# VPN-against-Firewall Lab: Bypassing Firewalls using VPN



## 1 Overview

Organizations, Internet Service Providers (ISPs), and countries often block their internal users from accessing certain external sites. This is called egress filtering. For example, to prevent work-time distraction, many companies set up their egress firewalls to block social network sites, so their employee cannot access those sites from inside their network. For political reasons, many countries set up egress filtering at their ISPs to block their people from accessing selected foreign web sites. Unfortunately, these firewalls can be easily bypassed, and services/products that help users bypass firewalls are widely available on the Internet. The most commonly used technology to bypass egress firewalls is Virtual Private Network (VPN). In particular, this technology is widely used by smartphone users that are affected by egress filtering; there are many VPN apps (for Android, iOS, and other platforms) that can help users bypass egress firewalls.

The learning objective of this lab is for students to see how VPN works in action and how VPN can help bypass egress firewalls. We will implement a very simple VPN in this lab, and use it to bypass firewalls. A typical VPN depends on two pieces of technologies: IP tunneling and encryption. The tunneling technology is the most essential one to help bypass firewalls; the encryption technology is for protecting the content of the traffic that goes through the VPN tunnel. For the sake of simplicity, we will only focus on the tunneling part, so the traffic inside our tunnel is not encrypted. We have a separate VPN lab, which covers both tunneling and encryption. If readers are interested, they can work on our VPN lab to learn how to build a complete VPN. In this lab, we only focus on how to use VPN tunnel to bypass firewalls.

## 2 Lab Environment

We need two VMs (VM1 and VM2). VM1 is a computer inside the firewall, and VM2 is outside the firewall. The objective is to help VM1 bypass the firewall, so it can reach out to the blocked sites on the Internet. We would like to emulate such a setup using virtual machines inside our host machine. Figure 1 depicts the lab setup.

VM1 and VM2 each connects to a NAT-Network adapter (different), so they both can access the Internet through their corresponding NAT servers. In the real world, that should be sufficient, and VM1 and VM2 can communicate. However, in our lab environment, to enable VM1 and VM2 to communicate with each other, we have to emulate the Internet. The easiest way is to put a router VM1's network and VM2's network, but that requires us to have three VMs. To minimize the number of VMs, we decided to directly connect VM1 and VM2 using a host-only network adapter (available in VirtualBox). This host-only network emulates the Internet that connects VM1 and VM2.

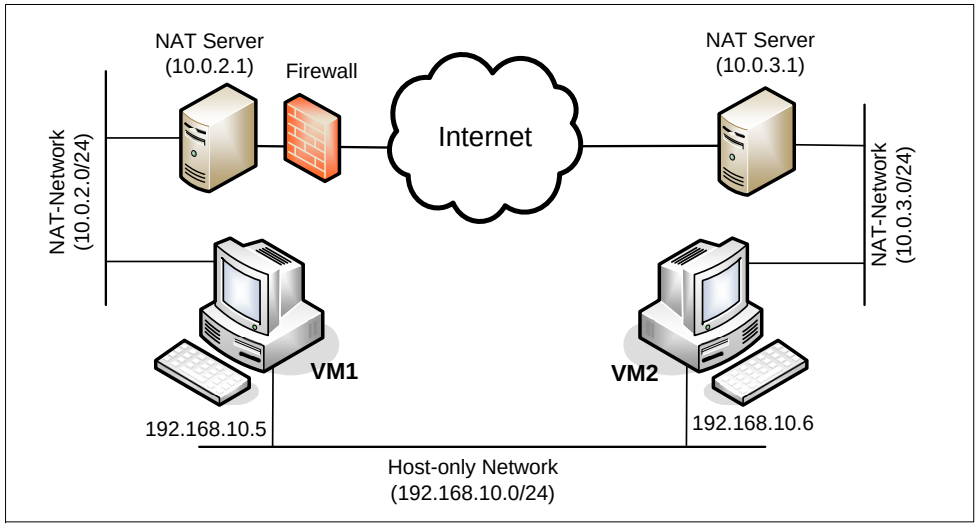



Figure 1: Lab Environment Setup

# 3 Lab Tasks

Students need to first implement a simple VPN for `Linux`, called `miniVPN`. Using this VPN, we can establish a tunnel between VM1 and VM2. When a user on VM1 tries to access the Internet, the traffic will not directly go through its own NAT server, because it will be subject to egress filtering. Instead, all the Internet-bound IP packets from VM1 will be redirected to the VPN tunnel and arrive at VM2. Once they arrive there, VM2 will route them to its own NAT server, and eventually release them to the Internet. When the reply packets come back, it will come back to VM2's NAT server, which will give it back to VM2. VM2 will then redirect the packets to the VPN tunnel, and eventually get the packet back to VM1. That is how the VPN helps VM1 to bypass firewalls.

Establishing the VPN to bypass firewalls is not trivial. We break down the mission into several tasks, and provide some detailed instructions about each task.

## 3.1 Task 1: Create a Host-to-Host Tunnel using TUN/TAP

The enabling technology for our VPN is TUN/TAP, which is now widely implemented in modern operating systems. TUN and TAP are virtual network kernel drivers; they implement network device that are supported entirely in software. TAP (as in network tap) simulates an Ethernet device and it operates with layer-2 packets such as Ethernet frames; TUN (as in network TUNnel) simulates a network layer device and it operates with layer-3 packets such as IP packets. With TUN/TAP, we can create virtual network interfaces.

A user-space program is usually attached to the TUN/TAP virtual network interface. Packets sent by an operating system via a TUN/TAP network interface are delivered to the user-space program. On the other hand, packets sent by the program via a TUN/TAP network interface are injected into the operating system network stack; to the operating system, it appears that the packets come from an external source through the virtual network interface.

When a program is attached to a TUN/TAP interface, the IP packets that the computer sends to this interface will be piped into the program; on the other hand, the IP packets that the program sends to the interface will be piped into the computer, as if they came from the outside through this virtual network

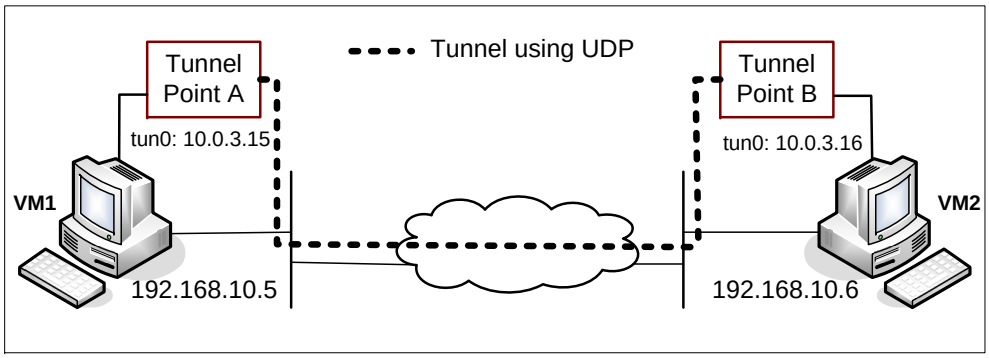



Figure 2: VPN Tunnel (in our example, the Internet is emulated by a host-only network that directly connects VM1 and VM1)

interface. The program can use the standard `read()` and `write()` system calls to receive packets from or send packets to the virtual interface.

Davide Brini has written an excellent tutorial article on how to use TUN/TAP to create a tunnel between two machines. The URL of the tutorial is `http://backreference.org/2010/03/26/tuntap-interface-tutorial`. The tutorial provides a program called `simpletun`, which connects two computers using the TUN tunneling technique. Students should go read this tutorial. When you compile the sample code from the tutorial, if you see error messages regarding `linux/if.h`, try to change `"<linux/if.h>"` to `"<net/if.h>"` in the `include` statement.

For the convenience of this lab, we have modified the Brini's `simpletun` program, and linked the code in the lab web page. Students can simply download this C program and run the following command to compile it. We will use `simpletun` to create tunnels in this lab:

```
$ gcc -o simpletun simpletun.c
```

**Creating Host-to-Host Tunnel.** The following procedure shows how to create a tunnel between two computers using the `simpletun` program. The `simpletun` program can run as both a client and a server. When it is running with the `-s` flag, it acts as a server; when it is running with the `-c` flag, it acts as a client.

1. **Launch two virtual machines.** For this task, we will launch these two VMs (VM1 and VM2) on the same host machine. These two machines are connected via a host-only network (see Figure 1). In the real world, these two machines should be on two different locations, and they are connected via the Internet. For example, to bypass the egress firewall of a country, one machine (the VPN client) will reside inside the country, while the other one (the VPN server) will reside outside of the country. In our lab environment, we put them on the same LAN, one inside the scope of the firewall and one outside.

   The IP addresses for the two machines are `192.168.10.5`, and `192.168.10.6`, respectively (you can choose any IP addresses you like). See the configuration in Figure 2.

2. **Tunnel Point A:** we use Tunnel Point A as the server side of the tunnel. Point A is on machine `192.168.10.5` (see Figure 2). It should be noted that the client/server concept is only meaningful when establishing the connection between the two ends. Once the tunnel is established, there is no difference between client and server; they are simply two ends of a tunnel. We run the following command (the `-d` flag asks the program to print out the debugging information):

```
On Machine 192.168.10.5:
# ./simpletun -i tun0 -s -d
```

After the above step, your computer now have multiple network interface, one is its own Ethernet card interface, and the other is the virtual network interface called `tun0`. This new interface is not yet configured, so we need to configure it by assigning an IP address.

We use the IP address from the reserved IP address space (`10.0.0.0/8`).

It should be noted that the above command will block and wait for connections, so, we need to find another window to configure the `tun0` interface. Run the following commands (the first command will assign an IP address to the interface `"tun0"`, and the second command will bring up the interface):

```
On Machine 192.168.10.5:
# ip addr add 10.0.3.15/24 dev tun0
# ifconfig tun0 up
```

3. **Tunnel Point B:** we use Tunnel Point B as the client side of the tunnel. Point B is on machine `192.168.10.6` (see Figure 2). We run the following command on this machine (The first command will connect to the server program running on `192.168.10.5`, which is the machine running the Tunnel Point A. This command will block as well, so we need to find another window for the second and the third commands):

```
On Machine 192.168.10.6:
# ./simpletun -i tun0 -c 192.168.10.5 -d
# ip addr add 10.0.3.16/24 dev tun0
# ifconfig tun0 up
```

4. **Routing Path:** After the above two steps, the tunnel will be established. Before we can use the tunnel, we need to set up the routing path on both machines to direct the intended outgoing traffic through the tunnel. The following routing table entry directs all the packets for the `10.0.3.0/24` network through the interface `tun0`, from where the packet will be hauled through the tunnel.

```
On Machine 192.168.10.5:
# route add -net 10.0.3.0 netmask 255.255.255.0 dev tun0

On Machine 192.168.10.6:
# route add -net 10.0.3.0 netmask 255.255.255.0 dev tun0
```

5. **Using Tunnel:** Now we can access `10.0.3.16` from `192.168.10.5` (and similarly access `10.0.3.15` from `192.168.10.6`). We can test the tunnel using `ping` and `ssh` (note: do not forget to start the `ssh` server first):

```
On Machine 192.168.10.5:
$ ping 10.0.3.16
$ ssh 10.0.3.16

On Machine 192.168.10.6:
$ ping 10.0.3.15
$ ssh 10.0.3.15
```

### 3.2 Task 2: Set up Host-to-Gateway Tunnel

So far, we have created a host-to-host tunnel, but we are not interested in accessing VM2 through the tunnel; we are interested in using VM2 to help us bypass the firewall on VM1. In other words, the packets that we want to send through the VPN tunnel are not for VM2; they are for the destinations on the Internet. To achieve that, once VM2 receives our packets coming from the tunnel, it has to route the packets out. That is, VM2 has to function like a gateway, which will make our tunnel a host-to-gateway tunnel. There are several things that we need to do to make the tunnel work.

1. **Set up IP forwarding:** Unless specifically configured, a computer will only act as a host, not as a gateway. In `Linux`, we need to enable the IP forwarding for a computer to behave like a gateway. IP forwarding can be enabled using the following command:

```
$ sudo sysctl net.ipv4.ip_forward=1
```

2. **Set up the routing table on VM2:** We also need to set up the routing table on VM2, so it routes the Internet traffic. VM2 is attached to a NAT-Network adapter, allowing it to reach the outside world through NAT. Students need to set up the routing table, so all packets that are not for the private network (i.e. 10.0.3.0/24) should all be routed to the NAT's IP address. The following example shows how to use the command `route` to configure routing tables (it is only an example, not the exact command for this step).

```
$ sudo route add -net 10.0.10.0 netmask 255.255.255.0 gw 10.0.20.1
```

3. **Get around the limitation of NAT:** When the destination sends packets back to the machine on the private network, it will reach the NAT first (because the source IPs of all the outgoing packets are changed to the NAT's external IP address (which is basically the host computer's IP address in our setup). Usually, the NAT will replace the destination IP address with the IP address of the original packet (i.e. 10.0.3.15 in our case), and give it back to whoever owns the IP address. Unfortunately, we have a problem here.

   Before the NAT sends out the packet, it needs to know the MAC address of the machine who owns 10.0.3.15, so it sends an ARP request. Our private network is virtual, so although 10.0.3.15 looks like on the same network as the NAT, it is only "attached" to the private network through the tunnel, so 10.0.3.15 will not receive the ARP request (even if it does, it has no use). The NAT will then drop the packet, because the recipient does not exist.

   The actual recipient should be VM2, even though it does not own the IP address 10.0.3.15. If we can configure the NAT as a gateway, we can ask the NAT to route the packets for 10.0.3.15 to the gateway VM2, which will eventually deliver the packets through the tunnel to VM1. However, we have not figured out how to configure the NAT as a gateway in VirtualBox, so we come up with a work-around solution. The idea is to "fool" the NAT to believe that the MAC address of 10.0.3.15 is VM2's MAC address, so the packet will be delivered to VM2 by the NAT. The solution is to do an ARP cache poisoning on the NAT, basically telling the NAT before hand about the MAC address of 10.0.3.15.

4. **Another way to get around the limitation of NAT:** Another way (a better way) to get round the limitation of the NAT is to create another NAT right on VM2, so all packets coming out of VM2 will have VM2's IP address as their source IP. To reach the Internet, these packets will go through another NAT, which is provided by VirtualBox, but since the source IP is VM2, this second NAT will have no

problem relaying back the returned packets from the Internet to VM2. Using this solution, we do not need to use ARP cache poisoning to "fool" the NAT any more. The following commands can enable the NAT on VM2 (in your case, the name of the NatNetwork adapter may not be called eth15; you just need to find its real name on your VM2):

```
Clean all iptables rules:
$ sudo iptables -F
$ sudo iptables -t nat -F

Add a rule on postrouting position to the NatNetwork adapter (eth15)
  connected to VPN server.
$ sudo iptables -t nat -A POSTROUTING -j MASQUERADE -o eth15
```

### 3.3 Task 3: Set up Firewall

Set up the firewall on VM1 to block the access of a target web site. You need to make sure that the IP address of the target web site is either fixed or in a fixed range; otherwise, you may have trouble to completely block this web site. Please refer to the Firewall Lab for details about how to blocking web sites.

In the real world, the firewall should run on a separate machine, not on VM1. To minimize the number of VMs used in this lab, we put the firewall on VM1. Setting up the firewall on VM1 requires the superuser privilege, and so does the setup of the VPN tunnel. One may immediately say that if we already have the superuser privilege, why cannot we just simply disable the firewall on VM1. This is a good argument, but keep in mind, we put the firewall on VM1 simply because we do not want to create another VM in the lab environment. Therefore, although you have the superuser privilege on VM1, you are not allowed to use the privileged to reconfigure the firewall. You have to use VPN to bypass it.

Compared to putting the firewall on an external machine, putting the firewall on VM1 does have a small issue that we need to deal with. When we set up the firewall to block packets, we need to make sure not to block the packets from getting to our virtual interface, or even our VPN will not be able to get the packets. Therefore, we cannot set the firewall rule before the routing, nor can we set the firewall rule on the virtual interface. We just need to set the rule on VM1's real network interface, so it will not affect the packets that go to the virtual interface. We can use the following `iptables` command to achieve the goal. The following example blocks the packets to the 128.230.210.0/24 network from going through the `eth12` interface, where `eth12` is VM1's real interfacea (in our setup, it is the interface connected to the NAT-Network adapter).

```
$ sudo iptables -t mangle -A POSTROUTING -d 128.230.210.0/24 -o eth12 -j DROP
```

### 3.4 Task 4: Bypassing Firewall

If you have done Tasks 1 to 3 correctly, you should be able to demonstrate how you can use the VPN to bypass the firewall. You should show that you can reach the blocked web site from VM1 via the VPN. Your solution should not only work for web traffic, but also for all other traffic. For example, if the blocked machine runs a `telnet` server, you should be able to `telnet` to this blocked server from VM1.

In your lab report, you should provide the evidence to show that your traffic did go through the VPN tunnel, not through some "side doors". The best way to show that is to capture the network traffic using Wireshark, and describe the path of your packets using the captured traffic. Without such an evidence, we have no idea whether your success is due to a mis-configured firewall (i.e. the targeted web site is not

blocked at all) or due to your VPN.

# 4 Guidelines

## 4.1 Watch the lecture video

Because we only came up with this lab during a weekend, we did not have much time to write down all the details here. I will discuss this extensively in my lecture, so please watch the lecture videos.

## 4.2 An example: using `telnet` in our VPN

To help you fully understand how packets from an application flow to its destination through our `MiniVPN`, we have drawn two figures to illustrate the complete packet flow path when users run `telnet 10.0.20.100` from a host machine, which is the Point A of a host-to-gateway VPN. The other end of the VPN is on a gateway, which is connected to the `10.0.20.0/24` network, where our `telnet` server `10.0.20.100` resides. The setup of the IP addresses is for illustration purposes, and it is not exactly the same as the setup required for our tasks.

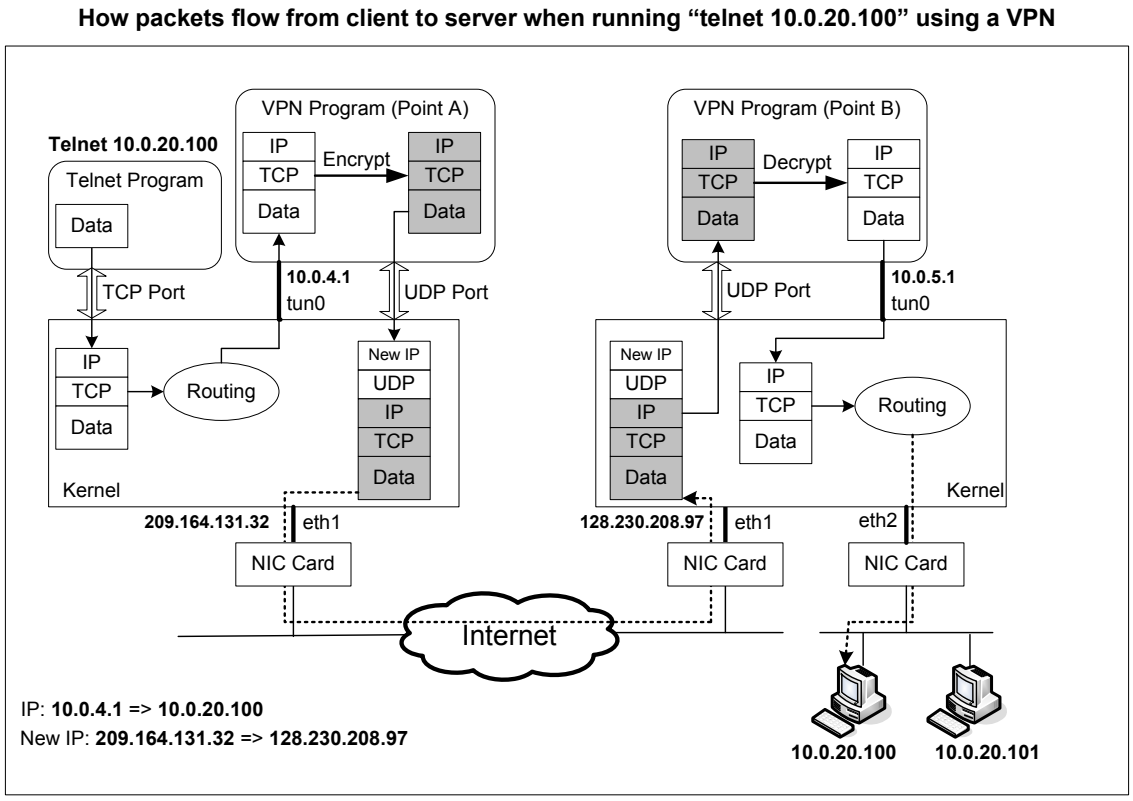
Figure 3(a) shows how a packet flow from the `telnet` client to the server. Figure 3(b) shows how a packet flow from the `telnet` server back to the client. We will only describe the path in Figure 3(a) in the following. The return path is self-explained from Figure 3(b) once you have understood the path in Figure 3(a).

1. The data of the packet starts from the `telnet` program.
2. The kernel will construct an IP packet, with the destination IP address being `10.0.20.100`.
3. The kernel needs to decide which network interface the packet should be routed through: `eth1` or `tun0`. You need to set up your routing table correctly for the kernel to pick `tun0`. Once the decision is made, the kernel will set the source IP address of the packet using the IP address of the network interface, which is `10.0.4.1`.
4. The packet will reach our VPN program (Point A) through the virtual interface `tun0`, then it will be encrypted, and then be sent back to the kernel through a UDP port (not through the `tun0` interface). This is because our VPN program use the UDP as our tunnel.
5. The kernel will treat the encrypted IP packet as UDP data, construct a new IP packet, and put the entire encrypted IP packet as its UDP payload. The new IP's destination address will be the other end of the tunnel (decided by the VPN program we write); in the figure, the new IP's destination address is `128.230.208.97`.
6. You need to set up your routing table correctly, so the new packet will be routed through the interaface `eth1`; therefore, the source IP address of this new packet should be `209.164.131.32`.
7. The packet will now flow through the Internet, with the original `telnet` packet being entirely encrypted, and carried in the payload of the packet. This is why it is called a *tunnel*.
8. The packet will reach our gateway `128.230.208.97` through its interface `eth1`.
9. The kernel will give the UDP payload (i.e. the encrypted IP packet) to the VPN program (Point B), which is waiting for UDP data. This is through the UDP port.

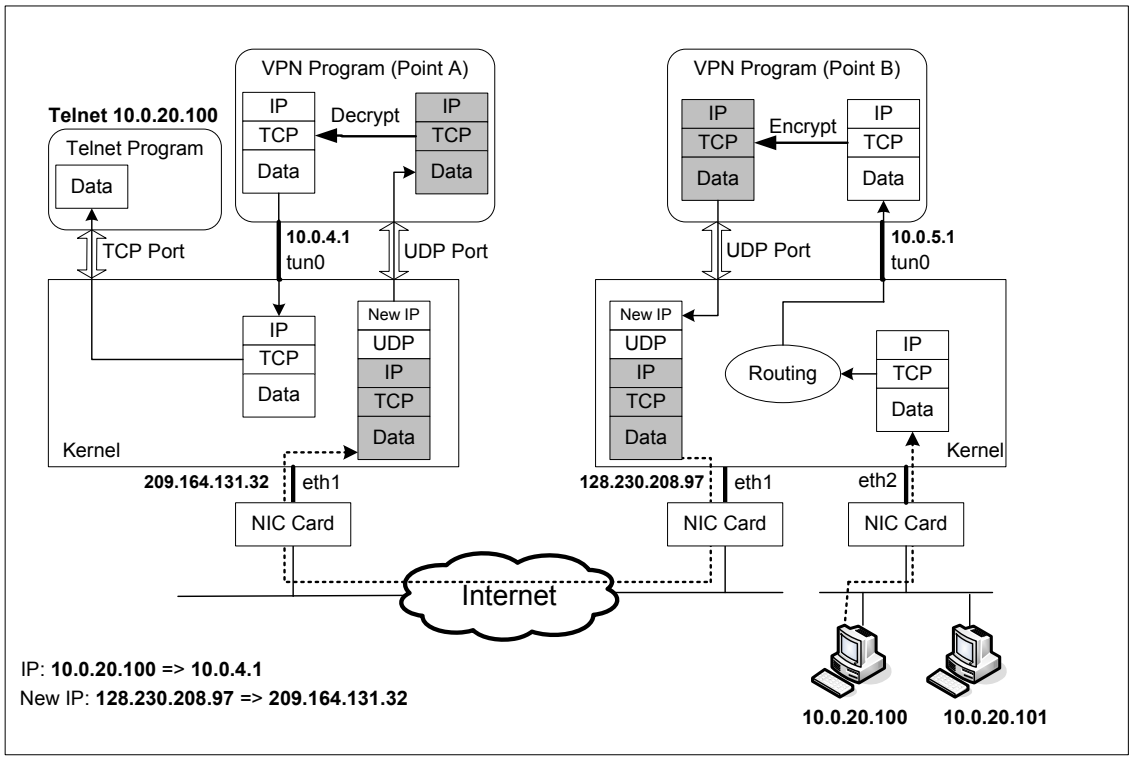
10. The VPN program will decrypt the payload, and then feed the decrypted payload, which is the original `telnet` packet, back to the kernel through the virtual network interface `tun0`.
11. Since it comes through a network interface, the kernel will treat it as an IP packet (it is indeed an IP packet), look at its destination IP address, and decide where to route it. Remember, the destination IP address of this packet is `10.0.20.100`. If your routing table is set up correctly, the packet should be routed through `eth2`, because this is the interface that connects to the `10.0.20.0/24` network.
12. The `telnet` packet will now be delivered to its final destination `10.0.20.100`.

# 5 Submission and Demonstration

You should submit a detailed lab report to describe how you set up your VPN to bypass firewalls. You should provide sufficient evidences to convince us that your technique works. We will not take your words for it; we must see your evidences. If your evidences are not convincing, we will not give your the credits.




(a) An Example of packet flow from telnet client to server in Host-to-Gateway Tunnel




(b) An Example of packet flow from telnet server to client in Host-to-Gateway Tunnel

Figure 3: An Example of Packet Flow in VPN.