# Design and Implementation of SIP based Embedded VoIP Phone

By

**Vipul A. Patel**

**(04MCE015)**


**Department Of Computer Science & Engineering**
**Institute of Technology**
**Nirma University of Science & Technology**
**Ahmedabad 382481**
**May 2006**

# Design and Implementation of SIP based Embedded VoIP Phone

**A Dissertation**

Submitted in partial fulfillment of the requirements

For the degree of

**Master of Technology in Computer Science & Engineering**

By
**Vipul A. Patel**
(04MCE015)

Guide

**Prof. Jaladhi Joshi**


**Department of Computer Science & Engineering**
**Institute of Technology**
**Nirma University of Science and Technology**
**Ahmedabad- 382481**
**May-2006**


This is to certify that the Dissertation entitled

# Design and Implementing of SIP based Embedded VoIP Phone

Presented by

**Vipul A. Patel**

has been accepted toward fulfillment of the requirement
for the degree of
Master of technology in Computer Science & Engineering

Professor In Charge

Head of The Department

Date: 29/04/06

# *CERTIFICATE*

This is to certify that the work presented here by **Mr.Vipul A. Patel** entitled "***Design and Implementing of SIP based Embedded VoIP Phone*"** has been carried out at **NirmaLabs** during the period **September 2005 – May 2006** is the bonafide record of the research carried out by him under my guidance and supervision and is up to the standard in respect of the content and presentation for being referred to the examiner. I further certify that the work done by him is his original work and has not been submitted for award of any other diploma or degree.

Prof. Jaladhi Joshi

Date: 29/04/06

# *Abstract*

Voice over Internet Protocol (VoIP) is an emerging technology that enables voice communication over the Internet using the Internet Protocol (IP). By integrating multiple modes of communication using this single infrastructure, it is possible to provide a variety of new services. The emergence of the Session Initiation Protocol (SIP) with SDP (Session Description Protocol), RTP (Real Time Transport Protocol) and UDP it is possible to develop PSTN features in IP network. SIP promises simple and efficient handling of multimedia sessions among multiple users.

Objective of Project VoIP at NirmaLabs was to create campus wide VoIP Architecture. In this VoIP architecture main component to be developed was Embedded VoIP Phone. SIP based VoIP software architecture was designed and implemented for Embedded VoIP Phone. SIP UA (User Agent), using SIP, Media and General Library, was implemented. This SIP-based VoIP system provides a wide range of services such as multiple audio encoding formats, and complex functionalities such as call muting and call-hold facilities, and personal mobility support. Also designed and implemented DAC (Digital to Analog Converter) and ADC (Analog to Digital Converter) modules. To make SIP UA compliant with target platform TS-7250, DAC and ADC modules are integrated in SIP UA. Miscellaneous problems generated during different phases of Project VoIP were solved.

# Table of Contents

# List of Figures

**Number** **Page**

# List of Tables

**Number** **Page**

# Acronyms

| | |
|---|---|
| ADC | Analog to Digital Converter |
| ALSA | Advanced Linux Sound Architecture |
| ATM | Asynchronous Transfer Mode |
| BSD | Berkeley Socket Distribution |
| CSRC | Contributing Source |
| DAC | Digital to Analog Converter |
| DAS | Data Acquisition System |
| DIO | Digital Input Output |
| DMA | Dynamic Memory Access |
| DNS | Domain Name Service |
| DTMF | Dual Tone Multi Frequency |
| FEC | Forward Error Correction |
| GSM | Global System for Mobile |
| GNU | GNU Not Unix |
| IETF | International Engineering Task Force |
| ITU | International Telecom Union |
| LCD | Liquid Crystal Display |
| Mbone | Multicast backbone |
| MMU | Memory Management Unit |
| MOSI | Master Out Slave Input |
| MISO | Master In Slave Out |
| NAT | Network Address Translation |
| NERF | Nirma Education Research Foundation |
| OSS | Open Sound System |
| PBX | Private Branch Exchange |
| PCM | Pulse Code Modulation |
| PDA | Personal Digital Assistant |
| PLC | Packet Loss Concealment |
| POST | Power-On Self Test |
| PSTN | Public Switch Telephone Network |

| | |
|---|---|
| RAM | Random Access Memory |
| RFC | Request For Comments |
| RTOS | Real Time Operating System |
| RTP | Real time Transport Protocol |
| RTSP | Real Time Streaming Protocol |
| SAP | Session Announcement Protocol |
| SCRC | Synchronous Source |
| SDP | Session Description Protocol |
| SDRAM | Synchronous Dynamic Random Access Memory |
| SIP | Session Initiation Protocol |
| SIP UA | SIP User Agent |
| SNMP | Simple Network Management Protocol |
| SPI | Serial Peripheral Interface |
| TFTP | Trivial File Transfer Protocol |
| ToIP | Text over Internet Protocol |
| TU | Transaction User |
| UDP | User Datagram Protocol |
| Wi-Fi | Wireless Fidelity |
| WLAN | Wireless Local Area Network |
| VAD | Voice Activity Detector |
| VoIP | Voice over Internet Protocol |

# *Acknowledgments*

The project has been long one and many people got associated with it. The work would have not matured without the constant feedback, suggestions and constructive criticism of many people who took some time out of their busy schedule to guide this project; it has been a pleasure knowing them and inspiration learning from them.

This project reflects the help and advice of many people. I would like to thank Dr. Madhu Mehta, Chief Architect, NirmaLabs and Mr.Thyagrajan, CEO, NirmaLabs for their inspiration and advice.

Prof. Jaladhi Joshi, Project Guide is the tremendous supporter during the project period & I will always be grateful for his help and patience.

Special thanks to Mr. Madhukar Pai for showing me realities of self directed thinking & for giving technical assistance. I am very thankful to Nirav, Jatin and Dhaval for providing constant support during project.

Very special thanks to my Course Coordinator Dr. S.N.Pradhan, who has supported and guided me in each and every step that has moved. He provided a very good technical help during project period.

Chapter 1

# Introduction

## 1.1 VoIP

Since the telephone was invented in the late 1800s, telephone communication has not changed substantially. Of course, new technologies like digital circuits, DTMF and caller ID have improved on this invention, but the basic functionality is still the same. Over the years, service provides made a number of changes "behind the scenes" to improve on the kinds and types of services offered to subscribers, including toll-free numbers, call-return, call forwarding, etc. By and large, users do not know how those services work, but they did know two things: the same old telephone is used and the service provider charges for each and every little incremental service addition introduced.

In the 1990s, a number of individuals in research environments, both in educational and corporate institutions, took a serious interest in carrying voice and video over IP networks, especially corporate intranets and the Internet. This technology is commonly referred to today as VoIP(Voice over Internet Protocol) and is, in simple terms, the process of breaking up audio or video into small chunks, transmitting those chunks over an IP network, and reassembling those chunks at the far end so that two people can communicate using audio and video.

### 1.1.1 Why is VoIP Important?

One of the most important things to point out is that VoIP is not limited to voice communication. In fact, a number of efforts have been made to change this popular marketing term to better reflect the fact that VoIP means voice, video, and data conferencing. All such attempts have failed up to this point, but do understand that video telephony and real-time text communication (ToIP), for example, is definitely within the scope of the VoIP.

VoIP is important because, for the first time in more than 100 years, there is an opportunity to bring about significant change in the way that people communicate. In addition to being able to use the telephones we have today to communicate in real-time, we also have the possibility of using pure IP-based phones, including desktop and wireless phones. We also have the ability to use videophones, much like those seen in science fiction movies. Rather than calling home to talk to the family, a person can call home to see the family.

One of the more interesting aspects of VoIP is that we also have the ability to integrate a stand-alone telephone or videophone with the personal computer. One can use a computer entirely for voice and video communications (softphones), use a telephone for voice and the computer for video, or can simply use the computer in conjunction with a separate voice/video phone to provide data conferencing functions, like application sharing, electronic whiteboarding, and text chat.

VoIP allows something else: the ability to use a single high-speed Internet connection for all voice, video, and data communications. This idea is commonly referred to as convergence and is one of the primary drivers for corporate interest in the technology. The benefit of convergence should be fairly obvious: by using a single data network for all communications, it is possible to reduce the overall maintenance and deployment costs. The benefit for both home and corporate customers is that they now have the opportunity to choose from a much larger selection of service providers to provide voice and video communication services. Since the VoIP service provider can be located virtually anywhere in the world, a person with Internet access is no longer geographically restricted in their selection of service providers and is certainly not bound to their Internet access provider.

In short, VoIP enables people to communicate in more ways and with more choices.

### 1.1.2 How Does VoIP Work?

Many people have used a computer and a microphone to record a human voice or other sounds. The process involves sampling the sound that is heard by the computer at a very high rate (at least 8,000 times per second or more) and storing those "samples" in memory or in a file on the computer. Each sample of sounds is just a very tiny bit of the person's voice or other sound recorded by the computer. The computer has the wherewithal to take all of those samples and play them, so that the listener can hear what was recorded.

VoIP is based on the same idea, but the difference is that the audio samples are not stored locally. Instead, they are sent over the IP network to another computer and played there.

Of course, there is much more required in order to make VoIP work. When recording the sound samples, the computer might compress those sounds so that they require less space and will certainly record only a limited frequency range. There are a number of ways to compress audio, the algorithm for which is referred to as a "compressor/de-compressor", or simply CODEC. Many CODECs exist for a variety of applications (e.g., movies and sound recordings) and, for VoIP, the CODECs are optimized for compressing voice, which significantly reduce the bandwidth used compared to an uncompressed audio stream. Speech CODECs are optimized to improve spoken words at the expense of sounds outside the frequency range of human speech. Recorded music and other sounds do not generally sound very good when passed through a speech CODEC, but that is perfectly OK for the task at hand.

Once the sound is recorded by the computer and compressed into very small samples, the samples are collected together into larger chunks and placed into data packets for transmission over the IP network. This process is referred to packetization. Generally, a single IP packet will contain 10 or more milliseconds of audio, with 20 or 30 milliseconds being most common.

Vint Cerf, who is often called the Father of the Internet, once explained packets in a way that is very easy to understand. Paraphrasing his description, he suggested to think of a packet as a postcards sent via postal mail. A postcard contains just a limited amount of information. To deliver a very long message, one must send a lot of postcards. Of course, the post office might lose one or more postcards. One also has to assemble the received postcards in order, so some kind of mechanism must be used to properly order to postcards, such as placing a sequence number on the bottom right corner. One can think of data packets in an IP network as postcards.

Just like postcards sent via the postal system, some IP data packets get lost and the CODECs must compensate for lost packets by "filling in the gaps" with audio that is acceptable to the human ear. This process is referred to as packet-loss concealment (PLC). In some cases, packets are sent multiple times in order to overcome packet loss. This method is called, appropriately enough, redundancy. Another method to address packet loss, known as forward-error correction (FEC), is to include some information from previously transmitted packets in subsequent packets. By performing mathematical operations in a particular FEC scheme, it is possible to reconstruct a lost packet from information bits in neighboring packets.

Packets are also sometimes delayed, just as with the postcards sent through the post office. This is particularly problematic for VoIP systems, as delays in delivering a voice packet means the information is too old to play. Such old packets are simply discarded, just as if the packet was never received. This is acceptable, as the same PLC algorithms can smooth the audio to provide good audio quality.

Computers generally measure the packet delay and expect the delay to remain relatively constant, though delay can increase and decrease during the course of a conversation. Variation in delay (called jitter) is the most frustrating for IP devices. Delay, itself, just means it takes longer for the recorded voice spoken by the first person to be heard by the user on the far end. In general, good networks have an end-to-end delay of less than 100ms, though delay up to 400ms is considered acceptable (especially when using satellite systems). Jitter can result in choppy voice or temporary glitches, so VoIP devices must implement jitter buffer algorithms to compensate for jitter. Essentially, this means that a certain number of packets are queued before play-out and the queue length

may be increased or decreased over time to reduce the number of discarded, late-arriving packets or to reduce "mouth to ear" delay. Such "adaptive jitter buffer" schemes are also used by CD recorders and other types of devices that deal with variable delay.

Video works in much the same way as voice. Video information received through a camera is broken into small pieces, compressed with a CODEC, placed into small packets, and transmitted over the IP network. This is one reason why VoIP is promising as a new technology: adding video or other media is relatively simple. Of course, there are certain issues that must be considered that are unique to video (e.g., frame refresh and much higher bandwidth requirements), but the basic principles of VoIP equally apply to video telephony.

Of course there is much more to VoIP than just sending the audio/video packets over the Internet. There must also be an agreed protocol for how computers find each other and how information is exchanged in order to allow packets to ultimately flow between the communicating devices. There must also be an agreed format (called payload format) for the contents of the media packets. Description of the popular VoIP protocols is given in the following chapters.

## 1.2 Project VoIP @ NirmaLabs

The work on VoIP is far from over, though. Many experts in the field are still actively working to make improvements on the technology. Over time, it should prove to be an adequate replace to the current PSTN used around the world today and is already an adequate replacement in limited deployments, such as enterprise environments where network quality-of-service (QoS) is well-managed. It also works extremely well for residential users who are willing to sacrifice a little voice quality for significantly lower telephone costs.

As India is globalizing, Indians are playing an ever increasingly important role in shaping/running the world. Today, Broadband Internet connection is growing faster in India. To maximize the utilization of this network capacity new services and its provider should be emerge. NirmaLabs, the most recent initiative of Nirma Education and Research Foundation (NERF), is an incubator to spawn High Tech Knowledge based

Wealth Generation ventures. Considering the VoIP venture as an emerging and lucrative, The Project VoIP was started in Sept-2005 at NirmaLabs with team of 8 students under the guidance of Assistant Prof. Jaladhi Joshi and NirmaLabs members.

### 1.2.1 Objective of Project VoIP @ NirmaLabs

To develop Low cost, full functional VoIP Architecture, includes Embedded VoIP Phones, Gateway & IP PBX, used to replace traditional organization wide PSTN based switched PBX and Centrex system.

Chapter 2

# Problem Description

VOIP is the technology of the century and can be a huge benefit in many ways. For beginners VOIP stands for Voice over IP or Voice over Internet Protocol and consists of a number of processes that interconnect and then convert a traditional voice signal into a stream of packets that are distributed over a packet network and then back.

To achieve our aim, we designed a VoIP architecture which provides SIP based VoIP services in IP networks. VoIP architecture also accommodates PSTN calls bridged into VoIP networks through gateways. Even though our architecture is based on SIP, the same concepts and design principles can also be applied to other VoIP networks, such as H.323-based VoIP networks. Figure 1 shows VoIP Architecture design. We divided the responsibility of development of each component between our team members.

## 2.1 VoIP Architecture

Basic components of VoIP Architecture are: IP PBX, Gateway, SIP Server and IP Phone (Soft/Embedded).

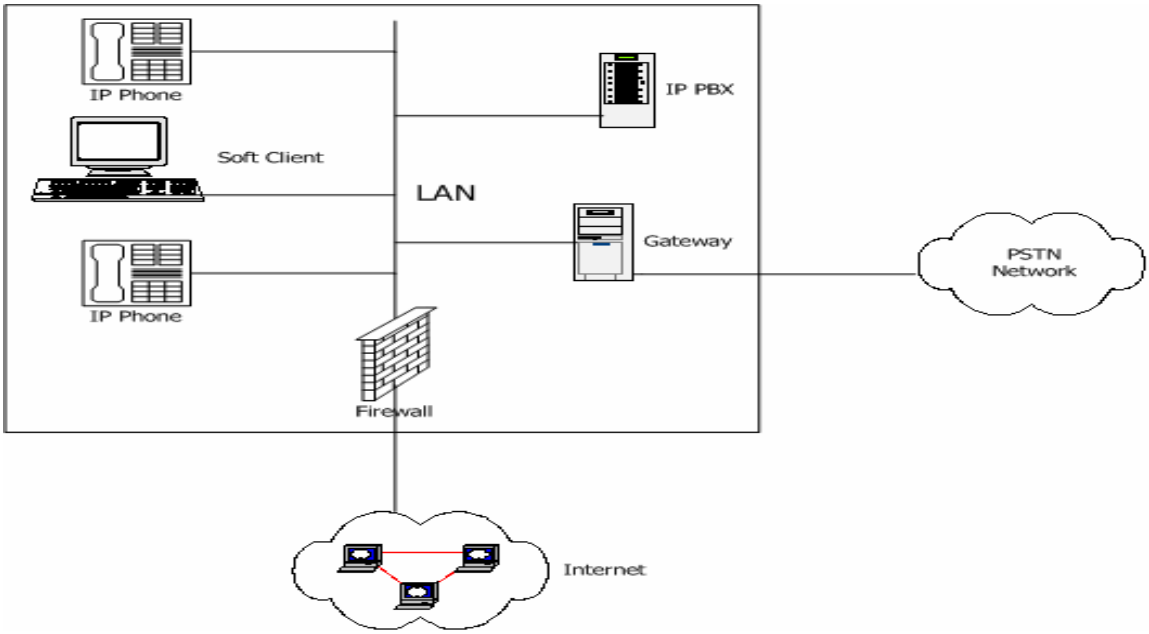



**Figure 1: VoIP Architecture**

### 2.1.1 IP PBX

IP-PBX (Internet Protocol-Private Branch Exchange) serves as a private telephone network in an organization. IP PBX allows multiple users in an organization to be connected to each other and also allows them to be connected to external phone lines. PBX has been around for a long time and users in the network can be given a three of four digit extension that they can dial to speak to each other. When users in a network need to call outside telephone lines, they typically dial a predefined number (e.g. 9 or #) that connects the user to the public telephone system or to the PBX operator.[14]

PBX can be defined as a communication device that initiates communication between two extensions and keeps the connection open till one or both parties disconnect the call. The PBX system also provides call detail recording and other relevant information for each extension.

IP PBX systems have many features like call transfer, voice mail, direct dialing, call forwarding service on busy or absence, music on hold, pre-recorded welcome messages, re-recorded instructions with options where the caller can dial different numbers to reach different services, automatic ring back, call distribution, waiting, pickup, park, conferencing, greetings, shared message box, automatic directory services, etc.

To develop IP PBX in Project VoIP, we decided to use Asterisk.

*Asterisk*

Asterisk is a complete PBX in software. It runs on Linux, BSD and Mac OS and provides all of the features you would expect from a PBX and more. Asterisk does voice over IP in many protocols, and can interoperate with almost all standards-based telephony equipment using relatively inexpensive hardware.

### 2.1.2 Gateway

The fundamental role of gateways in networks is to perform protocol conversion, allowing intercommunication between two different type of networks, such as between IP and PSTN. In particular, in voice over IP, a gateway converts an analog voice stream or a digitized version of voice, into IP packets. For example, a gateway can convert an analog stream from a PSTN line to a digital packet stream (SIP or H.323 protocol) and send it to the IP network.

Single line PSTN to IP gateway is one of the component our VoIP Architecture. Development of gateway using Digium X100P modem and Asterisk software is completed.

### 2.1.3 IP Phone

A VoIP phone is a telephone device that transports voice over a network using data packets instead of circuit switched connections over voice only networks. IP Telephony refers to the transfer of voice over the Internet Protocol (IP) of the TCP/IP protocol suite. Other Voice over Packet (VOP) standards exist for Frame Relay and ATM networks but most of people use the terms Voice over IP (VoIP) or “IP Telephony” to mean voice over any packet network.

IP Telephones originally existed in the form of client software running on multimedia PCs for low-cost PC-to-PC communications over the Internet know as VoIP Soft Phones. But now a day’s different types of VoIP phones are available.

- Hard Phones

  Self contained phones with an ethernet port.

- Dialup Hard Phones

  Self contained phones with a built-in modem.
- WLAN or Wi-Fi Phones

  Self contained phones with a built-in Wi-Fi transceiver.
- Combined WiFi/WLAN and GSM Phones

  Self contained phones with a built-in WiFi and GSM transceiver.
- Soft Phones

  IP Telephony in Software

Hard VoIP Telephone has several advantages over multimedia PCs with client software:

- Lower latency due to an embedded system implementation
- Familiar user paradigm of using a phone vs. a “PC-enabled phone”
- Greater reliability
- Lower station cost where a PC is also not required, e.g., conference room, production floor, etc.

So, our main focus was on developing Embedded VoIP Phone.

## 2.2 Design of Embedded VoIP Phone

Embedded System is a combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a dedicated function. VoIP Hard phone is a type of Embedded System with hardware software coordination.

### 2.2.1 Hardware Architecture:-

Figure 2 shows a block diagram of Embedded VoIP telephone hardware architecture. Embedded VoIP Hardware architecture consists of the User Interface, Voice Interface, Network Interface, and Processor Core and associated logic.

The Processor Core performs the voice processing, call processing, protocol processing, and network management software functions of the telephone. As shown in figure, it consists of a microprocessor. To ensure software upgradeability the telephone will make use of flash memory.

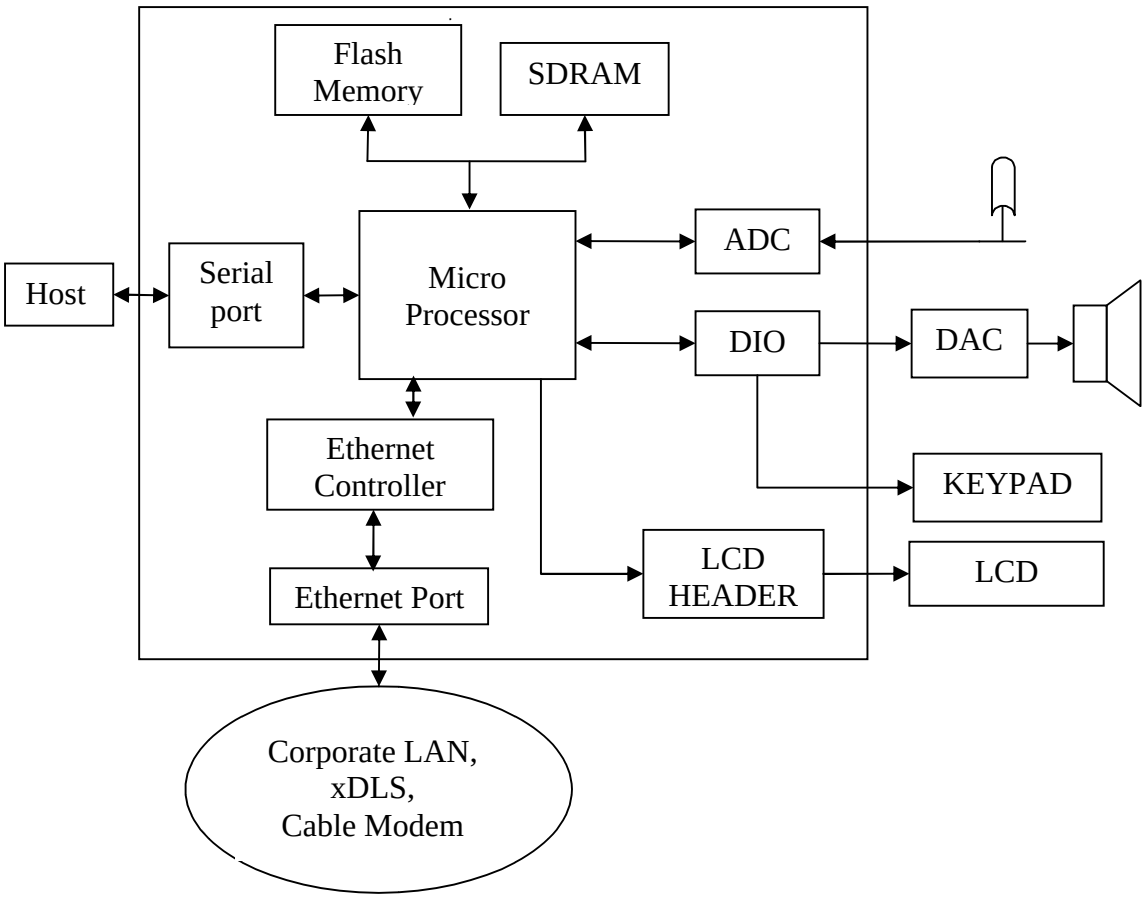



**Figure 2: Embedded VoIP Phone Hardware Architecture**

The User Interface provides the traditional user interface functions of a telephone. It consists of a keypad for dialing numbers (0-9, *, #) and an audible indicator for announcing incoming calls to the user. On more sophisticated telephone sets, additional keys are provided for features such as mute, redial, hold, transfer, conferencing, etc. An LCD is also typically provided for displaying user prompts, number dialed, Caller ID information for incoming calls, etc.

The Voice Interface provides the conversion of analog voice into digital samples. Speech signals from the microphone are sampled at a rate of 8 KHz to create a digitized 64kbps data stream. Similarly, the processor passes a 64kbps data stream in the return path to the speaker to convert digital samples back into speech.

The Network Interface allows transmission and reception of voice packets from/to the telephone. For corporate LANs this is most often either 10BaseT or 100BaseT Ethernet running TCP/IP protocols.

### 2.2.2 Software Architecture:-

Figure 3 shows the software architecture of an IP Telephone based on the SIP standard for VoIP. The software architecture consists of the following major subsystems: User Interface, Digital Signaling Processing, Telephony Signaling, Network Interface Protocols, Network Management Agent, and System Services.

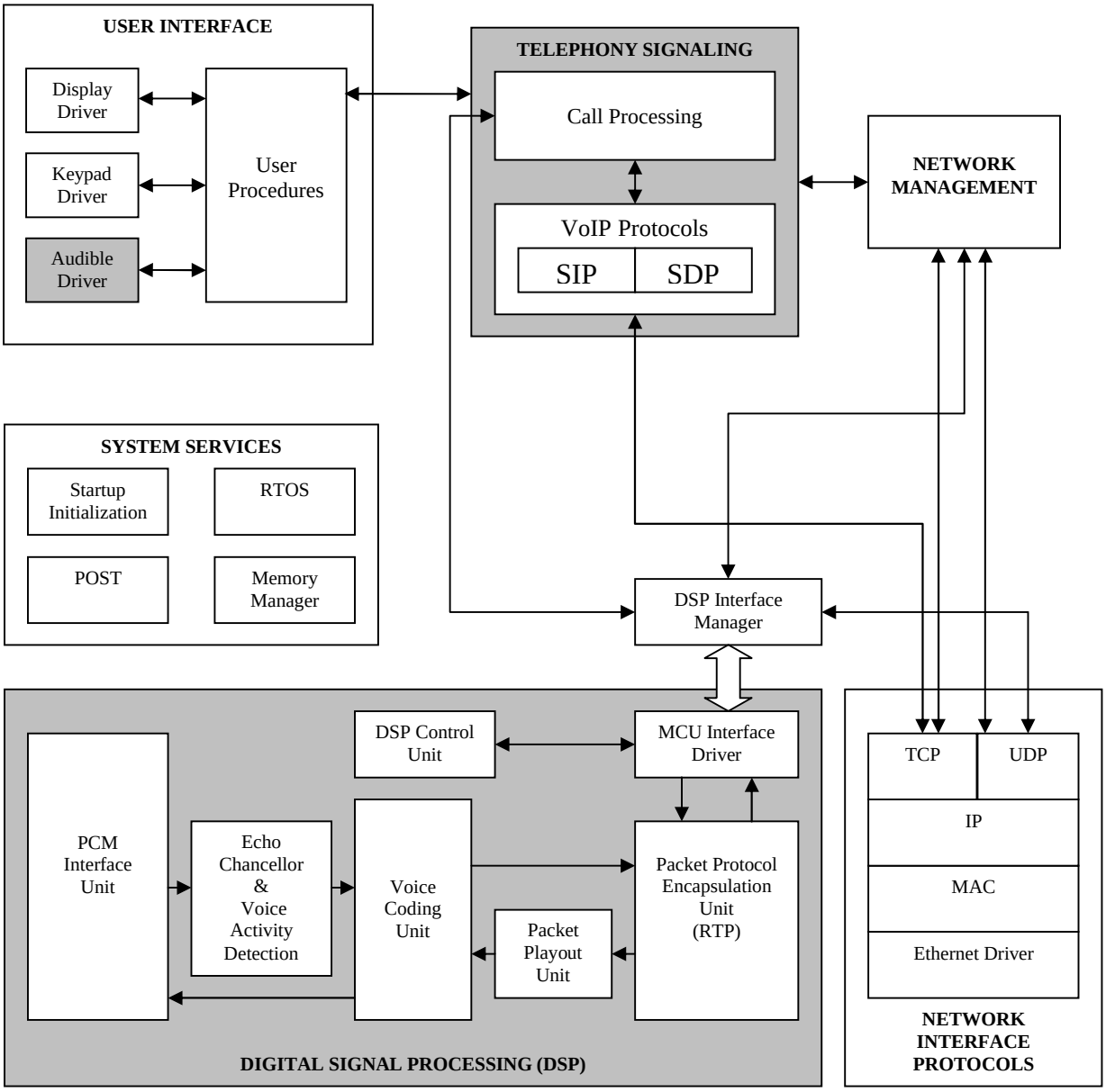



**Figure 3: Embedded VoIP Phone Software Architecture**

**User Interface**

The User Interface subsystem provides the software components that handle the interface to the user of the IP Telephone. It included display driver, keypad driver and audible drivers.

**Digital Signal Processing**

The Digital Signal Processing software is composed of the following software modules:

*PCM Interface Unit*

Receives PCM samples from the analog interface and forwards them to the appropriate DSP software module for processing. It also forwards processed PCM samples to the analog interface.

*Echo Canceller Unit*

Performs ITU G.165 (line echo cancellation) & G.168 (network echo cancellation) compliant echo cancellation on sampled, full-duplex voice port signals. Echo in a telephone network is caused by signal reflections generated by the hybrid circuit that converts between a 4-wire circuit (a separate transmit and receive pair) and a 2-wire circuit (a single transmit and receive pair). These reflections of the speaker's voice are heard in the speaker's ear. Echo is present even in a conventional circuit switched telephone network. However, it is acceptable because the round trip delays through the network are smaller than 50 msec. and the echo is masked by the normal side tone every telephone generates. Echo becomes a problem in Voice over Packet networks because the round trip delay through the network is almost always greater than 50 msec. Thus, echo cancellation techniques are required. ITU standard G.168 defines performance requirements that are currently required for echo cancellers. Echo is cancelled toward the packet network from the telephone network. The echo canceller compares the voice data received from the packet network with voice data being transmitted to the packet network. The echo from the telephone network hybrid is removed by a digital filter on the transmit path into the packet network.[14]

*Voice Activity Detector (VAD)*

Detects voice activity and activates or deactivates the transmission of packets in order to optimize bandwidth. When activity is not detected, the encoder output will not be transported across the network. This software also measures Idle Noise characteristics of the interface and reports this information to the Packet Voice Protocol for periodic forwarding to the remote IP Telephone or gateway. Idle noise is reproduced by the remote end when there is no voice activity so that the remote user does not feel that the line went "dead." [14]

*Voice Codec Unit*

Performs packetization of the 64 kbps data stream received from the user. Various compression algorithms exist which have different performance characteristics: G.711 PCM which operates at 64 kbps (no compression), G.726 ADPCM which operates at 16, 24, 32 and 40 kbps, G.723.1 which operates at 5.3 or 6.3 kbps and G.729 which operates at 8 kbps. Typically, voice algorithms that perform greater compression require much more processing power. It should be noted that high fidelity audio quality compression algorithms can also be used since an IP Telephone is not subject to the 4 kHz bandwidth restrictions found in the PSTN. This would provide better sounding audio than PCM and allow music to be faithfully reproduced.[14]

*Packet Playout Unit*

Performs compensation for network delay, network jitter and dropped packets. Many proprietary techniques are used to address these problems since there are currently no standards in place for packet playout.

*Packet Protocol Encapsulation Unit*

Performs encapsulation of the packet voice data destined for the network interface. For VoIP this encapsulation is per the Real-time Transport Protocol (RTP) which runs directly on top of UDP.

*DSP Control Unit*

Coordinates the exchange of monitor and control information between the Voice Processing Module and Telephony Signaling and Network Management modules. The

information exchanged includes software down line load, configuration data, signaling information and status reporting.

**Network Management**

The Network Management subsystem supports remote administration of the IP Telephone by a Network Management System.

**Telephony Signaling Subsystem**

The Telephony Signaling Subsystem performs the functions for establishing, maintaining and terminating a call. The Telephony Signaling consists of the following software modules:

*Call Processing*

Performs the state machine processing for call establishment, call maintenance and call tear down.

*SIP/SDP*

Session Initiation Protocol (SIP) is the IETF's standard for establishing VOIP connection. SIP is an application layer control protocol for creating, modifying and terminating sessions with one or more participants. The architecture of SIP is similar to that of HTTP (client-server protocol). Requests are generated by the client and sent to the server. The server processes the requests and then sends a response to the client. A request and the responses for that request make a transaction.

SDP is intended for describing multimedia sessions for the purposes of session announcement, session invitation, and other forms of multimedia session initiation. SDP is used from VOIP signaling protocols like SIP, H.323 and some minor VOIP protocols to transfer media setup information about a multi media client from one end point to other.

**Network Interface Protocols**

The Network Interface Protocols support communications over the Local Area Network (LAN) and consists of the following software modules:

*TCP*

The Transport Control Protocol (TCP) provides reliable transport of data including retransmission and flow control. It is used for web queries, and call signaling functions.

*UDP*

The User Datagram Protocol (UDP) provides efficient but unreliable transport of data. It is used for the transport of real-time voice data since retransmission of real-time data would add too much delay to the voice conversation and be unacceptable. UDP is also used for SNMP and TFTP network management traffic.

*IP*

The Internet Protocol (IP) provides a standard encapsulation of data for transmission over the network. It contains a source and destination address used for routing.

*MAC/ARP*

Performs Media Access Control (MAC) management functions and handles Address Resolution Protocol (ARP) for the device.

*Ethernet Driver*

Configures and controls the ethernet controller hardware, including setting up DMA operations.

**System Services**

System Services consists of the following software modules:

*Startup/Initialization*

Provides startup and initialization of the hardware and software components of the IP Telephone.

*POST*

Provides Power-On Self-Test (POST) functions of the IP Telephone.

*RTOS*

The Real-Time Operating System (RTOS) provides functions such as task management, memory management, and task synchronization.

## 2.3 Statement of Problem

The main objective of Project VoIP at NirmaLabs was to develop Embedded VoIP Phone. In this project my responsibilities were:

- To Design and implement SIP-base VoIP User Agent (UA) for target system with basic features and small size, suitable for embedded phone.
- To develop ADC (Analog to Digital Converter)/DAC (Digital to Analog Converter) driver modules and integrate with SIP User Agent (SIP UA).

## 2.4 Task Enlisted

- To study and design SIP-base VoIP Architecture
- To configure host and target platform.
- To design and implement SIP UA for x86 system.
- To analyze performance of SIP UA between two x86 Systems.
- To develop ADC/DAC driver.
- To integrate ADC/DAC modules with SIP UA.
- To build SIP UA for target architecture
- To port SIP UA on target development board system.
- To analyze performance between development board and PC within Intranet.

Chapter 3

# Literature Study

## 3.1 SIP based VoIP Architecture

SIP stands for Session Initiation Protocol. It is an application-layer control protocol which has been developed and designed within the IETF. The protocol has been designed with easy implementation, good scalability, and flexibility in mind.

The specification is available in form of several RFCs, the most important one is RFC3261 which contains the core protocol specification. The protocol is used for creating, modifying, and terminating sessions with one or more participants. By sessions we understand a set of senders and receivers that communicate and the state kept in those senders and receivers during the communication.[20] Examples of a session can include Internet telephone calls, distribution of multimedia, multimedia conferences, distributed computer games, etc.

SIP is not the only protocol that the communicating devices will need. It is not meant to be a general purpose protocol. Purpose of SIP is just to make the communication possible; the communication itself must be achieved by another means (and possibly another protocol). Two protocols that are most often used along with SIP are RTP and

SDP. RTP protocol is used to carry the real-time multimedia data (including audio, video, and text), the protocol makes it possible to encode and split the data into packets and transport such packets over the Internet. Another important protocol is SDP, which is used to describe and encode capabilities of session participants. Such a description is then used to negotiate the characteristics of the session so that all the devices can participate (that includes, for example, negotiation of codec used to encode media so all the participants will be able to decode it, negotiation of transport protocol used and so on).

SIP has been designed in conformance with the Internet model. It is an end-to-end oriented signaling protocol which means, that all the logic is stored in end devices (except routing of SIP messages). State is also stored in end-devices only, there is no single point of failure and networks designed this way scale well. The price that we have to pay for the distributiveness and scalability is higher message overhead, caused by the messages being sent end-to-end.

It is worth of mentioning that the end-to-end concept of SIP is a significant divergence from regular PSTN (Public Switched Telephone Network) where all the state and logic is stored in the network and end devices (telephones) are very primitive. Aim of SIP is to provide the same functionality that the traditional PSTN have, but the end-to-end design makes SIP networks much more powerful and open to the implementation of new services that can be hardly implemented in the traditional PSTN.

SIP is based on HTTP protocol. The HTTP protocol inherited format of message headers from RFC822. HTTP is probably the most successful and widely used protocol in the Internet. It tries to combine the best of the both. In fact, HTTP can be classified as a signaling protocol too, because user agents use the protocol to tell a HTTP server in which documents they are interested in. SIP is used to carry the description of session parameters; the description is encoded into a document using SDP. [9] Both protocols (HTTP and SIP) have inherited encoding of message headers from RFC822. The encoding has proven to be robust and flexible over the years.

### 3.1.1 SIP URI

SIP entities are identified using SIP URI (Uniform Resource Identifier). A SIP URI has form of sip:username@domain, for instance, sip:vipul@nirmalabs.org. As we

can see, SIP URI consists of username part and domain name part delimited by @ (at) character. SIP URIs are similar to e-mail addresses, it is, for instance, possible to use the same URI for e-mail and SIP communication, such URIs are easy to remember.

### 3.1.2 SIP Network Elements:-

Although in the simplest configuration it is possible to use just two user agents that send SIP messages directly to each other, a typical SIP network will contain more than one type of SIP elements. Basic SIP elements are user agents, proxies, registrars, and redirect servers.

Note that the elements are often only logical entities. It is often profitable to co-locate them together, for instance, to increase the speed of processing, but that depends on a particular implementation and configuration.

Internet end points that use SIP to find each other and to negotiate a session characteristics are called user agents. User agents usually, but not necessarily, reside on a user's computer in form of an application--this is currently the most widely used approach, but user agents can be also cellular phones, PSTN gateways, PDAs, automated IVR systems and so on.

User agents are often referred to as User Agent Server (UAS) and User Agent Client (UAC). UAS and UAC are logical entities only; each user agent contains a UAC and UAS. UAC is the part of the user agent that sends requests and receives responses. UAS is the part of the user agent that receives requests and sends responses.[1]

Because a user agent contains both UAC and UAS, we often say that a user agent behaves like a UAC or UAS. For instance, caller's user agent behaves like UAC when it sends an INVITE requests and receives responses to the request. Callee's user agent behaves like a UAS when it receives the INVITE and sends responses.

But this situation changes when the callee decides to send a BYE and terminate the session. In this case the callee's user agent (sending BYE) behaves like UAC and the caller's user agent behaves like UAS.

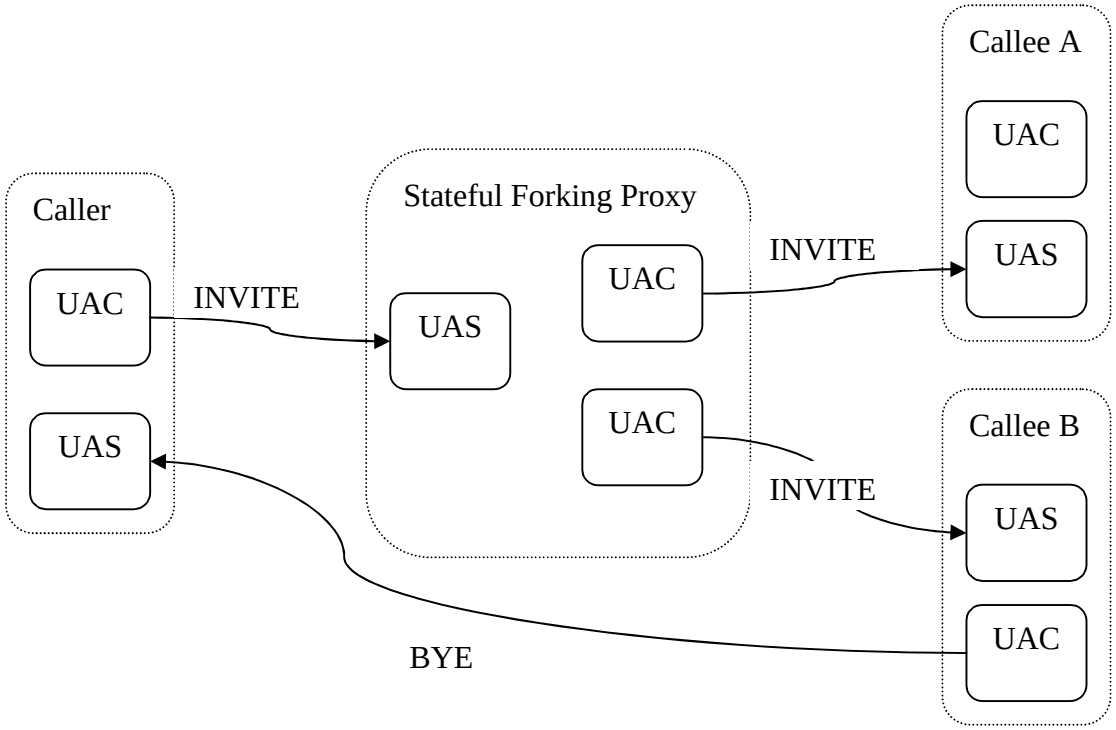



**Figure 4: SIP Network Elements**

Figure 4 shows three user agents and one stateful forking proxy. Each user agent contains UAC and UAS. The part of the proxy that receives the INVITE from the caller in fact acts as a UAS. When forwarding the request statefully the proxy creates two UACs, each of them is responsible for one branch.

In this example callee B picked up and later when he wants to tear down the call it sends a BYE. At this time the user agent that was previously UAS becomes a UAC and vice versa.

### 3.1.3 Proxy Servers

In addition to that SIP allows creation of an infrastructure of network hosts called proxy servers. User agents can send messages to a proxy server. Proxy servers are very important entities in the SIP infrastructure. They perform routing of a session invitations according to invitee's current location, authentication, accounting and many other important functions.

The most important task of a proxy server is to route session invitations "closer" to callee. The session invitation will usually traverse a set of proxies until it finds one which knows the actual location of the callee. Such a proxy will forward the session invitation directly to the callee and the callee will then accept or decline the session invitation.

There are two basic types of SIP proxy servers--stateless and stateful.

*Stateless Servers*

Stateless servers are simple message forwarders. They forward messages independently of each other. Although messages are usually arranged into transactions, stateless proxies do not take care of transactions.

Stateless proxies are simple, but faster than stateful proxy servers. They can be used as simple load balancers, message translators and routers. One of drawbacks of stateless proxies is that they are unable to absorb retransmissions of messages and perform more advanced routing, for instance, forking or recursive forwarding.

*Stateful Servers*

Stateful proxies are more complex. Upon reception of a request, stateful proxies create a state and keep the state until the transaction finishes. Some transactions, especially those created by INVITE, can last quite long (until callee picks up or declines the call). Because stateful proxies must maintain the state for the duration of the transactions, their performance is limited.

The ability to associate SIP messages into transactions gives stateful proxies some interesting features. Stateful proxies can perform forking, that means upon reception of a message two or more messages will be sent out. Stateful proxies can absorb retransmissions because they know, from the transaction state; if they have already received the same message (stateless proxies cannot do the check because they keep no state).

Stateful proxies can perform more complicated methods of finding a user. It is, for instance, possible to try to reach user's office phone and when he doesn't pick up then the

call is redirected to his cell phone. Stateless proxies can't do this because they have no way of knowing how the transaction targeted to the office phone finished.

Most SIP proxies today are stateful because their configuration is usually very complex. They often perform accounting; forking, some sort of NAT traversal aid and all those features require a stateful proxy.

### 3.1.4 Proxy Server Usage

A typical configuration is that each centrally administered entity (a company, for instance) has its own SIP proxy server which is used by all user agents in the entity. Let's suppose that there are two companies A and B and each of them has its own proxy server. The picture below shows how a session invitation from employee Vipul in company A will reach employee Ragin in company B.

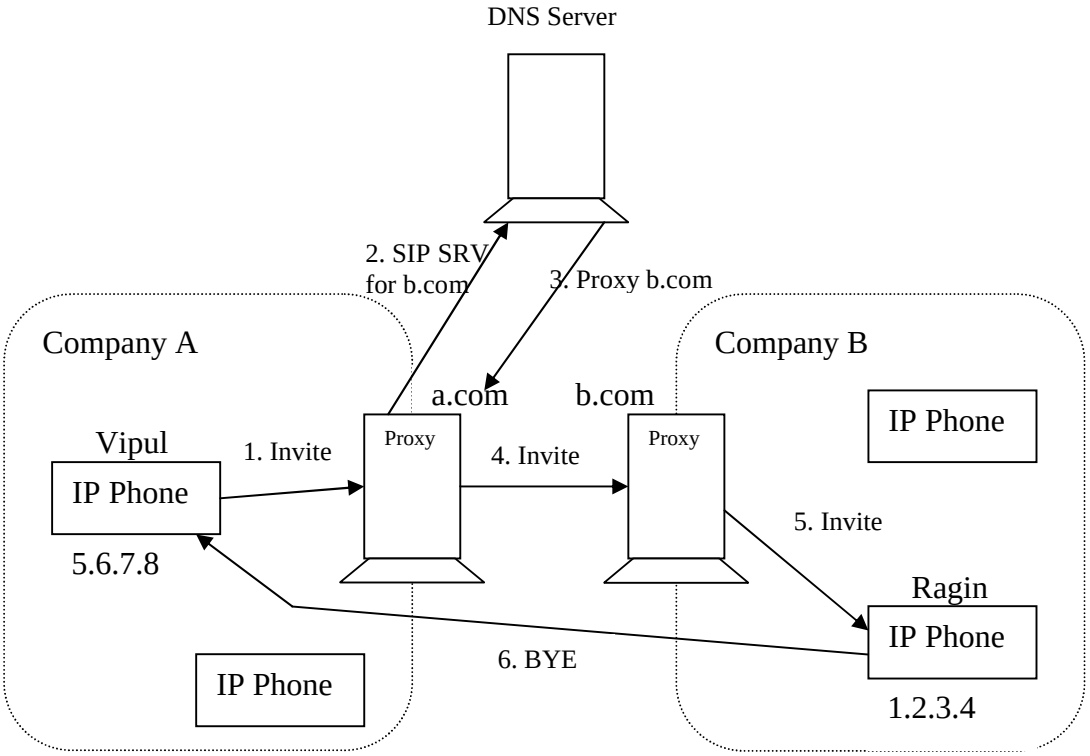



**Figure 5: SIP Proxy Server Usage**

User Vipul uses address sip:Ragin@b.com to call Ragin. Vipul's user agent doesn't know how to route the invitation itself but it is configured to send all outbound traffic to the company SIP proxy server proxy.a.com. The proxy server figures out that

user sip:Ragin@b.com is in a different company so it will look up B's SIP proxy server and send the invitation there. B's proxy server can be either pre-configured at proxy.a.com or the proxy will use DNS SRV records to find B's proxy server. The invitation reaches proxy.b.com. The proxy knows that Ragin is currently sitting in his office and is reachable through phone on his desk, which has IP address 1.2.3.4, so the proxy will send the invitation there.

*Registrar*

As mentioned in above example the SIP proxy at proxy.b.com knows current Ragin's location but hasn't mentioned yet how a proxy can learn current location of a user. Ragin's user agent (SIP phone) must register with registrar. The registrar is a special SIP entity that receives registrations from users, extracts information about their current location (IP address, port and username in this case) and stores the information into location database. Purpose of the location database is to map sip:Ragin@b.com to something like sip:Ragin@1.2.3.4:5060. The location database is then used by B's proxy server. When the proxy receives an invitation for sip:Ragin@b.com it will search the location database. It finds sip:Ragin@1.2.3.4:5060 and will send the invitation there. A registrar is very often a logical entity only. Because of their tight coupling with proxies registrars, are usually co-located with proxy servers.

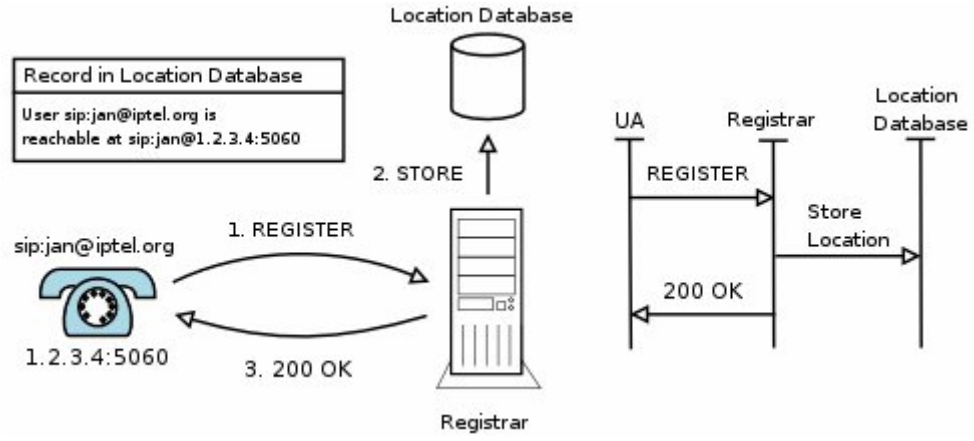



**Figure 6: SIP Registration**

The figure 6 shows a typical SIP registration. A REGISTER message containing Address of Record sip:jan@iptel.org and contact address sip:jan@1.2.3.4:5060 where

1.2.3.4 is IP address of the phone is sent to the registrar. The registrar extracts this information and stores it into the location database. If everything went well then the registrar sends a 200 OK response to the phone and the process of registration is finished.

Each registration has a limited lifespan. Expires header field or expires parameter of Contact header field determines for how long is the registration valid. The user agent must refresh the registration within the lifespan otherwise it will expire and the user will become unavailable.

*Redirect Server*

The entity that receives a request and sends back a reply containing a list of the current location of a particular user is called redirect server. A redirect server receives requests and looks up the intended recipient of the request in the location database created by a registrar. It then creates a list of current locations of the user and sends it to the request originator in a response within 3xx class.[20]

The originator of the request then extracts the list of destinations and sends another request directly to them. The following figure shows a typical redirection.

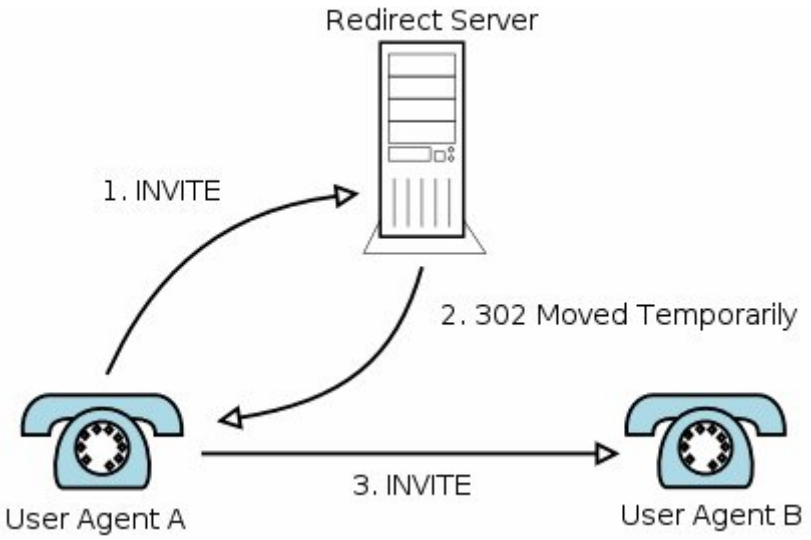



**Figure 7: Redirect Server**

### 3.1.5 SIP Messages:-

Communication using SIP (often called signaling) comprises of series of messages. Messages can be transported independently by the network. Usually they are transported in a separate UDP datagram each. Each message consists of "first line", message header, and message body. The first line identifies type of the message. There are two types of messages--requests and responses. Requests are usually used to initiate some action or inform recipient of the request of something. Replies are used to confirm that a request was received and processed and contain the status of the processing.

A typical SIP request looks like this:

```
INVITE sip:7170@nirmalabs.org SIP/2.0
 Via: SIP/2.0/UDP 195.37.77.100:5040;rport
 Max-Forwards: 10
 From: "jiri" <sip:jiri@nirmalabs.org>;tag=76ff7a07-c091-4192-84a0-d56e91fe104f
 To: <sip:jiri@bat.nirmalabs.org>
 Call-ID: d10815e0-bf17-4afa-8412-d9130a793d96@213.20.128.35
 CSeq: 2 INVITE
 Contact: <sip:213.20.128.35:9315>
 User-Agent: Windows RTC/1.0
 Proxy-Authorization: Digest username="jiri", realm="nirmalabs.org",
  algorithm="MD5", uri="sip:jiri@bat.nirmalabs.org",
  nonce="3cef753900000001771328f5ae1b8b7f0d742da1feb5753c",
  response="53fe98db10e1074b03b3e06438bda70f"
 Content-Type: application/sdp
 Content-Length: 451
 v=0
 o=jku2 0 0 IN IP4 213.20.128.35
 s=session
 c=IN IP4 213.20.128.35
 b=CT:1000
 t=0 0
 m=audio 54742 RTP/AVP 97 111 112 6 0 8 4 5 3 101
 a=rtpmap:97 red/8000
```

a=rtpmap:111 SIREN/16000
a=fmtp:111 bitrate=16000
a=rtpmap:112 G7221/16000
a=fmtp:112 bitrate=24000
a=rtpmap:6 DVI4/16000
a=rtpmap:0 PCMU/8000
a=rtpmap:4 G723/8000
a=rtpmap: 3 GSMi/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-16

The first line tells us that this is INVITE message which is used to establish a session. The URI on the first line--sip:7170@nirmalabs.org is called Request-URI and contains URI of the next hop of the message. In this case it will be host 195.37.77.100 and port 5060.

From and To header fields identify initiator (caller) and recipient (callee) of the invitation (just like in SMTP where they identify sender and recipient of a message). From header field contains a tag parameter which serves as a dialog identifier and will be described in section dialogs.

Call-ID header field is a dialog identifier and its purpose is to identify messages belonging to the same call. Such messages have the same Call-ID identifier. CSeq is used to maintain order of requests. Because requests can be sent over an unreliable transport that can re-order messages, a sequence number must be present in the messages so that recipient can identify retransmissions and out of order requests.

Contact header field contains IP address and port on which the sender is awaiting further requests sent by callee. Other header fields are not important and will be not described here.

Message header is delimited from message body by an empty line. Message body of the INVITE request contains a description of the media type accepted by the sender and encoded in SDP.

### 3.1.6 SIP Requests

I have described how an INVITE request looks like and said that the request is used to invite a callee to a session. Other important requests are:

*ACK*

This message acknowledges receipt of a final response to INVITE. Establishing of a session utilizes 3-way hand-shaking due to asymmetric nature of the invitation. It may take a while before the callee accepts or declines the call so the callee's user agent periodically retransmits a positive final response until it receives an ACK (which indicates that the caller is still there and ready to communicate).[3]

*BYE*

Bye messages are used to tear down multimedia sessions. A party wishing to tear down a session sends a BYE to the other party.

*CANCEL*

Cancel is used to cancel not yet fully established session. It is used when the callee hasn't replied with a final response yet but the caller wants to abort the call (typically when a callee doesn't respond for some time).

*REGISTER*

Purpose of REGISTER request is to let registrar know of current user's location. Information about current IP address and port on which a user can be reached is carried in REGISTER messages. Registrar extracts this information and puts it into a location database. The database can be later used by SIP proxy servers to route calls to the user. Registrations are time-limited and need to be periodically refreshed. [3]

The listed requests usually have no message body because it is not needed in most situations (but can have one). In addition to that many other request types have been defined but their description is out of the scope of this document.

### 3.1.7 SIP Responses

When a user agent or proxy server receives a request it sends a reply. Each request must be replied except ACK requests which trigger no replies.

A typical reply looks like this:

```
SIP/2.0 200 OK
 Via: SIP/2.0/UDP 192.168.1.30:5060;received=66.87.48.68
 From: sip:sip2@nirmalabs.org
 To: sip:sip2@nirmalabs.org;tag=794fe65c16edfdf45da4fc39a5d2867c.b713
 Call-ID: 2443936363@192.168.1.30
 CSeq: 63629 REGISTER
 Contact: Msip:sip2@66.87.48.68:5060;transport=udp>;q=0.00;expires=120
 Server: Sip EXpress router (0.8.11pre21xrc (i386/linux))
 Content-Length: 0
 Warning: 392 195.37.77.101:5060 "Noisy feedback tells:
   pid=5110 req_src_ip=66.87.48.68 req_src_port=5060 in_uri=sip:nirmalabs.org
   out_uri=sip:nirmalabs.org via_cnt==1"
```

As we can see, responses are very similar to the requests, except for the first line. The first line of response contains protocol version (SIP/2.0), reply code, and reason phrase.

The reply code is an integer number from 100 to 699 and indicates type of the response. There are 6 classes of responses:

1xx are provisional responses. A provisional response is response that tells to its recipient that the associated request was received but result of the processing is not known yet. Provisional responses are sent only when the processing doesn't finish immediately. The sender must stop retransmitting the request upon reception of a provisional response.

Typically proxy servers send responses with code 100 when they start processing an INVITE and user agents send responses with code 180 (Ringing) which means that the callee's phone is ringing.

2xx responses are positive final responses. A final response is the ultimate response that the originator of the request will ever receive. Therefore final responses

express result of the processing of the associated request. Final responses also terminate transactions. Responses with code from 200 to 299 are positive responses that mean that the request was processed successfully and accepted. For instance a 200 OK response is sent when a user accepts invitation to a session (INVITE request).

A UAC may receive several 200 messages to a single INVITE request. This is because a forking proxy (described later) can fork the request so it will reach several UAS and each of them will accept the invitation. In this case each response is distinguished by the tag parameter in To header field. Each response represents a distinct dialog with unambiguous dialog identifier.

3xx responses are used to redirect a caller. A redirection response gives information about the user's new location or an alternative service that the caller might use to satisfy the call. Redirection responses are usually sent by proxy servers. When a proxy receives a request and doesn't want or can't process it for any reason, it will send a redirection response to the caller and put another location into the response which the caller might want to try. It can be the location of another proxy or the current location of the callee (from the location database created by a registrar). The caller is then supposed to re-send the request to the new location. 3xx responses are final.

4xx are negative final responses. a 4xx response means that the problem is on the sender's side. The request couldn't be processed because it contains bad syntax or cannot be fulfilled at that server.

5xx means that the problem is on server's side. The request is apparently valid but the server failed to fulfill it. Clients should usually retry the request later.

6xx reply code means that the request cannot be fulfilled at any server. This response is usually sent by a server that has definitive information about a particular user. User agents usually send a 603 Decline response when the user doesn't want to participate in the session. [1]

In addition to the response class the first line also contains reason phrase. The code number is intended to be processed by machines. It is not very human-friendly but it is very easy to parse and understand by machines. The reason phrase usually contains a

human-readable message describing the result of the processing. A user agent should render the reason phrase to the user. The request to which a particular response belongs is identified using the CSeq header field. In addition to the sequence number this header field also contains method of corresponding request. In our example it was REGISTER request.

### 3.1.8 Transactions

Although SIP messages are sent independently over the network, they are usually arranged into transactions by user agents and certain types of proxy servers. Therefore SIP is said to be a transactional protocol.

A transaction is a sequence of SIP messages exchanged between SIP network elements. A transaction consists of one request and all responses to that request. That includes zero or more provisional responses and one or more final responses (remember that an INVITE might be answered by more than one final response when a proxy server forks the request).

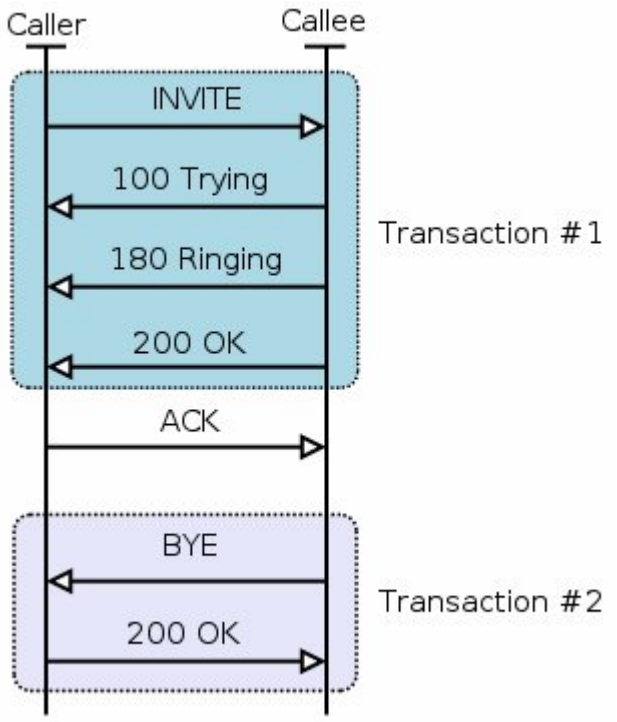



**Figure 8: SIP Call Transactions**

If a transaction was initiated by an INVITE request then the same transaction also includes ACK, but only if the final response was not a 2xx response. If the final response was a 2xx response then the ACK is not considered part of the transaction.

This is quite asymmetric behavior--ACK is part of transactions with a negative final response but is not part of transactions with positive final responses. The reason for this separation is the importance of delivery of all 200 OK messages. Not only that they establish a session, but also 200 OK can be generated by multiple entities when a proxy server forks the request and all of them must be delivered to the calling user agent. Therefore user agents take responsibility in this case and retransmit 200 OK responses until they receive an ACK.[3] Also note that only responses to INVITE are retransmitted!

SIP entities that have notion of transactions are called stateful. Such entities usually create a state associated with a transaction that is kept in the memory for the duration of the transaction. When a request or response comes, a stateful entity tries to associate the request (or response) to existing transactions. To be able to do it must extract a unique transaction identifier from the message and compare it to identifiers of all existing transactions. If such a transaction exists then it's state gets updated from the message.

In the previous SIP RFC2543 the transaction identifier was calculated as hash of all important message header fields (that included To, From, Request-URI and CSeq). This proved to be very slow and complex, during interoperability tests such transaction identifiers used to be a common source of problems.

In the new RFC3261 the way of calculating transaction identifiers was completely changed. Instead of complicated hashing of important header fields a SIP message now includes the identifier directly. Branch parameter of Via header fields contains directly the transaction identifier. This is significant simplification, but there still exist old implementations that don't support the new way of calculating of transaction identifier so even new implementations have to support the old way. They must be backwards compatible.

### 3.1.9 Dialogs

As shown what transactions are, that one transaction includes INVITE and it's responses and another transaction includes BYE and it responses when a session is being torn down. But it feels that those two transactions should be somehow related--both of them belong to the same dialog. A dialog represents a peer-to-peer SIP relationship between two user agents. A dialog persists for some time and it is very important concept for user agents.

Dialogs are identified using Call-ID, From tag, and To tag. Messages that have these three identifiers same belong to the same dialog. We have shown that CSeq header field is used to order messages; in fact it is used to order messages within a dialog. The number must be monotonically increased for each message sent within a dialog otherwise the peer will handle it as out of order request or retransmission. In fact, the CSeq number identifies a transaction within a dialog because we have said that requests and associated responses are called transaction. This means that only one transaction in each direction can be active within a dialog.[6] One could also say that a dialog is a sequence of transactions. The following figure shows which messages belong to the same dialog.

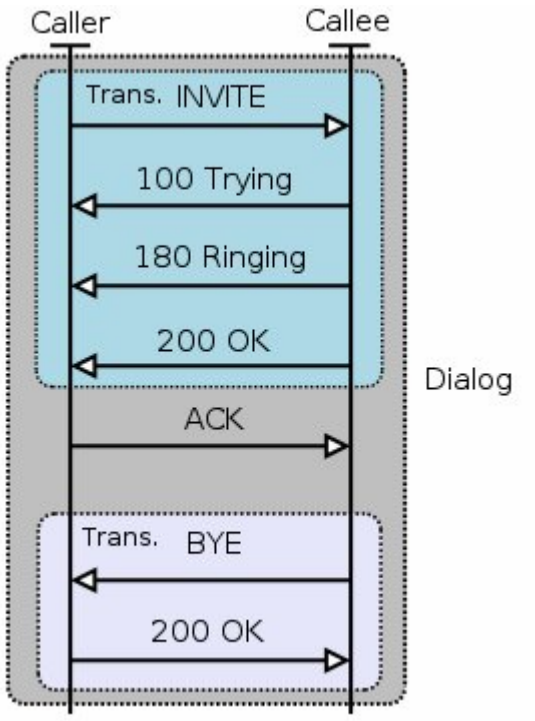



**Figure 9: SIP Dialogs**

Some messages establish a dialog and some do not. This allows to explicitly express the relationship of messages and also to send messages that are not related to other messages outside a dialog. That is easier to implement because user agent don't have to keep the dialog state.

For instance, INVITE message establishes a dialog, because it will be later followed by BYE request which will tear down the session established by the INVITE. This BYE is sent within the dialog established by the INVITE.

But if a user agent sends a MESSAGE request, such a request doesn't establish any dialog. Any subsequent messages (even MESSAGE) will be sent independently of the previous one.

*Dialog Identifiers*

Dialog identifiers consist of three parts, Call-Id, From tag, and To tag, but it is not that clear why are dialog identifiers created exactly this way and who contributes which part.

Call-ID is so called call identifier. It must be a unique string that identifies a call. A call consists of one or more dialogs. Multiple user agents may respond to a request when a proxy along the path forks the request. Each user agent that sends a 2xx establishes a separate dialog with the caller. All such dialogs are part of the same call and have the same Call-ID.

From tag is generated by the caller and it uniquely identifies the dialog in the caller's user agent. To tag is generated by a callee and it uniquely identifies, just like From tag, the dialog in the callee's user agent.

This hierarchical dialog identifier is necessary because a single call invitation can create several dialogs and caller must be able to distinguish them.

### 3.1.10 Typical SIP Scenarios

This section gives a brief overview of typical SIP scenarios that usually make up the SIP traffic.

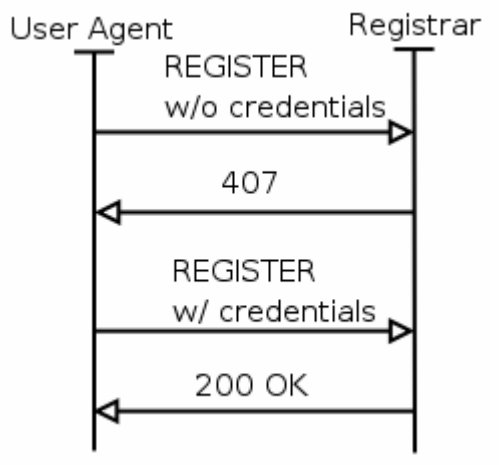



**Figure 10: SIP Registration**

*Registration*

Users must register themselves with a registrar to be reachable by other users. A registration comprises a REGISTER message followed by a 200 OK sent by registrar if the registration was successful. Registrations are usually authorized so a 407 reply can appear if the user didn't provide valid credentials. Figure 3.7 shows an example of registration.

*Session Invitation*

A session invitation consists of one INVITE request which is usually sent to a proxy. The proxy sends immediately a 100 Trying reply to stop retransmissions and forwards the request further.

All provisional responses generated by callee are sent back to the caller. See 180 Ringing respons in the call flow. The response is generated when callee's phone starts ringing.

A 200 OK is generated once the callee picks up the phone and it is retransmitted by the callee's user agent until it receives an ACK from the caller. The session is established at this point.

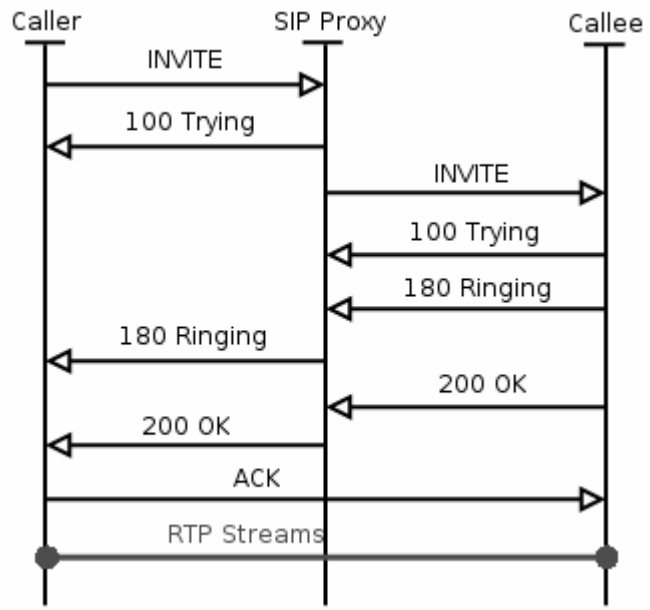



**Figure 11: SIP Session Invitations**

*Session Termination*

Session termination is accomplished by sending a BYE request within dialog established bye INVITE. BYE messages are sent directly from one user agent to the other unless a proxy on the path of the INVITE request indicated that it wishes to stay on the path by using record routing.

Party wishing to tear down a session sends a BYE request to the other party involved in the session. The other party sends a 200 OK response to confirm the BYE and the session is terminated. See BYE call-flow, left message flow.

### 3.1.11 Record Routing

All requests sent within a dialog are by default sent directly from one user agent to the other. Only requests outside a dialog traverse SIP proxies. This approach makes SIP network more scalable because only a small number of SIP messages hit the proxies.

There are certain situations in which a SIP proxy needs to stay on the path of all further messages. For instance, proxies controlling a NAT box or proxies doing accounting need to stay on the path of BYE requests.

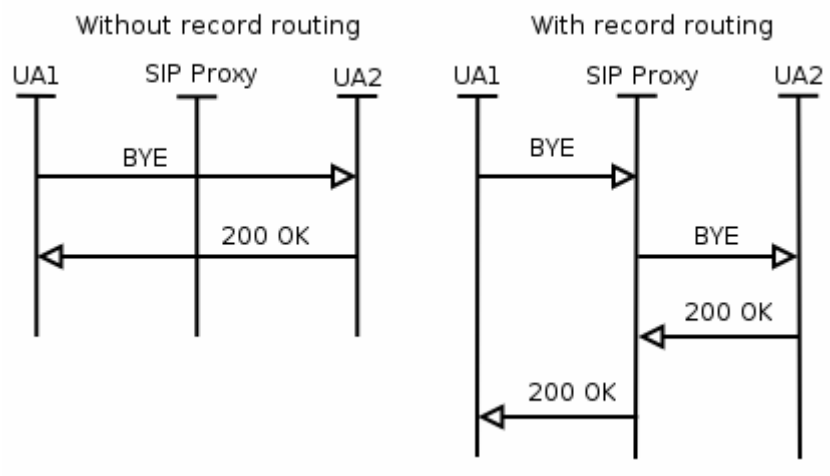



**Figure 12: Record Routing**

Mechanism by which a proxy can inform user agents that it wishes to stay on the path of all further messages is called record routing. Such a proxy would insert Record-Route header field into SIP messages which contain address of the proxy. Messages sent within a dialog will then traverse all SIP proxies that put a Record-Route header field into the message.

The recipient of the request receives a set of Record-Route header fields in the message. It must mirror all the Record-Route header fields into responses because the originator of the request also needs to know the set of proxies.

Left message flow of figure 3.9 shows how a BYE (request within dialog established by INVITE) is sent directly to the other user agent when there is no Record-Route header field in the message. Right message flow show how the situation changes when the proxy puts a Record-Route header field into the message.

*Strict versus Loose Routing*

The way how record routing works has evolved. Record routing according to RFC2543 rewrote the Request-URI. That means the Request-URI always contained URI of the next hop (which can be either next proxy server which inserted Record-Route header field or destination user agent). Because of that it was necessary to save the original Request-URI as the last Route header field. This approach is called strict routing.

Loose routing, as specified in RFC3261, works in a little bit different way. The Request-URI is no more overwritten; it always contains URI of the destination user agent. If there are any Route header field in a message, than the message is sent to the URI from the topmost Route header field. This is significant change--Request-URI doesn't necessarily contain URI to which the request will be sent. In fact, loose routing is very similar to IP source routing.

Because transit from strict routing to loose routing would break backwards compatibility and older user agents wouldn't work, it is necessary to make loose routing backwards compatible. The backwards compatibility unfortunately adds a lot of overhead and is often source of major problems.

### 3.1.12 Event Subscription and Notification

The SIP specification has been extended to support a general mechanism allowing subscription to asynchronous events. Such evens can include SIP proxy statistics changes, presence information, session changes and so on.

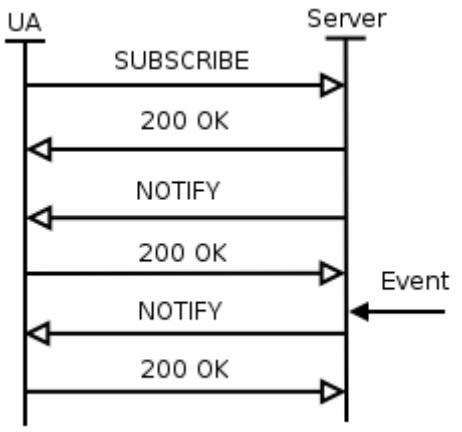



**Figure 13: Event Subscription and Notification**

The mechanism is used mainly to convey information on presence (willingness to communicate) of users. Figure 3.10 shows the basic message flow.

A user agent interested in event notification sends a SUBSCRIBE message to a SIP server. The SUBSCRIBE message establishes a dialog and is immediately replied by the server using 200 OK response. At this point the dialog is established. The server sends a NOTIFY request to the user every time the event to which the user subscribed changes. NOTIFY messages are sent within the dialog established by the SUBSCRIBE.[6]

Note that the first NOTIFY message in the call flow is sent regardless of any event that triggers notifications. Subscriptions--as well as registrations--have limited lifespan and therefore must be periodically refreshed.

### 3.1.13 Instant Messages:-

Instant messages are sent using MESSAGE request. MESSAGE requests do not establish a dialog and therefore they will always traverse the same set of proxies. This is the simplest form of sending instant messages. The text of the instant message is transported in the body of the SIP request.[5]

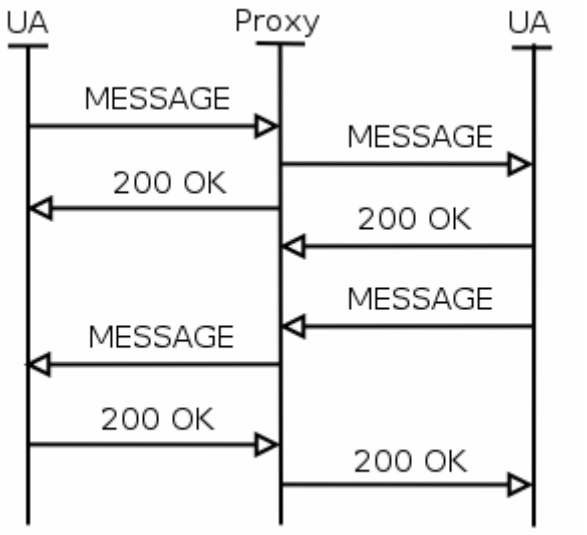



**Figure 14: Instance Messages using SIP**

## 3.2 SDP: Session Description Protocol

The Session Description Protocol (SDP) describes multimedia sessions for the purpose of session announcement, session invitation and other forms of multimedia session initiation.

Session directories assist the advertisement of conference sessions and communicate the relevant conference setup information to prospective participants. SDP is designed to convey such information to recipients. SDP is purely a format for session description - it does not incorporate a transport protocol, and is intended to use different transport protocols as appropriate including the Session Announcement Protocol (SAP) , Session Initiation Protocol (SIP) , Real-Time Streaming Protocol (RTSP) , electronic mail using the MIME extensions, and the Hypertext Transport Protocol (HTTP) .
SDP is intended to be general purpose so that it can be used for a wider range of network environments and applications than just multicast session directories. However, it is not intended to support negotiation of session content or media encodings.[4]

On Internet Multicast backbone (Mbone) a session directory tool is used to advertise multimedia conferences and communicate the conference addresses and conference tool-specific information necessary for participation. The SDP does this. It communicates the existence of a session and conveys sufficient information to enable participation in the session.

Many of the SDP messages are sent by periodically multicasting an announcement packet to a well-known multicast address and port using SAP (session announcement protocol). These messages are UDP packets with a SAP header and a text payload. The text payload is the SDP session description. Messages can also be sent using email or the WWW (World Wide Web).

The SDP text messages include:

- Session name and purpose
- Time the session is active
- Media comprising the session
- Information to receive the media (address etc.)

## 3.3 RTP: Real-Time Transport Protocol

The real-time transport protocol (RTP) provides end-to-end delivery services for data with real-time characteristics, such as interactive audio and video or simulation data,

over multicast or unicast network services. Applications typically run RTP on top of UDP to make use of its multiplexing and checksum services; both protocols contribute parts of the transport protocol functionality. However, RTP may be used with other suitable underlying network or transport protocols. RTP supports data transfer to multiple destinations using multicast distribution if provided by the underlying network. [2]

RTP itself does not provide any mechanism to ensure timely delivery or provide other quality-of-service guarantees, but relies on lower-layer services to do so. It does not guarantee delivery or prevent out-of-order delivery, nor does it assume that the underlying network is reliable and delivers packets in sequence. The sequence numbers included in RTP allow the receiver to reconstruct the sender's packet sequence, but sequence numbers might also be used to determine the proper location of a packet, for example in video decoding, without necessarily decoding packets in sequence.

RTP consists of two closely-linked parts:

- The real-time transport protocol (RTP), to carry data that has real-time properties.
- The RTP control protocol (RTCP), to monitor the quality of service and to convey information about the participants in an on-going session. The latter aspect of RTCP may be sufficient for "loosely controlled" sessions, i.e., where there is no explicit membership control and set-up, but it is not necessarily intended to support all of an application's control communication requirements.

*Protocol Structure - RTP (Real-Time Transport Protocol)*

<table>
<tr><th>2</th><th>3</th><th>4</th><th>8</th><th>9</th><th>16bit</th></tr>
<tr><td>V</td><td>P</td><td>X</td><td>CSRC count</td><td>M</td><td>Payload type</td></tr>
<tr><td colspan="4">Sequence number</td><td colspan="2">Timestamp</td></tr>
<tr><td colspan="4">SSRC</td><td colspan="2">CSRC (variable 0 - 15 items, 2 octets each)</td></tr>
</table>

**Figure 15: RTP Header**

- V - Version. Identifies the RTP version.

- P - Padding. When set, the packet contains one or more additional padding octets at the end which are not part of the payload.
- X - Extension bit. When set, the fixed header is followed by exactly one header extension, with a defined format.
- CSRC count -Contains the number of CSRC identifiers that follow the fixed header.
- M - Marker. The interpretation of the marker is defined by a profile. It is intended to allow significant events such as frame boundaries to be marked in the packet stream.
- Payload type - Identifies the format of the RTP payload and determines its interpretation by the application. A profile specifies a default static mapping of payload type codes to payload formats. Additional payload type codes may be defined dynamically through non-RTP means.
- Sequence number - Increments by one for each RTP data packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence.
- Timestamp - Reflects the sampling instant of the first octet in the RTP data packet. The sampling instant must be derived from a clock that increments monotonically and linearly in time to allow synchronization and jitter calculations.
- SSRC - Synchronization source. This identifier is chosen randomly, with the intent that no two synchronization sources within the same RTP session will have the same SSRC identifier.
- CSRC - Contributing source identifiers list. Identifies the contributing sources for the payload contained in this packet.

Chapter 4

# Development Environment

## 4.1 Target Platform

In embedded system development one of the vital tasks is to choose appropriate processor for target system. Choice of processor depends on the processing power required, cost and its market significance. Major processing tasks in embedded VoIP phone are network packets processing and digital signal processing. CirrusLogic's EP-9302 processor with 200MHz ARM9 core is suitable for it. We chose it as a processor in embedded VoIP phone development.

One of the major tasks in Project VoIP was to develop embedded VoIP phone hardware architecture. To make the development of embedded phone hardware easy, it was require that target development board should be with CirrusLogic's EP-9302 processor and provides all the schematics. It should also have other components required for VoIP phone. Technologic System's TS-7250 ARM Single board computer is a best development platform for VoIP Phone software design. It has following features:

- 200MHz ARM9 processor with MMU
- 32 MB SDRAM (64 MB optional)

- 32 MB Flash disk (128MB optional)
- 10/100 Ethernet
- USB Flash drives supported
- 2 USB ports
- 2 COM ports
- 20 DIO
- PC/104 expansion bus
- 5 channels 12-bit A/D converter
- Optional 8 channel A/D

Digital Signal Processing (DSP) is a decisive functionality in embedded VoIP Phone. Two main processing parts of DSP required in VoIP Phone are: Analog to Digital Conversion and Digital to Analog Conversion. These functions are performed by ADC and DAC respectively. We selected optional MAX-197 8-channel ADC provided by Technologic System with TS-7250 and TI's TLV5616 12-bit DAC.

Platform required to run VoIP applications is provided by Optimized Embedded Linux kernel. One of our VoIP team member optimized Embedded Linux kernel for VoIP Applications.

## 4.2 Host platform

- 1.5GHz Intel X86 system
- 256MB RAM
- Debian Serge 3.1 with GNU/Linux 2.4.27

## 4.3 Scratchbox – A Cross Compilation Toolkit

Scratchbox is a cross-compilation toolkit designed to make embedded Linux application development easier on ARM and x86 targets. It provides glibc and uClibc as C-library choices. It also provides a full set of tools to integrate and cross-compile an entire Linux distribution.

## 4.4 Ethereal – A Network Protocol Analyzer

Ethereal is used by network professionals around the world for troubleshooting, analysis, software and protocol development, and education. It has all of the standard features expected in a protocol analyzer. Its open source license allows talented experts in the networking community to add enhancements. It has support to analyze VoIP calls, SIP Signaling and RTP streams. It runs on all popular computing platforms, including UNIX, Linux, and Windows.

Chapter 5

# Design and Implementation of SIP UA

## 5.1 SIP UA

An SIP UA (User Agent), a logical entity at VoIP End points comprises of User Agent Client (UAC) and User Agent Server (UAS). The UA component, when integrated into a device, enables the device to originate and terminate calls without and with SIP Proxy Server.

### 5.1.1 User Agent Client (UAC)

A user agent client is a logical entity that creates a new request, and then uses the client transaction state machinery to send it. The role of UAC lasts only for the duration of that transaction. In other words, if a piece of software initiates a request, it acts as a UAC for the duration of that transaction. If it receives a request later, it assumes the role of a user agent server for the processing of that transaction.

### 5.1.2 User Agent Server (UAS)

A user agent server is a logical entity that generates a response to a SIP request. The response accepts, rejects, or redirects the request. This role lasts only for the duration of that transaction. In other words, if a piece of software responds to a request, it acts as a UAS for the duration of that transaction. If it generates a request later, it assumes the role of a user agent client for the processing of that transaction.

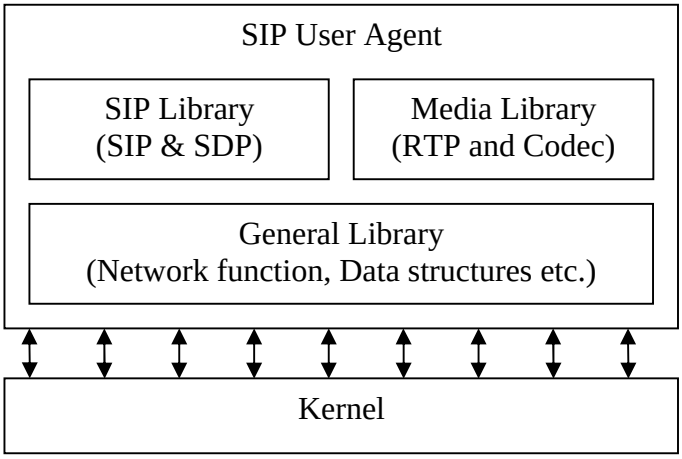



**Figure 16: Block diagram of SIP UA**

SIP User Agents establish VoIP call session between caller and callee using SIP signaling protocol stack. After completion of signaling part, transfer of voice start using RTP media streams, created between two SIP User Agents. When caller or callee wants to terminate call, SIP protocol is used again. Thus, Functionalities of SIP UA can be divided into three parts: SIP protocol stack for signaling, RTP media streams for Voice and basic networking functions for packet transfer.

Figure 16 shows block diagram and interfacing of SIP UA with kernel. SIP User Agent required main three Libraries:

- SIP Library
- Media Library
- General Purpose Library

## 5.2 SIP Library

SIP library contains implementation of a compliant, scalable, and high performance SIP stack. It implements core SIP features. SIP is an application-layer control protocol that can establish, modify, and terminate multimedia sessions (conferences) such as Internet telephony calls. SIP can also invite participants to already existing sessions, such as multicast conferences. Media can be added to (and removed from) an existing session. SIP transparently supports name mapping and redirection services, which supports personal mobility, users can maintain a single externally visible identifier regardless of their network location.

### 5.2.1 Structure of the SIP Protocol

SIP is structured as a layered protocol, which means that its behavior is described in terms of a set of fairly independent processing stages with only a loose coupling between each stage. The protocol behavior is described as layers for the purpose of presentation, allowing the description of functions common across elements in a single section and easy to implementation.

The lowest layer is the transport layer. It defines how a client sends requests and receives responses and how a server receives requests and sends responses over the network. All SIP elements contain a transport layer.

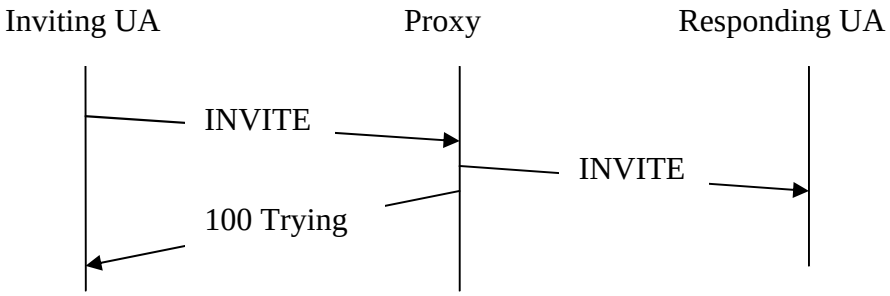
The second layer is the transaction layer. A transaction is a request sent by a client transaction (using the transport layer) to a server transaction, along with all responses to that request sent from the server transaction back to the client. Any task that a user agent client accomplishes takes place using a series of transactions. Stateless proxies do not contain a transaction layer.

The layer above the transaction layer is called the transaction user (TU). Each of the SIP entities, except the stateless proxy, is a transaction user.

***Example***

Structure of the SIP Protocol can easily understand by an example. In this example the Initiating UA send Invite request to stateful proxy and proxy forward it to Responding UA.

Figure 17 shows the following three steps of SIP Call setup:-




Processing done at each stage in figure 16 is:

1. The UAC (User Agent Client) TU:-
    - Creates the initial INVITE request
    - Creates a new client transaction (in the transaction layer) and passes it the INVITE message, plus the IP address, port and transport.
2. The new “INVITE” client transaction (state=”calling”) is identified by the CSeq header field and the “branch” parameter of the Via header fields. T1 timer is started (if UDP) before passing message request to transport.
3. The UAC (User Agent Client) TU:-
    - Creates the initial INVITE request
    - Creates a new client transaction (in the transaction layer) and passes it the INVITE message, plus the IP address, port and transport.
4. The new “INVITE” client transaction (state=”calling”) is identified by the CSeq header field and the “branch” parameter of the Via header fields. T1 timer is started (if UDP) before passing message request to transport.
5. Server Transport:
    - Before sending the request: insert the “sent-by parameter in the via header field”
    - When receiving the request: by examining the “sent-by” parameter in the top Via header field, match it to the relevant server transaction and add the “received” parameter.

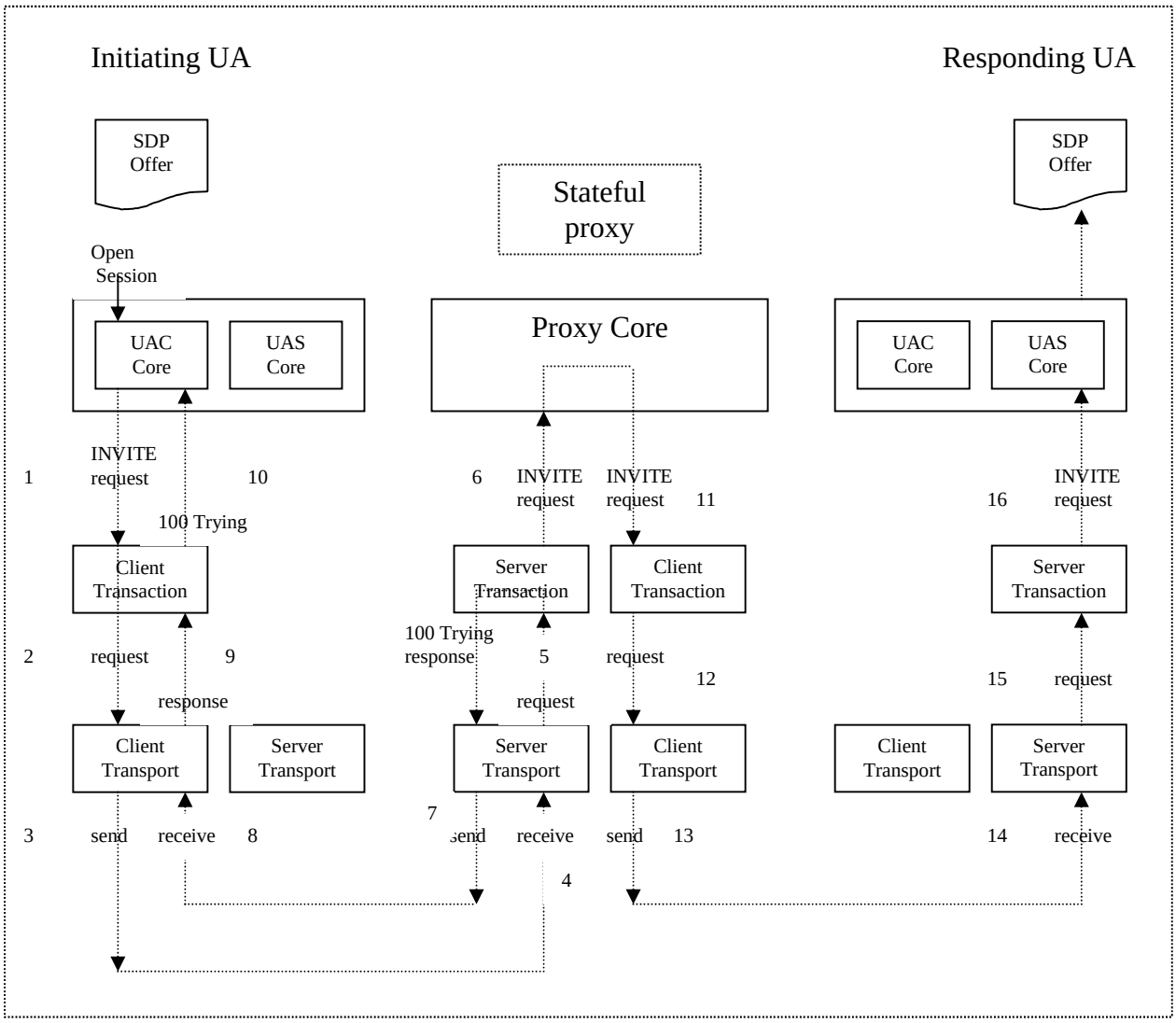



**Figure 17: Structure of SIP Protocol through example**

6. The new “Invite” server transaction (state=”proceeding”) is created by the proxy core (not acting as TU). The server transaction transmits the INVITE request to the TU.
7. The server transaction sends back a 100 Trying response. Before sending a response: retrieve IP@ and port from “sent-by” and “received”
8. When receiving a response: match it to relevant client transaction by examining “sent-by” parameter in top Via header field.
9. When receiving 1xx response: “proceeding” and T1 reset.
10. On receiving 100 trying UAC wait for further reply.
11. For forwarding the INVITE request, the proxy core creates a new client transaction.

12. Client transaction send INVITE request to client transport.
13. Client transport sends it to lower layers to further processing and send to responding UA.
14. On receiving INVITE request, Server transport on Responding UA creates server transaction.
15. Server transport sends it to Server transaction for further processing.
16. Server transaction sends invite request to UAS core of SIP Responding UA.

Other steps of VoIP call setup are mostly similar. Structural architecture of SIP is very useful in SIP Library's design and implementation. Following section describe the detail of Architecture of SIP Library.

### 5.2.2 Architecture of SIP Library

Figure 18 shows Communication Diagram of SIP Library. It shows how (SIP) messages are passed back and forth among SIP components.

***The Endpoint***

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

- It has memory pool factory, and allocates pools for all SIP components.
- It has timer heap instance, and schedules timers for all SIP components.
- It has the transport manager instance. The transport manager has SIP transports and controls message parsing and printing.
- It owns a single instance of ioqueue. The ioqueue is a proactor pattern to dispatch network events.
- It provides a thread safe polling function, to which applications threads can poll for timer and socket events
- It manages SIP modules. SIP module is the primary means for extending the stack beyond message parsing and transport.
- It receives incoming SIP messages from transport manager and distributes the message to modules.

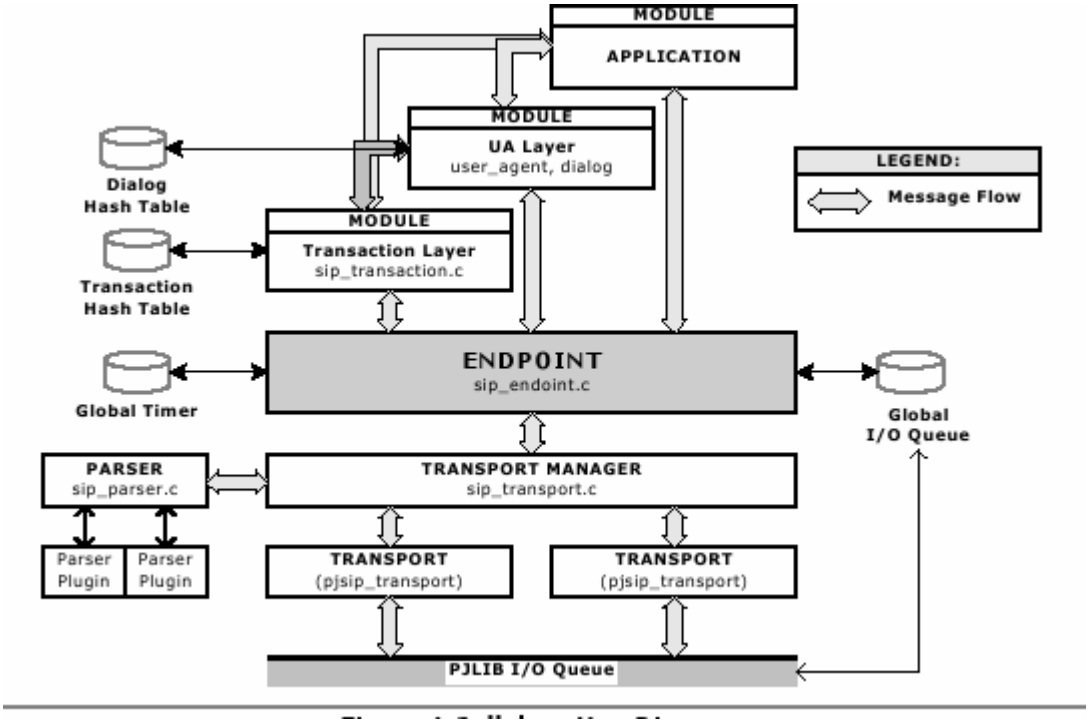



**Figure 18: Communication Diagram of SIP Library**

***Pool Allocations and Deallocations***

All memory allocations for the SIP components will be done via the endpoint, to guarantee thread safety and to enforce consistent policies throughout the entire application. An example of policy that can be used is pool caching, where unused memory pools are kept for future use instead of destroyed.

The endpoint provides these functions to allocate and release memory pools:

- sip_endpt_create_pool(),
- sip_endpt_release_pool().

When the endpoint is created, application MUST specify the pool factory that will be used by the endpoint. Endpoint keeps this pool factory pointer throughout its lifetime, and will use this to create and release memory pools.

***Timer Management***

The endpoint keeps a single timer heap instance to manage timers, and all timer creation and scheduling by all SIP components will be done via the endpoint.

The endpoint provides these functions to manage timers:

- sip_endpt_schedule_timer(),
- sip_endpt_cancel_timer().

The endpoint checks for timers' expiration when the endpoint polling function is called.

***Polling the Stack***

The endpoint provides a single function call (sip_endpt_handle_events()) to check the occurrence of timer and network events. Application can specify how long it is prepared to wait for the occurrence of such events.

SIP stack never creates threads. All execution throughout the stack runs on behalf of applications created thread, either when an API is called or when application calls the polling function.

The polling function is also able to optimize the waiting time based on the timer heaps contents. For example, if it knows at a timer will expire in the next 5 ms, it will not wait for socket events for longer than this; doing so will unnecessarily make the application wait for longer than it should when there is no network events occurs. The precision of the timer will of course vary across platforms.

## 5.3 General Library

General Library is a small foundation library written in C for making scalable applications. Because of its small footprint, it can be used in embedded applications, but yet the library is also aimed for facilitating high performance protocol stacks.

### 5.3.1 Features of General Library:-

- Portability
    - Can run on any kind of processor (16-bit, 32-bit, or 64-bit, big or little endian, single or multi-processors) and operating system.
    - Floating point or no floating point.
    - Multi-threading or not.

- It can even run in environment where no ANSI LIBC is available.

- Small in Size
  - Smaller Footprint for Embedded Applications.
  - user can enable/disable specific functionalities to get the desired size/performance/functionality balance.
- No Dynamic Memory Allocation.
  - not use malloc() at all, but instead should get the memory from a preallocated storage pool.
  - *alloc() is a O(1) operation.*
  - no mutex is used inside alloc(). It is assumed that synchronization will be used in higher abstraction by application anyway.
  - no *free() is required. All chunks will be deleted when the pool is destroyed.*
- Operating System Abstraction
  - Threads

    Portable thread manipulation.
  - Thread Local Storage.

    Storing data in thread's private data.
  - Mutexes

    Mutual exclusion protection.
  - Semaphores.

    Semaphores.
  - Atomic Variables

    Atomic variables and their operations.
  - Critical sections.

    Fast locking of critical sections.
  - Lock Objects

    High level abstraction for lock objects.
  - Event Object.

    Event object.
  - Time Data Type and Manipulation.

    Portable time manipulation.
  - High Resolution Timestamp

    High resolution time value.
- Time Management

- Various Data Structures.
  - String Operations
  - Array helper.
  - Hash Table
  - Linked List
  - Red/Black Balanced Tree
- Exception handling

  TRY/CATCH like construct to propagate errors
- Logging Facility
  - Consists of macros to write logging information to some output device.
  - The verbosity can be fine-tuned both at compile time (to control the library size) or run-time (to control the verbosity of the information).
  - Output device is configurable (e.g. stdout, printk, file, etc.)
  - Log decoration is configurable.
- Random and GUID generation.
- Low-Level Network I/O
  - It has very portable abstraction and fairly complete set of API for doing network I/O communications
  - Socket Abstraction
  - A highly portable socket abstraction, runs on all kind of network APIs such as standard BSD socket, Windows socket, Linux kernel socket, PalmOS networking API, etc.
  - Network Address Resolution
  - Portable address resolution, which implements gethostbyname().
- High-Level Network I/O
  - At higher abstraction, It provides I/O Event Dispatching Queue, which promotes creating high performance network applications by managing asynchronous I/O. This is a passive framework that utilizes the most effective way to manage asynchronous I/O on a given platform, such as: on Linux it can use either /dev/epoll or aio or to fall back to use select() .
  - At even a higher abstraction, It provides Event Queue, which combines asynchronous I/O with timer management and thread management to facilitate creating truly high performance, event driven application.

## 5.4 Media Library

Media library contains objects that implements multimedia capabilities. It can be used with signaling libraries such as SIP to create a complete multimedia communication endpoint. Media library contains framework to manage multimedia communications. It contains:

- RTP/RTCP
- Codec abstraction
- Stream framework
- Sound Abstraction

### 5.4.1 RTP/RTCP

This module is designed according to RFC 1889 (RTP: A Transport Protocol for Real Time Applications). It also contains the RTCP (Real Time Control Protocol) functions to manage and control RTP streams.

### 5.4.2 Codec Abstraction

To minimize the bandwidth requirement and improve the quality of audio, it is required to decode the sample voice signal using one of the standard coder at sender side and decode that samples using encoder at receiving side. Codec Abstraction module contains implementation of G.711 A-law, G.711 u-law, GSM and Speex coder.

### 5.4.3 Stream Framework

Stream Framework contains a bidirectional multimedia communication between two endpoints. It corresponds to a media description (m= line) in SDP. A media stream consists of two unidirectional channels:

1. Encoding channel, which transmits unidirectional media to remote
2. Decoding channel, which receives unidirectional media from remote.

Application normally does not need to create the stream directly; it creates media session instead. The media session will create the media streams as necessary, according to the media descriptors that present in local and remote SDP.

### 5.4.4 Sound Abstraction

This module contains functions required to integrate media stream and media device. So, it depends on the media device driver used.

For Linux based X86 system, media devices are abstracted by one of the two standard media libraries: ALSA or OSS. To provide support of one of this library this stream abstraction module includes PortAudio library. PortAudio is an excellent sound device library. It has the following characteristics that make it perfect for our use:

- It is portable and complete
- It supports multiple back-ends on Windows, Windows CE (WinCE)/PocketPC, Linux, Unix, and MacOS. More platforms may be supported.
- It is callback based
- The callback based for supplying/retrieving frames from the audio device is perfect for our use.
- C based library.
- Actively maintained.

On TS-7250 the target development board, there is not an in-built sound device. We use ADC and DAC chips for that function. Design and integration of these DAC/ADC modules in place of sound abstraction module was done.

SIP Library, Media Library and General library were integrated to implement working SIP User Agent. It is command line SIP user agent. SIP User Agent has features, such as:

- Basic SIP registration.
- Basic INVITE session.
- Call features: call hold, call transfer.
- IM functionality.
- Presence of user.

## 5.5 Size of SIP UA

SIP User Agent is compiled statically and dynamically using GCC. Table 1 shows the size of executable SIP User Agent after compilation.

| SIP User Agent | Size |
|---|---|
| Statically Compiled | 845KB |
| Dynamically Compiled | 354KB |

**Table 1: Size of SIP UA**

If optimization options of GCC are used then size of it will reduced to less than 200KB. Thus, the main problem of applications size in design of Embedded VoIP Phone is solved.

Chapter 6

# Interfacing of ADC/DAC Modules with SIP UA

## 6.1 ADC (Analog to Digital Converter)

An analog-to-digital converter is an electronic circuit that converts continuous signals to discrete digital numbers. In VoIP call, analog voice signals send as digitized packets. So, in embedded VoIP Phone there should be an ADC. Technologic System provides optional MAX-197BCNI ADC on TS-7250 development board. We procured that optional ADC with our development board TS-7250.

MAX197BCNI is a DAS (Data Acquisition System) with following feature:-

- 12 bit resolution.
- Single +5v operation.
- 8 Analog input channels.
- 6 us conversion time.
- 100ksps sampling rate.

On TS-7250 board an ADC header is provided to access all the 8 channels of MAX197 ADC.

### 6.1.1 ADC Driver Module

The next task was to build ADC driver module. ADC driver module built for MAX197 with following features:

- Once the driver is installed and loaded, program can access the data through the newly created 'dev' files, '/dev/adc1' -> 'dev/adc8'.
- It uses interrupt driven strategy to acquire samples.
- The timer TC3 of 32bit is use to generate interrupt at specific intervals.
- Interrupt routine acquired the sample of 12bit.
- The sampling rate can be easily set through the proc file system interface.
  eg.
  echo "8000" > /proc/driver/adc/frequency
  will set the sampling rate to 8000S/s

- A sampling interval can also be set, if you want sampling rate less than 1S/s
  eg.
  echo "3000" > /proc/driver/adc/interval
  will set the sampling rate to 0.333S/s or 3000ms per sample
- The sampling format can all be gets/sets. (binary/ string)

## 6.2 DAC(Digital to Analog Converter )

A Digital to Analog Converter (DAC) is a device for converting a digital (usually binary) code to an analog signal (current, voltage or charges). In VoIP Phone, there should be a device, which converts digitized audio into analog sound. In TS-7250, there is not DAC but there is a DIO header with a Serial Peripheral Interface (SPI).

SPI bus is basically a relatively simple synchronous serial interface for connecting low speed external devices using quite minimal number of wires. SPI (serial peripheral interface) is an interface standard defined by Motorola on the MC68HCxx line of microcontrollers. A synchronous clock shifts serial data into and out of the

microcontrollers in blocks of 8 bits. SPI is used frequently in handheld and other mobile platform systems.

SPI bus is a master/slave interface. Whenever two devices communicate, one is referred to as the "master" and the other as the "slave" device. The master drives the serial clock. When using SPI, data is simultaneously transmitted and received, making it a full duplex protocol. Motorola's names for the signals are as follows: SCLK for serial clock, which is always driven by the master: MISO is master-in slave-out data: MOSI is master-out slave-in data. In a typical application, connect the microcontroller's SCLK to the converter's SCLK input, connect the MISO to the converter's DOUT pin, and connect the MOSI pin to the converter's DIN pin. Serial protocols such as SPI, a chip-select input is required to enable the IC.

We used IT's TLV5615 Digital to Analog Converter. It has following features:

- 12-bit voltage output DAC
- Programmable setting time vs. power consumption.
  - 3us in fast mode
  - 9us in slow mode
- Can drive using SPI interface

*Top View of TLV5616*

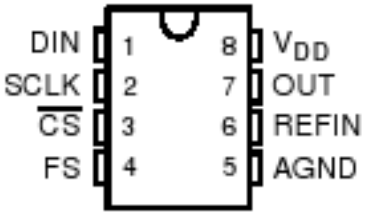



**Figure 19: Top View of TLV5661**

The TLV5616 is a 12-bit single supply DAC based on resistor string architecture. The device consists of a serial interface, speed and power-down control logic, a reference input buffer, a resistor string, and a rail-to-rail output buffer.

### 6.2.1 Functional Block Diagram of TLV5616

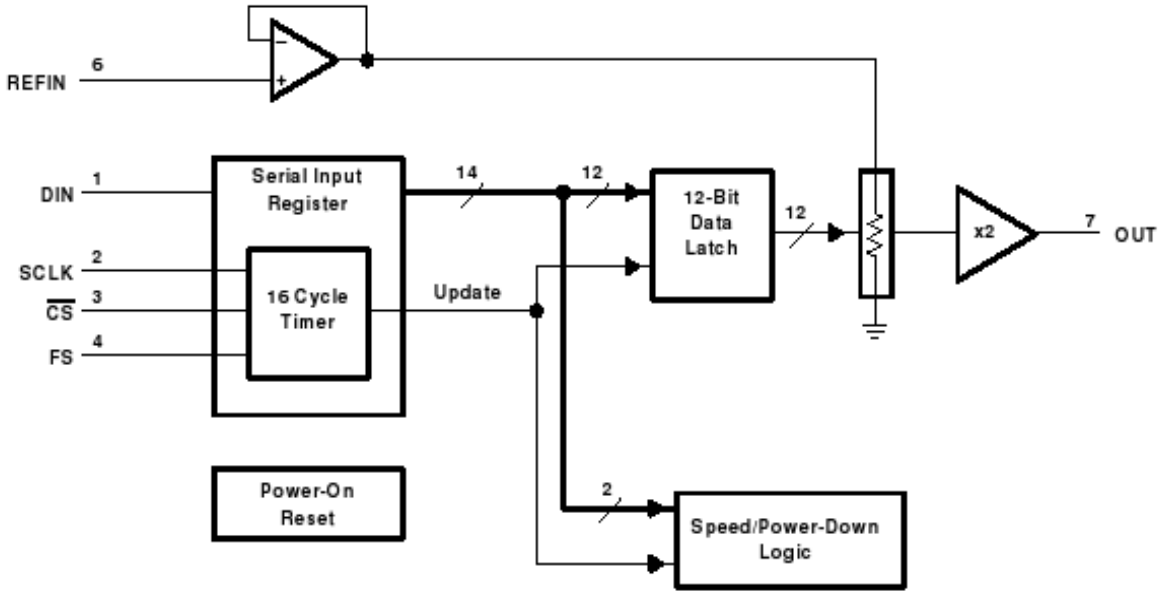



**Figure 20: Functional block diagram of TLV5616**

The output voltage (full scale determined by external reference) is given by:

2* REF *(CODE/2^n) [V]

Where

REF is the reference voltage

CODE is the digital input value within the range of 0 to 2^n–1(decimal),

n = 12 (bits).

The 16-bit data word, consisting of control bits and the new DAC value.

| D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| X | SPD | PWD | X | New DAC value (12 bits) | | | | | | | | | | | |

[16-bit Data Word]

The 16-bit data word for the TLV5616 consists of two parts:

Control bits (D15 . . . D12)

New DAC value (D11 . . . D0)

X: don't care

SPD: Speed control bit. 1 = fast mode 0 = slow mode

PWR: Power control bit. 1= power down 0=normal operation

### 6.2.2 Interfacing of TLV5616 (DAC) with TS-7250

The TLV5616 is a 12-bit voltage output digital-to-analog converter (DAC) with a flexible 4-wire serial interface. The 4-wire serial interface allows glueless interface to SPI serial ports.

TLV5616 is connected to TS-7250 Board's SPI Interface as shown in figure. In this connection TLV5616 work as slave device which is controlled by CIRRUS logic's EP-9302 processor on TS-7250.

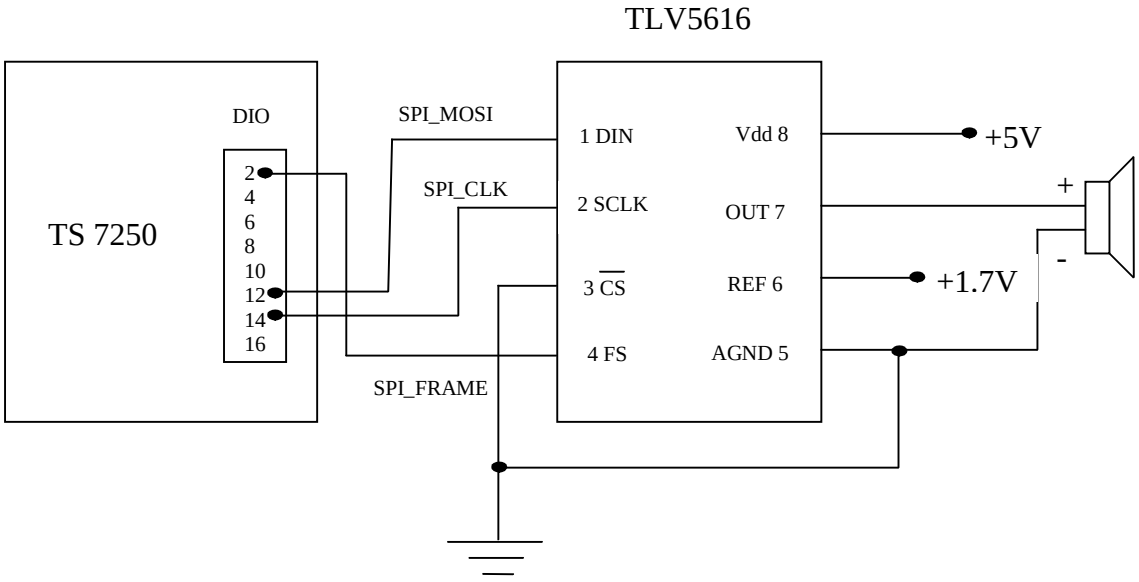



**Figure 21: Interfacing of TLV5616 (DAC) with TS-7250**

We developed DAC driver module. But due to some technical problem we are not getting proper output from it.

## 6.3 Interfacing of ADC/DAC module with SIP UA

ADC/DAC modules are interfaced with SIP UA to make working SIP UA for TS-7250 board. For that, RTP library's sound abstraction module is changed. As explained in previous chapter, this module contains functions required to integrate media stream and media device.

The basic VoIP Call setup steps using SIP User Agent:

- SIP signaling to setup session.
- Create RTP Streams.
  - Encoding Stream. (Data Transfer).
  - Decoding Stream. (Data Receive).
- Call ended with BYE message of SIP Protocol.
- Destroy RTP Streams.

### 6.3.1 RTP Streams

A full duplex RTP Stream consists of two different streams for audio record and play. Created two different streams are Encoding Stream and Decoding stream. In SIP UA, these two different streams should work as two different threads.

***Encoding Stream***

Encoding stream in SIP UA capture samples, encode it using proper encoder, create packets and send it to socket for further transmission. A single Record Callback Thread is created encoding stream. Procedure of this thread is:

*Record Callback Thread Procedure*

1. Open analog audio sample acquisitions device (ADC).
2. Set the sampling rate of ADC to 8000 s/S.
3. Lock Channel Mutex.
4. Capture samples for 20ms. (160 samples if sampling rate is 8000s/S).
5. Encode samples using standard encoder.
6. Create RTP Header.

7. Encapsulate encoded samples into RTP payload.
8. Send RTP Packet to socket.
9. Unlock Channel Mutex.
10. Go to step 3.

***Decoding Stream***

In SIP UA, Decoding Stream's main functions are: Capture packets, extract payload and put it into Jitter buffer, decode payload samples with proper decoder, Send it to proper device buffer for play. Stream Decode Thread and Play Callback Thread are created for decoding stream functionality. Stream Decoder Thread collects packet and put it in Jitter buffer while Play Callback Thread extract samples from Jitter buffer one by one and send to player device buffer for play.

*Stream Decode Thread Procedure*

1. Wait for packet.
2. Get RTP packet from socket.
3. Start locking channel mutex.
4. Decode RTP packet into header and payload
5. Copy RTP payload to buffer.
6. Unlock channel mutex.
7. Go to 1.

*Play Callback Thread Procedure*

1. Open playback device (DAC).
2. Start Locking channel mutex.
3. Get a Frame from buffer.
4. Decode frame using standard decoder.
5. Put decoded frame into sound buffer for play.
6. Unlock channel mutex.
7. Go to 2.

After integrating ADC and DAC modules in SIP UA, it is cross compiled using Scratchbox and ported on TS-7250. When call is made between development board and PC:

- ✓ SIP signaling to setup session.
- ✓ Create RTP Streams.
  - ✓ Encoding Stream. (Data Transfer).
  - X **Decoding Stream. (Data Receive).**
- ✓ Call ended with BYE message of SIP Protocol.
- ✓ Destroy RTP Streams.

As shown above, Encoding Stream is working fine means RTP packets are transferred properly from development board to PC and sound is hear on PC. But in Decoding Stream RTP packets are transfer properly but due to technical problem with DAC sound is not properly generated at development side.

Chapter 7

# Analysis and Results

To analyze the performance of SIP UA, Asterisk Registrar Sever and SIP UAs are run on separate machines in a LAN. Two SIP UAs, one on development board and other on a PC are run. Initially when SIP UA start, it register it self with server. SIP UA Registration Time, Call Setup Time, Bandwidth, Delay and Jitter are analyzed using captured packets during VoIP call.

## 7.1 Registration Time

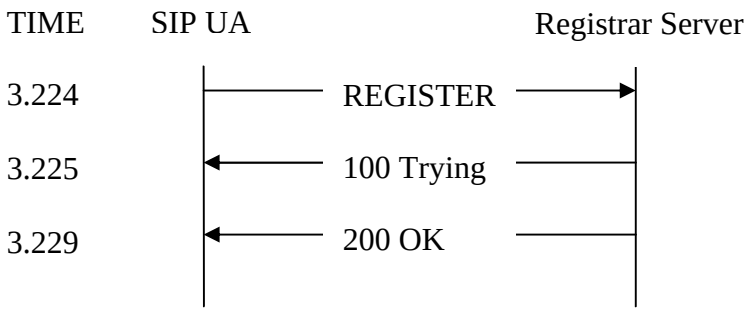



**Figure 22: Registration Time**

Registration time is the time consumed for a new SIP UA to register itself with the registrar server before it can begin using the service. When SIP UA starts, it registers itself with registration Server. Analyzed SIP Registration Time from captured packets is shown in figure 22.

As shown in figure 22 registration time from SIP UA to Asterisk Registrar Server is 5 milliseconds. Thus, the time required to make registration with server is very less.

### 7.2 Call Setup Time:-

Call setup time is the amount of time required to invite a user to a VoIP session and receive a response.

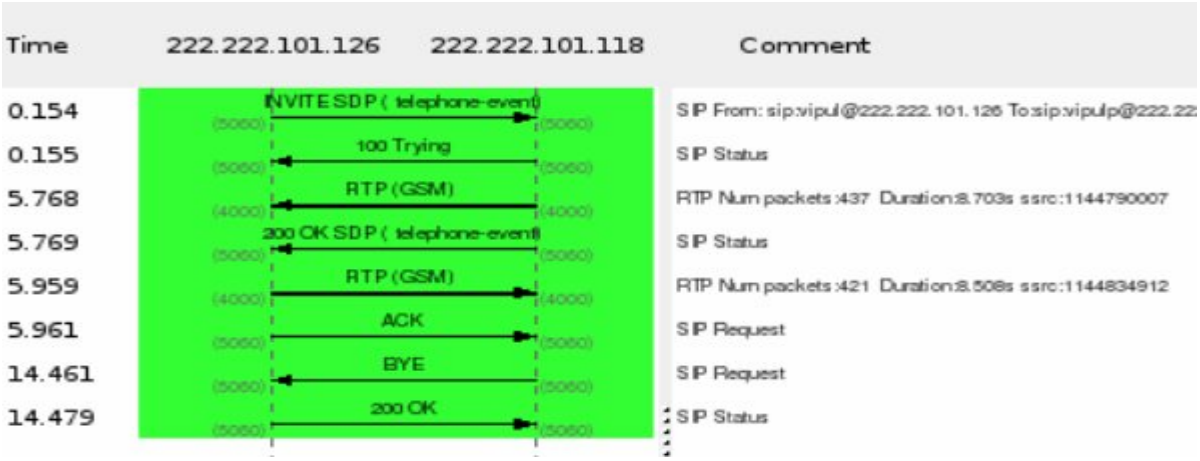



**Figure 23: Call Setup Time**

Figure 23 shows the sip call setup timing between two SIP UAs. Ethereal use its own timer which starts when data capture is started. From figure we conclude that..

| Signaling Steps | Signaling Time in ms |
|---|---|
| Call Setup Time<br>(Time difference between INVITE and its response) | 1 |
| Call Teardown Time<br>(Time difference between BYE and its response) | 18 |

**Table 2: SIP Call Setup Time**

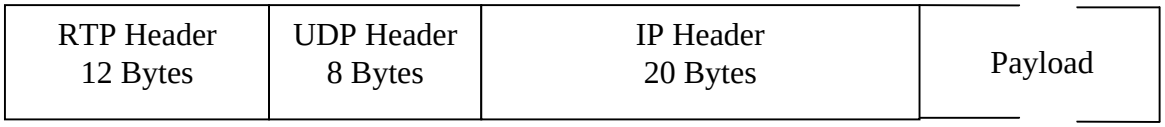
Times recorded for the various steps (as shown in Table 2) show that the signaling time is quite low, which implies fast call-set up and management.

## 7.3 Bandwidth

In VoIP call using SIP UA, after completion of SIP signaling, Media Stream is established between two SIP User Agents. Media Stream used RTP protocol to transfer media between two end points.

### 7.3.1 Bandwidth required by Headers in RTP packet

RTP Packet is consisting of




Total payload size in each RTP packet = 12 + 8 + 20 = 40 Bytes

In VoIP call, RTP payload is consist of digitized speech. It is recommended that the speech should send in 20ms samples in each packet.

Therefore, in 1 second Number of RTP packets sent = 1000/20 = 50 Packets.

And, Bandwidth required by Headers in RTP packet = 40 * 50 * 8 = 16 Kbps

Thus, whatever will be the coder used to codec the speech signal 16 Kbps bandwidth is required by packet headers.

### 7.3.2 Bandwidth required by Payload in RTP packet

To Transfer analog speech signal between two end point using IP networks, it’s required to convert analog voice into digital form. There are lots of type of coders are available. In SIP User Agent there was support of PCMA and PCMU codec. Support of Speex and GSM codec is added in SIP UA.

Frequency of Human voice is up to 4 KHz. According to Nyquist theorem the sampling frequency should be double. So,

Number of Sample taken in 1 Second = 8000 Samples.

Number of Samples taken in 20 milliseconds = 8000*20 / 1000 = 160 samples

Each Sample is PCM 16 bit,

Number of bits in 20 milliseconds digitized speech = 16 * 160 = 2.56 Kbits

Figure 24 represent the screen shot taken from Ethereal, when PCMA coder is used.

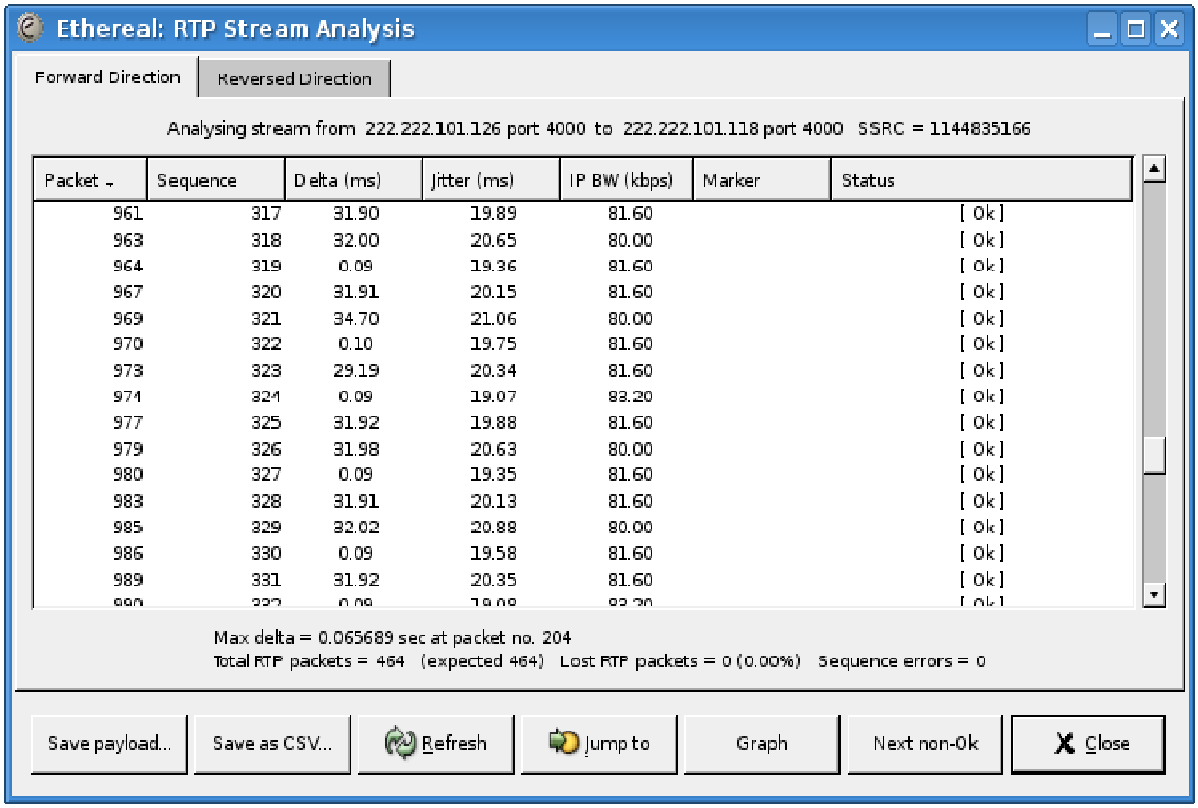



**Figure 24: Screen Shot of One side RTP Stream**

Each of the above coder take 160 samples input and compresses it. Table 3 explains compression of each of the coder and bandwidth required by them.

| Encoding Format | Number of bits in compressed 160 samples | Number of bits in 1 second digitized speech (payload bandwidth)(bps) | Total bandwidth (Header(16K) +payload) bps | Bandwidth get through Ethereal(bps) |
|---|---|---|---|---|
| PCMA | 8*160 = 1280 | 1280 * 50 = 64K | 80K | 80K ~ 81.6K |
| PCMU | 1280 | 64K | 80K | 80K ~ 81.6K |
| GSM | 264 | 13.2K | 29.2K | 29.2K ~ 30.37K |
| SPEEX | 160 | 8K | 24K | 24K ~ 24.98K |

**Table 3: Bandwidth required by different coder**

## 7.4 Delay and Jitter

Jitter is variation in delay. All networks, encoding and decoding processes add delay. If you only had delay (and no packet loss) you would not need buffering as you would know always when the next voice frame was going to arrive you could just play the voice sample. But because of congestion, multiple network paths, and other variations there will be always times when a voice frame arrives late. Because, you always want to have something play (not just silence) the designers of a VoIP system always will have buffer at least as long maximum tolerable delay variation or jitter, plus some breathing space.

For jitter levels under 100 milliseconds then it may be acceptable to increase the jitter buffer size in end-systems or to enable adaptive jitter buffer operation.

For jitter levels over 100 milliseconds then increasing the jitter buffer size to avoid packet discards will introduce significant delay and cause conversational problems; in this case it is preferable to take steps to isolate the source of jitter and eliminate the problem at source.

Ethereal calculates RTP delay and jitter as described in RFC 1889, that is:

If Si is the RTP timestamp from packet i, and Ri is the time of arrival in RTP timestamp units for packet i, then for two packets i and j, D may be expressed as

$$D(i,j)=(Rj-Ri)-(Sj-Si)=(Rj-Sj)-(Ri-Si)$$

The inter arrival jitter is calculated continuously as each data packet i is received from source SSRC_n, using this difference D for that packet and the previous packet i-1 in order of arrival (not necessarily in sequence), according to the formula

$$J=J+(|D(i-1,i)|-J)/16$$

After establishment of VoIP Call between two SIP UAs, RTP Streams are captured and analyzed using Ethereal. Table 4 shows the results of jitter for different coder.

| Coder format | Jitter (ms) |
|---|---|
| PCMA | 19.0 ~ 20.9 |
| PCMU | 19.07 ~ 21.06 |
| GSM | 19.1 ~ 21.03 |
| SPEEX | 19.1 ~ 20.8 |

**Table 4: Jitter for different coder format**

From above result we can concluded that, the delay variation is between 19 to 22 ms and it is adequate for VoIP call.

Chapter 8

# Conclusions and Scope of Future Works

The design, implementation, and performance evaluation of a SIP-based Voice over IP (VoIP) architecture is discussed and presented. Design supports a rich set of call signaling functionality to established point-to-point audio call. In addition, the implementation also provides several enhanced features including call holding, call forwarding and mobility support. Also design, implementation and integration of DAC/ADC modules with SIP UA and make Embedded SIP UA for target platform (TS-7250).

Project VoIP's future plan is to develop ten Embedded VoIP Phones similar to target platform (TS-7250) and make campus wide VoIP Architecture. If it will succeed with good quality of audio, next plan is to replace traditional PSTN based IP Centrex

system within campus with SIP based VoIP architecture. In SIP UA more functionality such as multipoint audio conference, Video support can be added. It is also possible to make wireless IP Phone using SIP UA.

# References


1. J. Rosenberg and H. Schulzrinne and G. Camarillo and A. Johnston and J. Peterson and R. Sparks and M. Handley and E. Scholarship, Session Initiation Protocol, RFC 3261,Internet Engineering Task Force, June 2002;

2. H. Schulzrinne and S. Casner and R. Frederick and V. Jacobson, RTP: A Transport Protocol for Real-Time Applications, Audio-Video Transport Working Group,RFC 1889, Internet Engineering Task Force, January 1996;

3. M. Handley ACIRI, H. Schulzrinne, E. Schooler , J. Rosenberg, SIP: Session Initiation Protocol, RFC2543,IETF, March 1999; □

4. M. Handley and V. Jacobson, SDP: Session Description Protocol, RFC 2327, Internet Engineering Task Force, April 1998;

5. B. Campbell, J. Rosenberg, H. Schulzrinne,C. Huitema, D. Gurle, Session Initiation Protocol (SIP) Extension for Instant Messaging, RFC 3428, December 2002;

6. A. B. Roach, Session Initiation Protocol (SIP)-Specific Event Notification, Network Working Group, RFC 3265, June 2002;

7. S. Zeadally and F. Siddiqui , Design and Implementation of a SIP-based VoIP Architecture, High-Speed Networking Laboratory, Department of Computer Science Wayne State University, IEEE;

8. H Chang, M Hsua, M. Chang, V Hsua, The Internetworking functions of VoIP protocols, http://www.ccrc.nthu.edu.tw/excellence/activity/911015/2/s-2/3.pdf;

9. David H. Crocker, Standard for the format of ARPA Internet text messages, RFC 822, August 13, 1982.

10. Taking Charge of Your VoIP Project, John Q. Walker, Jeffrey T. Hicks, Cisco Press, Paperback, Published February 2004;

11. EP9302's User Guide;

12. Datasheet for TLV5616CP;

13. TS-7250 ARM Single Board Computer, http://www.embeddedarm.com/epc/ts7250-spec-h.html;

14. VOIP,http://www.gaoresearch.com/resources/whitepapers/other/voip.php;

15. Scratchbox, http://www.scratchbox.org/;

16. SIP Protocol Structure through Example, http://www.tech-invite.com/Ti-SIP_Protocol_structure.pdf;

17. SIP RFCs and Drafts, http://www.cs.columbia.edu/sip/drafts.html;

18. SIP, RTP and General Library Modules, http://www.pjproject.net/;

19. Network Protocol Analyzer Ethereal, http://www.ethereal.com/;

20. SIP Protocol , http://www.iptel.org/drupal/sip/intro/scenarios/;