Utilizing Multi-core for optimized Data Exchange via VoIP

SOHRAB AZAMI GHADIM
Utilizing Multi-core for optimized Data Exchange via VoIP

BY
SOHRAB AZAMI GHADIM

Thesis
Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Biomedical Engineering/Image Processing
in the Graduate College of the
Royal Institute of Technology at Stockholm

Stockholm-Sweden

Advisors:
Professor Ahmed Hemani
Dr. Iyad Al Khatib
To Anybody cares
ACKNOWLEDGEMENTS

This project would not have been possible without the support of many people. Many thanks to my adviser, Ahmed Hemani, who read my numerous revisions and helped. Also thanks to my technical advisor, Iyad AlKhatib, who helped me with technical help whenever I needed. Also thanks to my school coordinator, Mats Nilsson, who offered guidance and support. Thanks to the Royal Institute of Technology Graduate College for awarding this opportunity to work on this project, providing me with any means to complete this project. And finally, thanks to my family and numerous friends who endured this long process with me, always offering support and love.
# Contents

List of Tables ......................................................................................................................................... 7
List of Figures ........................................................................................................................................ 7

## CHAPTER 1: Introduction

1.1. Motivation .................................................................................................................................. 10
1.2. Contribution ................................................................................................................................. 11
1.3. Organization of thesis .................................................................................................................. 12

## CHAPTER 2: VoIP Background

2.1. Introduction .................................................................................................................................. 13
2.2. Communication protocols ............................................................................................................. 13
2.3. Hour glass design ......................................................................................................................... 15
2.4. SIP/Mobile voice & video communication .................................................................................... 16
2.5. Session Initiation Protocol ........................................................................................................... 16
2.6. Functions of SIP ........................................................................................................................... 17
2.7. Components of SIP ....................................................................................................................... 17
2.7.1. Clients: ..................................................................................................................................... 18
2.7.2. Servers: .................................................................................................................................... 18
2.8. Commands of SIP .......................................................................................................................... 18
2.9. A Typical Example of SIP Session ................................................................................................ 19
2.10. Structure of SIP Protocol ............................................................................................................ 21
2.11. Registration in SIP ....................................................................................................................... 23
2.12. PJSIP Implementation .................................................................................................................. 29
2.12.1. PJSIP: SIP & Multimedia Communication Library ............................................................... 29
2.12.2. Features .................................................................................................................................. 29
2.12.3. Architecture ............................................................................................................................ 29
2.12.4. The Endpoint .......................................................................................................................... 30
2.12.5. Module .................................................................................................................................... 31
2.12.6. Module Capabilities ............................................................................................................... 31
2.3.2. Message Elements .................................................................................................................... 32
2.3.2.1. Universal Resource Indicator ............................................................................................. 32
2.3.3. SIP Message in PJSIP ................................................................................................................. 33
2.3.3.1. Header Fields ........................................................................................................................ 34
2.3.3.2. SIP Status Codes .................................................................................................................. 34
2.3.4. Sending Messages ..................................................................................................................... 34
2.3.5. Transactions ............................................................................................................................... 35
2.3.6. Timers and Retransmission ....................................................................................................... 35
2.4. RTP Header .................................................................................................................................................. 35
2.5. RTP ................................................................................................................................................................ 36
2.6. RTP/RTCP Features Description and UDP vs. TCP .................................................................................. 38
2.7. RTP Timestamp and Sequence number ..................................................................................................... 39
2.8. RTCP: Real-Time Control Protocol ............................................................................................................. 40
2.8.1. Tasks of RTCP ......................................................................................................................................... 40
2.8.2. RTCP Packets structure ......................................................................................................................... 40
2.8.3. RTCP Header structure ............................................................................................................................ 40
2.8.4. SR Report Format .................................................................................................................................... 41
2.8.5. RTCP Source Description, SDES ........................................................................................................... 41
2.9. Related Work ................................................................................................................................................. 42
2.10. Summary: .................................................................................................................................................... 43

CHAPTER 3: iRTP protocol .................................................................................................................................... 44
3.1. Introduction .................................................................................................................................................... 44
3.2. iRTP Structure ............................................................................................................................................ 44
3.3. Summary: .................................................................................................................................................... 46

CHAPTER 4: Parallel Design ............................................................................................................................... 47
4.1. Introduction ................................................................................................................................................... 47
4.2. VoIP Parallel Architecture ............................................................................................................................ 47
4.3. VOIP Parallel Architecture .......................................................................................................................... 47
4.4. Design of VOIP Parallel Architecture with OOD/OOP ............................................................................. 52
4.4.1. UML Representation of VOIP Parallel Architecture ............................................................................. 52
4.4.2. Design of RTP Architecture .................................................................................................................. 53
4.4.3. Algorithms of VoIP multimedia using RTP protocol ............................................................................. 53
4.4.4. VoIP architecture over iRTP protocol ................................................................................................... 55
4.4.5. Design of VOIP multimedia objects with iRTP protocol ................................................................... 56
4.4.6. Algorithms of VOIP using iRTP protocol ............................................................................................. 56
4.4.6.1. Client1 & Client2 sides ....................................................................................................................... 56
4.5. SIP Architecture .......................................................................................................................................... 57
4.5.1. SIP and RTP/iRTP state diagram architecture ..................................................................................... 58
5.6. Summary ....................................................................................................................................................... 59

CHAPTER 5: Test and Results ............................................................................................................................. 61
5.1. Introduction ................................................................................................................................................... 61
5.2. Test Design .................................................................................................................................................. 61
5.4.1. Only Audio (OA) ................................................................................................................................... 62
5.4.1.1. Only Audio over RTP (O AoRTP) ...................................................................................................... 63
5.4.1.1.1. LAN setting for (O AoRTP) ........................................................................................................... 63
List of Tables

Table 1: User agent (ANDY) is trying to establish a session .......................................................... 25
Table 2: DNS Server is not necessary, since ANDY’s proxy server is .................................................. 26
Table 3: DAVID eventually is signed up and ANDY ................................................................. 27
Table 4: User agent (ANDY) can try to establish a SIP session again .............................................. 28

Table of Figures

Figure 1: Real-time Multimedia, Protocols [12] ........................................................................... 14
Figure 2: Network layers organization ISO standard [12] ................................................................. 14
Figure 3: Hour glass design ........................................................................................................... 15
Figure 4: Example of two web application communicating via http .................................................. 15
Figure 5: IP/ISDN gateway and video MCU/bridge for PSTN connectivity to LAN for video along ...... 16
Figure 6: Example of a SIP Response message sent by user2 to user1 for a Request sent by user1 ......... 19
Figure 7: A simple SIP session with SIP trapezoid [1] ................................................................... 20
Figure 8: SIP negotiation diagram [1] ......................................................................................... 21
Figure 9: Typical Structure of the SIP protocol [13] ....................................................................... 22
Figure 10: A typical SIP Messaging followed by a RTP multimedia streaming [12] ......................... 24
Figure 11: SIP messaging for only one agent signed up [12] ............................................................. 25
Figure 12: ANDY sends SUBSCRIBE message to DAVID, to get notifications when he is available [12] 26
Figure 13: Ex-Not-signed-in user agent eventually is signed up and available [12] ......................... 27
Figure 14: ANDY calls DAVID when being acknowledged DAVID is available [12] ....................... 28
Figure 15: Collaboration Diagram [8] .......................................................................................... 29
Figure 16: URI class diagram [8] ................................................................................................. 32
Figure 17: pjsip header class diagram [8] ..................................................................................... 33
Figure 18: Code 21 SIP status code constants [8] ......................................................................... 34
Figure 19: Transfer latency in packet arrival [12] ......................................................................... 37
Figure 20: Jitter in Packet arrival [12] .......................................................................................... 37
Figure 21: SDP format for an audio conference ............................................................................. 38
Figure 22: RTP packet sequence showing timestamp and sequence number [12] ......................... 39
Figure 23: iRTP over SIP data over datagram. iRTP is compatible with RTP. AD stands for .......... 44
Figure 24: Example of SIP Request Message description ............................................................... 45
Figure 25: Example of SIP Response Message ............................................................................. 45
Figure 26: VoIP architecture over RTP protocol ............................................................................ 49
Figure 27: RTP class Diagram in ATL COM technology ................................................................. 53
Figure 28: Architecture in UML representation over iRTP Protocol ............................................... 54
Figure 29: RTP/iRTP architecture and SIP state diagram architecture, circled numbers show .......... 58
Figure 30: end-to-end latency measurement for international testbed .......................................... 61
Figure 31: end-to-end latency measurement for city network testbed ............................................ 62
Figure 32: End-to-end latency measurement for Local Area Network testbed ............................... 62
Figure 33: Averages of latency for LAN setting for Only Audio over RTP test runs ..................... 63
Figure 34: Averages of latency for Only Audio over RTP-LAN versus .......................................... 64
Figure 35: Averages of Latency for Only Audio over RTP for one country setting test run ............. 64
Figure 36: Averages of Latency for Only Audio over iRTP test run for LAN setting test run ........... 65
Figure 37: Average of Only Audio over RTP versus Average of Only Audio over iRTP ............... 66
Figure 38: Averages of latency for Only Audio over iRTP for one country setting test run.........................66
Figure 39: Averages of latency for Only Medical Data over TCP for LAN setting test run.........................67
Figure 40: Averages of Latency for Only Medical Data over TCP for one country test run.........................68
Figure 41: Only Medical Data over iRTP over LAN test run......................................................................69
Figure 42: Averages of latency for Only Medical Data over iRTP for one country test run.........................70
Figure 43: Averages of Latency for Only Audio over RTP and Medical Data over TCP for LAN.............71
Figure 44: Average Latency for Audio over RTP and Medical Data over TCP test run..............................71
Figure 45: Average Latency Audio over RTP & Medical Data over TCP; Client with All Wired.............72
Figure 46: Average Latency Audio over RTP and 12 channels Medical Data over TCP; Client and Server72
Figure 47: Average Latency Audio over RTP and Medical Data over TCP; Client and Server..................73
Figure 48: Average Latency Audio over RTP and Medical Data over TCP; Client with wired.............73
Figure 49: Average Latency Audio over RTP and Medical Data over TCP; Client with wireless............74
Figure 50: Largest Latency Audio over RTP, Medical Data over TCP, and audio and medical data over 74
IRTP..................................................................................................................................................75
Figure 51: Average Latency Audio over RTP, Medical Data over TCP, and audio and medical data over 75
IRTP..................................................................................................................................................75
Figure 52: Averages of Latency of several thousands packets for Audio and Medical Data over iRTP for 76
several test runs.....................................................................................................................................76
Figure 53: Averages of Latency for Audio and Medical Data over iRTP for several.................................76
Figure 54: Average Latency Audio & Medical Data over IRTP over 3G Network.................................77
Figure 55: Average Latency Audio & Medical Data over IRTP; Client & Server both have.................77
Figure 56: Average Latency Audio & Medical Data over IRTP; Client and Server............................78
Figure 57: Average Latency Audio & Medical Data over IRTP; Client and Server............................78
Figure 58: Average Latency Audio & Medical Data over IRTP; Client...............................................79
Figure 59: Latencies of all types of protocol per different geo-references. The two bars on the top of the 79
graph for international test is very important criterion to compare (RTP + TCP) vs. IRTP protocols........80
Figure 60: Flow chart to process captured packets results and delete unwanted captured packets........82
Abstract:

In contemporary IT industry, Multi-tasking solutions are highly regarded as optimal solutions, because hardware is equipped with multi-core CPUs. With multicore technology, CPUs run with lower frequencies while giving same or better performance as a whole system of processing. This thesis work takes advantage of multi-threading architecture in order to run different tasks under different cores such as SIP signaling and messaging to establish one or more SIP calls, capture voice, medical data, and packetize them to be streamed over internet to other SIP agents. VoIP is designed to stream voice over IP. There is inter-protocol communication and cooperation such as between the SIP, SDP, RTP, and RTCP protocols in order to establish a SIP connection and afterwards stream media over the Internet. We use the Microsoft COM technology in order to better the C++ component design. It allows us to design and develop code once and run it anywhere on different platforms. Using VC++ helps us reduce software design time and development time. Moreover, we follow software design standards setup by software engineers’ society. VoIP technology uses protocols such as the SIP signaling protocol to locate the user agents that communicate with each other. Pjsip is a library that allows developers to extend their design with SIP capability. We use the PJSIP library in order to sign up our own developed VoIP module to a SIP server over the Internet and locate other user agents. We implement and use the already-designed iRTP protocol instead of the RTP to stream media over the Internet. Thus, we can improve RTP packet delays and improve Quality of Service (QoS). Since medical data is critical and must not be lost, the iRTP guarantees no loss of medical data. If we want to stream voice only, we would not need iRTP, because RTP is a good protocol for voice applications. Due to the increasing Internet traffic, we need to use a reliable protocol that can detect packet loss of medical data. iRTP resolves the issue and leverages QoS. This thesis work focuses on streaming medical data and medical voice-calls using VoIP, even over small bandwidths and in high traffic periods. The main contribution of this thesis is in the parallel design of iRTP and the implementation of this very design in order to be used with multicore technology. We do so via multi-threading technology to speed up the streaming of medical data and medical voice-calls. According to our tests, measurements, and result analyses, the parallel design of iRTP and the multithreaded implementation on VC++ leverage performance to a level where the average decrease in delay is 76.46% when using iRTP for audio and medical data instead of the nowadays applied case of using an RTP stream for audio and multiple TCPs streams for medical data.
CHAPTER 1: Introduction

1.1. Motivation

The last decade has witnessed a growth in the need for better telemedicine applications to reduce the cost of healthcare, provide better and faster healthcare, and provide larger access to healthcare resources especially for remote areas. In Telemedicine, nowadays, hundreds of medical applications and systems try to send their information over heterogeneous networks, which may lead to higher than bearable use of bandwidth of parts of networks. Consequently, this slows down the entire communication. State of the art solutions, try to overcome this problem by investing in more network hardware (HW) resources. However, such solutions increase the cost, and the tradeoff between cost and quality in telemedicine is a very difficult issue since the telecommunication in medical applications prioritizes human life, i.e. the performance of the network.

We propose a solution that can utilize the same type of available networks, but solves the problem of performance via making new application-layer protocols that take into consideration the quality of service (QoS) for medical applications. At the same time, the protocol solution is not the only part that can aid in reaching our goal of decreasing cost while providing acceptable performance. The other part is to make use of the current HW designs (e.g. Multi-core) in order to improve computational power and decrease communication delay. Moreover, handheld devices are nowadays used to transmit vital real-time medical signals that need to be monitored constantly. Some applications require the transmission of multiple signals from the same patient to the same medical center. For example, the 12-lead electrocardiogram consists of 12 signals from the same patient, and the devices that store or transmit such signals may fill up to many giga bytes per patient per day. Therefore, network traffic grows rapidly, and the use of better protocols and faster electronic devices are necessary.

On the other hand, CPUs nowadays are designed with multi-core technology to reduce the clock rate, in order to decrease power consumption while the processing speed is the same. Therefore, there is a need to take into consideration three issues at the same time: the Multi-core HW (with its related parallel modules for faster computation), the protocol issue, and the QoS to satisfy medical needs. Therefore, in this work, we aim at designing a Voice over IP (VoIP) architecture that takes care of connecting the patients to medical centers in such a way that there could be several data and media signals concurrently sent from the same patient to the same center without loss in performance. In particular, we use Session Initiation Protocol (SIP) as the communication protocol to initiate a session, and we apply the required changes in this protocol to suit both: our real-time protocol (RTP) for data transmission and the paralleling for Multi-core HW. We use the iRTP protocol [15][16], which was designed already for transmitting multiple data sessions between the same sender and receiver. The iRTP implementation is, therefore, done mainly for medical applications. Our choice of multi-core and multi-threading is motivated by the fact that combining SIP, RTP, iRTP, parallel modules in the design of the solution for Multi-core can lead to sending the required data (with higher frequencies of data sampling) without degrading the QoS. We believe this way of using protocol design and parallel modules for Multi-core can also help in increasing battery life and satisfy medically critical requirements.

We use VoIP to stream data to network nodes that are mobile and independent of physical IP address. Thus, no device is bound to a physical IP address and the SIP protocol can locate the destination node over networks and then establish a connection where we can use different protocols to initiate a multimedia session (e.g. RTP & iRTP).
This design was already proposed by [15], where it allows text-based medical report at the same time of e medical call. In this thesis, we use the iRTP as it is and propose a re-design in its SW part to suit Multi-core by paralleling its modules. To keep medical institutions posted about their patients’ data and their state-of-health in real-time, Multi-core plays a large role in decreasing delay (keeping required performance level). Our solution is thus a paralleled iRTP that gives a boost to an efficient iRTP protocol that suits Multi-core and that serves the main aim of guaranteeing satisfactory performance for medical applications.

1.2. Contribution

We have designed and implemented a parallel set of modules for a SIP application, which can capture voice and medical data at the same time from one patient and stream it via the already designed iRTP protocol to a medical center. Moreover, unlike the classical iRTP, ours is paralleled iRTP to suit Multi-core.

We also implemented the paralleled FULLDUPLEX support for a stream of voice so that both sides can talk and listen to other user agents. Thus, user agents can have any discussion about the data they have prepared to be streamed from one user agent to another one. Voice capture is performed by the HW and then digitized and accessed by a software interface. Then, we use our paralleling for the Multi-core modules to divide the computation on multiple modules over many processors. This interface runs in a separate thread, because it is a background real-time task. The main thread does necessary initializations so that SIP application can start to operate. We use the PJSIP library for the core implementation of the basic SIP services such as registration, SIP negotiation (to decide about codec and media protocol to use to establish a media session), and the SIP signaling protocol. However, a part of our contribution includes the changes made in PJSIP in order to suit our parallel and medical needs at the same time. PJSIP allows us to create a multi-core based code. We could make more than one call to setup different types of media session in order to achieve our objectives. Each call is implemented in such a way that it runs in a different thread. Threads run in different cores of the CPU; therefore, it is very efficient, yet voice, video, and other media could be captured and conditioned to be packetized in real-time and sent over either RTP or iRTP.

We worked on a new implementation of the already available RTP and iRTP, and we used the Microsoft ATL COM technology. Thus, once the object is designed it can be modified, compiled, and built and new versions of the object can be used directly in the client without the client being rebuilt. The PJSIP library is written in C++ and it is not ATL COM, therefore, we faced a big challenge to migrate them together. Hence, a part of the contribution is to make this parallel code function within the required SW, HW, and medical needs using MS Windows and PJSIP. This can also be considered as a general contribution that can benefit both: Windows programmers and the PJSIP social network. Moreover, we customized the SIP application implementation in such a way that PJSIP library could fit the RTP and iRTP architectures. A difficult challenge that was solved was the use of our own voice capture implementation outside the PJSIP voice capture. The results show that iRTP improves performance drastically by decreasing delays for sending audio and data between two ends points. The iRTP shows around 74% decrease in delay than using the classical RTP with multiple (12) TCP connections that send the data of interest between two end points.
1.3. Organization of thesis

CHAPTER2: VoIP Background: explains protocols such UDP, TCP, SDP, SIP, and so forth. We touch upon SIP protocol and show how it evolved the way communication systems operate in a cheaper and more efficient way. We elaborate about how SIP message and response travel between two user agents to establish a SIP call. SIP protocol cooperates with different Protocols to create SIP services. At the end of chapter 2 we’d like to introduce PJSIP library as a good choice to create a SIP application that functions in a professional level.

Error! Reference source not found.: describes RTP (Real Time Protocol) which concerns about how to packetize real-time data in a RTP packet. RTCP (Real Time Control Protocol) which controls the traffic, therefore, there is no jitter, delay, and high QoS.

CHAPTER3: iRTP protocol: gives a brief explanation about a new Protocol which we implement in this thesis work. iRTP cooperates with SIP, RTP, and RTCP protocols in order to establish a faster network to stream real-time data especially medical data. We show iRTP protocol is superb to TCP protocol in both: speed and delay characteristics.

CHAPTER4: Parallel Design: elaborates about the architecture of the thesis work. It contains architecture design for both: RTP and iRTP. It explains about the blocks of the architecture and objects that play a major role in the architecture to implement the RTP, RTCP, and iRTP. There is an architecture which describes a state diagram of how SIP, RTP, RTCP, and iRTP interact with each other to establish a SIP call. SIP call is for voice and medical data stream.

CHAPTER 5: Test and Results: To test our design we design a testing strategy to measure network delays for different setting in local, country, and international networks. This chapter contains valuable graphs which illustrates the delays. We conclude from the results that iRTP has better performance than RTP in terms of delay of networks.

CHAPTER 6: Conclusion and Future Work: summarizes all the work we have done in previous chapters and gives a short explanation about the future continuation of this thesis work.
CHAPTER 2: VoIP Background

2.1. Introduction

Telecommunication technology has witnessed a drastic improvement with help of VoIP. It is a new technology which allows caller and callee establishes a phone call even though, they may be mobile. It uses IP layer to stream multimedia information. It replaces IP address with logical address. VoIP contains many protocols that cooperate to establish a phone call. Should a new VoIP service require a new protocol; we would design it and integrate it into VoIP. SIP is the back bone protocol of the VoIP. It is a simple signaling protocol which locates a user agent over networks and establishes a SIP call. SIP is extendable; therefore, it is possible to build a new SIP service. Multimedia information contains voice, video, and data which are streamed by RTP protocol. RTP is widely used protocol to stream voice and video. Data stream is an extensive field to discuss, but basically it needs to be conditioned before it is streamed over RTP. RTCP controls the QoS.

2.2. Communication protocols

Encapsulation is a technique to enforce the stack protocol form of the communication architecture in order to enable a connection between different logical and physical entities of network. We use TCP and UDP as major communication protocols which have different purposes for the different kinds of communication among nodes in the network, for instance with UDP we can establish process-to-process communication which is logical entity and it is not guaranteed that every packet reaches to the destination. UDP is a light protocol and less costly than TCP. As far as video conferencing is concerned SIP is the best feasible way to reduce the cost of calls, yet use internet as to stream multimedia packets.

Figure 1 shows real-time Multimedia streaming protocols. There are application layer protocols such as SIP, SDP, RTP, and RTCP. SIP is a simple signaling protocol to establish a connection among SIP devices. It cooperates with other protocols to do tasks that are not supported in SIP. QoS protocols help the communication system constantly check the traffic, regulate the speed of real-time multimedia data flow, and guarantees a good quality of service. SIP can work with both: TCP and UDP; however we use UDP protocol for this thesis work. UDP is a multicast protocol is for transmission of real-time multimedia stream which is used in VoIP. RTP is a real-time multimedia protocol that takes multimedia data with MPEG and H.261 (video compression for low end-to-end latency and less jitter) formats and transforms it into a RTP packet; it is finally encapsulated in UDP datagram in order to stream over networks.
Figure 1: Real-time Multimedia, Protocols [12]

Figure 2 shows how the encapsulation concept has divided the communication systems to different layers so that each new protocol follows ISO model and can cooperate with other protocols to provide new services and improve performance. SIP came in picture as an application layer in this entire ISO model just to change the thinking of how we can communicate with each other. SIP allows us to be mobile, yet to be accessible to the rest of the world via our mobile phone.

Figure 2: Network layers organization ISO standard [12]
It is shown in Figure 2 network layers in ISO model which is sliced into different levels.

1. Application Layer: one can implement the protocol in an application using encapsulation concept.
2. Transport Layer: transport layer ensures that source and destination addresses are correct to stream the packet.
4. Data Link Layer: this is the layer which deals with hardware such as Ethernet.
5. Physical Layer

2.3. Hour glass design

Figure 3 illustrates the so called hour glass design; UDP and TCP protocols are integrated into one IP layer; it is a single protocol at network layer that ensures packets will get from source to destination while allowing for flexibility.

![Hour glass design](image)

Figure 4 shows a connection between two hosts using TCP/IP protocol; the interesting point in this architecture is that two different protocols in data link layers are used. One is Ethernet and the other one is token ring and they are able to connect to each other through an IP layer.

![Example of two web application communicating via http](image)
2.4. SIP/Mobile voice & video communication

SIP (Session Initiation Protocol) is increasingly becoming major protocol for voice-video conferencing. On the contrary ISDN and other protocols are less important while SIP is evolving everyday; it is concerned about logical address whereas other conferencing protocols are concerned about IP address (physical address). Thus, it is a mobile communication protocol. It uses IP layer to stream real time data among hosts over RTP via internet.

SIP uses advanced signaling and controlling functionality over IP networks; it is vital for proper connectivity over IP for audio/video conferencing; SIP is instant messaging event notification which is widely used to control network devices. Signaling of SIP is exploited to setup the transport of audio/video via RTP.

Older version of conferencing systems use protocols such as ISDN. It has attracted huge amount of investment over years and to be able to use them a lot of work is under way to configure ISDN networks with IP layer as medium to stream multimedia data. IP/ISDN Gateway takes the role of conversion of ISDN conferencing over IP as a protocol for audio/video conferencing, this way PSTN connectivity to the LAN will be the ultimate solution. Figure 5 illustrates this configuration which allows a LAN system to access PSTN network video systems for multipoint video conferencing.

![Figure 5: IP/ISDN gateway and video MCU/bridge for PSTN connectivity to LAN for video along](image)

It is obvious a sipphone on LAN can make a video conference with another SIP phone in PSTN cloud.

2.5. Session Initiation Protocol

SIP (Session Initiation Protocol) is a signaling protocol which runs on IP based networks. It creates, manages, and terminates a session. A session can have different forms such as a simple telephone call or collective multi-media conference session. Since VoIP is an internet based which uses IP network to stream multimedia data, therefore, it has voice e-commerce, web applications, all kinds of technical applications, and SMS. SIP allows building buddy lists in an IP based environment which utilizes the categorization of contacts [14].
Main application of SIP is in VoIP (Voice over IP). RFC3261 are devised by Internet Engineering Task Force (IETF). It is under development to be adapted as a standard for telecommunication systems around the globe. Although it is a simple protocol, but has changed the way communication is conducted. It registers its clients to location service servers around the world; therefore, physical IP address concept is no longer concerned and is replaced by SIP address (sohrab14@iptel.org) [14].

It allows participants of a session negotiate about how they want to stream multimedia data among them. Codec is a parameter which caller and callee negotiate within a session before they join the session. The participants should be able to agree on the same protocol and codec [14].

It is a signaling protocol that provides primitive services. Thus, using the primitives enable us to build new services. We send SIP Request message to initiate a session. SDP is used to build a SDP OFFER. This offer is placed to other user agents to start a session. If a user agent sends a picture of the user agent along with the SDP offer we can build a CALL ID SERVICE. SIP is a simple signaling protocol that allows us to use other protocols to build a conference management service using its primitives [14].

2.6. Functions of SIP

SIP provides methods to start, modify, and end a session. It serves four major purposes:

1. Finding location of the user by mapping user SIP address to its relevant IP address at current location.
2. SIP negotiations among user agents in order to agree on type of codec and protocol.
3. A user agent can join, leave, and pause a session. A flexible control on calls by SIP protocol. A buddy list utilizes caller to choose among many SIP users in order to start a call with a friend.
4. It is possible for a caller to change the codec within a session in order to improve QoS [14].

Please notice that SIP is a simple session control protocol that needs to work with other protocols such as RTP, RTCP, SDP, and iRTP (in the thesis work) to do conference management, QoS, and improve multimedia streaming [14].

SIP creates a framework for cooperation of different protocols to create a comprehensive, intelligent, and high quality communication system [14].

2.7. Components of SIP

User Agents are the participants in a SIP session which join a collective conference to stream real time multimedia data. User Agent (UA) can take two roles in a SIP session User Agent Client (UAC) which means it creates requests and sends it off to servers over internet. User Server (UAS) receives requests, translates them into proper responses and creates the response messages and send them back to the client [14].

Please notice that a user agent can take on role of both: a client and a server, yet there are occasions that it may strictly act as a server. For instance, an external Hard disk needs to upload some info over IP telephony to another client. Downloading music streams over internet could be a good example of it. As a client we send request in order to call a person in buddy list which can be
accepted or denied by the callee. A music web server can accept or deny a call request from another user agent [14].

2.7.1. Clients:

Client is an endpoint (i.e. a windows application). It may be an instance of XLITE or a messaging device. It creates and sends a SIP request when one tries to place a call to another user agent over internet which consequently a SIP message request is sent to a server (proxy server). A server may respond to the request with a confirmation acknowledgement which results in a continuation of a phone call. Since SIP is concerned with sip address rather than physical address it is irrelevant where client is at the time being. If client is busy he can put on hold the new call or forward it to someone else [14].

2.7.2. Servers:

A Server has predefined set of tasks in order to handle a call or SIP Request. There are several varieties of servers that have vital duties to accomplish. A server failure results in collapse of communication system. If registrar server encounters a failure then no one can sign in to the SIP server; they will not be accessible over internet by other SIP user agents [14].

Proxy Server: this is a vital server in the SIP environment. When a request is created, IP address of the recipient is not known; therefore, client sends the request to a proxy server. It forwards the request to another proxy server until callee is found [14].

Redirect Server: When the redirect server redirects the request back to the client, it means that the client ought to modify the path to get hold of the callee. It is basically the case when a callee has moved from its original position either temporarily or permanently [14].

Registrar: Registrar server detects the location of a user in a network. Thus, a user ought to register his locations to a Registrar server before he is reachable. It is vital that a user occasionally send re-registration of their locations to a Registrar server [14].

Location Server: The addresses registered to a Registrar are stored in a Location Server [14].

2.8. Commands of SIP

1. INVITE: Invites a user agent to accept or deny a call placed by a caller.

2. ACK: Acknowledgement is used to facilitate reliable message exchange for INVITEs.

3. BYE: Terminates a sip call by one of the user agents in the session.

4. CANCEL: Terminates a request, or search, for a user. It is used if a client sends an INVITE messages and then client decides to abort the call to callee before the dialog is established.

5. OPTIONS: informs user agents of another user agent or a server's capabilities.

6. REGISTER: Registers the user's current location
7. **INFO:** Used for mid-session signaling

8. **PRACK:** Provisional **R**esponse **A**cknowledgement adds reliability for provisional response for SIP/PSTN calls. It is vital that 183 and 180 response codes are not lost on the way from PSTN to SIP. Callie keeps sending Response code 183 until it receives PRACK message then callee stops sending Response. This increases reliability of SIP session management and signaling system in interaction with PSTN (non-mobile-phones).

9. **UPDATE:** **U**pdate refreshes the media session and codec parameters before a Dialog is established. It is similar to the re-invite method except it is not called in middle of SIP session when media session of Dialog is already established. We will discuss the format of some of the above SIP commands in more detail shortly [14].

2.9. A Typical Example of SIP Session

It is shown in Figure 6 a simple SIP response message. SIP proxy server finds the location of the end point, places a Request for a connection, negotiates the terms and parameters of a multimedia session, stops the session and destroys all the resources used by the session [14].

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP server10.biloxi.com;branch=z9Hg4BkNASHDS8;RECEIVED=192.0.2.3
Via:SIP/2.0/UDP bigbox3.site.atlanta.com;branch=z9hG4bK77ef4c2312983.1;received=192.0.2.2
Via:SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK77ef4c2312983.1;received=192.0.2.2
To:user2@user1@biloxi.com;tag=a6c85cf
From:user1@atlanta.com;tag=1928301774
Call-ID:a84b4c76e66710@pc33.atlanta.com
Cseq:314159 INVITE
Contact:sip:user2@192.0.2.4
Content_Type:application/sdp
Content-Length:131
```

**Figure 6:** Example of a SIP Response message sent by user2 to user1 for a Request sent by user1 to make a phone call to user2 [1].

It is shown in user2 sends back a Response message to user1 to confirm the call. SDP body is not shown here, but it contains all the necessary parameters such as codec and protocol type. There are three via headers are added by proxy server2, proxy server1, and user2. The proxy servers are stateful otherwise there would not exist no via headers would be added. Stateless proxy servers only pass on SIP messages to the next proxy in the ROUTE header [1].

SIP signaling is a proxy server-client event driven environment. Figure 7 shows a full cycle of SIP negotiation [1].

We firstly introduce trapezoid diagram and describe how SIP methods are called. User1’s softphone calls the SIP phone of user2. Server1 and server2 help to establish a session by taking on a proxy role for two users. The cooperation of the proxies, User1, and User2 is called "SIP Trapezoid" as shown by the dotted line in Figure 7. The messages appear vertically in the order they appear i.e. the M1 label next to INVITE on top means that (INVITE M1) is first request message that is sent. The direction of arrows shows the sender and recipient of each message. Each message contains a 3-digit-number followed by a name and each one is tagged by 'M' and a serial number. The 3-digit-number
is the numerical code of the associated message comprehended easily by machines. Human users use the name to identify the message [1].

User1 creates and sends an INVITE request to user2; user1 doesn't know where user2 is physically located over network so that the IP address of user2 is not known. Thus, user1 sends the request to server1. Server1 forwards an INVITE request to server2 for user2. Server2 sends a (100 Trying) response back to user1 informing that it is trying to get hold of user2. Please refer to Figure 25 in iRTP Structure. When (INVITE M2) from server1 arrives to server2, it works in a similar fashion as server1. It forwards an INVITE request to user2. Firstly (INVITE M3) is forwarded and then server2 issues a (100 Trying) response to server1. Server2 knows the location of user2, but if Server2 may not know that, therefore, INVITE request may travel through several proxies before arrival to the recipient. When the INVITE request gets to the user2’s softphone, it will ring user2. It sends a (180 Ringing) response to server2 which reaches user1 via server1. User1 receives a (180 Ringing M8) that means user2 has received the INVITE request. User2 at this point has a choice to accept or decline the call. When User1 accepts it then a 200 OK response is sent by the softphone to server2. When User1 receives it, the softphone of user1 sends an ACK message to confirm the end of SIP negotiation successfully. This handshaking (INVITE+OK+ACK) is necessary to establish a call [1].

The RTP connection is established between User1 and User2. When one party wants to end the call, it sends a BYE message to the other user agent. Recipient user agent sends a 200 OK message to confirm the end of the session [1].

When a caller calls a callee, a series of dialogs will occur. Several transactions are in this phenomenon. Figure 8 elaborates about it [1].
A caller may have connections to a number of callees at a time forming a number of dialogs. All these dialogs make a single call. It is shown in Figure 8 that a Dialog contains two transactions. Thus, SIP signaling control different aspects of a one or more dialogs. Each transaction is intended to perform a certain controlling task such as initiate or terminate a call [1].

2.10. Structure of SIP Protocol

Figure 9 shows structure of SIP protocol which is layered and organized logically, yet physically the SIP components are implemented and manufactured in different places. Transaction is a request sent to the server and it receives a response from server to confirm or deny it. Response could be 1xx and 2xx to accept it [13].

Figure 9 shows that how User Agent makes a call which initiates a SDP offer to transaction user layer. User layer creates an INVITE request and creates a client transaction layer and passes INVITE request to it along with IP address, port number and transport. The state of transaction is calling and we have transaction tagged by two parameters CSeq and a branch parameter. In transport layer we send the Request to Proxy server, but we need to add via header to it and place the send-by parameter in it so that proxy server knows where to return the response next time it sees the response. UDP layer is responsible to transmit this message to proxy server [13].

Proxy server receives the Request and extracts the sent-by from top via header and corresponds to the server desired transaction then it generates a Receive parameter. Proxy core creates an INVITE transaction server to process the Request and pass it on to the TU layer of proxy1. There is a timer that will determine a response will be ready within a limited time couple hundreds of milliseconds. This response is not (100 Trying) [13].

Proxy1 realizes the authentication data is missing; therefore, it sends back a 407 Response that is (Proxy Authentication Required). This is a typical of the Structure of SIP protocol and it is quite
interesting to see how a switching protocol can be comprehensive enough to make SIP the most intelligent protocol of all [13].

Figure 9: Typical Structure of the SIP protocol [13]
2.11. Registration in SIP

In a typical SIP Session caller doesn't know the address of the callee initially. The location of the callee is discovered by the proxy servers. Every user registers its present location to a server called SIP REGISTRAR. Application layer sends a REGISTER message which posts its current location to the location server. It stores the information in a table; therefore it maps the user’s logical address to his current IP address. Hence, Location service helps the proxy servers to locate the user [13].

It is shown in Figure 10 caller’s current IP address (195.31.65.152.). As soon as user agent signs up it gets registered with location service. One of SIP address advantages is that guarantees user mobility, therefore one can log in from different locations with same logical address and one becomes accessible on the network. After log in, the application REGISTERS the user name with the IP of a given computer. User agent ought to refresh its registration from time to time [12].

Figure 10 illustrates both: SIP signaling and RTP protocols cooperate in order to stream a phone call, a SMS, and a MMS between two user agents in detail [12].

DNS Server using special algorithms queries DNS for IP address of a next proxy hop, client, and backup client. DNS server resolves URI to IP address, port number, and transport control of next hop via which to get connected to desired user agents proxy server [12].

Location Service: this is a data-bank of user agents that maps URI to IP address. It queries globally IP address of a user agent [12].

SIP Proxy: is a middle ware which does most of the work of SIP session between two SIP clients. It takes Requests and returns final Response, or provisional Response. There are two kinds of Proxy servers: Stateful and stateless. Stateful proxy server stores the state of a transaction from INVITE until BYE (is called to stop the media session) method; stateless proxy server only passes the Request message to next proxy hop without storing its state [12].

SIP Registrar: registers a client to the Location service when a SIP client signs up. There is a databank inside which stores IP address of registered user agents [12].

PSTN gateway: it connects a SIP system to a legacy PSTN telecommunication system. SIP phones use this gateway to call other ordinary phones, or SIP phones [12].

Notice that SIP can adapt itself to different applications and communication systems with new software and hardware that are produced with vendors [12].
Figure 10: A typical SIP Messaging followed by a RTP multimedia streaming [12]
Figure 11: SIP messaging for only one agent signed up [12]

Table 1: User agent (ANDY) is trying to establish a session with another user agent (DAVID) who is not signed in. [12]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INVITE</td>
</tr>
<tr>
<td>2</td>
<td>TO: sip:<a href="mailto:DAVID@iptel.org">DAVID@iptel.org</a></td>
</tr>
<tr>
<td>3</td>
<td>100 Trying</td>
</tr>
<tr>
<td>4</td>
<td>DNS query: iptel.org</td>
</tr>
<tr>
<td>5</td>
<td>Response: 192.0.2.4</td>
</tr>
<tr>
<td>6</td>
<td>INVITE</td>
</tr>
<tr>
<td>7</td>
<td>TO: sip: <a href="mailto:DAVID@iptel.org">DAVID@iptel.org</a></td>
</tr>
<tr>
<td>8</td>
<td>100 Trying</td>
</tr>
<tr>
<td>9</td>
<td>LS query: sip:<a href="mailto:DAVID@iptel.org">DAVID@iptel.org</a></td>
</tr>
<tr>
<td>10</td>
<td>DAVID is not signed in</td>
</tr>
<tr>
<td>11</td>
<td>480 Temporarily not available</td>
</tr>
<tr>
<td>12</td>
<td>ACK</td>
</tr>
</tbody>
</table>
Figure 12: ANDY sends SUBSCRIBE message to DAVID, to get notifications when he is available [12]

Table 2: DNS Server is not necessary, since ANDY’s proxy server is aware of IP of DAVID’s proxy. After ANDY acknowledged DAVID is not signed in he sends a SUBSCRIBE request, therefore, as soon as DAVID signs up; ANDY receives a notification [12].

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>SUBSCRIBE</strong>&lt;br&gt;TO: sip:<a href="mailto:DAVID@iptel.org">DAVID@iptel.org</a></td>
</tr>
<tr>
<td>2</td>
<td><strong>SUBSCRIBE</strong>&lt;br&gt;TO: sip:<a href="mailto:DAVID@iptel.org">DAVID@iptel.org</a></td>
</tr>
<tr>
<td>3</td>
<td><strong>SUBSCRIBE</strong>&lt;br&gt;TO: sip:<a href="mailto:DAVID@iptel.org">DAVID@iptel.org</a></td>
</tr>
<tr>
<td>4</td>
<td>200 OK</td>
</tr>
<tr>
<td>5</td>
<td>200 OK</td>
</tr>
<tr>
<td>6</td>
<td>200 OK</td>
</tr>
<tr>
<td>7</td>
<td>NOTIFY &lt;not signed in&gt;</td>
</tr>
<tr>
<td>8</td>
<td>NOTIFY &lt;not signed in&gt;</td>
</tr>
<tr>
<td>9</td>
<td>NOTIFY &lt;not signed in&gt;</td>
</tr>
<tr>
<td>10</td>
<td>200 OK</td>
</tr>
<tr>
<td>11</td>
<td>200 OK</td>
</tr>
<tr>
<td>12</td>
<td>200 OK</td>
</tr>
</tbody>
</table>
Figure 13: Ex-Not-signed-in user agent eventually is signed up and available [12]

Table 3: DAVID eventually is signed up and ANDY is notified that DAVID is available [12].

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 1 | **INVITE**  
   | TO: sip:DAVID@iptel.org |
| 2 | 100 Trying  |
| 3 | DNS query: iptel.org  |
| 4 | Response: 192.0.2.4  |
| 5 | **INVITE**  
   | TO: sip: DAVID@iptel.org |
| 6 | 100 Trying  |
| 7 | LS query: sip:DAVID@iptel.org  |
| 8 | DAVID is not signed in  |
| 9 | 480 Temporarily not available  |
| 10 | ACK  |
| 11 | 480 Temporarily not available  |
| 12 | ACK  |
Figure 14: ANDY calls DAVID when being acknowledged DAVID is available [12]

Table 4: User agent (ANDY) can try to establish a SIP session again. Since ANDY proxy server has fetched DAVID’s IP address, there is no need to DNS consultation. RTP Multimedia session is established [12].

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INVITE</td>
</tr>
<tr>
<td></td>
<td>TO: sip:<a href="mailto:DAVID@iptel.org">DAVID@iptel.org</a></td>
</tr>
<tr>
<td>2</td>
<td>100 Trying</td>
</tr>
<tr>
<td>3</td>
<td>INVITE</td>
</tr>
<tr>
<td></td>
<td>TO: sip:<a href="mailto:DAVID@iptel.org">DAVID@iptel.org</a></td>
</tr>
<tr>
<td>4</td>
<td>100 Trying</td>
</tr>
<tr>
<td>5</td>
<td>LS query: sip:<a href="mailto:DAVID@iptel.org">DAVID@iptel.org</a></td>
</tr>
<tr>
<td>6</td>
<td>Response: sip:<a href="mailto:DAVID@iptel.org">DAVID@iptel.org</a></td>
</tr>
<tr>
<td>7</td>
<td>INVITE</td>
</tr>
<tr>
<td></td>
<td>TO: sip:<a href="mailto:DAVID@iptel.org">DAVID@iptel.org</a></td>
</tr>
<tr>
<td>8</td>
<td>180 Ringing</td>
</tr>
<tr>
<td>9</td>
<td>180 Ringing</td>
</tr>
<tr>
<td>10</td>
<td>180 Ringing</td>
</tr>
<tr>
<td>11</td>
<td>200 OK</td>
</tr>
<tr>
<td>12</td>
<td>200 OK</td>
</tr>
<tr>
<td>13</td>
<td>200 OK</td>
</tr>
<tr>
<td>14</td>
<td>ACK</td>
</tr>
</tbody>
</table>
2.12. PJSIP Implementation

Following sections elaborate about the PJSIP library.

2.12.1. PJSIP: SIP & Multimedia Communication Library

PJSIP is a free and open source multimedia communication library written in C language implementing standard based protocols such as SIP, SDP, RTP, STUN, TURN, and ICE. It combines signaling protocol (SIP) with rich multimedia framework and NAT traversal functionality into high level API that is portable and suitable for almost any type of language and platform: desktops, embedded systems, and mobile handsets [8].

2.12.2. Features

PJSIP is both; compact and feature rich. It supports audio, video, presence, and instant messaging; it is extensive documentation is impressive. PJSIP is very portable. On mobile devices, it abstracts system dependent features; therefore, it is able to utilize the native multimedia capabilities of the device in most cases [8].

2.12.3. Architecture

![Collaboration Diagram](image-url)

Figure 15: Collaboration Diagram [8]
**Figure 15** illustrates the diagram which elaborates how SIP message is passed among PJSIP components.

PJSIP architecture implements the SIP stack. The core is endpoint.c which performs quite a lot of tasks; it owns pool factory (allocates memory to different SIP units), an instance of PJLIB ioqueue belongs to end point which dispatches network events, contains an instance of Transport manager which has transport units that controls messages and parse them, manages modules which perform tasks beyond message parsing and transport, and receives incoming messages from transport manager in order to distribute them to modules [8].

Transaction is the request placed by a client to a server. As soon as a request is received by server; it sends back a (180 trying) back to the client in order to inform the client that it is being processed. Transaction is created by placing an SDP Offer by the client. A dialog consists of one or more transaction [8].

Dialog is a call placed by a caller to a callee. A dialog is finished with a bye message from caller to callee or vice versa [8].

Transport manager creates transport sockets of type UDP or TCP so that they connect to IP layer to send/receive SIP or RTP packets. A hash table can keep track of the transport channels. Hence, when they are not used anymore they will be destroyed [8].

Parser is a unit to parse messages and URIs in order to extract useful data for further processing. It is extensible to, a plug-in parser, if a certain version of parsing is needed [8].

Callback functions are the client created functions that are called by the server in order to process part of a dialog and transaction. They are usually called by server in case of an event occurrence automatically to handle the event for the dialog [8].

UA (user agent) module is primarily called when application module wants to place a SIP call to another user agent. It creates a client or server module to send an INVITE request, or a corresponding response to an INVITE message, by a server in order to further process it which leads to a SIP connection establishment [8].

Dialog Hash tables and Transaction Hash table respectively store list of Dialog and Transactions that are active at the time being. Hence, they are easily accessible to the application when they are needed [8].

It is shown in **Figure 15** the path which message travels in PJSIP architecture. It represents the SIP message flow clearly. It is the SIP structure that ensures the message flow. PJSIP is created a SIP stack, that is, every SIP component is placed in a logical layer of ISO model [8].

**2.12.4. The Endpoint**

At the heart of the SIP stack is the SIP endpoint, which is represented with opaque type pjsip_endpoint. The endpoint has the following properties and responsibilities:

- Pool factory which allocates pools for all SIP subunits.
- Timer manager which schedules timers for all SIP components.
- It has the one transport manager. The transport manager has SIP transports and supervises parse and print of the message to a special message buffer.
- It owns a single instance of PJLIB’s ioqueue. The ioqueue acts as a proxy to dispatch network events.
- Provides a thread safe polling function, therefore, applications’ threads can poll for timer and socket events (PJSIP does not create any threads by itself).
- It manages PJSIP modules. Pjsip module is the primary means for extending the stack beyond message parsing and transport.
- It receives incoming SIP messages from transport manager and distributes them to modules [8].

2.12.5. Module

Module framework is responsible for distributing SIP messages among software components in PJSIP application. All software components in PJSIP, including the transaction layer and dialog layer, are implemented as module. Without modules, the core stack (pjsip_endpoint and transport) simply wouldn’t be able to handle SIP messages [8].

The module framework is built based on a simple, yet powerful interface abstraction. For incoming messages, the endpoint (pjsip_endpoint) distributes the message to all modules starting from module with highest priority, until one of modules process it. For outgoing messages, the endpoint distributes the outgoing messages before they are transmitted to the wire, therefore, to allow modules to do last modification on the message if they wish [8].

When a message arrives it will be put into a buffer pjsip_rx_data and then parsed by transport manager and saved. This message will be passed on to the endpoint. Endpoint distributes it to different modules by calling on_rx_request (callback function), or on_rx_response (callback function). When one of registered modules handles the message, it returns with PJ_SUCCESS, and endpoint will stop distributing this message to any other module [8].

Registered modules assigned different priority. The module with highest priority gets to process the incoming message. Module priority can be modified. We have a global timer manager that creates events to interrupt incoming messages; therefore application does not wait for incoming messages forever [8].

2.12.6. Module Capabilities

Module may modify its capabilities to the endpoint. Currently the endpoint manages these capabilities:

- Allowed SIP methods(ALLOW header field),
- Supported SIP extension (supported header field).
- Supported content type (Accept header field).

These header fields will be added to outgoing requests, or responses, where appropriate automatically.

A module declares new capability by calling pjsip_endpt_add_capability () function [8].
Complete form of function: `Pj_status_t pjsip_endpt_add_capability (pjsip_endpoint *endpt, Pjsip_module *mod, int htype, const pj_str_t *hname, unsigned count, const pj_str_t *tags[])` [8];

We should register new capabilities to the endpoint. The type argument above specifies to which header to add the capabilities; there are following capabilities: PJSIP_H_ACCEPT, PJSIP_H_ALLOW, and PJSIP_H_SUPPORTED. The `hname` argument is optional; it is only used to specify capabilities in header fields that are not recognized by the core stack. Thus, `count` specifies number of capabilities and `tags[]` argument specifies an array of capability names to be added to the header field [8].

Function, `Const pj_str* pjsip_endpt_get_capability (pjsip_endpoint *endpt, int htype, const pj_str_t *hname)` [8], gets capability header field which contains list of capabilities that have been registered to the endpoint for the specified header field [8].

2.3.2. Message Elements

2.3.2.1. Universal Resource Indicator

It is an identifier just like a name in order to assign it to a message. `Pjsip_uri` is a general parent structure for all other types of URIs such as `pjsip_sip_uri`; therefore it can be casted to `pjsip_uri`. It is shown in Figure 16. `Pjsip_uri` is the highest level parent [8].

```
<table>
<thead>
<tr>
<th>Pjsip_uri</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes:</td>
</tr>
<tr>
<td>Operations:</td>
</tr>
<tr>
<td>const pj_str_t pjsip_get_uri_scheme(uri);</td>
</tr>
<tr>
<td>pjsip_uri * pjsip_uri_get_uri(uri);</td>
</tr>
<tr>
<td>pj_status_t pjsip_cmp_uri(uri1, uri2);</td>
</tr>
<tr>
<td>int pjsip_uri_print(context, uri, buf, maxlen);</td>
</tr>
<tr>
<td>pjsip_uri * pjsip_uri_clone(uri);</td>
</tr>
</tbody>
</table>
```

```
| pjsip_sip_uri |
| Attributes: |
| Operations: |
| void pjsip_sip_uri_init(pool, secure); |
| void pjsip_sip_uri_assign(pool, dst_uri_src_uri); |
```

```
| pjsip_uri |
| Attributes: |
| Operations: |
| const pj_str_t pjsip_get_uri_scheme(uri); |
| pjsip_uri * pjsip_uri_get_uri(uri); |
| pj_status_t pjsip_cmp_uri(uri1, uri2); |
| int pjsip_uri_print(context, uri, buf, maxlen); |
| pjsip_uri * pjsip_uri_clone(uri); |
```

```
| pjsip_name_addr |
| Attributes: |
| Operations: |
| Pjsip_name_addr * pjsip_name_addr_create(); |
```

New types of uri can be defined

```
Pjsip_uri
| Attributes: |
| Operations: |
| const pj_str_t pjsip_get_uri_scheme(uri); |
| pjsip_uri * pjsip_uri_get_uri(uri); |
| pj_status_t pjsip_cmp_uri(uri1, uri2); |
| int pjsip_uri_print(context, uri, buf, maxlen); |
| pjsip_uri * pjsip_uri_clone(uri); |
```

```
Pjsip_sip_uri
| Attributes: |
| Operations: |
| pjsip_sip_uri * pjsip_sip_uri_create(pool, secure); |
| void pjsip_sip_uri_init(uri, secure); |
| void pjsip_sip_uri_assign(pool, dst_uri_src_uri); |
```

```
Pjsip_uri
| Attributes: |
| Operations: |
| const pj_str_t pjsip_get_uri_scheme(uri); |
| pjsip_uri * pjsip_uri_get_uri(uri); |
| pj_status_t pjsip_cmp_uri(uri1, uri2); |
| int pjsip_uri_print(context, uri, buf, maxlen); |
| pjsip_uri * pjsip_uri_clone(uri); |
```

```
Pjsip_name_addr
| Attributes: |
| Operations: |
| Pjsip_name_addr * pjsip_name_addr_create(); |
```

Figure 16: URI class diagram [8]

Pjsip_uri implements SIP and SIP URIs. It is part of Request line, From/To header, contact header, and Route/Record-Route header. SIP message contains URIs and they are vital part of the message. Each message header is stored in proxy servers and client/server in order to know how to direct the request and response back to the client from server through stateful proxy servers [8].
2.3.3. SIP Message in PJSIP

PJSIP complies with SIP protocol message format precisely. Thus it is really extensive for an overview. There are library virtual functions that can be used to access any part of the SIP message from header details to message body. Each part of the program has capability to manipulate the message before sent out the message to other Endpoints over network. Please refer to PJSIP documentation [8].

![pjsip header class diagram](image)

Figure 17: pjsip header class diagram [8]
Figure 17 shows the snippet of PJSIP header class diagram. There are more headers than the ones displayed in the diagram; PJSIP library implements all headers that are specified in the core SIP specification (RFC 3261). Other headers will be implemented in the corresponding PJSIP extension module [8]. In the following sections we elaborate about the header fields.

### 2.3.3.1. Header Fields

All header fields in PJSIP share common header properties such as header type, name, shortname, and virtual function table. Therefore, all header fields can be treated uniformly by the stack. Please refer to Figure 17 [8]. For instance, pjsip_via_hdr header contains address of proxy server which helps to locate the callee over internet. Pjsip_to_hdr is the callee’s logical address (e.g. sohrab@iptel.org). Thus, it is so forward to extract these headers in order to process the Request and create a suitable response to send back to the caller.

### 2.3.3.2. SIP Status Codes

In Figure 18 shows the declaration of SIP status code for PJSIP. There is only part of status codes in this snippet [8].

```c
enum pjsip_status_code
{
    PJSIP_SC_TRYING_100,
    PJSIP_SC_RINGING_180,
    PJSIP_SC_CALL_BEING_FORWARDED_181,
    PJSIP_SC_QUEUE_182,
    PJSIP_SC_PROGRESS_183,
    PJSIP_SC_OK_200,
    PJSIP_SC_MULTIPLE_CHOICES_300,
    PJSIP_SC_MOVED_PERMANENTLY_301,
    PJSIP_SC_MOVED_TEMPORARILY_302,
    PJSIP_SC_USE_PROXY_305,
    PJSIP_SC_ALTERNATIVE_SERVICE_380,
    PJSIP_SC_BAD_REQUEST_400,
    PJSIP_SC_UNAUTHORIZED_401,
};
```

Figure 18: Code 21 SIP status code constants [8]

SIP negotiation among SIP user agents starts per a Request message from client to server as shown in Figure 8; if it results in success RTP streaming of multimedia continues afterwards [8]; otherwise the SIP servers take necessary action to gracefully handle the situation. They may issue a failure error, or ask for more information to be able to continue the process of the request message.

### 2.3.4. Sending Messages
The core task of SIP application is send/receive SIP messages. Receive of Request message is handled in on_rx_request() and response is handled by on_rx_response() callbacks; module contains the callbacks as part of its structure [8].

PJSIP provides rich API plus a transmit buffer to generate request or response messages. There are various ways to create messages. PJSIP treats non INVITE requests as stateless and responds to them with response code 500. (Please refer to pjsip documentation) [8].

Sending response involves creation of a standard response message for the request with status code and status text. If status text is NULL default status text will be used [8].

Stateless proxy forwarding is a subject in RFC 3261. Proxy may choose to forward a stateless request (stateless means not processed). This could happen because of non-authorized user trying to interfere with system, servers, proxies, or any other problems (Please refer to examples of 7.3) [8].

2.3.5. Transactions

Transaction is created in PJSIP by a call to pjsip_tsx_endpt_create_uac, or pjsip_tsx_create_uas functions. It is a vital concept in pjsip core; we need to elaborate about this important part of pjsip library [8].

- Client/server transaction is created by pjsip_tsx_endpt_create_uac / pjsip_tsx_create_uas.
- After initializing UAS transaction, application needs to call pjsip_tsx_recv_msg() to pass on the initial request message. Transaction state can move from NULL to Trying; therefore, subsequent request retransmissions will be absorbed by the transaction.
- When application wants to send request or response message using the transaction, it will call pjsip_tsx_send_msg.
- Transaction’s state changes as messages are passed on, either by endpoint for incoming message, or transaction user for outgoing message, or timer elapses, and transaction user are notified via on_tsx_state callback.
- Transaction will be destroyed as soon as its state has reached to PJSIP_TSX_STATE_TERMINATED. Application can also preemptively terminate the transaction by calling pjsip_tsx_terminate [8].

2.3.6. Timers and Retransmission

Transaction only has two types of timers: retransmission timer and timeout timer. The value of both timer types are automatically set by the transaction according to the transaction type (UAS or UAC), transport (reliable or no-reliable), and method (INVITE or non-INVITE) [8].

Pjsip is a huge library of code that needs to be read from documentation and it is beyond this thesis work. For more detailed information about how SIP works please refer to RFC 3621 [8].

2.4. RTP Header

<table>
<thead>
<tr>
<th>UDP Header</th>
<th>RTP Header</th>
<th>RTP Payload</th>
<th>Padding</th>
<th>Pad Count</th>
</tr>
</thead>
</table>
Each RTP packet is encapsulated inside a UDP datagram and not in a TCP packet, because TCP does not support multicast. Its retransmission of a lot of packets because of few lost packets would introduce a big undesirable latency in traveling packets [12].

Optimization between packetization delay and overhead of network traffic is the criteria to choose the size of payload in a RTP packet. Packet size affects traffic congestion and slowing down the movement of RTP packets on networks. We chose 20 milliseconds of voice before codec encryption is an industry standard [12]. Padding is needed to satisfy some codec algorithms which require fixed size blocks.

We need a small RTP header to have less overhead. It has variable size and it is based on what kind of application sends RTP packet [12].

2.5. RTP

RTP is a real-time multimedia protocol. Thus, production, transmission and play of data are simultaneously. RTP protocol provides a special buffer, called jitter buffer, in order to make sure that playing of the data on receiver side is not affected by varying of latency in arriving packets. Congested line can cause voice choppy [12].

We can stream already prepared DVD videos over internet which is only Multimedia traffic. This can be done using RTP or RSTP. RSTP is secure RTP which does not allow hacking of the packets to steal data over networks [12].

There are also real-time data which is not multimedia, for example one can have bunch of files which are piping data into an application and one can take them and send them as voice over internet. For example text is one of many applications. ToIP (Text over IP) is conversion of text to voice before it can be packetized with voice and video in an iRTP packet. Finally packet is sent out to clients or other servers over internet [12].

Radio, IP phone, SIP phone, and Video conference are examples of live A/V streaming over internet. Digital communication with high quality has a better performance than analog, because analog system is concerned with signal-to-noise ratio which determines the quality of signal. We sample the voice and video to convert it into digital [12].
Time delay shows that when a packet of size 100 milliseconds travels over internet will arrive to destination computer with 1 second delay and this delay is constant in consequent packet afterwards. Thus, there is no varying delay in arriving packets. **Figure 19** shows that all three packets (FIRST PACKET, SECOND PACKET, AND THIRD PACKET) arrived with 1 second delay at different times; therefore, we don’t have jitter, yet we have time delay introduced by the network [12].

**Figure 20: Jitter in Packet arrival [12]**
Jitter means variable time gap among consequent packets arrival time destination computer. Thus, **Figure 20** illustrates client computer’s first and second packets with 5 seconds delay while between second and third packets we have only 2 seconds delay. Server has a clock that is synchronized with client clock. Time stamps of the packets are fetched from clock and when the delay of the network is included into the arrival time one can see how jitter really happens. Packets are played with different arrival times than real time stamps [12].

Jitter is removed with addition of jitter buffer which buffers incoming packets and then plays them using packet timestamp [12].

The following is a sample of SDP message that is sent to another SIP phone and it has codec information in it [12].

**Figure 21** shows a sample SDP message for a SIP phone which attempts to negotiate a SIP connection with another SIP phone for an audio conference. This is sort of introduction of two strangers when they encounter one another for first time. They should be interested to communicate with each other and they should use the same language or protocol. This is exactly the same concept [12].

(o) = it is the original of the signal being sent from sender SIP phone. It is input with IP address with IPV4 format from IP address 192.168.0.100

(m) = it stands for media specifications of media session. It is streamed on port 4000 over RTP protocol. Audio and video is being streamed.

(a) It stands for attribute of SDP offer. It expresses the payload type could be PCMU or PCMA with clock rate (sampling frequency) of 8000. A rtcp channel on port 4001 which is input with IP address of IPV4 format.

```
1 v=0
2 o=- 3594877494 3594877494 IN IP4 192.168.0.100
3 s= pjmedia
4 t=0 0
5 m=audio 4000 RTP/AVP 0 8 96
6 c=IN IP4 192.168.0.100
7 b=TIAS:64000
8 a=rtcp:4001 IN IP4 192.168.0.100
9 a=sendrecv
10 a=rtpmap:0 PCMU/8000
12 a=rtpmap:8 PCMA/8000
```

**Figure 21**: SDP format for an audio conference

### 2.6. RTP/RTCP Features Description and UDP vs. TCP

When we use *timestamps* in a RTP session with UDP protocol, jitter is canceled out from our conversations. It improves the playback of received packets. Thus, jitter is removed which leads to have real timing for each received packets. *Sequence number* ensures that all the packets get correct
order at the destination. The presence of a user agent is notified by sequence number and if there is a lot of pause then he may be forced to end his session. We need to multicast packets to all of the participants in a RTP session. RTCP/RTP protocols cooperate to control the bandwidth used in a media session that is when traffic increases we can use RTCP to allow the user agents to change their codec to get a better bandwidth which leads to improved performance of the media session. Codec ensures that participants in a RTP session get the best of the available bandwidth. Multimedia data could be mixed in a mixer and sent off over internet. This is a quite complicated process because of the many details involved [12].

TCP is not a good choice because of the retransmission of all the packets, because of few lost packets, which creates quite large latency. Thus, voice and video quality would not be good furthermore efficiency is poor. It is not possible to have a media session with multiple participants because TCP does not support multicast. We have lager header in TCP and it is waste of bandwidth [12].

In conclusion UDP is a good choice rather than TCP for real-time/non-real-time multimedia applications [12].

2.7. RTP Timestamp and Sequence number

As we know Timestamp of a RTP packet is calculated based on clock rate or sampling rate of audio signal into digital values. There is a simple formula to calculate the timestamp of each audio packet created in sender side as we will mention below.

Clock rate of timestamp is 8000 hertz. This is equal to 128 microseconds. 20 milliseconds of audio digital sample is carried by one RTP packet. 160 is the timestamp increase for each RTP packet and it calculated in following way: \( 20000/128 \) microseconds = 160. This value does not change even if we send silence packet. Number of packets sent each second is \( 1/20 \) milliseconds = 50. Number of bits for this payload is \( 160 \times 16 = 2560 \) bits. This value is for uncompressed payload but if one uses G711 this value would be 1280-bits. Sequence number is usually incremented by one for each new RTP packet [12].

```
<table>
<thead>
<tr>
<th>Packet (n-1)</th>
<th>Timestamp = x</th>
<th>Sequence Number = y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet (n)</td>
<td>Timestamp = x+160</td>
<td>Sequence Number = y+1</td>
</tr>
</tbody>
</table>
```

Figure 22: RTP packet sequence showing timestamp and sequence number [12]

Figure 22 shows the concepts explained above. Notice the sequence number and timestamp increases [12].

RTP is a protocol to send real-time voice, video, and data. It has many applications in business and industry [12].
2.8. RTCP: Real-Time Control Protocol

RTP is a protocol that allows real time stream in the basic transport layer, but does not provide any mechanism for error and flow control, congestion control, quality feedback and synchronization. For that purpose the RTCP protocol helps in order to RTP to provide end-to-end monitoring, data delivery, and traffic congestion control [12].

2.8.1. Tasks of RTCP

- It checks the performance of the sender and network.
- It shows the identity of the sender [12].

2.8.2. RTCP Packets structure

<table>
<thead>
<tr>
<th>UDP Header</th>
<th>RTCP Header</th>
<th>RTCP Data</th>
</tr>
</thead>
</table>

Port number for RTCP = Port number for RTP + 1

It is always a pair of ports for RTP and RTCP protocols. Thus, if you want to include 5 types of multimedia data in one application you would need 5 pairs of ports as mentioned above or total 10 ports. It is customary to include several RTCP packets in on UDP packet to reduce the overhead [12].

In a session with many participants even though one party talks and the rest of participants are silent many RTCP packets arrives to participants to reduce the negative effect of RTCP packet we need to keep the bandwidth of RTCP only 20 percent of total bandwidth that means every four RTP packet sent we only send one RTCP packet. Each participant should only take 5 percent of RTCP bandwidth [12].

2.8.3. RTCP Header structure

<table>
<thead>
<tr>
<th>Version</th>
<th>P</th>
<th>RR Count</th>
<th>Packet Type</th>
<th>Message Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 bits</td>
<td>1bit</td>
<td>5bits</td>
<td>8bits</td>
<td>16bits</td>
</tr>
</tbody>
</table>

Version: 2; P: Padding bit; RR Count: Reception reports blocks count; Packet Type: Number in range of (192-207) [12].
### 2.8.4. SR Report Format

<table>
<thead>
<tr>
<th>Version</th>
<th>P</th>
<th>RR Count</th>
<th>PT = 200</th>
<th>Message Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SSRC for Sender</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- NTP Timestamp: absolute time that report is sent</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- RTP Timestamp: relative time calculated from sampling clock rate</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Number of RTP packets sent</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- SSRC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Number of packets lost since last report</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Jitter</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Timestamp of the last SR received</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Delay status since last SR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NTP Timestamp is used to synchronize the clocks of the computers involved in the chat room. We have used pool.ntp.org as reference clock please refer to chapter 5 for more detailed information. The blue section above is used to synchronize audio and video signal and this is very important in multimedia streaming. The green section is reception report for each source and it is tagged by one SSRC value [12].

### 2.8.5. RTCP Source Description, SDES

<table>
<thead>
<tr>
<th>Version</th>
<th>P</th>
<th>RR Count</th>
<th>PT= 202</th>
<th>Message Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CSRC &amp; SSRC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SDES items</td>
</tr>
</tbody>
</table>

This is a way to introduce a user agent to the rest of the world. There is detailed information about identity of the participant in any conference. The following items are sort of information one can find in this section Name, Email, Location, and CName [12].

There other RTCP reports that are similar to this such as Receiver Report with Report Type = 201 and Bye Message = 203 which take care of situation where a lengthy silence of a source is not due to network problems rather the source has left the session. RTCP is just a report protocol and it will not resolve any issue, yet its job is really critical in the real-time multimedia streaming [12].

Please notice that reports issued by RTCP and how they are resolved are very complicated. For instance, some algorithm is used to recover the lost packets or minimize the loss of packet, but it is beyond the scope of the thesis work.
2.9. Related Work

In [20] Authors explain DICOM (Digital Imaging and Communications in Medicine) protocol: This protocol is suitable to stream digital medical images, file structure of biomedical images, and image related data. This is a very popular protocol for distributed medical imaging systems. This thesis work uses iRTP protocol instead of DICOM protocol. We wanted to test the performance and features of iRTP compare to other protocol such as RTP.

The Authors in the paper [17] tries to use parallel programming methods for imaging applications on Multi-Core platforms. On the other hand our thesis work uses multi-threading on Multi-Core platform, yet we process text data and voice instead of bitmaps. Text data is converted to voice and then streamed over VoIP; therefore we have a different contribution than this paper.

Authors of the paper [19] investigate 3D reconstruction of images using RF signals received from Ultrasound array probes with different geometry and imaging schemes. The software receives Raw RF data and then the images are reconstructed as 3D cross-sectional images. The software uses data-level parallelism in beamforming and image processing operations. But our thesis work samples voice and processes text medical data using multi-threading on a multi-core CPU platform. We generally can extend our thesis work to be adapted to handle many ECG machines. Thus, our work contributed in a wireless environment to process different sort of data other than imaging data using DICOM protocol. Actually our thesis work could be a good addition to this article that is when we want to use a wireless ultrasound probe.

Authors in the article [21] explain that VoIP media phone are built on a single CPU and single process platform. The voice processing framework captures the voice, processes one audio packet, and delivers it to IP interface. On the other hand in this thesis work, we have created a set of the multi-threading libraries therefore it is possible to extend number of calls in a conference as well as number of media channels. Thus, one of our thesis contributions is multi-threading framework which enables many media phone calls and media channels within a multi-core platform.

In [18] authors created SAPPHIRE software in Iowa state university which is a middleware and software development kit, developed to increase productivity in using parallelism to decrease execution time of analysis of video packets, such as medical video. It is a real-time image quality measurement in colonoscopy procedures. There are 28 threads that process 90 video frames per second. This thesis work presents a multi-threading scheme that can increase number of threads that take or make SIP calls up to 100 calls if the hardware is update for load balancing. Our contribution is through using a new protocol that can speed up streaming of medical data, such as ECG data without losing any bit of medical data. Each packet contains several channels of ECG data therefore for 100 calls we actually can stream up to several hundred channels without using TCP protocol which is very time consuming protocol. Modification of the software allows us to increase the number of ECG data channel up to several thousand channels which is enough to serve 100 ECG machines.
2.10. Summary:

RTP is a light protocol which can be used for real-time applications. It is proven to be the best protocol for voice and video streaming. RTP cannot be easily replaced unless there is a protocol with better features and puts less stress on bandwidth. Per five RTP packets we only sent 1 RTCP packet. RTP/RTCP ratio depends on mainly how many participants are in the conference because we don’t want RTCP takes the whole bandwidth. We discovered that SIP is very simple, but very intelligent protocol that allows a call to be conducted over internet among mobile communication systems. SIP has many industrial applications. It is a powerful tool to connect mobile SIP-enabled devices to perform complicated tasks. PJSP library is implementation of SIP stack in c language. It supports core functionality of the SIP. It allows one to create SIP phone, extend SIP stack, add new services, create a conference to mix signals, and direct many SIP calls in a chat room. Its PJSUA library is stack built over PJSIP to simplify more complicated projects. It is a flexible and powerful library to be used in visual c++ applications.
CHAPTER 3: iRTP protocol

3.1. Introduction

The iRTP (iyad Real Time Protocol) [15][16] is a protocol created by Dr. Iyad Al Khatib, and the aim of the protocol is to enhance end to end performance via the packetization of voice and data in one IP packet of max 1500 bytes.

3.2. iRTP Structure

iRTP is a media and data protocol, which is designed to carry both media and data, in a fashion that is backwards compatible with RTP, but has more signals for control for data loss. Since iRTP was designed to suit the devices that also use RTP, then iRTP packet is encapsulated inside the RTP. Hence the first part of the iRTP header is nothing but the original RTP header. Former protocols can only stream 20 milliseconds of voice/video data. Thus, with an 8000 hertz data sampling clock rate, and with a sample of size 2 bytes, each frame size is equal to 320 bytes. Moreover, using the G.711 codec reduces the frame size of the voice/video to 160 bytes. Consequently, each RTP packet of size 200 bytes will be transmitted over IP networks. The maximum allowed size for an IP packet is 1500 bytes over Ethernet, and with the RTP header being of size 12 bytes, the UDP payload is 1472 bytes. There is about 1300 bytes of payload that is empty in the RTP packet and this is an overhead on networks that reduces bandwidth of the network.

Figure 23 shows payload size is not limited in size anymore; therefore one can packetize more data in an iRTP packet compare to RTP that is less pressure on bandwidth. It is obvious that there is a wider bandwidth using iRTP vs. RTP.

Figure 23: iRTP over SIP data over datagram. iRTP is compatible with RTP. AD stands for application dependent size for indicated field. Vi stands for volume i of ith data (of the ith application). di stands for data of ith application. sqi stands for sequ
We send within the SIP request as illustrated in Figure 24 a new item named iRTP

<table>
<thead>
<tr>
<th>Method</th>
<th>Address</th>
<th>SIP Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Via (Route)</td>
<td>From</td>
<td>To</td>
</tr>
<tr>
<td>Call-ID</td>
<td>CSeq</td>
<td>Max-Forwards</td>
</tr>
<tr>
<td>User Agent</td>
<td>Content Length</td>
<td>Allow</td>
</tr>
<tr>
<td>User Agent</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 24: Example of SIP Request Message description

Figure 24 shows a SIP Request message start with a line called ‘Request Line’ (green line in Figure 24) is made of three subfields ‘Method’ field (e.g. INVITE, BYE, etc.), ‘Address’ field (where the message is being sent e.g. sip:sohrab14@iptel.org), and ‘SIP Version’ field (SIP Version is being used e.g. SIP/2.0).

Next line is called ‘Message Header’ field (blue lines in Figure 24 is made of four subfields ‘Via’ field (with information about Route of message e.g Via: SIP/2.0/UDP 192.168.0.100:5060), ‘From’ field (e.g. source address From: <sip:simpleuac@192.168.0.100>), ‘To’ field (destination address e.g To: sip:sohrab14@iptel.org), ‘Contact’ field with information about sender (e.g. Contact: <sip:simpleuac@192.168.0.100:5060>).

Next line in message header field is made of ‘Call-ID’ field that is a unique id for a call (e.g. Call-ID: 9579efbc5d21435aa9da3ce8f072ef34), ‘CSeq’ field which is a sequence number of SIP Request (e.g. CSeq: 18467 INVITE), ‘Max-Forwards’ field (with information about maximum number of proxies that can forward the sip message e.g. Max-Forwards: 70), and ‘Allow’ field (with information about capabilities of the SIP Agent, SIP is an intelligent protocol and should instruct the entire system how it can start and end a media session with old legacy, new, and futuristic communication systems. For instance following capabilities are possible; Allow: INVITE, ACK, BYE, CANCEL, UPDATE, PRACK, iRTP). As one can see iRTP is in the list of methods and features of SIP Agent therefore user agent can initiate a media session with other SIP Agents using iRTP. Thus, other user agents will negotiate with iRTP-enabled agent and if they have iRTP capability they would start an iRTP session to stream iRTP packets over networks.

Message Body contains supported codecs by each SIP client and it is important because no RTP/iRTP media streaming is possible if different codecs are used by either client that is we will not hear voice, see video, and no transmitted data will be received correctly. It is shown in Figure 24 an example of a SIP Response message.

<table>
<thead>
<tr>
<th>SIP Version</th>
<th>Code Of Response</th>
<th>Name of Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Via(Route)</td>
<td>To</td>
<td>From</td>
</tr>
<tr>
<td>CSeq</td>
<td>Server</td>
<td>Content Length</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 25: Example of SIP Response Message
It shows in Figure 24 a Response message. The top row contains the SIP version; code of response which is a number comes next (e.g. 200); name of the response completes the response message (e.g. OK). The blue part contains the header fields which are like Request message (via, To, From, Call-ID, CSeq, Server, and Content Length). Message Body contains all stream and codec parameters that each user agent needs to have to make sure that every one setup multimedia parameters similarly so that they can initiate a media session.

Medical Systems are improving very rapidly and the major focus is on better image resolution for less radiation, telemedicine developments, and less expensive medical systems, shortening of examination time, more precise therapeutical systems in cancer therapy. Many factors plays vital role to make a medical device valuable except the one mentioned above.

In this thesis work several objectives are achieved. To send biomedical signals over internet from source where medical data is acquired from a given patient to destination where the medical staff tries to analyze them for diagnosis. Obviously we need to consider technologies that can save us time, money, energy, design time, design modification time, and scaling of the project.

Sending biomedical signals along with iRTP allows to present medical data and voice that are more appealing and richer, therefore medical staff can access information from anywhere with best precision; data presentation is always very important part of scientists work in medical systems and other technologies because any information we acquire are in form of numbers and mathematical models; the numbers are obtained from sampled data, therefore presentation Presentation of data sessions with voice files among different medical centers or exchange of such rich content would be a good feature.

From the parallel processing point of view, the Multi-core architecture allows us to perform SIP negotiation for each call in such a way that each call is processed in a core (CPU) of the computer; this way we reduce the time of the SIP signaling process shorter and more economical. As we know multi-core solution allows us to use CPUs with lower clock rates for the same amount of work performed in the same period of time; less CPU frequency leads to less power consumption and more battery life and greener call.

The main contribution in this thesis is the parallel design of the already available iRTP architecture. Hence our iRTP implementation in this thesis is Component based and distributed (COM/DCOM) and it is built on top a multi-core CPU device; it has multi-processing and multi-threading capabilities which allows us to perform different tasks in the system in different cores of the CPU. The COM/DCOM based architecture allows us to have a solution that can be scaled, expanded, and modified in later stages (i.e. for future changes, when we need to increase functionality of the solution by adding new features).

3.3. Summary:

The iRTP protocol allows several channels of data to be packetized in one iRTP payload with voice together. Furthermore, our contribution in this thesis is not in the design of iRTP but rather in the parallel architecture for implementing iRTP on multi-core devices. Our parallel implementation is achieved within a multi-threading frame work, which results in an additional boost in speed of processing too.
CHAPTER4: Parallel Design

4.1. Introduction

Our design criterion has following requirements: parallel architecture, component based design, SIP capability for VoIP, and low power consumption. Therefore, we had to face challenges of migration of all mentioned technology above in one design. Furthermore, we designed a comprehensive test scheme for optimal QoS for the voice and data delivery in a SIP session. A logging file system within our thesis work and PJSIP library help us to debug a good parallel design. Our design has an acceptable quality of voice and data delivery, provided the hardware and network within existing latency limitation.

4.2. VoIP Parallel Architecture

We firstly present the core of architecture of the RTP protocol which is designed using ATL (Active Template Library) and COM (Component Object Model). We explain mechanism of packet streaming inside this architecture from source to destination.

Secondly the Design of architecture using Object oriented methodology such as BOOCH. OOD/OOP allows creating a reusable software solution with less development time and memory space for a given task. COM stands for component object model which enables designer to split the architecture into smaller reusable components just like parts of a big machine that can be divided into smaller parts. The smaller parts are designed, built, and assembled together to create bigger parts of final product. The hierarchical model eliminates redundant components. The method facilitates maintenance and modification of the software. Thirdly we needed to create an architecture that is scalable for future requirements such as installation on larger systems. Since we may need to support cloud computing, that requires COM+ scalability in client/server architecture. Currently all the components that we have designed supports COM+ but all the components are installed only locally on one PC. Furthermore, the possibility of adding components to the architecture that ought to serve many clients simultaneously. This capability is not there unless the component is designed as COM+ object which gives it scalability.

4.3. VOIP Parallel Architecture

The VOIP architecture is the core of the thesis it is designed to meet RFC regulations. They must be followed in order to design correct architecture for protocols used in VOIP. RTP packet is then encapsulated into UDP packet to be sent over internet over IP protocol. It is shown in Figure 26 the architecture of VOIP which is a base for our final design which is used for ECG telemetry; this architecture has following specifications:

1. Object oriented design (modular and reusable)
2. Client/Server model
3. COM+ design (Scalable objects)
4. Multi-threading server objects
5. Multi-core platform
6. ATL: Active Template Library: ATL objects are small size
7. STL: Standard Template Library; efficient, solid, and concise code
8. A parallel architecture for a pure multi-core solution perform the session initiation in SIP for a VOIP session
Figure 26: VoIP architecture over RTP protocol
VOIP architecture over RTP protocol contains of four different architectures as follows:

1. VOIP architecture: it manages the sessions and entire activities such as making calls, start of SIP Session, successful SIP negotiation, RTP streaming among SIP clients over internet.
2. Multimedia architecture: it contains all multimedia related objects that capture/or play audio, video, and data. There are special tasks relevant to multimedia data manipulations that are performed in this module.
3. RTP architecture: it contains all objects that implement the Real Time Protocol architecture. This module should interact with PJSP library to setup proper media channel for real-time multimedia streaming.
4. RTCP architecture: it manages the quality of the traffic for RTP data stream.

This architecture is extendable. It is shown in Figure 26 the core objects that play a vital role in VOIP architecture; it is easily comprehended how tasks are distributed among objects. We describe them briefly to shed light on some of the details that clarify the architecture’s functionality.

IDemo is the main client that starts a VOIP session. IDemo initializes COM library and creates source, initializes multimedia real-time objects that capture voice and data. It creates almost all of threads and events in order to capture and manipulate voice data and start a PJSIP module. IDemo calls other main objects of the VOIP architecture to manage a VOIP session; ISession is responsible to manage a SIP session. IDemo will use IMedicalCapDevice to capture ECG signals and process it to be packetized together with voice. This packet is sent to another user agent in a SIP session. MedicalCapDevice has necessary interfaces to communicate with an ECG device to collect data (simulates an ECG device). IDemo has system resources such as events to wait on worker threads to do their initializations while starting a SIP call. It waits for the PJSIP module to make a SIP call to another callee user agent; a SIP INVITE message is created and sent to callee in order to start a SIP negotiation and as soon as a successful negotiation is achieved the RTP voice packet is prepared streamed over RTP over internet.

A SIP multimedia source can be both: microphone and ECG medical device (this is simulated by a disk file of ECG text data). SIP device send both: voice and ECG text data over two separate UDP channels. Voice is sent over RTP protocol over connectionless UDP protocol and ECG data over reliable TCP protocol.

ISessionStat is going to collect statistical data about how session progresses as the participants join a session, behave in a session, and exit the session. RFC regulations determine the tasks this object is expected to perform. This object is for future use.

Real-time multimedia architecture is an important part of the VOIP architecture. We need to capture voice, sample it in order to convert to digital signal, and packetize it into a RTP packet. On the other hand ECG data stream is sampled and packetized into a TCP packet and sent over internet in a separate multimedia data channel because it is a data stream not voice; furthermore we can add video to have a richer content and improve data presentation.

There are two transport layers in thesis. Firstly we use SIP transport layer to communicate with SIP server proxies to locate callee in order to perform SIP negotiation for establishing a connection between two user agents. This needs a transport layer which is either TCP or UDP, further we need another transport layer which is UDP through which RTP streaming is performed.
Multimedia module uses Microsoft Windows DIRECT SOUND from multimedia library (DirectX object) to capture sound under windows vista/windows7/windows8 operating systems. It is very simple DirectX object of windows multimedia library; Directx has a rich set of functionality for voice and video content; Directx is very powerful, high quality, small latency, and high resolution tool to create multimedia projects such as VOIP, computer games, image processing, and entertainment applications. It is the best product Microsoft has developed for multimedia applications; please notice that there are many different objects in the DirectX module that can do similar tasks, but Direct Sound object has sufficient features to satisfy our high quality of data stream for the thesis work. Direct Sound has a library with high speed and quality which makes it favorable for data acquisition for real-time applications.

Multimedia module contains three core objects IAudioCapDevice, IVideoCapDevice, and IMedicalCapDevice. IAudioCapDevice has all the features we need to capture and play voice. We can add new capabilities such as special effects and filtering of captured voice. Therefore, COM development offers better ways to extend the features than regular c++. Thus, integration of COM objects with PJSIP library provides us with a powerful tool to select and mix any part of the voice, video, and data. Ultimately we have a solution that can stream multimedia data with flexible mixture. One can throw selected part of real-time multimedia into a PJSIP conference. The final product could be a selected speech mixed with certain movie clips. This could be tagged with medical data as a wall paper in the background.

IVideoCapDevice implements video related functionality; it is beyond this thesis work. Please notice that COM technology implements a protocol rather than individual line of code so that allows the objects interact with one another to perform a task. Each interface adds a new functionality which is not necessarily needed to be implemented at the time being, but it is part of the VoIP architecture. Hence, to follow the COM protocol we should have them as virtual interfaces and leave it until time comes to implement those virtual interfaces.

IMedicalCapDevice is responsible to capture, sample, condition ECG digital signal, and write the results into a file that other parts of the VoIP architecture could take the digital data and encapsulate it in RTP packets and send it off to the destination user agent over LAN/WAN.

RTP architecture takes the role of the packet-level management of VoIP system; RTP module sends packets over networks; it uses the UDP transport level protocol to stream real-time voice over internet.

1. RTP module contains three main objects IRtpSndPkt, IRtpRecvPkt, IRtpCom.
2. IRtpSndPkt: it implements entire functionality of a RTP packet sender.
3. IRtpRecvPkt: it implements functionality of a RTP packet receiver.
4. IRtpCom: it is responsible to establish a UDP connection between two clients or a client and a server. We used this to establish a connection in LAN to have a voice conference between a client and server. (PJSIP library provides its own transport level connection)

RTCP architecture contains objects to measure quality assurance of the communication between two clients. RTCP creates reports which are the base for calculations of packet delay, jitter, and lost packets. Reports are used to change the codec so that RTP packet traffic flows smoothly among participants of a SIP session. QoS is guaranteed so that lost packets jitter, and packet latency is minimized for a good quality of VoIP service.
RTCP has reports that can inform the clients of a QoS of a session in order to adjust the packet rate in such a way that the other side can handle the arrived packet traffic; therefore, it will balance the traffic over internet for optimal quality.

RTCP factors in calculations latency of the network due to traffic, physical RC of the networks, and hardware devices, therefore, the packet sender can take necessary steps to cancel out latency and packet delay; this is important when voice and video are sent simultaneously. Parameters such as jitter measurement helps to regulate the packet rate constantly; RTCP sends reports between clients which participate in a session to keep track of packet traffic in accordance of number of lost packets.

4.4. Design of VOIP Parallel Architecture with OOD/OOP

The VOIP architecture design meets the requirements of object oriented design technology; therefore, it is a reusable code; therefore, future modification and extension to the architecture and implementation are easily possible. This methodology is vital to architecture design.

4.4.1. UML Representation of VOIP Parallel Architecture

UML is graphical language to develop small to large systems; it is created based on BOOSCH methodology. UML is a tool that allows the designer to design his software system in form of both: black boxes convert and pseudo code. Modules consequently are broken down into smaller components; this hieratical design form is basically the key to any system design especially when one deals with very large complicated ones; therefore, large systems that are designed in accordance with UML language follow the OOP/OOD design rules.

UML design rules enforces object oriented design rules that are vital to achieve a sound system design with little or no redundancy. Inheritance and aggregation design rules that are vital key rules to OOD/OOP are carefully administered so that we can reduce the size of code and make each block of code reusable for future developments. Each block of code that is called an object can be extended to meet design modifications in the system; this is a great feature, because we need to harness very complicated system that usually looks very unorganized into very organized set of objects; each object is responsible to perform certain tasks that hardly overlaps with tasks assigned to another object; redundancy is not tolerable in OOD/OOP design rules; we always enjoy design of any architecture using OOD/OOP, because it brings out the real power of OOD/OOP and makes a project very successful.
4.4.2. Design of RTP Architecture

Figure 27: RTP class Diagram in ATL COM technology

4.4.3. Algorithms of VoIP multimedia using RTP protocol

Client1, client2, ..., client(n-1), client(n) will perform the following algorithm below. There is a component of SIP which is called Source. Source represents a microphone, webcam, and any source of medical data acquisition machine.

- Create an ISource component from RTP class Diagram in Figure 27. ISource has all necessary to take the role of a RTP Multimedia source.
- MarshalSrc2Header marshals ISource to IHeader object which consequently will marshal SeqNo and SSRC variables from ISource to IRtpHeader object.
- Acquire IAudioCapDevice and fetch timestamp to tag a RTP packet with timestamp and then send it.
- Acquire IRtpHeader->put_timestamp to marshal timestamp to the IRtpHeader from IAudio CapDevice.
- Call PrecreateRTPPacket to make necessary intiation in the packet and create parameters necessary for RTP Packet.
Figure 28: Architecture in UML representation over iRTP Protocol
4.4.4. VoIP architecture over iRTP protocol

IRTP architecture is designed so that we can reuse RTP architecture objects. Furthermore it contains objects which are specific to IRTP architecture.

VoIP architecture contains IDemo which is the main client. ISource supports both voice and medical data sources. Multimedia architecture consists of IIiRTPAudioCapDevice (captures voice which is packetized into an iRTP packet), IIiRTPVideoCapDevice (captures video which is packetized in an IRTP packet), and IIiRTPMedicalCapDevice (captures medical data and packetize it into an IRTP packet).

IRTP architecture is responsible to create and send/receive iRTP packets. IIiRTPSndPkt is responsible to cooperate with other objects in VoIP architecture to create header and payload of an IRTP packet. IIiRTPRecvPkt is responsible to receive an IRTP Packet.

RTCP architecture is responsible to create RTCP packets. It takes role of QoS in VoIP and if there is a slow down in packets flow automatically user agents can switch to other codecs to compensate. If a user has suppression of voice, RTCP packet is used as an indication of liveliness of the user agent in a session.

IRTP retransmit unit collects information about lost packets, processes them, and retransmits lost packets again in the next packets. This is part of the IRTP protocol. Following Algorithm describes retransmit unit operation:

**Algorithm:**

1. Create an in-process table to save all the medical data sent to/from sender client/receiver client.
2. Every x milliseconds receiver sends a list of lost packets, first received packet, and last received packet in an acknowledgement packet.
3. Sender receives acknowledgement packet from receiver so that it retransmit the lost packets back to the receiver client. This process continues until all the packets are received.
4. We would give higher priority to lost packets to be retransmitted before sending new packets.
5. We create two different ports one for sending new packets and one for resending lost packets. We need to make sure that this is not overlapping each other and there is some sort of synchronization between these threads.
6. If port 5528 is for sending new packets then port 5529 could be for resending of lost packets.
4.4.5. Design of VOIP multimedia objects with iRTP protocol

Figure 28 shows iRTP architecture design. It has four main modules that interact with each other. PJSIP library architecture objects are used sparsely in different parts of iRTP protocol architecture. It is noticeable that we reuse RTP modules and objects to design and implement iRTP architecture. There is an inter-architectural relationship among the iRTP objects that are important to find these relationships are base of a protocol for iRTP objects to communicate with one another. A client that uses iRTP objects takes advantage of the architectural frame work when it creates an object of the iRTP architecture and then performs the tasks that an iRTP sender and receiver are supposed to do. Most of the objects we have created so far are related to the creation of IRTP packet that is send/Receive of the iRTP packets. The text data object (IiyadDataBuffer) which manages text medical data.

4.4.6. Algorithms of VOIP using iRTP protocol

Codec parameters: (Sampling rate: 8000; 16-bit stereo; 1 channels)

4.4.6.1. Client1 & Client2 sides

1. Create an IiyadSource to encapsulate all the activities of the source. Source could be a webcam, MIC, a data acquisition tool, Medical Device, and so forth.
2. Create IiyadCapDevice object. It represents all the data acquisition activities. It can store the data coming from different medical devices, store them. It can use other objects in different modules to process it and then output it to display screen or a controlling system. It could be blood pressure presentation in form of nice graphs or statistics or even giving alerts to user.
3. MarshalSrc2IiyadHeader object. The data stored in IiyadSource object needs to be fed to the IiyadHeader object.
4. Query IDataBuf from IiyadCapDevice (IMedicalCapDevice) object.
5. Query IiyadHeader from IiyadSndPkt.
6. Acquire IiyadSndPkt object from IiyadSource.
7. Acquire IiyadHeader from IiyadSndPkt.
8. Acquire IiyadPacket from IiyadSndPkt. (Dynamic cast<…>);
9. pAudioCapDevce.p->AddRef();
10. pDataBuf->put_Bytes2Write (...);
11. pDataBuf->put_Data (...);
12. Create IiyadDataBuffer (...). (pIyadDataBuffer)
13. pIyadDataBuffer->MoveFirst (...);
14. Create a Thread here and allow the control process to loop and keep sending Medical data or pause whenever you want.

iRTP will guarantee no blood pressure data is lost over UDP connection. This protocol will check for lost data and resend it again. It is more selective than TCP and better performance.
We find first record and last record to use for re-transmit of Medical Data to receiver again.

| 1 | BLDPR 1 |
| 2 | BLDPR 2 |
| n | BLDPR n |

Algorithm:

7. Create an in-process table to save all the medical data sent to from sender client to receiver client.
8. Every x milliseconds receiver sends a list of lost packets in an acknowledgement packet, along with first packet received and last packet received.
9. Sender receives acknowledgement packet and resends the lost packets back to the receiver client. This process can go on forever until all the packets are received.
10. We would give higher priority to lost packets to be resent rather than sending new packets.
11. We create two different ports one for sending new packets and one for resending lost packets. We need to make sure that this is not overlapping each other and there is some sort of synchronization between these threads.
12. If port 5528 is for sending new packets then port 5529 could be for resending of lost packets.
13. IiyadSsrcStats encapsulates some of the tasks.

Algorithm for Resend:

1. Set up a timer that sends a trigger to the IiyadResendPkt object, consequently it will send a request to IiyadSsrcStats to send a list of lost packets.
2. IiyadSsrcStats create an Acknowledgement packet with a list of lost packets and sends it back to the IiyadResendPkt object.
3. A timer is created to send a trigger to the IiyadResendPkt object every 5[sec]. five seconds seems a good time interval for this.
4. When IiyadSsrcStats finishes sending acknowledgements it waits for the resent data and as soon as the data is received it cleans up the lost packet table.
5. When IiyadResendPkt resends the all the lost packets requested it will delete them from the list as soon as it receives another a

4.5. SIP Architecture

SIP Architecture is designed using PJSIP Library. It is a free library and it is developed in visual c++ 2005 in C language.
It is a multi-threading architecture which can create sip messages in order to send a sip invitation and request to establish a sip connection. callee decides to accept the call sends a response message 200 OK back to caller user agent. Thus, SIP negotiation is successfully done and both user agents try to setup a multimedia channel with a chosen codec to stream multimedia data specifically voice and medical data, multimedia protocol used is RTP or IRTP. Please notice we used SIP, RTP, IRTP as preferred protocol UDP to encapsulate our multimedia data. UDP will be streamed over internet via IP protocol. we’d like to emphasize that we create a UDP connection so that user agents negotiate to enter a conference, afterwards the same UDP channel will be used to stream multimedia data. Although the same UDP channel is used for both SIP and RTP, but port numbers are different.

4.5.1. SIP and RTP/iRTP state diagram architecture

There are three architectures take part in this thesis. It is shown in Figure 29 the structure of interaction among SIP, RTP, iRTP architectures. Main thread of thesis does necessary initialization and starts the SIP thread simultaneously meantime it starts capturing voice as PCM format.

![Figure 29: RTP/iRTP architecture and SIP state diagram architecture, circled numbers show sequence of actions taken.](image)

Main thread performs initializations. Thus, it contains RTP and iRTP architecture creation, creation of system voice capture thread, create siprtp_thread. Multi-threading architecture uses events to stop/start threads.

It is shown in Figure 29 that voice capture thread waits on an event until the SIP negotiation is complete. This ensures proper sequence in multimedia streaming over RTP or iRTP. The architecture in Figure 29 forces a mechanism that checks whether we use RTP or iRTP protocol and this is known
only by the time SIP negotiation is successful. ALLOW field in SIP message INVITE and Response to that will set a flag that determines to switch between RTP or iRTP protocols.

HandleNotification will set size of each voice frame ranging from 5-20 milliseconds. This is the control process for voice capture system thread. It is called when the voice in primary buffer reaches a certain level. Media_thread1 box is called by handle notification box which is created a different thread. Media_thread1 handles both: iRTP and RTP packets. Please notice that sampling rate, number of channels, and number of bits per sample are determined in siprtp_thread from negotiated codec between client and server. Thus, frame size of voice raw audio pcm format is determined by codec that user agents agreed on.

Siprtp_thread does all of the SIP initialization, creation of thread to watch the port for sip messages, create INVITE message, process response to INVITE, create SDP message for codec negotiation, and create media session. Media_thread1 creates RTCP packet and sends them over UDP.

Figure 29 shows that RTP-thread upstreams PCM packet, SIPRTP-thread downstreams the PCM packet, and send it over UDP to another SIP user agent.

Figure 29 shows a state diagram and how RTP and SIP architectures interact with one another. Please notice the numbers next to legs of the architecture that show flow of state diagram. We create a thread under RTP architecture and in turn this creates a child thread with name SIPRTP-Thread and runs in SIP architecture (pjsip architecture), therefore consequently RTP-thread and SIPRTP-thread interact with one another. RTP-thread captures voice and passes it to the SIPRTP-thread and SIPRTP-Thread encodes PCM packet into PCMU (G711/ULAW) format, packetize it into the RTP packet, and sends it to callee user agent. We must make sure that the other sip party also has PCMU (G711/ULAW) decoder in order to decode the PCM packet back to its original size. G711 achieves compression ratio of 2:1. (Refer to G711 codec for detailed information) SIPRTP-thread negotiates with the other SIP party to send compressed RTP packets over internet. When a digital sample is encoded with G711 its size shrinks to half size of original signal. Thus, (8000 hertz, mono channel, and 8-bit size) are characteristics of the encoded RTP packet.

G711 Codec with following characteristics: (G711 is a free codec and it is mostly used for voice/Video.)

Sampling frequency/clock rate: 8000 hertz

Each sample is 16-bit size

Only one channel (mono)

5.6. Summary

RTP and IRTP architectures migrate with PJSIP architecture to build a component based architecture. Therefore, we are able to take advantage of each technology which improves the overall design. PJSIP and Microsoft window provide different SDK functions to create threads. Thus, this poses challenges in design and implementation which were resolved. PJSIP has a robust threading mechanism that can handle many calls at the same time. Microsoft ATL COM is a very reliable technology which allows us to develop once and use it forever policy in any platform and environment. Interaction between PJSIP and ATL COM is very vital to proper operation of
the thesis. We experienced many problems in order to synchronize their threads to cooperate with each other to have a working system.
CHAPTER 5: Test and Results

5.1. Introduction

We designed three different test benches as follows: Local Area Network testbed, City Network within one country testbed, and International testbed. We measure latency of RTP and iRTP packets in order to compare them. Therefore, this is a good criterion to determine which of the protocols has a better QoS. We use wireshark to measure the packet delays for both: RTP and iRTP protocols.

5.2. Test Design

The methodology is designed in order to discover performance of iRTP versus RTP plus TCP in a media session in an almost non-controlled environment. We perform the experiments for LAN (controlled environment), but in case of internet then the data travels through routers and servers to reach its destination (complex test for uncontrolled environment). Additionally the comparison of each individual voice and data of iRTP to RTP and TCP protocols gives a better detailed on where we get a better performance. Analysis of the results shows is iRTP generally an acceptable protocol and what are its advantages and flaws over RTP and TCP.

5.3. Testbeds

![Diagram of testbeds](image)

Figure 30: end-to-end latency measurement for international testbed
5.4. Test Types

5.4.1. Only Audio (OA)

We only intend to measure latency of audio stream over RTP and iRTP when only audio signal is streamed in a media session. This will help us to compare performance of RTP versus iRTP protocols when only audio is streamed.
5.4.1.1. Only Audio over RTP (OAoRTP)

To measure latency between the two user agents who stream RTP packets we need to synchronize time on two laptops but this is not easily possible. Thus, we decided to measure latency between two packets received sequentially on receiver end which we call it back-to-back latency for two RTP packets.

5.4.1.1.1. LAN setting for (OAoRTP)

Measure the back-to-back latency of sequential packets over RTP where two user agents establish a SIP connection and consequently a media session established over RTP protocol over a local area network.

![Graph showing latency measurements for LAN setting for OAoRTP test runs](image)

Figure 33: Averages of latency for LAN setting for Only Audio over RTP test runs
5.4.1.1.2. One Country (O AoRTP)

Measure back-to-back latency of sequential packets over RTP where two user agents establish a SIP connection and consequently establish a RTP media session over internet in one country.

Figure 34: Averages of latency for Only Audio over RTP-LAN versus Average of Averages of latency for Only Audio over iRTP for LAN setting test run

Figure 35: Averages of Latency for Only Audio over RTP for one country setting test run
5.4.1.1.3. International

Measure back-to-back latency of two sequential packets over RTP where two user agents establish a SIP connection and consequently a RTP media session over internet in two different countries.

5.4.1.2. Only Audio over iRTP (OAoi)

We measure back-to-back latency of sequential iRTP packets between two user agents which stream back and forth packets over iRTP.

5.4.1.2.1. LAN setting for (OAoi)

Measure the back-to-back latency of two consequent packets over iRTP where two user agents establish a SIP connection and RTP media session over a LAN.

![Graph showing average latency for Only Audio over iRTP test run for LAN setting test run]

Figure 36: Averages of Latency for Only Audio over iRTP test run for LAN setting test run

5.4.1.2.2. One Country (OAoi)

Measure the back-to-back latency of two consequent packets over iRTP where two user agents establish a SIP connection and RTP media session over internet in one country.
5.4.1.2.3. International (OAoi)

Measure the back-to-back latency of two consequent packets over iRTP where two user agents establish a SIP connection and iRTP media session over internet in two different countries.
5.4.2. Only Medical Data (text) (OMD)

Medical data is blood pressure in text format in a data file. We measure back-to-back latency of two consequent packets carrying only blood pressure text data sent over TCP and IRTP. We analyze performance of the IRTP versus TCP which helps us to understand VoIP over IRTP and how it can help to speed up the media session of a SIP call.

5.4.2.1. OMD over TCP (OMDoTCP)

To measure OMDoTCP we needed a TCP client to send TCP text data over TCP to a TCP server. We measure round-trip-time of TCP packets carrying text format data. Please refer to [2.6] for TCP protocol features and deficiencies.

5.4.2.1.1. OMDoTCP- LAN setting

We measure round-trip-time of TCP packets which carry text format data in a LAN setup. Thus there is a TCP client and a TCP server connected over a LAN and TCP client send TCP packet and TCP server receives TCP packet.

![Total-Average-of-Only-Medical-Data-over-TCP (OMDoTCP-LAN)](image-url)

Figure 39: Averages of latency for Only Medical Data over TCP for LAN setting test run
5.4.2.1.2. OMDoTCP-within one country

We measure round-trip-time of TCP packets which carry text format data over internet within one country. There are TCP client/server in the same city connected over TCP protocol. The TCP client sends text format data over TCP to the TCP server.

![Figure 40: Averages of Latency for Only Medical Data over TCP for one country test run](image)

5.4.2.1.3. OMDoTCP-International

We measure round-trip-time of TCP packets which carry text format data. There is a TCP client which sends TCP packets which carry text format data to a TCP server over internet between two different countries.

5.4.2.2. OMD over iRTP

We stream text data in PCM format for voice over iRTP. Technically RTP packet is streamed over networks, but each RTP packet encapsulates an iRTP packet. A caller places a call to a callee to establish a SIP call which consequently leads to an iRTP media session. iRTP packets flow from source pc to destination pc to measure back-to-back latency of two consequent iRTP packets.

5.4.2.2.1. LAN setting for OMDoi

We send text data in pcm format as voice between two SIP phones. They get connected via SIP server and SIP proxies and then start an iRTP media session. The SIP phones are connected in a
LAN; therefore the SIP phones are located in local network, yet they are physically located in different buildings, offices, and so forth.

![Figure 41: Only Medical Data over iRTP over LAN test run](image)

**5.4.2.2. One country OMDoi**

We firstly convert medical text data to voice, packetize it in an iRTP packet, and stream it between two user agents. User agents are located in different cities of the same country. We measure back-to-back latency of two consequent iRTP packets in destination SIP phone and it is a credible measure of latency of the network.
5.4.2.2.3. International OMDoi

We stream text data in pcm format as voice between two user agents. They are located in two different countries. The SIP phones are connected over SIP protocol via SIP proxies. Therefore, they send/receive iRTP packets carrying text data.

5.4.3. Audio and Medical Data (AMD)

We pack together audio and medical data in pcm format. There are two SIP phones connected over SIP and established a media session using RTP, TCP, and iRTP.

5.4.3.1. Audio over RTP and Medical data over TCP

We send audio over RTP and medical data over TCP. We have two SIP phones that are connected using SIP protocol and established a RTP media session. SIP phones send and receive audio signals over RTP. We have two TCP client and server which send and receive TCP packets that carry text format data.

5.4.3.1.1. LAN setting for AoR-MDoT

We measure back-to-back latency of two consequent RTP and TCP packet that are sent in two different channels. One channel is RTP channel and the other one is TCP channel. There are two types of back-to-back latencies are measured in receiver end. Back-to-back latency for RTP audio packets and back-to-back latency for TCP packets which carry text format data. The test bed contains two SIP phones and two TCP client and server applications that are connected over a local area network.
Figure 43: Averages of Latency for Only Audio over RTP and Medical Data over TCP for LAN

5.4.3.1.2. One country AoR-MDoT

We do similar measurements as mentioned in section [5.4.3.1.1] but test bench is setup for two different cities in one country instead of over LAN.

Figure 44: Average Latency for Audio over RTP and Medical Data over TCP test run
5.4.3.1.3. International AoR-MDoT

We measure similarly to what is in section [5.4.2.1.1] with one difference in testbed setup. We setup our SIP phone along with TCP client server in two different countries instead of over local area network.

![Graph showing average latency](image)

**Figure 45:** Average Latency Audio over RTP & Medical Data over TCP; Client with All Wired and Server with G3 Connection to Network

![Graph showing average latencies](image)

**Figure 46:** Average Latency Audio over RTP and 12 channels Medical Data over TCP; Client and Server have All Wired Connection to Network
Figure 47: Average Latency Audio over RTP and Medical Data over TCP; Client and Server with wireless Connection to Network

Figure 48: Average Latency Audio over RTP and Medical Data over TCP; Client with wired and Server with wireless Connection to Network
Figure 49: Average Latency Audio over RTP and Medical Data over TCP; Client with wireless and Server with wired Connection to Network

Figure 50: Largest Latency Audio over RTP, Medical Data over TCP, and audio and medical data over IRTP

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Type of Data</th>
<th>Largest Back-To-Back Latency</th>
<th>Number of Sockets Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTP, RTCP, and TCP</td>
<td>Audio, control information, and Medical Data</td>
<td>854.378</td>
<td>1</td>
</tr>
<tr>
<td>IRTP</td>
<td>Audio, Control information, and Medical Data</td>
<td>686.551</td>
<td>2</td>
</tr>
</tbody>
</table>
Figure 51 shows the comparison between the average delays of iRTP and RTP plus 12 TCP channels for medical data. The enhancement in case of iRTP is around 74% compared to that of the RTP sending audio over RTP simultaneously with 12 TCP channels of medical data.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Type of Data</th>
<th>Average Back-To-Back Latency</th>
<th>Number of Sockets Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTP, RTCP, and TCP</td>
<td>Audio, control information, and Medical Data</td>
<td>58.36377501</td>
<td>1</td>
</tr>
<tr>
<td>IRTP</td>
<td>Audio, Control information, and Medical Data</td>
<td>13.74</td>
<td>2</td>
</tr>
</tbody>
</table>

5.4.3.2. Audio and Medical data over iRTP (AMDoi)

There is only one protocol in this section that is being used and it is iRTP. The testbed contains two laptops that are connected over SIP server and established a media session over iRTP.

5.4.3.2.1. LAN setting for AMDoi

The test bench is in local area network and packets travel over a LAN. We measure back-to-back latency of two consequent packets over iRTP protocol.
5.4.3.2.2. One country AMDoi

The test bench contains two laptops that are connected via SIP servers and established an iRTP media session over iRTP protocol. We measure back-to-back latency of consequent packets over iRTP protocol in the destination laptop using wire shark tool.
5.4.3.2.3. International AMDoi

**Figure 54**: Average Latency Audio & Medical Data over IRTP over 3G Network

**Figure 55**: Average Latency Audio & Medical Data over IRTP; Client & Server both have Wireless Network
Figure 56: Average Latency Audio & Medical Data over IRTP; Client and Server have wired Connection to Network

Figure 57: Average Latency Audio & Medical Data over IRTP; Client and Server have both Wired Connection to Network
The test bench contains two laptops in two different countries which are connected via SIP servers and established an iRTP media session over iRTP. We measure back-to-back latency for two consequent packets sent over iRTP in both source and destination sides.

5.4.4. Analysis of measurement

There are three types of test bench configuration as follows:

- LAN configuration: two laptops are connected over LAN (Figure 32)
- One country configuration: two laptops are connected over WAN (Figure 31)
- International configuration: two laptops are connected over WAN (Figure 30)

5.4.4.1. Problems

The results of tables above are spread in a large range which raised our suspicion what could be wrong with the test bench or possibly measurement methods. We discovered following problems with our measurement methods and test bench.

- Clock synchronization was done poorly.
- Latency cannot be measured in two wiresharks because the time on two stations is not synchronized.
- Workstation latency is a bottle neck in process of large size packets

5.4.4.2. Solution
The measurement of arrival time of two packets on receiver side station represents end-to-end latency. Thus, we measure back-to-back latency of packets (PKTᵢ and PKTᵢ₋₁) on destination station. Thus, absolute time difference between two sequential packets is similar to end-to-end packet latency.

Slow workstation is a problem which not always can be solved. It depends on the budget one has to buy a laptop that has low latency and high capacity to packetize large packets of size more than 640 bytes. This is especially important when one wants to run heavily multi-threading applications. DIRECTSOUND library capture audio with voice and digitize it and this is fast but it could be slow if other threads run at the same time along with voice capturing thread and that would affect the quality of the entire application.

Notice without multi-threading it could not be possible to make a VoIP application because many phone calls and many tasks are running at the same time which is not feasible to have only one thread to run the entire thesis application.

This thesis work is designed for real-time applications. Thus, it will constantly consume a lot of energy, however, since we have a multi-threading on a multi-core platform with smaller clock rate we save energy. This thesis work is a green solution. We tried to design software that guarantees a longer battery life time.

![Latency of All Test Results](image)  

**Figure 59:** Latencies of all types of protocol per different geo-references. The two bars on the top of the graph for international test is very important criterion to compare (RTP + TCP) vs. IRTP protocols.
It is noticeable that most of the tests we have done carry a weight for certain applications the VoIP protocols are concerned. Speed of the travel of the packets around the world is the biggest concern. Therefore, we study the international test results for this thesis work as a good criterion to prove IRTP is superior to (RTP + TCP); therefore, IRTP is a good choice to stream voice and data together rather than RTP plus TCP. As Medical Data grows, there is more stress on the bandwidth of networks. User acknowledges that TCP is not only to stream data, but also to receive acknowledgements from destination in order to send next packets or resend the previously packets. Therefore, it is critical to know that TCP can be very hard to use when size of the chunk of data grows. Thousands of patients wait to send their vital signs such as, ECG data via 12 wires; will send 12000s of packets over TCP which is not feasible. Thus, IRTP is much better choice for Medical applications. IRTP allows us to send voice and medical data via VoIP. Notice the calculation how much performance improvement we gain if we choose IRTP over RTP and TCP.

It is shown in Figure 59 the Latency for audio stream over RTP and IRTP and we use them to calculate the performance decrease as it is shown in equation (1).

\[
\text{Percentage decrease in delay when using iRTP instead of RTP} = \frac{(58.36377501 - \text{13.74})}{58.36377501} = 76.46 \quad (1)
\]

According to the result shown in Figure 59 and calculations in (1) we conclude that a performance gain of 76.46 percent is gained via using IRTP protocol rather than RTP plus TCP. This was main objective of this thesis work.

- This section is in reference to draw conclusion with regard to tests in CHAPTER 5: Test and Results. Obviously iRTP shows quite good strength in transport of voice and data together in one packet; it is noticeable that it results in less bandwidth usage compare to RTP and TCP. Hence, using both: RTP and TCP simultaneously in order to stream voice is more expensive than iRTP protocol where only lost packets are retransmitted (TCP protocol retransmits all packets in occurrence of only one packet loss).
- iRTP is a protocol suitable for medical application where voice and data ought to be transmitted simultaneously, but for voice-enabled application RTP is rather feasible.

5.4.5. Data Processing in Excel

There is visual basic application inside Microsoft office products such as excel application that can be launched for automation of data processing. We have used this tool to clean some of the test results.

VB Algorithm in excel to clean irrelevant packets that are not used in our calculations:
5.5. Summary

The test proves that, when medical data is to be sent together with audio, iRTP delivers better performance than RTP. Using iRTP with 12 data channels shows 74 percent decrease in delay rather than using RTP for sending audio in parallel with 12 TCP channels of data separately. iRTP creates less congestion over networks and poses less stress on bandwidth. Therefore, iRTP has a big potential to be used for wireless medical applications.
CHAPTER 6: Conclusion and Future Work

6.1. Conclusion

iRTP has the potential to enhance performance for audio and Medical Data stream over large and small area networks via the novel idea of sending the multimedia with textual data encapsulated in the same packet, hence decreasing overhead, delays, and bandwidth utilization. Our main contribution was to create a parallel architecture for the iRTP and implement it. We succeeded in doing so, and our test results show a significant improvement to the classical methods that use separate connections for each channel in order to stream audio and medical data. The international results show a decrease in delay when using our implementation of iRTP by around 74%. Two major reasons lead to such a decrease: (1) the iRTP protocol design, and (2) implementing the iRTP in a parallel and modular way within Multi-core platform. Therefore, the iRTP, which was already designed for modern communication needs, needs a smart code design that fits its main aim, i.e. to enhance performance and mainly decrease delay. To do so, we made a SW architecture that is paralleled to suit Multi-core. We discussed our SW architecture using COM objects. The technical results show that the iRTP as a protocol is very successful and that our SW architecture for Multi-core paralleling also adds value, so that iRTP parallel and modular implementation on Multi-core is a fairly fast protocol and could increase the bandwidth in medical applications.

The numbers/figures of results show that the decrease in execution time and bandwidth when using our iRTP implementation is 76.46% better than of “RTP plus TCP”.

6.2. Future Work

PJSIP library does not allow video conference in the thesis work which we should wait until in new version video stream is implemented too. We can use video capability to send medical data such as heart or brain angiography films which shows more detailed information about the patient sickness. Mixing of video, voice, and medical data could be superb features in future. We can video conference, send medical data, and so forth.

We can pick ECG signal of the patient of a certain date throw it in the newly ECG signals, and medical staff on the destination site; therefore, underline certain anamoly in QRST signal. Doctors always use the information to compare data and review it for further analysis.

These features take a lot of work and are very complicated.

6.3. Related Work

DICOM (Digital Imaging and Communications in Medicine) protocol: This protocol is suitable to stream digital medical image, file structure of biomedical images, and image related data. This is a very popular protocol for distributed medical imaging systems. This thesis work uses iRTP protocol instead of DICOM protocol. We wanted to test the performance and features of iRTP compare to other protocol such as RTP.
According to paper “Medical Image Viewing on Multi-Core Platforms Using software Patterns for Parallel Programming”, parallel programming methods is usable for imaging applications on Multi-Core platforms. On the other hand our thesis work uses multi-threading on Multi-Core platform, yet we process text data and voice instead of bitmaps. Text data is converted to voice and then streamed over VoIP; therefore we have a different contribution than this paper[17].

In reference to the paper “GPU-based Real-Time Imaging Software Suite for Medical Ultrasound”, there are RF signals received from Ultrasound array probes with different geometry and imaging schemes. The software receives Raw RF data and then the images are reconstructed as 3D cross-sectional images. The software uses data-level parallelism in beamforming and image processing operations. Our thesis work samples voice and processes text medical data using multi-threading on a multi-core CPU platform. We generally can extend our thesis work to be adapted to handle many ECG machines. Thus, our work contributed in a wireless environment to process different sort of data other than imaging data which uses DICOM protocol.

According to the article “Designing a VoIP media phone framework” most of the VoIP media phone are built on a single CPU and single process platform. The voice processing framework captures the voice, processes one audio packet, and delivers it to IP interface. In this thesis work; we have created a set of the multi-threading libraries therefore it is possible to extend number of calls in a conference as well as number of media channels. Thus, one of our thesis contributions is multi-threading framework which enables many media phone calls and media channels within a multi-core platform.

SAPPHIRE software is a contribution from Iowa state university which is a middleware and software development kit, developed to increase productivity in using parallelism to decrease execution time of analysis of video packets, such as medical video. It is a real-time image quality measurement in colonoscopy procedures. There are 28 threads that process 90 video frames per second. This thesis work presents a multi-threading scheme that can increase number of threads that take or make SIP calls up to 100 calls if the hardware is update for load balancing. Our contribution is through using a new protocol that can speed up streaming of medical data, such as ECG data without losing any bit of medical data. Each packet contains several channels of ECG data therefore for 100 calls we actually can stream up to several hundred channels without using TCP protocol which is very time consuming protocol. Modification of the software allows us to increase the number of ECG data channel up to several thousand channels which is enough to serve 100 ECG machines [18].
References


[19] Jung Woo Choe¹, Amin Nikoozadeh¹, Ömer Oralkan², and Butrus T. Khuri-Yakub¹, "GPU-Based Real-Time Imaging Software Suite for Medical Ultrasound,” 1Edward L. Ginzton Laboratory, Stanford University, Stanford, CA 2Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC

