MINIVPN

Problem Statement:
A Virtual Private Network (VPN) is used for creating a private scope of computer communications or providing a secure extension of a private network into an insecure network such as the Internet. VPN is a widely used security technology. VPN can be built upon IPSec or Secure Socket Layer (SSL). These are two fundamentally different approaches for building VPNs. MiniVPN is an example to latter category which is SSL based ones.

Related works:
There are numerous VPN solutions which some of them are close sourced and some of them are open sourced. OpenVPN is one of the most widely used VPN solution. OpenVPN is an open source VPN solution. Also, there are relatively small sized VPN solutions compared to the OpenVPN. These type of VPN solutions are trying to solve very specific problems generally. GoHop[1] can be given as an example to these type of projects. Its main aim is to escape from censorship and intelligent package inspection.[2]

Approach:
MiniVPN is developed for Ubuntu which is one of the Linux distributions. This project can be transferred to the other Unix-like operating systems easily with minor changes.

There are two essential features of the any VPN program. One of them is creating virtual network between host machines of the VPN users. Second of them is that communication between the host machines in this virtual network should be encrypted.
For the first essential feature which is creating virtual network, tunneling technique is used in this project. There are different tunneling techniques for the Linux. One of these is tun/tap interface feature. **Tun/tap interfaces** are a feature that can do *userspace networking*, that is, allow userspace programs to see raw network traffic (at the ethernet or IP level) and do whatever they like with it.[3] Tun/tap interfaces are software-only interfaces which means that they exist only in the kernel and they have no physical component like regular network interfaces. These interfaces provide abstraction to the OS while sending packets through network stack because OS behaves to the tun/tap interfaces like regular network interfaces. Due to these properties, tun/tap interfaces are suitable for creating tunnels between machines. Tunnel will be created with the help of the userspace program which runs on the all the machines which are parts of the VPN. These userspace programs will read from the tun/tap interface which is initialized before and write to the real network interface which is connected to outside word. Also, user-space program will read from the real network interface if there are any available packets on it and it will write these packets to the tun/tap interface. If the users of the VPN wants to use VPN connections for their normal internet usage in their host machines, regular network traffic should be directed to the tun/tap interface. These configuration can be changed with the help of the tools like 'route'[4] in the Linux machines.

For the second essential feature which is encrypted communication, SSL is decided to use. Initially, we planned to use SSL for only control channel which will be used for interchanging symmetric encryption keys mainly. After interchange the keys, symmetric encryption will be used between host machines to continue to the encrypted communication. However, after second thought, we decided that it will be like reinvent the TLS and we can introduce some vulnerability in this process and we decided to use a widely used and well tested protocol which is DTLS. The DTLS protocol provides communications privacy for datagram protocols.[6] Main reason of the using this protocol is that it is based on the UDP which will prevent the 'TCP over TCP[5]' problem. We decided to use OpenVPN implementation of the DTLS for this project.

Implementation:

Since this project is planned as proof-of-concept, there are some missing features such as dynamic IP allocation, more than one client on the VPN or user authentication. Allocated IP's are predetermined and hard-coded in the init scripts.

As stated in the 'Approach' part, tun/tap interfaces are used for creating tunnels. Initially, tun/tap interface should be initialized before using them. 'ip'[7] command will be used for this purpose. Before starting the client or server VPN program, init scripts need to be called. After tun/tap interfaces are initialized VPN programs are ready to started. There are two types of this programs. One of them is client and one of them is server. Server client is listening whether there are any incoming requests and if there are any incoming requests, server program creates new thread and communication between client and server continues in this thread. Since it stated that UDP is used in this project before, it can be confusing to say that server program is creating new connection between client and server. However, as stated before, DTLS is used for encryption and DTLS creates connections like in the TCP. DTLS usage will be explained in detailed later on. As can be guessed, client program connects to the server program in the beginning of the program and start to send packets if there are any new incoming packets from tun/tap interface. This explains reason of the communication between two machines using UDP. Also, there is an in-host communication which will be done using tun/tap interfaces. Tun/tap interfaces are used for directing the client's network traffic to the server and vice versa. Client and server programs are connecting to the tun/tap interfaces which are initialized before. All the network traffic which is directed to the tun/tap interface in the client side will be sent to the server through real network.

One of the usages of the VPN is having different IP while communicating with the other hosts in the outside network. In order to make this happened, all the network traffic of the client machine should be directed to the tun/tap interface. However, if all the network traffic is directed to the tun/tap interface, server connection will be closed. Therefore, all the network traffic except through server IP one should be directed to tun/tap interface.

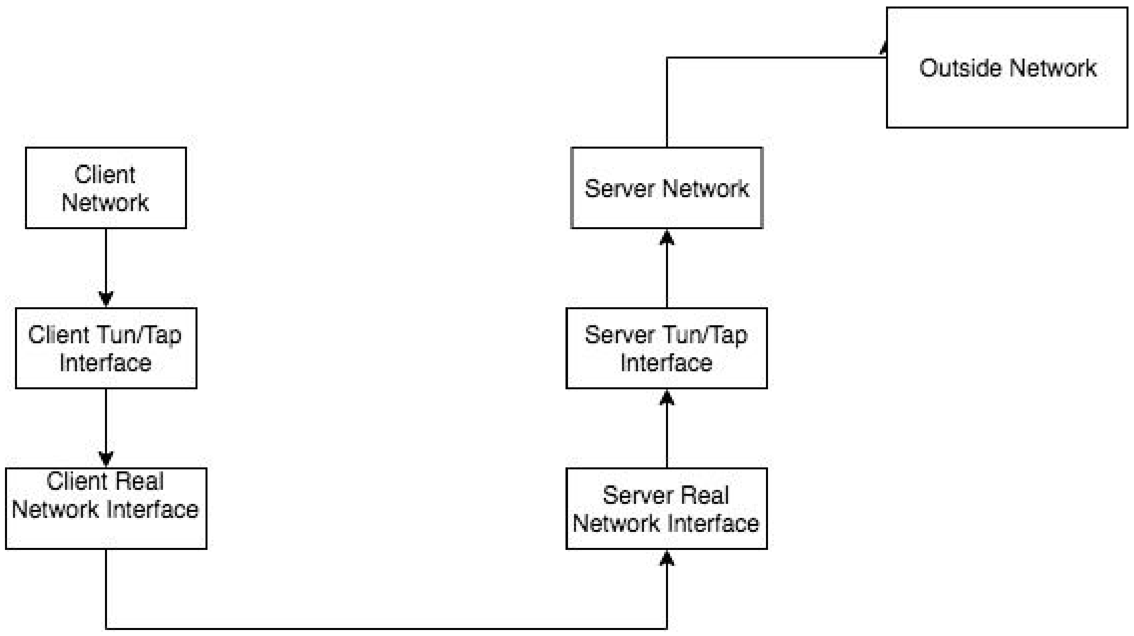
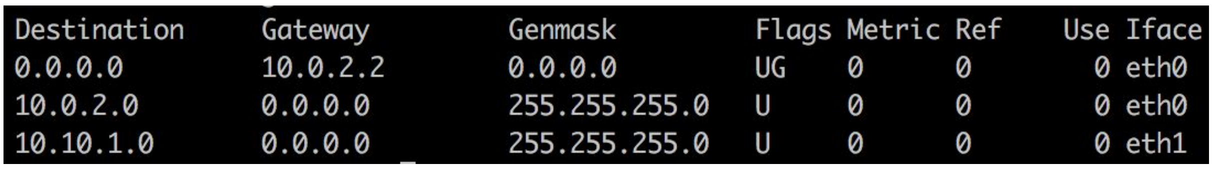


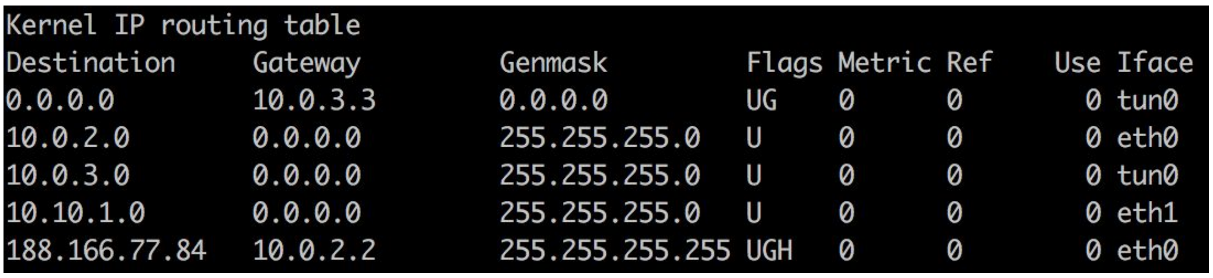
Figure 1: Demonstration of the tun/tap tunneling

Server IP traffic should continue through real network interface which is 'eth0' with our project config. Redirecting network traffic will be done with 'route' command in the Linux. 'route' command is used for configuration of the gateway address of the destination addresses. As stated in the above, all the network traffic except server traffic will be directed to the tun/tap interface IP. You can see before and after route table in the Figure 2 and Figure 3. As you can see in the Figure 2, all the network traffic which is noted with 0.0.0.0 is directed to the eth0 interface with IP address 10.0.2.2 which is gateway address to outside network. After configuration with the 'route' command, all the network traffic is directed to the tun0 interface with local IP 10.0.3.3. However, there is one more entry which is network traffic to 188.166.77.84 is directed to the eth0 interface. As you can guess, 188.166.77.84 is server IP address and in order to continue the tunnel connection with the server, client machine continues to connect server IP through eth0 interface which connects client to the outside network. There will be made some configuration on the server too. All the traffic which coming through tun/tap interface should be forwarded to the 'eth0' interface in order to send request of the client to the outside word. With this way, all network traffic of the client will be through server and outside world will see server IP instead of client IP. Forwarding from tun/tap interface to eth0 interface will be done using 'iptables'[8] command.
Scenario which is described on the above is used for accessing outside world with the different IP. VPN can be used other than the specified purpose too. It is just one of the usages of the VPN.



```
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.2.2        0.0.0.0         UG    0      0        0 eth0
10.0.2.0        0.0.0.0         255.255.255.0   U     0      0        0 eth0
10.10.1.0       0.0.0.0         255.255.255.0   U     0      0        0 eth1
```

Figure 2: Route table before configuration



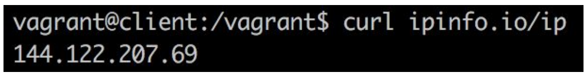
```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.3.3        0.0.0.0         UG    0      0        0 tun0
10.0.2.0        0.0.0.0         255.255.255.0   U     0      0        0 eth0
10.0.3.0        0.0.0.0         255.255.255.0   U     0      0        0 tun0
10.10.1.0       0.0.0.0         255.255.255.0   U     0      0        0 eth1
188.166.77.84   10.0.2.2        255.255.255.255 UGH   0      0        0 eth0
```

Figure 3: Route table after configuration

All the above configurations are done with the shell scripts which is written for these specific purposes. Essential implementation of the VPN is made on the server and client program. As stated on the above, server program listens for incoming connection requests from the client and if there is one, starts new thread and communication between server and client continues in this thread in the server side. Both client side and server side create file descriptor for reading from tun/tap interface and writing to tun/tap interface. Both network socket and tun/tap descriptor are started to listen and these interfaces are checked with using 'select'[9] functions whether any waiting packets in these interfaces. In case of new packet arriving in the one of the interface, blocking 'select' function notifies the program and program continue to execution according to the interface which packet arrived. If packet arrived through real network interface which 'eth0' in our project, it should be directed to the tun/tap interface because packets which arrived from real network interface have to be VPN packet therefore should be directed to the tun/tap interface. Packets which is directed to the tun/tap interface will be directed to the respectective userspace program by kernel as it came from the real network interface. If packet arrived through tun/tap interface, it should be directed to the real network interface. With this way, VPN connection will be provided because any userspace program which sending its packet to the tun/tap interface will be redirected to the server IP address and with this way, all the traffic of the user will be through server IP.

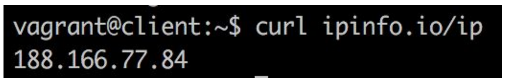
As stated in the 'Approach' part, second essential part of the VPN is encrypted communication between server and client. Encryption is provided with the help of DTLS implementation of the OpenSSL. The DTLS protocol provides privacy of communication for datagram protocols. The DTLS protocol is based on the TLS protocol therefore SSL key and SSL certificate should be generated for secure communication. Keys and certificates are generated with the help of OpenSSL. Generated keys and certificates will be given necessary methods in the client and server. After necessary initializations are done in the client and server programs, file descriptor of the real network interface sockets will be fed to the SSL object which is created before. Newly generated SSL objects will be used for secure communication between client and server. All the network traffic through newly generated SSL objects will be encrypted.

Evaluation:



```
vagrant@client:/vagrant$ curl ipinfo.io/ip
144.122.207.69
```

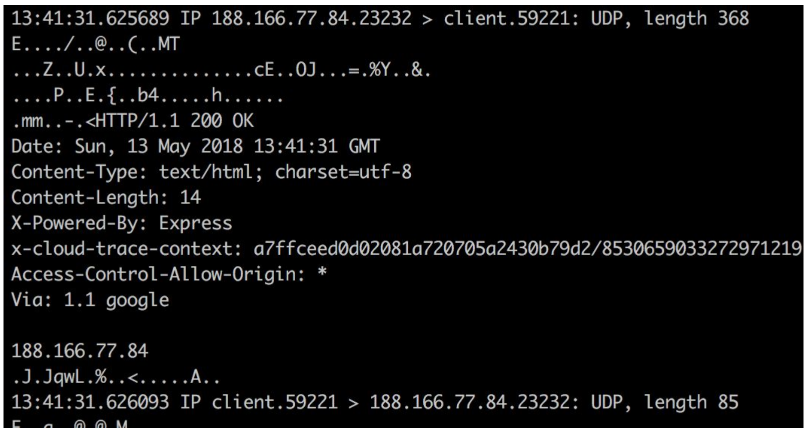
Figure 4: Public IP address of the client before VPN connection



```
vagrant@client:~$ curl ipinfo.io/ip
188.166.77.84
```

Figure 5: Public IP address of the client after VPN connection

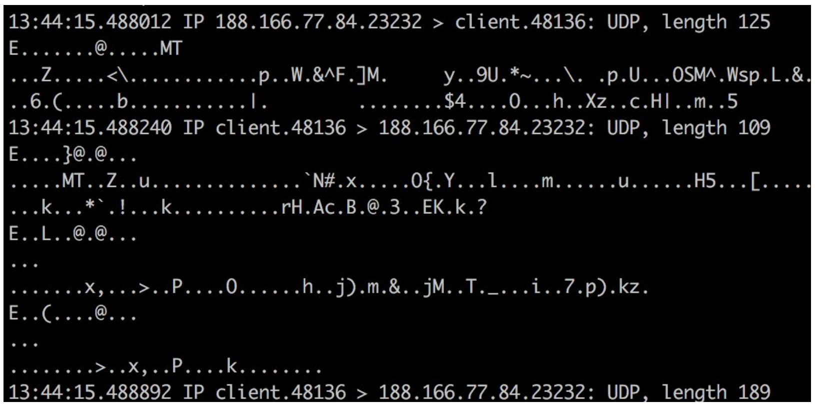
As we stated before, there are two essential purposes of the VPN. One of them is creating virtual network between host machines of the VPN users. Second of them is that communication between the host machines in this virtual network should be encrypted.
As can be seen in the Figure 4 and Figure 5, client's network traffic will go through VPN server after initializing the VPN client and server and making the necessary configurations. This shows that client and server behave like they are in the same private network.



```
13:41:31.625689 IP 188.166.77.84.23232 > client.59221: UDP, length 368
E..../..@..(..MT
...Z..U.x..............cE..OJ...=.%Y..&.
....P..E.{..b4.....h......
.mm..-.<HTTP/1.1 200 OK
Date: Sun, 13 May 2018 13:41:31 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 14
X-Powered-By: Express
x-cloud-trace-context: a7ffceed0d02081a720705a2430b79d2/8530659033272971219
Access-Control-Allow-Origin: *
Via: 1.1 google

188.166.77.84
.J.JqwL.%..<.....A..
13:41:31.626093 IP client.59221 > 188.166.77.84.23232: UDP, length 85
```

Figure 6: Screenshot of the tcpdump while sniffing the communication between server and client without encryption



```
13:44:15.488012 IP 188.166.77.84.23232 > client.48136: UDP, length 125
E.......@.....MT
...Z.....<\............p..W.&^F.]M.     y..9U.*~...\. .p.U...OSM^.Wsp.L.&.
..6.(.....b...........|.         ........$4....O...h..Xz..c.H|..m..5
13:44:15.488240 IP client.48136 > 188.166.77.84.23232: UDP, length 109
E....}@.@...
.....MT..Z..u..............`N#.x.....O{.Y...l....m......u......H5...[.....
...k...*`.!...k..........rH.Ac.B.@.3..EK.k.?
E..L..@.@...
...
.......x,...>..P....O......h..j).m.&..jM..T._...i..7.p).kz.
E..(....@...
...
........>..x,..P....k........
13:44:15.488892 IP client.48136 > 188.166.77.84.23232: UDP, length 189
```

Figure 7: Screenshot of the tcpdump while sniffing the communication between server and client with encryption

Communication between server and client is analyzed with tcpdump in order to show communication channel is encrypted. As you can see in the Figure 6, human readable plaintext header are captured from the communication channel between client and server when communication channel is not encrypted. However, when channel was started to encrypted with DTLS, no plaintext header in the tcpdump is observed. This shows that communication channel between client and server is started to encrypt.

Conclusion:

In this project, we have tried to implement proof-of-concept VPN solution for Linux. There are still major missing features in this project as stated in the 'Approach' part. However, it contains most essential components of a VPN such as tunneling and encrypted communication between client and server and can be used as learning material.

[1]: https://censorbib.nymity.ch/pdf/Wang2014a.pdf
[2]: https://censorbib.nymity.ch/pdf/Wang2014a.pdf
[3]: http://backreference.org/2010/03/26/tuntap-interface-tutorial/
[4]: http://man7.org/linux/man-pages/man8/route.8.html
[5]: http://sites.inka.de/bigred/devel/tcp-tcp.html
[6]: https://tools.ietf.org/html/rfc6347
[7]: https://linux.die.net/man/8/ip
[8]: http://ipset.netfilter.org/iptables.man.html
[9]: http://man7.org/linux/man-pages/man2/select.2.html