Implement Virtual Private Network

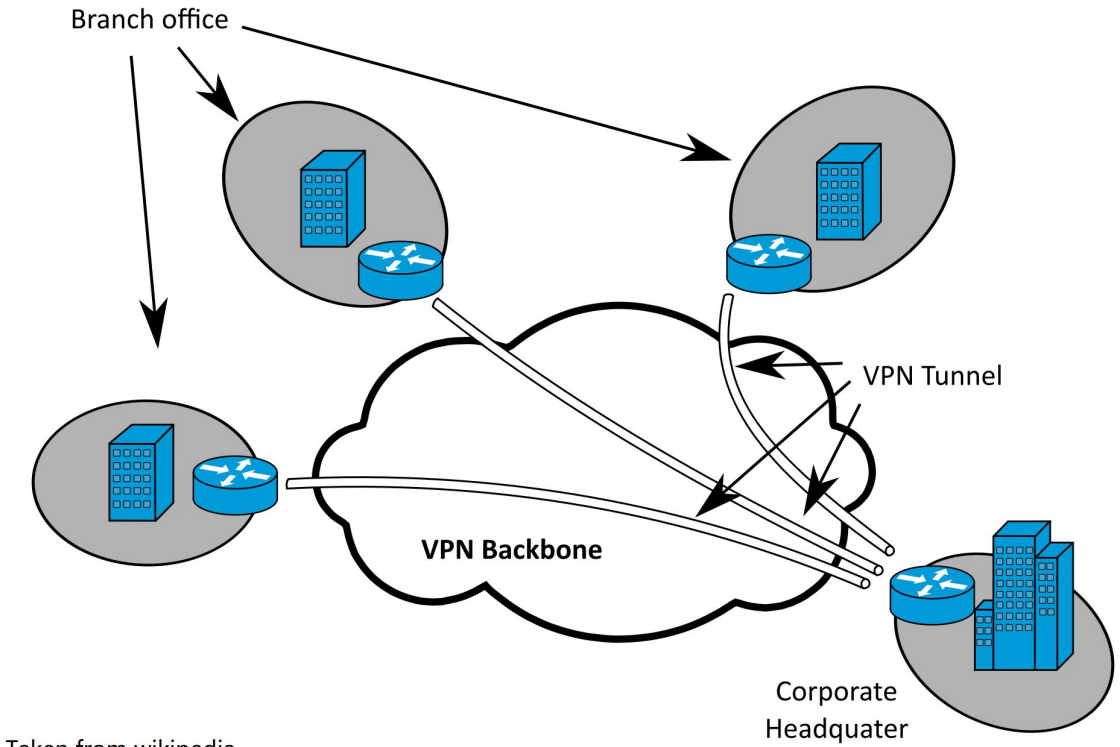
By Group 9: Sijie Yu, Yufei Zhang, Honghui Wang

Content

- Motivation
- Problem
- Challenges
- Solution
- Results
- Learned Lessons

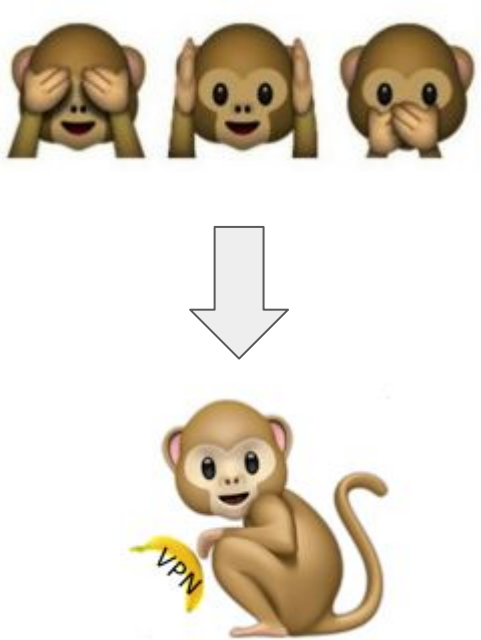
Motivation

Motivation: Secure and Confidential Communication



Taken from wikipedia

Motivation: Bypassing Censorship



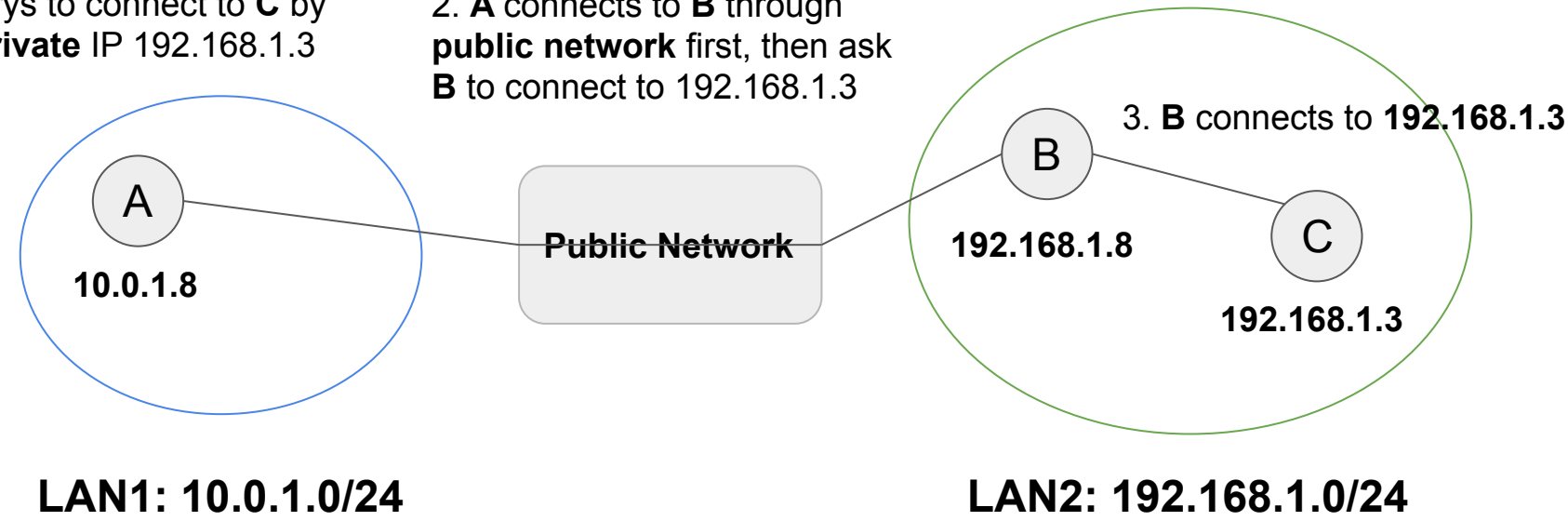
Motivation: Be Anonymous

Problem

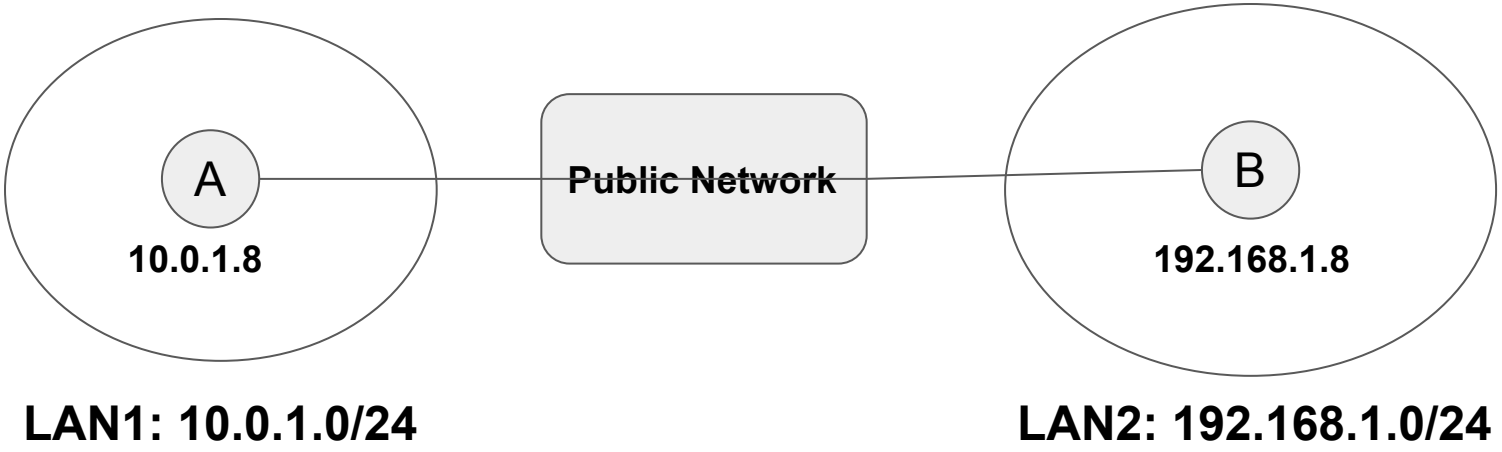
Problem: What is VPN

1. **A** trys to connect to **C** by C's **private** IP 192.168.1.3

2. **A** connects to **B** through **public network** first, then ask **B** to connect to 192.168.1.3

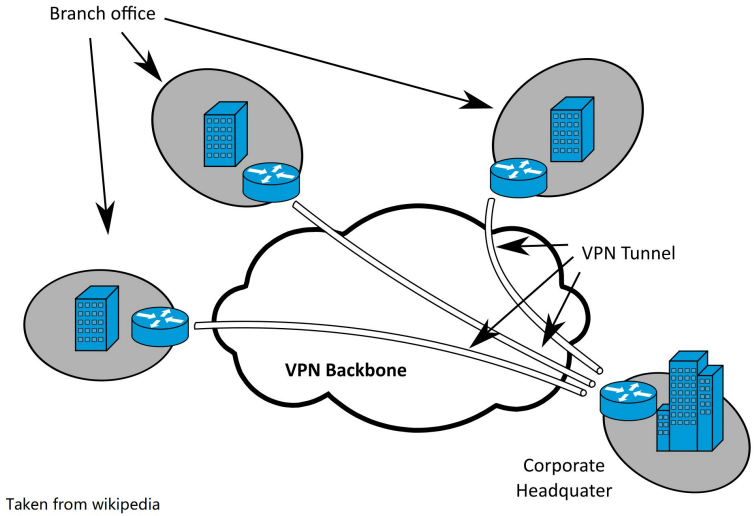


Problem: What is VPN



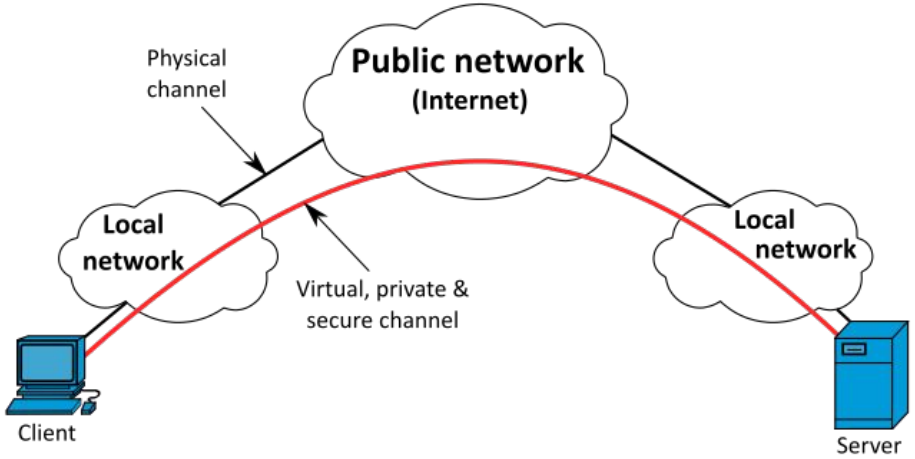
- The conversation between A and B (**VPN Tunnel**) is exposed to others in the public network.
- It secures the conversation by **encrypting** the data.
  - IP-IN-IP, IP-IN-UDP, OpenVPN, ...

Problem: Types of VPN



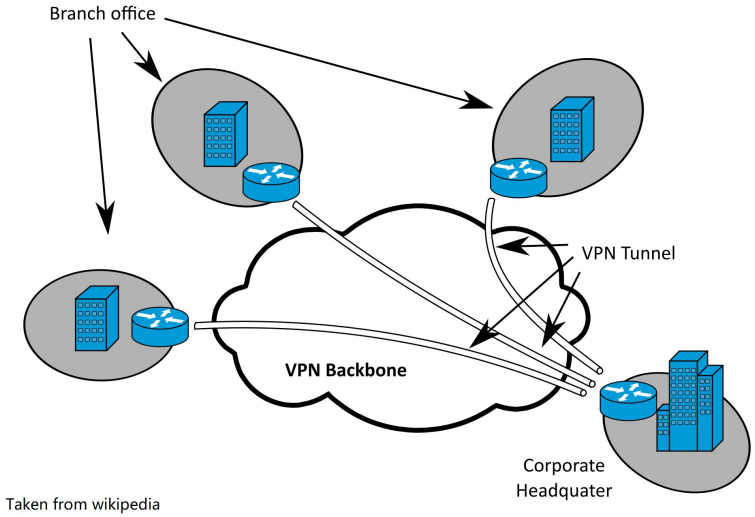
Taken from wikipedia

site-to-site VPN



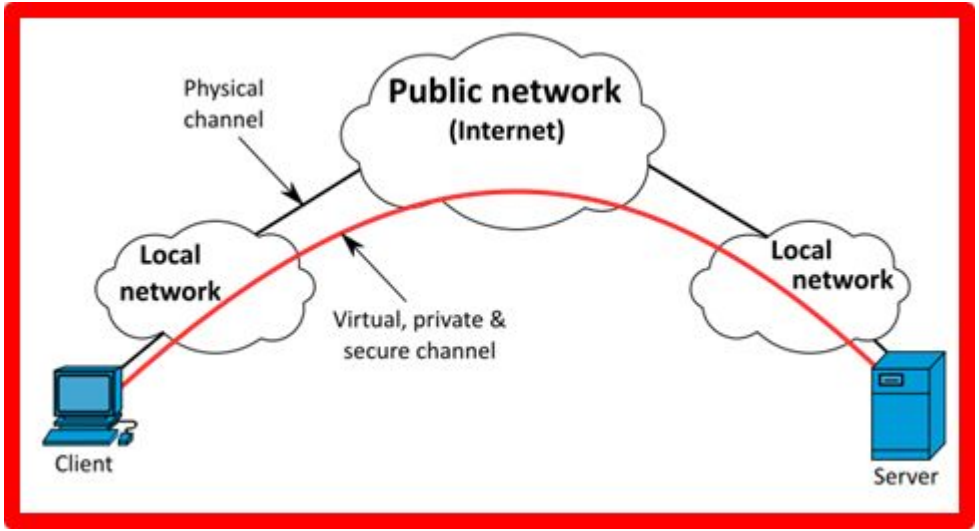
remote VPN

Problem: What we focus on



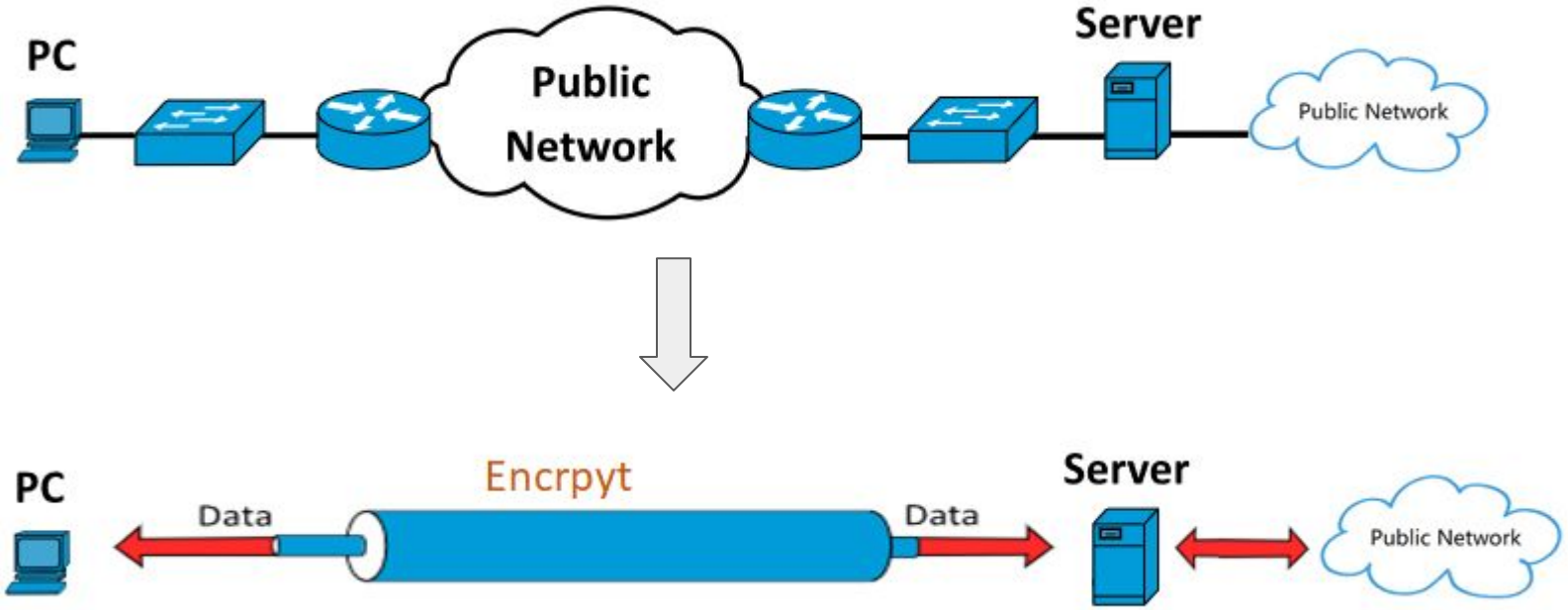
Taken from wikipedia

site-to-site VPN



remote VPN

Problem: Remote VPN



Challenges

Challenges

- Encrypt and maintain VPN tunnel
  - TUN/TAP device
  - Encyption types: IP-In-UDP, IPSec, OpenVPN, ...
- Multi-connections on VPN server
  - Garbage Collecting
- VPN server as proxy
  - Spoof and Forward packet

Solution

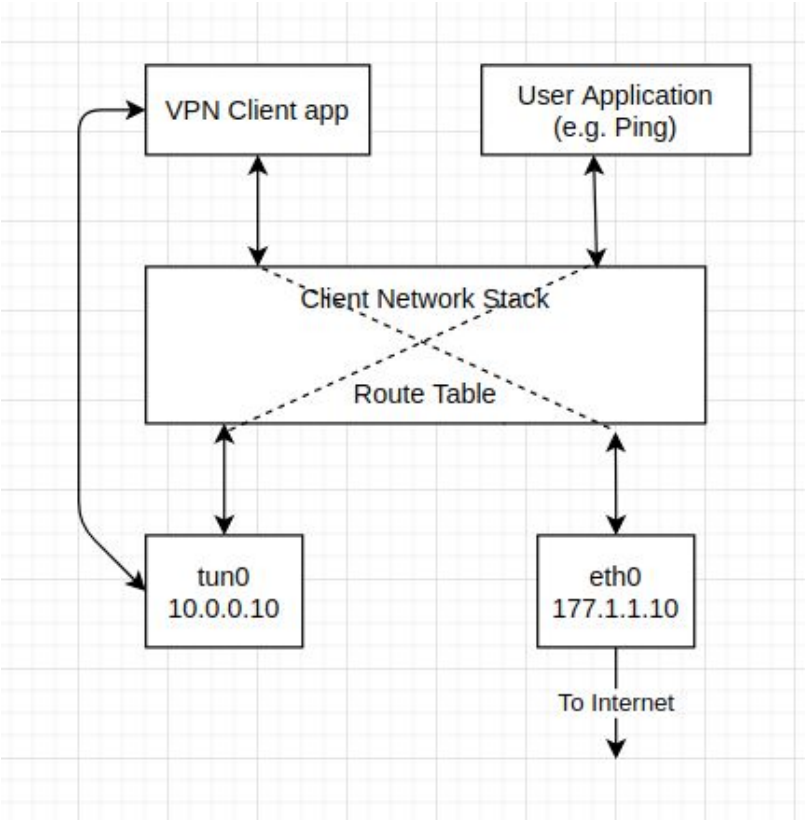
Solution: IP-in-UDP + Linux tun

Q : What is TUN?

A : TUN is a Linux virtual network device. It is on the same level with eth0.

eth0 will send data packet to real world internet.

however, tun will send data packet to a user space application.

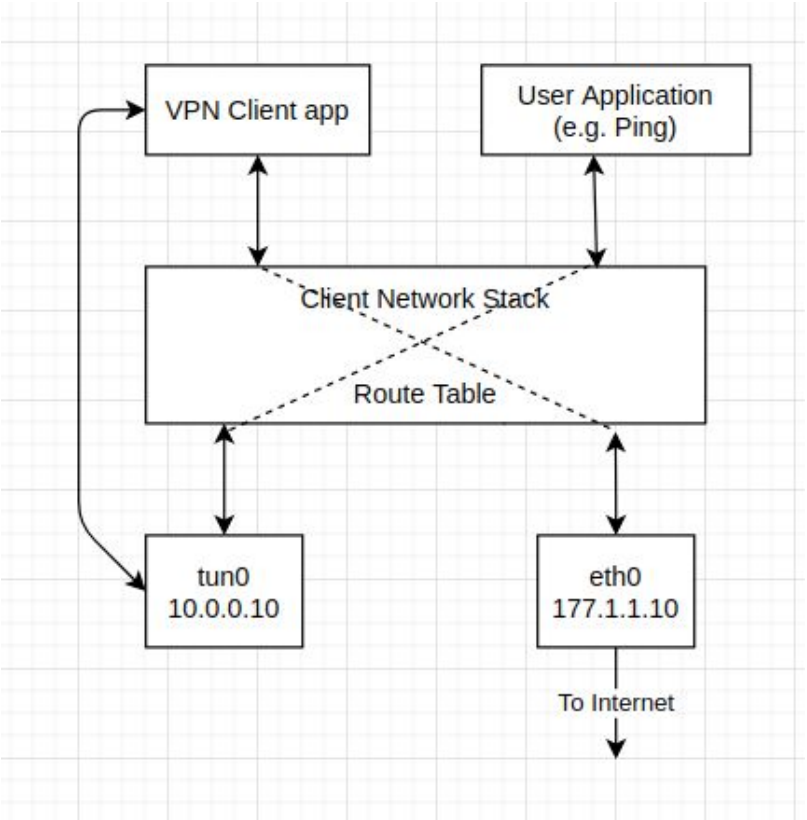


Solution: IP-in-UDP + Linux tun

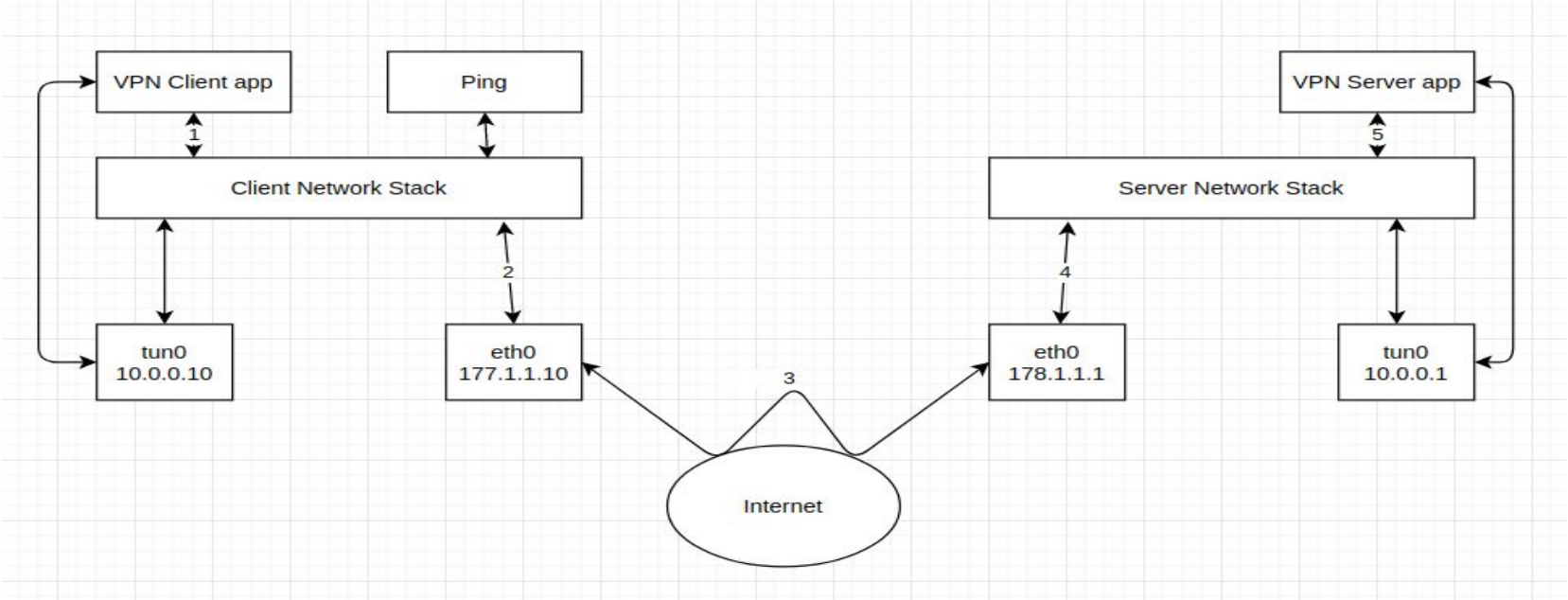
1. ping compose an ICMP request packet, and want to send it to 178.1.1.1 via 177.1.1.10 (eth0)

2. We modify the route table, so the ICMP packet goes to 10.0.0.10 (tun0)

3. VPN Client app will get the packet, with IP header and ICMP payload

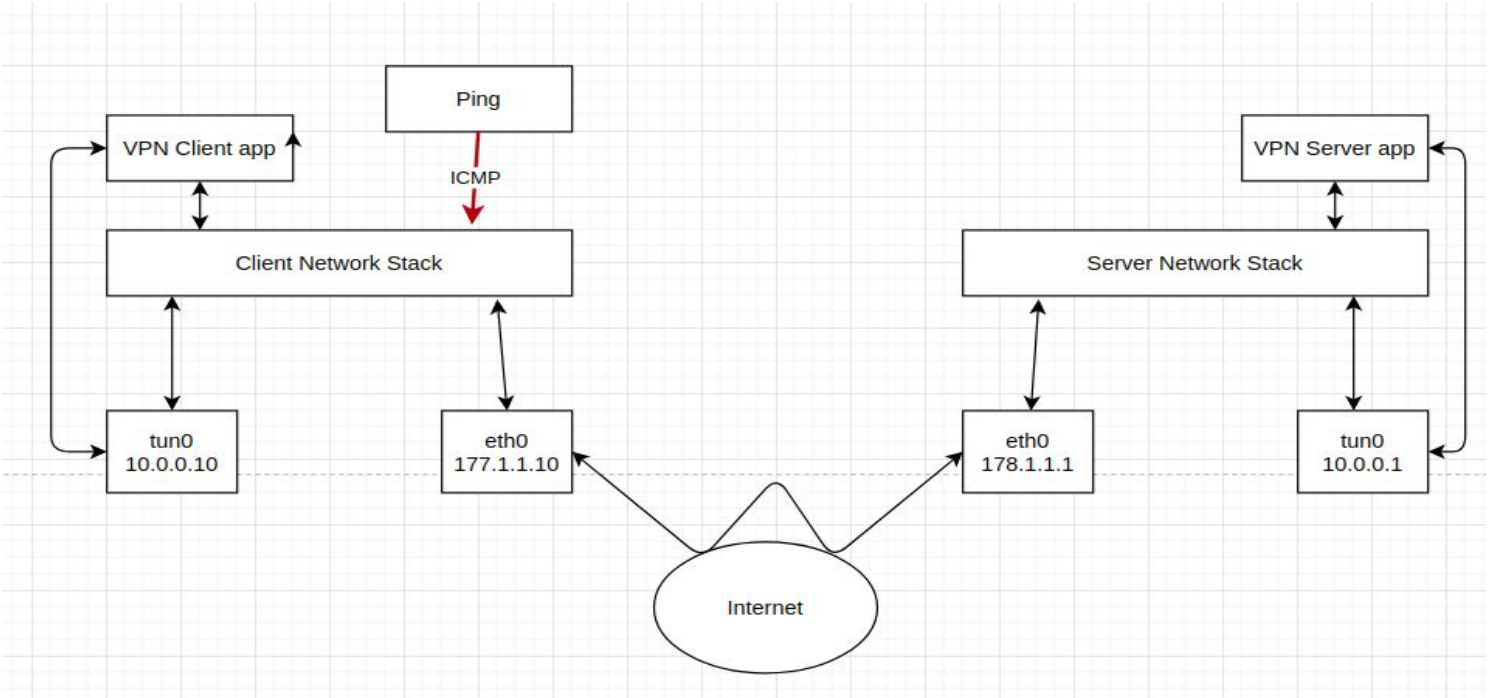


Solution: IP-in-UDP + Linux tun



1. VPN Client and VPN Server communicate withnormal UDP socket (it is connectionleess)

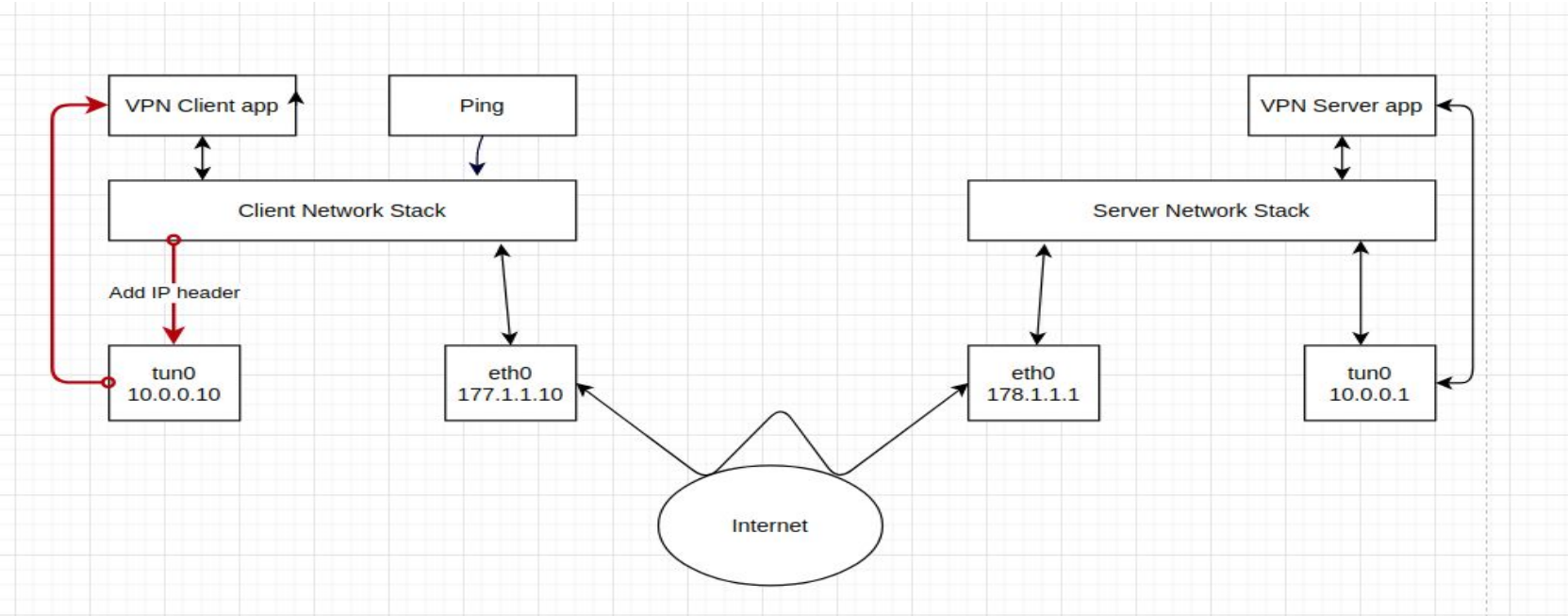
Solution: IP-in-UDP + Linux tun



2. ping on client machine wants to send a ICMP packet to google.com

Packet = ICMP(type = echo-request)

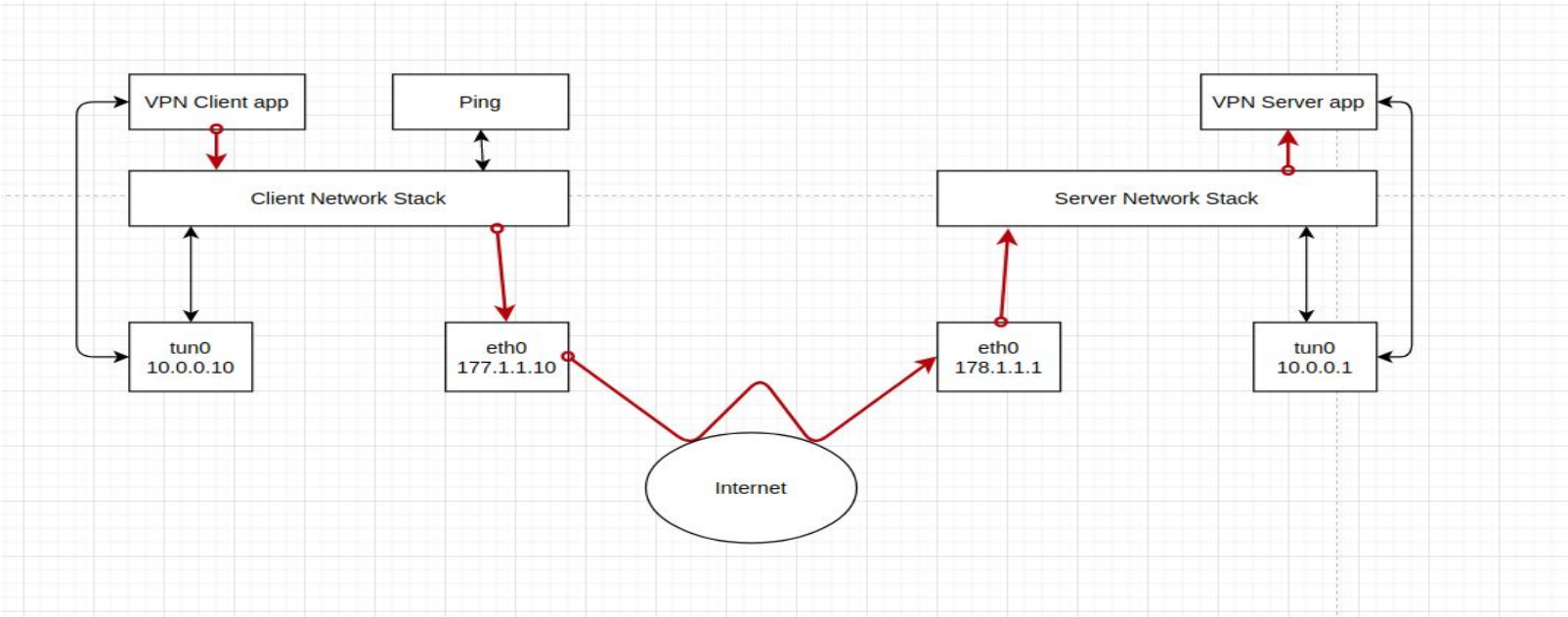
Solution: IP-in-UDP + Linux tun



3. tun0 on client machine intercept this packet, wrap it with IP header and transfer it to VPN Client

Packet = IP(src = 10.0.0.10, dst = 8.8.8.8)/ICMP(type = echo-request)

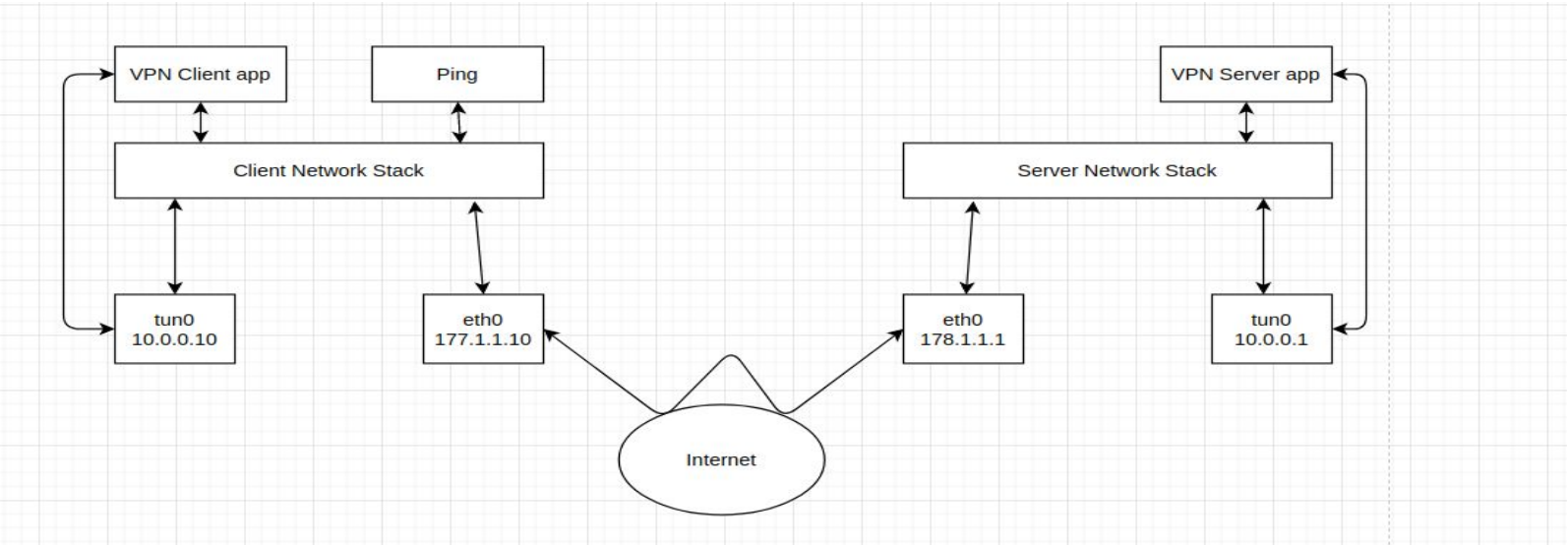
Solution: IP-in-UDP + Linux tun



4. VPN Client encapsulate the IP packet into a UDP datagram and send to VPN server

Packet = IP(src= 177.1.1.10 dst = 178.1.1.1) / UDP() / IP(src = 10.0.0.10, dst = 8.8.8.8) / ICMP(type = echo-request)

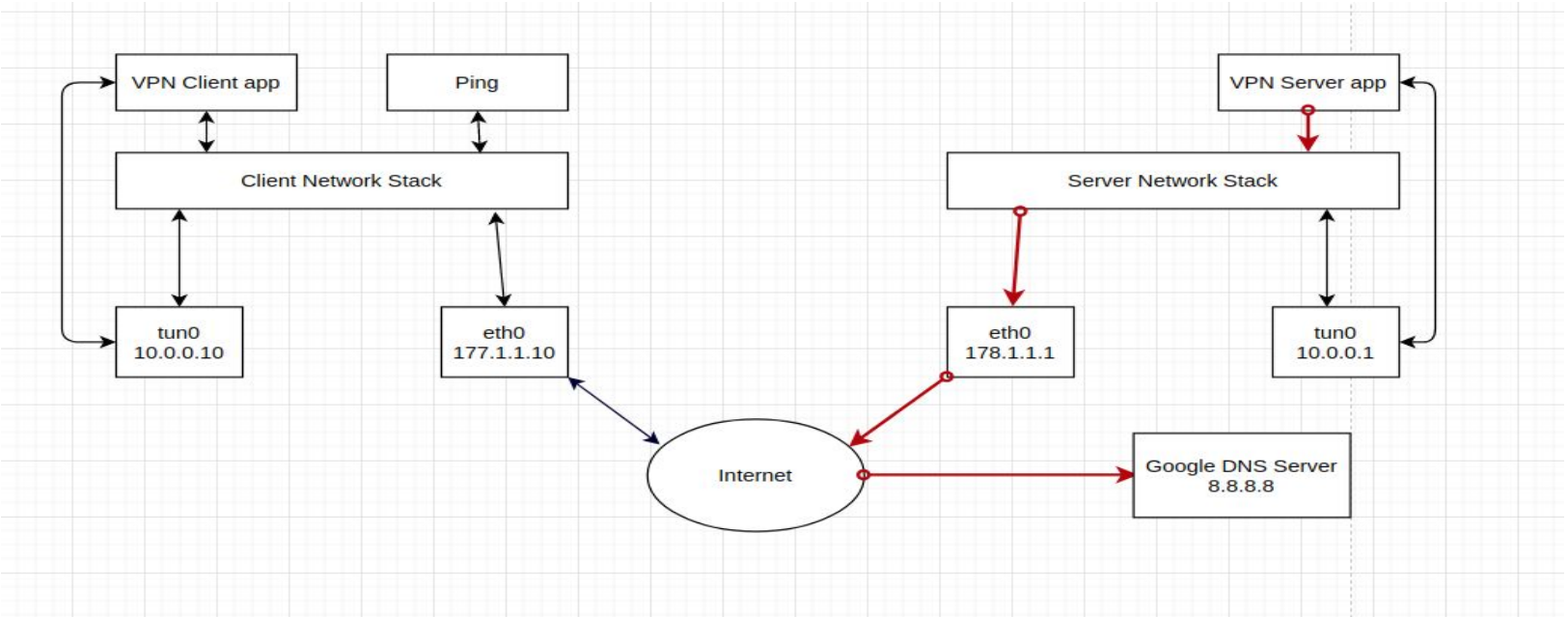
Solution: IP-in-UDP + Linux tun



5. VPN Server receives the UDP datagram and retrieve the inside IP packet. VPN Server changes the source IP address of the IP packet to the ip address of eth0 on server machine, recalculate checksum (using a raw socket)

Packet = IP(src = 178.1.1.1, dst = 8.8.8.8) / ICMP(type = echo-request)

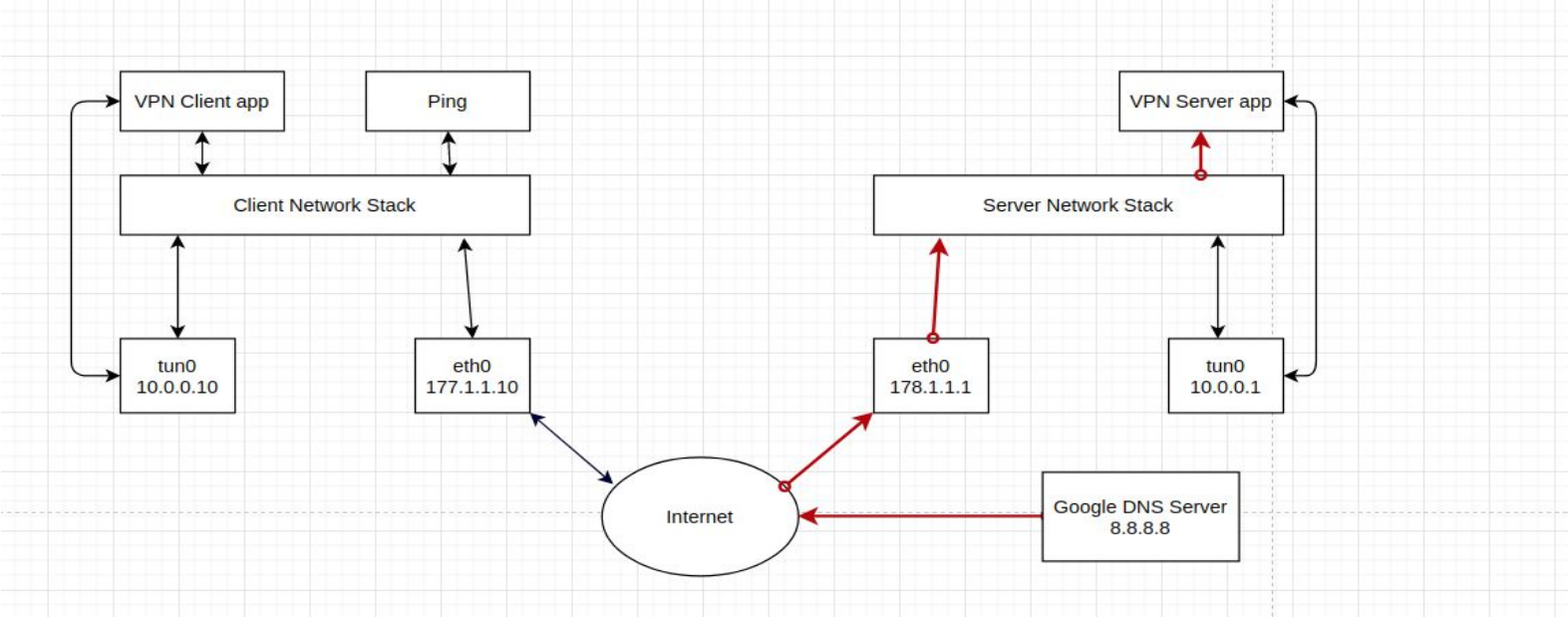
Solution: IP-in-UDP + Linux tun



6. VPN Server send the IP packet to google.com

Packet = IP(src = 178.1.1.1, dst = 8.8.8.8) / ICMP(type = echo-request)

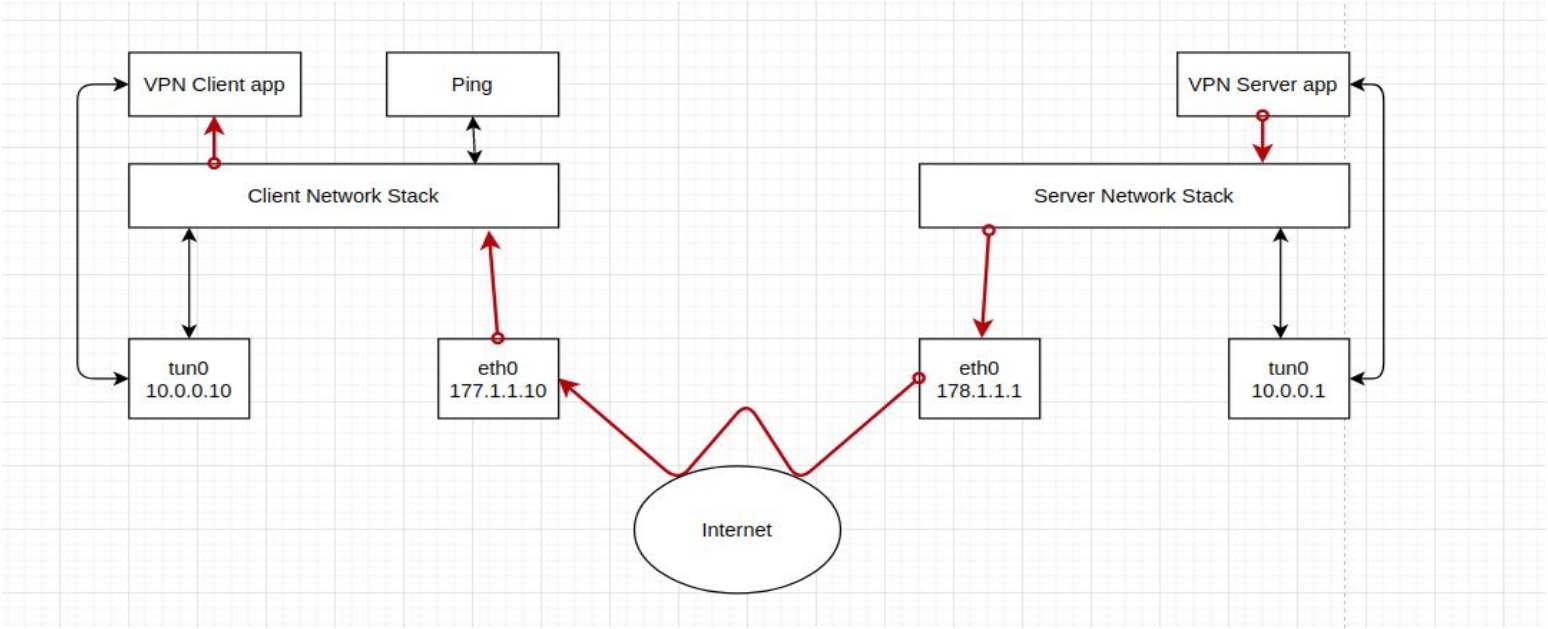
Solution: IP-in-UDP + Linux tun



7. Google Reply

Packet = IP(src = 8.8.8.8, dst = 178.1.1.1) / ICMP(type = echo-reply)

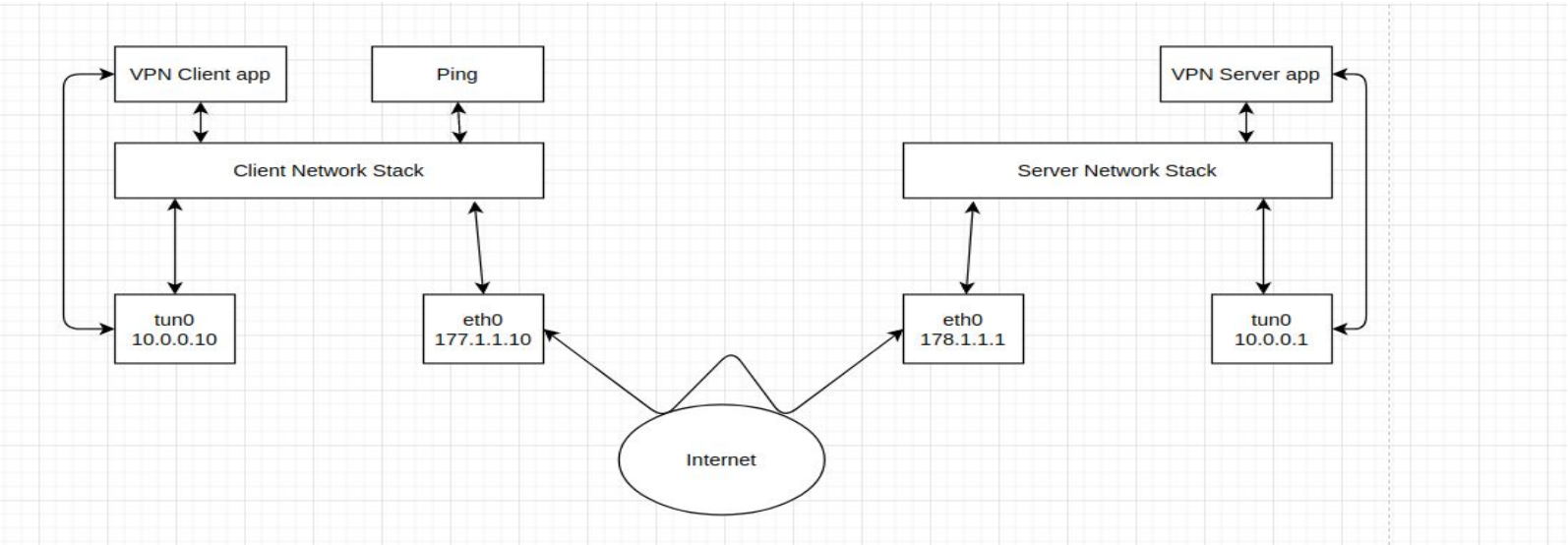
Solution: IP-in-UDP + Linux tun



8. VPN Server encapsulate the google's response IP packet into a UDP datagram, and send it back to VPN Client.

Packet = IP(src= 178.1.1.1 dst = 177.1.1.10) / UDP() /IP(src = 8.8.8.8, dst = 178.1.1.1) / ICMP(type = echo-reply)

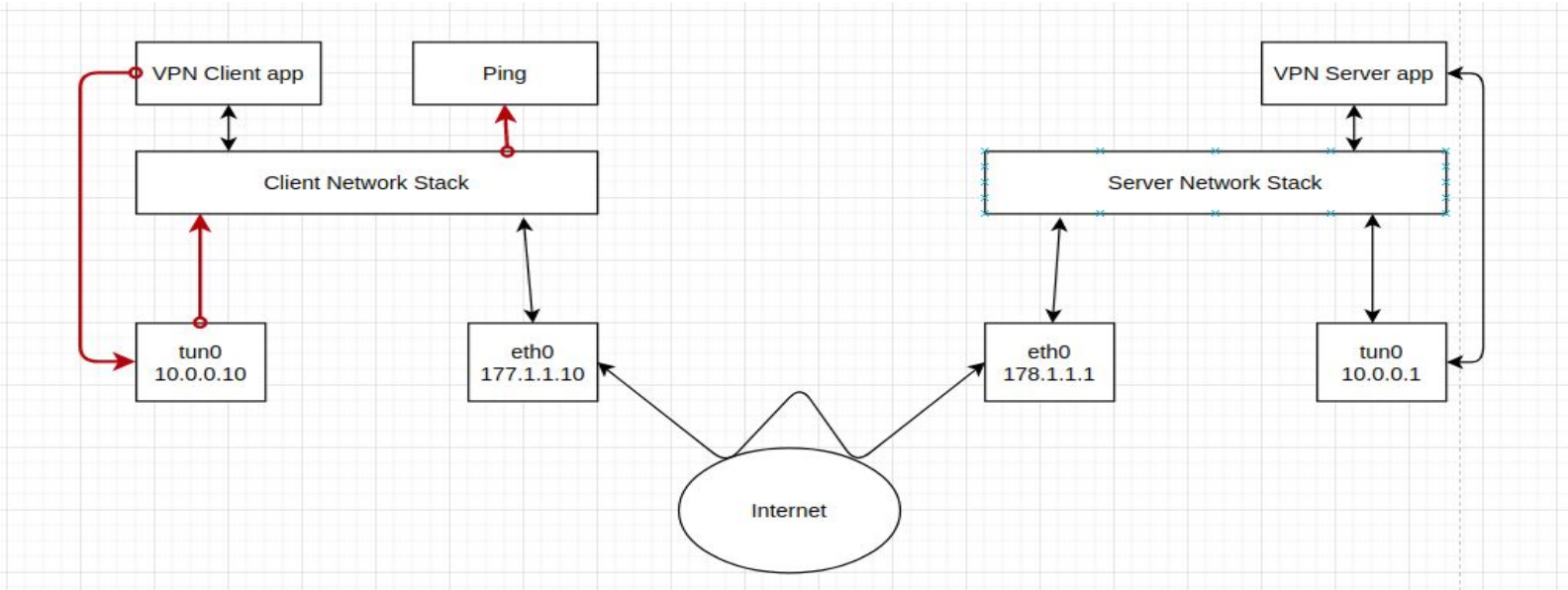
Solution: IP-in-UDP + Linux tun



9. VPN Client receives the UDP datagram and retrieve the inside IP packet. VPN Client changes the destination IP address of this IP packet to the ip address of tun0 on client machine. Then it recalculate the checksum

Packet = IP(src = 8.8.8.8, dst = 10.0.0.10) / ICMP(type = echo-reply)

Solution: IP-in-UDP + Linux tun



10. VPN Client send IP packet to tun0. tun0 transfer the IP packet to client machine's kernel network stack, and user space application receives it

Packet = IP(src = 8.8.8.8, dst = 10.0.0.10) / ICMP(type = echo-reply)

Solution: IP-in-UDP + Linux tun

1. VPN Client and VPN Server communicate withnormal UDP socket (it is connectionleess)
2. ping on client machine wants send a ICMP request to google.com
3. tun0 on client machine intercept this packet, wrap it with IP header and transfer it to VPN Client
4. VPN Client encapsulate the IP packet into a UDP datagram and send to VPN server
5. VPN Server receives the UDP datagram and retrieve the inside IP packet. VPN Server changes the source IP address of the IP packet to the ip address of eth0 on server machine, recalculate checksum (using a raw socket)
6. VPN Server send the IP packet to google.com
7. Google.com reply
8. VPN Server encapsulate the google's response IP packet into a UDP datagram, and send it back to VPN Client.
9. VPN Client receives the UDP datagram and retrieve the inside IP packet. VPN Client changes the destination IP address of the IP packet to the ip address of tun0 on client machine.Then it recalculate the checksum.
10. VPN Client send IP packet to tun0. tun0 transfer the IP packet to client machine's kernel network stack, and user space application receives it

Results

Results

1. we implemented multiple client connect to one server
2. server can forward packet to websites like google and forward response packets to client
3. so the major logic of VPN is done, but we still have

TODOs:

1. checksum recalculation
2. client-server encryption
3. test suite and documentation

Result

Server:

8	4.927195227	192.168.1.97	8.8.8.8	ICMP	98 Echo (ping) request
9	4.943681206	8.8.8.8	192.168.1.97	ICMP	98 Echo (ping) reply
13	5.937988503	192.168.1.97	8.8.8.8	ICMP	98 Echo (ping) request
14	6.043978800	8.8.8.8	192.168.1.97	ICMP	98 Echo (ping) reply
18	6.961915013	192.168.1.97	8.8.8.8	ICMP	98 Echo (ping) request
19	7.070502536	8.8.8.8	192.168.1.97	ICMP	98 Echo (ping) reply
23	7.983531109	192.168.1.97	8.8.8.8	ICMP	98 Echo (ping) request
24	8.000422978	8.8.8.8	192.168.1.97	ICMP	98 Echo (ping) reply

Client:

132	55.167526342	10.0.0.2	8.8.8.8	ICMP	84 Echo (ping) request
133	55.299600822	8.8.8.8	10.0.0.2	ICMP	84 Echo (ping) reply
134	56.191509565	10.0.0.2	8.8.8.8	ICMP	84 Echo (ping) request
135	56.225848526	8.8.8.8	10.0.0.2	ICMP	84 Echo (ping) reply
136	57.215698150	10.0.0.2	8.8.8.8	ICMP	84 Echo (ping) request
137	57.255789326	8.8.8.8	10.0.0.2	ICMP	84 Echo (ping) reply

Checksum:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.2	8.8.8.8	ICMP	84	Echo (ping) request id
2	0.043601043	8.8.8.8	10.0.0.2	ICMP	84	Echo (ping) reply id
3	1.010141224	10.0.0.2	8.8.8.8	ICMP	84	Echo (ping) request id
4	1.068852146	8.8.8.8	10.0.0.2	ICMP	84	Echo (ping) reply id
5	2.033967954	10.0.0.2	8.8.8.8	ICMP	84	Echo (ping) request id
6	2.084782123	8.8.8.8	10.0.0.2	ICMP	84	Echo (ping) reply id
7	3.057841081	10.0.0.2	8.8.8.8	ICMP	84	Echo (ping) request id
8	3.147622550	8.8.8.8	10.0.0.2	ICMP	84	Echo (ping) reply id

```
▸ Frame 2: 84 bytes on wire (672 bits), 84 bytes captured (672 bits) on interface 0
  Raw packet data
▾ Internet Protocol Version 4, Src: 8.8.8.8, Dst: 10.0.0.2
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 84
    Identification: 0x0000 (0)
  ▸ Flags: 0x0000
    Time to live: 57
    Protocol: ICMP (1)
  ▸ Header checksum: 0xaf90 incorrect, should be 0x6798(may be caused by "IP checksum offload"?)
    [Header checksum status: Bad]
    [Calculated Checksum: 0x6798]
    Source: 8.8.8.8
    Destination: 10.0.0.2
```

Learned Lessons

Learned Lessons

- How Linux **TUN** device works with **Network Stack**
  - At Layer 3 IP level.
- How Linux **Route Table** works
  - Default Gateway
- Python **socket** & **selector**
  - listening events of sockets and TUN
- **Spoof** IP Packet
  - TODO: re-calculate checksum

Thank You!