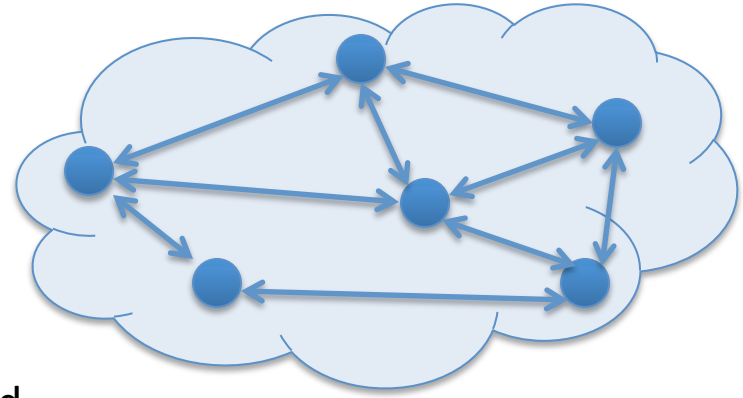
Virtual Private Networks

Chester Rebeiro
IIT Madras

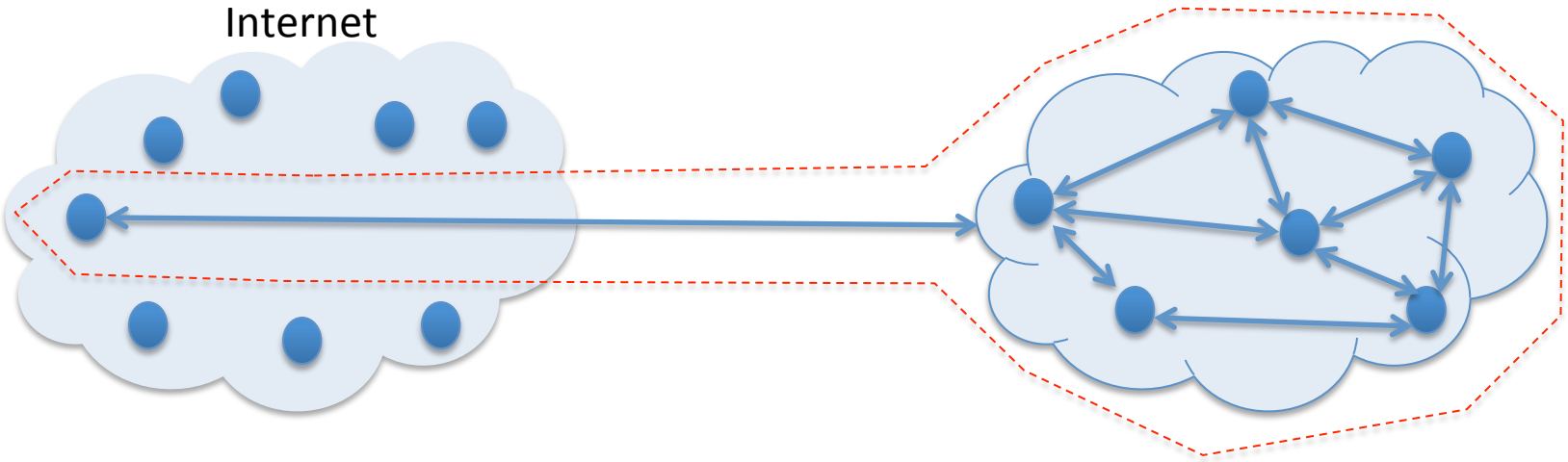
Private Networks



Physically disconnected from the outside Internet. Three properties:

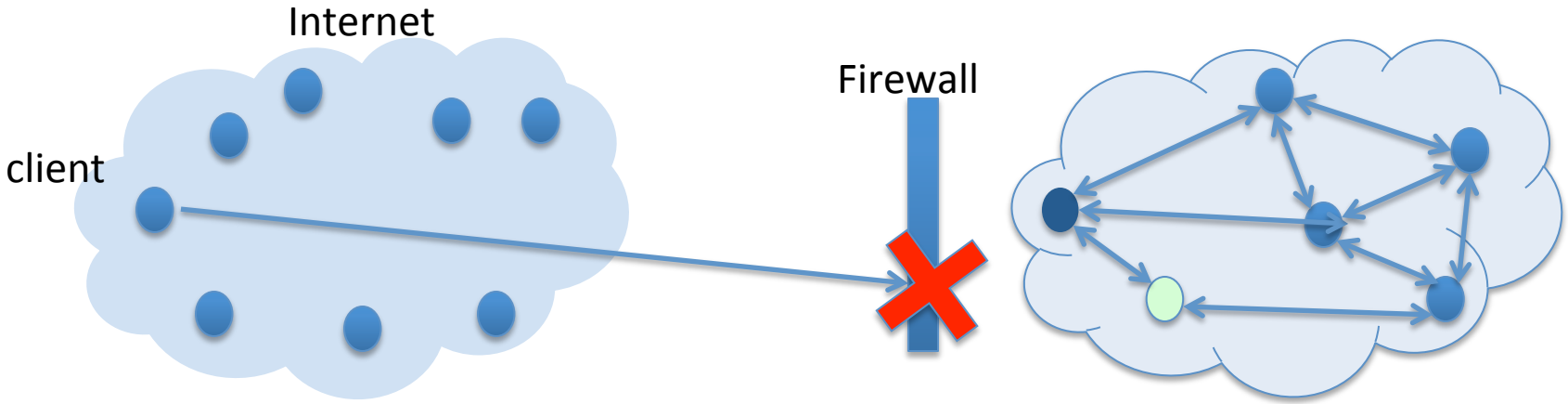
- **Users Authenticated.**
  Users are authorized and their identities verified.
- **Content Protected.**
  Communication within the private network cannot be sniffed from outside.
  cables are physically secured
- **Integrity Preserved.**
  Nobody from outside the network can spoof

Virtual Private Networks



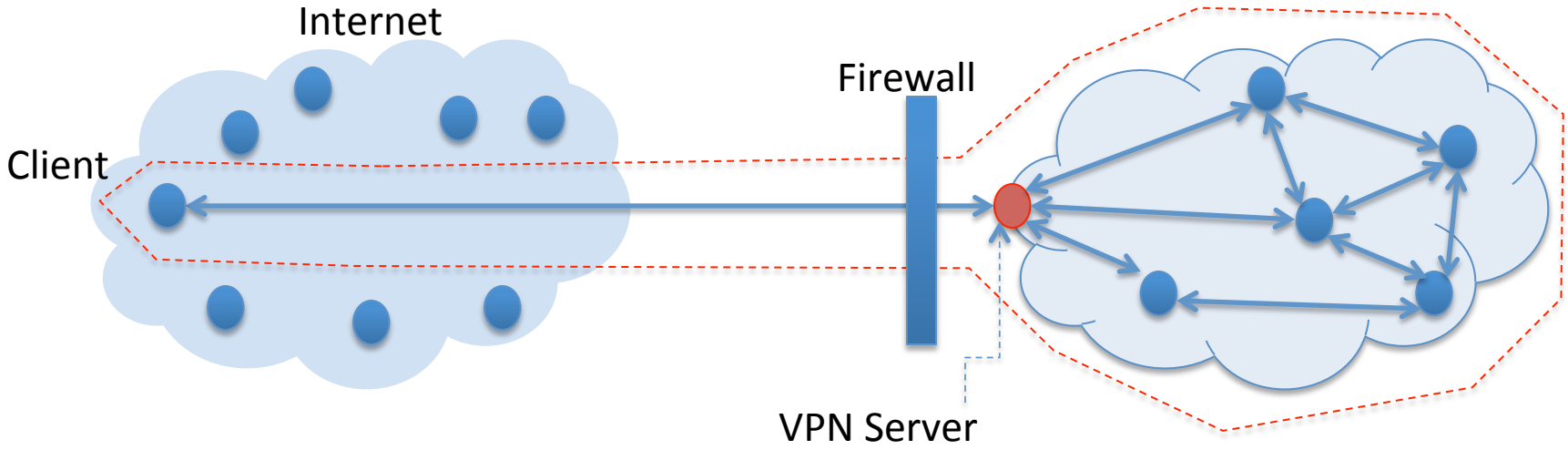
Able to achieve: Users Authentication, Content Protection, and Integrity Preserved without being physically located

Virtual Private Networks



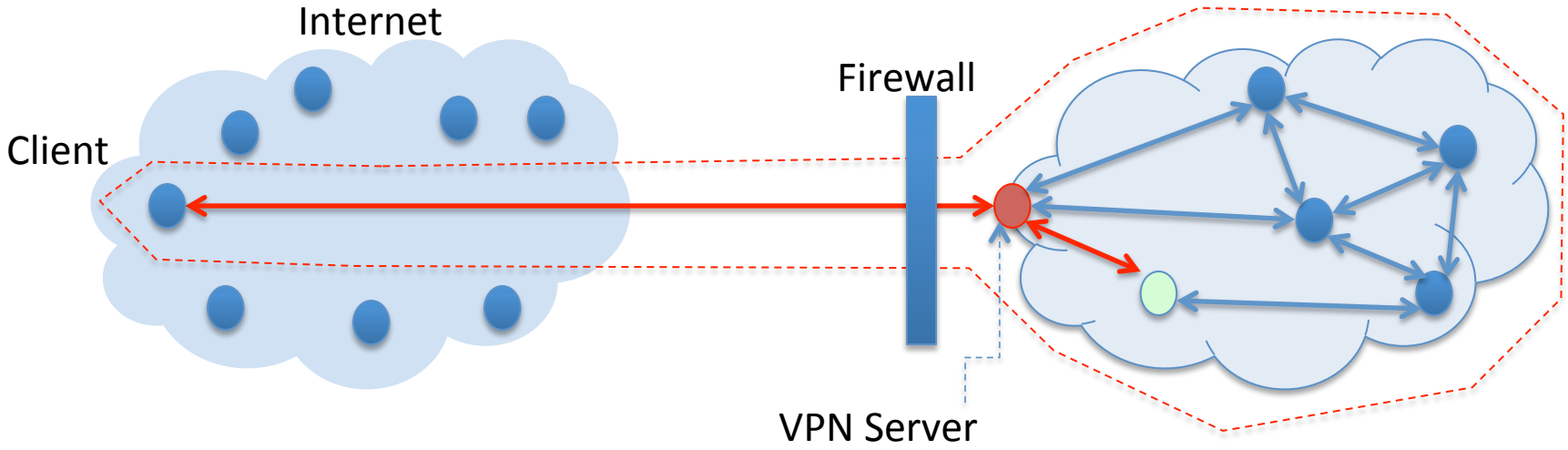
Any attempt to directly connect to a computer inside the private network will be stopped by the firewall. Moreover, the IP address may not be valid.

Virtual Private Networks



VPN Server: exposed to the outside network.
Outside computers will be authenticated by the VPN server. Once authenticated, a secure channel is established between the VPN server and client, so packets are encrypted and integrity preserved.

Virtual Private Networks

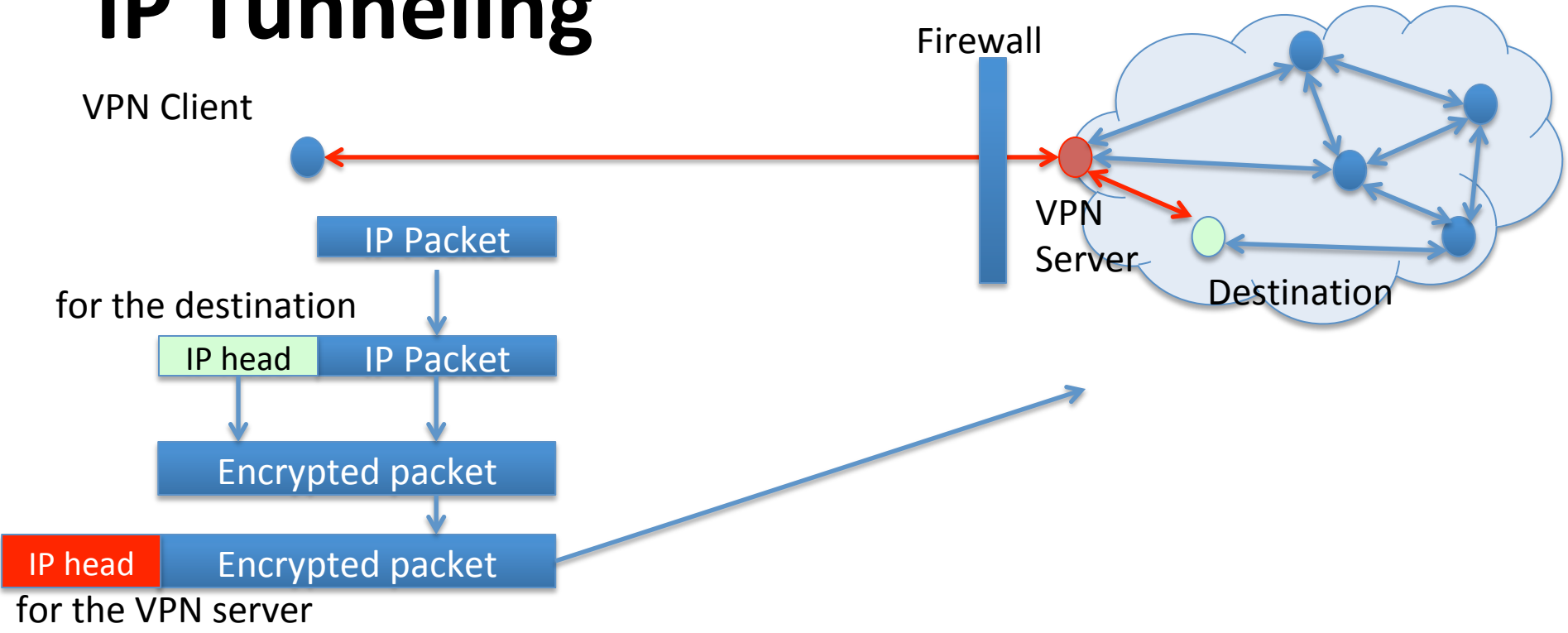


Only way to connect to a system in the private network is via the VPN server.
Needs to be Transparent. The VPN client should be ignorant that it is a remote client.

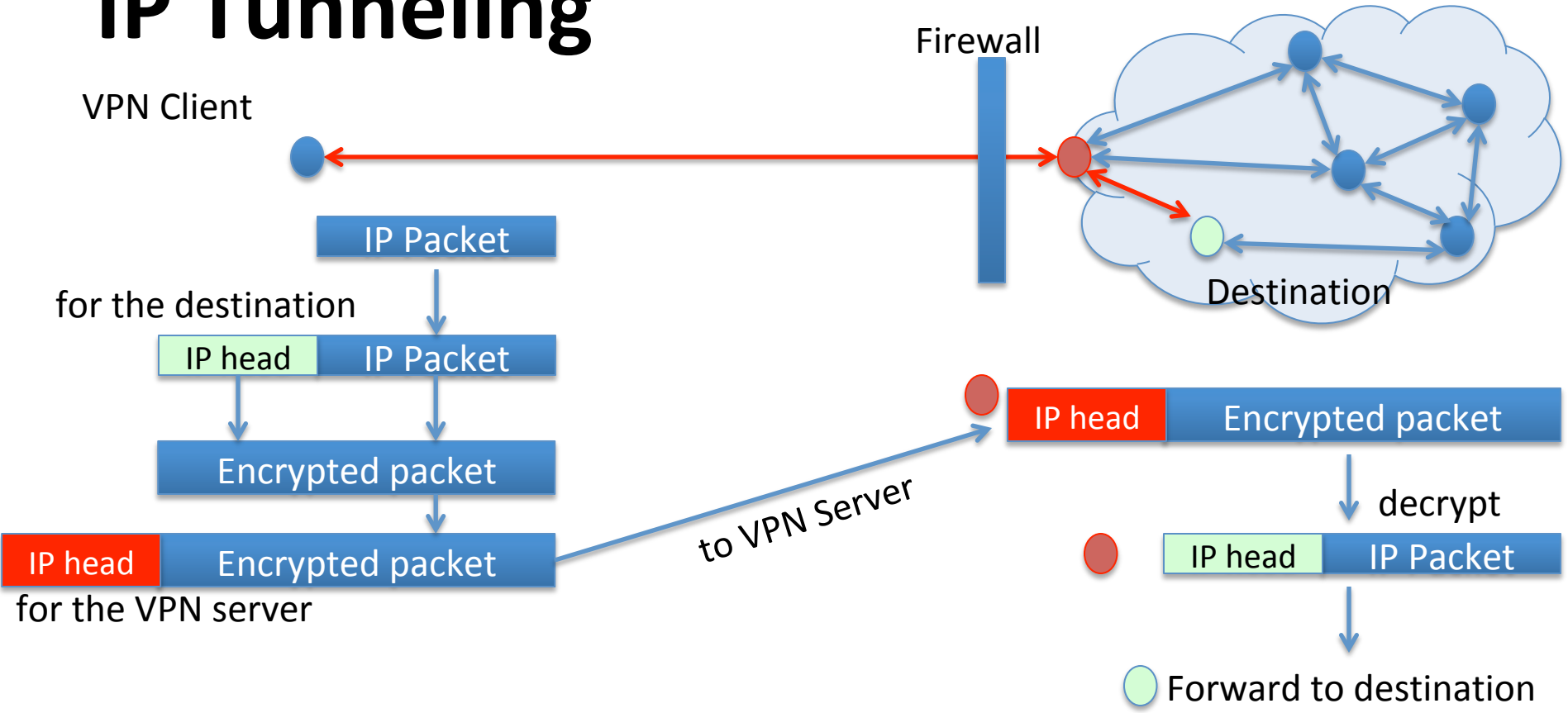
VPN vs Application Level Security

- This is different from a regular application security, where TLS can be used.
  - IP spoofing / sniffing can be done
  - Client needs to open and initiate a TLS connection, thus no transparency
- For VPN, the IP headers need to be encrypted
  - However, traffic cannot be routed

IP Tunneling

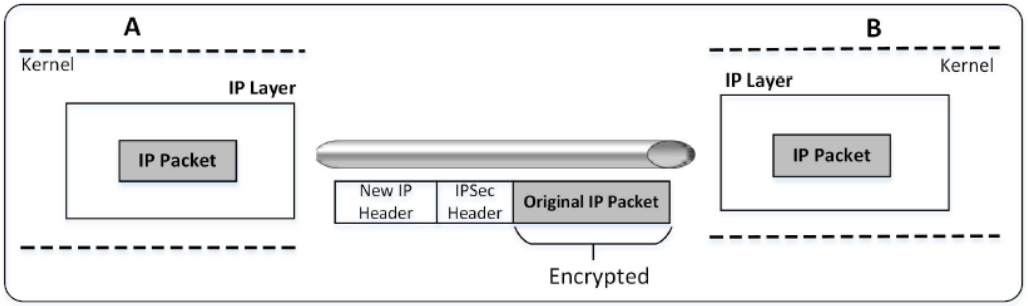


IP Tunneling



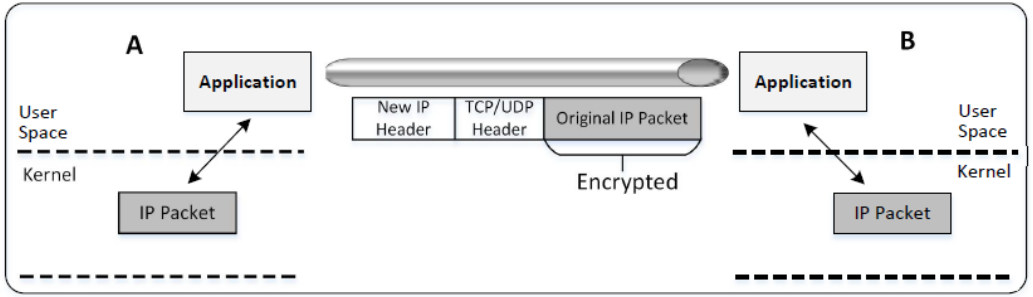
IP Tunneling

- Two ways of achieving IP Tunneling
  - **IPSec tunneling:** uses IP Sec 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

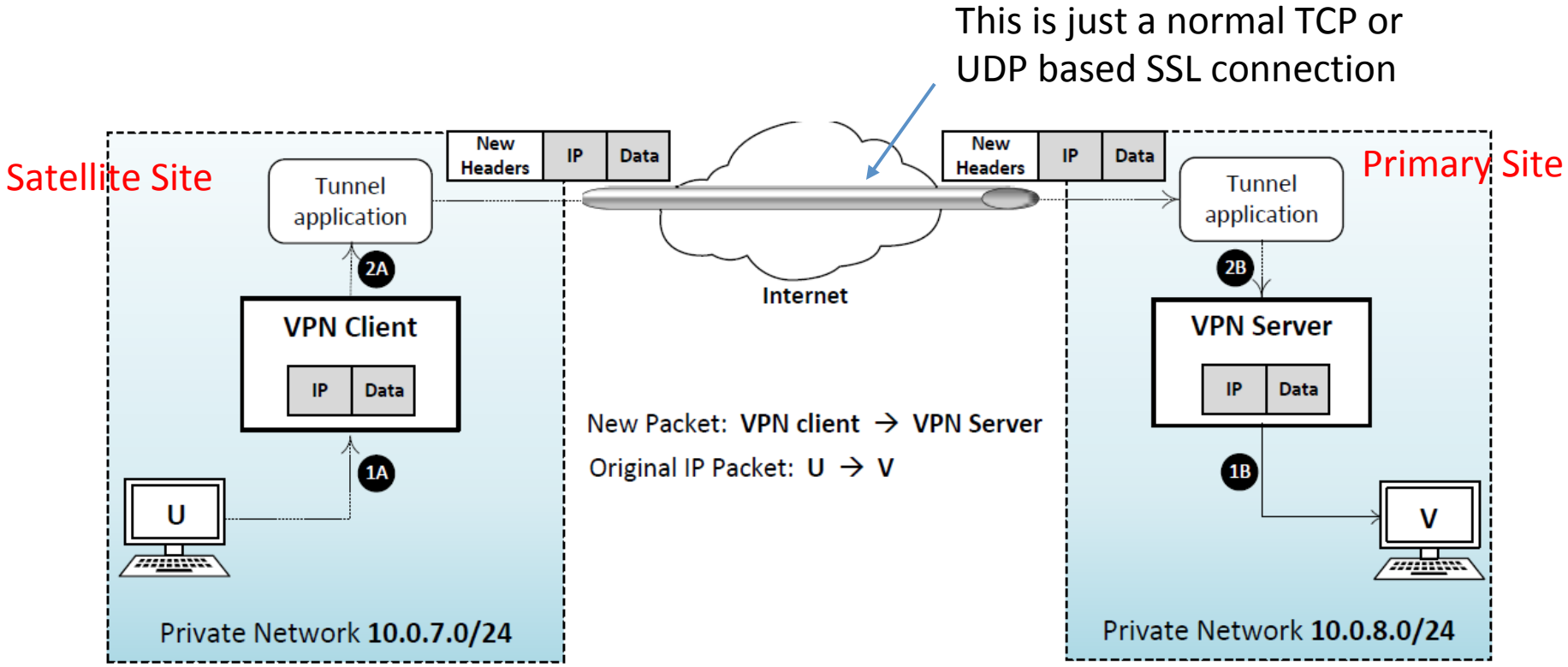


IP Tunneling

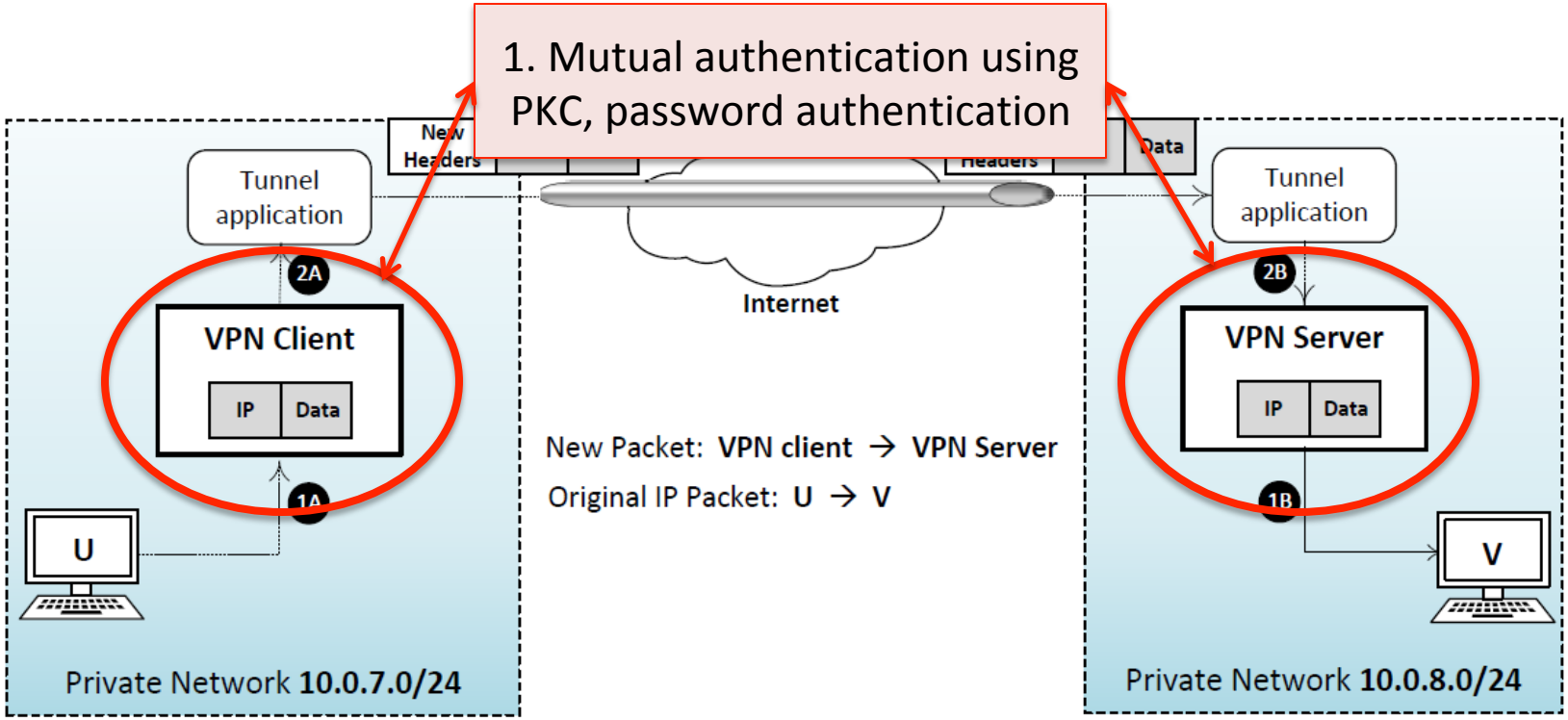
- Two ways of achieving IP Tunneling
  - **TLS tunneling:** uses TLS library at the application layer to achieve tunneling
  - The entire IP packet is encapsulated into a new TCP/UDP packet with a new header added
  - Done at the application level



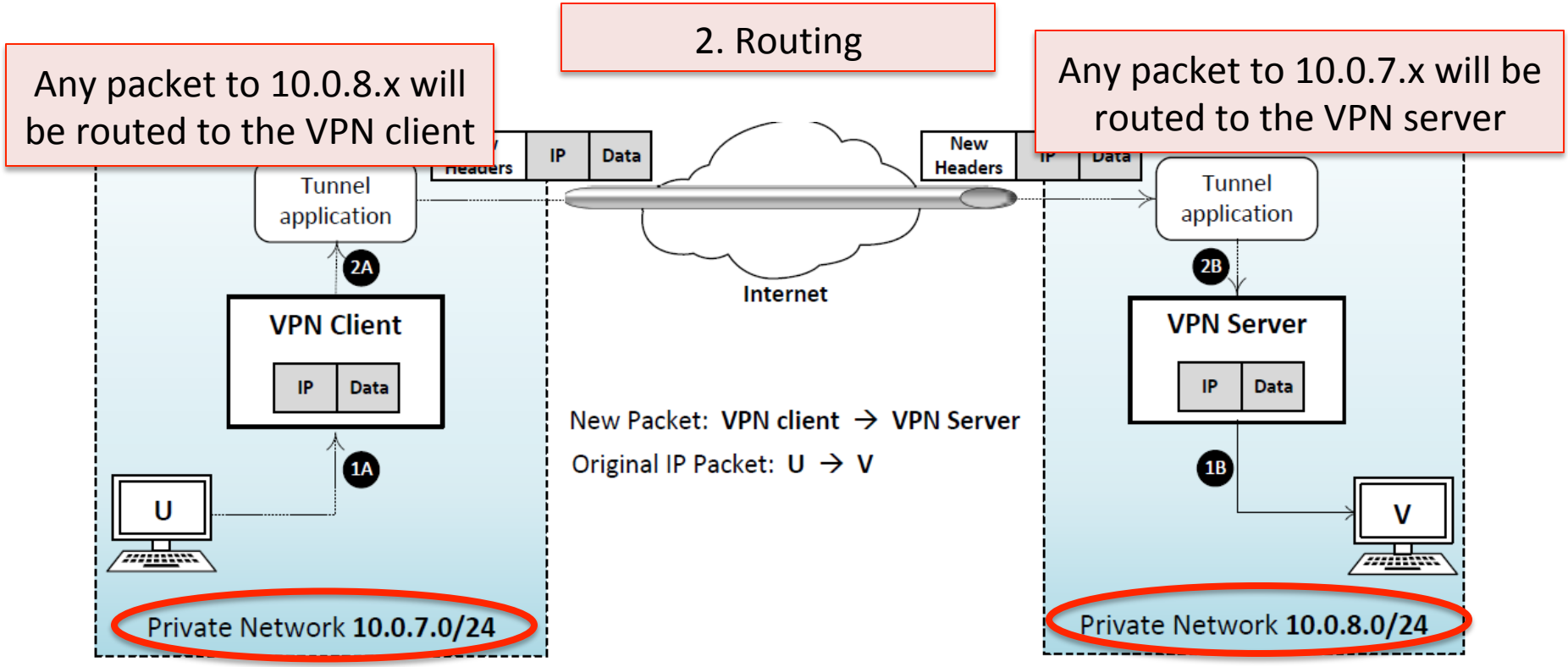
An Overview of How TLS/SSL VPN Works



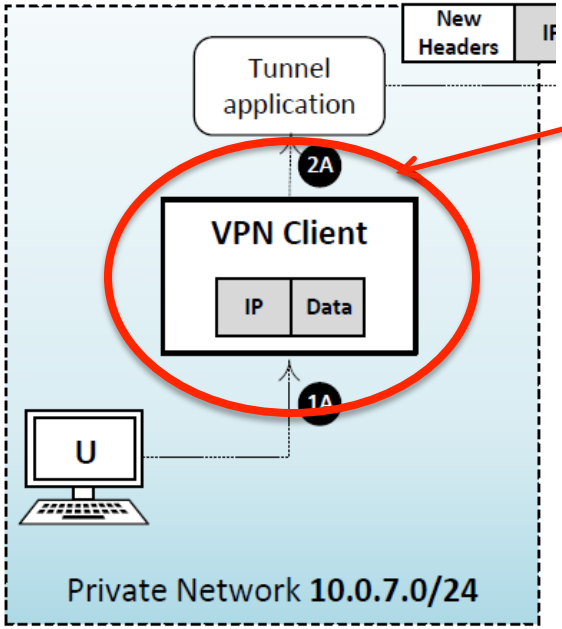
An Overview of How TLS/SSL VPN Works



An Overview of How TLS/SSL VPN Works



An Overview of How TLS/SSL VPN Works



Needs to encapsulate the frame received in a TLS packet and directed to the VPN server.

Needs to be done in the application layer.

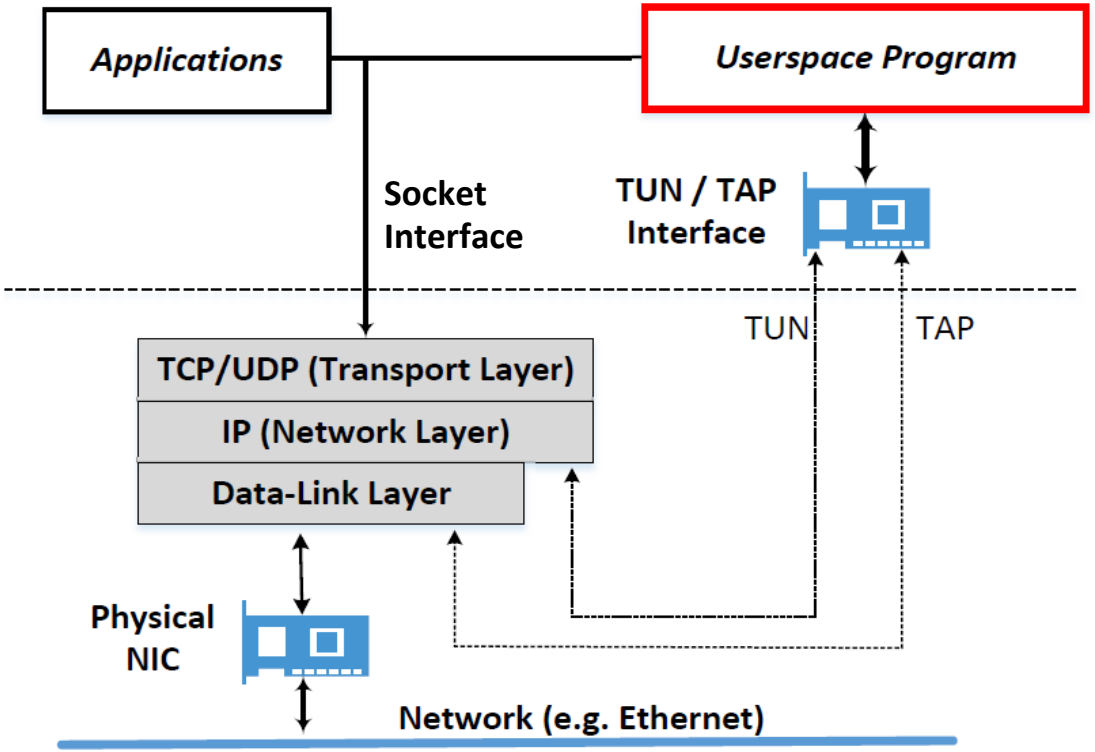
Not easily achieved.
Promiscuous mode, Raw packets, filtering

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: It is 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 Interfaces



Creating a TUN Interface

```c
int main () {
   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);

   printf("TUN file descriptor: %d \n", tunfd);
   // We can interact with the device using this file descriptor.
   // In our experiement, we will do the interaction from a shell.
   // Therefore, we launch the bash shell here.
   execve("/bin/bash", NULL,NULL);

   return 0;
}
```

The flag IFF_TUN specifies that we are creating a TUN interface

Register a TUN device with the kernel

Needs CAP_NET_ADMIN

Configure the TUN Interface

Find the TUN interface

```
root@optiplex:/home/chester/Desktop/netsec/vpn# ifconfig -a
```

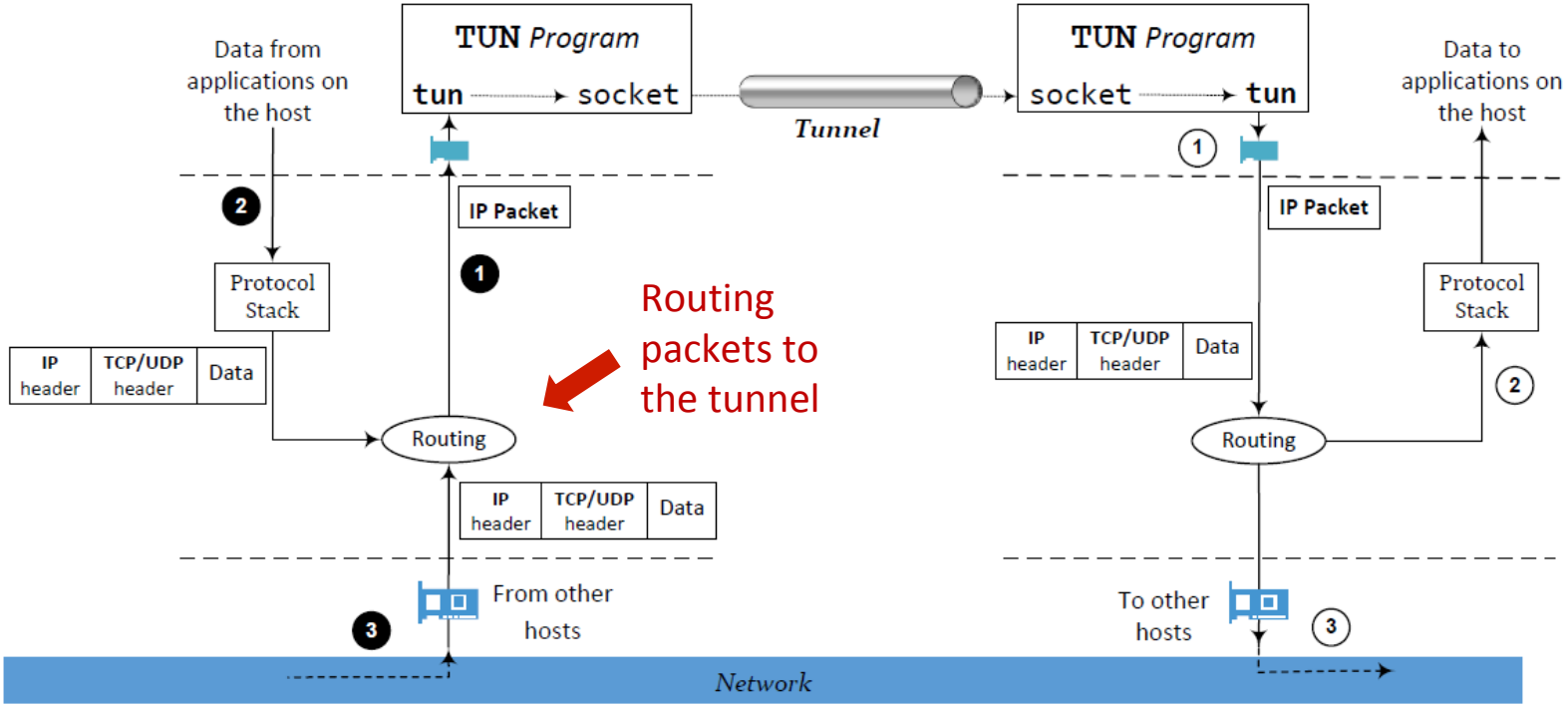
```
tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-0
-00
          POINTOPOINT NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Configure the TUN Interface

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

```
[</Desktop/netsec/vpn# sudo ifconfig tun0 10.0.8.99/24
[root@optiplex:/home/chester/Desktop/netsec/vpn# ifconfig

tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-0(
-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  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Set UP the Routing



Setup the Routing

```
<op/netsec/vpn# sudo route add -net 10.0.8.0/24 tun0

root@optiplex:/home/chester/Desktop/netsec/vpn# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.21.239.254   0.0.0.0         UG    100    0        0 eth0
10.0.8.0        0.0.0.0         255.255.255.0   U     0      0        0 tun0
10.0.8.0        0.0.0.0         255.255.255.0   U     0      0        0 tun0
10.21.224.0     0.0.0.0         255.255.240.0   U     100    0        0 eth0
10.24.4.7       10.21.239.254   255.255.255.255 UGH   100    0        0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 eth0
```

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

Ping to the TUN interface

```
chester@optiplex:~$ ping 10.0.8.99
PING 10.0.8.99 (10.0.8.99) 56(84) bytes of data.
64 bytes from 10.0.8.99: icmp_seq=1 ttl=64 time=0.027 ms
64 bytes from 10.0.8.99: icmp_seq=2 ttl=64 time=0.035 ms
64 bytes from 10.0.8.99: icmp_seq=3 ttl=64 time=0.045 ms
64 bytes from 10.0.8.99: icmp_seq=4 ttl=64 time=0.048 ms
```

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.

IP Header

```
root@optiplex:/home/chester/Desktop/netsec/vpn# xxd <&3
00000000: 4500 0054 e4dc 4000 4001 3149 0a00 0863  E..T..@.@.1I...c
00000010: 0a00 0821 0800 612d 0710 0001 9703 be5c  ...!..a-.......\
00000020: 0000 0000 798e 0200 0000 0000 1011 1213  ....y...........
00000030: 1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  ............ !"#
00000040: 2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
```

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:

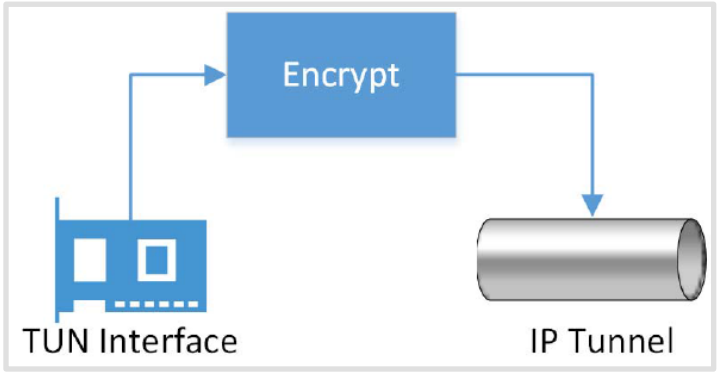
```
# cat packetfile >& 3
```

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

Establish a Transport-Layer Tunnel

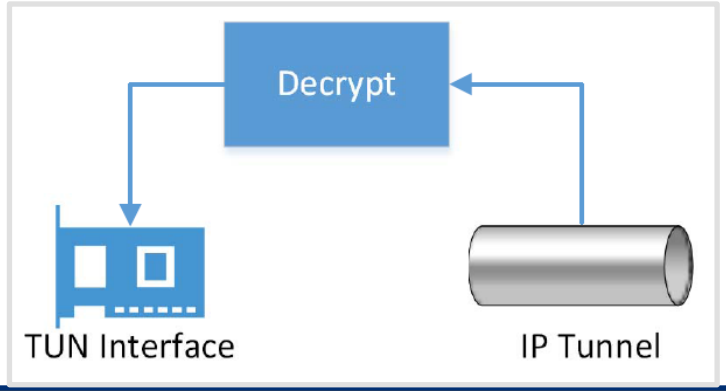
- A tunnel is just a TLS/SSL connection.
- Two applications (VPN client and server applications) just establish a TLS/SSL connection between themselves.
- Traffic inside are protected by TLS/SSL
- What makes this TLS/SSL connection a tunnel?
  - The payloads inside are IP packets
  - That is why it is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

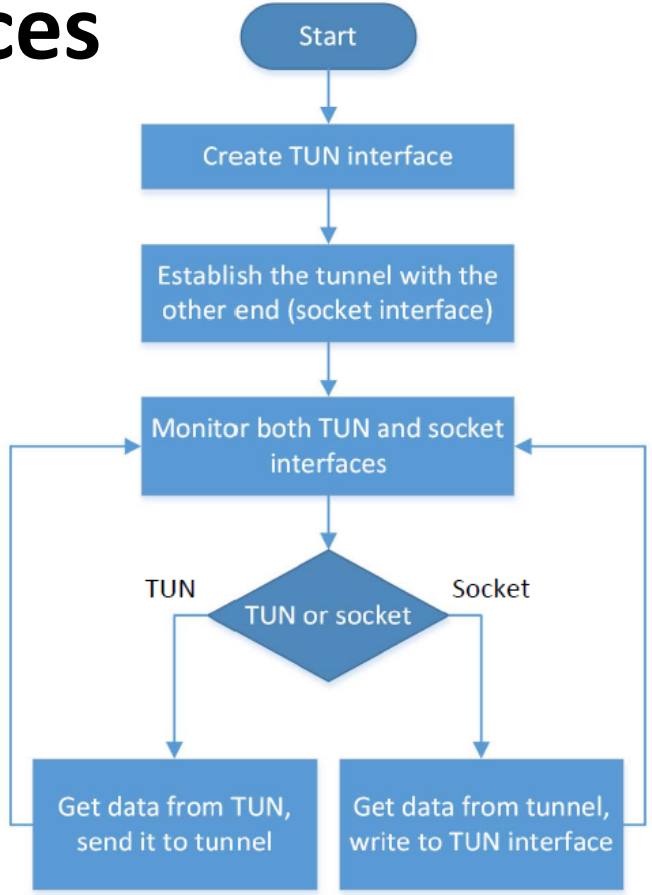


Receiving a packet from the tunnel

- 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



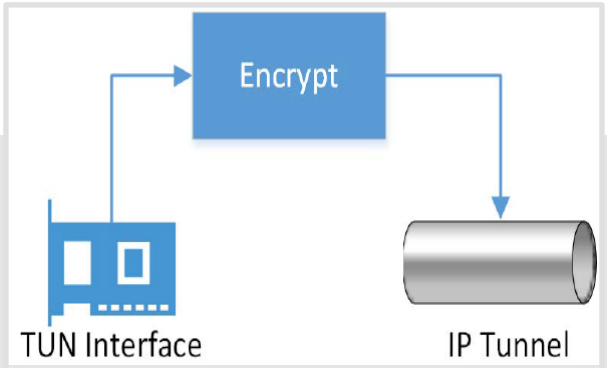
Implementation (Monitoring the 2 Interfaces)

```
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)

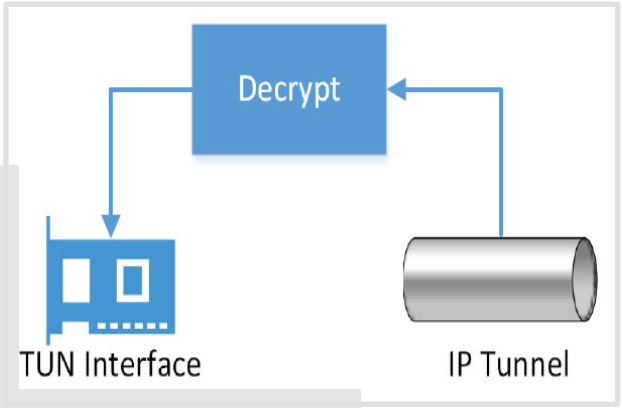
```
void tunSelected(int tunfd, int sockfd){
    int  len;
    char buff[BUFF_SIZE];

    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)

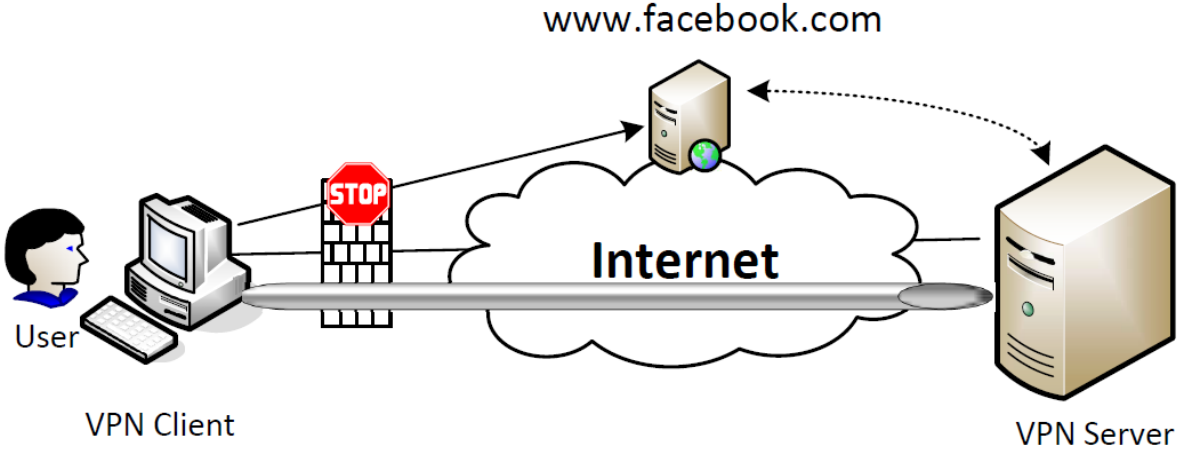
```
void socketSelected (int tunfd, int sockfd){
    int  len;
    char buff[BUFF_SIZE];

    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)

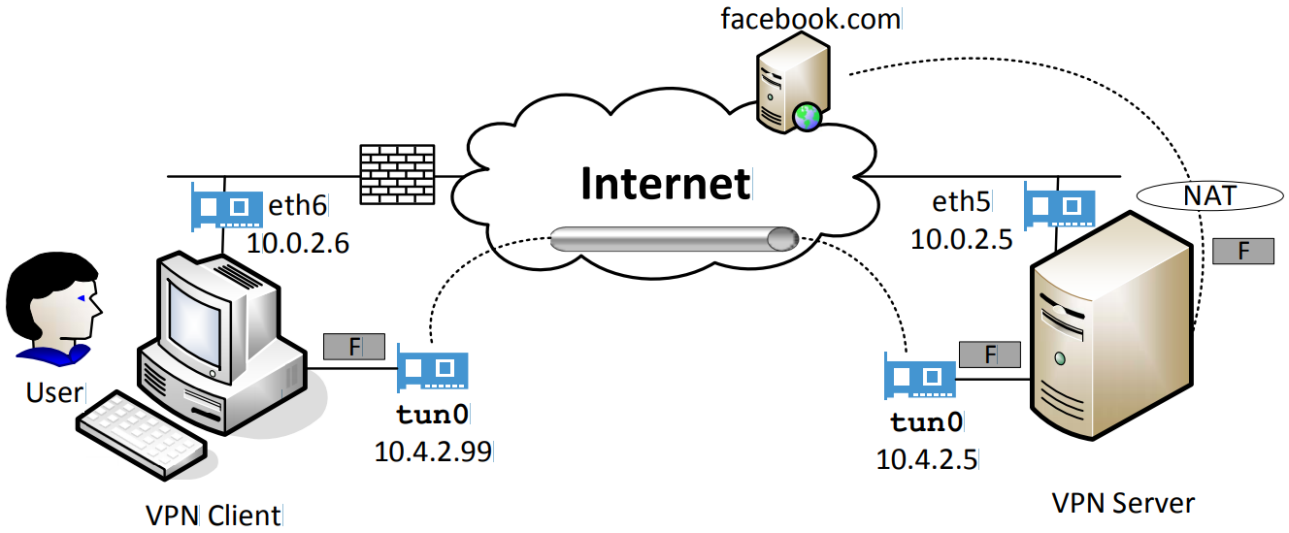
Bypassing Firewalls using VPN

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.