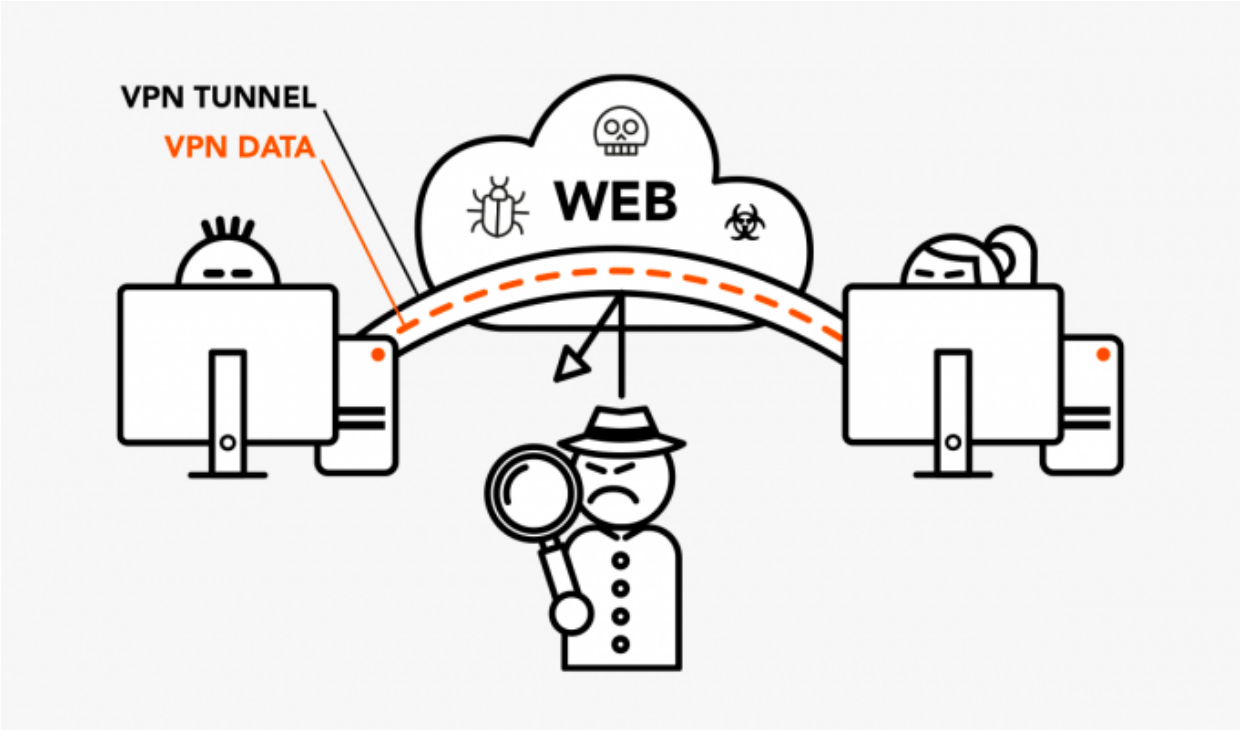
Virtual Private Network



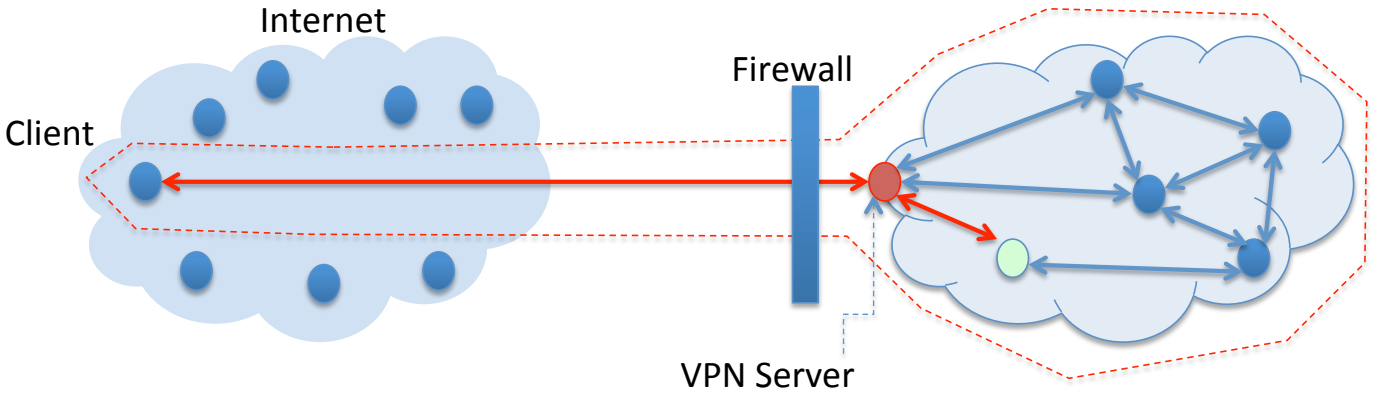
Introduction

- Private network - physically disconnected from the outside Internet
  - Users Authenticated
    - Still vulnerable if the internal resources use IP address as the basis for authentication
  - Content Protected
    - Communication within the private network cannot be sniffed from outside.
  - Integrity Preserved
    - Nobody from outside the network can spoof.
- If we grant access from outside to the private network, the attack surface will significantly broaden.

Virtual Private Network

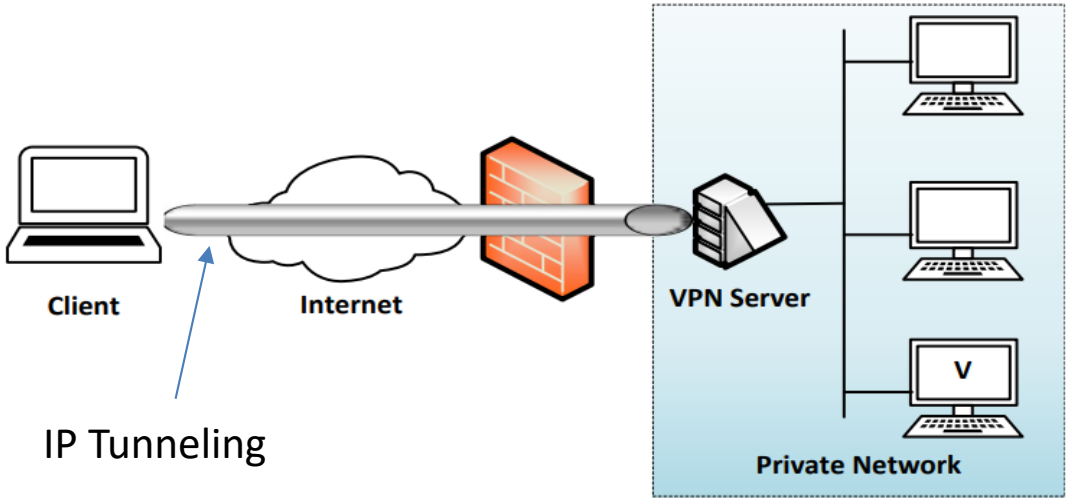
VPN allows users to create a secure, private network over a public network, such as the Internet.

- Outside computers must go through **the VPN server** to reach the hosts inside a private network via authentication.
- VPN server is exposed to the outside, and the internal computers are still protected via firewalls or reserved IP addresses.

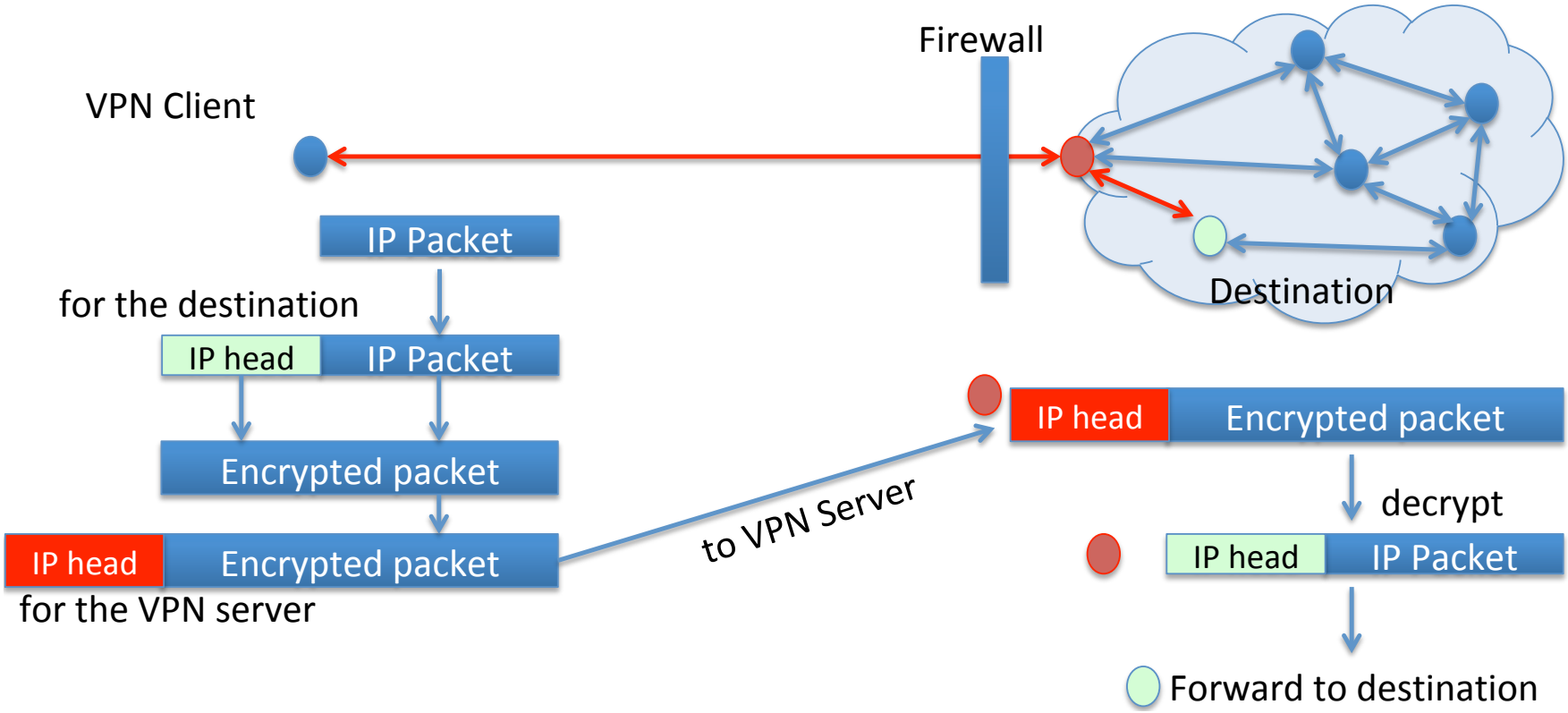


A Typical Setup

This is a typical VPN setup where the “Client” machine wants to connect with machine “V” on a private network. “Client” uses the “VPN Server” to get authenticated to the private network

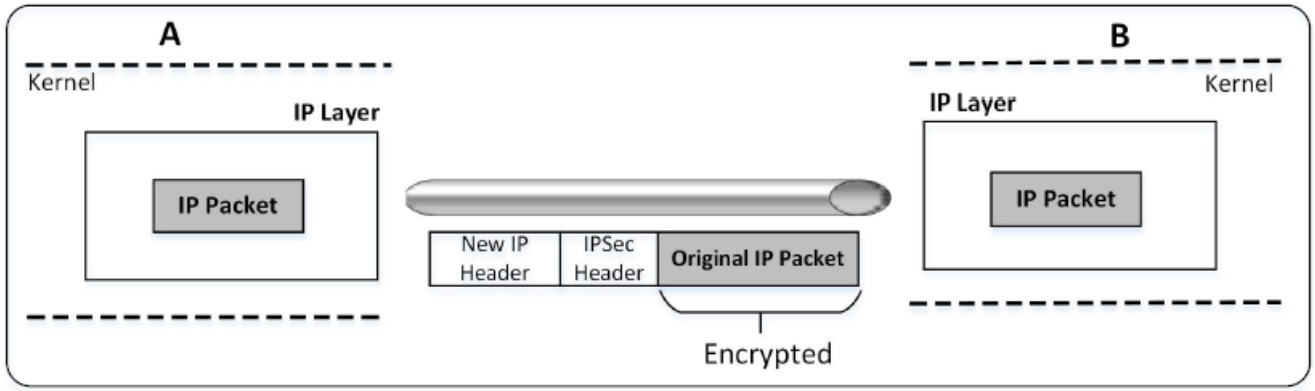


IP Tunneling



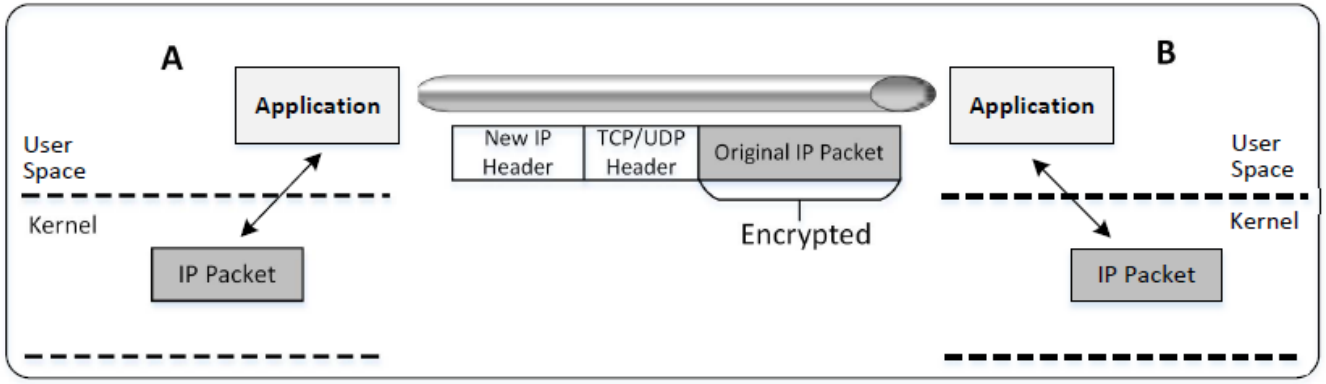
Two Types of IP Tunneling

- **IPSec tunneling**
  - It uses IPSec protocol which operates at the IP layer and has a tunneling mode.
  - The entire IP packet is encapsulated into a new IP packet with a new header added.
  - Done at the kernel level

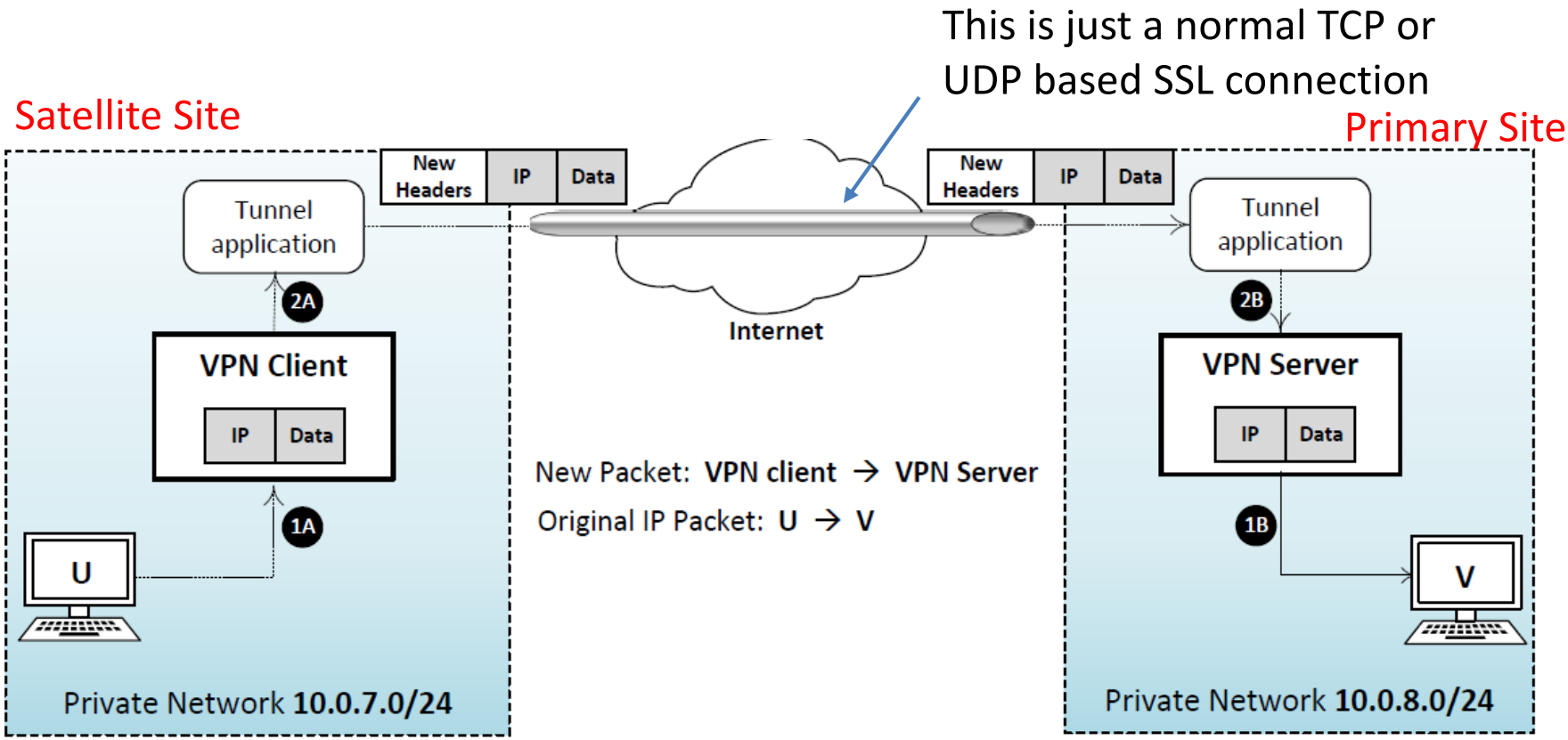


Two Types of IP Tunneling

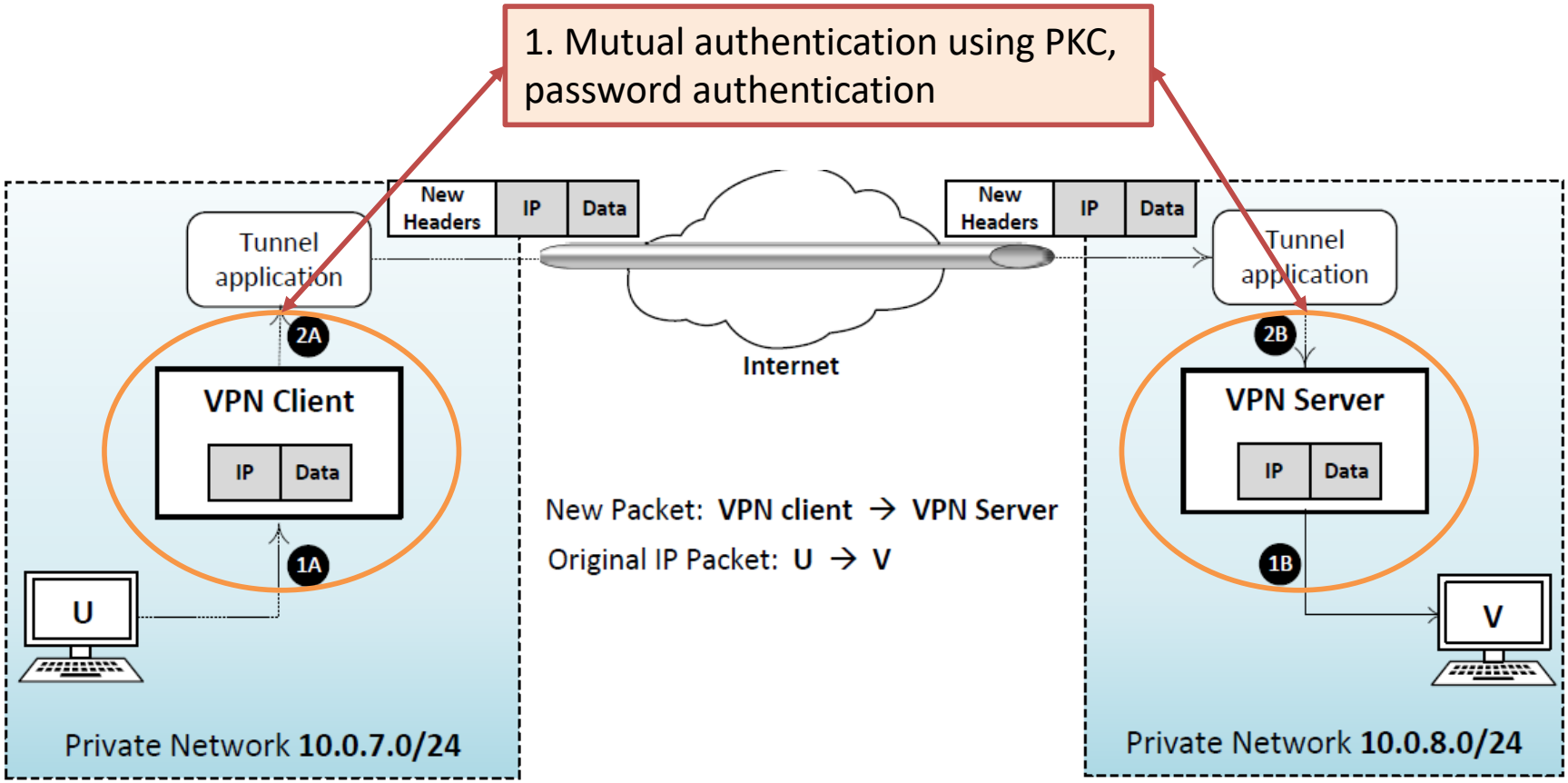
- TLS tunneling
  - It uses TLS library at the application layer to achieve tunneling.
  - The entire IP packet is encapsulated into a new TCP/UDP packet with a new header added.
  - Done at the application level



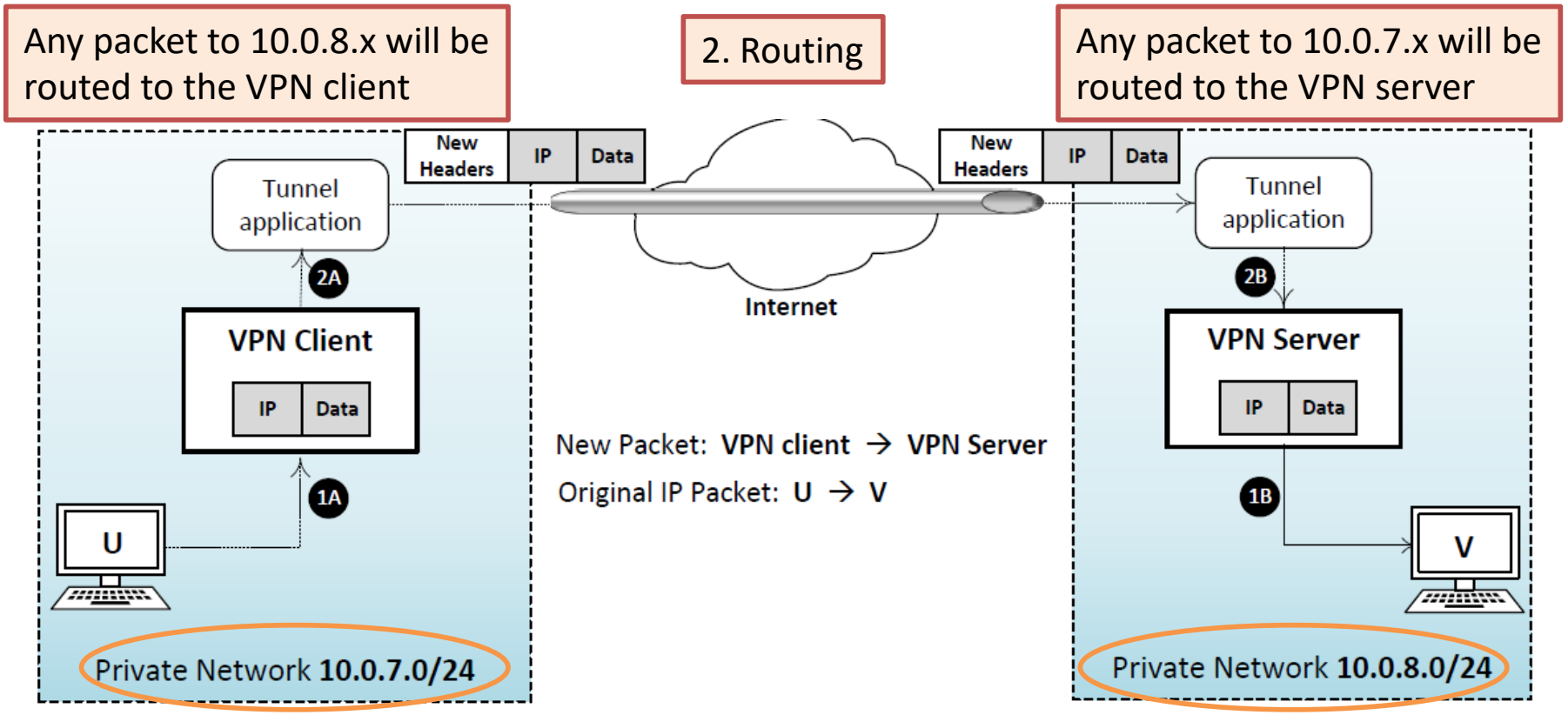
An Overview of How TLS/SSL VPN Works



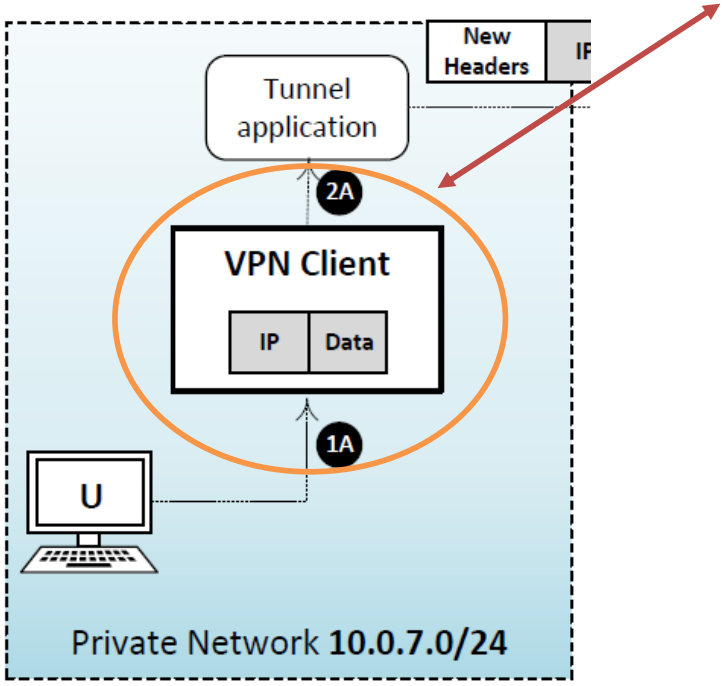
An Overview of How TLS/SSL VPN Works



An Overview of How TLS/SSL VPN Works



An Overview of How TLS/SSL VPN Works



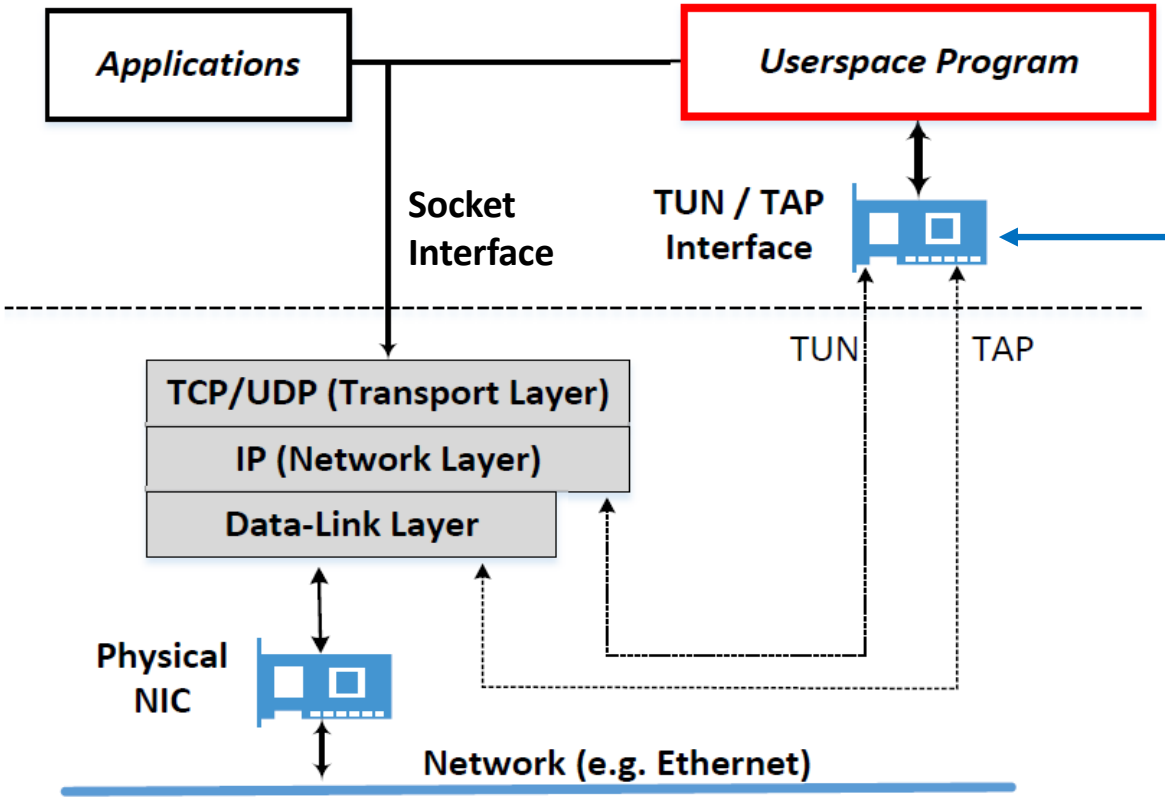
- Encapsulate the frame received in a TLS packet and directed to the VPN server
- Done in the application layer
- Not easily achieved

  Promiscuous mode, Raw packets, filtering
- Alternatively: Virtual Network Cards

Virtual Network Cards

- Most operating systems have two types of network interfaces:
  - Physical: Corresponds to the physical Network Interface Card (NIC)
  - Virtual: A virtualized representation of computer network interfaces that may or may not correspond directly to the NIC card. Example: *loopback* device
- TUN Virtual Interface
  - Work at OSI layer 3 or IP level
  - Sending any packet to TUN will result in the packet being delivered to user space program
- TAP Virtual Interfaces
  - Work at OSI layer 2 or Ethernet level
  - Used for providing virtual network adapters for multiple guest machines connecting to a physical device of the host machine

TUN/TAP Interface



- How can the Tunnel application get an IP packet?
  - Typically, applications interact with kernel using socket
  - Using socket, kernel only gives the data part of a packet to applications
  - Applications need to use a different way to interact with kernel

Creating a TUN Interface

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

ifr.ifr_flags = IFF_TUN | IFF_NO_PI;   ①

tunfd = open("/dev/net/tun", O_RDWR); ②
ioctl(tunfd, TUNSETIFF, &ifr);        ③
```

The flag IFF_TUN specifies that we are creating a TUN interface

Register a TUN device with the kernel

Configure the TUN Interface

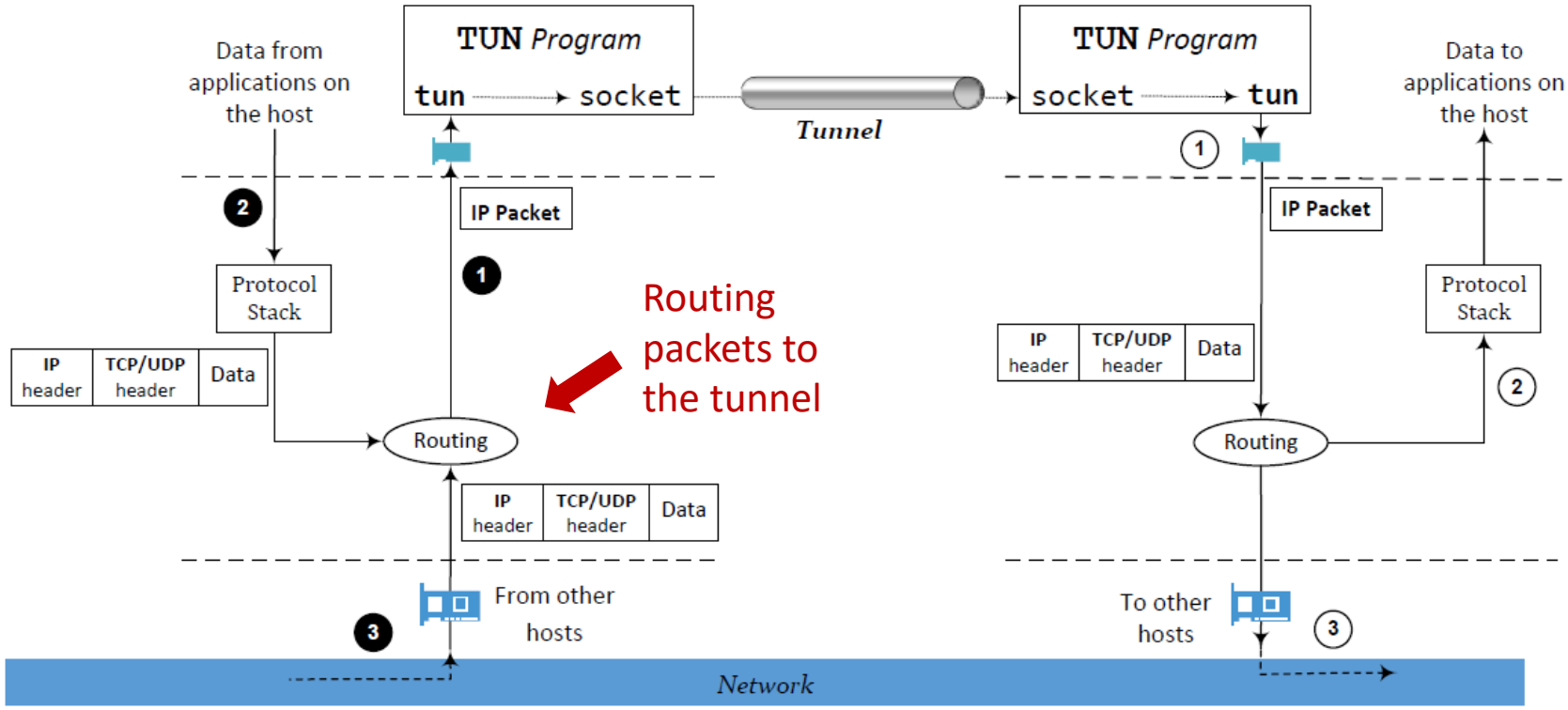
- Find the TUN interface

```
% ifconfig -a
tun0  Link encap:UNSPEC  HWaddr 00-00-00 ...
      POINTOPOINT NOARP MULTICAST  MTU:1500 ...
```

- Assign an IP address to the TUN interface and bring it up

```
% sudo ifconfig tun0 10.0.8.99/24 up

% ifconfig
tun0 Link encap:UNSPEC  HWaddr 00-00-00 ...
     inet addr: 10.0.8.99 P-t-P:10.0.8.99 Mask: 255.255.255.0
     UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500 ...
```

Set UP the Routing



Set UP the Routing

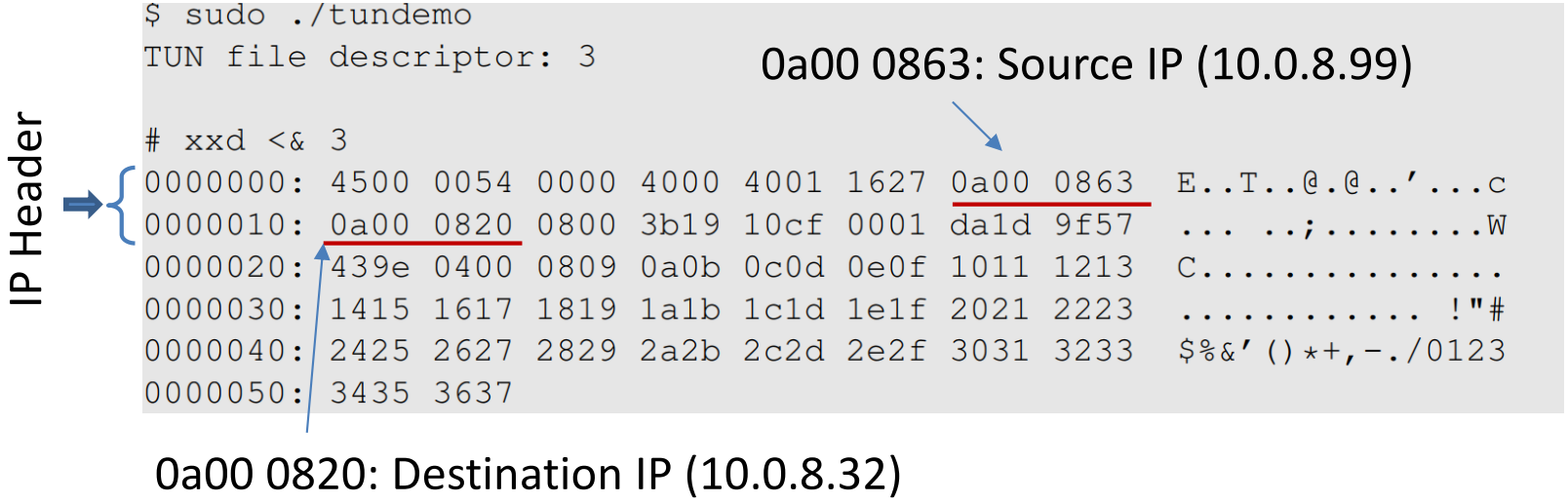
```
$ sudo route add -net 10.0.8.0/24 tun0
$ route -n
Destination  Gateway   Genmask         Flags Metric Ref Use Iface
0.0.0.0      10.0.2.1  0.0.0.0         UG    0      0   0 eth18
10.0.2.0     0.0.0.0   255.255.255.0   U     1      0   0 eth18
10.0.8.0     0.0.0.0   255.255.255.0   U     0      0   0 tun0
```

Packets to this destination should be routed to the `tun0` interface, i.e., they should go through the tunnel.

All other traffic will be routed to this interface, i.e., they will not go through the tunnel

Experiment: Reading From TUN Interface

We did an experiment by sending a ping packet to 10.0.8.32. The packet was sent to the TUN interface and then to our program. We use “xxd” to read from the interface and convert the into hexdump.



Experiment: Writing To TUN Interface

- We can write data to TUN interfaces.
- We can create a valid packet using the same “xxd” command.
- Copy-paste the xxd output from the previous slide into a file called “hexfile” and run “xxd –r hexfile > packetfile”.
- Now we write the packetfile to the interface:

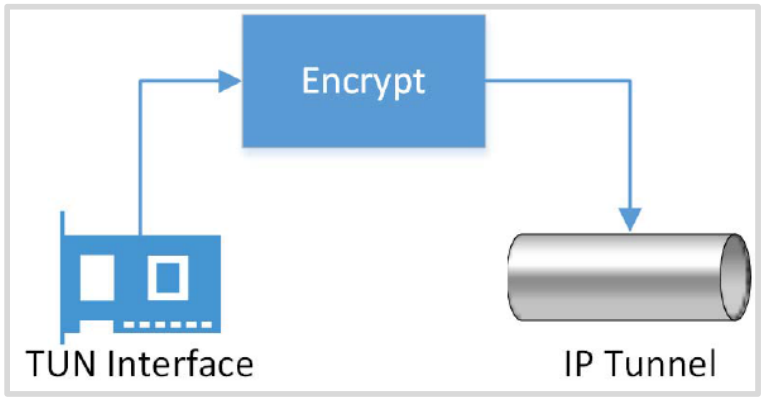
```
# cat packetfile >& 3
```

- We should be able to observe the packet using Wireshark.

Establish a Transport-Layer Tunnel

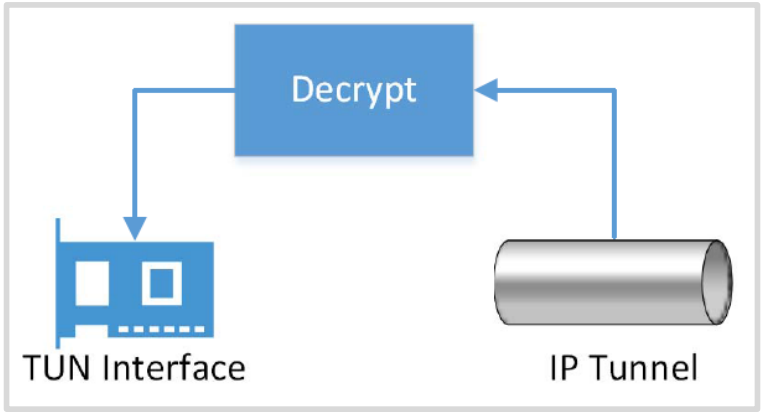
- A tunnel is just a TLS/SSL connection.
- Two applications (VPN client and server applications) just establish a TLS/SSL connection between themselves.
- Traffics inside are protected by TLS/SSL
- What makes this TLS/SSL connection a tunnel?
  - The payloads inside are IP packets
  - That is why it is called IP tunnel

How to Send/Receive Packets via Tunnel



Sending a packet via the tunnel

- Get an IP packet from the TUN interface
- Encrypt it (also add MAC)
- Send it as a payload to the other end of the tunnel

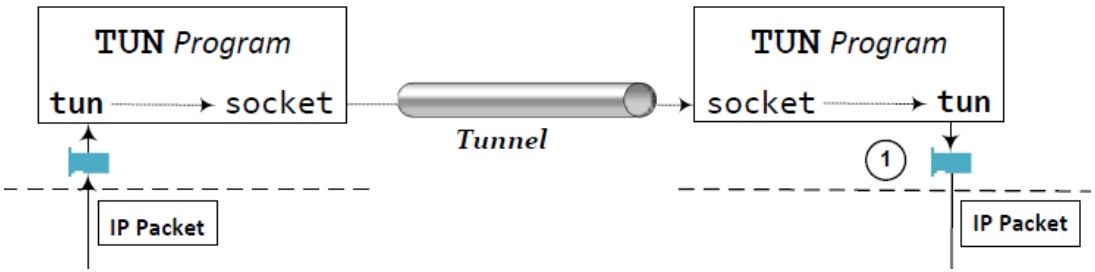


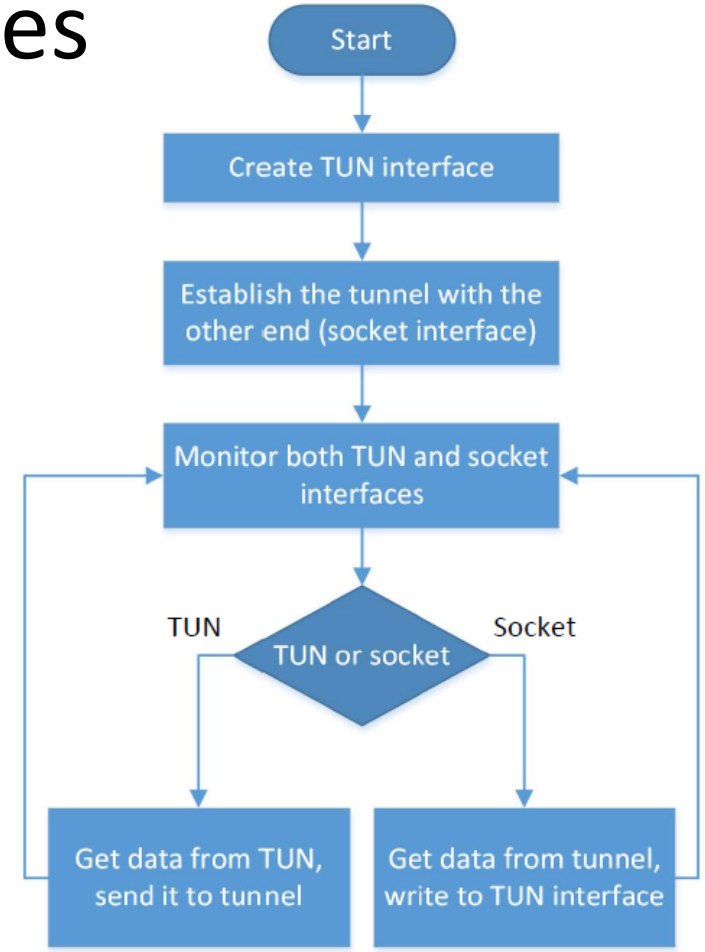
Receiving a packet from the tunnel

- Get a payload from the tunnel
- Decrypt it and verify its integrity
- We get the actual packet
- Write the packet to the TUN interface

Monitoring Both Interfaces

- Each tunnel application has two interfaces: socket and TUN
- Need to monitor both
- Forward packets between these two interfaces





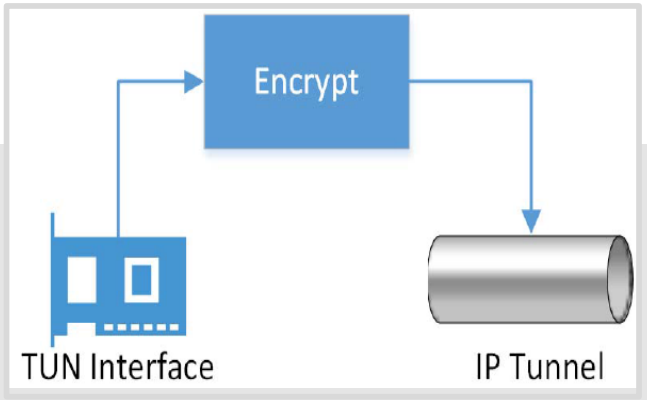
Implementation (Monitoring the 2 Interfaces)

```
int main (int argc, char * argv[]) {
   int tunfd, sockfd;

   tunfd  = createTunDevice();
   sockfd = connectToUDPServer();

   // Enter the main loop
   while (1) {
     fd_set readFDSet;

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

     if (FD_ISSET(tunfd,  &readFDSet)) tunSelected(tunfd, sockfd);
     if (FD_ISSET(sockfd, &readFDSet)) socketSelected(tunfd, sockfd);
   }
}
```

`select()` will be blocked until one of the interfaces has data.

Implementation (TUN → Socket)

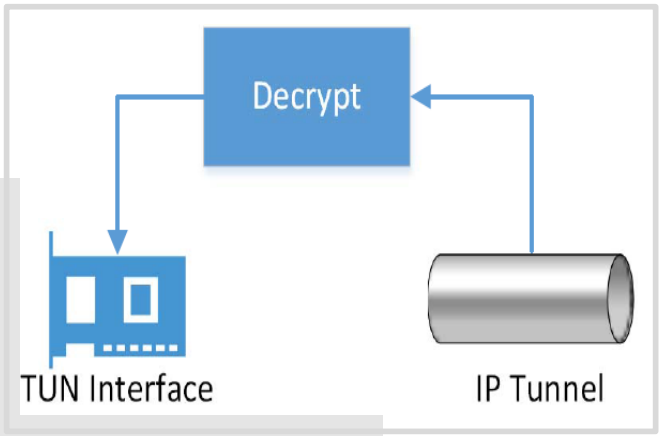
```
void tunSelected(int tunfd, int sockfd){
    int  len;
    char buff[BUFF_SIZE];

    printf("Got a packet from TUN\n");

    bzero(buff, BUFF_SIZE);
    len = read(tunfd, buff, BUFF_SIZE);
    sendto(sockfd, buff, len, 0, (struct sockaddr *) &peerAddr,
                  sizeof(peerAddr));
}
```



Note: the encryption step is omitted from the code (for the sake of simplicity)

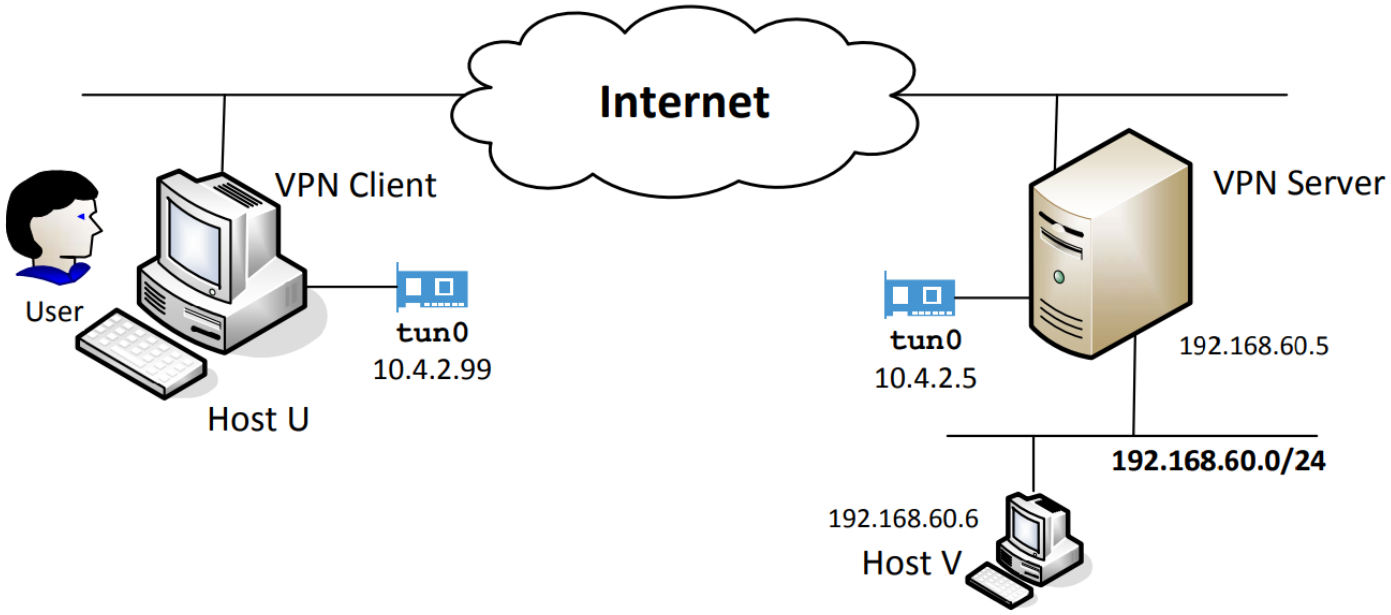
Implementation (Socket → TUN)

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

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

    bzero(buff, BUFF_SIZE);
    len = recvfrom(sockfd, buff, BUFF_SIZE, 0, NULL, NULL);
    write(tunfd, buff, len);
}
```

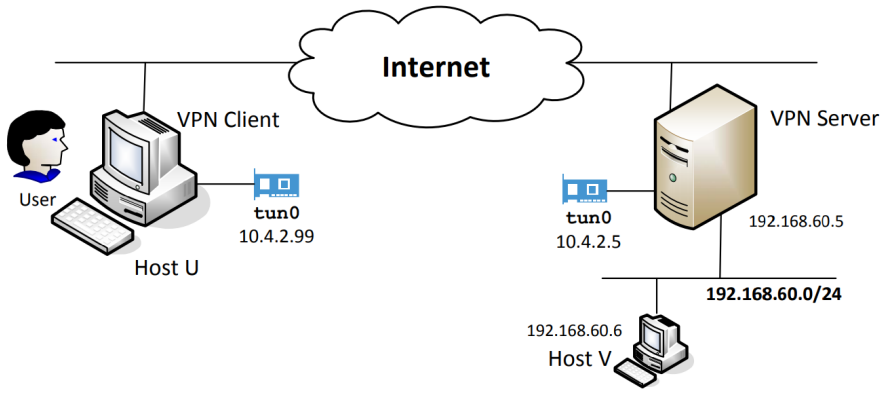


Note: the decryption step is omitted from the code (for the sake of simplicity)

Case Study: Configuring a VPN



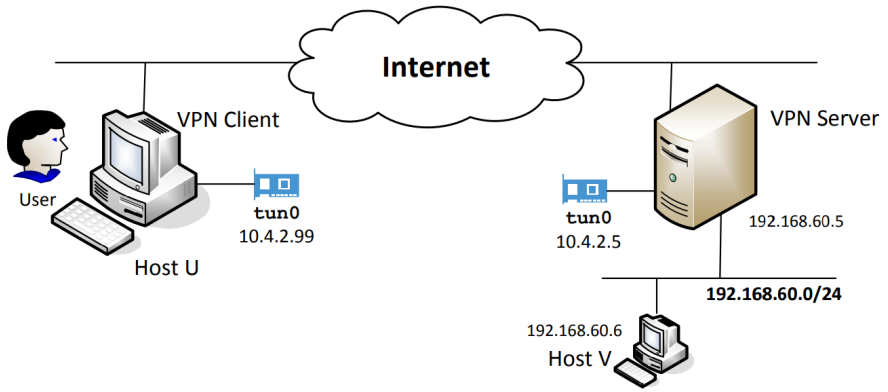
Configure VPN Server



- On VPN Server, we first run the server program.
- Configure the tun0 interface.
  - We use 10.4.2.0/24 as IP prefix for the TUN interface (for both VPN Client and VPN Server)
- The following two commands assign the IP address to the tun0, bring it up and then add a corresponding route to routing table.

```
$ sudo ifconfig tun0 10.4.2.5/24 up
$ sudo route add -net 10.4.2.0/24 tun0
```

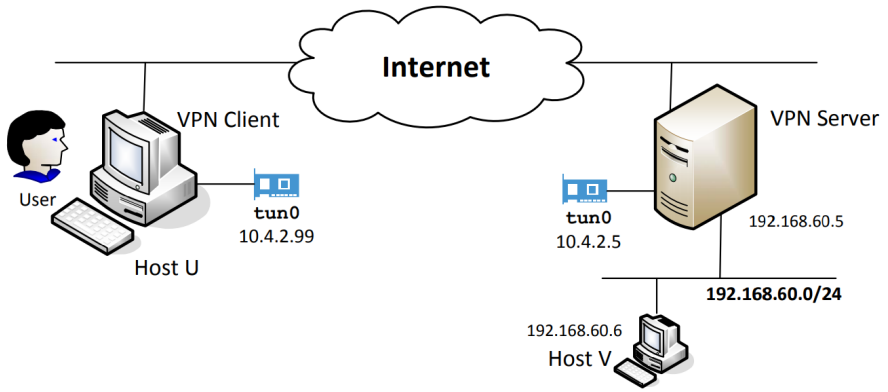
Configure VPN Client



- On VPN Client, we first run the client program.
- Add route for the 10.4.2.0/24 network.
- Add a route, so that all the packets for 192.168.60.0/24 are routed to the tun0 interface.

```
$ sudo ifconfig tun0 10.4.2.99/24 up
$ sudo route add -net 10.4.2.0/24 tun0
$ sudo route add -net 192.168.60.0/24 tun0
```

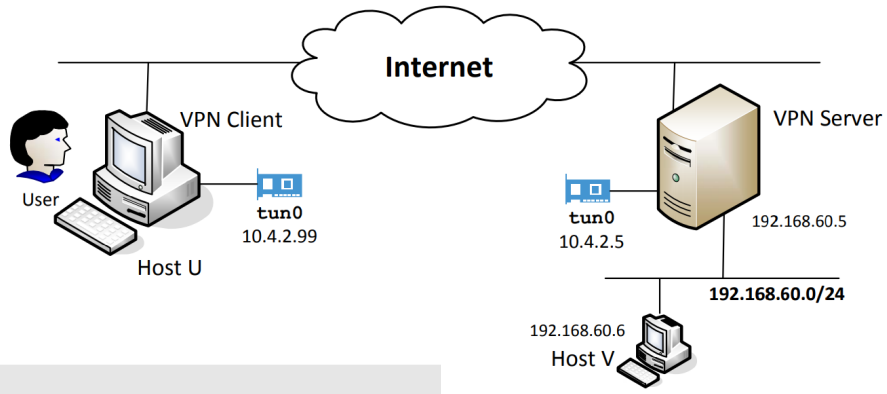
Configure Host V



- The reply packets should go back via the same VPN tunnel, so that they are protected.
- To ensure that, route all packets for the 10.4.2.0/24 network toward the tunnel.
- For Host V, we route such packets to VPN Server.
- Add the following routing entry to Host V:

```
$ sudo route add -net 10.4.2.0/24 gw 192.168.60.5 eth1
```

Testing VPN: `ping` Testing



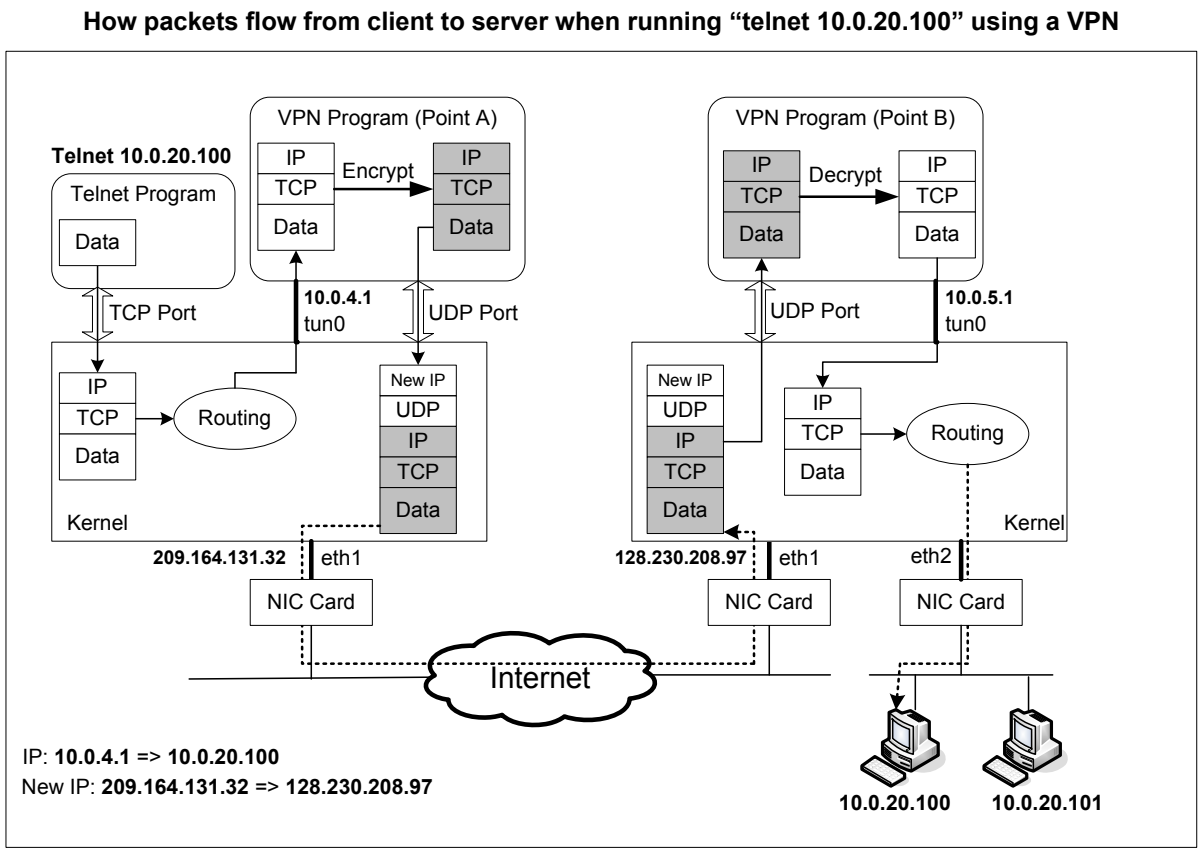
- Ping Host V from Host U and we see the following result:

```
seed@User(10.0.2.6):$ ping 192.168.60.6
PING 192.168.60.6 (192.168.60.6) 56(84) bytes of data.
64 bytes from 192.168.60.6: icmp_req=1 ttl=63 time=2.41 ms
64 bytes from 192.168.60.6: icmp_req=2 ttl=63 time=1.48 ms
```

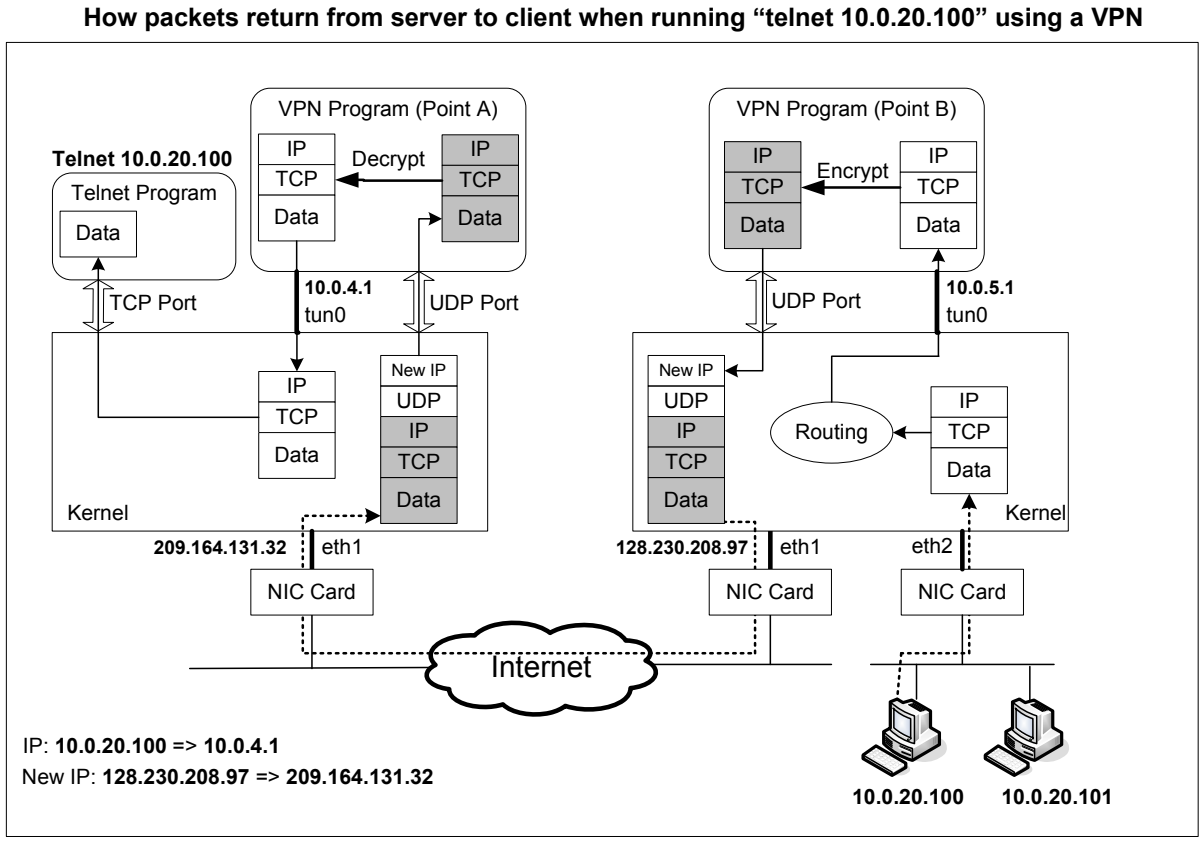
- The following figure shows the packets generated when we ping Host V (192.168.0.6).

No.	Source	Destination	Protocol	Length	Info
1	10.4.2.99	192.168.60.6	ICMP	100	Echo (ping) request id=0x0e85, seq=1/256, ttl=64
2	10.0.2.6	10.0.2.5	UDP	128	Source port: 59793 Destination port: 55555
3	10.0.2.5	10.0.2.6	UDP	128	Source port: 55555 Destination port: 59793
4	192.168.60.6	10.4.2.99	ICMP	100	Echo (ping) reply id=0x0e85, seq=1/256, ttl=63
5	10.4.2.99	192.168.60.6	ICMP	100	Echo (ping) request id=0x0e85, seq=2/512, ttl=64
6	10.0.2.6	10.0.2.5	UDP	128	Source port: 59793 Destination port: 55555
7	10.0.2.5	10.0.2.6	UDP	128	Source port: 55555 Destination port: 59793
8	192.168.60.6	10.4.2.99	ICMP	100	Echo (ping) reply id=0x0e85, seq=2/512, ttl=63

Packet Flow from Telnet Client to Server

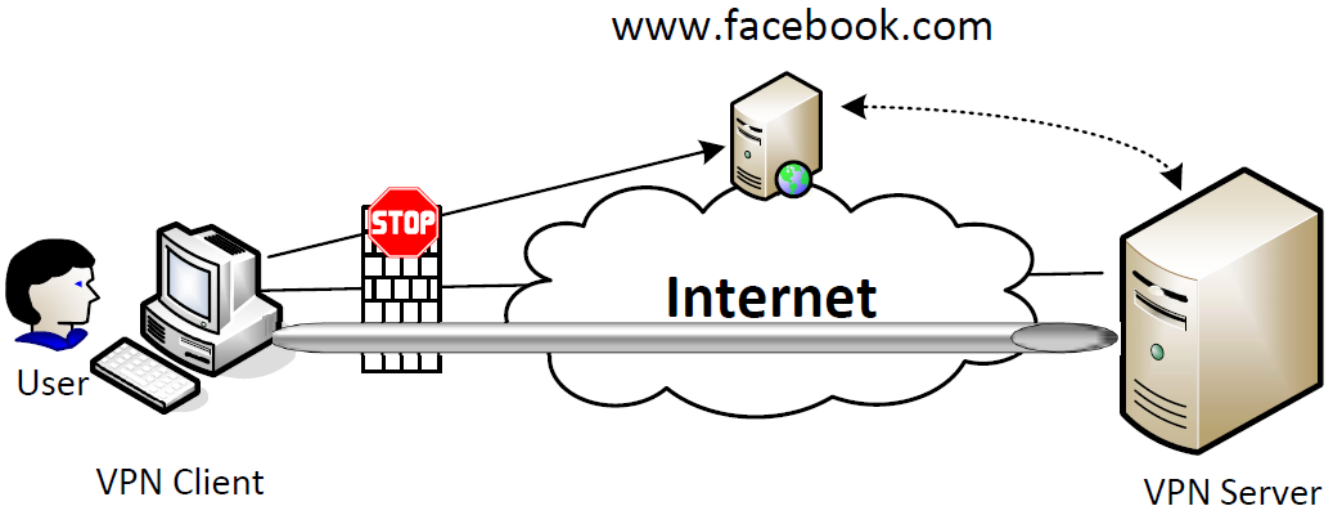


Packet Flow from Telnet Server to Client



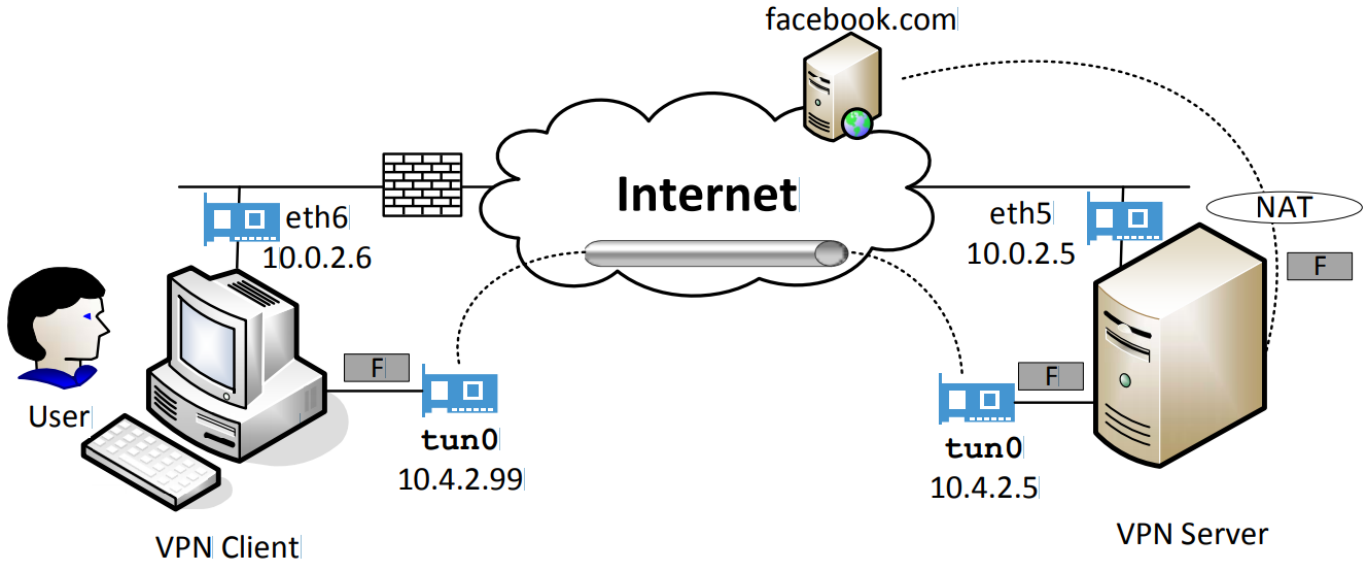
Bypassing Firewalls using VPN

Bypassing Firewall using VPN: the Main Idea



- Send our Facebook-bound packets to the TUN interface towards VPN server
- VPN server will release our Facebook-bound packets to the Internet
- Facebook's reply packets will be routed to the VPN server (question: why?)
- VPN server sends the reply packets back to us via the tunnel

Experiment: Network Setup



Setting UP Firewall

- Setup firewall to block User from accessing Facebook
- We run the following command to get the list of IP prefixes owned by Facebook:

```
$ whois -h whois.radb.net -- '-i origin AS32934'
```

- We can also get IP addresses returned by Facebook's DNS server by running the following command (this IP address can change):
  `dig www.facebook.com`

Blocking Facebook

One of the IP prefixes belong to Facebook

```
$ sudo ufw enable
$ sudo ufw deny out on eth6 to 31.13.0.0/16
$ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
31.13.0.0/16               DENY OUT    Anywhere on eth6
```

Facebook becomes unreachable

```
seed@User(10.0.2.6):~$ ping www.facebook.com
PING star-mini.c10r.facebook.com (31.13.71.36) 56(84) bytes of data.
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
```

Bypassing the Firewall

- We add a routing entry to the user machine, changing the route for all Facebook traffic. Instead of going through eth6, we use the TUN interface:

```
$ sudo route add -net 31.13.0.0/24 tun0
```

- The Facebook-bound packets are going through our tunnel.
- The Facebook-bound packets are hidden inside a packet going to the VPN server, so it does not get blocked.
- VPN server will release the packet to the Internet.
- Replies from Facebook will come back to VPN server, which will forward it back to us via the tunnel.

Summary

- What is VPN?
- IP tunneling
- IP tunneling using TLS/SSL
  - TUN/TAP interface
- Building a VPN using TUN/TAP interface
- Using VPN to bypass firewalls