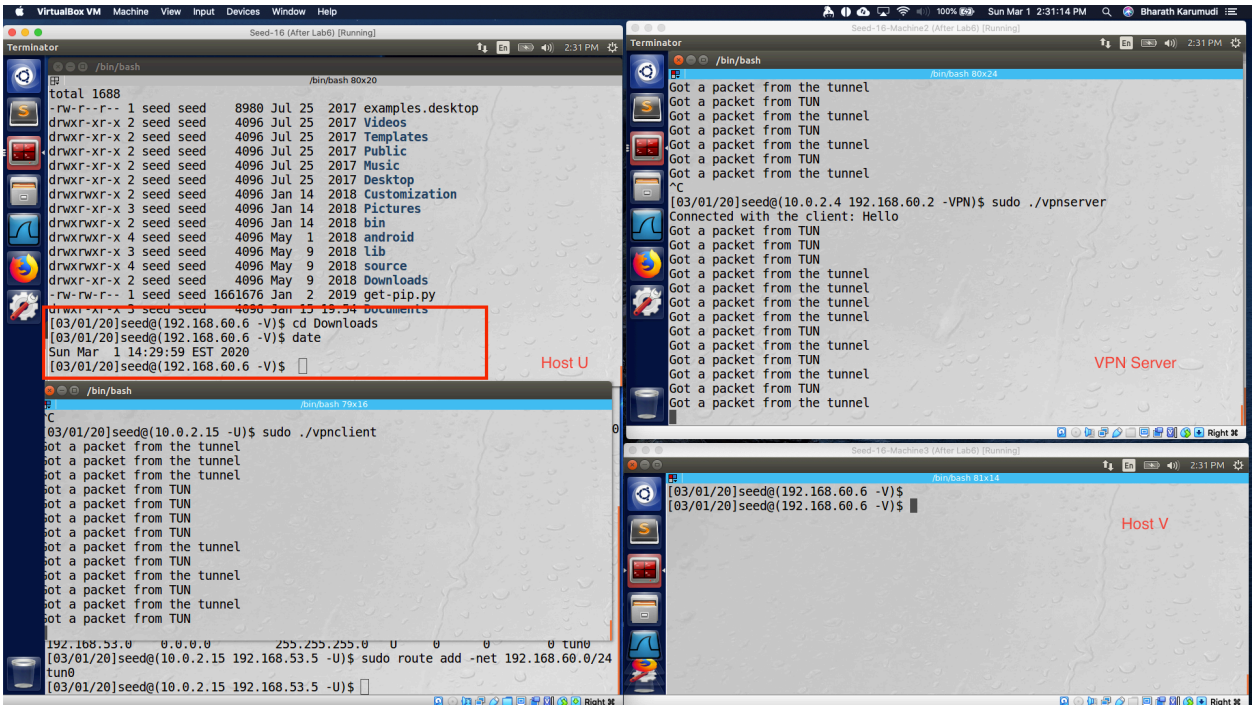

Fig12: Re-established the tunnel

Observation: Established a telnet connection from Host U to Host V and by terminating the tunnel, the telnet connection was just not responding anything we type but it was not broken. After re-establishing the tunnel between VPN Client and VPN server all the packets that are in buffer were sent out and got the response.

Explanation:

Established a telnet from Host U to Host V by having a VPN connection (tunnel) between Host U and VPN server (Fig 11). Interrupted the tunnel but the **telnet** window on Host U is still **active** and issued some commands (cd Downloads and date) but nothing was visible on the terminal. Once re-established the tunnel between Host U and VPN server (have to add the routing entries again on client and bring the tunnel up on both client VPN server) the commands that were typed earlier were sent to Host V and got the response – meaning the typed commands are in the buffer and once the connection established those packets are sent out through the tunnel.