# Tunneling UDP packets through SSH to improve VoIP security

by

*Sameer Jain*
*Saeesh Sinai Kakodkar*
*Viren Sinai Nadkarni*
*Pranav Prem*

Dissertation submitted in partial fulfillment of the requirements for the degree of
Bachelor of Engineering
Computer Engineering


Department of Computer Engineering
Goa College of Engineering, Farmagudi, Goa

Goa University
(2014)

# *Tunneling UDP packets through SSH to improve VoIP security*

# Goa College of Engineering

Farmagudi, Ponda - Goa 403401

## "Tunneling UDP packets through SSH to improve VoIP security"

*Bona fide record of work done by*


| | |
|---|---|
| **Sameer Jain** | **(0263)** |
| **Saeesh Rajendra Sinai Kakodkar** | **(0274)** |
| **Viren Vijaikumar Sinai Nadkarni** | **(0276)** |
| **Pranav Prem** | **(0256)** |


*Dissertation submitted in partial fulfillment of requirements for the degree of*

**Bachelor of Engineering**
**(Computer Engineering)**

**Goa University**
**(June 2014)**

_____ _____

Project Guide Head of Dept.

_____ _____

Internal Examiner External Examiner

# Acknowledgement

This project is the result of the dedication and encouragement of many individuals. Our sincere and heartfelt appreciation goes to all of them.

Our thanks go to the Principal, Prof. V. N. Shet and Head of Department, J. A. Laxminarayana for giving us this opportunity to take up this opportunity to take up this project. We are thankful to our project guide, Prof. Nagraj K. Vernekar for all the guidance, support and encouragement provided to us in the course of this project work so far. We also thank Prof. Siddhesh Salelkar for giving us initial advice regarding the project.

# Table of Contents

*page intentionally left blank*

# i. Abstract

Conventionally a TCP network setup is considered better than a UDP network setup because TCP offers connection establishment and acknowledgements ensuring delivery of every packet. However in certain situations a UDP connection outperforms a TCP connection such as that of real time data transfer in the scenario of a VoIP. This project establishes that the choice between a TCP and a UDP connection is application specific and then provides a means of securing UDP transmission by creating a hybrid tunnel that uses connection establishment principles of a TCP network and transmitting UDP packets over established tunnel with enforced encryption similar to that of SSH at the application layer itself and no retransmission or ordering. This combines some of the security and integrity features of TCP with the speed of UDP. This secured UDP transmission is then tested using a VoIP setup.

# ii. List of Figures

# iii. List of Tables

*page intentionally left blank*

# *I. Introduction*

# I. Introduction

When doing sensitive communication over the internet, three basic requirements need to be assessed:

- Confidentiality
  Is necessary to protect the information that is being sent between the communicating parties. A strong encryption algorithm is necessary to prevent an eavesdropper in reading confidential information in clear text.
- Integrity
  It is important to verify that the received information is the same as it was when it was sent to you. In the digital world this is solved through digital signatures and hash functions.
- Authentication
  It is necessary to verify that the information has come from whom it is supposed to, and that it is received by who is supposed to receive it, i.e. mutual authentication.

All these requirements are addressed by SSH, SSL/TLS and IPsec. Each of these solutions operate at different levels of the Internet/OSI model.

| | |
|---|---|
| Application | SSH |
| Transport | SSL/TLS |
| Network | IPsec |
| Physical | |

*Table 1 - Protocols operating over various layers*

In the TCP networking model, with it's four layers, application, transport, network and physical layer, the different technologies IPsec, TLS/SSL and SSH has its individual place as described in the above figure.

SSH fits in at the top of the model at the application layer. This makes SSH an application by nature and work beside other network applications like ftp, http and others. SSH can be used in a port-forwarding mode to create a tunnel for other applications.

TLS/SSL provides security for the transport layer. TLS/SSL is not a single application like SSH, but provide security through implementation into applications. SSL was designed by Netscape with HTTP usage in mind. TLS is the latest version of the SSL technology. IPsec provides security at the IP packet layer; it is not integrated at higher levels like TLS/SSL. IPsec is a network-level protocol incorporated into servers and/or clients, e.g. into a router, a firewall or into an operating systems' kernel.

It is important to remember that there is no interoperability between SSH, TLS/SSL and IPsec, they all operate in different levels in the TCP model and are designed with different uses in mind.

UDP (User Datagram Protocol) is a communications protocol that offers a limited amount of service when messages are exchanged between computers in a network that uses the Internet Protocol (IP) to get packets of data (called datagrams in this case) from source to destination. It provides only a single service over IP which is a port number to distinguish between user requests. UDP doesn't provide sequencing of the packets that the data arrives in. Neither does it provide any handshaking or connection establishment. While UDP is at transport layer in the OSI schema, it becomes the responsibility of the application at application layer to make sure comprehensible data is received. TCP (Transmission Control Protocol) is a set of communications protocol used along with the Internet Protocol(IP) to send data in the form of message units between computers over the Internet . TCP divides the message into packets and provides the services (to IP) of ensuring the delivery of every packet by using connection establishment and handshaking signals to acknowledge the delivery of every packet. TCP works in the transport layer and additionally maintains the order of receiving packets at the destination. It establishes a connection and maintains it until such time that the messages to be exchanged by the application programs have been exchanged.

From this it is fairly evident why TCP is considered a standard while UDP is used for more trivial tasks that require less security. This misconception is stressed on in many cases such as the usage of UDP in a file transfer protocol that is labeled Trivial File Transfer Protocol (TFTP). TFTP is extremely limited and provides no authentication. It is known for its simplicity.

With the increase of bandwidth and the revolution of handheld devices, real time media transfer has increased significantly over the last few years. This shift has led to focus returning to UDP transmissions for communication.

In early April 2014, a critical bug that manifested in SSL/TLS implementation was discovered. The bug was named "Heartbleed". The bug was a buffer underflow issue that came from 'heartbeat' signals that are sent in SSL/TLS to check if a server is alive. Each 'heartbeat' involves data to be returned and the size of data to be returned if server is alive. The server is to read the size and return the requested data of that size if it is alive. The problem arises when the size of the data requested to be returned is larger

than the data supplied to be returned. According to SSL/TLS mechanism, the server continues sending data from the input buffer until the amount of data asked for is provided. This data could contain session data of other people logged into the server causing major breach of security.

This problem is a clear example of how security that spans through more than one layer of the OSI architecture is prone to serious flaws because of inter-layer coordination. Hence, the paper proposes a system that has all its security implemented in the application layer itself. All packets are encrypted in all other layers as well as shielded by the tunnel which maintains the connection and makes sure there is no leak of information.

# *II. Motivation*

# a. Analysis of Popular Chat Clients

In order to explain the need for a more secure system for voice communication, we will take a look at existing solutions available for use.

## Skype

Skype is a peer-to-peer VoIP client developed by KaZaa in 2003. Skype claims that it can work almost seamlessly across NATs and firewalls and has better voice quality than the MSN and Yahoo, IM applications. It claims to encrypt calls end-to-end, and store user information in a decentralized fashion. Skype also supports instant messaging and conferencing. The communication protocol and source code of Skype are undisclosed. It uses high strength encryption and random port number selection, which render the traditional flow identification solutions invalid.

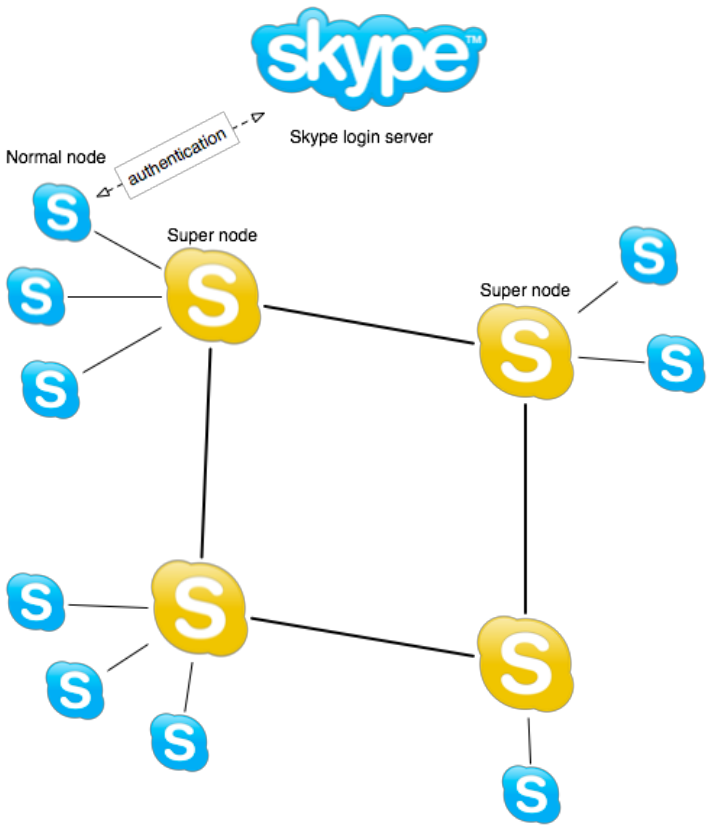



*Figure 1 - Skype architecture*

Skype is currently the largest peer-to-peer VoIP application. Skype nodes join the skype overlay network on successful authentication with the centralised Skype Login server. Skype nodes are either

"normal" nodes or "super nodes." Super nodes act as relays and help establish connectivity between normal nodes, should there be any issues in direct connectivity.

Though Skype has not publicly disclosed its protocol, there are numerous research papers that infer details through experiments.

We analyzed Skype functions such as login, NAT and firewall traversal, call establishment, media transfer, codecs, and conferencing using wireshark (a packet sniffer). On analysis we could conclude that Skype conversation could easily be intercepted and the so publicized claims about Skype's seamless security are untrue.

# Yahoo! Messenger

Yahoo Messenger is an advertisement-supported instant messaging client and associated protocol provided by Yahoo. Yahoo messenger uses YMSG protocol. YMSG uses a client-server architecture for normal operations as well as voice-chat service. It can support multiple users within the same voice-chat session and each user can specify their own voice specification with the central voice server based on their network speed.

YMSG is infamous for its history of security flaws. YMSG voice traffic is routed through a centralized voice-chat server. Clients first contact a setup server "vc.yahoo.com" which then redirects the client to the voice-chat hosting server. YMSG encrypts only the login authentication details, not the actual data.

# Google Talk

Google Talk is an instant messaging service that provides both text and voice communication. The instant messaging service is colloquially known as "gtalk" or "gchat" to its users, although Google does not endorse this name.

Google Talk is also the name of the client applications previously offered by Google to use the service. Google Talk applications were available for Microsoft Windows (XP, Server 2003, Vista, and Windows 7), Android, Blackberry, and Chrome OS operating systems. Google Talk web app had also been previously available for Android. In 2013, Google replaced Google Talk client software offerings with those of Google+ Hangouts.
Because the Google Talk servers communicate with clients using an open protocol, Extensible Messaging and Presence Protocol XMPP, the service can also be used with any other client that

supports XMPP. Such clients are available for a number of operating systems not supported by the Google Talk client.

Google Talk used extensions to XMPP for voice/video signaling and peer-to-peer communication. As of August 2012, Google Talk's implementation differs slightly from the draft XMPP Jingle specifications. In 2012, Google had stated that an update was under way. Since May 2013, support for the XMPP instant messaging protocol is dropped.

The connection between the Google Talk client and the Google Talk server is encrypted, except when using Gmail's chat over HTTP, a federated network that doesn't support encryption, or when using a proxy like IMLogic. End-to-end messages are unencrypted. Google has revealed plans to add support for chat and call encryption in a future release. Some XMPP clients natively support encryption with Google Talk's servers. It is possible to have end-to-end encryption over the Google Talk network using OTR (off-the-record) encryption using other chat clients like Adium (for Mac) or Pidgin (for Linux and Windows).

## MSN Messenger

Windows Live Messenger (formerly MSN Messenger) is a deprecated instant messaging client developed by Microsoft for various platforms. It connected to the Microsoft Messenger service while also having compatibility with Yahoo! Messenger and Facebook Messenger. The client was first released as MSN Messenger in 1999 and was marketed under the MSN branding until 2005 when it was rebranded under Windows Live and has since been officially known by its present name, although its previous name was still used colloquially by most of its users.

Following the acquisition of Skype Technologies, Microsoft began to promote Skype and introduced the ability to merge their Skype accounts with a Microsoft account, allowing users to communicate with Messenger contacts via Skype, which had additional features and a wider user base.

Windows Live Messenger used the Microsoft Notification Protocol (MSNP) over TCP (and optionally over HTTP to deal with proxies) to connect to Microsoft Messenger service—a service offered on port 1863 of "messenger.hotmail.com.". The protocol is not completely secret; Microsoft disclosed version 2 (MSNP2) to developers in 1999 in an Internet Draft, but never released versions 8 or higher to the public. The Messenger service servers currently only accept protocol versions from 8 and higher, so the syntax of new commands sent from versions 8 and higher is only known by using packet sniffers like Wireshark. This has been an easy task because – in comparison to many other modern instant messaging protocols, such as XMPP – the Microsoft Notification Protocol does not provide any encryption and everything can be captured easily using packet sniffers. The lack of proper

encryption also makes wiretapping friend lists and personal conversations a trivial task, especially in unencrypted public Wi-Fi networks.

# Conclusion

- Achieving end-to-end security in a VoIP session is a challenging task
- Though every popular VoIP service has their own strength, the security they offer is way below the ‘expected’ and ‘publicised’ levels
- No VoIP service has capabilities to stop unprivileged participants of the network to perform traffic analysis and determine when one user calls another user.
- There is scope for more improvement in VoIP network security.

# *III. Literature Survey*

# a. TCP/IP Suite

The TCP/IP Suite is the networking model and a set of communications protocols used for the Internet and similar networks. Its most important protocols, the Transmission Control Protocol (TCP) and the Internet Protocol (IP), were the first networking protocols defined in this standard.

TCP/IP provides end-to-end connectivity specifying how data should be formatted, addressed, transmitted, routed and received at the destination. This functionality has been organized into four abstraction layers which are used to sort all related protocols according to the scope of networking involved. From lowest to highest, the layers are the link layer, containing communication technologies for a single network segment (link), the internet layer, connecting independent networks, thus establishing internetworking, the transport layer handling process-to-process communication, and the application layer, which interfaces to the user and provides support services.

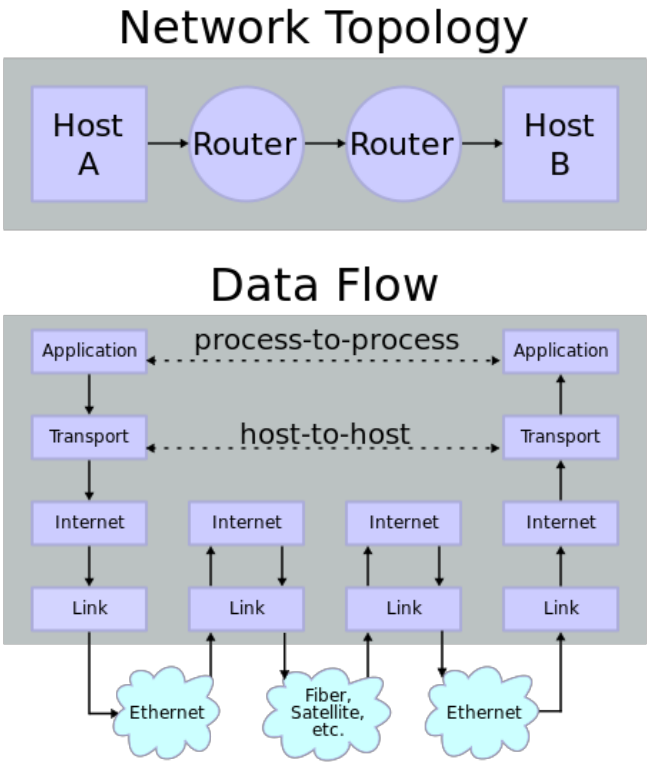



*Figure 2 - TCP/IP Model*

The model uses encapsulation to provide abstraction of protocols and services. Encapsulation is usually aligned with the division of the protocol suite into layers of general functionality. In general, an

application (the highest level of the model) uses a set of protocols to send its data down the layers, being further encapsulated at each level.

The layers of the protocol suite near the top are logically closer to the user application, while those near the bottom are logically closer to the physical transmission of the data. Viewing layers as providing or consuming a service is a method of abstraction to isolate upper layer protocols from the details of transmitting bits over, for example, Ethernet and collision detection, while the lower layers avoid having to know the details of each and every application and its protocol.

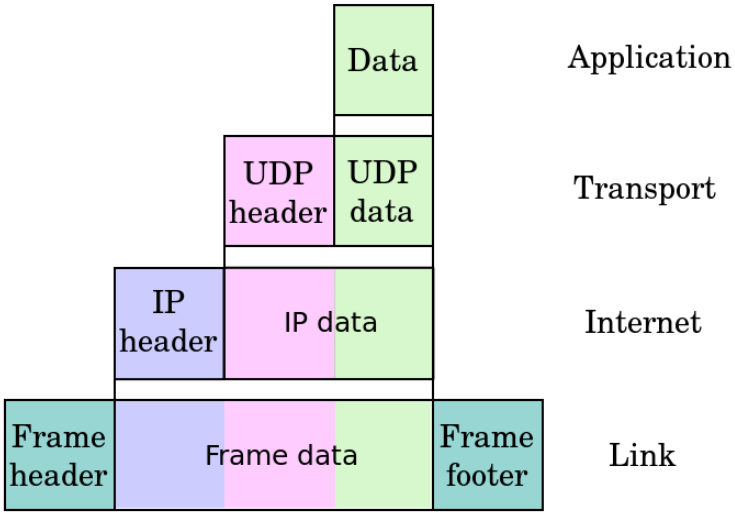



*Figure 3 - Encapsulation*

# b. SIP/VoIP Suite

VoIP (Voice over Internet Protocol) is a set of technologies which allows voice sessions over Internet Protocol networks. Rather than using Public Switched Telephone Network, VoIP provisions communication services over Internet.

The steps involved in VoIP are:

- signalling
- channel setup
- digitisation of the analogue voice signals
- encoding
- packetisation
- transmission

The process is similar to traditional phone systems with few differences. VoIP, being operated over Packet Switched Network, are subject to latency and jitter. Hence to provide good Quality of Service, session controlling and signalling protocols are employed.

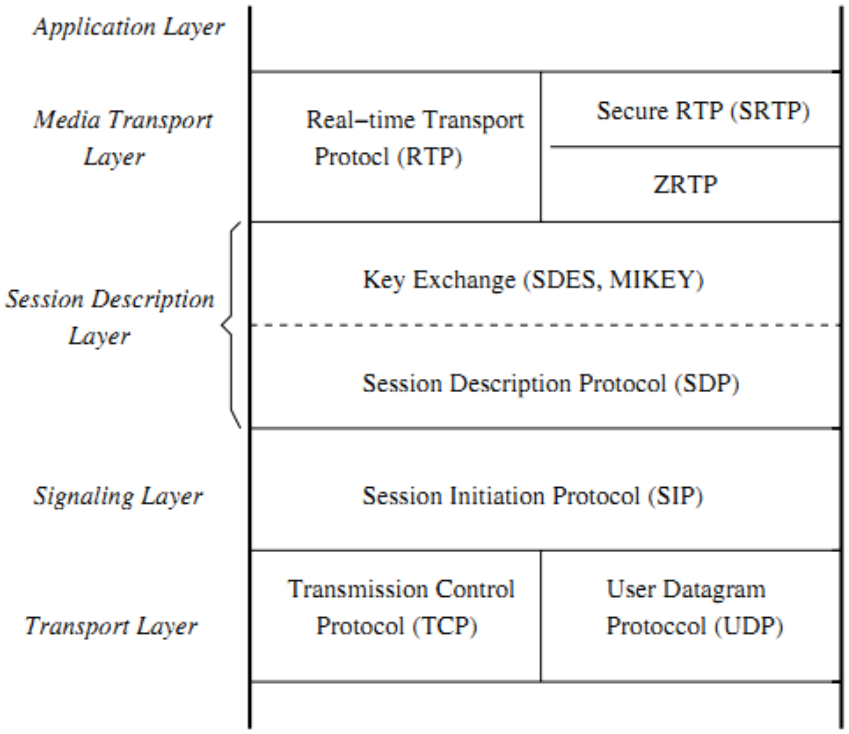



*Figure 4 - VoIP protocol stack*

The VoIP protocol stack is shown in figure. For the purposes of analysis, it is divided into four layers: signaling, session description, key exchange and secure media (data) transport. This division is quite natural, since each layer is typically implemented by a separate protocol.

Signaling is an application-layer control mechanism used for creating, modifying and terminating VoIP sessions with one or more participants. Signaling protocols include Session Initiation Protocol (SIP), H.323 and MGCP. Session description protocols such as SDP are used for initiating multimedia and other sessions, and often include key exchange as a sub-protocol. Key exchange protocols are intended to provide a cryptographically secure way of establishing secret session keys between two or more participants in an untrusted environment.

This is the fundamental building block in secure session establishment. Security of the media transport layer—the layer in which the actual voice datagrams are transmitted—depends on the secrecy of session keys and authentication of session participants. Since the established key is typically used in a symmetric encryption scheme, key secrecy requires that nobody other than the legitimate session participants be able to distinguish it from a random bitstring. Authentication requires that, after the key exchange protocol successfully completes, the participants' respective views of sent and received messages must match.

# Security Issues

A protocol may be secure when executed in isolation, but the composition of protocols in different layers may be insecure. Moreover, a protocol may make assumptions about
another protocol that the latter does not satisfy.

The first is a replay attack on SDES key exchange which causes SRTP to use the same keystream in multiple sessions, thus allowing the attacker to remove encryption from SRTP-protected data streams. The second is an attack on ZRTP caused by unauthenticated user IDs, which allows the attacker to disable authentication mechanisms and either trick a ZRTP participant into establishing a shared key with the attacker, or cause the protocol to terminate prematurely. The third is a "certification" issue: due to the lack of proper randomness extraction in MIKEY key derivation, MIKEY cannot be proved cryptographically secure.

The User Datagram Protocol is used to transmit voice data over a VoIP network. UDP is a 'send and forget' protocol with no requirement for the transmitter to retain sent packets should there be a transmission or reception error. If the transmitter did retain sent packets, the flow of real-time voice would be adversely affected by a request for retransmission or by the retransmission itself; especially if there is a long path between transmitter and receiver.

# c. SSH

Secure Shell is a network protocol for secure data communication, remote command-line login, remote command execution, and other secure network services between two networked computers that connects, via a secure channel over an insecure network, a server and a client (running SSH server and SSH client programs, respectively). SSH is typically used to log into a remote machine and execute commands, but it also supports tunneling and forwarding TCP ports. It can transfer files using the associated SSH file transfer (SFTP) or secure copy (SCP) protocols. It uses the client-server model.

SSH is a very inexpensive, in fact it is free for non-commercial use and costs little for commercial use. SSH is available in two versions SSH-1 and SSH-2. SSH-2 is the latest and most secure version. SSH-1 is still very popular (SSH-1 can be found as GPL license for all major platforms) but have some limitations in features and it has some dangerous security issues, e.g. uses a CRC for integrity protection which is not secure. SSH have high availability and runs on almost every platform. SSH-2 support a lot of encryption algorithms like 3DES, IDEA, Blowfish, Twofish and Cast.

A major benefit with SSH is that it is possible to tunnel TCP based applications through SSH, e.g. email protocols, programming tools and even business applications like Oracle. To most users SSH appears to be terminal emulator similar to Telnet. The users do not see the encryption and therefore the security is transparent for the user. For system administrators SSH is a popular remote administration platform.

TLS/SSL and IPsec is almost totally transparent to use, but SSH is not, to use SSH you have to be logged on to user account to utilize the transport layer security. SSH is used for scripting applications, whereas TLS/SSL and IPsec is incorporated into applications and the TCP/IP stack. UDP and ICMP is also a problem with SSH. It is not possible to tunnel UDP or ICMP traffic directly.

## Merits of SSH over alternatives

First, SSH operates at Application layer of the TCP/IP model, and all previously described protocols operate at other layers. Operation at Application layer provides some advantages, such as avoiding encapsulation by making it work independently of other layers unlike SSL/TLS where application layer protocols have to interface with those in transport layer.

Main problems associated with encapsulation include security issues and processing efficiency. Security considerations require the design of proper protective mechanism avoiding protocol attacks attempted by non-friendly network nodes by providing incorrect cross-layer information in order to trigger certain

behavior. The problem with processing efficiency is related to the additional costs of the routers' hardware associated with cross-layer information processing.

Second, an existing protocol has been chosen instead of making a new one from scratch, which spares design and implementation hassles. SSH is also cross-platform, which leaves out porting issues.

Third, this is something that has never been tried before, and so it becomes more of a research oriented project that may show more potential in the later stages.

# d. Tunneling and Encapsulation

A tunnel is a mechanism used to ship a foreign protocol across a network that normally wouldn't support it. Tunneling protocols allow the use of, for example, IP to send another protocol in the "data" portion of the IP datagram. Most tunneling protocols operate at layer 4, which means they are implemented as a protocol that replaces something like TCP or UDP.

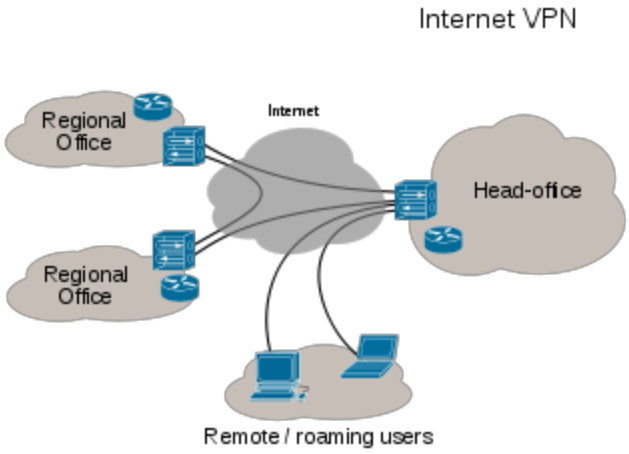



*Figure 5 - VPN tunneling*

VPN tunnels allow remote clients to tunnel into our network. This supports the previous notion of tunnels being used for "unsupported protocols," even though that may not be apparent.

A few interesting things to note about the VPN tunnel are: once your data hits the internal network it's already been unencrypted, and when your data is traversing the Internet there is extra "stuff" attached to the packet.

Unmentioned, but probably obvious, is that VPN protocols will also encrypt your data before transmission. It doesn't matter for understanding tunneling, but it's worth mentioning. Take notice that the encryption is not end-to-end, i.e. you and the server's communication are not truly secure. Surely it's secure from prying eyes between yourself and your work, but as soon as packets are shipped beyond the VPN server, they're once again unencrypted.

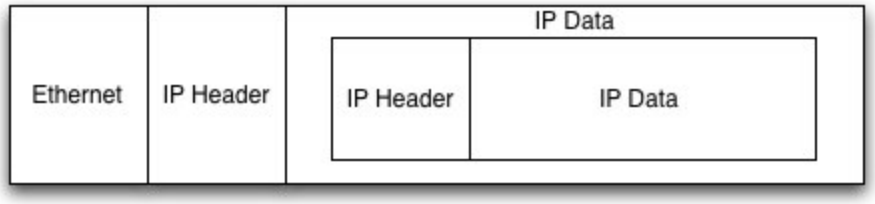



*Figure 6 - Format of a packet*

The data portion of an IP packet contains an entirely new IP packet. This works the same way as VPN tunnels, excluding the encryption. When the packet has the "extra header" on top of it, it cannot send as much data, because the first IP header uses up 20 bytes.

# e. Analysis of Existing Tunneling Solutions

## TOR network

TOR (The Onion Router) protocol is an internet security protocol that is used for complete anonymity over the world wide web. To connect to the TOR network, a TOR circuit is established with relays between a source and destination. In TOR routing, the intermediate relays themselves do not know the entire circuit. Only the source and destination are privy of that information. The data is encrypted at the source multiple times and passed into the relay circuit. Each relay in the circuit decrypts only the header to find out the address of the next relay in the circuit and then forwards the packet onwards with one less layer of security. The final exchange will involve plain text exchange between the last relay in the circuit and the destination. To avoid this, data is secured with SSL / TLS before entering the TOR circuit.

The TOR network uses a tunneling proxy to serve as the TOR tunnel and to maintain anonymity. SOCKS is the Internet Protocol used by TOR to maintain anonymity. It manages exchange of data between source and destination by routing data through a proxy server. SOCKS server proxies TCP connection to an arbitrary IP address and provides a means for UDP packets to be forwarded.

## VPN tunnel

VPN (Virtual Private Network) is the extension of a private network that includes links across shared or public networks such as the internet. VPN helps send packets from source to destination over a network in a way that it emulates a point to point link. This is done by creating a tunnel between source and destination.

In case of Windows VPN, Point to Point Tunneling Protocol (PPTP) is used to tunnel packets over the public network. Tunnel is established with both of the tunnel endpoints negotiating the configuration variables such as address assignments, encryption and comparison parameters. After the tunnel is established, data is sent. The tunnel uses a tunnel data transfer protocol to prepare the data for transfer. This involves appending tunnel data transfer protocol headers or encryption.

PPTP encapsulates Point to Point Protocol frames into IP datagrams for transmission over an IP based network. PPTP is described in RFC 2637 in IETF Database. It uses a TCP connection known as the PPTP control connection to create, maintain and terminate the tunnel. PPTP uses a modified version of

Generic Routing Encapsulation (GRE) to encapsulate PPP frames as tunneled data. The payloads of the encapsulated frames can be encrypted, compressed or both.

Additionally, L2TP (Layer Two Tunneling Protocol) is a combination of PPTP and Layer 2 Forwarding, a technology developed by Cisco systems. L2TP encapsulated PPP frames to be sent over IP, X.25, frame relay or ATM networks when sent over an IP network, L2TP frames are encapsulated as User Datagram Protocol (UDP) messages. L2TP can be used as a tunneling protocol over internet or over private intranets.

L2TP uses UDP messages over IP networks for both tunnel maintenance and Tunneled data. This creates the issue that L2TP tunnel maintenance and tunnel data have the same structure.

# *IV. Analysis and Software Requirement Specifications*

# a. Software Development Model

The development models are the various processes or methodologies that are being selected for the development of the project depending on the project's aims and goals. There are many development life cycle models that have been developed in order to achieve different required objectives. The models specify the various stages of the process and the order in which they are carried out.

The selection of model has very high impact on the testing that is carried out. It will define the what, where and when of our planned testing, influence regression testing and largely determines which test techniques to use.

## Rapid Prototyping

A rapid prototype is a working model that is functionally equivalent to a subset of the product.

The first step in the rapid-prototyping life-cycle model is to build a rapid prototype and let the client and future users interact and experiment with the rapid prototype. Once the client is satisfied that the rapid prototype indeed does most of what is required, the developers can draw up the specification document with some assurance that the product meets the client's real needs.

A major strength of the rapid-prototyping model is that the development of the product is essentially linear, proceeding from the rapid prototype to the delivered product. There are a number of reasons for this. First, the members of the development team use the rapid prototype to construct the specification document. Because the working rapid prototype has been validated through interaction with the client, it is reasonable to expect that the resulting specification document will be correct. Second, consider the design. Even though the rapid prototype has (quite rightly) been hurriedly assembled, the design team can gain insight from it—at worst it will be of the "how not to do it" variety. The prototype has gives some insights to the design team, even though it may reflect only partial functionality of the complete target product.

Once the product has been accepted by the client and installed, post delivery maintenance begins. Depending on the specific maintenance task that has to be performed, the cycle is re entered either at the requirements, analysis, design, or implementation phase.

An essential aspect of a rapid prototype is embodied in the word *rapid*. The developers should endeavor to construct the rapid prototype as rapidly as possible to speed up the software development process. After all, the sole use of the rapid prototype is to determine what the client's real needs are;

once this has been determined, the rapid prototype implementation is discarded but the lessons learned are retained and used in subsequent development phases. For this reason, the internal structure of the rapid prototype is not relevant.

Implementation comes next. In the rapid-prototyping model, the fact that a preliminary working version of the software product has already been built tends to lessen the need to repair the design during or after implementation. The prototype has given some insights to the design team, even though it may reflect only partial functionality of the complete target product.

Once the product has been accepted by the client and installed, post delivery maintenance begins. Depending on the specific maintenance task that has to be performed, the cycle is re-entered either at the requirements, analysis, design, or implementation phase.

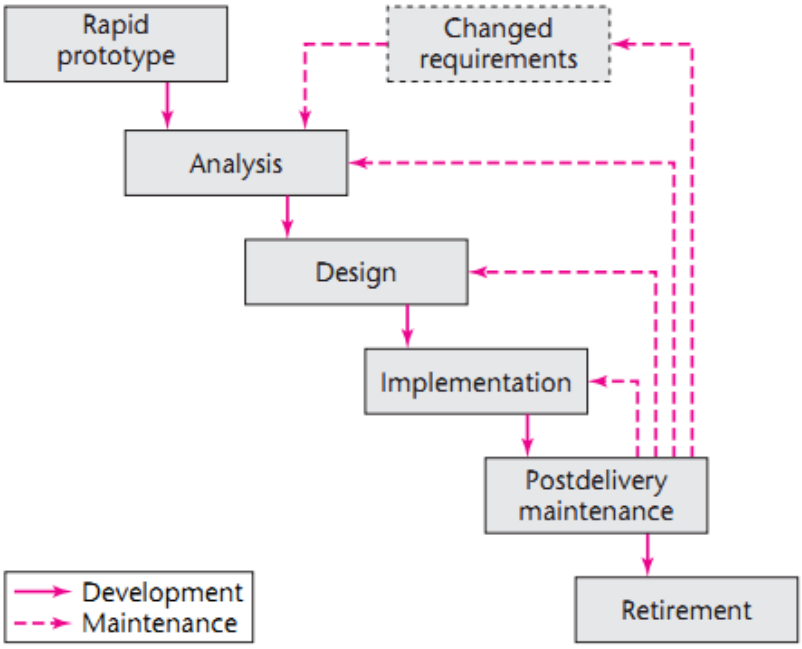



*Figure 7 - Rapid Prototyping Development Model*

Considering the fact that project is testing oriented, we will be using Rapid Prototyping life cycle model in developing our software.

A modified Synchronise and Stabilize model will be used to combine the work done by every project member at the end of every session. Git will be used for this purpose.

# b. Initial Prototype

Because a Rapid Prototype development is being followed, an initial prototype capable of carrying out basic VoIP function was developed. It was a very basic program, capable of switching between UDP and TCP modes of transmission. The prototype was developed for sole purpose of analysing the performance effects of using TCP versus UDP in VoIP. The Microsoft Network Emulator for Windows toolkit was used to simulate real life network conditions. The results of the analysis are produced in following section.

## Observations

The initial prototype was tested using the network tool to obtain the following table and graph with various packet loss condition simulations in one way traffic for UDP and TCP.

| Packet loss probability | Packets sent | Packets received | Received/Sent |
|---|---|---|---|
| 0.0 | 428 | 430 | 1.00467289719626 |
| 0.1 | 428 | 374 | 0.87383177570093 |
| 0.2 | 428 | 352 | 0.82242990654205 |
| 0.3 | 428 | 246 | 0.57476635514018 |
| 0.4 | 428 | 249 | 0.58177570093457 |
| 0.5 | 428 | 205 | 0.47897196261682 |
| 0.6 | 428 | 169 | 0.39485981308411 |
| 0.7 | 428 | 121 | 0.28271028037383 |
| 0.8 | 428 | 92 | 0.21495327102803 |
| 0.9 | 428 | 34 | 0.07943925233644 |
| 1.0 | 428 | 0 | 0.00000000000000 |

*Table 2- Packet loss*

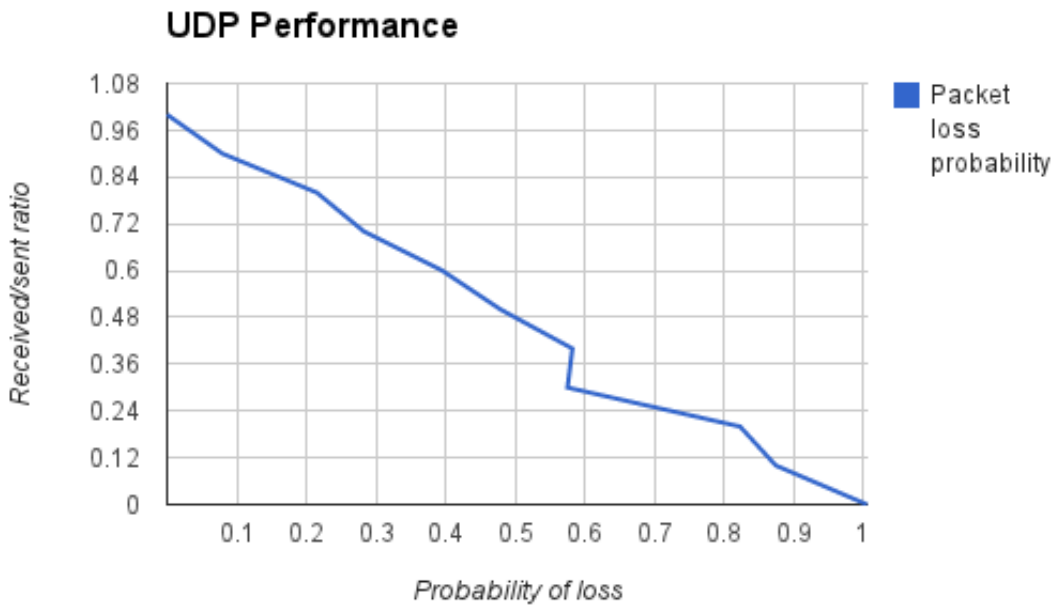



*Figure 8 - Graph of Received/Sent ratio vs. Probability of Loss*

As can be seen from the above plot, the line is somewhat linear, with a sudden dip when packet loss probability is set between 0.5 and 0.6.

When similar conditions were tested for TCP, it was found that there was an increasing time delay between the sending and receiving of packets but all packets were eventually received.

| **TCP** | **UDP** |
|---|---|
| Broken streams of data exchange | Steady stream of data exchange |
| Priority for next packet to be received | No priorities |
| All packets received | Packets are lost |
| Used when integrity of data is more important than a steady stream of data to be transmitted. When an old packet that has not been received is more important than a new packet of the same. | Used where real time data exchange is needed. When a new packet is more important than an old one that has not been received |

*Table 3: Comparison of UDP with TCP*

# c. Software Requirement Specifications

As is clearly seen in the previous section, there are specific and documented scenarios where a UDP connection is prefered over that of TCP. For such cases, the system proposes a new means of traffic by using a hybrid tunnel. This tunnel will combine the key features of TCP that are essential for communication such as a reasonable amount of reliability in transmission and a steady stream of data with no retransmission to cause any stutter or lag in cases where a UDP connection is important for the application.

Another important requirement in question is the security of data to be transmitted. The proposed system must effectively enforce security i.e. it must provide a secure channel without greatly impacting the efficiency of transmission with excessive overheads on the communication channel.

The system proposes to accomplish these tasks using a tunnel mechanism that combines the benefit of TCP and UDP along with security by encrypting every packet that is transmitted through the said tunnel.

# *V. Design*

# a. Proposed Hybrid Tunnel

Once established that datagram traffic was essential, a means of securing datagram communication is prepared which involves a tunneling architecture. According to the scheme, a tunnel is established using TCP mechanism involving handshakes and acknowledgements. This tunnel, once established is then made to transmit UDP packets without any mechanism for retransmission of lost packets or ordering as per UDP style. The tunnel establishes the ports for each channel before transmission and then enforces strict encryption on every UDP packet that was transmitted through it thereby ensuring security (in an SSH fashion).

The TCP tunnel is prepared, as per the diagram, to transmit datagrams. The diagram shows unidirectional traffic. The bi-directional implementation involves the same process in the opposite direction. The tunnel itself is capable of handling bidirectional traffic.

The architecture involves multiple clients exchanging UDP data connected to one station. The station listens on a preset port for incoming data. Any data transmitted onto the said port is then tunneled via the established tunnel to a preset destination that is set up with the tunnel. From this destination, the address of the received packets are then resolved and retransmitted to the ports excepting the data.

The design involves a server thread for each port on client side (supporting UDP) per client/port on server side (with NAT). Client side server sends everything to the same forward destination with different source forwarding port to same forward destination port on server side. UDP packets are encapsulated appropriately before transmission.
Thus the system takes advantage of TCP services while establishing the tunnel to make sure ports are available and then uses the merits of UDP services during communication to ensure speedy delivery of data over the tunnel.

This system also provides the additional advantage of allowing applications to decide the kind of encryption they wish to enforce by using conventional encryption mechanisms such as 3DES, IDEA, Blowfish, Twofish or CAST. Security implemented on the application layer implies there will be no lapse of security arising from the clash of functions between different OSI layers in a multilayer security implementation like SRTP and SSL/TLS.

***Comparison with TOR:***
In the established tunnel implementation, a means of creating a TCP connection and forwarding UDP packets to various ports is used by NAT translating packet headers to send them to the right ports. This implements additional anonymity over the tunnel, similar to the TOR protocol. Unlike TOR however, there is no relay circuit and hence even if someone were to latch on to the socket on the

client side, the proxy would disable metadata analysis attacks and there would be no chance of him obtaining plain text like in case of TOR.

***Comparison with L2TP and PPTP:***

Like PPTP , the system also establishes a TCP connection to create, maintain and terminate the tunnel. Like L2TP, the packets passing through the tunnel are encapsulated and encrypted into an optimized hybrid for transmission of UDP packets in the tunnel. Unlike PPTP, the system works for better for a UDP transmission and unlike L2TP, the tunnel maintenance is done by a TCP connection which have a different encapsulated structure thereby making it more secure.

To summarize, the hybrid tunnel model being used, combines a proxy solution similar to SOCKS5 used by TOR network, the tunnel establishment technique used by PPTP and the transmission and encapsulation technique used by L2TP to create an optimized hybrid for secure real time data transmission.

The various tunneling solutions are summarised below:

| **TOR** | **PPTP** | **L2TP** | **Proposed model** |
|---|---|---|---|
| Uses Socks5 proxy system with a proxy tunnel involving all data transmitted through a proxy server | No proxy in implementation | No proxy in implementation | Address translation at sending and receiving end to implement proxy on data. |
| Uses a relay circuit with layered encryption on all packets such that last hop transfers plain text unless text is encrypted before entering TOR circuit | Encryption is dependent on user. No relays | Encryption is dependent on user. No relays. | All data through tunnel is encrypted. No possibility of plain text being transferred. No relays. |
| Tunnel established with proxy server as medium using SOCKS5 proxy mechanism | TCP tunnel establishment and maintenance | UDP tunnel establishment and maintenance. | TCP tunnel establishment and maintenance |
| TCP and UDP traffic supported but not optimized for either. SOCKS5 proxy tunnel used | Optimized for TCP traffic | Optimized for UDP traffic | Optimized for UDP traffic |

*Table 4: Comparison of different tunneling techniques*

# b. Tunnel Architecture

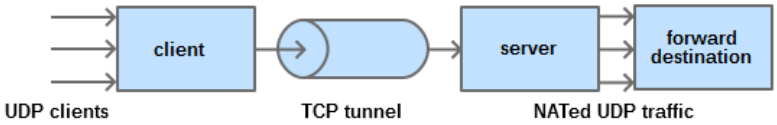



*Figure 9: Tunnel architecture*

## Socket Server

The *SocketServer* module simplifies the task of writing network servers by providing various services to interface the sockets and how the handle data.

It provides 2 basic server classes:

1. *TCPServer* uses the Internet TCP protocol, which provides for reliable streams of data between the client and server.
2. *UDPServer* uses datagrams, which are discrete packets of information that may arrive out of order or be lost while in transit.

These classes process requests synchronously thus each request must be completed before the next request can be started. This isn't suitable if each request takes a long time to complete, because it requires a lot of computation, or because it returns a lot of data which the client is slow to process.

The system uses the Python module *Pickle* for serialization which involves flattening objects i.e. converting an object hierarchy into a byte stream. Raw packets are serialized before sending to other end of the tunnel

The system uses symmetric encryption i.e. the sender and receiver of a message use the same key to encrypt and decrypt the message. It uses AES 256 (Advanced Encryption Standard, an improvement over DES and triple DES) for encryption. A 32 bit key and 16 bit block cipher with Cipher Block Chaining mode of operation are used which involves each block of plaintext being XORed with previous ciphertext block before being encrypted.

## Threading

The system is fully threaded and uses resource locks wherever appropriate to maintain mutual exclusion. Thus, care is taken that no two concurrent processes are in their critical section at the same time.

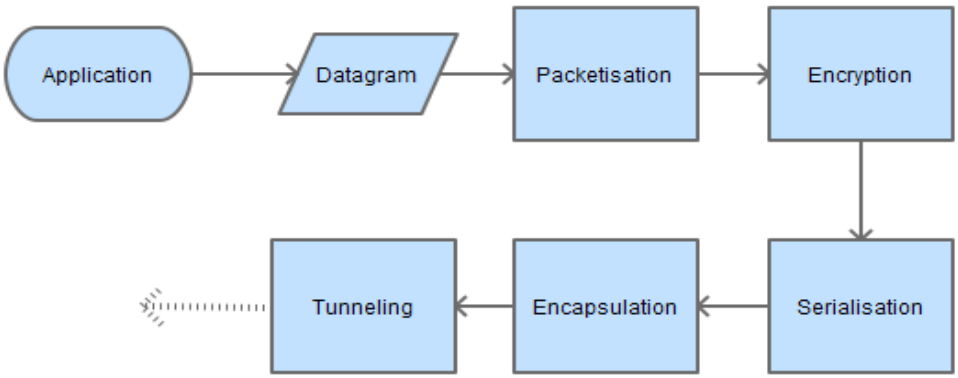



*Figure 10: Tunnel operation*

# c. Design Specifications

The design of the VoIP is kept to basic as the stress is on the additional security features being provided by using a SSH tunnel. Features can be added to the VoIP without affecting the security aspect of the system.

## Use Case

The diagram shows two types of actors, a client and a server. Two clients have been used to establish clarity considering there will be at least two people involved in each call.

The multiplicity has been shown to be many to many for clients showing many clients can make multiple calls and have multiple conversations while a single conversation can have multiple clients.

The server is shown to be maintaining connections. There will be only one one main server managing multiple sessions.

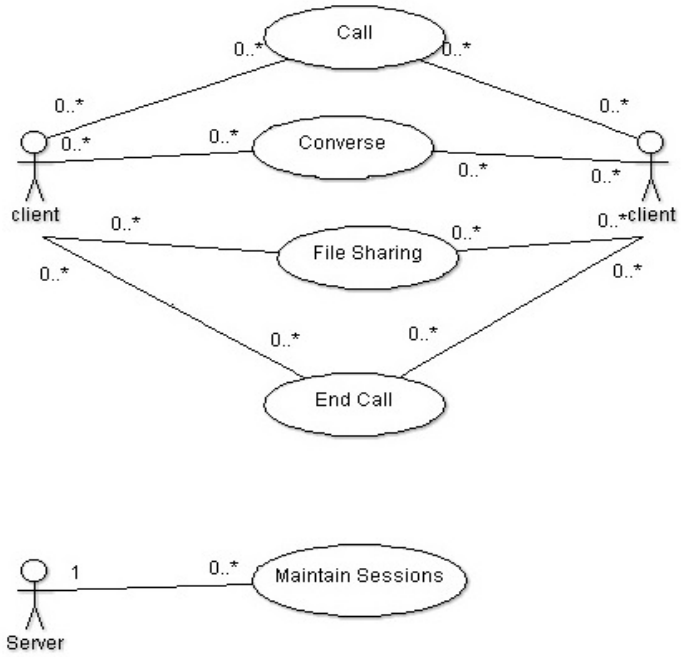



*Figure 11: Usecase Diagram*

# Sequence Diagram

The sequence diagram highlights the steps involved in a basic VoIP connection setup. Once signalling is established, a channel is set up before analog voice signals are digitized, encoded and packetized. These packets are then transmitted.

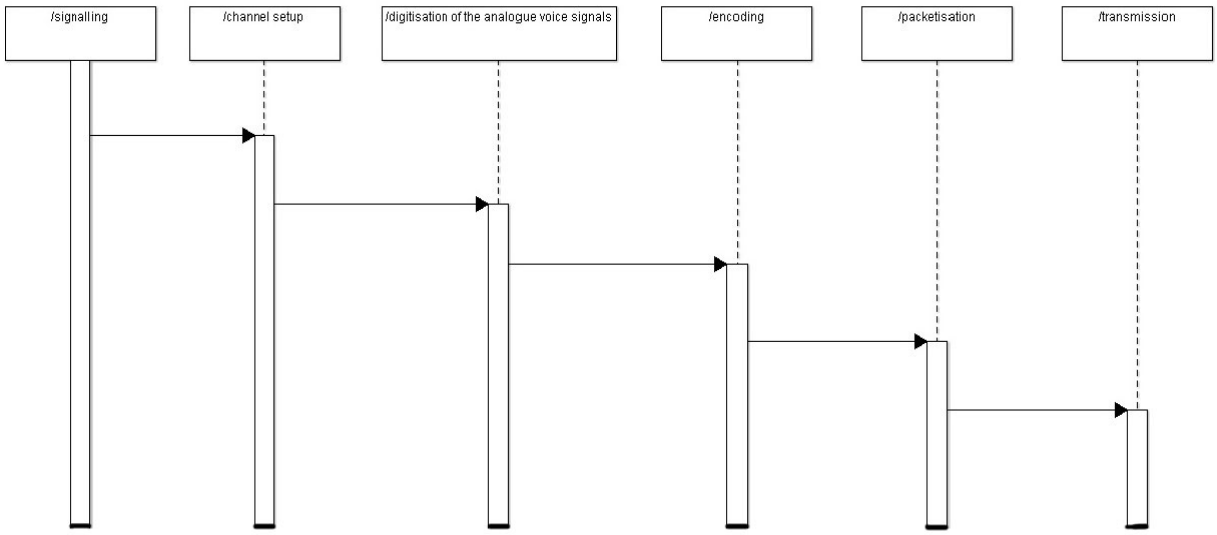



*Figure 12: Sequence Diagram*

# Block diagram

The block diagram shows two clients with their basic setup involving recording, transcoding and encapsulation. Each client records data transcodes, encapsulates and transmits it through SSH tunnel. The other client will expand it to remove encapsulation, remove the transcoding and give the final recording that will be played by the speaker on the client's end.

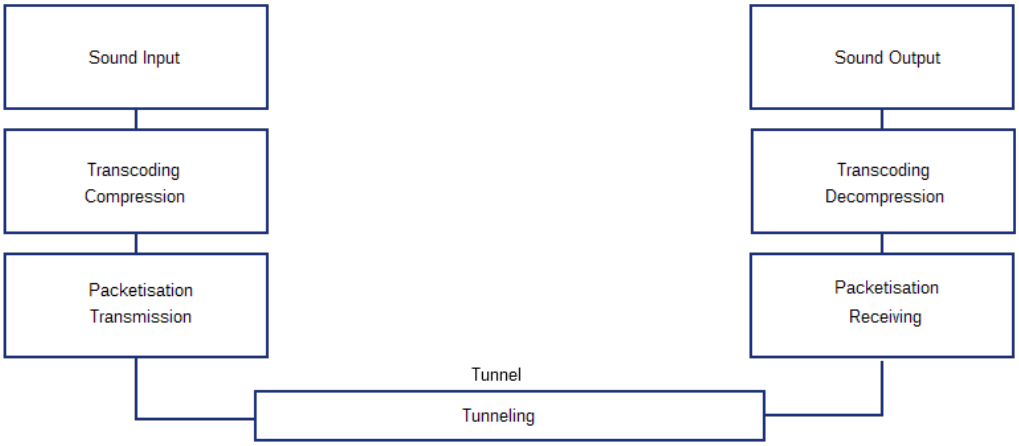



*Figure 13 - Block diagram*

# *VI. Implementation*

# a. Development Tools and Libraries

## Python

To implement the project we have chosen Python 2.7 for reasons explained below. The implementation focus remains on creating a basic VoIP and testing it heavily to provide adequate security while simultaneously providing optimal call quality.

Python is a general purpose, high-level, interpreted programming language that emphasises code readability. It enables programmers to work faster with lesser lines of codes needed to express concepts as compared to languages like C or Java. It has a very easy to use syntax which makes it expressive and succinct. Python has a rich set of supporting libraries. Python being interpreted, has provides useful diagnostic information when something goes wrong. It can also handle all of the tedious housekeeping that is part of how programs make use of the computer's resources. Python is also heavily documented and an easy language to learn.

This, along with the fact that python is very dynamic in its code by allowing usage of variable without specifying a data type, makes it ideal for rapid programming.

Some Python modules being used extensively in this project are:

- *Pyaudio*: Audio port python binding
  Used to get Audio from microphone for the VoIP.

- *Tkinter*: Tcl/Tk GUI toolkit
  Used to build GUI interface

- *Paramiko*: SSH2 implementation
  Paramiko will be modified to support UDP and then used to tunnel the VoIP through the SSH tunnel.

- *Pycrypto*: Cryptography toolkit
  Used for added encryption to SSH.

- *Speex/Opus*: Audio codec optimised for speech
  Used to compress audio optimally to send through tunnel.

# b. Practical Application of the Tunnel: A VoIP client

The GUI is made on wxPython module for GUI development in Python. wxPython is a cross platform toolkit for creating desktop GUI applications. With wxPython developers can create applications on Windows, Mac and on various Unix systems. wxPython is a wrapper around wxWidgets, which is a mature cross platform C++ library.

## Chat

It has a text chatting feature with a log of previously sent and received messages maintained by a queue. Voice chat can be enabled using a check box. This is to show that both voice and text can be handled at the same time. PyAudio module is used for the voice element of the VoIP to bind to the microphone. “Send” button is the handle used to send messages.

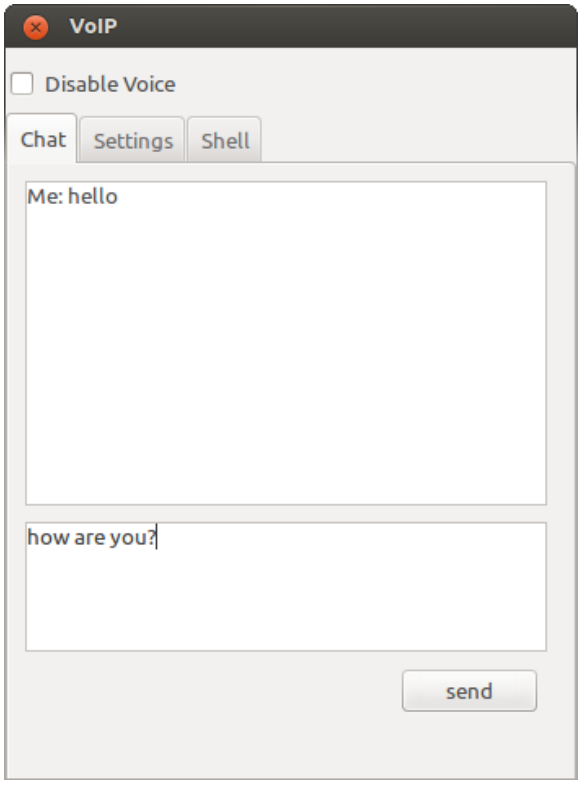



*Figure 14: VoIP - Chat tab*

# Settings

A text file labeled config.txt maintains settings of the VoIP which includes client address, client port, server address, destination address and port, audio settings and encryption settings. These can be changed by the user and the “save changes” button updates the configuration in the config.txt file.

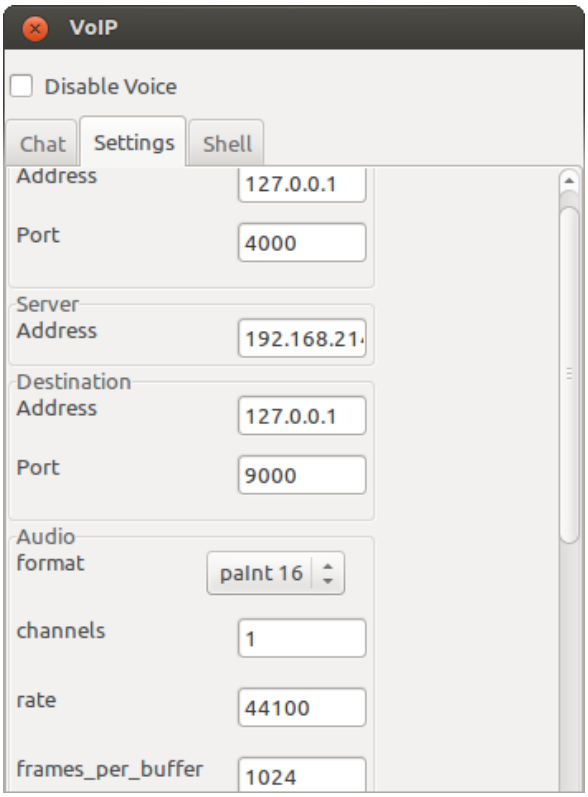



*Figure 15: VoIP - Settings*

Audio settings:

- format: the type of audio codec to use
- channels: the number of audio channels
- rate: the bitrate of audio transmitted
- frames_per_buffer: the size of transmission frames

Encryption setting:

- algorithm: the type of symmetric encryption algorithm
- block mode: set the single block operation of the algorithm
- block size: set the size of encryption block
- key size: set the size of the key, and thus the strength of encryption
- init_vector: the initialization vector for block chaining

- key: the encryption key

## Shell

This tab enables a shell where basic SSH commands can be run from. It runs `bash` as a subprocess. Due to technical limitations, shell built-in commands, such as `ls` do not work. The shell is wrapped by the tunnel; any connections initiated in the shell are made through the tunnel.

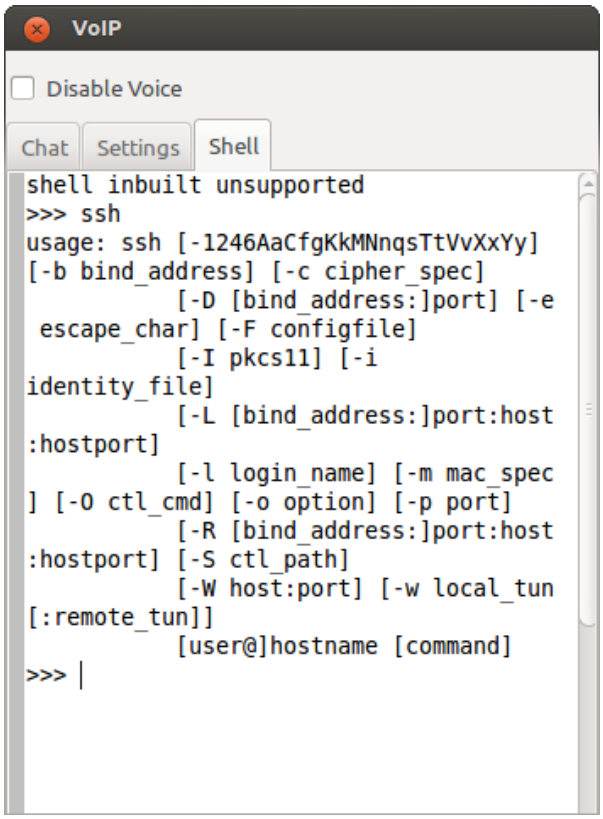



*Figure 16: VoIP - Shell tab*

# *VII. Testing*

# VII. Testing

Testing phase is notionally designed to be carried out after system development is complete. The testing phase measures the actual versus expected outcome of the system. Unlike quality control measures which are designed to evaluate a developed work product and include audits to assess cost of correcting defects, the goal of testing is to find defects through the execution of the system or software package.

Since the project was built using rapid prototyping for an innovative design scenario, it was found to be crucial to test it extensively for bugs and for various performance figures. The system must be clear of any structural flaw such as heartbleed that can arise due to lapses at the same time it must also be free of poorly coded errors.

The system must simultaneously be tested for how much of an impact the overhead of encryption and tunneling will affect the performance. This is done by comparing the system after using the cryptography and tunneling modules to the system before their integration and analysing the differences. The system must function as close to normal UDP performance as possible while providing reliability and security to the data being transmitted.

Hence, following the initial build, the system was tested extensively for security and performance efficiency in a controlled environment following which it was beta tested in a closed group and usage was monitored along with feedback.

The testing done on the system can be broadly divided into three parts:

1. White Box Testing
2. Black Box Testing
3. Alpha Testing and Debugging
4. Beta Testing and Feedback.

# a. White Box Testing

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing). In white-box testing an internal perspective of the system, as well as programming skills, are used to design test case.

## Unit testing

Unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures are tested to determine if they are fit for use. Since the entire system was made in the form of modules, each module was tested individually post development of said modules. Each module was found to be functioning independent of all other modules while tested in a local environment.

- Tunnel module `run.py`

  To test the tunnel module, the ports and address links for the end points of the tunnel were left open and dynamically set while the module ran along with text data that was transmitted via the tunnel to observe results. The other modules were not integrated at this point to the tunnel module and text data was transmitted through the tunnel. Objective was to check the working of bindings of ports and addresses along with appropriate NAT translations to implement proxies and to observe the appropriate establishment of the tunnel. Log messages were made to be displayed in the console during the appropriate completion of each step in the tunnel setup and working to observe its proper running.

  The log messages were found to be in appropriate order and displayed at constant and desired intervals of time to show the packets of text were being transmitted and received appropriately over the tunnel.

- Cryptography module `cry.py`

  Packets were encrypted according to various encryption algorithms available using the PyCrypto modules in python and their encryption was examined to see if it follows encryption standards. Objective was to check if the cryptography algorithms comply with international standards and to assure the security of data with top notch encryption. The encryption algorithm to be used was set at runtime and a set of packets were encrypted and examined.

All encryption standards were met with the cryptography module and they were all tested and found to implement best possible security encryption.

During this phase, the flaw of using the random number generators implemented in python was observed to plague the encryption system that requires large random primes as the random numbers generated in python were found to be predictable. To combat this issue, POSIX random number generators were used to generate random numbers. These use hard drive entropy as the basis for the pseudo random number generation thereby ensuring that there is no predictability in the random numbers chosen enforcing extra protection on the encryption algorithms.

- Communications module `comm.py`

  The communication unit has classes that deal with each independent task related to communication such as a class for sending and receiving text, a class for sending and receiving voice data, a class to handle all log messages etc. Each class is individually tested by exchanging data that the class handles to see if it works independent of all classes. Objective was to ensure that each class works independent of all other classes to enable reuse as per object orientation with every function of the class tested and to test the overall communications unit independent of all other modules.

  Each class is found to be syntactically correct and working independently of all other class when run in the software. All functions of all classes were examined and found to be working as per requirement.

- GUI module `gui.py`

  Various GUI elements were tested individually to see if they were performing the intended task and the settings menu was examined to see if it was able to modify the configuration file from where the other modules import settings.
  The GUI elements such as tabs, text and checkboxes were found to be working independent of the main module and the GUI was successfully able to manipulate a configuration file.

# Integration testing

Integration testing (sometimes called integration and testing, abbreviated I&T) is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit

tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for further testing.

To begin with, the communication module integrated with tunnel module and then tested to see if each communication module class was performing all functions using the tunnel. All packet exchange that happens in the communication module was passed through the tunnel with appropriate log messages and every function was tested in this manner. The appropriate log messages were obtained at regular and expected intervals thereby concluding successful integration of tunnel and communication modules.

Cryptography module was then integrated into the system and tested. Encryption was now applied to all the packets going through the tunnel. Packets were encrypted before they entered the tunnel thereby ensuring security. The combined system was then tested with appropriate log messages and they were seen to appear as expected from the system. All packets in the tunnel were found to be encrypted thus concluding the successful integration of the cryptography module.

GUI module integrated into system and tested. GUI elements were bound to functions from the earlier module and were tested individually to see if each function works as intended by calling the necessary routines from the previously integrated modules to tunnel packets through the system.

The entire system is found to be working as desired by transmitting text and voice over the designed hybrid tunnel to implement security. The working was examined by displaying log messages at the console whenever a routine was called and some action was performed. The timing and order of these log messages were then analyzed to ensure that the system is working as desired.

# Verification and Validation testing

Verification and validation testing involves checking the system following the final build to see if it still complies with the specification document drafted before development began. The system was analysed as per the requirements and specifications drafted and the following conclusions were drawn:

A tunnel is being used as specified to optimally transmit UDP packets. The tunnel is integrated with a VoIP solution as specified and found to be transmitting voice as desired. The tunnel is integrated with a cryptography module that provides security and encryption features similar to that of SSH as specified. As specified, all UDP packets through the tunnel are encrypted and optimized for best possible performance.

# b. Black Box Testing

Black-box testing is a method of software testing that examines the functionality of an application (e.g. what the software does) without peering into its internal structures or workings. As part of black box testing, the following tests were performed:

## Security testing

A Wireshark test was performed on the system similar to the one done on existing VoIP solutions to obtain packets by latching onto a socket. The packets obtained were found to be encrypted with various cryptographic algorithms thereby assuring security of data in the system.
Once packets entered the tunnel, they were found to be infeasible to intercept considering they were encapsulated and then NAT address translated to effectively implement a proxy. Intercepted packets were encrypted and due to the proxy, their destination address was unknown.

The only possible position of obtaining plain packets with no encryption or security was before and after the packets left the tunnel and was provided to the application. This requires the application itself to use encryption as the VoIP solution does in the system developed. Else, while the packets are securely transmitted, they may be intercepted outside the scope of the tunneling mechanism if not handled efficiently.

## Load testing

As per Load testing, the performance of the system was examined under heavy load conditions. To emulate heavy load, a large number of packets (10,000) were sent in a burst through the tunnel and the number of packets received were monitored.

| Test number | Number of Packets sent | Number of Packets Received |
|---|---|---|
| 1 | 10,000 | 9915 |
| 2 | 10,000 | 9913 |
| 3 | 10,000 | 9922 |
| 4 | 10,000 | 9920 |
| 5 | 10,000 | 9920 |

*Table 5 - Load Testing - Packet Loss*

As seen in the table, five bursts of 10,000 packets were sent and the average number of packets received was found to be 9918 which gives an average loss of 82 packets in 10,000. Thus the packet loss percentage can be estimated to be 0.82% which is at par with a normal UDP transmission of packets over similar conditions.

Thus it can be said that the additional encryption and security with tunneling does not impact the efficiency of the system under heavy load conditions.

# Endurance testing

As per endurance testing, the performance of the system was examined under poor network conditions. To emulate poor network conditions, a network emulator was used with manually set network loss percentages. This test was performed specifically to examine if the system of the hybrid still exhibited UDP-like characteristics of providing a steady stream of data irrespective of the losses in the network.

| **Network Loss (%)** | **Delay (ms)** |
|---|---|
| 0 | 83 |
| 10 | 87 |
| 20 | 93 |
| 30 | 95 |
| 40 | 99 |
| 50 | 115 |
| 60 | 124 |

*Table 6 - Delay as per Network Condition*

As seen in the table, the delay was observed with degrading network conditions. The delay was found to increase steadily as network conditions worsened and more packets were lost but the increase in delay is seen to be minute in the order of 40-50ms to losses of up to 60%.
This indicates the characteristics of UDP are still maintained where even under heavy losses, a steady stream of packets will be obtained by the receiver.

# Comparison testing

Under comparative testing the system is compared to existing security solutions to analyze the inherent merits and demerits in comparison with each other. The system is compared to TOR for its proxy implementation and its found to be faster than the TOR protocol as TOR takes time to setup a relay circuit before it begins transmission. The system is found to have the best of both PPTP and L2TP in the way that it behaves like a UDP connection by providing the steady stream of data at the same time uses TCP messages to control the tunnel and provide reliability to the transmission.

# c. Alpha and Beta Testing

## Alpha Testing and Debugging

Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. As per Alpha Testing, the system was tested amongst the members of the project group in a controlled environment and detailed analysis was done.

Testing, debugging and optimization revealed a bug in the system i.e., unlike TCP which can accept packet of any size with the range specified in the RFC, UDP needs a fixed packet size owing to the fact that packet size is not a parameter in a UDP header. This bug was immediately fixed by specifying a packet size within the tunnel.

## Beta Testing and Feedback

Beta testing comes after alpha testing and can be considered a form of external user acceptance testing. Versions of the software, known as beta versions, are released to a limited audience outside of the programming team. As per Beta Testing, the software was released to a controlled group and the usage was monitored and regular feedback was taken.

While most end users were oblivious to the tunnel and coding, some were interested in the ability to use the tunnel to run apps that needed real time UDP data transfer that is normally blocked by the firewall because the tunnel uses a proxy and additional security, firewalls were oblivious to data sent through it. This enabled real time UDP transmission that was previously disabled.

## Tools Used

The various development tools that were used during the testing are documented below:

- Python Debugger

  They Python Debugger module in python enables effective debugging of the code post mortem or live. It enables various debugging functions such as setting breakpoints and observing code implementation line by line to find flaws as they arise. The tool was used to work through the code to find various coding related errors and correct them.

- NEWT

  NEWT (Network Emulator for Windows Toolkit) is a software-based emulator that can emulate the behavior of both wired and wireless networks using a reliable physical link, such as an Ethernet. A variety of network attributes are incorporated into the NEWT emulation model, including round-trip time across the network (latency), the amount of available bandwidth, queuing behavior, a given degree of packet loss, reordering of packets, and error propagations. The tool was used to simulate various network conditions that was needed for various tests done on the system.

# *VIII. Conclusion and Future Scope*

# VIII. Conclusion and Future Scope

Existing tunneling solutions and their merits and demerits are discussed in the project and that the choice between TCP and UDP is specific to the application in question has been established. A means of securing a UDP communication channel thereby improving the existing means in which datagrams are transmitted has been provided. The proposed system can be adapted to be used in the case of VoIP to show how real time UDP traffic can be secured by the system.

The VoIP was developed under rapid prototyping model where an initial prototype was set up and it was developed in modules to finally reach a complete prototype. This complete prototype has an independent tunnel, cryptography, communication and GUI module to enable reuse and efficient coding practises. The system was extensively tested and found to be compliant with the necessary requirement of secure and efficient real time voice transmission.

The hybrid TCP/UDP tunnel has been seen to have scope extending well beyond the VoIP solution in the project. Because the tunnel module was developed as a separate piece, it can easily be optimized to use with other applications. By setting up the tunnel to fixed ports between to IPs, all traffic needing to be secured may be transmitted through this tunnel to secure the transmission.

First we can consider the case of wireless ad-hoc routers which transmit advertisement and receive password and authentication packets using UDP. This channel is known to be insecure and has been exploited by capturing these UDP packets while password exchange takes place and then deciphering the password. Using the tunnel, we can establish a tunnel after solicitation message, advertisement and initial request to connect have been exchanged and then tunnel password and other important connection messages through the tunnel to establish secure communication.

On purchase of original software, the licenses are exchanged for verification using UDP packets. This exchange is also exploited by intercepting packets and then explicitly modifying and sending false license acknowledgements. This can be stopped using the tunnel. If the tunnel is established before communicating the licenses, it can be assured that even the intercepted packets will be heavily encrypted and thus useless to someone who wishes to falsify license acknowledgements.

The tunneling system can be expanded to support many to many type connections involving multiple tunnels to different parties to engage in secure communication with multiple parties and multiple tunnels between two users to have multiple secure channels for different communications simultaneously. This makes it ideal for multi-user real time data transfer scenarios such as that of the VoIP mentioned earlier. Because UDP-like traits are still present, along with a reliable channel for communication, it is

ideal in these scenarios. Other real time data exchange scenarios include but are not limited to online multiplayer gaming and live streaming which can all be secured using the system developed.

In real time gaming, servers are known to be attacked by falsifying game data. At present, servers go through extensive amounts of checking to find anomalies in game data to find misfeasance among users and ban them. Using the tunnel module here will ensure secure transmission of game data that cannot be intercepted and modified to generate false information. This can be used in all areas of gaming including PC, mobile and console devices made exclusively for this purpose which use multiplayer gaming as key feature.

Live streaming packets can sometimes be intercepted which results in access to paid channels on certified web sites for live streaming without paying for said channels. Using the tunnel, access can be restricted to only legitimate users who paid for the services.

# *Appendix*

# A. Abbreviations

| | |
|---|---|
| UDP | User Datagram Protocol |
| VoIP | Voice over Internet Protocol |
| OSI | Open Systems Interconnect |
| SSL | Secure Socket Layer |
| TLS | Transport Layer Security |
| HTTP | Hyper Text Transfer Protocol |
| TCP | Transmission Control Protocol |
| IP | Internet Protocol |
| SSH | Secure Shell |
| SIP | Session Initiation Protocol |
| QoS | Quality of Service |
| SDES | Session Description Protocol Security Descriptions |
| SRTP | Secure Real-time Transport Protocol |
| ZRTP | Z Real-time Transport Protocol |
| MIKEY | Multimedia Internet Keying |
| SFTP | Secure File Transfer Protocol |
| SCP | Session Control Protocol |
| DES | Data Encryption Standard |
| IDEA | International Data Encryption Algorithm |
| VPN | Virtual Private Network |
| NAT | Network Address Traversal |
| TOR | The Onion Router |
| SOCKS | Socket Secure |
| PPTP | Point-to-Point Tunneling Protocol |
| L2TP | Layer Two Tunneling Protocol |
| GRE | Generic Routing Encapsulation |
| ATM | Asynchronous Transfer Mode |
| NEWT | Network Emulator for Windows Toolkit |

# B. Bibliography


[1] Ole Martin Dahl, Limitations and Differences of using IPsec, TLS/SSL or SSH as VPN-solution: *www.olemartin.com/projects/VPNsolutions.pdf,* 2004

[2] Vandyke Software, An Overview of Secure Shell: *www.vandyke.com/solutions/ssh_overview/ssh_overview.pdf,* 2005

[3] Massimo Maresca, Nicola Zingirian, Pierpaolo Baglietto, Internet Protocol Support for Telephony: *Proceedings of the IEEE, Vol. 92, No. 9,* 2004

[4] Thomas J. Walsh, D. Richard Kuhn, Challenges in Securing Voice over IP: *IEEE Security & Privacy Magazine, May/June,* 2005

[5] Osamu Honda, Hiroyuki Ohsaki, Makoto Imase, Mika Ishizuka, Junichi Murayama, Understanding TCP over TCP: Effects of TCP Tunneling on End-to-End Throughput and Latency: *Proceedings of the SPIE, Volume 6011, pg. 138-146,* 2005

[6] Maurizio Dusi, Francesco Gringoli, Luca Salgarelli, A Preliminary Look at the Privacy of SSH Tunnels: *17th IEEE International Conference on Computer Communications and Networks (ICCCN'08),* 2008

[7] Maurizio Dusi, Francesco Gringoli, Luca Salgarelli, A Model for the Study of Privacy Issues in Secure Shell Connections: *The Fourth International Conference on Information Assurance and Security,* 2008

[8] Stephan R. Schach, Object Oriented and Classical Software Engineering: *Tata McGraw Hill,* 2007

[9] Dingledine, Roger; Mathewson, Nick; Syverson, Paul; Tor: The Second-Generation Onion Router. *Proce. 13th USENIX Security Symposium. San Diego, California,* 2004

[10] RFC 2341 Cisco Layer Two Forwarding (Protocol) "L2F"(a predecessor to L2TP)

[11] RFC 2637 Point-to-Point Tunneling Protocol (PPTP) (a predecessor to L2TP)

[12] RFC 768 – User Datagram Protocol

[13] RFC 793 – TCP v4

[14] RFC 1323 – TCP-Extensions

[15] RFC 675 – Specification of Internet Transmission Control Program, December 1974

# C. Research Paper

## A Transmission Controlled Tunnel for Datagrams

Sameer Jain, Viren Sinai Nadkarni, Pranav Prem, Nagraj Vernekar
Computer Engineering Department,
Goa College of Engineering, Farmagudi, Ponda- Goa
samir4nov@gmail.com, viren.nadkarni@gmail.com, pranavprem93@gmail.com, nkv_2447@yahoo.com

**Abstract—Conventionally a TCP network setup is considered better than a UDP network setup because TCP offers connection establishment and acknowledgements ensuring delivery of every packet. However in certain situations a UDP connection outperforms a TCP connection such as that of real time data transfer in the scenario of a VoIP examined in this paper. This paper establishes that the choice between a TCP and a UDP connection is application specific and then provides a means of securing UDP transmission by creating a hybrid tunnel that uses connection establishment principles of a TCP network and transmitting UDP packets over established tunnel with enforced encryption at the application layer itself and no retransmission or ordering. This combines some of the security and integrity features of TCP with the speed of UDP.**

**Keywords—User Datagram Protocol (UDP), Transmission Control Protocol (TCP), Datagrams, Packets, Tunneling**

### I. INTRODUCTION

UDP (User Datagram Protocol) is a communications protocol that offers a limited amount of service when messages are exchanged between computers in a network that uses the Internet Protocol (IP) to get packets of data (called datagrams in this case) from source to destination [5]. It provides only a single service over IP which is a port number to distinguish between user requests. UDP doesn't provide sequencing of the packets that the data arrives in. Neither does it provide any handshaking or connection establishment. While UDP is at transport layer in the OSI schema, it becomes the responsibility of the application at application layer to make sure comprehensible data is received. TCP (Transmission Control Protocol) is a set of communications protocol used along with the Internet Protocol(IP) to send data in the form of message units between computers over the Internet [6]. TCP divides the message into packets and provides the services (to IP) of ensuring the delivery of every packet by using connection establishment and handshaking signals to acknowledge the delivery of every packet. TCP works in the transport layer and additionally maintains the order of receiving packets at the destination. It establishes a connection and maintains it until such time that the messages to be exchanged by the application programs have been exchanged.

From this it is fairly evident why TCP is considered a standard while UDP is used for more trivial tasks that require less security. This misconception is stressed on in many cases such as the usage of UDP in a file transfer protocol that is labeled Trivial File Transfer Protocol (TFTP). TFTP is extremely limited and provides no authentication. It is known for its simplicity.

With the increase of bandwidth and the revolution of hand-held devices, real time media transfer has increased significantly over the last few years. This shift has led to focus returning to UDP transmissions for communication.

In early April 2014, a critical bug that manifested in SSL/TLS implementation was discovered. The bug was named "Heartbleed". The bug was a buffer underflow issue that came from 'heartbeat' signals that are sent in SSL/TLS to check if a server is alive. Each 'heartbeat' involves data to be returned and the size of data to be returned if server is alive. The server is to read the size and return the requested data of that size if it is alive. The problem arises when the size of the data requested to be returned is larger than the data supplied to be returned. According to SSL/TLS mechanism, the server continues sending data from the input buffer until the amount of data asked for is provided. This data could contain session data of other people logged into the server causing major breach of security [2].

This problem is a clear example of how security that spans through more than one layer of the OSI architecture is prone to serious flaws because of inter-layer co-ordination. Hence, the paper proposes a system that has all its security implemented in the application layer itself. All packets are encrypted in all other layers as well as shielded by the tunnel which maintains the connection and makes sure there is no leak of information.

The paper is organized as follows; the second section is an analysis of existing tunneling technologies, their merits and demerits. Following that is an analysis to show why and by how is a UDP channel better than a TCP channel in case of real time applications. The fourth section shows how the paper proposes to secure a UDP connection and combine some essential features of both TCP and UDP using a hybrid tunnel. The section after that shows how this hybrid tunnel is different from the existing tunneling mechanisms. The final section is about the scope of implementing such architecture to secure a UDP channel.

## II. Existing Tunneling Technology

### A. TOR Network

TOR (The Onion Router) protocol is an internet security protocol that is used for complete anonymity over the World Wide Web. To connect to the TOR network, a TOR circuit is established with relays between a source and destination. In TOR routing, the intermediate relays themselves do not know the entire circuit. Only the source and destination are privy of that information. The data is encrypted at the source multiple times and passed into the relay circuit. Each relay in the circuit decrypts only the header to find out the address of the next relay in the circuit and then forwards the packet onwards with one less layer of security. The final exchange will involve plain text exchange between the last relay in the circuit and the destination. To avoid this, data is secured with SSL / TLS before entering the TOR circuit [1].

The TOR network uses a tunneling proxy to serve as the TOR tunnel and to maintain anonymity. SOCKS is the Internet Protocol used by TOR to maintain anonymity. It manages exchange of data between source and destination by routing data through a proxy server. SOCKS server proxies TCP connection to an arbitrary IP address and provides a means for UDP packets to be forwarded.

### B. VPN tunnel

VPN (Virtual Private Network) is the extension of a private network that includes links across shared or public networks such as the internet. VPN helps send packets from source to destination over a network in a way that it emulates a point to point link. This is done by creating a tunnel between source and destination.

In case of Windows VPN, Point to Point Tunneling Protocol (PPTP) is used to tunnel packets over the public network. Tunnel is established with both of the tunnel endpoints negotiating the configuration variables such as address assignments, encryption and comparison parameters. After the tunnel is established, data is sent. The tunnel uses a tunnel data transfer protocol to prepare the data for transfer. This involves appending tunnel data transfer protocol headers or encryption.

PPTP encapsulates Point to Point Protocol frames into IP datagrams for transmission over an IP based network [4]. PPTP is described in RFC 2637 in IETF Database. It uses a TCP connection known as the PPTP control connection to create, maintain and terminate the tunnel. PPTP uses a modified version of Generic Routing Encapsulation (GRE) to encapsulate PPP frames as tunneled data. The payloads of the encapsulated frames can be encrypted, compressed or both.

Additionally, L2TP (Layer Two Tunneling Protocol) is a combination of PPTP and Layer 2 Forwarding, a technology developed by Cisco systems. L2TP encapsulated PPP frames to be sent over IP, X.25, frame relay or ATM networks when sent over an IP network, L2TP frames are encapsulated as User Datagram Protocol (UDP) messages [3]. L2TP can be used as a tunneling protocol over internet or over private intranets.

L2TP uses UDP messages over IP networks for both tunnel maintenance and Tunneled data. This creates the issue that L2TP tunnel maintenance and tunnel data have the same structure.

## III. UDP Preferred Scenario

To establish the performance difference between a UDP connection and a TCP system the specific case of a VoIP system is considered. In such a case, there is a real time exchange of Voice data which implies a steady stream of information is needed more than assured delivery of data. Hence typically a UDP connection should be preferred.

To analyze the given condition a prototype capable of carrying out basic VoIP function is developed in Python which uses PyAudio to capture audio and then transfers it from a server machine to a client machine. It is capable of switching between UDP and TCP modes of transmission. The Microsoft Network Emulator for Windows toolkit is used once a channel was established to simulate various real life packet loss scenarios.

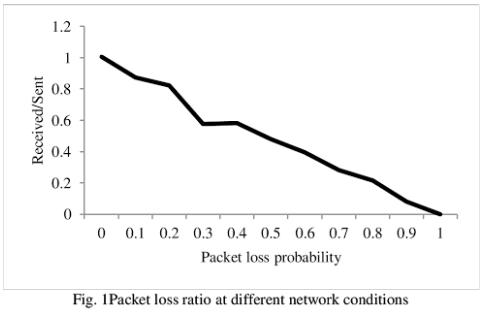



Fig. 1Packet loss ratio at different network conditions

The prototype is tested using the network tool to obtain the table and graph with various packet loss condition simulations in one way traffic for UDP and TCP. It is immediately seen that while the packet loss percentage was increased, the average delay time in case of TCP connection runs into seconds and then into minutes while the latency of a UDP connection remains almost unaffected. At the same time, the amount of packets lost in case of UDP is seen to be considerable.

The effect of the phenomenon as seen in the graphs on the audio transmitted is profound. In case of TCP, there is a significant amount of latency and stutter whereas in case of UDP there is a steady stream of audio which keeps getting more difficult to recognize with increasing loss percentages. At

highloss percentages it while it is difficult to recognize the audio track with UDP, it is impossible to do so with TCP which played pieces of sounds at intervals of up to a couple of minutes.

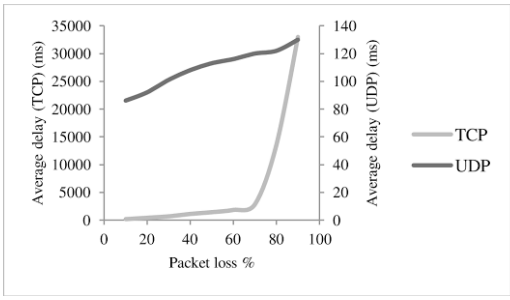



Fig. 2 Latency comparison of UDP and TCP

Thus it is seen that in such cases involving real time data transfer, a steady stream of data without latency and stutter is always preferred because lost packets are less important than packets yet to be received.

Hence a UDP connection is preferred in such scenarios. Table I summarizes the comparison of UDP with TCP.

TABLE I
COMPARISON OF UDP WITH TCP

| **TCP** | **UDP** |
|---|---|
| Broken streams of data exchange | Steady stream of data exchange |
| Priority for next packet to be received | No priorities |
| All packets received | Packets are lost |
| Used when integrity of data is more important than a steady stream of data to be transmitted. When an old packet that has not been received is more important than a new packet of the same. | Used where real time data exchange is needed. When a new packet is more important than an old one that has not been received |

## IV. PROPOSED HYBRID TUNNEL

Once established that datagram traffic was essential, a means of securing datagram communication is prepared which involves a tunneling architecture.According to the scheme, a tunnel is established using TCP mechanism involving handshakes and acknowledgements. This tunnel, once established is then made to transmit UDP packets without any mechanism for retransmission of lost packets or ordering as per UDP style. The tunnel establishes the ports for each channel before transmission and then enforces strict encryption on every UDP packet that was transmitted through it thereby ensuring security(in an SSH fashion).

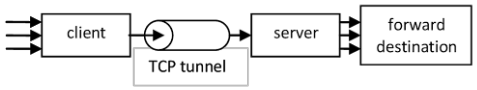



Fig. 3 Hybrid Tunnel Architecture

The TCP tunnel is prepared, as per the diagram, to transmit datagrams. The diagram shows unidirectional traffic. The bi-directional implementation involves the same process in the opposite direction. The tunnel itself is capable of handling bi-directional traffic.

The architecture involves multiple clients exchanging UDP data connected to one station. The station listens on a preset port for incoming data. Any data transmitted onto the said port is then tunneled via the established tunnel to a preset destination that is set up with the tunnel. From this destination, the address of the received packets are then resolved and retransmitted to the ports excepting the data.

The design involves a server thread for each port on client side (supporting UDP) per client/port on server side (with NAT). Client side server sends everything to the same forward destination with different source forwarding port to same forward destination port on server side. UDP packets are encapsulated appropriately before transmission.

Thus the system takes advantage of TCP services while establishing the tunnel to make sure ports are available and then uses the merits of UDP services during communication to ensure speedy delivery of data over the tunnel.

This system also provides the additional advantage of allowing applications to decide the kind of encryption they wish to enforce by using conventional encryption mechanisms such as 3DES, IDEA, Blowfish, Twofish or CAST. Security implemented on the application layer implies there will be no lapse of security arising from the clash of functions between different OSI layers in a multilayer security implementation like SRTP and SSL/TLS.

## V. COMPARISON OF EXISTING TUNNELS WITH PROPOSED HYBRID

In the established tunnel implementation, a means of creating a TCP connection and forwarding UDP packets to various ports is used by NAT translating packet headers to send them to the right ports. This implements additional anonymity over the tunnel, similar to the TOR protocol. Unlike TOR however, there is no relay circuit and hence even if someone were to latch on to the socket on the client side, the proxy would

TABLE II
COMPARISON OF EXISTING TUNNELS WITH PROPOSED HYBRID

| TOR | PPTP | L2TP | Proposed model |
|---|---|---|---|
| Uses Socks5 proxy system with a proxy tunnel involving all data transmitted through a proxy server | No proxy in implementation | No proxy in implementation | Address translation at sending and receiving end to implement proxy on data. |
| Uses a relay circuit with layered encryption on all packets such that last hop transfers plain text unless text is encrypted before entering TOR circuit | Encryption is dependent on user. No relays | Encryption is dependent on user. No relays. | All data through tunnel is encrypted. No possibility of plain text being transferred. No relays. |
| Tunnel established with proxy server as medium using SOCKS5 proxy mechanism | TCP tunnel establishment and maintenance | UDP tunnel establishment and maintenance. | TCP tunnel establishment and maintenance |
| TCP and UDP traffic supported but not optimized for either. SOCKS5 proxy tunnel used | Optimized for TCP traffic | Optimized for UDP traffic | Optimized for UDP traffic |

disable metadata analysis attacks and there would be no chance of him obtaining plain text like in case of TOR.

In a manner similar to that of PPTP, the system establishes a TCP connection to create, maintain and terminate the tunnel. In a manner similar to that of L2TP, the packets passing through the tunnel are encapsulated and encrypted into an optimized hybrid for transmission of UDP packets in the tunnel. Unlike PPTP, the system works better for a UDP transmission and unlike L2TP, the tunnel is maintained by a TCP connection which has a different encapsulated structure thereby making it more secure.

To summarize, the hybrid tunnel model being used combines a proxy solution similar to SOCKS5 used by TOR network, the tunnel establishment technique used by PPTP and the transmission and encapsulation technique used by L2TP to create an optimized hybrid for secure real time data transmission.

Table II summarizes the comparison of Existing Tunnels with Proposed Hybrid.

## VI. CONCLUSION AND FUTURE SCOPE

Existing tunneling solutions and their merits and demeritsare discussed in the paper and that the choice between TCP and UDP is specific to the application in question has been established. A means of securing a UDP communication channel thereby improving the existing means in which datagrams are transmitted has been provided. The proposed system can be adapted to be used in every case where datagram traffic is preferred over packet data in TCP format.

Some classic cases include VoIP scenarios (as established earlier), establishing connections with Access Points (both wired and otherwise) involvespassword exchange using datagrams that are currently not secure enough and license exchange in original software purchases are done with datagrams. Other than the described scenarios, all real-time data transfer takes place on a UDP connection and the system provides a means to secure all such channels.


## ACKNOWLEDGEMENT

We would like to thank Dr. J. A. Laxminarayana, Head of Dept. of Computer Engineering forproviding us the necessary resources and infrastructure; Prof. Nagraj Vernekar from Dept. of Computer Engineering for supporting and mentoring us; Prof. Sherica Menezes for her valuable advice on writing this paper; and Prof. Siddesh Salelkar for initial insight in the field.


## REFERENCES


[1] Dingledine, Roger; Mathewson, Nick; Syverson, Paul (13 August 2004). "Tor: The Second-Generation Onion Router". Proc. 13th USENIX Security Symposium. San Diego, California.

[2] Heartbleed - https://heartbleed.com (published 7th of April 2014, ~19:00 UTC)

[3] RFC 2341 Cisco Layer Two Forwarding (Protocol) "L2F" (a predecessor to L2TP)

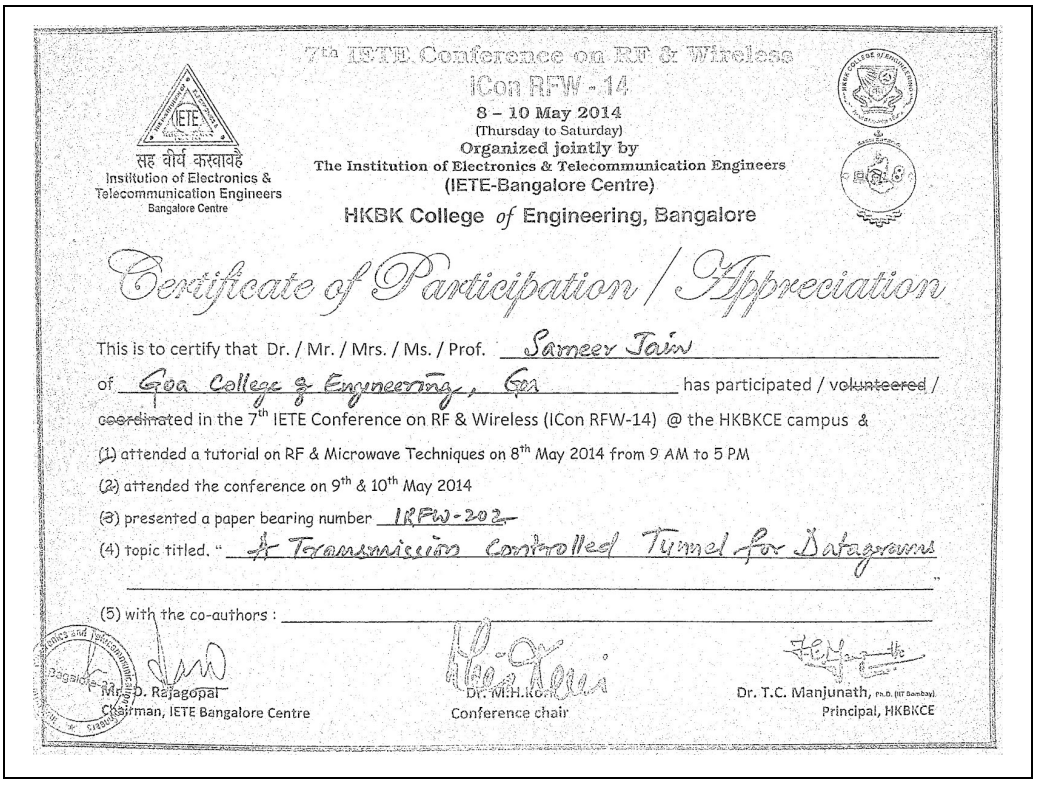
[4] RFC 2637 Point-to-Point Tunneling Protocol (PPTP) (a predecessor to L2TP)

[5] RFC 768 – User Datagram Protocol

[6] RFC 793 – TCP v4; RFC 1323 – TCP-Extensions;RFC 675– Specification of Internet Transmission Control Program, December 1974 Version

# D. Certifications

The research paper was presented and certified at the prestigious IETE Conference at the HKBK College of Engineering in Bangalore on May 10, 2014 .



7th IETE Conference on RF & Wireless

ICon RFW - 14

8 – 10 May 2014

(Thursday to Saturday)

Organized jointly by

The Institution of Electronics & Telecommunication Engineers

(IETE-Bangalore Centre)

HKBK College *of* Engineering, Bangalore

सह वीर्यं करवावहै

Institution of Electronics & Telecommunication Engineers

Bangalore Centre

## *Certificate of Participation / Appreciation*

This is to certify that Dr. / Mr. / Mrs. / Ms. / Prof. Sameer Jain

of Goa College of Engineering, Goa has participated / ~~volunteered~~ / ~~coordinated~~ in the 7th IETE Conference on RF & Wireless (ICon RFW-14) @ the HKBKCE campus &

(1) attended a tutorial on RF & Microwave Techniques on 8th May 2014 from 9 AM to 5 PM

(2) attended the conference on 9th & 10th May 2014

(3) presented a paper bearing number IRFW-202

(4) topic titled. " A Transmission Controlled Tunnel for Datagrams "

(5) with the co-authors :

Dr. D. Rajagopal
Chairman, IETE Bangalore Centre

Dr. M.H.Ko...
Conference chair

Dr. T.C. Manjunath, Ph.D. (IIT Bombay)
Principal, HKBKCE

7th IETE Conference on RF & Wireless

ICon RFW - 14

8 – 10 May 2014

(Thursday to Saturday)

Organized jointly by

The Institution of Electronics & Telecommunication Engineers

(IETE-Bangalore Centre)

HKBK College *of* Engineering, Bangalore

सह वीर्यं करवावहै

Institution of Electronics & Telecommunication Engineers

Bangalore Centre

# *Certificate of Participation / Appreciation*

This is to certify that Dr. / Mr. / Mrs. / Ms. / Prof. Viren Sinai Nadkarni

of Goa college of Engineering, Goa has participated / ~~volunteered~~ / ~~coordinated~~ in the 7th IETE Conference on RF & Wireless (ICon RFW-14) @ the HKBKCE campus &

(1) attended a tutorial on RF & Microwave Techniques on 8th May 2014 from 9 AM to 5 PM

(2) attended the conference on 9th & 10th May 2014

(3) presented a paper bearing number IRFW-202

(4) topic titled, " A Transmission control Tunnel for Datagrams "

(5) with the co-authors : _____

Mr. D. Rajagopal
Chairman, IETE Bangalore Centre

Conference chair

Dr. T.C. Manjunath, Ph.D. (IIT Bombay)
Principal, HKBKCE

7th IETE Conference on RF & Wireless

ICon RFW - 14

8 – 10 May 2014

(Thursday to Saturday)

Organized jointly by

The Institution of Electronics & Telecommunication Engineers

(IETE-Bangalore Centre)

HKBK College *of* Engineering, Bangalore

सह वीर्यं करवावहै

Institution of Electronics & Telecommunication Engineers

Bangalore Centre

# *Certificate of Participation / Appreciation*

This is to certify that Dr. / Mr. / Mrs. / Ms. / Prof. Pranav Prem

of Goa College of Engineering, Goa has participated / ~~volunteered~~ / ~~coordinated~~ in the 7th IETE Conference on RF & Wireless (ICon RFW-14) @ the HKBKCE campus &

~~(1)~~ attended a tutorial on RF & Microwave Techniques on 8th May 2014 from 9 AM to 5 PM

~~(2)~~ attended the conference on 9th & 10th May 2014

~~(3)~~ presented a paper bearing number IRFW-202

(4) topic titled, " A Transmission Control Tunnel for Datagrams "

(5) with the co-authors : _____

Mr. D. Rajagopal
Chairman, IETE Bangalore Centre

Conference chair

Dr. T.C. Manjunath, Ph.D. (IIT Bombay)
Principal, HKBKCE

# E. Source Code

./run.py

---

```
#!/usr/bin/env python

import sys
import pickle
import select
import struct
import logging
import getopt
import SocketServer
from datetime import datetime
from bidict import bidict
from comm import *
from config import *
from cry import *
from gui import *

global udpservers, udpserverslock, client, localaddress, serveraddress, port, remotefwdto, udpports, tcpconnections,
tcpconnsock2key, udpportnat

udpserverslock = threading.Lock()

udpservers = dict()
tcpconnections = dict()
tcpconnsock2key = dict()
udpportnat = bidict()


def usage():
          print sys.argv[0], '[OPTION]...'
          print 'Runs as server when no arguments are given'
          print '-c, --client  run as client'
          print '-h, --help   display this help & exit'
          sys.exit()

console_lock = threading.Lock()
logging.basicConfig(format='%(asctime)s %(message)s', datefmt='%H:%M:%S', level=logging.INFO)
config_reader = Config()
config_dict = config_reader.readall()
creeper = Cry(config_dict['key'], config_dict['init_vector'])


def log(*args):
          if not config_dict['logging']:
          return
          with console_lock:
          logging.info(''.join([str(x) for x in args]))


def clientsendpktcallback(client_address, server_address, sendingto, udp):
          global udpserverslock
          rport = server_address[1]
          sport = client_address[1]

          dst = (sendingto, rport)
          f = (('0.0.0.0', udpportnat[:sport]))
          log("(UDP) forwarding packet from ", f, " to ", dst)

          try:
          with udpserverslock:
          udpservers[f].socket.sendto(udp, dst)
          except KeyError:
          log(udpservers.items())


def serversendpktcallback(client_address, server_address, sendingto, udp):
          global udpserverslock
          rport = server_address[1]
          sport = client_address[1]
```

```
          dst = (sendingto, rport)
          f = (('0.0.0.0', udpportnat[:sport]))
          log("(UDP) forwarding packet from ", f, " to ", dst)

          try:
          with udpserverslock:
          udpservers[client_address].socket.sendto(udp, dst)
          except KeyError:
          log(udpservers.values())
          raise


class TCPConnection(object):
          def __init__(self, tcp_socket, tcp_socket_lock, src):
          self.tcp_socket = tcp_socket
          self.tcp_socket_lock = tcp_socket_lock
          self.src = src
          self.bf = ''


class ThreadedUDPServer(SocketServer.ThreadingMixIn, SocketServer.UDPServer):
          last_received = datetime.now()

          def set_client_mode(self):
          self.clientmode = True

          def set_tcp_socket(self, tcp_socket, tcp_socket_lock):
          self.tcp_socket = tcp_socket
          self.tcp_socket_lock = tcp_socket_lock


class ThreadedUDPHandler(SocketServer.BaseRequestHandler):
          def handle(self):
          self.server.lastreceived = datetime.now()

          src = [x for x in self.client_address]
          dst = [x for x in self.server.server_address]

          log("(UDP)(not NATed) received from ", src, " for ", self.server.server_address)

          if not (getattr(self.server, 'clientmode', None)):
          rport = dst[1]
          log("(UDP) port translated from ", dst, ' to ', udpportnat[rport:])
          rport = udpportnat[rport:]
          dst[1] = rport

          # payload to be sent is in 'self.request[0]'
          if config_dict['encryption']:
          print 'encrypt'
          s = pickle.dumps((tuple(src), tuple(dst), creeper.encrypt(self.request[0]),), pickle.HIGHEST_PROTOCOL)
          else:
          s = pickle.dumps((tuple(src), tuple(dst), self.request[0],), pickle.HIGHEST_PROTOCOL)

          if getattr(self.server, 'clientmode', False):
          # tcpc = None
          key = tuple(map(None, src, (dst[1],)))

          if not tcpconnections.has_key(key):
                    tcp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                    log("(TCP) connecting to ", serveraddress, ":", port, " to forward from ", src, " to ",
                    (remotefwdto, dst[1]))
                    tcp_socket.connect((serveraddress, port))
                    tcp_socket_lock = threading.Lock()

                    tcpconnections[key] = TCPConnection(tcp_socket, tcp_socket_lock, src)
                    tcpconnsock2key[tcp_socket] = key

                    b = "%s\n%s" % (self.client_address, remotefwdto)
                    b += ' ' * (128 - len(b)) if len(b) < 128 else ''
                    tcp_socket.send(b)
                    self.server.set_tcp_socket(tcp_socket, tcp_socket_lock)

          log("(TCP) sending (NATed) UDP from ", src, " to ", dst, " on TCP ")
          self.server.tcp_socket_lock.acquire()
          self.server.tcp_socket.send(struct.pack("!I%ds" % len(s), len(s), s))
          self.server.tcp_socket_lock.release()
```

```
def read_tcp(tcp_socket, tcp_socket_lock, bf, sendingto, srvcallback=None, sendpktcallback=None):
          r = None
          tcp_socket_lock.acquire()
          tcp_socket.settimeout(0.1)
          try:
          readin = tcp_socket.recv(4096)
          if len(readin) == 0:
          raise socket.error
          bf += readin
          if len(bf) > 4:
          msglen = struct.unpack('!I', bf[:4])[0]
          if len(bf) >= msglen + 4:
                    (msglen, msg) = struct.unpack("!I%ds" % msglen, bf[:msglen + 4])
                    # bf = bf[struct.calcsize("!I%ds" % msglen):]
                    (client_address, server_address, udp) = pickle.loads(msg)
                    if config_dict['encryption']:
                    udp = creeper.decrypt(udp)

                    log("(TCP) received UDP from ", client_address, " addressed to ", server_address)

                    if srvcallback is not None:
                    r = srvcallback(client_address, sendingto, tcp_socket, tcp_socket_lock)

                    if sendpktcallback is not None:
                    sendpktcallback(client_address, server_address, sendingto, udp)

          except socket.timeout:
          log("Socket read timed out for ", tcp_socket)
          finally:
          tcp_socket_lock.release()
          return r


class ThreadedTCPServer(SocketServer.ThreadingMixIn, SocketServer.TCPServer):
          def __init__(self, server_address, request_handler_class, bind_and_activate=True):
          SocketServer.TCPServer.allow_reuse_address = True
          SocketServer.TCPServer.__init__(self, server_address, request_handler_class, bind_and_activate)


class ThreadedTCPHandler(SocketServer.BaseRequestHandler):
          def handle(self):
          def start_udp_server(claddress, sendingto, tcp_socket, tcp_socket_lock):
          rport = claddress[1]
          with udpserverslock:
                    if not udpservers.has_key(claddress):
                    log("(TCP) started UDP server for ", claddress)
                    udpservers[claddress] = ThreadedUDPServer(('0.0.0.0', 0), ThreadedUDPHandler)
                    udpportnat[:rport] = udpservers[claddress].socket.getsockname()[1]
                    udpportnat[udpservers[claddress].socket.getsockname()[1]:] = rport
                    self.server_address = udpservers[claddress].socket.getsockname()

                    log("(TCP) bound incoming port ", rport, " as NAT ", (sendingto, udpportnat[:rport]))
                    udpservers[claddress].set_tcp_socket(tcp_socket, tcp_socket_lock)
                    server_thread = threading.Thread(target=udpservers[claddress].serve_forever)
                    server_thread.setDaemon(True)
                    server_thread.start()

                    return (claddress, udpservers[claddress])
                    else:
                    return None

          # fwdto = None
          # fa = None
          udps = []

          nb = ''
          while len(nb) < 128:
          fwdto = self.request.recv(128)
          if len(fwdto) == 0:
                    raise socket.error
          nb += fwdto
          src, nm = [x.lstrip().rstrip() for x in nb[:128].split('\n')]
          sendingto = "127.0.0.1"
          log("(TCP) started server forwarding to ", nm, " as ", sendingto, " for other source ", src)
          # nb = nm[128:]

          bf = ''
          try:
```

```
          while True:
                    (rs, ws, ex) = select.select([self.request], [], [])
                    r = read_tcp(self.request, threading.Lock(), bf, sendingto, start_udp_server, serversendpktcallback)
                    if r is not None:
                    udps.append(r)
          except (socket.error,), e:
          log(e)
          self.request.close()

          with udpserverslock:
          log("(TCP) server for ", nm, " with source ", src, " stopping ", len(udps), " UDP server(s)")
          for u in udps:
                    u[1].shutdown()
                    del udpservers[u[0]]
          log("(TCP) shutting down server forwarding for ", nm, " as ", sendingto, " other source ", src)


##############################################################################################################
##############################################################################################################

if __name__ == '__main__':
          try:
          opts, args = getopt.getopt(sys.argv[1:], "hcs", ["help", "client"])
          except getopt.GetoptError, err:
          print str(err)
          sys.exit()

          client = False
          for o, a in opts:
          if o in ('-c', '--client'):
          client = True
          elif o in ('-h', '--help'):
          usage()
          sys.exit()
          else:
          assert False, "unhandled option"

          localaddress = config_dict['clientaddr']
          serveraddress = config_dict['serveraddr']
          remotefwdto = config_dict['destaddr']
          firstport = None
          port = config_dict['destport']
          udpports = config_dict['clientport']

          if client:
          log("Running as client")
          if config_dict['mode'] == 1:
          th = Text(True, config_dict)
          th.start()
          elif config_dict['mode'] == 2:
          th = Voice(True, config_dict)
          th.start()
          else:
          th = FileAudio(True, config_dict)
          th.start()

          if config_dict['do_kand']:
          print 'doing kand'
          kand = Kand(True, config_dict)
          kand.start()

          if config_dict['gui']:
          g = GUI(th)
          g.start()

          for rport in udpports:
          with udpserverslock:
                    udpservers[(localaddress, rport)] = ThreadedUDPServer((localaddress, rport), ThreadedUDPHandler)
                    udpservers[(localaddress, rport)].set_client_mode()
                    udpportnat[rport:] = rport
                    udpportnat[:rport] = rport
                    server_thread = threading.Thread(target=udpservers[(localaddress, rport)].serve_forever)
                    server_thread.setDaemon(True)
                    server_thread.start()

          while True:
          rs = [k.tcp_socket for k in tcpconnections.values() if k.tcp_socket is not None]
          ws = None
```

```
          ex = None
          (rs, ws, ex) = select.select(rs, [], [], 10)
          for v in rs:
                    try:
                    try:
                              o = tcpconnections[tcpconnsock2key[v]]
                    except KeyError:
                              log(tcpconnections.items())
                              raise
                    assert o.tcp_socket == v
                    read_tcp(o.tcp_socket, o.tcp_socket_lock, o.bf, o.src[0], None, clientsendpktcallback)

                    except socket.error, e:
                    log("(TCP) client connection for ", o.src, " was closed")
                    del tcpconnections[tcpconnsock2key[o.tcp_socket]]
                    del tcpconnsock2key[o.tcp_socket]
                    o.tcp_socket.close()
                    del o

          else:
          if config_dict['do_kand']:
          print 'doing kand'
          kand = Kand(False, config_dict)
          kand.start()
          if config_dict['mode'] == 0:
          th = FileAudio(False, config_dict)
          elif config_dict['mode'] == 1:
          th = Text(False, config_dict)
          elif config_dict['mode'] == 2:
          th = Voice(False, config_dict)
          else:
          log('config file error')
          sys.exit()
          th.start()

          if config_dict['gui']:
          g = GUI(th)
          g.start()

          log("Running as server")
          server = ThreadedTCPServer(('', port), ThreadedTCPHandler)
          server.serve_forever()
```

./config.py

---

```
import ConfigParser

class Config:
          def __init__(self):
          self.configfile = 'config.txt'
          self.cp = ConfigParser.ConfigParser()
          self.cp.read(self.configfile)

          def readall(self):
          cfg = dict()

          cfg['logging'] = self.cp.getboolean('general', 'logging')
          a = self.cp.get('general', 'mode')
          if a == 'debug':
          cfg['mode'] = 0
          elif a == 'text':
          # mode 0 = debug (wav trans)
          # mode 1 = text
          # mode 2 = voice
          cfg['mode'] = 1
          elif a == 'voice':
          cfg['mode'] = 2
          cfg['gui'] = self.cp.getboolean('gui', 'enabled')

          cfg['clientaddr'] = self.cp.get('client', 'address')
          cfg['serveraddr'] = self.cp.get('server', 'address')
          cfg['destport'] = self.cp.getint('destination', 'port')
          cfg['destaddr'] = self.cp.get('destination', 'address')
          cfg['clientport'] = [self.cp.getint('client', 'port')]
```

```
        cfg['key'] = self.cp.get('crypto', 'key').decode('string_escape')
        cfg['init_vector'] = self.cp.get('crypto', 'init_vector').decode('string_escape')
        cfg['encryption'] = self.cp.getboolean('crypto', 'enabled')

        cfg['do_kand'] = self.cp.getboolean('kand', 'do_kand')
        cfg['kand_address'] = self.cp.get('kand', 'address')
        cfg['kand_port'] = self.cp.getint('kand', 'port')
        # print 'key =', cfg['key'].encode('string_escape')
        # print 'init_vector =', cfg['key'].encode('string_escape')

        cfg['channels'] = self.cp.getint('audio', 'channels')
        cfg['rate'] = self.cp.getint('audio', 'rate')
        cfg['frames_per_buffer'] = self.cp.getint('audio', 'frames_per_buffer')
        cfg['delay'] = float(self.cp.get('general', 'delay'))

        return cfg

        def read(self, section, item):
        return self.cp.get(section, item)

        def save(self, section, item, value):
        self.cp.set(section, item, value)
        self.cp.write(open(self.configfile, 'w'))
```

./cry.py

---

```
__author__ = 'viren'

import Crypto.Cipher.AES as AES
import Crypto.Random.OSRNG as OSRNG


# uses AES256 by default
class Cry:
        def __init__(self, key, init_vector, block_size=16, key_size=32):
        self.block_size = block_size
        self.key_size = key_size
        self.init_vector = init_vector         #self.gen_init_vector(block_size)
        self.key = key      #self.gen_key(key_size)
        self.aes = AES.new(self.key, AES.MODE_CBC, self.init_vector)

        def pad(self, data):
        if len(data) % self.block_size == 0:
        return data
        pad_len = 15 - (len(data) % self.block_size)
        data = '%s\x80' % data
        return '%s%s' % (data, '\x00' * pad_len)

        def unpad(self, data):
        if not data:
        return data
        data = data.rstrip('\x00')
        if data[-1] == '\x80':
        return data[:-1]
        else:
        return data

        def encrypt(self, data):
        return self.aes.encrypt(self.pad(data))

        def decrypt(self, data):
        return self.unpad(self.aes.decrypt(data))


def gen_init_vector(block_size=16):
        try:
        return OSRNG.posix.new().read(block_size)
        except:
        return OSRNG.new().read(block_size)


def gen_key(key_size=32):
        try:
        return OSRNG.posix.new().read(key_size)
        except:
```

```
        return OSRNG.new().read(key_size)
```

./comm.py

---

```
import pyaudio
import threading
import socket
import wave
# import time
import Queue


class Text(threading.Thread):
        def __init__(self, client_mode, param):
                threading.Thread.__init__(self)

                self.queue_lock = threading.Lock()
                self.queue_msg = Queue.Queue(64)

                self.daemon = True
                self.client_mode = client_mode

                self.addr = param['clientaddr']
                self.port = param['clientport'][0]

                self.msg = ''
                self.msg_list = []

                if client_mode:
                self.s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
                else:
                self.s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
                self.s.bind((param['destaddr'], param['clientport'][0]))

        def run(self):
                if self.client_mode:
                while True:
                        #self.msg = raw_input('>>> ')
                        if self.msg != '':
                        self.s.sendto(self.msg, (self.addr, self.port))
                        self.msg = ''
                else:
                i = 0
                while True:
                        dat, addr = self.s.recvfrom(1024)
                        print '>>> ', addr, ':', dat
                        #with self.queue_lock:
                        #         self.queue_msg.put(dat)
                        self.msg_list.append(dat)


        def get_message(self):
                msg_list = self.msg_list
                self.msg_list = []
                return msg_list

                #with self.queue_lock:
                #         if not self.queue_msg.empty():
                #         d1 = self.queue_msg.get()
                #         return d1
                #         else:
                #         return None

        def put_message(self, msg):
                self.msg = msg


class Voice(threading.Thread):
        def __init__(self, client_mode, param):
        threading.Thread.__init__(self)

        self.param = param
        self.client_mode = client_mode
        self.addr = param['clientaddr']
        self.port = param['clientport'][0]
```

```
        self.enabled = True
        self.daemon = True
        self.p = pyaudio.PyAudio()
        self.stream = self.p.open(format=pyaudio.paInt16,
                                  channels=param['channels'],
                                  rate=param['rate'],
                                  frames_per_buffer=param['frames_per_buffer'],
                                  input=True,
                                  output=True)

        if client_mode:
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        else:
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.sock.bind((param['destaddr'], param['clientport'][0]))

        def run(self):
        if self.client_mode:
        while True:
                  if self.enabled:
                  data = self.stream.read(self.param['frames_per_buffer'])
                  self.sock.sendto(data[:1024], (self.addr, self.port))
                  self.sock.sendto(data[1024:], (self.addr, self.port))

        else:
        while True:
                  if self.enabled:
                  data, addr = self.sock.recvfrom(1024)
                  self.stream.write(data)


class FileAudio(threading.Thread):
        def __init__(self, client_mode, param):
        threading.Thread.__init__(self)

        self.param = param
        self.client_mode = client_mode
        self.addr = param['clientaddr']
        self.port = param['clientport'][0]

        self.enabled = True
        self.daemon = True

        self.wf = wave.open('sample.wav', 'rb')
        self.p = pyaudio.PyAudio()
        self.stream = self.p.open(format=pyaudio.paInt16,
                                  channels=2,
                                  rate=44100,
                                  frames_per_buffer=256,
                                  output=True)

        if client_mode:
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        else:
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.sock.bind((param['destaddr'], 4000))

        def run(self):
        if self.client_mode:
        while True:
                  if self.enabled:
                  data = self.wf.readframes(256)
                  if len(data) == 0:
                            break
                  self.sock.sendto(data, (self.addr, self.port))

        else:
        while True:
                  if self.enabled:
                  data, addr = self.sock.recvfrom(1024)
                  self.stream.write(data)
```

./gui.py

---

```
import wx
```

```
import wx.py

import subprocess
import wx.lib.scrolledpanel as scrolled

from config import *

import threading
import Queue
import time
from subprocess import Popen, PIPE


class BashProcessThread(threading.Thread):
          def __init__(self, readlineFunc):
          threading.Thread.__init__(self)
          self.readlineFunc = readlineFunc
          self.outputQueue = Queue.Queue()
          self.setDaemon(True)

          def run(self):
          while True:
          line = self.readlineFunc()
          self.outputQueue.put(line)

          def getOutput(self):
          lines = []
          while True:
          try:
                    line = self.outputQueue.get_nowait()
                    lines.append(line)
          except Queue.Empty:
                    break
          return ''.join(lines)

class Interpretor(object):
          def __init__(self, locals, rawin, stdin, stdout, stderr):
          self.introText = "shell inbuilt unsupported"
          self.locals = locals
          self.revision = 1.0
          self.rawin = rawin
          self.stdin = stdin
          self.stdout = stdout
          self.stderr = stderr

          self.more = False
          self.bp = Popen('bash', shell=False, stdout=PIPE, stdin=PIPE, stderr=PIPE)

          self.outputThread = BashProcessThread(self.bp.stdout.readline)
          self.outputThread.start()

          self.errorThread = BashProcessThread(self.bp.stderr.readline)
          self.errorThread.start()

          def getAutoCompleteKeys(self):
          return [ord('\t')]

          def getAutoCompleteList(self, *args, **kwargs):
          return []

          def getCallTip(self, command):
          return ""

          def push(self, command):
          command = command.strip()
          if not command: return
          self.bp.stdin.write(command+"\n")
          time.sleep(0.1)

          self.stdout.write(self.outputThread.getOutput())
          self.stderr.write(self.errorThread.getOutput())

class getmsg(threading.Thread):
    def __init__(self,th,tc1):
           threading.Thread.__init__(self)
           self.th = th
           self.tc1=tc1
    def run(self):
```

```
          while True:
                    x = self.th.get_message()
                    for i in x:
                              self.tc1.AppendText("Them : "+i+"\n")
                    time.sleep(1)

class PageOne(wx.Panel):
    def __init__(self, parent,th):
          self.th = th

          wx.Panel.__init__(self, parent)
          #self.tc1 = wx.TextCtrl(self, -1, '', (10,10), (320,200))
          self.tc1 = wx.TextCtrl(parent = self, id = -1, pos = (10, 10), size = (320, 200), style =
wx.TE_MULTILINE|wx.TE_READONLY|wx.TE_AUTO_URL|wx.TE_RICH)
          #run.TextChat
          #self.tc2 = wx.TextCtrl(self, -1, '', (10,220), (320,80))
          self.tc2 = wx.TextCtrl(parent = self, id = -1, pos = (10, 220), size = (320, 80), style = wx.TE_MULTILINE)
          button1 = wx.Button(self, -1, "send",pos=(240,310))
          button1.Bind(wx.EVT_BUTTON, self.OnClick)
          self.gm=getmsg(th,self.tc1)
          self.gm.start()

    def OnClick(self,event):
          str = self.tc2.GetValue()
          self.th.put_message(str)
          self.tc2.Clear()
          self.tc1.AppendText("Me: "+str+"\n")
          # self.fg = wx.Colour(200,80,100)
  #       self.at = wx.TextAttr(fg)
  #       self.status_area.SetStyle(1, 3, at)



class PageTwo(scrolled.ScrolledPanel):
    def __init__(self, parent):
          self.cfg = Config()
          scrolled.ScrolledPanel.__init__(self, parent, -1)

          vbox = wx.BoxSizer(wx.VERTICAL)

          Lbl = wx.StaticBox(self, -1, 'Client')
          vbox1 = wx.StaticBoxSizer(Lbl, wx.VERTICAL)

          hbox=wx.BoxSizer(wx.HORIZONTAL)
          ca = wx.StaticText(self, -1, "Address  ")
          hbox.Add(ca,5)
          str = self.cfg.read('client','address')
          self.tca = wx.TextCtrl(self, -1, str, (100,20))
          hbox.Add(self.tca,3)
          vbox1.Add(hbox,1)

          hbox=wx.BoxSizer(wx.HORIZONTAL)
          cp = wx.StaticText(self, -1, "Port  ")
          hbox.Add(cp,5)
          str = self.cfg.read('client','port')
          self.tcp = wx.TextCtrl(self, -1, str, (100,20))
          hbox.Add(self.tcp,3)
          vbox1.Add(hbox,1)

          vbox.Add(vbox1,2)

          Lbl2 = wx.StaticBox(self, -1, 'Server')
          vbox2 = wx.StaticBoxSizer(Lbl2, wx.VERTICAL)
          hbox=wx.BoxSizer(wx.HORIZONTAL)
          sa = wx.StaticText(self, -1, "Address  ")
          hbox.Add(sa,5)
          str = self.cfg.read('server','address')
          self.tsa = wx.TextCtrl(self, -1, str, (100,20))
          hbox.Add(self.tsa,3)
          vbox2.Add(hbox,1)
          vbox.Add(vbox2,1)

          Lbl3 = wx.StaticBox(self, -1, 'Destination')
          vbox3 = wx.StaticBoxSizer(Lbl3, wx.VERTICAL)
          hbox=wx.BoxSizer(wx.HORIZONTAL)
          da = wx.StaticText(self, -1, "Address  ")
          hbox.Add(da,5)
          str = self.cfg.read('destination','address')
```

```
self.tda = wx.TextCtrl(self, -1, str, (100,20))
hbox.Add(self.tda,3)
vbox3.Add(hbox,1)

hbox=wx.BoxSizer(wx.HORIZONTAL)
dp = wx.StaticText(self, -1, "Port  ")
hbox.Add(dp,5)
str = self.cfg.read('destination','port')
self.tdp = wx.TextCtrl(self, -1, str, (100,20))
hbox.Add(self.tdp,3)
vbox3.Add(hbox,1)
vbox.Add(vbox3,2)

Lbl4 = wx.StaticBox(self, -1, 'Audio')
vbox4 = wx.StaticBoxSizer(Lbl4, wx.VERTICAL)
hbox=wx.BoxSizer(wx.HORIZONTAL)

af = wx.StaticText(self, -1, "format  ")
hbox.Add(af,4)
formatlist = [ 'paInt 16','paInt 32' ]
self.choice1 = wx.Choice(self, -1, choices=formatlist)
self.choice1.SetSelection(2)
self.choice1.SetFocus()
self.choice1.Bind(wx.EVT_CHOICE, self.OnClick)
hbox.Add(self.choice1,3)

# str = self.cfg.read('audio','format')
# self.taf = wx.TextCtrl(self, -1, str, (100,20))

vbox4.Add(hbox,1)

hbox=wx.BoxSizer(wx.HORIZONTAL)
ac = wx.StaticText(self, -1, "channels  ")
hbox.Add(ac,5)
str = self.cfg.read('audio','channels')
self.tac = wx.TextCtrl(self, -1, str, (100,20))
hbox.Add(self.tac,3)
vbox4.Add(hbox,1)

hbox=wx.BoxSizer(wx.HORIZONTAL)
ar = wx.StaticText(self, -1, "rate  ")
hbox.Add(ar,5)
str = self.cfg.read('audio','rate')
self.tar = wx.TextCtrl(self, -1, str, (100,20))
hbox.Add(self.tar,3)
vbox4.Add(hbox,1)

hbox=wx.BoxSizer(wx.HORIZONTAL)
afpb = wx.StaticText(self, -1, "frames_per_buffer  ")
hbox.Add(afpb,5)
str = self.cfg.read('audio','frames_per_buffer')
self.tafpb = wx.TextCtrl(self, -1, str, (100,20))
hbox.Add(self.tafpb,3)
vbox4.Add(hbox,1)

vbox.Add(vbox4,4)

Lbl5 = wx.StaticBox(self, -1, 'Crypto')
vbox5 = wx.StaticBoxSizer(Lbl5, wx.VERTICAL)

hbox = wx.BoxSizer(wx.HORIZONTAL)
cca = wx.StaticText(self, -1, "algorithm  ")
hbox.Add(cca,7)
cryptlist = [ 'AES', 'asd' ]
self.choice2 = wx.Choice(self, -1, choices=cryptlist)
self.choice2.SetSelection(2)
self.choice2.SetFocus()
self.choice2.Bind(wx.EVT_CHOICE, self.OnClick)
hbox.Add(self.choice2,3)
vbox5.Add(hbox,1)

hbox=wx.BoxSizer(wx.HORIZONTAL)
ccb = wx.StaticText(self, -1, "block_mode  ")
hbox.Add(ccb,5)
str = self.cfg.read('crypto','block_mode')
self.tccb = wx.TextCtrl(self, -1, str, (100,20))
hbox.Add(self.tccb,3)
vbox5.Add(hbox,1)
```

```
            hbox=wx.BoxSizer(wx.HORIZONTAL)
            ccbs = wx.StaticText(self, -1, "block_size  ")
            hbox.Add(ccbs,5)
            str = self.cfg.read('crypto','block_size')
            self.tccbs = wx.TextCtrl(self, -1, str, (100,20))
            hbox.Add(self.tccbs,3)
            vbox5.Add(hbox,1)

            hbox=wx.BoxSizer(wx.HORIZONTAL)
            t1 = wx.StaticText(self, -1, "key size  ")
            hbox.Add(t1,5)
            str = self.cfg.read('crypto','key_size')
            tc2 = wx.TextCtrl(self, -1, str, (100,20))
            hbox.Add(tc2,3)
            vbox5.Add(hbox,1)


            hbox=wx.BoxSizer(wx.HORIZONTAL)
            t1 = wx.StaticText(self, -1, "init_vector  ")
            hbox.Add(t1,5)
            tc2 = wx.TextCtrl(self, -1, '', (100,20))
            hbox.Add(tc2,3)
            vbox5.Add(hbox,1)

            hbox=wx.BoxSizer(wx.HORIZONTAL)
            t1 = wx.StaticText(self, -1, "key  ")
            hbox.Add(t1,5)
            tc2 = wx.TextCtrl(self, -1, '', (100,20))
            hbox.Add(tc2,3)
            vbox5.Add(hbox,1)

            vbox.Add(vbox5,6)

            button1=wx.Button(self, -1, "Save Changes")
            vbox.Add(button1,1)
            button1.Bind(wx.EVT_BUTTON, self.OnClick)

            self.SetSizer(vbox)
            self.SetAutoLayout(1)
                     self.SetupScrolling()


    def OnClick(self,event):
           self.cfg.save('client','address',self.tca.GetValue())
           self.cfg.save('client','port',self.tcp.GetValue())
           print self.tsa.GetValue()
           self.cfg.save('server','address',self.tsa.GetValue())

           self.cfg.save('destination','address',self.tda.GetValue())
           self.cfg.save('destination','port',self.tdp.GetValue())

           self.cfg.save('audio','format',self.choice1.GetStringSelection())
           self.cfg.save('audio','channels',self.tac.GetValue())
           self.cfg.save('audio','rate',self.tar.GetValue())
           self.cfg.save('audio','frames_per_buffer',self.tafpb.GetValue())

           self.cfg.save('crypto','algorithm',self.choice2.GetStringSelection())
           self.cfg.save('crypto','block_mode',self.tccb.GetValue())
           self.cfg.save('crypto','block_size',self.tccbs.GetValue())


class PageThree(wx.Panel):
    def __init__(self, parent):
           wx.Panel.__init__(self, parent)
           #t = wx.StaticText(self, -1, "This is a PageThree object", (50,50))
           wx.py.shell.Shell(self,size=(350,450),InterpClass=Interpretor)


class MainFrame(wx.Frame):
    def __init__(self,th,kand):
           wx.Frame.__init__(self, None,-1, title="VoIP",style= wx.SYSTEM_MENU | wx.CAPTION | wx.CLOSE_BOX,size=(350,450))
           self.kand = kand

           p = wx.Panel(self,pos=(0,0),size=(350,40))
           p1 = wx.Panel(self,pos=(0,40),size=(350,450))
           nb = wx.Notebook(p1,size=(350,450))
           self.cb1=wx.CheckBox(p, -1 ,'Disable Voice', (0, 10))
```

```
            page1 = PageOne(nb,th)
            page2 = PageTwo(nb)
            page3 = PageThree(nb)
            #self.button1=wx.Button(p, -1, "Connect",pos=(250,5))
            #self.button1.Bind(wx.EVT_BUTTON, self.OnClick)

            nb.AddPage(page1, "Chat")
            nb.AddPage(page2, "Settings")
            nb.AddPage(page3, "Shell")

            sizer = wx.BoxSizer(wx.VERTICAL)
            #sizer.Add(self.button1,0)
            sizer.Add(nb, 2)
            p.SetSizer(sizer)
            #self.SetSizer(vbox)
                     self.Bind(wx.EVT_CHECKBOX, self.OnCb1, self.cb1)

    def OnCb1(self, evt):
           #self.cb2.SetValue(evt.IsChecked())
           #self.th.enable()
           self.kand.switch()

    #def OnClick(self, e):
    #    p = subprocess.call("run.py -c",shell=True)
    #    self.button1.SetLabel("disconnect")

class GUI(threading.Thread):
    def __init__(self,th):
           self.th=th
           self.kand = kand
           self.app = wx.App()
           threading.Thread.__init__(self)

    def run(self):
           m = MainFrame(self.th)
           m.Show()
           self.app.MainLoop()


if __name__ == "__main__":
    g = GUI()
    g.start()
```