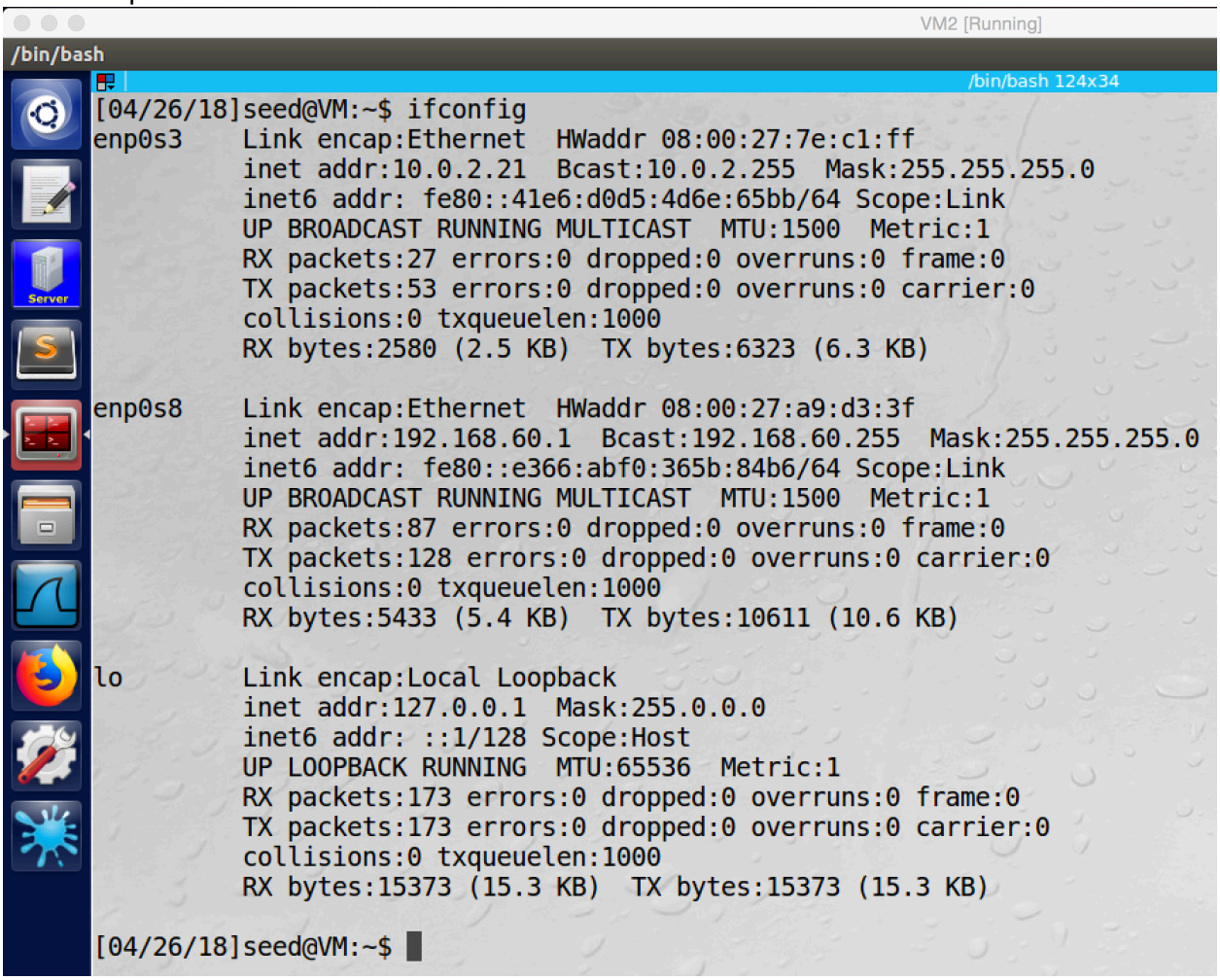
CSE644 FINAL PROJECT (VPN)
Yishi Lu
5/2/2018

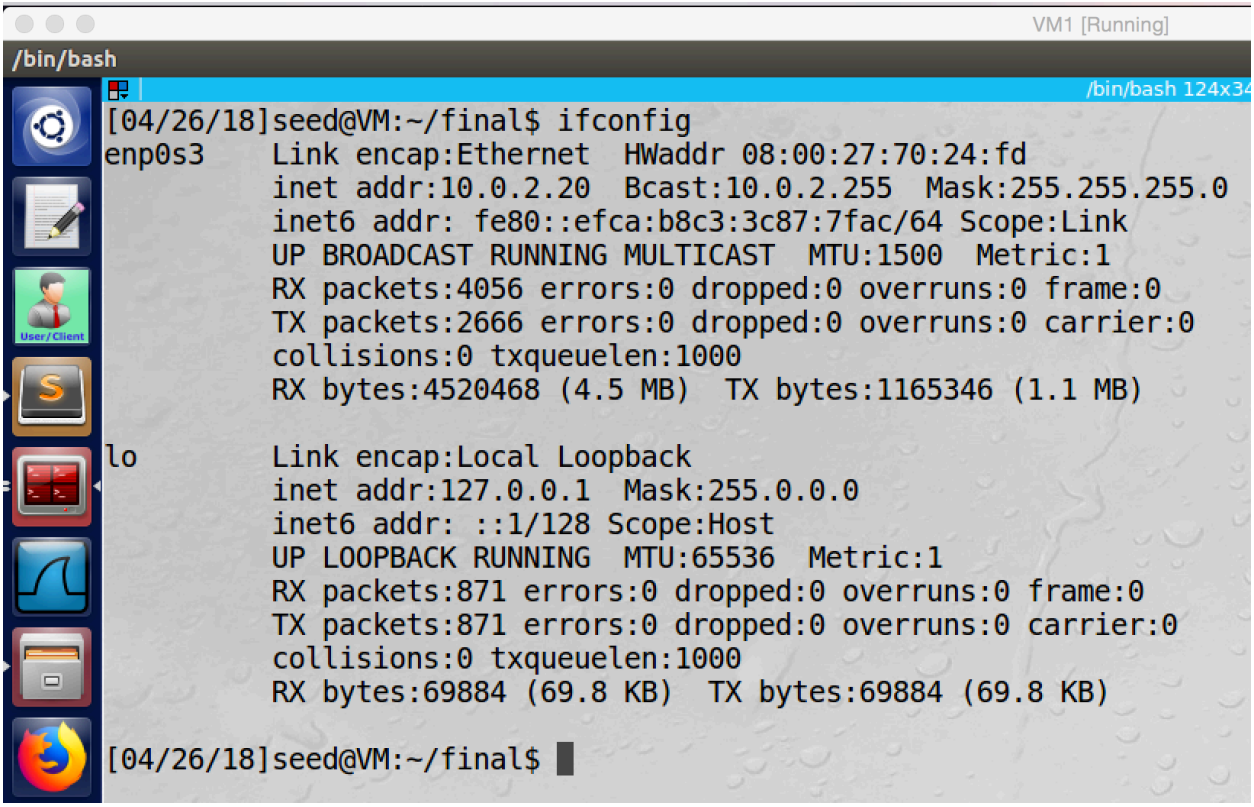
In this project, I use three VMs. They are
VM1, Nat Network IP address 10.0.2.20**, it is Host U**
VM2, Nat Network IP address 10.0.2.21, Internal Network IP address 192.168.60.1, **it is VPN server**
VM3, Internal Network IP address 192.168.60.101, **it is Host V**

**Task1: VM setup**

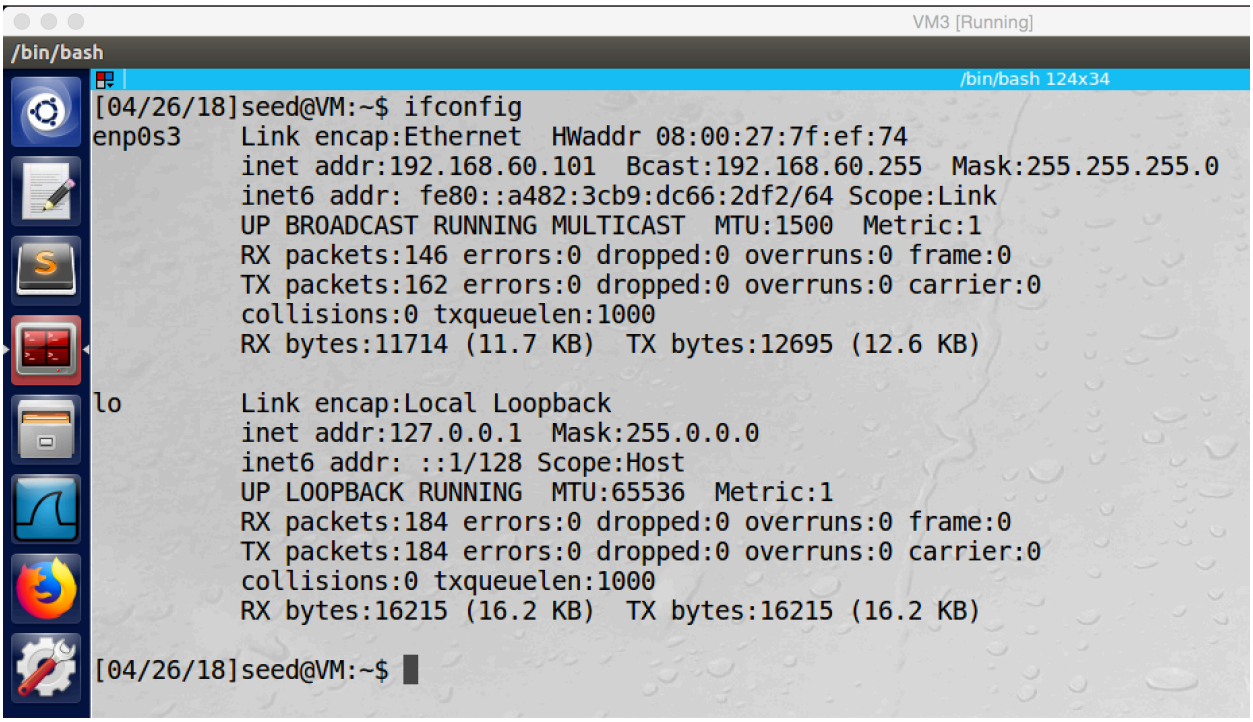
After setup all VMs



VM2, the VPN server



VM1, Host U



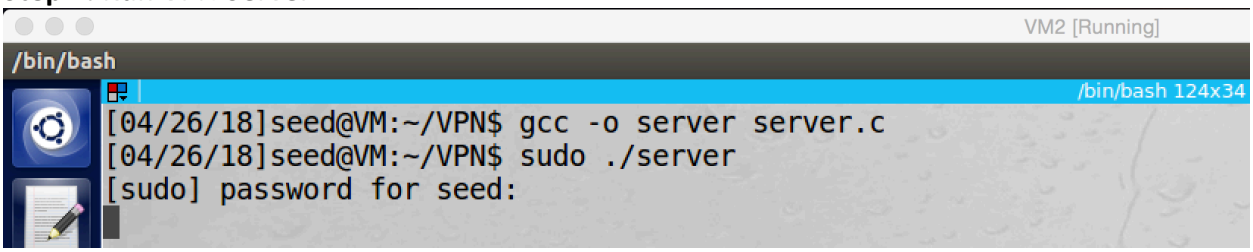
VM3, Host V

**Observation and Explanation:**

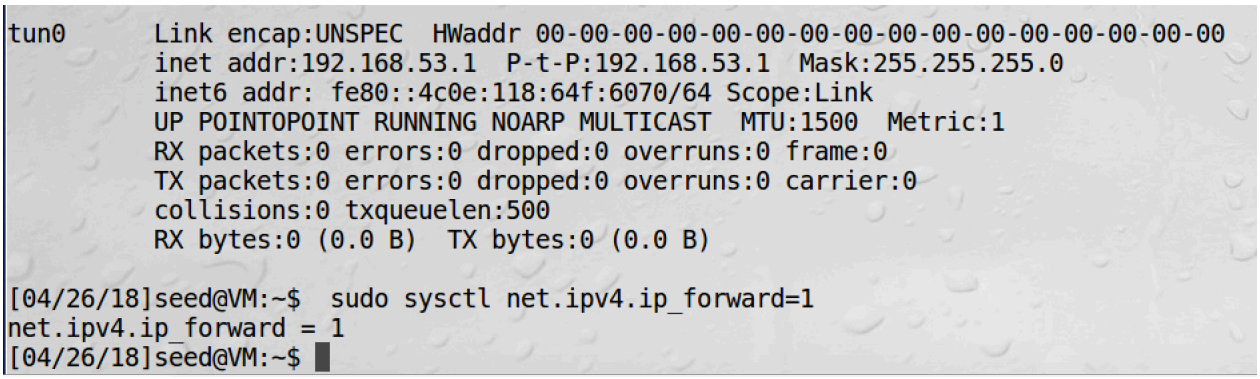
I followed lab description to setup all VMs. For VPN server (VM2), it has internal network with IP address 192.168.60.1 and NatNetwork with IP address 10.0.2.21 (screesnhot1). For Host U (VM1), it has NatNetwork with IP address 10.0.2.20 (screesnhot2). For Host V (VM3), it has internal network 192.168.60.101 (screenshot3).

**Task2: Creating a VPN Tunnel using TUN/TAP**

**Step 1: Run VPN Server**

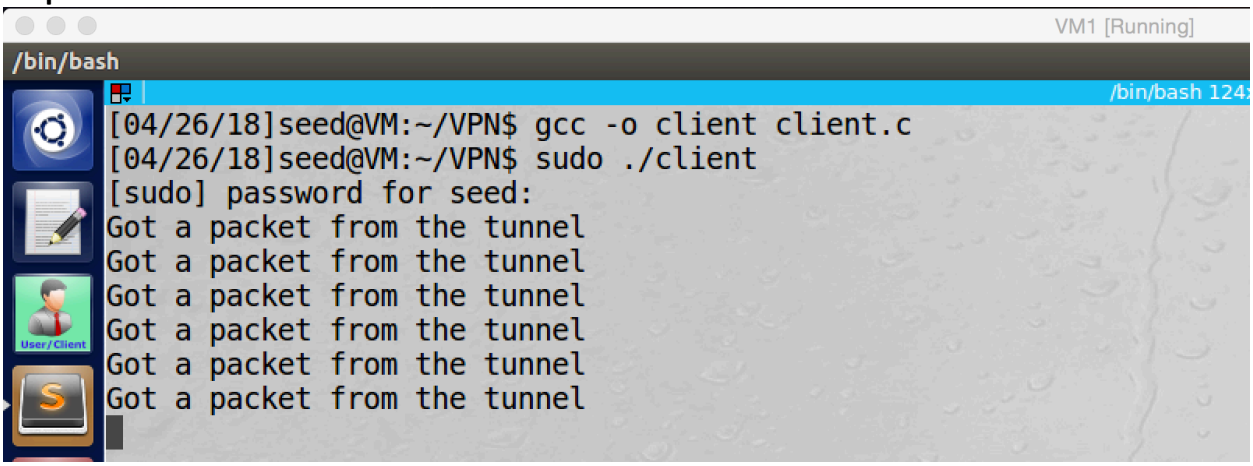



screenshot1. Running VPN server program on VM2

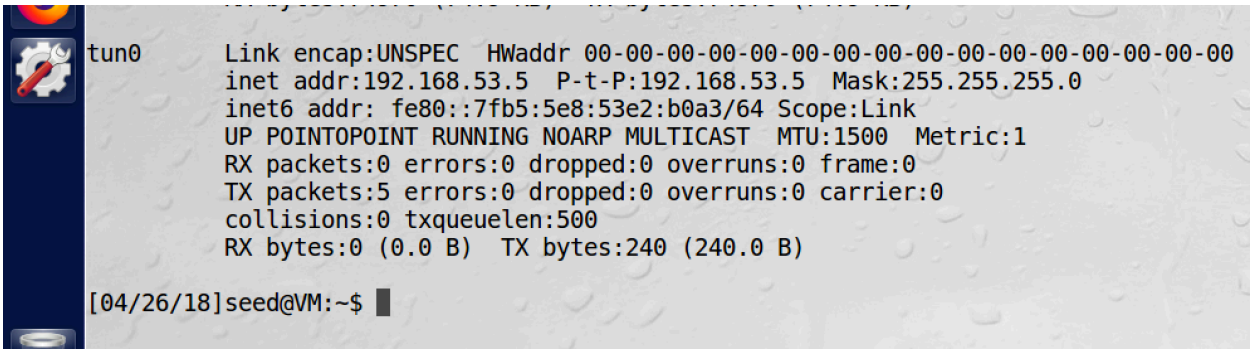



screenshot2. Assigning IP address to tun0, and enabling IP forwarding

**Step 2: Run VPN client**

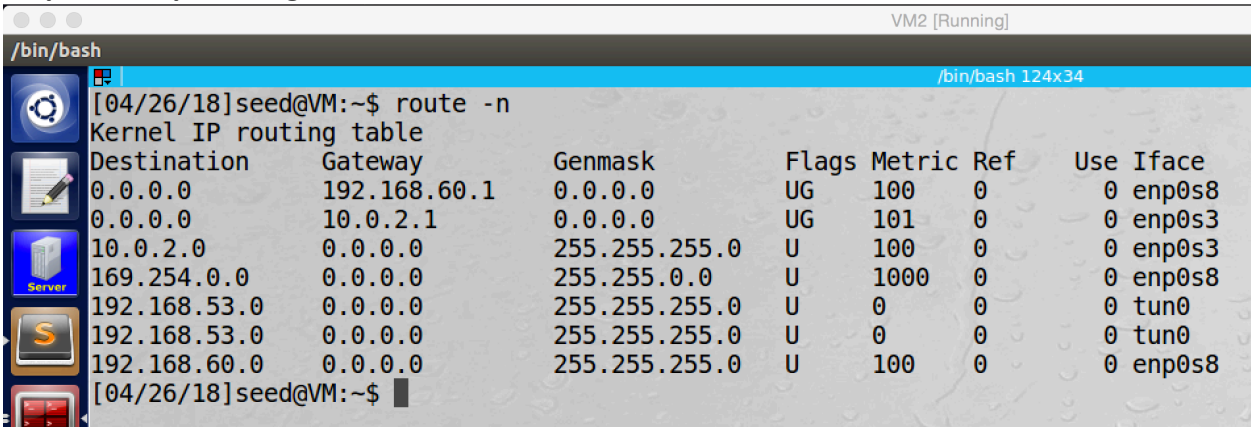



screenshot3. Running VPN client program on VM1

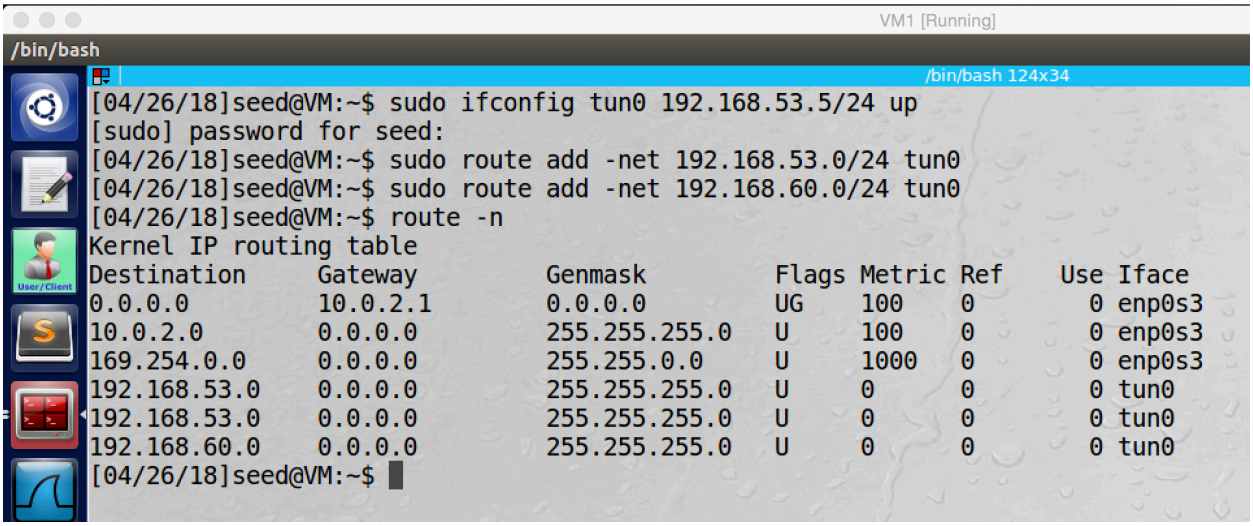



screenshot4. Assign IP address to tun0

**Step 3: Set Up Routing on Client and Server VMs**

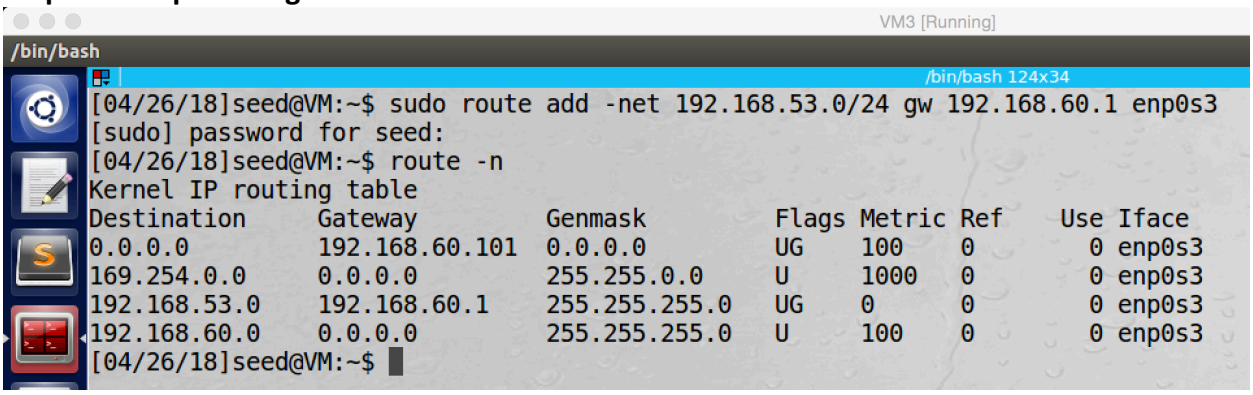



screenshot5. Adding routing rule on VPN server, which directing any packet with IP address 192.168.53.0/24 to tun0



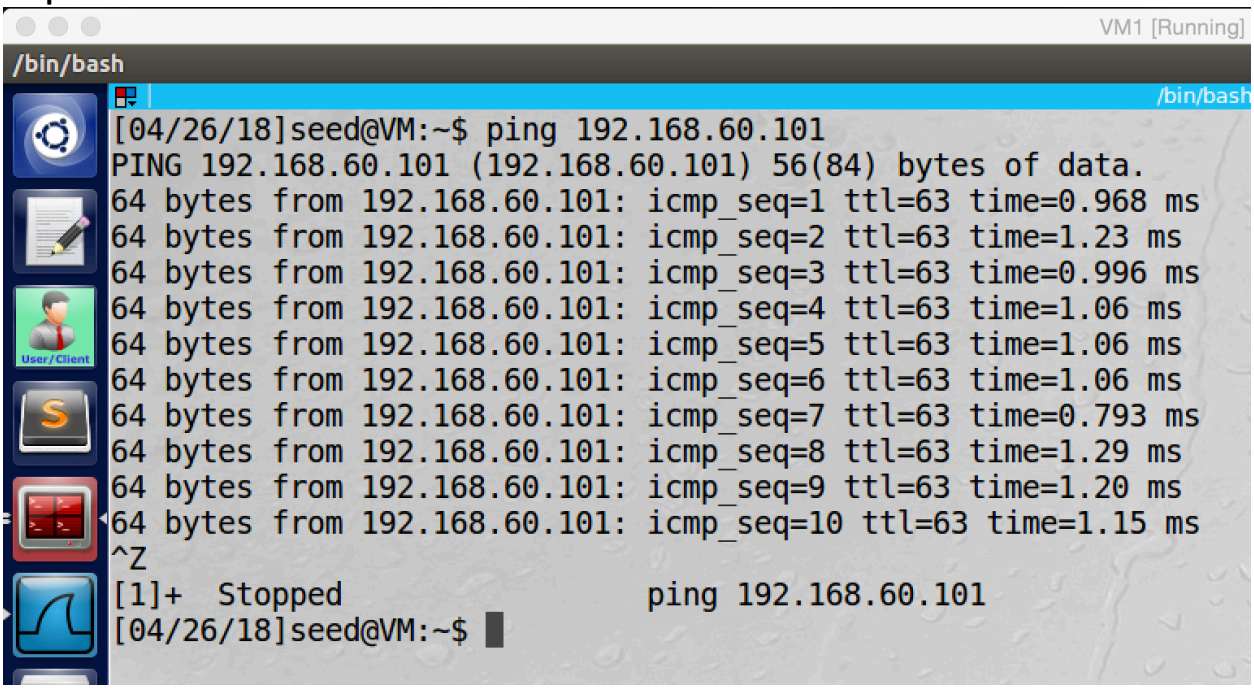

screenshot6. Adding routing rules on VPN client, which directing any packet with IP address 192.168.53.0/24 and 192.168.60.0/24 to tun0

**Step 4: Set Up Routing on Host V**

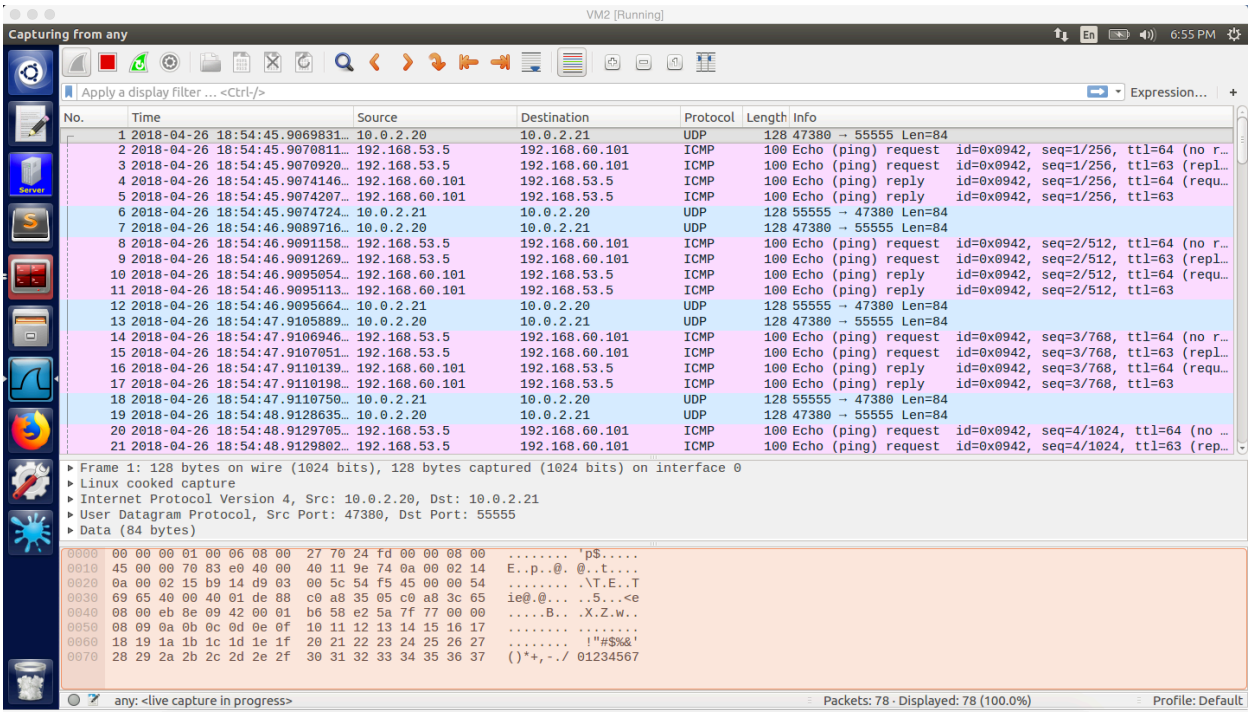



screenshot7. Adding routing rule on Host V, which directing any packet go to IP address 192.168.53.0/24 to the VPN server (192.168.60.1)

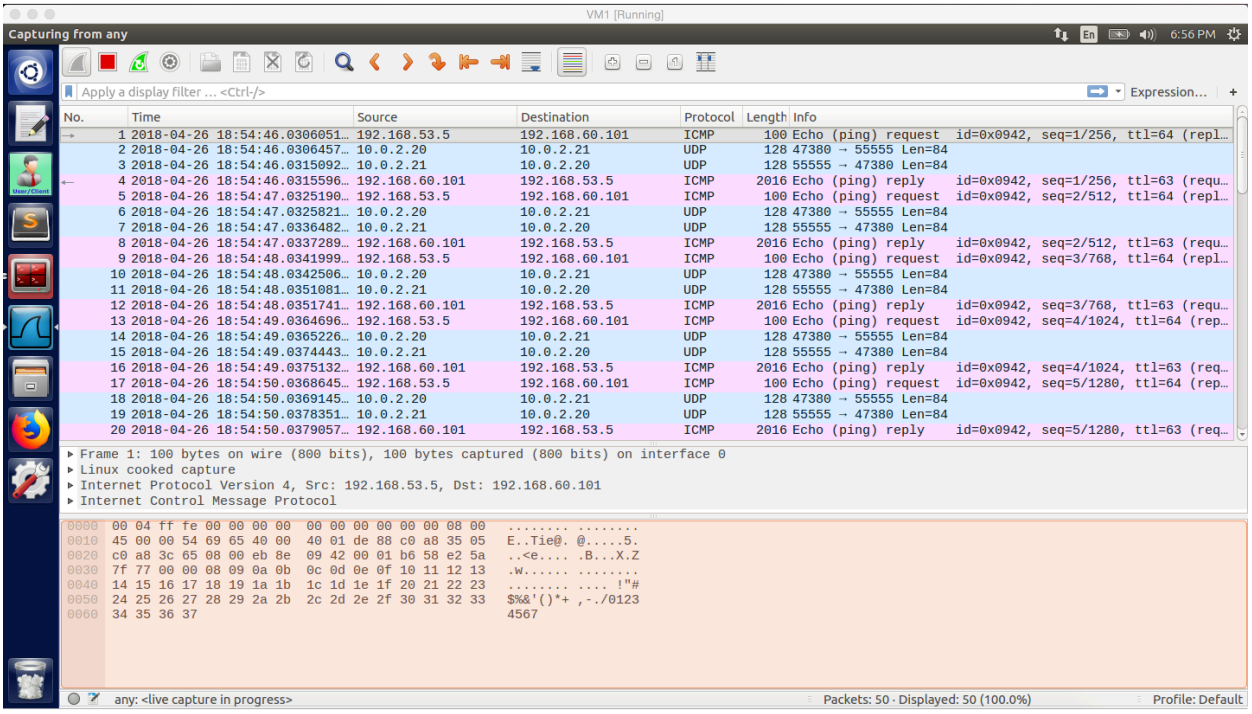
**Step 5: Test the VPN tunnel**



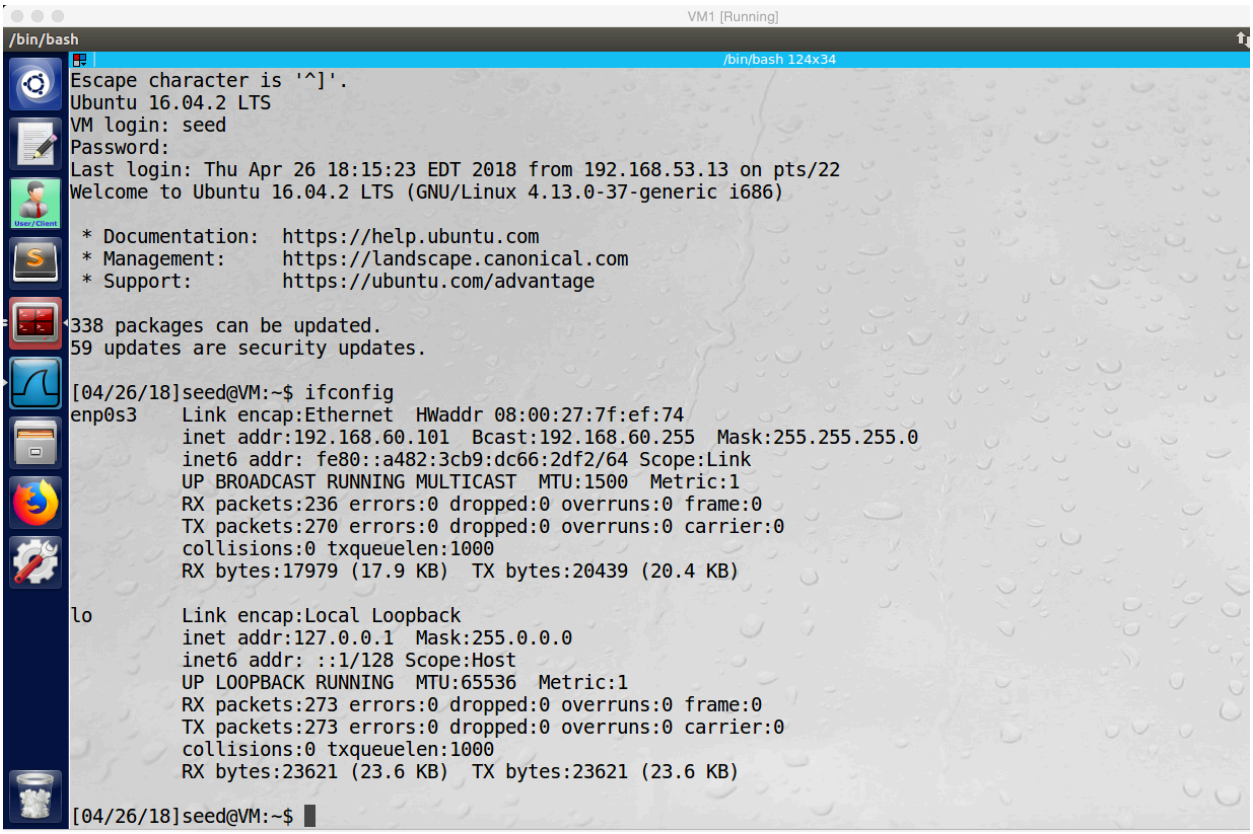

screenshot8. Successfully ping Host U from Host V.



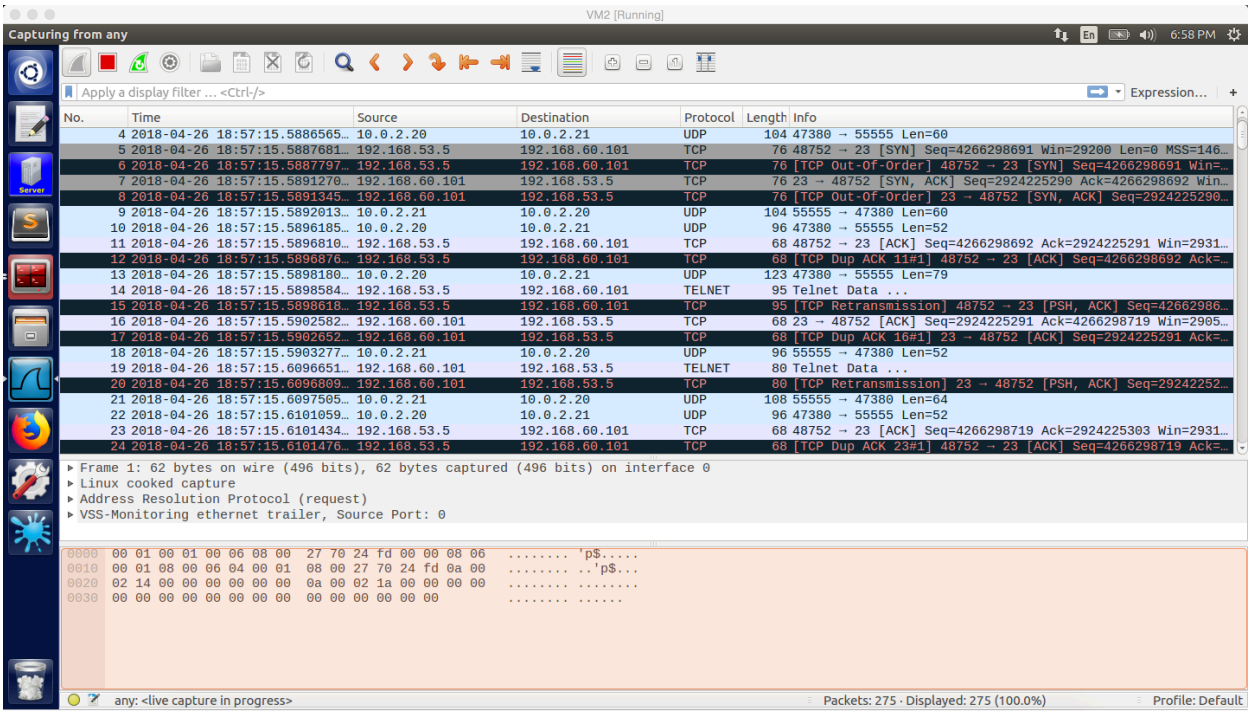

screenshot9. Wireshark captures the whole traffic. (From VPN server)



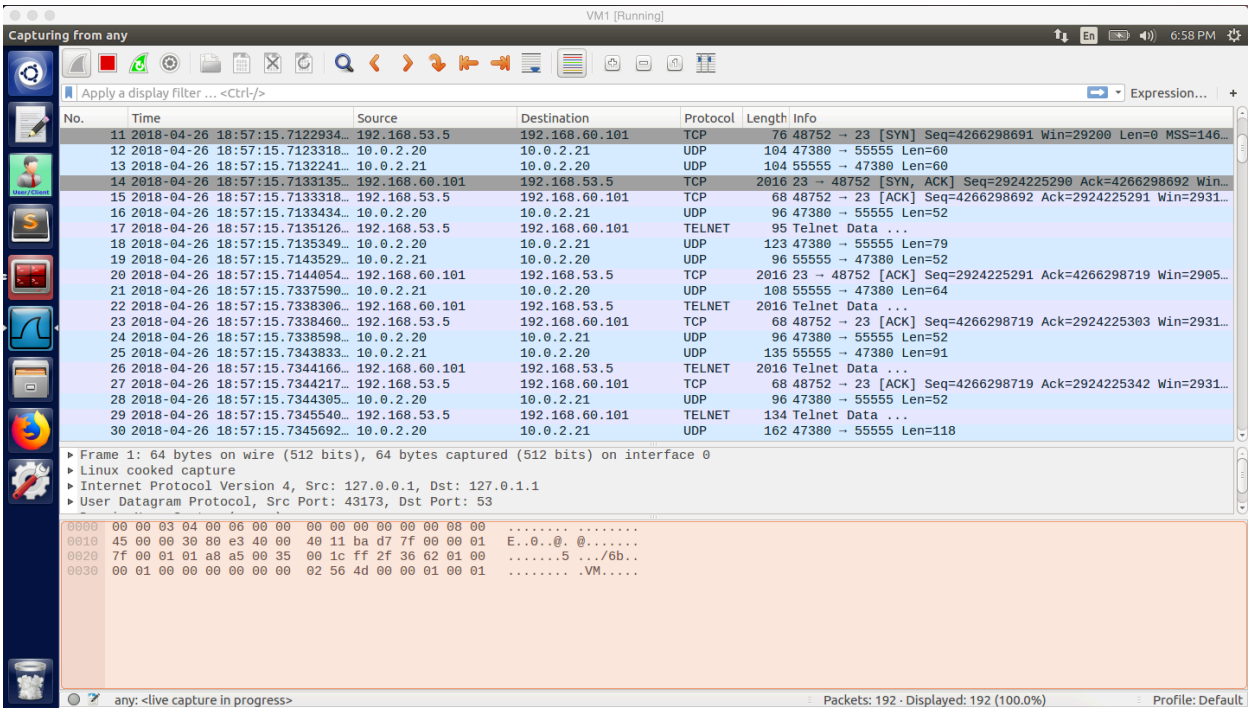

screenshot10. Wireshark captures the whole traffic. (From VPN client)




screenshot11. Successfully telnet from Host U to Host V




screenshot12. Wireshark captures the whole traffic. (From VPN server)



screesnhot13. Wireshark captures the whole traffic. (From VPN client)

**Observation and Explanation for step1 to step5:**

After I setup all VMs, I run VPN programs on them. On step 1, I run VPN server program on VM2. And I assigned IP address 192.168.53.1 to its TUN interface. Otherwise, because VPN server need to forward packets between the VPN tunnel and the private network, so we need to enable the IP forwarding (screenshot 1 and 2).

On step 2, I run VPN client program on VM1. And I assigned IP address 192.168.53.5 to it (screenshot 3 and 4).

On step 3, I set routing table on VPN server and client. For VPN server, we need to direct packet with IP address 192.168.53.0/24 to tun0, we add such rule on routing table (screenshot5). On the VPN client machine, excepting direct packets with IP address of 192.168.53.0/24 to tun0, we also need to direct packets with IP address 192.168.60.0/24 to tun0. Therefore, the client machine can send packet to the private network (screenshot6).

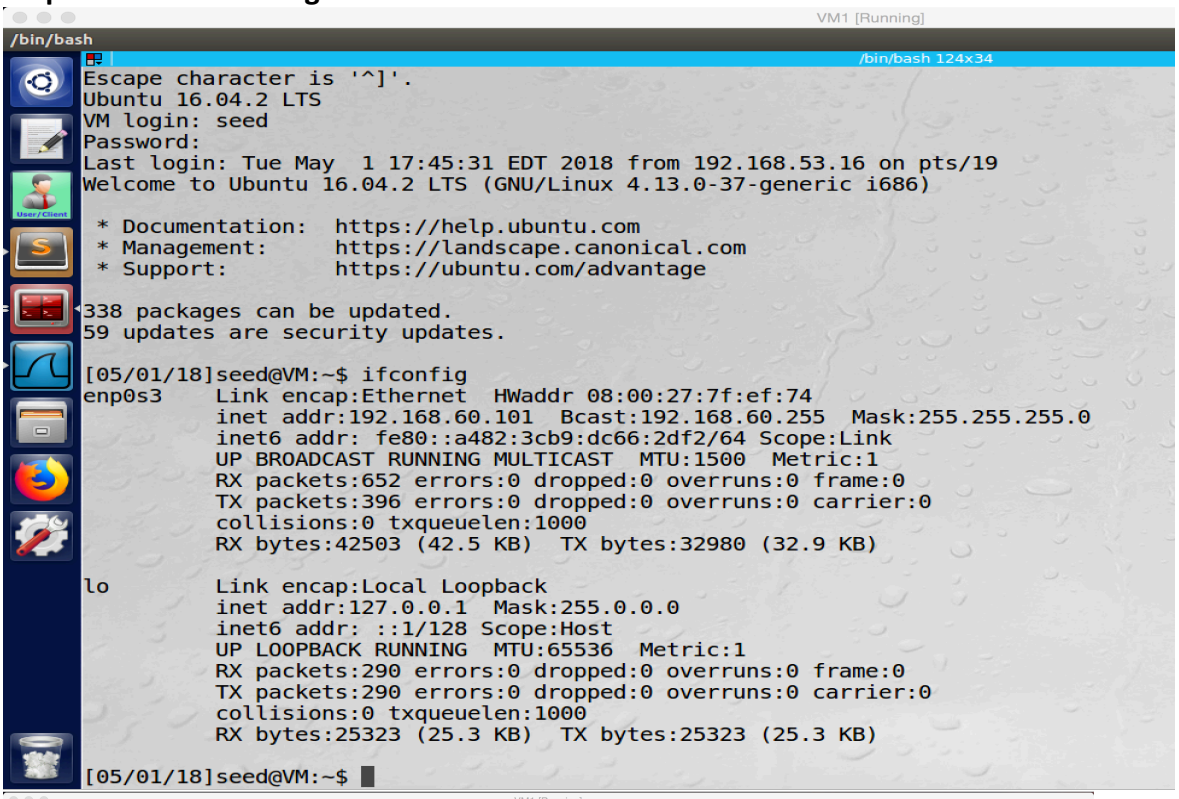
On step 4, we add rule to routing table of Host V. When Host V receives the packet from Host U, it need send packet back as well. However, the packet from Host U has source IP address of 192.168.53.5, so Host V does not know where to send this packet (192.168.53.0/24 cannot reach by Host V). So we need to add rule to direct these packets to VPN server, then the VPN server can send the packet back to Host U by the tunnel. (screenshot7)

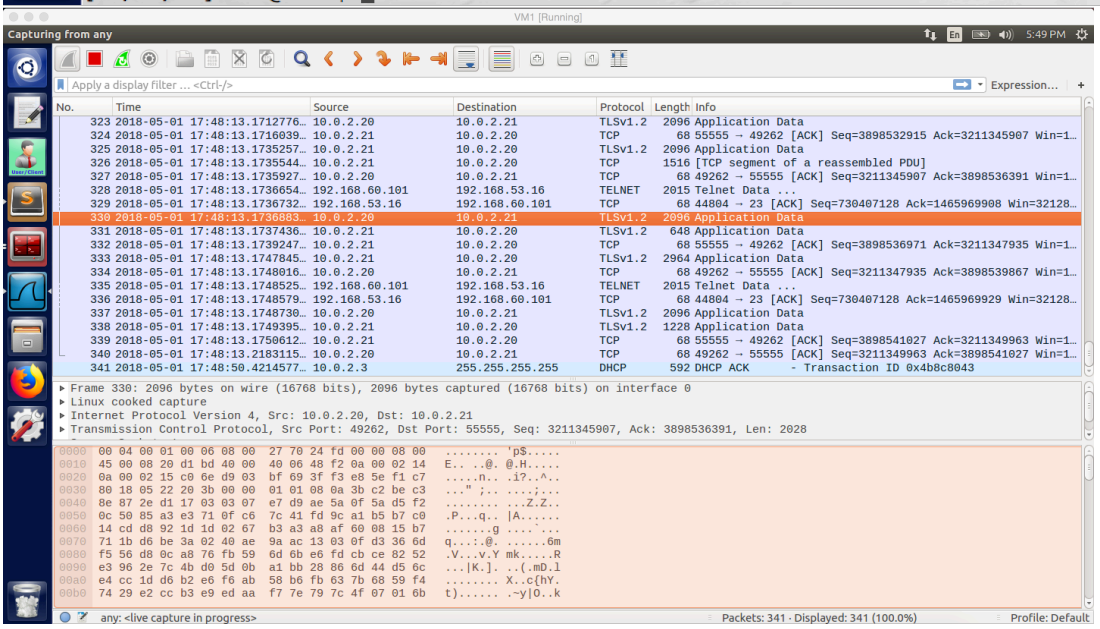
On step 5, we tested the tunnel. We first ping Host V from Host U, and we succeeded (screenshot 8 to 10). The ICMP packet is sent out from Host U, because the destination address is 192.168.60.101, the packet is routed to tun0 of Host U, and then it puts the ICMP packet in a UDP packet and send it out. After the UDP packet reaches the VPN server, the system will send it to the VPN program, and then the VPN program will take the ICMP packet out and forward its destination Host V. After Host V receives the ICMP request, it sends ICMP reply back. Because

the ICMP packet has source IP address 192.168.53.5, it will be routed to VPN server. Then VPN server will send it back to Host U in the same path with reverse order.

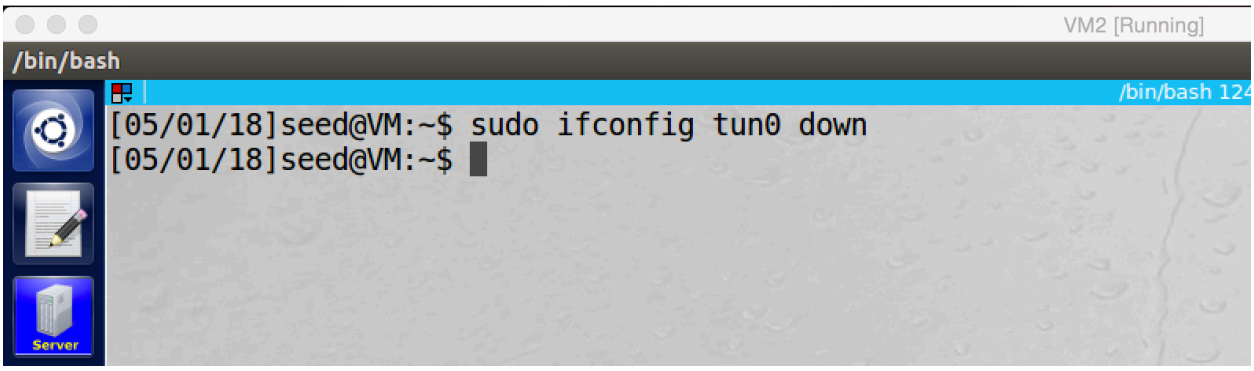
And then I also try telnet, and I also succeeds (screenshot 11 to 13). The packet traversal path and mechanism are same as the ICMP packet. But this time, the VPN program will put TCP packets in the UDP packets to send out (telnet uses TCP).

**Step 6: Tunnel-Breaking Test.**

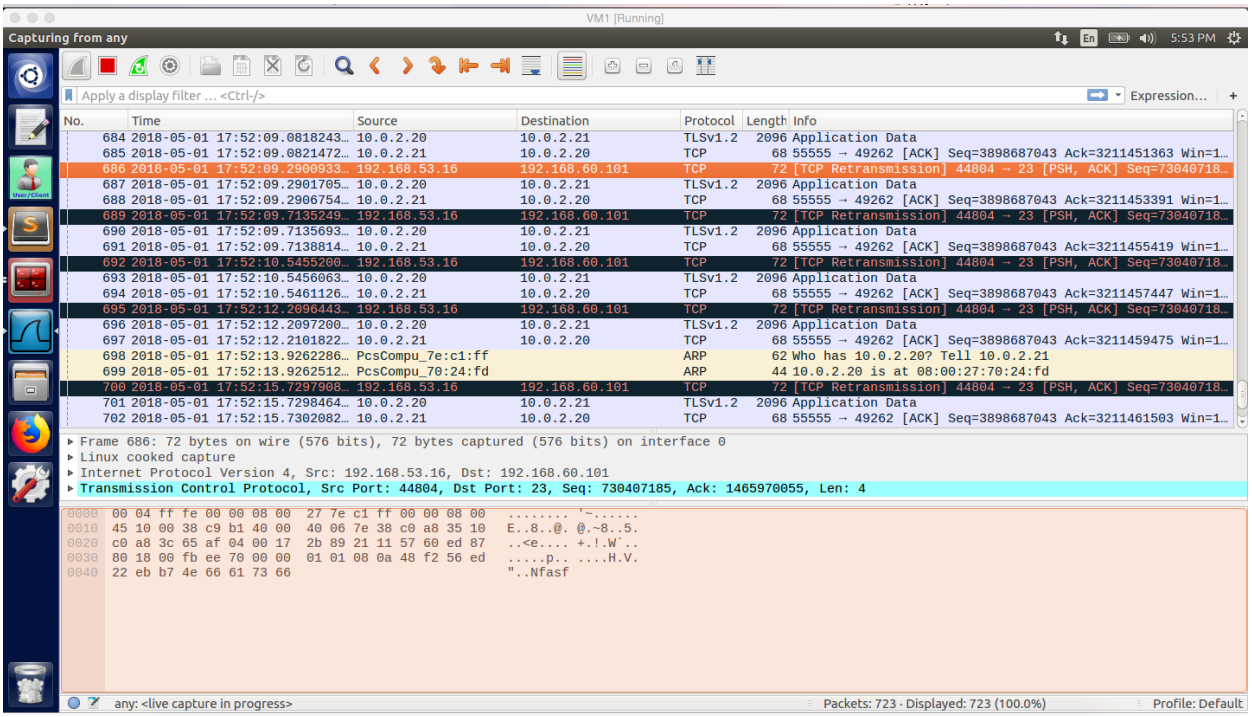


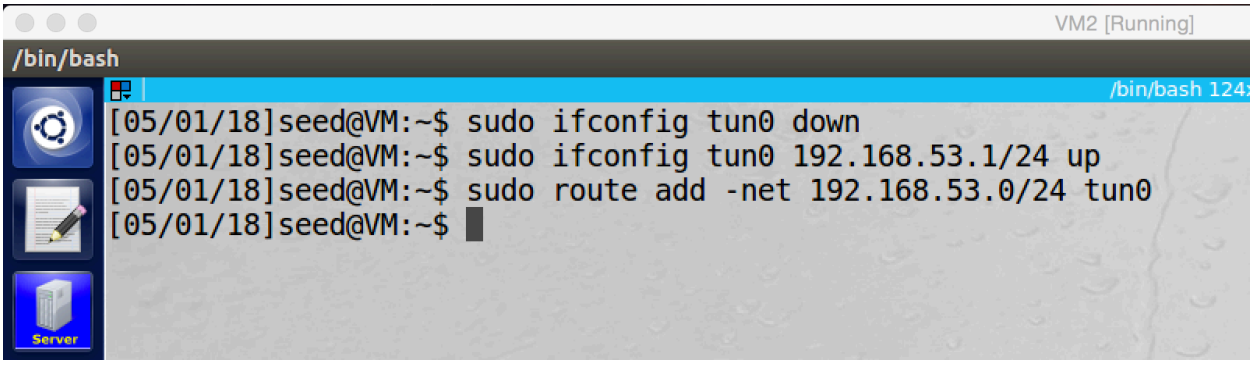

screenshot14. Telnet from Host U to Host V

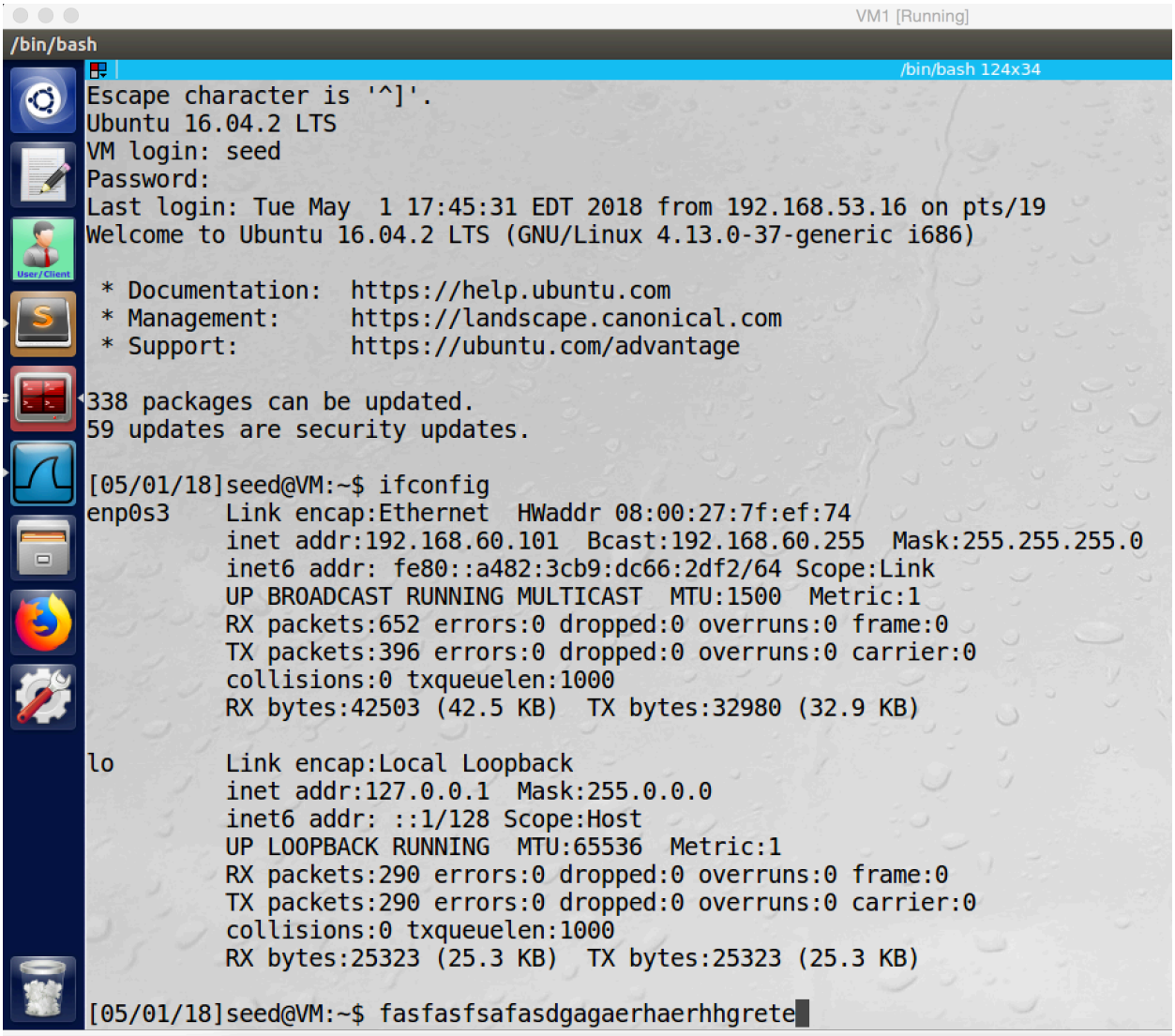



screenshot15. I broken the connection by delete tun0 on VPN server

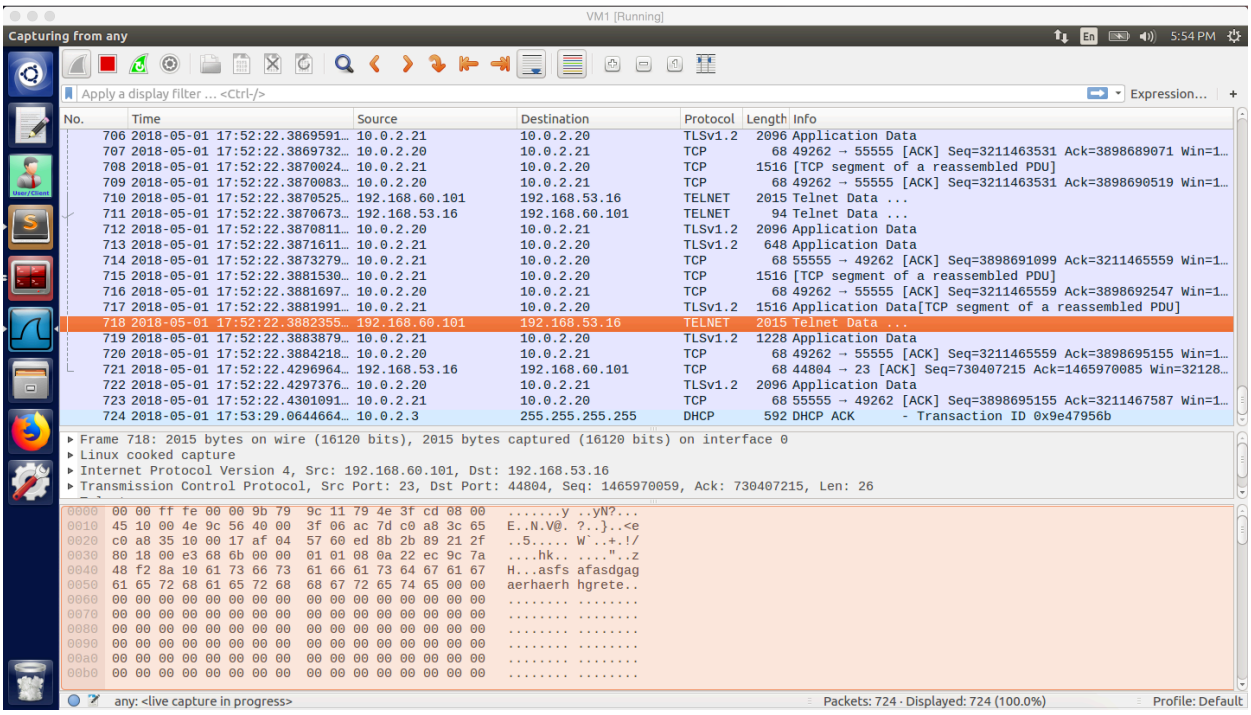



screenshot16. After I broke the connection, there is a lot of retransmission packets, and the telnet is frozen.






screenshot17. After I reconnect the VPN tunnel, the telnet connection is resumed. Anything which I typed before is displayed on the screen.



screenshot18. After I resume the tunnel, the traffic resumes

**Observation and Explanation for step 6:**

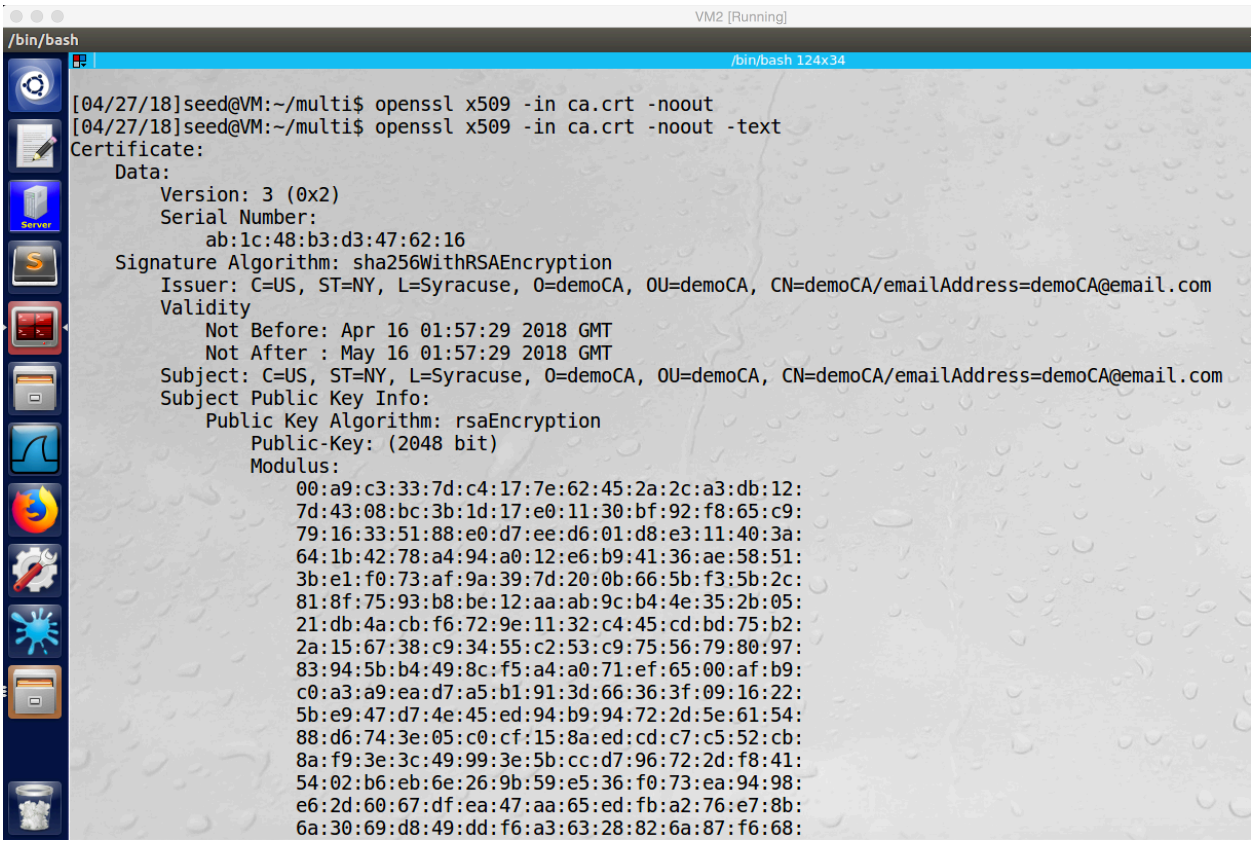
First I telnet from Host U to Host V by using the VPN tunnel (screesnshot14). After I broke the tunnel, the telnet is frozen (screenshot15). When I type, nothing is displayed on the screen. After I resumed the VPN tunnel, anything I typed before is display on the screen. And the telnet connection is resumed (screenshot17).

The reason is, when the telnet connection is established, it actually uses TCP protocol to transfer packets. So when the connection is broken, the data is still in the TCP buffer. And TCP keeps trying to resend these packets, so there is a lot of retransmission packets (screenshot16). After I restart the VPN server program, the tunnel is resumed, so TCP packets can be sent out now. Therefore, the whole traffic is resumed (screenshot18).

**Task 4: Authenticating the VPN server**

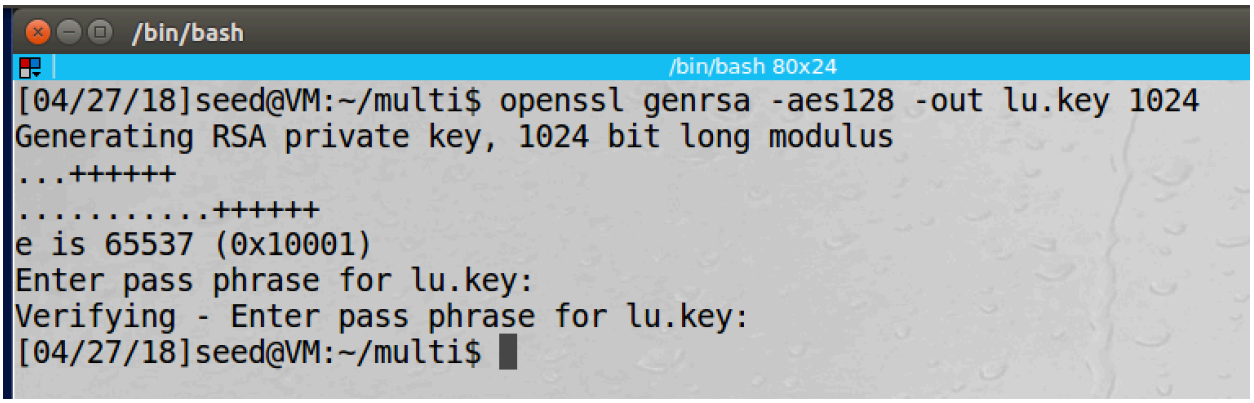
Before we can encrypt the VPN tunnel, we must do key exchange. For doing key exchange, the server must provide its public key. For security purpose, we must make sure that the public key of the server is valid, so the server needs to provide a valid certificate which is signed by trustable CA; and then the client must verify it as well. In fact, I finished task4 before finishing task3. So we look at task4 first.

Before we can assign certificate to the server, we must have a CA to sign the certificate; so we create demoCA first

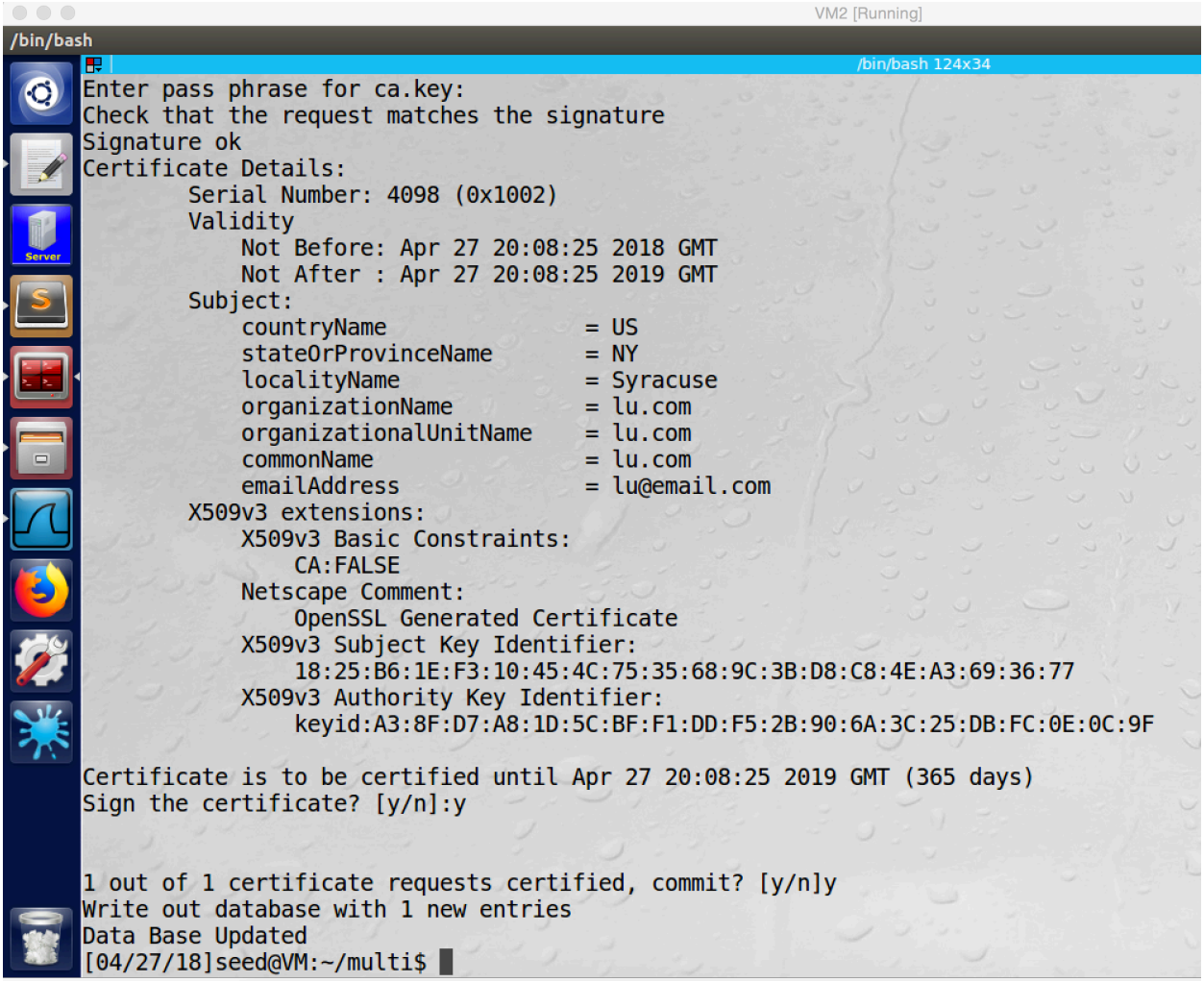


screenshot1. Certificate of demoCA

And then we need to create public/private key and certificate for the VPN server, which called lu.com

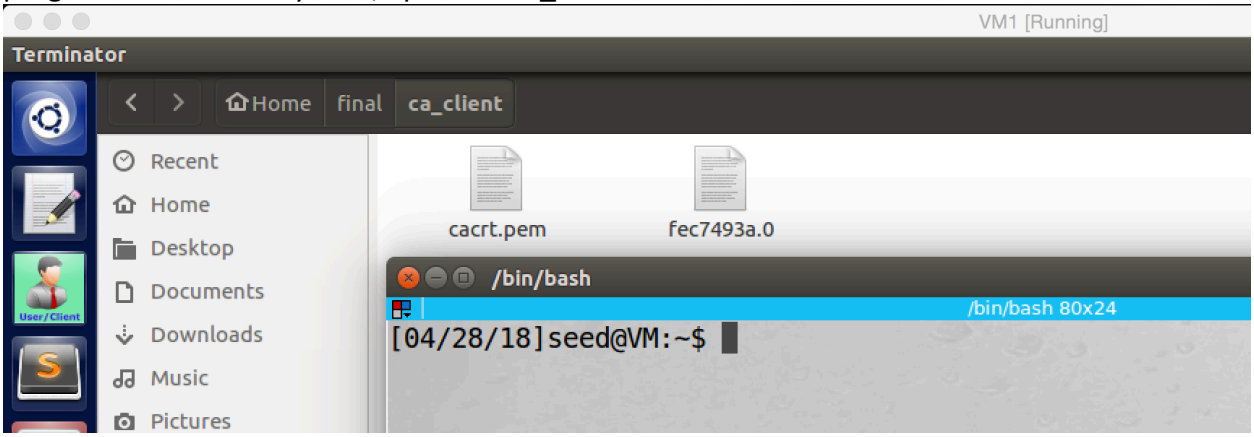


screenshot2. Creating private/public key for the VPN server



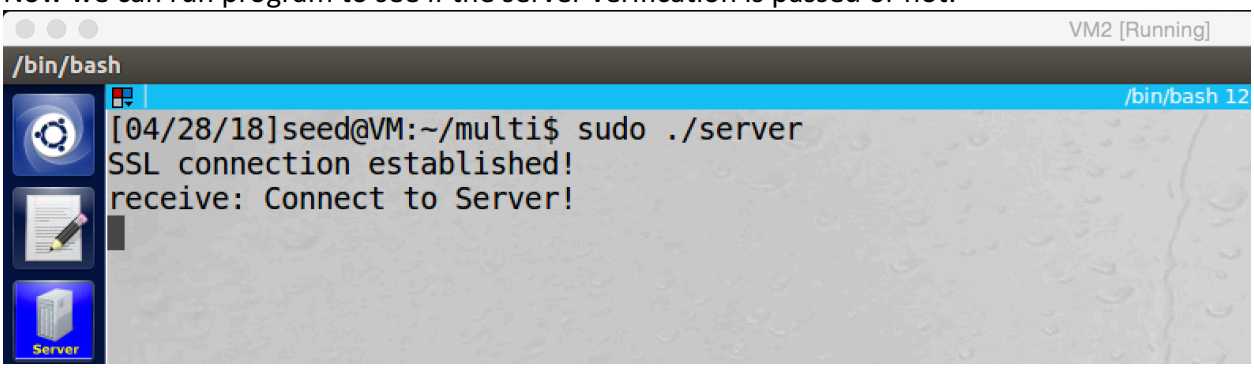
screenshot3. Creating certificate for VPN server lu.com

Now the server has a valid certificate, but the client also need to have the demoCA's certificate to verify the server's certificate. So we need to put the CA's certificate in the VPN client program's folder. In my case, I put it in ca_client

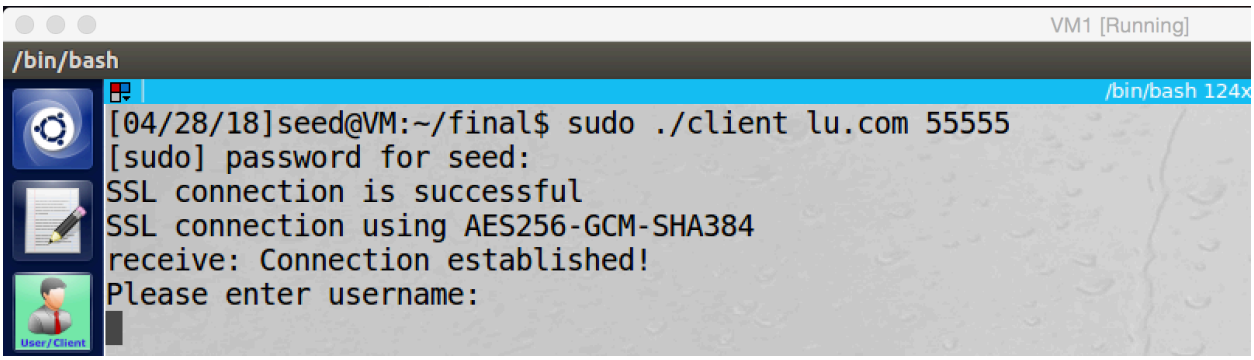


screenshot4. CA's certificates in client program, and I also create a symbolic link of the CA's certificate

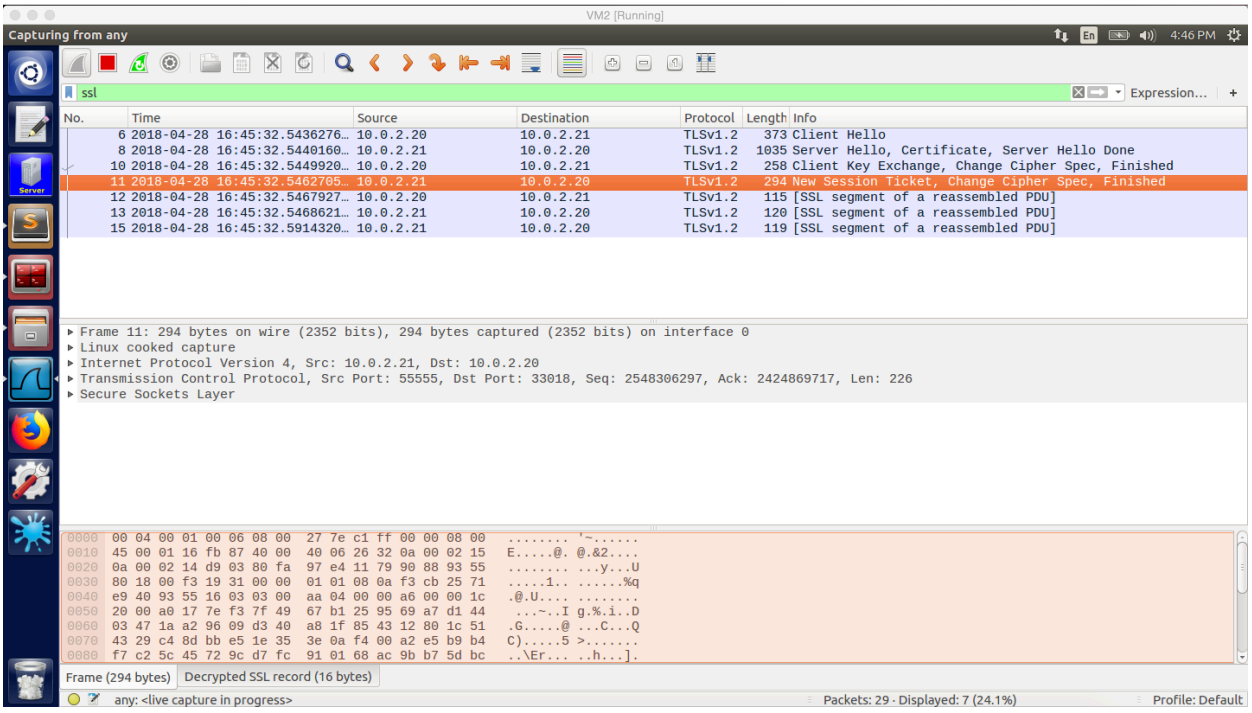
Now we can run program to see if the server verification is passed or not.




screenshot5. Server receives the connection from client




screenshot6. Client successfully connect to server




screenshot7. Wireshark captures the whole traffic, and the server's certificate verification is passed, and key exchange succeeds.

**Observation and Explanation:**

In this task, we perform the server certificate verification and key exchange. First, we create a demoCA, so we can have valid certificate for the VPN server (screesnhot1). And then we create public/private key for the VPN server (screenshot2). Afterwards, we use demoCA to assign a certificate to the VPN server, and the common name on the certificate is lu.com, this is important (screenshot3). Otherwise, we also put the CA's certificate to the client program and create a symbolic link for the certificate, so the client program can use this certificate to verify the server's certificate (screenshot4). Finally, we run the server and client program, and they successfully finished TCP and TLS handshake, and they also finished key exchange (screenshot 5 to 7).

For the server verification, there are three things which the client program must check. First, the server's certificate is valid. Second, the server is the owner of the certificate. Third, the server is the user intended server. For first and second things, the client program can use the CA's certificate to verify them. And the code of verification is following

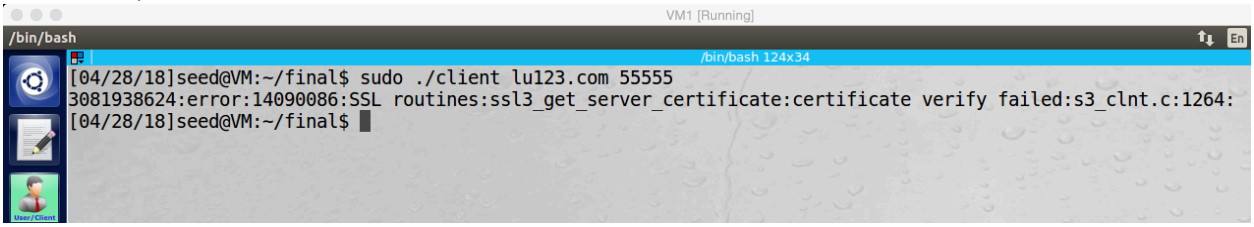
```
SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER, NULL);
SSL_CTX_load_verify_locations(ctx,NULL, CA_DIR)
```

The first line is used to set the verification flag, so when the client connects to the server, a callback function will be called to check the certificate. This function can be set by the programmer, if there is no special callback function, then the default ctx callback function will be used. Because we need to use the CA's certificate to verify the certificate of the server, so the second is used to tell the TLS where is the CA's certificate.

For third thing, because the client program allows the user to input the hostname and port number of the VPN server, so we need to add two lines in the client program, so the program will be enforced to check it. These two lines are

```
X509_VERIFY_PARAM *vpm = SSL_get0_param(ssl);
X509_VERIFY_PARAM_set1_host(vpm, hostname, 0);
```

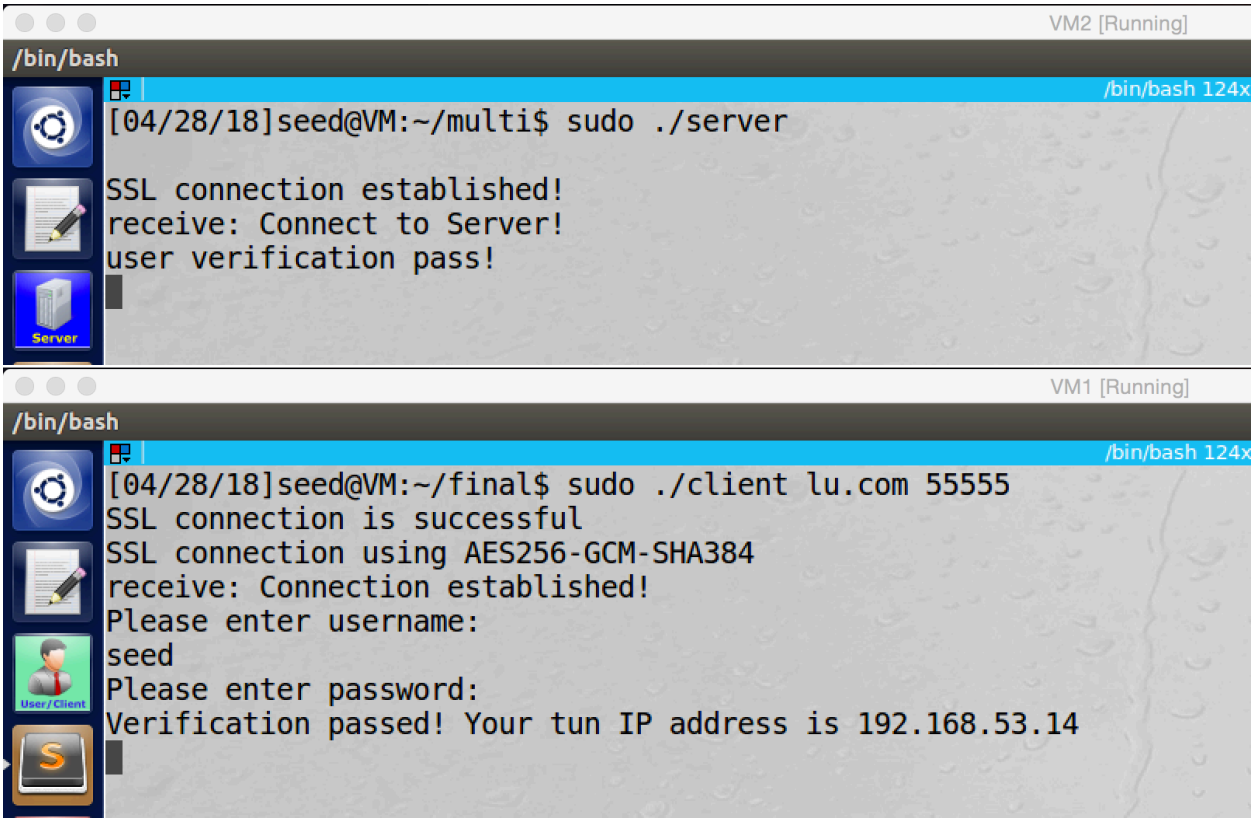
These two lines will check the host name which typed by the users in the **command line** with the common name on the server's certificate. If they match, then the verification will pass. Otherwise, the client will disconnect from the server.



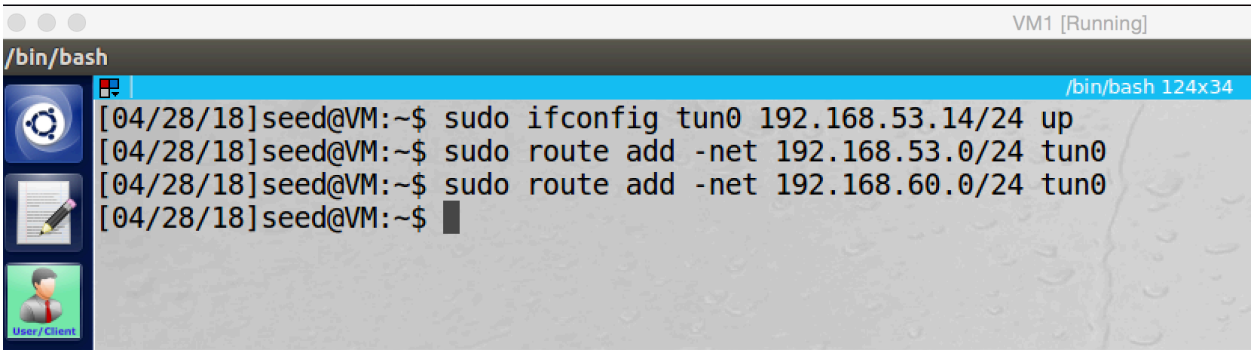

when we use different host name to connect the VPN server, the connection cannot be established

**Task 3: Encrypting the Tunnel**

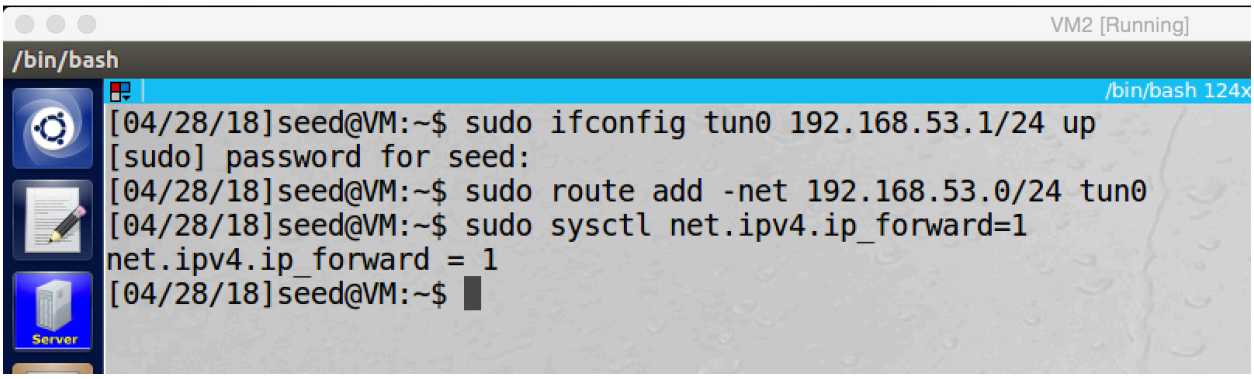
After the TCP connection and TLS session established, now the server and client can communicate with each other. Now we need to establish the VPN tunnel and set routing table, then the VPN tunnel can work and the tunnel will be encrypted.



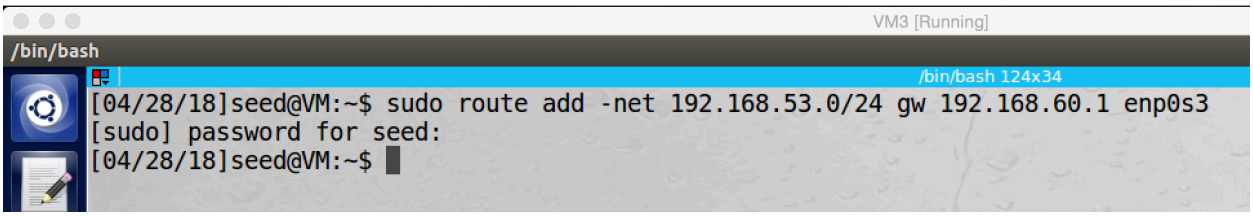

screenshot1. The VPN tunnel is established, and the server assigned tun IP address to the current client



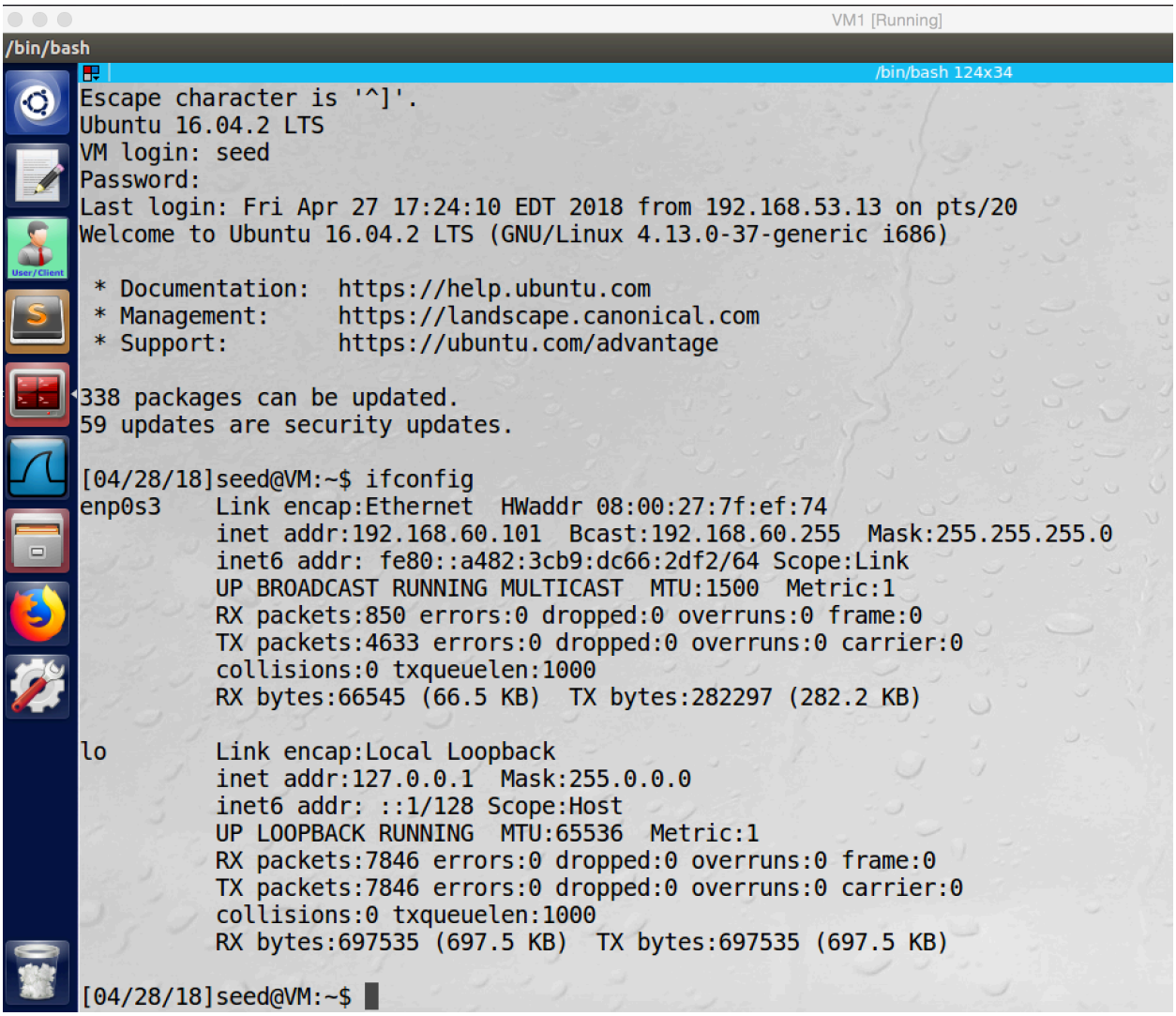

screenshot2. Setup routing table on VPN client.



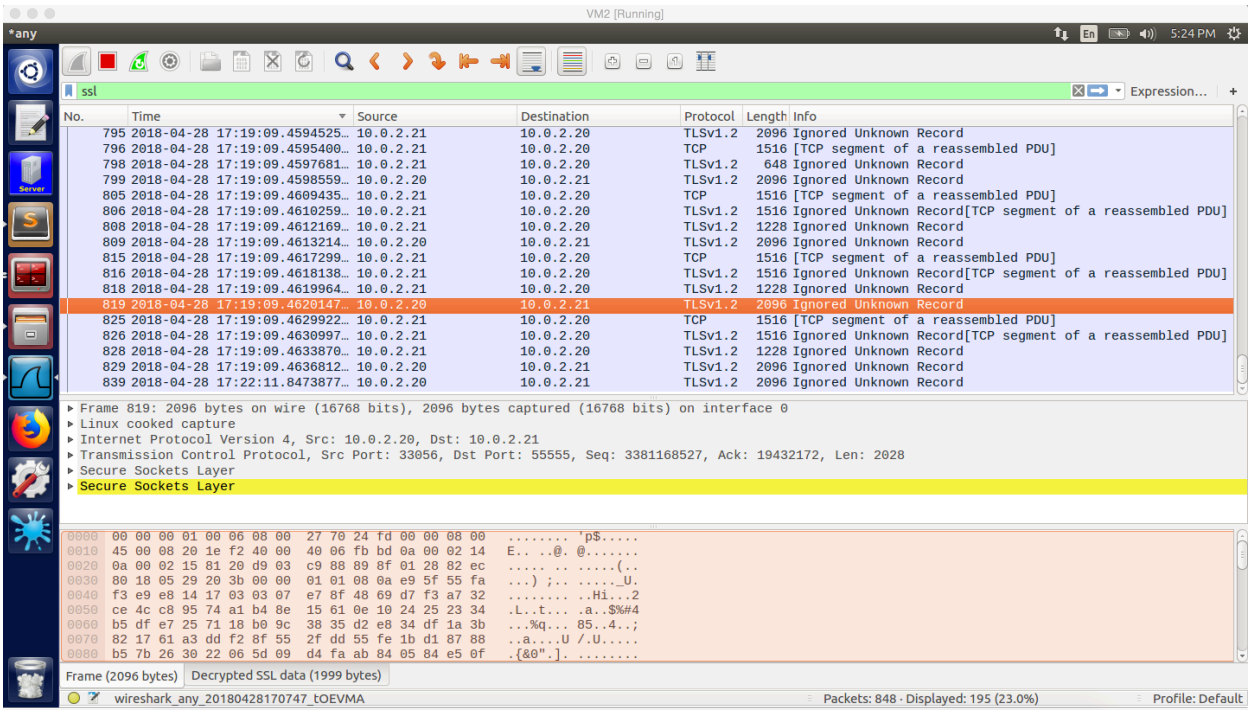
screenshot3. Setup routing table on VPN server



screesnhot4. Adding rule on routing table of Host V



screenshot5. telnet from VPN client to Host V successfully




screesnhot6. Wireshark captures the whole traffic; all packets are encrypted. Because I applied the private key of the server into Wireshark, so Wireshark can decrypt those packets.

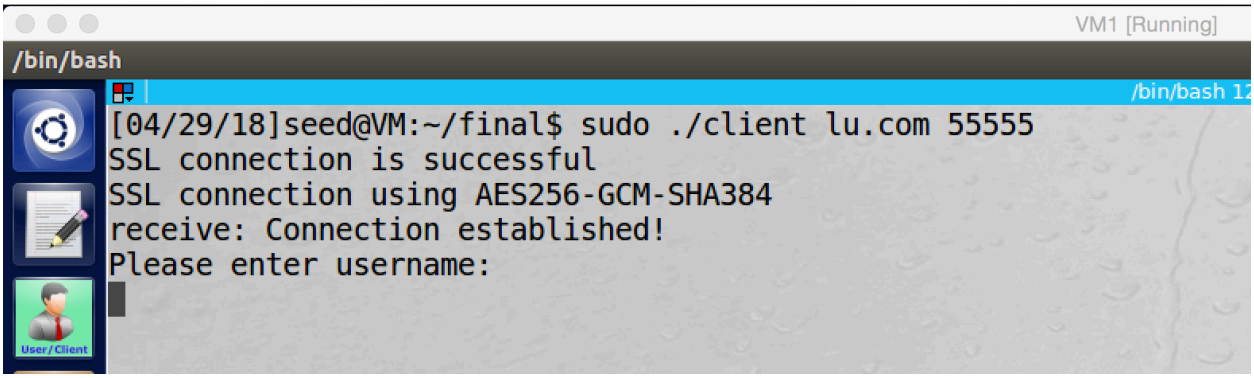
**Observation and Explanation:**

After server and client exchange key, they can start communication and use the key to encrypt their communication. After the connection is established, the serve will send a tun IP address to the client (screenshot1). And then we need to setup the routing table of the client, server and Host V (screenshot 2 to 4). This is same as we did in previous task. Once the setup is finished, the VPN tunnel is established. Now, we can use the tunnel to reach Host V from VPN client. In my case, I did telnet from VPN client to Host V. As the screenshot5 shows, the telnet succeeds. In the screenshot6, the Wireshark captures the whole traffic, and we see all packets transferred by the VPN tunnel have protocol TLSv1.2, which means they are encrypted. The decrypted tag in the bottom is because I applied private key of the server to the Wireshark, so Wireshark can decrypt SSL packets from the server (10.0.2.21) with port 55555.

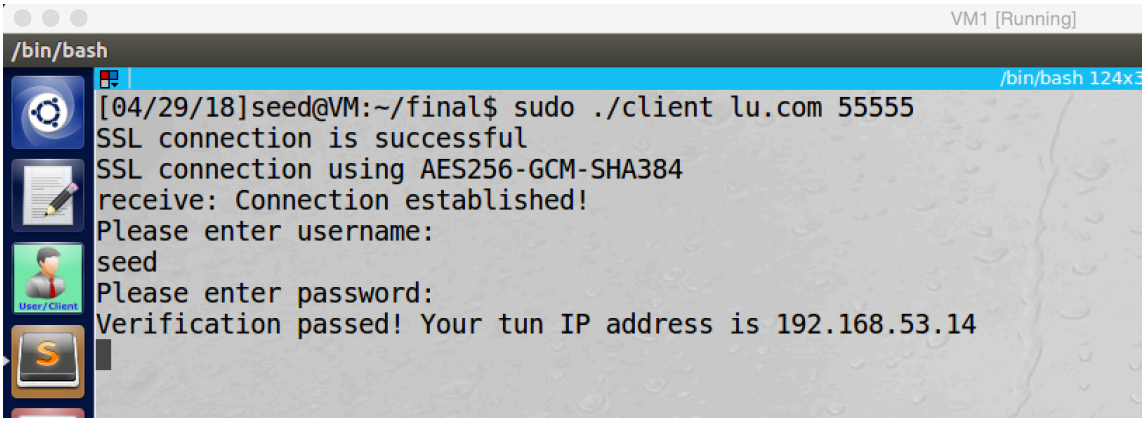
Once the key exchange is finished, client and server can transfer packets and encrypted these packets by the key. The encryption and decryption are done by two methods of openssl library, SSL_write () and SSL_read (). The SSL_write () will encrypt packets, and then it will send the packet out by system call write (). SSL_read () will call the system call read () to read one or more records from the TCP steam and decrypt them, verify their MAC, decompress the data before giving the date to the application.

**Task 5: Authenticating the VPN Client**

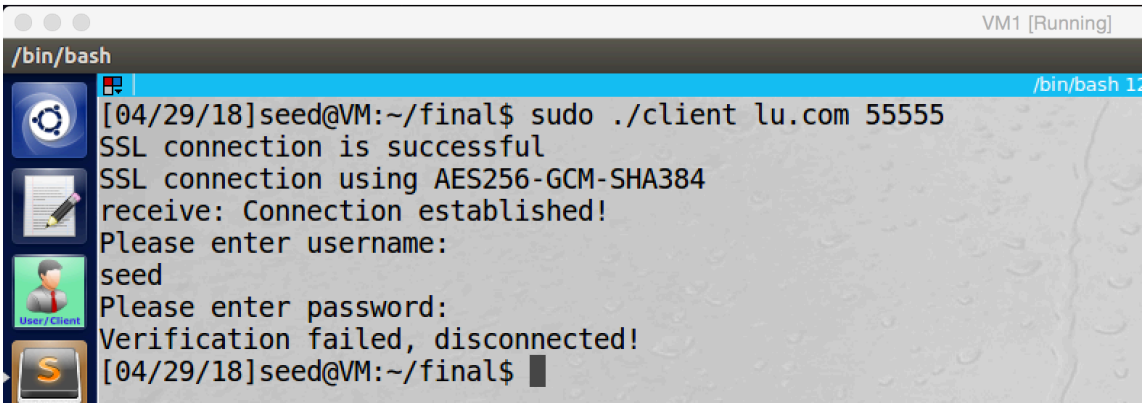
After client verify the server is valid, the TCP connection and TLS session will be established. And then the server also need to verify the user, the typical way is username and password.




screenshot1. After the connection is established, the client must enter valid username and password. The username and password is stored in the shadow file on the server.




screenshot2. After the user sends the username and password, the server will compare the username and password with the credential in the shadow file. If they match, the server will send a tun IP address to the client. Moreover, for the security purpose, I hide the password which is typed by user on client machine.



screenshot3. If the client sends wrong username or password, the verification fails. And the server will send fail message to the user and disconnect the connection.

**Observation and Explanation:**

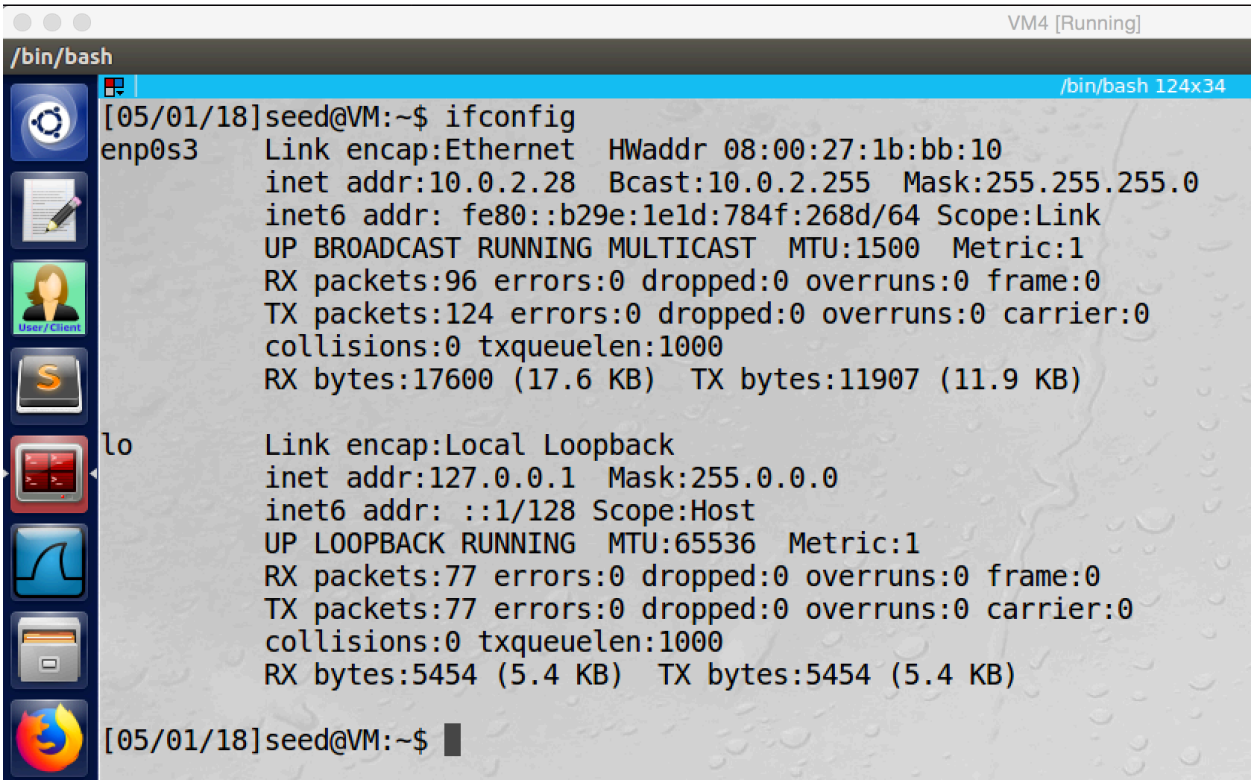
Client uses certificate to verify server, but because client usually does not have certificate, server cannot use the same method to verify client. The typical way for the server to verify client which is to use username and password. I did same thing in this task. After the client verified server, the TCP connection and TLS session are established. And then the server will require client to send his/her credentials (screenshot1). If the client sends valid credential, the verification will pass, and the server will allocate a tun IP address for the current client (screenshot2). If the client sends invalid credential, then the server will send error message to the client and disconnect the connection (screenshot3).

On the server side, the username and password is stored in the shadow file, once the server receive the credential which sent by the user, the server will compare it with the credential in the shadow file. If they match, it will send tun IP address to the client. Otherwise, it will disconnect the connection.

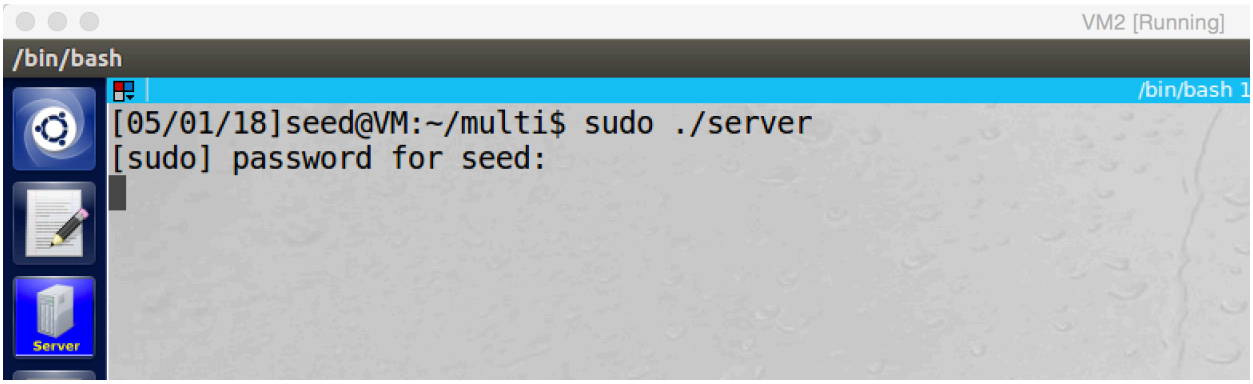
On the client side, once the connection is established, client will receive message to enter user name and password. For security purpose, the password typed by the user will not appear on the screen. After the client passes the verification, he/she can setup the routing table and use the VPN.

**Task 6: Supporting Multiple Clients**

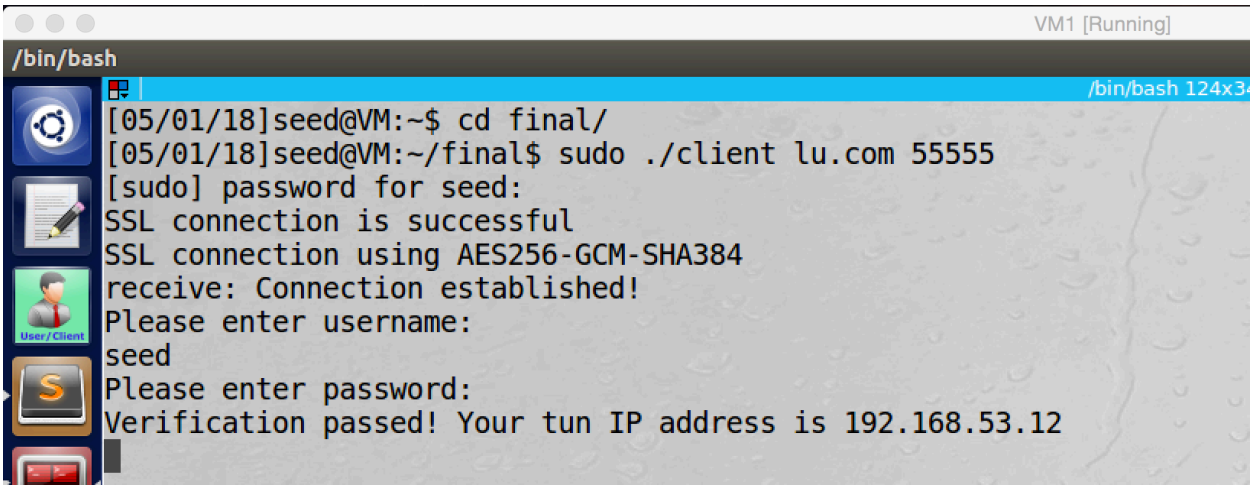
In this task, I add a new VPN client (VM4), which only connects to the Nat Network with IP address 10.0.2.28.



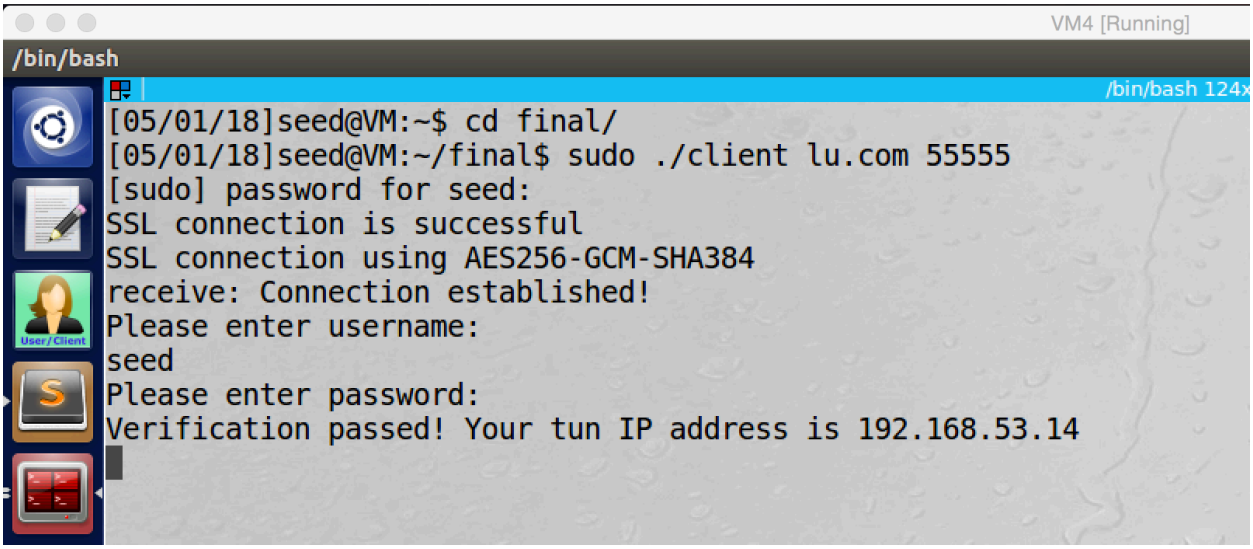

screenshot0. The information for the new VPN client.



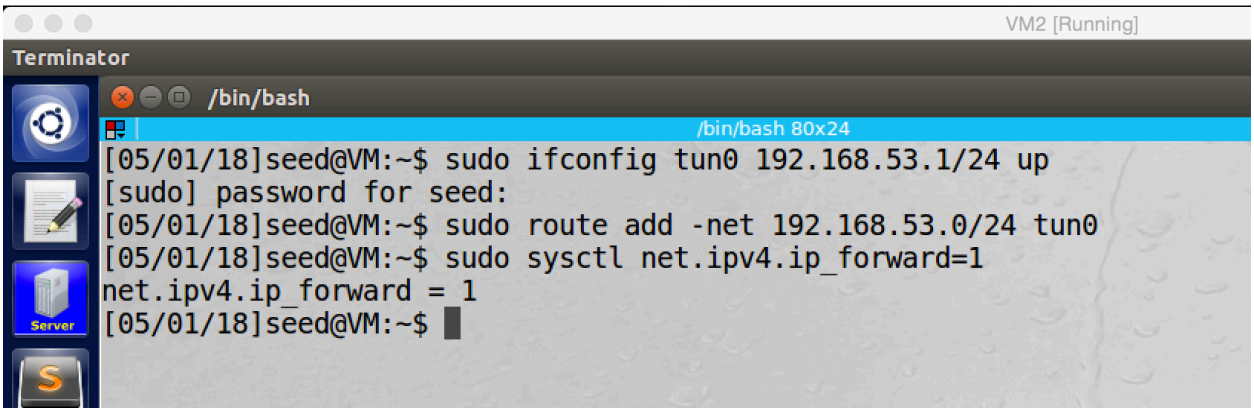

screenshot 1. Starting the VPN server



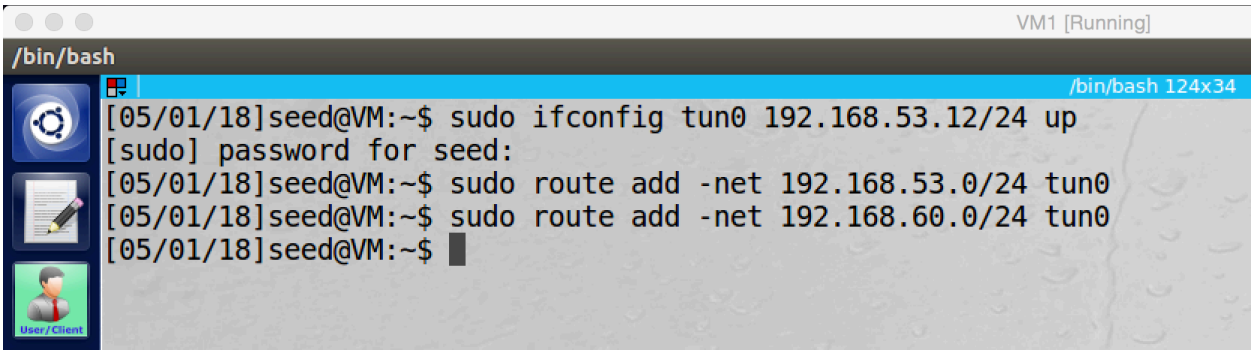

screenshot2. Clinet1 connects to VPN server



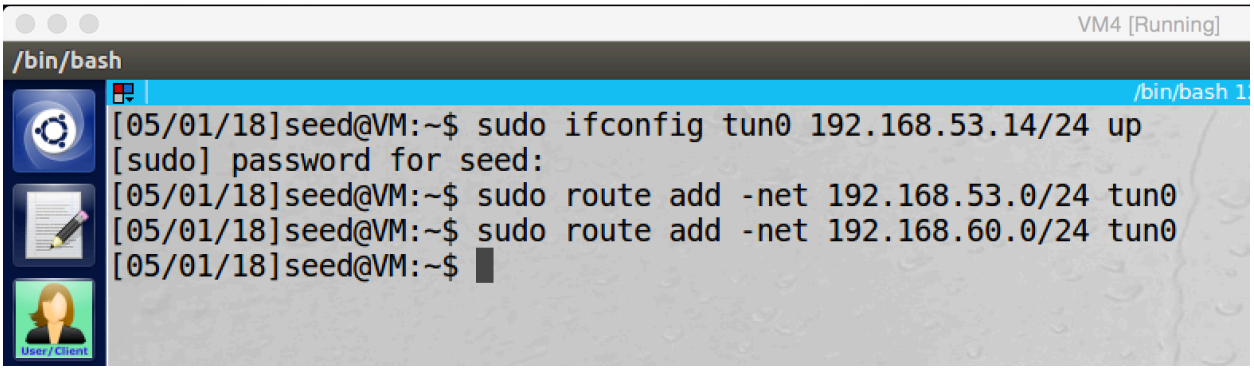

screenshot3. Client2 connects to VPN server



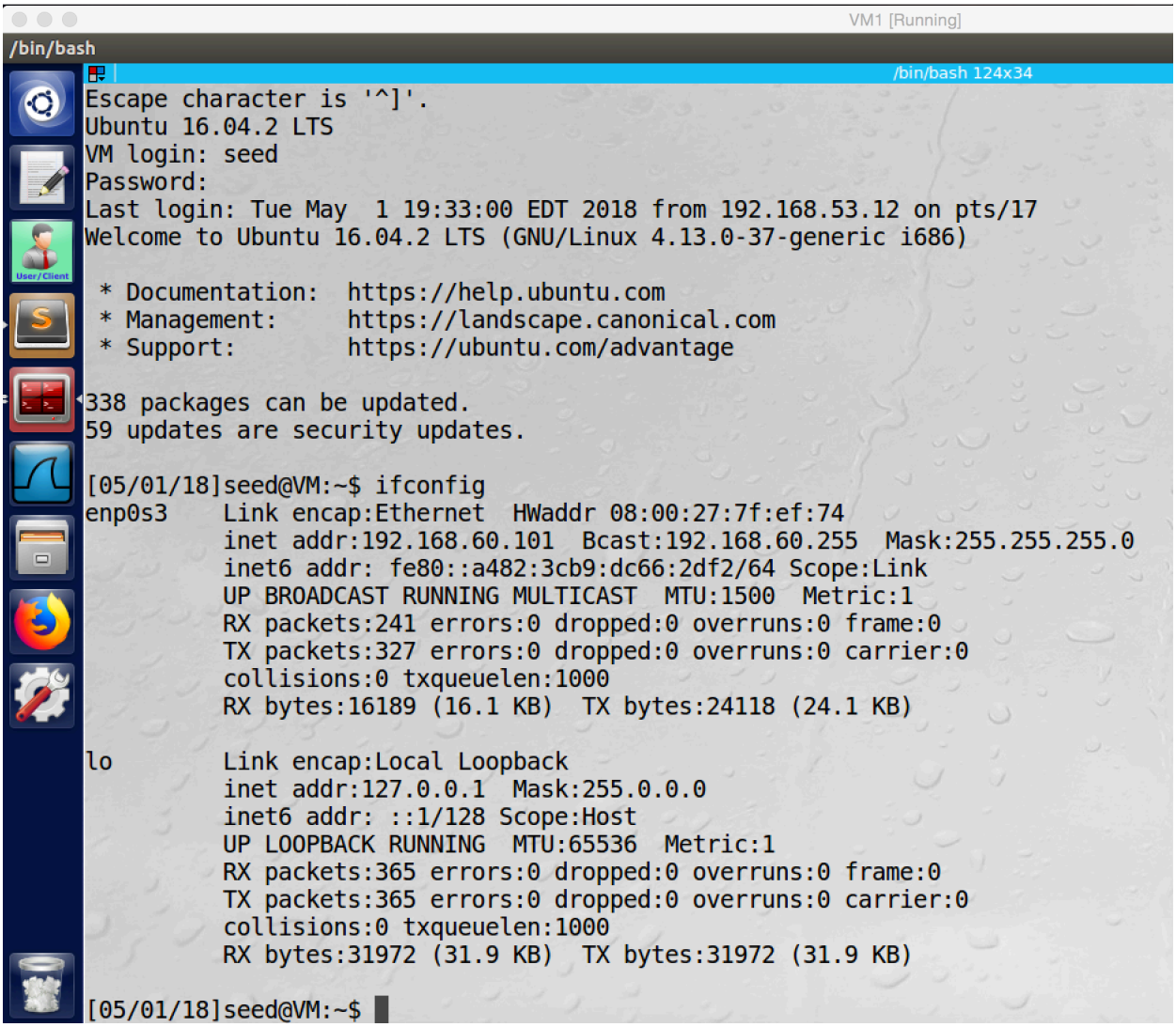

screenshot4. Configuring routing table on VPN server



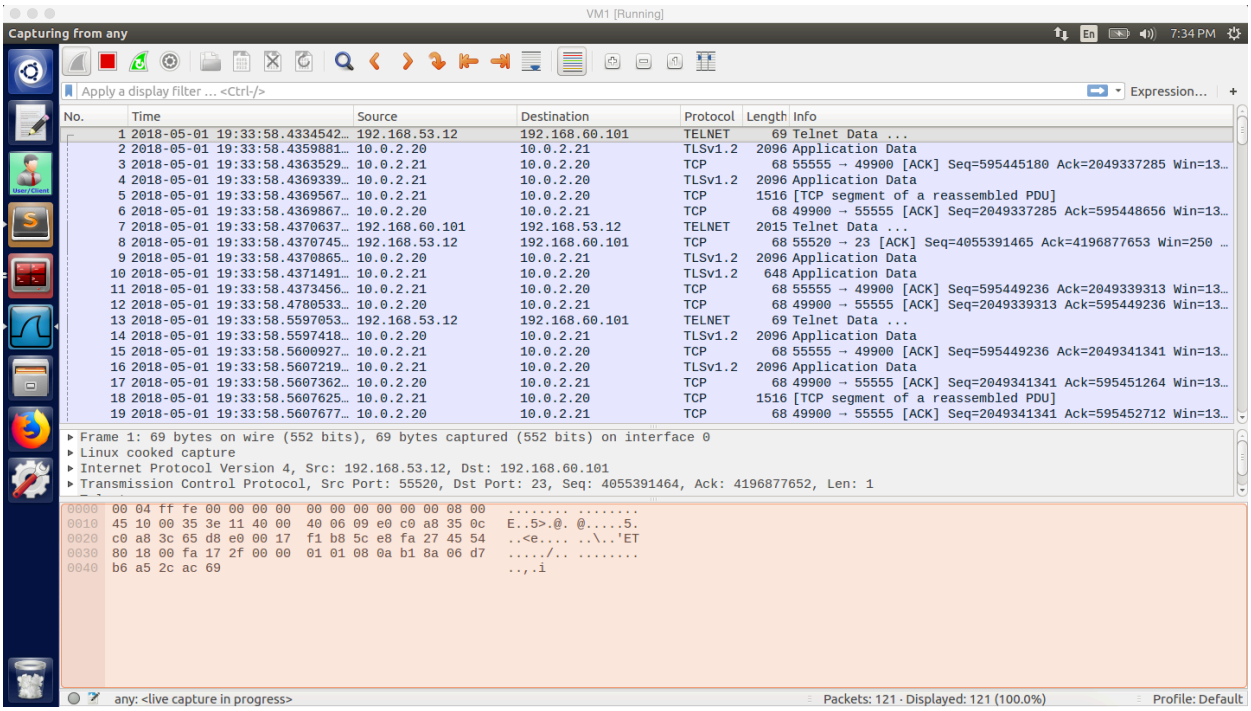
screenshot5. Configuring routing table on Clinet1



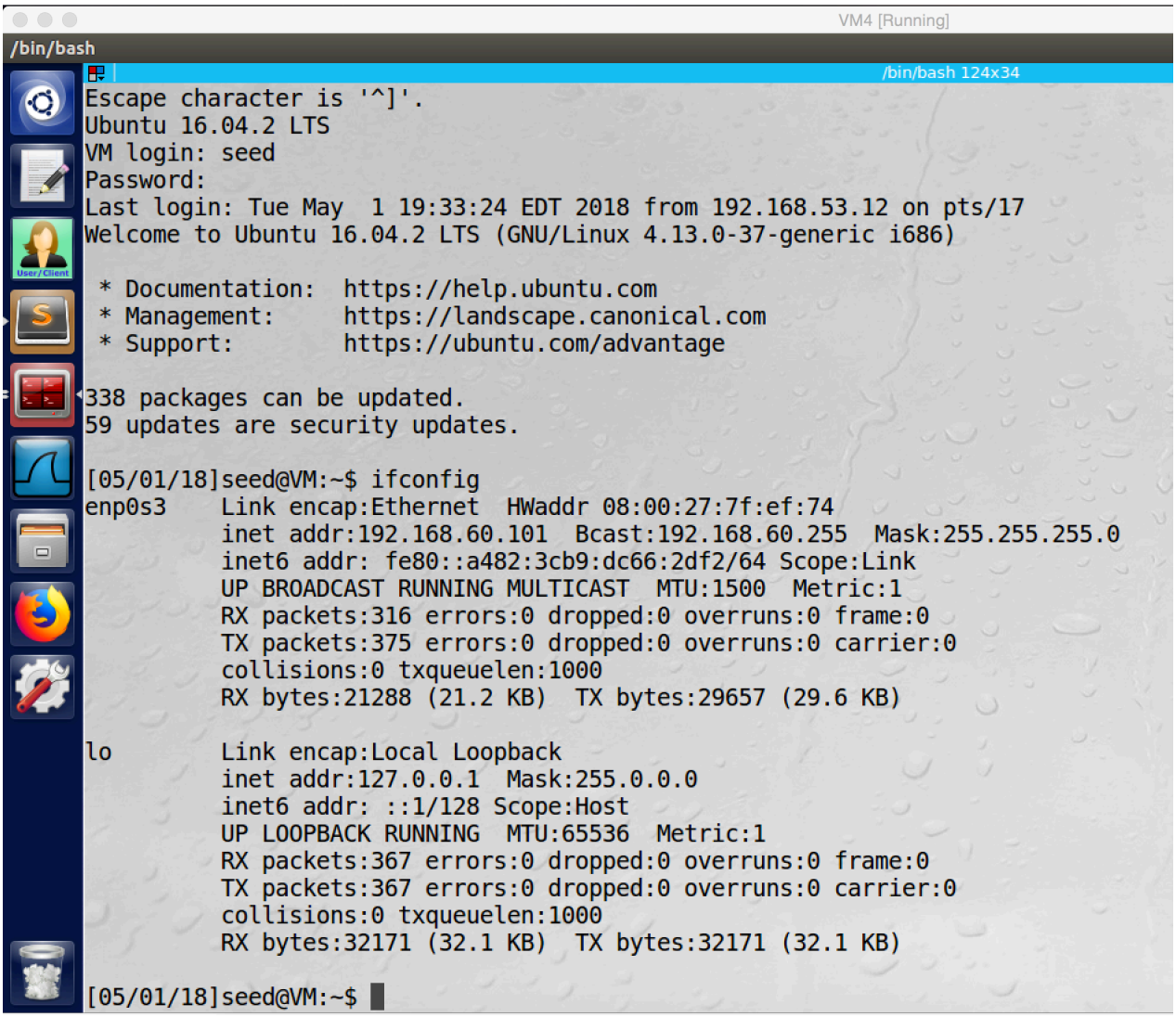
screenshot6. Configuring routing table on Clinet2



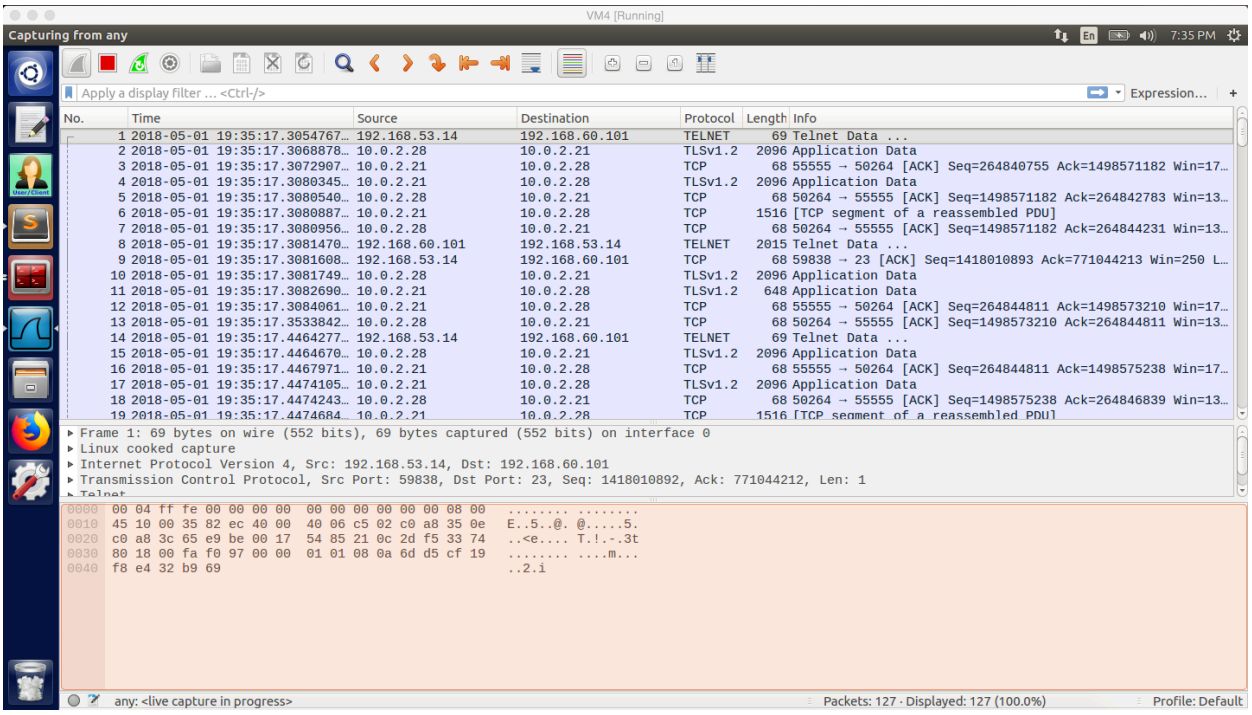
screenshot7. Telnet from Clinet1 to Host V successfully



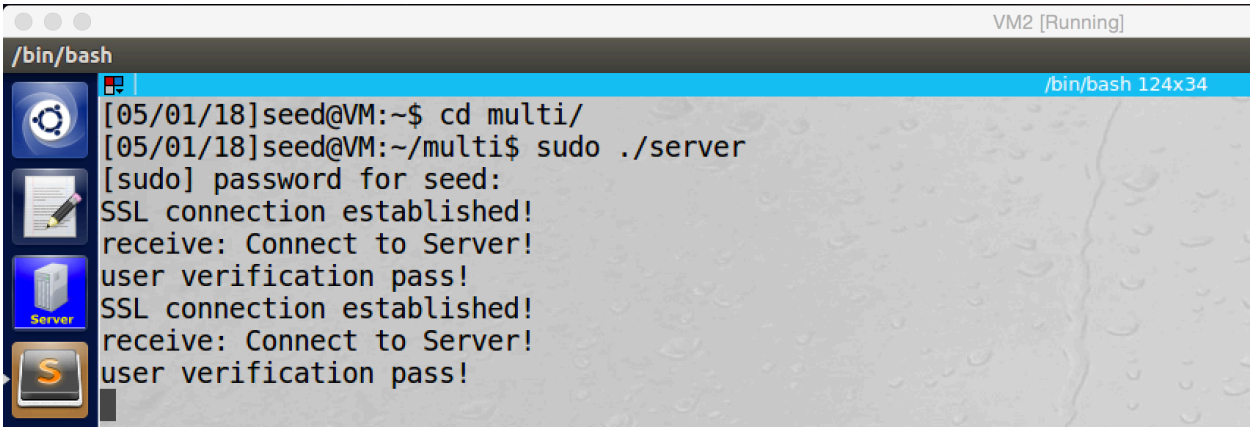

screenshot8. Wireshark captures the whole traffic, packets are transferred by the VPN tunnel, and the tunnel is encrypted.



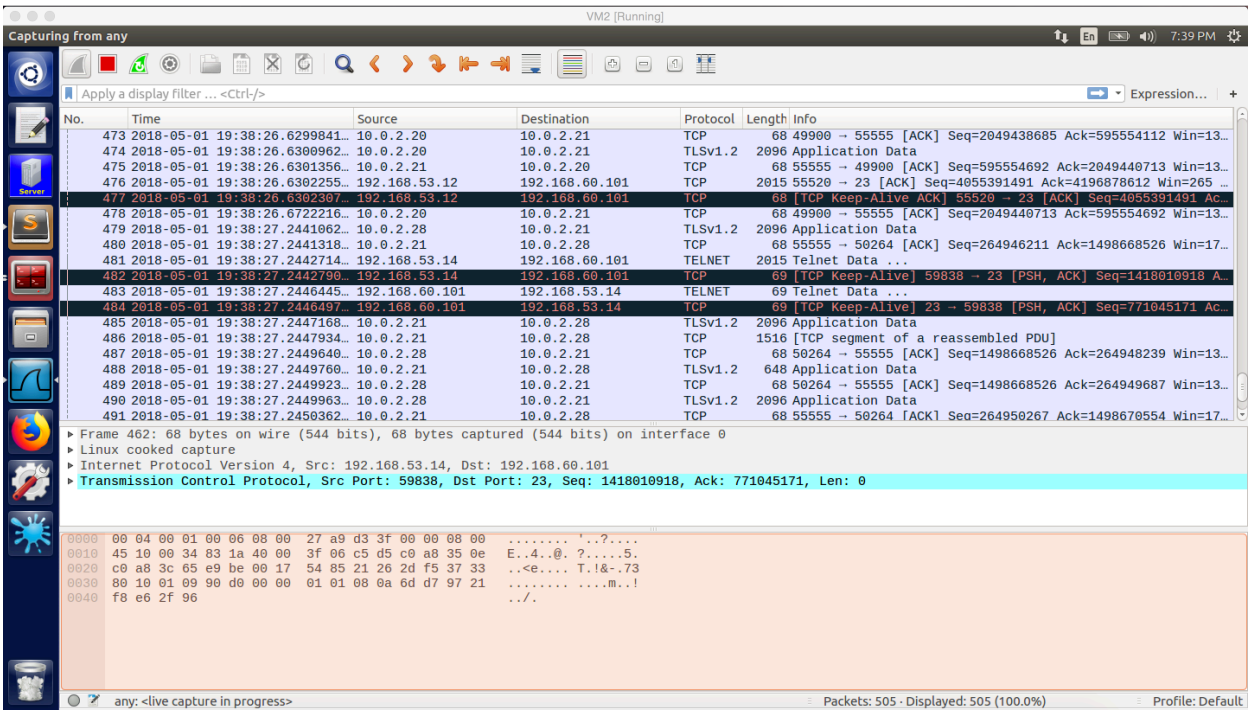
screenshot9. Telnet from Client2 to Host V successfully




screenshot10. Wireshark captures the whole traffic, packets are transferred by the VPN tunnel, and the tunnel is encrypted.




screenshot11. VPN server receive two connections from two different clients




screenshot12. Wireshark on VPN server, it captures packets from two different tunnels. Tunnel of Client1 (10.0.2.20 and 10.0.2.21), tunnel of Client2 (10.0.2.28 and 10.0.2.21)

**Observation and Explanation:**

As the above screenshots show, two VPN clients connect to the VPN server at same time, and they work well. First, I start up the VPN server, and it is waiting for connection from client (screenshot1). And then I run clinet1 and client2 (screenshot 2 to 3), and they get different tun IP address, for client1 it is 192.168.53.12, for client it is 192.168.53.14. From screenshot 4 to 6, I configure the routing table and up the tun0 for VPN server, clinet1 and client2. The Host V is already setup, so now I can telnet from clients to Host V. I first telnet from client1 to Host V (screenshot 7 to 8), the telnet succeeds. And then I telnet from client2 to Host V (screenshot 9 to 10), the telnet succeeds as well. And then I go back to VPN server, it has received two connections, and the connections are established well, no error message printed (screenshot 11). The screenshot12 shows the whole traffic, there are two tunnels, tunnel of Client1 (10.0.2.20 and 10.0.2.21) and tunnel of Client2 (10.0.2.28 and 10.0.2.21).

If there is a connection from client, the VPN server parent program need to create a child process for the connection, so the parent program need to use accept () to monitor the incoming connection; on the other hand, the parent program also need to monitor any packet from the private network, and then it can forward it to the corresponding child process. However, all these two duties will block the parent process, so it may cannot do these two things at same time. In my approach, when the VPN server run, the parent process will allocate a range of IP address, and it will create an array to map these IP address to each child. If there are 10 IP address, then there are 10 child processes (the number can be change, now it can support up to 255 IP addresses, which means 255 child processes). After the allocation, the parent will also allocate same number of pipes for each IP address. And then, the parent process will fork such number of children, and these children takes care to accept new

connection from clients. Once there is client connection, an arbitrary child process will accept it and create an individual TCP connection and TLS session for it, it will also send the tun IP address to the client as well if the verification succeeds. Because children block on the accept () to accept new client connection, the parent process can focus on monitor packets from private network. If there is a packet from private network, the parent will read its destination IP address, and it will use this IP address to find the corresponding child process by the array and send the packet to the child process by pipe.

**Appendix:**

**Code for VPN Server**

```
#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>
#include <openssl/ssl.h>
#include <openssl/err.h>
#include <netdb.h>
#include <unistd.h>
#include <shadow.h>
#include <crypt.h>
#include <stdlib.h>

#include "myheader.h"

#define PORT_NUMBER 55555
#define BUFF_SIZE 2000
#define CHK_SSL(err) if ((err) < 1) { ERR_print_errors_fp(stderr); exit(2); }
#define CHK_ERR(err,s) if ((err)==-1) { perror(s); exit(1); }

struct sockaddr_in peerAddr;

//Once the server started, it will allocate a range of IP address for following client
void allocateIP(char **childPipe, char *base, int increment, int pipNum)
{
   char *new_str;
   for(int i=0; i<pipNum; i++){
      int length = snprintf( NULL, 0, "%d", increment );
      char* convert = malloc( length + 1 );
      snprintf( convert, length + 1, "%d", increment );
      if((new_str = malloc(strlen(base)+strlen(convert)+1)) != NULL){
         new_str[0] = '\0';   // ensures the memory is an empty string
         strcat(new_str,base);
         strcat(new_str,convert);
      }
      //map the child i to current IP address
```

```
        childPipe[i] = new_str;
        //increment the IP address
        increment = increment+1;
    }
    return;
}

//compare the username and password which sent by the client
//with the credential in the shadow file
//if they match, return 1 else return -1
int login(char *user, char *passwd)
{
    struct spwd *pw;
    char *epasswd;
    pw = getspnam(user);
    if (pw == NULL) {
        return -1;
    }
    epasswd = crypt(passwd, pw->sp_pwdp);
    if (strcmp(epasswd, pw->sp_pwdp)) {
        return -1;
    }
    return 1;
}

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 initTCPServer() {
    struct sockaddr_in sa_server;
    int listen_sock;

    listen_sock= socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    CHK_ERR(listen_sock, "socket");
    memset(&sa_server, 0, sizeof(sa_server));
    sa_server.sin_family = AF_INET;
    sa_server.sin_addr.s_addr = htonl(INADDR_ANY);
    sa_server.sin_port = htons(PORT_NUMBER);

    int err = bind(listen_sock, (struct sockaddr*)&sa_server, sizeof(sa_server));
    CHK_ERR(err, "bind");
    err = listen(listen_sock, 5);
    CHK_ERR(err, "listen");

    return listen_sock;
}
```

```
void socketSelected (int tunfd, int sockfd, SSL *ssl){
   int  len;
   char buff[BUFF_SIZE];

   //printf("Got a packet from the tunnel\n");

   bzero(buff, BUFF_SIZE);
   int err = SSL_read (ssl, buff, sizeof(buff)-1);
   buff[err] = '\0';
   write(tunfd, buff, err);
}

//child processes use this function to receive packet from parent
void parentSelected (int parentPipe, int sockfd, SSL *ssl){

   int  len;
   char buff[BUFF_SIZE];
   //read packet from pipe
   len = read(parentPipe, buff, sizeof(buff)-1);
   //send packet out by correct tunnel
   SSL_write (ssl, buff, sizeof(buff)-1);
}

void startVPN (int tunfd, int sockfd, SSL* ssl, int parentPipe) {

   while (1) {
      fd_set readFDSet;

      FD_ZERO(&readFDSet);
      FD_SET(sockfd, &readFDSet);
      FD_SET(tunfd, &readFDSet);
      FD_SET(parentPipe, &readFDSet);
      select(FD_SETSIZE, &readFDSet, NULL, NULL, NULL);

      if (FD_ISSET(sockfd, &readFDSet)) socketSelected(tunfd, sockfd, ssl);
      if (FD_ISSET(parentPipe, &readFDSet)) parentSelected(parentPipe, sockfd, ssl);
   }
}

int main (int argc, char * argv[]) {

   //initialize TLS
   char readbuf[2000];
   char readbuf2[2000];
   SSL_METHOD *meth;
   SSL_CTX* ctx;
   SSL *ssl;
   int err;
   int sockfd;
   int pipNum = 10;//number of child process, we will have. This number can be changed
   char *childPipe[pipNum];//create a table, which will map every child process to different tun IP address
   char *base = "192.168.53.";//base IP address of TUN
   int increment = 10;//the tun IP address will start from 192.168.53.10

   //allocating a range IP address and pipe for following client
```

```
allocateIP(childPipe, base, increment, pipNum);

// Step 0: OpenSSL library initialization
SSL_library_init();
SSL_load_error_strings();
SSLeay_add_ssl_algorithms();

// Step 1: SSL context initialization
meth = (SSL_METHOD *)TLSv1_2_method();
ctx = SSL_CTX_new(meth);
SSL_CTX_set_verify(ctx, SSL_VERIFY_NONE, NULL);
// Step 2: Set up the server certificate and private key
SSL_CTX_use_certificate_file(ctx, "lu-cert.pem", SSL_FILETYPE_PEM);
SSL_CTX_use_PrivateKey_file(ctx, "lu-key-dec.pem", SSL_FILETYPE_PEM);
// Step 3: Create a new SSL structure for a connection
ssl = SSL_new (ctx);

//setup TCP socket, waiting for connection
struct sockaddr_in sa_client;
size_t client_len;
int listen_sock = initTCPServer();

//create tun interface
int tunfd  = createTunDevice();

//allocate pipe for each child processes
int count = 0;
int fd[2*pipNum];
pid_t pid;
//initialize all pipes
for(int i = 0; i < pipNum; i++){
    pipe(&fd[2*i]);
}

while(1){

    //after fork child processes, parent keeps to check tun interface to see if there is any packet
    //if there is packet from private network, the parent will forward it to correct child process
   while(count == pipNum){
                int  len = 0;
                char buff[BUFF_SIZE];

        //parent got packet from tun
                bzero(buff, BUFF_SIZE);
                len = read(tunfd, buff, BUFF_SIZE);

        //get the head of the packet and check it IP address fields
                struct ipheader* ip = (struct ipheader*)buff;
                int ip_header_len = ip->iph_ihl*4; // compute IP header length

        //comparing the destination IP with each tunnel IP
        //and forwarding the packet to corresponding child process
               for(int i=0; i<pipNum;i++){
                   if(strcmp(inet_ntoa(ip->iph_destip), childPipe[i]) == 0){
                      write(fd[2*i+1], buff, sizeof(buff));
                   }
```

```
        }

 }

//fork child processes
if((pid = fork()) == -1) {
    perror("fork");
    exit(1);
}
if(pid>0) { //parent proces
    close(fd[count*2]); // Close the input end of the pipe.
}
else { //child process
    close(fd[count*2+1]); // Close the output end of the pipe.
    int parentPipe = fd[count*2]; //get current child pipe
    int childID = count; //get current child ID
    while(1){
        //child processes take the responsibility to accept following client
        sockfd = accept(listen_sock, (struct sockaddr*)&sa_client, &client_len);

        //TCP socket is ready, setup TLS section
        SSL_set_fd (ssl, sockfd);
        err = SSL_accept (ssl);
        CHK_SSL(err);
        printf ("SSL connection established!\n");
        err = SSL_read (ssl, readbuf, sizeof(readbuf)-1);
        readbuf[err] = '\0';
        printf("receive: %s\n", readbuf);
        err = SSL_write (ssl, "Connection established!", strlen("Connection established!"));

        //client verification
        err = SSL_write (ssl, "Please enter username:", strlen("Please enter username:"));
        err = SSL_read (ssl, readbuf, sizeof(readbuf)-1);
        readbuf[err] = '\0';

        err = SSL_write (ssl, "Please enter password:", strlen("Please enter password:"));
        err = SSL_read (ssl, readbuf2, sizeof(readbuf)-1);
        readbuf2[err] = '\0';

        //if the user does not provide correct credentials
        //then the connection will be terminated
        if(login(readbuf, readbuf2) == 1){
            printf("user verification pass!\n");
            err = SSL_write (ssl, childPipe[childID], sizeof("xxx.xxx.xxx.xxx"));
            //credential are correct, so start VPN
            startVPN(tunfd, sockfd, ssl, parentPipe);
        }
        else {
            printf("wrong username or password, disconnected!\n");
            err = SSL_write (ssl, "bad", sizeof("bad"));
            close(sockfd);
            SSL_shutdown(ssl);
            SSL_free(ssl);
        }

        close(sockfd);
```

```
            SSL_shutdown(ssl);
            SSL_free(ssl);
          }
        }
        count++;
    }

}
```

**Code for VPN Client**

```
#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>
#include <openssl/ssl.h>
#include <openssl/err.h>
#include <netdb.h>
#include <netinet/in.h>
#include <fcntl.h>
#include <termios.h>

#define BUFF_SIZE 2000
#define CHK_SSL(err) if ((err) < 1) { ERR_print_errors_fp(stderr); exit(2); }
#define CA_DIR "ca_client"

struct sockaddr_in peerAddr;
struct addrinfo hints, *result;
int PORT_NUMBER = 55555;
const char *hostname;

//this function is used to hide the password which typed by the user
int getch() {
  struct termios oldtc;
  struct termios newtc;
  int ch;
  tcgetattr(STDIN_FILENO, &oldtc);
  newtc = oldtc;
  newtc.c_lflag &= ~(ICANON | ECHO);
  tcsetattr(STDIN_FILENO, TCSANOW, &newtc);
  ch=getchar();
  tcsetattr(STDIN_FILENO, TCSANOW, &oldtc);
  return ch;
```

```
}

//initialize a tun device
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;
}

//initialize ssl parameter
SSL* setupTLSClient(const char* hostname,  SSL_CTX* ctx)
{
   // Step 0: OpenSSL library initialization
   // This step is no longer needed as of version 1.1.0.
   SSL_library_init();
   SSL_load_error_strings();
   SSLeay_add_ssl_algorithms();

   SSL_METHOD *meth;
   SSL* ssl;

   meth = (SSL_METHOD *)TLSv1_2_method();
   ctx = SSL_CTX_new(meth);

   //this line is used to set the verification flag
   //so when the client connect to the server, a callback function will be called
   //to check the certificate, this function can be set by programmer
   //if there is no special callback function, then the default ctx callback function will be used
   SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER, NULL);

   //Telling the client program where are CA's certificates
   //which are used to verify server's certificate
   if(SSL_CTX_load_verify_locations(ctx,NULL, CA_DIR) < 1){
      printf("Error setting the verify locations. \n");
      exit(0);
   }

   //create an SSL data structure, which will be used for making a TLS connection
   ssl = SSL_new (ctx);

   //these two lien is used to check the hostname typed by the user
   //matches the common name on the server's certificate or not
   X509_VERIFY_PARAM *vpm = SSL_get0_param(ssl);
   X509_VERIFY_PARAM_set1_host(vpm, hostname, 0);

   return ssl;
}

int connectToTCPServer(const char *hostname){
```

```
    //get host IP address from hostname
    hints.ai_family = AF_INET;
    int error = getaddrinfo(hostname, NULL, &hints, &result);
    if (error) {
      fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(error));
      exit(1);
    }
    struct sockaddr_in* ip = (struct sockaddr_in *) result->ai_addr;

    // Create a TCP socket
    int sockfd= socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

    //fill in destination infromation
    memset(&peerAddr, 0, sizeof(peerAddr));
    peerAddr.sin_family = AF_INET;
    peerAddr.sin_port = htons(PORT_NUMBER);
    peerAddr.sin_addr.s_addr = inet_addr((char *)inet_ntoa(ip->sin_addr));

    //connect to the destination
    connect(sockfd, (struct sockaddr*) &peerAddr,
          sizeof(peerAddr));

    return sockfd;
}

void tunSelected(int tunfd, int sockfd, SSL *ssl){
    int  len;
    char buff[BUFF_SIZE];

    bzero(buff, BUFF_SIZE);
    len = read(tunfd, buff, BUFF_SIZE);
    //encrypt and send packet by SSL_write
    SSL_write (ssl, buff, sizeof(buff)-1);
}

void socketSelected (int tunfd, int sockfd, SSL *ssl){
    int  len;
    char buff[BUFF_SIZE];

    bzero(buff, BUFF_SIZE);
    //receive and decrypt by SSL_read
    int err = SSL_read (ssl, buff, sizeof(buff)-1);
    buff[err] = '\0';
    write(tunfd, buff, err);
}

void startVPN (int sockfd, SSL* ssl) {

      int tunfd  = createTunDevice();

      //monitor transfer packets between two file descriptors (one for TUN,
      //one for socket) by using system call select()
      while (1) {
        fd_set readFDSet;
        //using FD_SET to store monitored file descriptors in a set
```

```
        FD_ZERO(&readFDSet);
        FD_SET(sockfd, &readFDSet);
        FD_SET(tunfd, &readFDSet);
        //given the set to select()
                                //this function will block the process until data are available on one
                                //of the file descriptors
        select(FD_SETSIZE, &readFDSet, NULL, NULL, NULL);
        //FD_ISSER is used to know which file descriptor has received data
        if (FD_ISSET(tunfd,  &readFDSet)) tunSelected(tunfd, sockfd, ssl);
        if (FD_ISSET(sockfd, &readFDSet)) socketSelected(tunfd, sockfd, ssl);
    }
}


int main (int argc, char * argv[]) {

  const char *hostname;
  SSL_CTX *ctx;
  int tunfd, sockfd;

  if (argc > 1) hostname = argv[1];
  else {
    printf("Please enter a legal host name.\n");
    return 0;
  }
  if (argc > 2) PORT_NUMBER = atoi(argv[2]);

  //TLS initialization
  SSL *ssl = setupTLSClient(hostname, ctx);

  //create a TCP conncetion
  sockfd = connectToTCPServer(hostname);

  //TLS handshake
  char readbuf[2000];
  SSL_set_fd(ssl, sockfd);
  int err = SSL_connect(ssl); CHK_SSL(err);
  printf("SSL connection is successful\n");
  printf ("SSL connection using %s\n", SSL_get_cipher(ssl));

  err = SSL_write (ssl, "Connect to Server!", strlen("Connect to Server!"));
  err = SSL_read (ssl, readbuf, sizeof(readbuf)-1);
  readbuf[err] = '\0';
  printf("receive: %s\n", readbuf);

  //user verification
  char username[20];
  char password[20];
  char check[50];
  err = SSL_read (ssl, readbuf, sizeof(readbuf)-1);
  readbuf[err] = '\0';
  printf("%s\n", readbuf);

  scanf("%s", username);
  err = SSL_write (ssl, username, sizeof(username));
```

```
  err = SSL_read (ssl, readbuf, sizeof(readbuf)-1);
  readbuf[err] = '\0';
  printf("%s\n", readbuf);

  //hide user password
  int ch, i = 0;
  ch = getch();
  for(;;){
                ch = getch();
                if(ch == '\n') {
                        password[i] = '\0';
                        break;
                }
                else {
                        password[i] = (char)ch;
                        i++;
                }
  }
  err = SSL_write (ssl, password, sizeof(password));

  err = SSL_read (ssl, readbuf, sizeof(readbuf)-1);
  readbuf[err] = '\0';
  strncpy(check, readbuf, sizeof(readbuf));
  //if the credential typed by user is not correct
  //the server will send "bad" to client
  //and the connection will be closed
  //otherwise, the connection will established
  if (strcmp(check, "bad") == 0){
                printf("Verification failed, disconnected!\n");
    close(sockfd);
                SSL_shutdown(ssl);
                SSL_free(ssl);
    exit(0);
  }
  else {
                printf("Verification passed! Your tun IP address is %s\n", check);
  }

  //TCP connection and TLS session are established
  //strat the VPN
  startVPN(sockfd, ssl);

  close(sockfd);
  SSL_shutdown(ssl);
  SSL_free(ssl);
}
```