VOIP OVER WIRELESS LAN

By

Upendra Patel

(04MCE014)

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

VOIP OVER WIRELESS LAN

A Dissertation

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Computer Science & Engineering

By

Upendra Patel

(04MCE014)

Guide

Dr. S.N. Pradhan

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

VOIP OVER WIRELESS LAN

Presented by

Upendra Patel
(04MCE014)

has been accepted toward fulfillment of the requirements

for the degree of

Master of technology in Computer Science & Engineering

Professor In Charge	Head of The Department
Dr. S. N. Pradhan	**Prof. D. J. Patel**

Date: 29/04/06

CERTIFICATE

This is to certify that the work presented here by **Mr. Upendra Patel** entitled "**VOIP OVER WIRELESS LAN"** has been carried out at **Nirma Institute Of Technology** during the period **September 2005 – April 2006** is the 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.

Dr. S. N. Pradhan

Date: 29/04/06

Acknowledgement

"Sometimes our light goes out
but is blown into flame by another human being.
Each of us owes deepest thanks
to those who have rekindled this light"

It gives me great pleasure in expressing thanks and profound gratitude to **Dr. S. N. Pradhan,** M.Tech In-Charge, Department of Computer Engineering, Institute of Technology, Nirma University, Ahmedabad for his valuable guidance and continual encouragement throughout the Major project. I heartily thankful to him for his time to time suggestion and the clarity of the concepts of the topic that helped me a lot during this study.

I would like to give my special thanks to Prof. **D. J. Patel,** Head , Department of Computer Engineering, Institute of Technology, Nirma University for his continual kind words of encouragement and motivation throughout the Major Project. I am also thankful to **Dr. H.V. Trivedi** , Director, Institute of Technology, Nirma University.

I am thankful to all **faculty members** for their special attention and suggestion towards the project work. I extend my sincere thanks to my colleagues for their support in my work.

I am really thankful to **GOD** who gives me courage to face difficulties and overcome them. I would like to express my gratitude towards my family members who have always been my source of inspiration and motivation.

Upendra Patel

Abstract:

(VOIP) networks combine both voice and data communications networking technologies. VOIP networks combine the best of voice and data communications networking technologies. But that *combination* also creates some challenges, as the industry attempts to meld the best of circuit switching (from the voice side) and packet switching (from the data side) into single technology. TCP/IP protocol stack is not sufficient for VOIP.

Seamless wireless LAN is fast becoming a reality. VOIP over WLAN as technology enables IP voice to be sent over an (802.11) WLAN. Voice over IP over Wireless LAN is getting great attention from the industry and the products for VOIP OVER WLAN deployment are emerging rapidly. This technology can bring people many benefits. It has all the advantages of VOIP systems along with greater mobility. Mobility usually means increased productivity since it also reduces the cost for deployment of wired phones. Just as when introducing voice to IP networks, VOIP Over WLAN has the same problems as VOIP regarding deployment. Some characteristics of wireless networks cause additional difficulties. Carrying voice over wireless networks introduces new challenges. The WLAN industry is working hard to enable 802.11-based networks to accommodate the technical characteristics of VOIP.

In this project, the work has been carried out is implementing SIP Soft Phone in **simputer** which is a ARM based handheld device running Linux and having wireless environment. The performance of call flow and media streams has been also measured and analyzed.

List of Tables:

List of Figures:

Contents:

Acronyms and abbreviations

VOIP:	Voice Over Internet Protocol
VOWAN:	Voice Over Wireless LAN
SIP:	Session Initiation Protocol
RTP:	Real Time Protocol
RTCP:	Real Time Control Protocol
MGCP:	Media Gateway Control Protocol
SDP:	Session Description Protocol
OSS:	Open Sound System
SSRC:	Synchronization source
CSRC:	Contributing source

1.Introduction

1.1. Motivation

Good communication services and universal access are necessary for a higher standard of living and economic growth. However the high cost of legacy PSTN call may not be affordable to every body. Voice over IP (VOIP) introduces voice into the packet switching world, which brings the convergence of packet and circuit networks. Today with VOIP, we can make economical long distance calls. IEEE 802.11 wireless local area networks have become an increasingly popular user access technology and a flexible alternative to wired access. WLANs are now appearing not only as wireless extensions of private corporate networks, but also in homes and public hot spots. Wireless Local Area Network (WLAN) gives freedom to the people from the wired network connections and hence users now can enjoy greater mobility.

Recently, many people begun to show interests in delivering voice over WLAN, which promises further mobility to users. Years of experience with wireless LAN has made this technology quite mature. However, the introduction of voice into WLAN has brought new challenges. The WLAN industry is working hard to enable 802.11-based networks to accommodate the technical characteristics of VOIP. VOIP over WLAN as technology enables IP voice to be sent over an (802.11) WLAN .

As said earlier (VOIP) networks combine both voice and data communications networking technologies. VOIP networks combine the best of voice and data communications networking technologies. But that *combination* also creates some challenges, as the industry attempts to meld the best of circuit switching (from the voice side) and packet switching (from the data side) into single technology. TCP/IP protocol stack is not sufficient for VOIP.

1.2 The connection-oriented/connectionless dichotomy

Traditional voice networks are classified as *connection-oriented networks*, in which a path from the source to destination is established, prior to any information transfer. When the end user takes the telephone off-hook, they notify the network that service is requested. The network then returns dial tone, and the end user dials the destination number. When the destination party answers, the end-to-end connection is confirmed through the various switching offices along the path. When the conversation is complete, the two parties hang up, and their network resources can be re-allocated for someone else's conversation.

One of the disadvantages of this process is the consumption of resources spent setting up the call (a process called *signaling*, which we will consider in a future tutorial). One of the advantages, however, is that once that call has been established, and a path through the network defined, the characteristics of that path, such as propagation delay, information sequencing, etc. should remain constant for the duration of the call. Since these constants add to the reliability of the system, the term *reliable network* is often used to describe a connection-oriented environment. The **Transmission Control Protocol** (TCP) is an example of a connection-oriented protocol.

In contrast, traditional data networks are classified as *connectionless networks*, in which the full source and destination address is attached to a packet of information, and then that packet is dropped into the network for delivery to the ultimate destination. An analogy to connectionless networks is the postal system, in which we drop a letter into the mailbox, and if all works according to plan, the letter is transported to the destination. We do not know the path that the packet (or letter) will take, and depending upon the route, the delay could vary greatly. It is also possible that our packet may get lost or be miss-delivered within the network, and therefore not reach the destination at all. For these reasons, the terms *best efforts* and *unreliable* are often used to describe a connectionless environment. The **Internet Protocol** (IP) and the **User Datagram Protocol** (UDP) are examples of connectionless protocols.

Recall from your Internet History 101 class, that the Internet protocols, including TCP, IP, and UDP were developed in the 1970s and 1980s to support three key applications: file transfers (using the **File Transfer Protocol**, or FTP), electronic mail (using the **Simple Mail Transfer Protocol**, or SMTP), and remote host computer access (using the **TELNET** protocol). All of these applications were data- (not voice-) oriented, and were therefore based upon IP's connectionless network design. Layering TCP on top of IP gave the entire system enhanced reliability (albeit with additional protocol overhead), but the rigors of a true connection-oriented, switched infrastructure (like the telephone network) was not necessary to support these applications.

1.3. Enhancement for TCP/IP

Fast forward a few decades to the new millennium where visions of voice, fax, and video over IP dominate. These applications *are* sensitive to sequencing and delay issues, and the idea of a "best efforts" service which brings us to the challenging question: How do we support *connection-oriented applications* (such as voice and video) over a *connectionless environment* (such as IP), without completely redesigning the network infrastructure? The solution is to enhance IP with additional protocols that fill in some of its data-centric gaps. These include:

- **Multicast Internet Protocol** (Multicast IP), defined in RFCs 1112 and 2236. Multicast allows information from a single source to be sent to multiple destinations (as may be required for conferencing).
- **Real-time Transport Protocol** (RTP), defined in RFC 3350. RTP provides functions such as payload identification, sequence numbering, and timestamps on the information.

- **RTP Control Protocol** (RTCP), also defined in RFC 3350. RTCP monitors the quality of the RTP connection.
- **Resource Reservation Protocol** (RSVP), defined in RFC 2205. RSVP requests the allocation of network resources, to assure adequate bandwidth between sender and receiver.
- **Real-Time Streaming Protocol** (RTSP), defined in RFC 2326. RTSP supports the delivery of real-time data, including retrieval of information from a media server or support for conferencing.
- **Session Initiation Protocol (SIP)** defined in RFC 3261**.** Session Initiation Protocol is a signaling protocol for Internet conferencing, telephony, presence, events notification and instant messaging.
- **Session Description Protocol** (SDP), defined in RFC 2327. SDP conveys information about the media streams for a particular session, including session name, time the session will be active, what media (voice, video, etc.) is to be used, the bandwidth required.

So SIP is used for creating, maintaining and terminating voice session on IP based Wireless Environment. SIP and RTP stacks are used in this project.

1.4 Voice over Wireless LAN

VOIP over WLAN refers to the provisioning of IP voice services across wireless LANs, usually 802.11-based (also known as voice over Wi-Fi). A VOIP over WLAN system works by translating a (PBX) telephone call to IP packets and sends these IP packets over an 802.11 WLAN.

This technology can bring people many benefits. It has all the advantages of VOIP systems along with greater mobility. Mobility usually means increased productivity since it also reduces the cost for deployment of wired phones. Just as when introducing voice to IP networks, VOIP over WLAN has the same problems as VOIP regarding deployment . Some characteristics of wireless networks cause additional difficulties.

Generally speaking, there are several major issues need to be solved before this technology will be fully accepted.

(1) Latency-induced VOIP performance degradation as users roam. This latency is usually caused by re-authentication required when associating with a new access point. ITU recommends that the total latency in a voice call should not exceed 150 milliseconds. In VOWLAN practice, it's agreed that the delay of roaming and authentication needs to be kept under 50 milliseconds.

(2) Lack of QOS (Quality of Service) mechanisms. In order to keep the real time features of voice traffic, voice packets should be assigned higher priority then normal data packets to maintain the quality of voice. In this sense, QOS is needed. Although voice gets priority treatment, it cannot get a reserved bandwidth as it is not guaranteed that low priority frames will always wait until all higher priority frames are transmitted, so in order to provide better service to streams with higher priority, a reserved bandwidth may be helpful.

(3) Today, VOIP handsets have less security than other data devices. This may leave the network vulnerable to potential spoofing. Security measures for data may introduce more latency than voice can tolerate, so a special mechanism for voice should be developed. 802.11i is a task group that aims to enhance security that is stronger and better suited to wireless voice. 802.11i developed algorithms for confidentiality, data integrity, and data source authentication. It also includes a protocol for mutual authentication and key management.

(4) Limited number of voice calls. Because of the capability of access points for handling calls simultaneously, the number of calls is limited. Currently,

when the number of calls reach 30, an access point becomes overloaded, thus the quality degrades.
(5) Rapid drain of handset battery. To achieve true mobility, low power is one of the key point, since users expects long battery life. This should be improved in the future by adding sleep mode to the 802.11 phones. Moreover, low power required low power consumption of each component of a phone, so power saving may involve carefully design of every part.

(6) Insufficient support for video on WLAN. Most products today support only 802.11b WLAN 802.11b's 11Mbps data rate translates into 4-5 Mbps of real throughput. Video requires lots of bandwidth, usually between 128kbps to 2 Mbps, so video now is still restricted on WLAN. On an 802.11a WLAN, the 54 Mbps data rate translates into 20Mbps to 25Mbps of real throughput and even this may not be enough when many users share the bandwidth. Many WLAN vendors are developing solutions to support IP Multicast on the WLAN through QoS to improve this.

2.Background

2.1 VOIP

Many years ago we discovered that sending a signal to a remote destination could have be done also in a digital fashion: before sending it we have to digitalize it with an ADC (analog to digital converter), transmit it, and at the end transform it again in analog format with DAC (digital to analog converter) to use it.

VOIP works like that, digitalizing voice in data packets, sending them and reconverting them in voice at destination[3]. Digital format can be better controlled: we can compress it, route it, convert it to a new better format, and so on; also we saw that digital signal is more noise tolerant than the analog one .

TCP/IP networks are made of IP packets containing a header (to control communication) and a payload to transport data: VOIP use it to go across the network and come to destination. The packet switching technology that is widely used for data communication was not designed for real time content delivery, e.g. voice. But packet switching has advantages over circuit switching as is used by normal voice systems, such as the PSTN (Public Switching Telephone System). In circuit switching, a 'circuit' is maintained for the whole duration of the conversation, thus, a large part of the telephone network resource is wasted at any given time. In contrast, using data networks to deliver voice not only avoids the need for two separate systems for data and voice, but also makes better use of the network's resources. For these reasons, VOIP was introduced and has gained great popularity.

2.2 The VOIP Phone System: Packet Switching

Data networks do not use circuit switching. Your Internet connection would be a lot slower if it maintained a constant connection to the web page you were viewing at any given time. Instead, data networks simply send and retrieve data as you need it. And, instead of routing the data over a dedicated line, the data packets flow through a chaotic network along thousands of possible paths. This is called packet switching.

While circuit switching keeps the connection open and constant, packet switching opens a brief connection -- just long enough to send a small chunk of data, called a packet, from one system to another. It works like this:

- The sending computer chops data into small packets, with an address on each one telling the network devices where to send them.
- Inside of each packet is a payload. The payload is a piece of the e-mail, a music file or whatever type of file is being transmitted inside the packet.
- The sending computer sends the packet to a nearby router and forgets about it. The nearby router sends the packet to another router that is closer to the recipient computer. That router sends the packet along to another, even closer router, and so on.
- When the receiving computer finally gets the packets (which may have all taken completely different paths to get there), it uses instructions contained within the packets to reassemble the data into its original state.

Packet switching is very efficient. It lets the network route the packets along the least congested and cheapest lines. It also frees up the two computers communicating with each other so that they can accept information from other computers, as well

2.3 Interactive audio over packet based networks

The idea of using a packet based network such as the Internet for interactive audio is not new[3]. Despite this, a commercial interest in IP based telephony did not appear until recently, usually in the form of public telephone operators offering public switched telephone network (PSTN) to Internet gateway services, or as private branch exchange (PBX) replacements at large companies. In the future it is likely that most phone calls will be carried by IP end-to-end, at least in countries with wide spread broadband Internet and Intranet infrastructures. When providing telephony services over a packet based network such as the Internet the applications must be able to handle the loss, delay, and jitter characteristics inherent in such a network. In this chapter I describe the mechanisms developed to handle these network characteristics, I start this chapter by providing an overview of the path of the audio data from the generation at the sender's lips and input to their microphone to emission by the recipient's speaker for perception at their ear

2.3.1 Overview of audio data processing

For a real-time application such as interactive voice it is important that the end to- end delay bound is not rigid, but values in the range 150-400 ms are commonly given. delays above the *knee* (□180 ms) have a worse impact on the perceived quality. The components contributing to the delay are either related to the different audio processing mechanisms at the sender and receiver, the Coding and Decoding (CODEC), or to the process of moving the data packets over the Internet (or Intranet).

Sender side: Generating and transmitting audio data

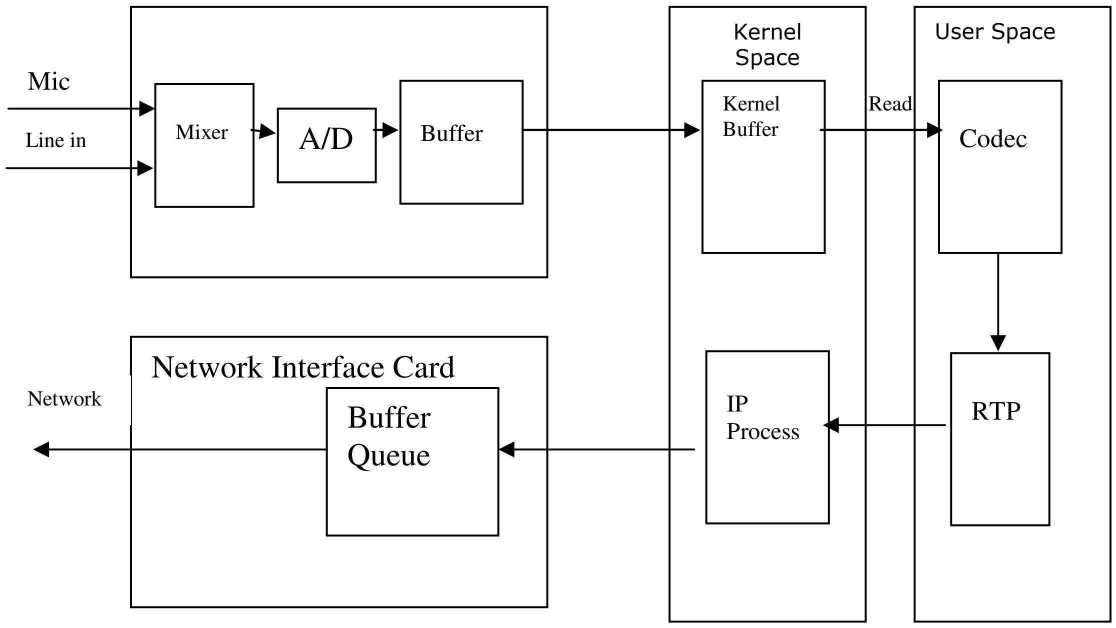


Figure:1 Generating and transmitting audio data

In this section an overview will be given for the mechanisms contributing to delay at the sender and receiver. Consider an analog audio signal captured by a microphone and encoded as pulse code modulation (PCM) samples in the audio hardware, as shown in Fig. 2.1. In this analog-to-digital (A/D) conversion process the signal first passes through a low pass filter (to avoid *aliasing*), a sampler (converting it to a time discrete signal), and a quantized (creating an amplitude discrete signal) where upon the signal can be encoded usually with 8, 16 or 24 bits of resolution Also, in the path there is usually an input mixer to allow for multiple audio sources to be *mixed* or simply selected, although for IP telephony applications only a single source (the microphone) is commonly used. Each of these inputs might be stereo, although for IP telephony we generally use only one channel. The audio samples are first stored in a small buffer/queue inside the audio hardware, from where the audio data is subsequently moved to a kernel space buffer in main memory using direct memory access (DMA) under control of the audio driver. The audio driver provides a software interface allowing   applications to *read* (one or more) samples for further audio processing in user space1.

The audio CODEC usually processes audio in blocks (here referred to as audio *frames*). Hence, to improve efficiency it is desirable if the application can read the PCM samples from kernel space a frame at a time rather than a sample at a time. In the audio CODEC a frame with a set of code words are processed and encoded using a suitable audio format, as selected by the *session description* The sum of the (frame) buffering delay and the look-ahead delay is referred to as the Codec's *algorithmic delay*, and is the minimum delay added by a CODEC. This is in contrast to the *processing delay* of the CODEC, which varies depending on the CODEC implementation and the processing environment. The CODEC may also perform voice activity detection (VAD), whereby frames with speech are encoded as usual, otherwise frames are either encoded as noise or simply not sent at all. VAD could lead to delay at the sender when the CODEC does *look-ahead* .Now that voice frames is sent on RTP and goes for IP processing and enters into NIC.

Receiver side: Receiving and playing audio data

At the receiver the inverse process occurs[3]. In addition the receiver has mechanisms to handle loss and jitter experienced by the audio stream during the traversal of the Internet. When receiving an audio packet first the regular IP/UDP processing is performed, then based upon the packet type and port number the RTP packet is delivered to the application. The RTP payload type specifies the format of the payload, and

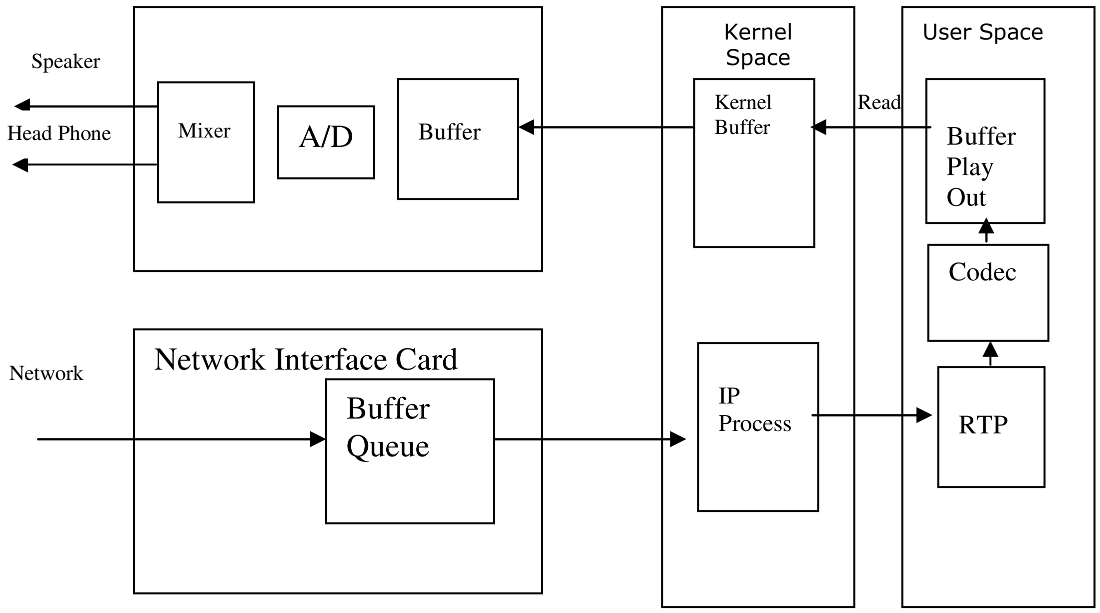


Figure:2 Receiving and playing audio data

the receiving application passes the payload to the appropriate CODEC for decoding .As audio packets traverse the Internet they will experience different delays .In order to smooth out the effect of the delay variation (jitter) the receiver implements a play out (de-jitter) buffer. The goal is that audio data should be played with the same relative timing as it was generated (using timestamp information provided by RTP). Furthermore, the play out delay (*Tplayout*) should be set long enough so that most audio packets will arrive before their scheduled play out time. However, this play out delay increases the mouth-to ear delay, and using too large a value will not be acceptable for interactive applications such as telephony – this leads to a trade-off between delay and packet loss. Many audio applications make use of an adaptive play out buffer management algorithm, in which the play out point adapts to changes in network delay jitter.

2.4 VOIP PROTOCOL STACK:

There is a VOIP stack for Voice Over IP communication[5]. There are different protocol that can be classified in three main categories namely Signaling, Gateway Control, and Media protocol. H.323 and SIP are the main protocols which are used for signaling .MGCP is used for Media Gateway control. RTP, RTCP,RSCP are used for transferring media whether it is audio or video.

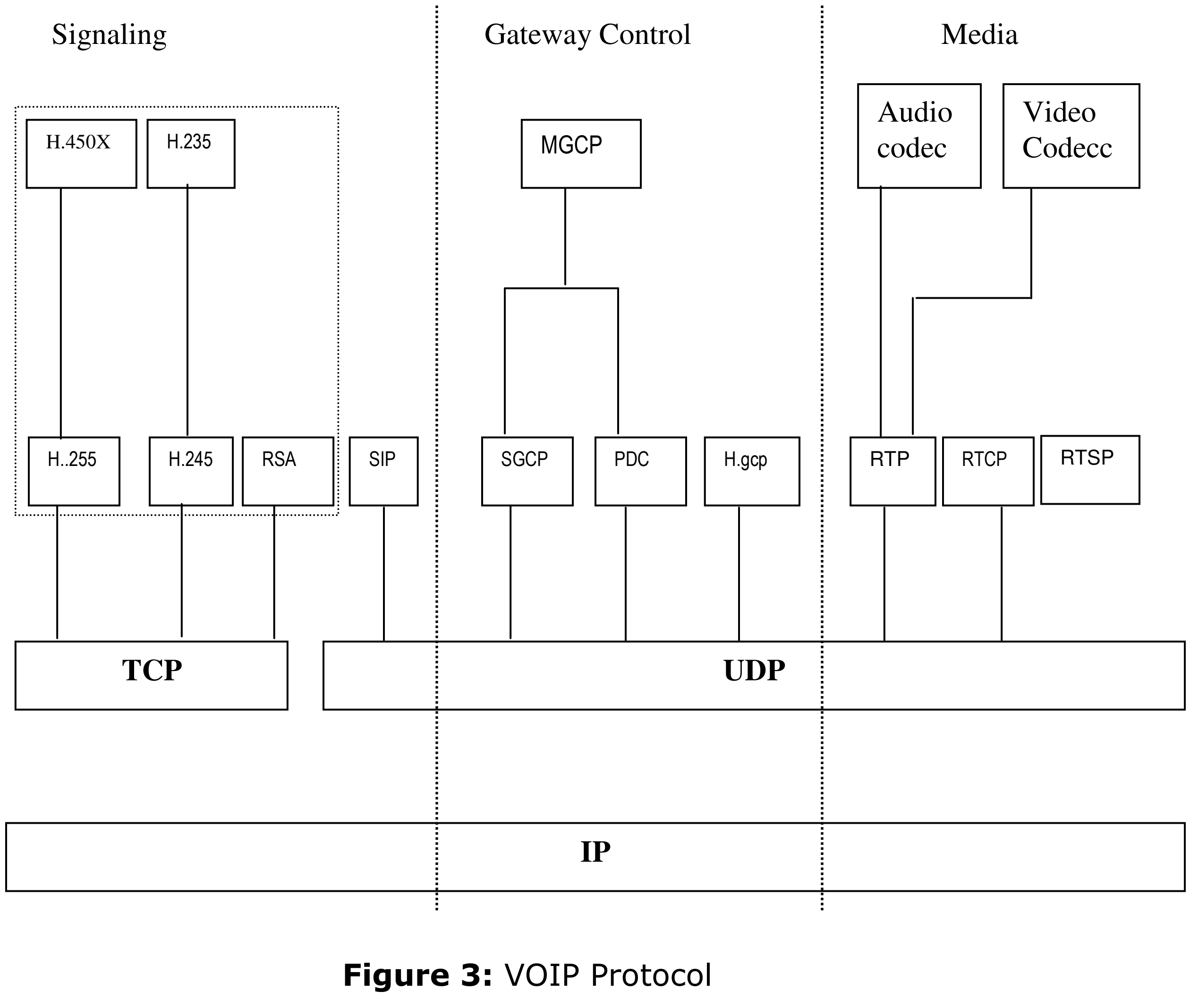


Figure 3: VOIP Protocol

2.5 SIP

2.5.1 SIP base System:

Following diagram illustrates the overall system network. Such a network would allow seamless multiple access options for most of the more prevalent voice and data services.

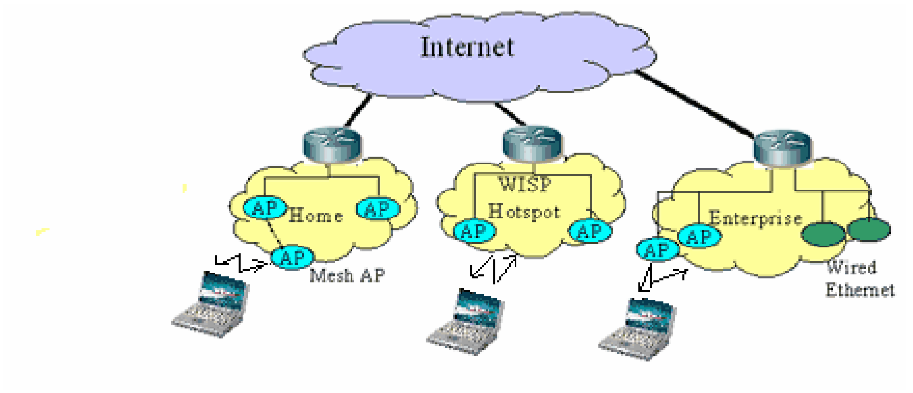


Figure:4 SIP based System Network

The SIP architecture begins by building upon two other Internet application protocols, the Simple Mail Transfer Protocol (SMTP), which is defined in RFC 2821, and specifies the format for electronic mail messages, and the Hypertext Transfer Protocol (HTTP), which is defined in RFC 2616, and specifies the format for web-based multimedia communication. In addition, SIP uses functions defined by the Real Time Transport Protocol/Real Time Control Protocol (RTP/RTCP), defined in RFC 3550, which specifies the formats for multimedia packets over Internet Protocol (IP) networks, and the Session Description Protocol (SDP), defined in RFC 2327, which specifies the characteristics and parameters of a multimedia session. Thus, SIP builds upon other IETF-developed protocols.

SIP is an application layer signaling protocol for session establishment. The main functions of this protocol are to establish a session, modifying the session, and terminate it when the call is to be finished. The sessions can be established with single or multiple participants. SIP is a simple text based protocol similar to HTTP (Hyper Text Transfer Protocol) and follows the client-server architecture. The transport protocol for SIP can be Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) or Stream Control Transmission Protocol (SCTP). Because of its simplicity, it is scalable in terms of the number of sessions and compatible with different protocol architectures. These are the reasons that SIP is becoming the industry standard for Voice over IP applications and products.

2.5.2 SIP Header format

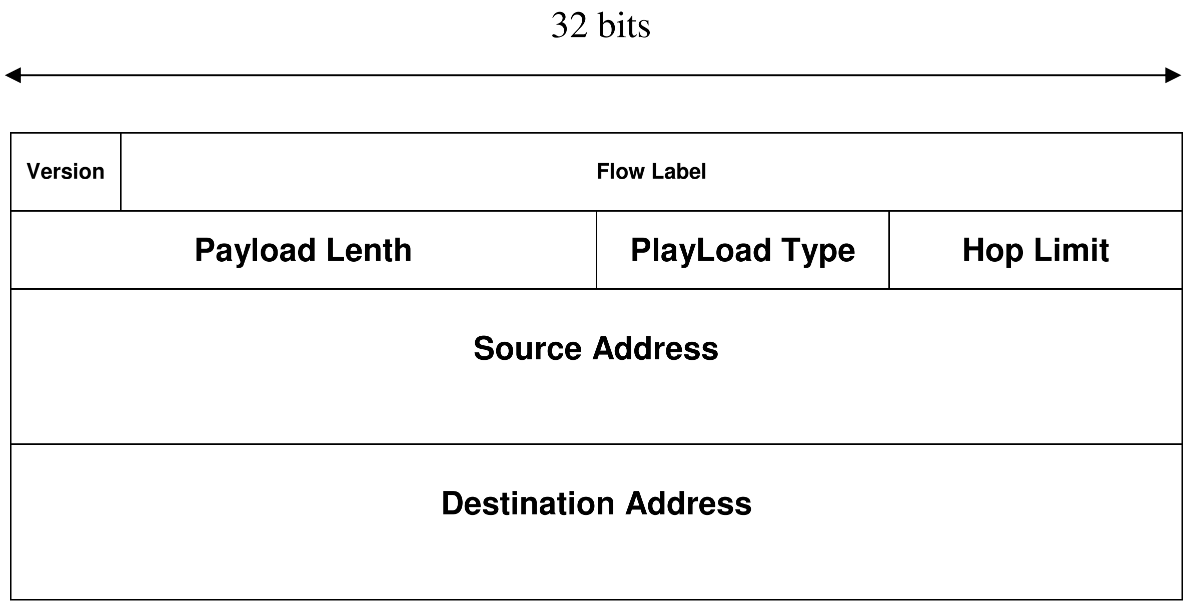


Figure:5 SIP Header Format

Version

The version field in SIP distinguishes SIP from IP and any other internet protocol that uses the same link layer frame as IP such as Stream Protocol.

Flow Label

The Flow Label is an expansion of the IP Service field. It is used to label packets as belonging to a particular traffic flow for which the server requires specific handling , for example real time service or non default quality of service

Payload Length

This specifies the length of the SIP packet. It does not include the header. This is a change from IP , where the receiver had to check that the payload length was not less than the header size , this is not necessary in SIP , and results in one less thing to be checked.

Payload Type

This uses the same values as the IP Protocol field. It specifies the type of the header immediately following the SIP header , such as TCP or UDP. It has been renamed to avoid confusion as to what is being referred to as the IP protocol - the protocol field or IP itself

Hop Limit

The Hop Limit is set to some nonzero value, and decremented by one by each system that forwards the packet. The packet is discarded if the hop limit reaches zero. This is to prevent the packet getting stuck in a forwarding loop.

Other uses include limiting the propagation of multicast packets, and it can also be used for diagnostic purposes. The "time to live" field in IP provided the same function, plus one extra one. This was to limit the amount of time that a packet spent in transit. This was discarded because it proved too costly to implement, and in some cases impossible to implement, for example in large subnets whose transit time is unpredictable. In practice many IP routers implemented time to live as hop limit, SIP legitimized this. Any higher level functions that cannot tolerate delivery delays, must provide their own method of recognizing old packets.

Source Address

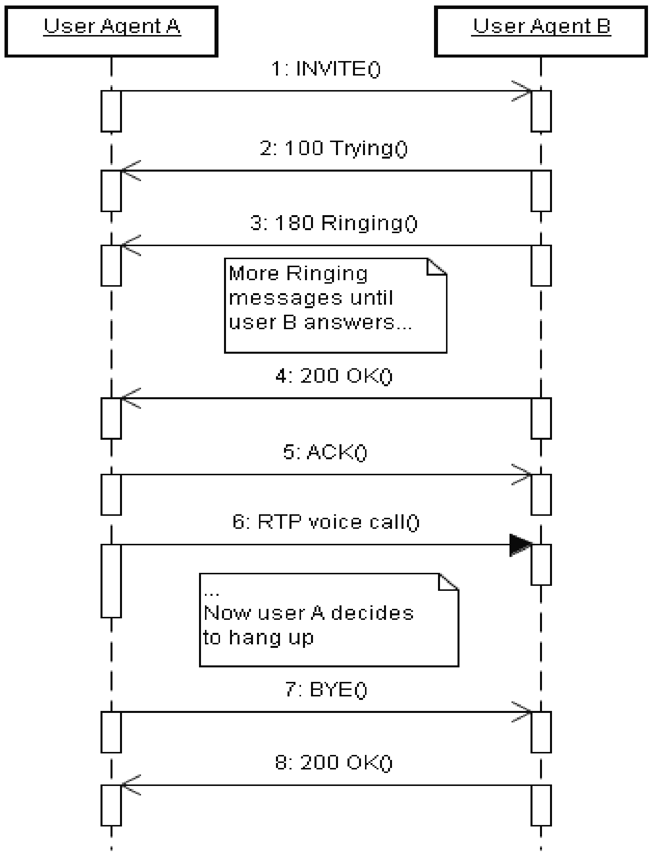
This is the 64 bit address of the sender. SIP addressing does not impose any strict class structure on addresses.

Destination Address

This is the 64 bit address of the destination. SIP addressing does not impose any strict class structure on addresses.

2.5.2 SIP call Flow

As discussed earlier SIP is Signaling Protocol. following diagram illustrate the basic sip call flow .first user agent A sends Invite message to the User Agent B. Then Trying message is replied by User Agent B .Ringing messages are sent until user B answers. It is followed by OK message. Now multimedia session has been established so real time communication using RTP protocol follows. For terminating the session bye message is invoked by any user. Bye message is followed by OK message



. **Figure:6** SIP Call Flow

There are five types of services that SIP offers,

User Location : To find the location of the end system for communication.

User Availability: To find if the called party is willing to communicate.

User Capabilities: To negotiate and determine the media capabilities, e.g. a voice codec that is supported by both calling party and the called party.

Call (session) Setup: Ringing and establishing call parameters at both called and calling party.

Session Management: The transfer and termination of the calls

SIP basically has two components

1. SIP User Agents
2. SIP Network Servers

The User agent is the component in the end system and consists of two parts:
(a) The client element called User Agent Client (UAC) used for call initiation;
(b) The server element called the User Agent Server (UAS) that is used to answer calls.

The SIP servers' functions include resolving the name and providing user locations, as end users usually don't know the IP address or the hostname of the called party.

Following are several examples of SIP servers:

Registrar server The registrar server receives Register requests from the users. The Register request associates the user’s SIP address called a SIP URI (Uniform Resource Identifier) with the current machine where they are located. This association is stored by the Registrar in the Location Service (LS). Authentication Of SIP user is required in case of registrar server system. How authentication is accomplished is shown in the following diagram**.**

SIP Authentication

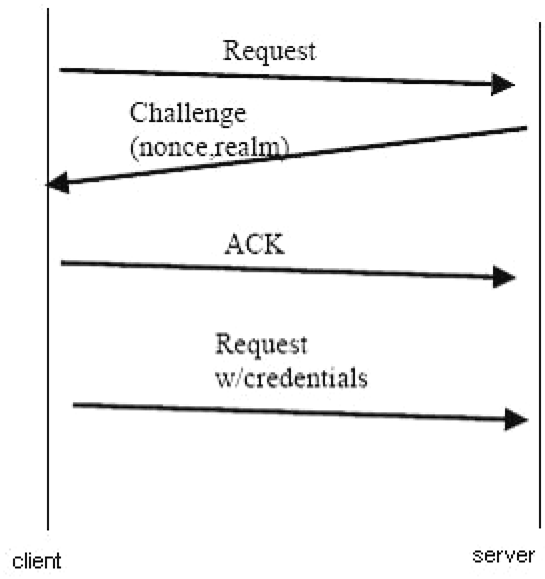


Figure:7 SIP Authentication

Proxy Server Users send their SIP requests to the Proxy Server, which forwards the requests to the next hop proxy server or to a proxy server close to the called user. The proxy server can also modify and add information in some parts of the SIP requests if required. A Domain Name System (DNS) Server can be used to find the location of the Proxy server.

Redirect Server The Redirect server receives the request from the clients, but unlike Proxy Servers, it does not forward the request to another server or the user. Rather, it sends back a response to the calling user with the information about the destination.

SIP Methods & Response Codes:

Methods:

INVITE	**Initiate Call**
ACK	**Confirm final response**
BYE	**Terminate and transfer call**
CANCEL	**Cancel searches and "ringing"**
OPTIONS	**Features support by other side**
REGISTER	**Register with location service**

Table:1 ***SIP Methods***

Response Codes:

1xx	**Searching ,ringing ,queuing**
2xx	**Success**
3xx	**Forwarding**
4xx	**Client mistakes**
5xx	**Server failures**
6xx	**Busy, refuse, not available anywhere**

Table:2 Response Codes

2.6 RTP

RTP [2] is the Internet-standard protocol for the transport of real-time data, including audio and video. It can be used for media-on-demand as well as interactive services such as Internet telephony. RTP consists of a data and a control part. The latter is called RTCP.

The data part of RTP is a thin protocol providing support for applications with real-time properties such as continuous media (e.g., audio and video), including timing reconstruction, loss detection, security and content identification. RTCP provides support for real-time conferencing of groups of any size within an internet. This support includes source identification and support for gateways like audio and video bridges as well as multicast-to-uni cast translators. It offers quality-of-service feedback from receivers to the multicast group as well as support for the synchronization of different media streams.

Following figure shows how the real time data is being wrapped when transported using RTP protocol.

MAC Header	IP Header	UDP Header	RTP Header	Payload

First the data is wrapped by RTP header then it is consider as payload for UDP .UDP Packet is then consider as payload of IP packet which is then wrapped by data link layer frame.

2.6.1 RTP Header format

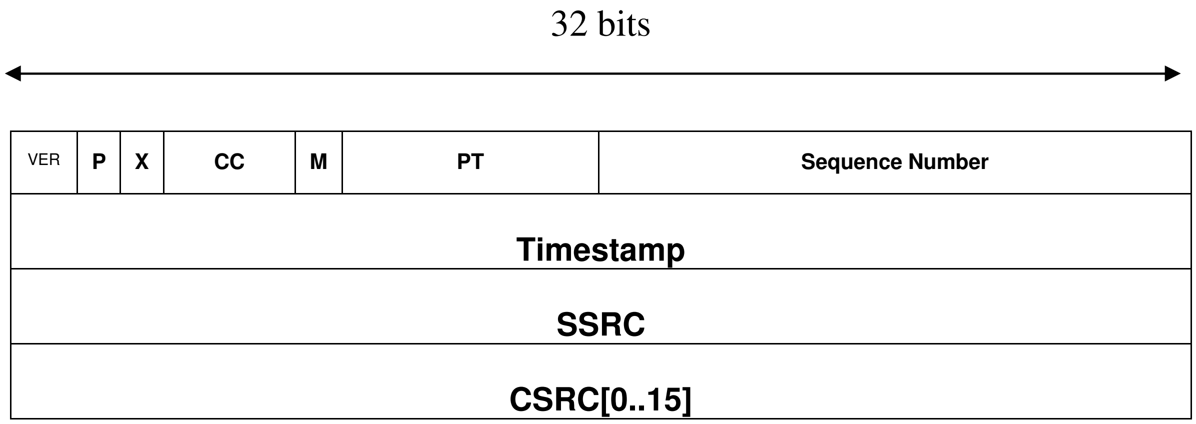


Figure:8 RTP Header

Ver, Version. 2 bits.

RTP version number. Always set to 2.

P, Padding. 1 bit.

If set, this packet contains one or more additional padding bytes at the end which are not part of the payload. The last byte of the padding contains a count of how many padding bytes should be ignored. Padding may be needed by some encryption algorithms with fixed block sizes or for carrying several RTP packets in a lower-layer protocol data unit.

X, Extension. 1 bit.

If set, the fixed header is followed by exactly one header extension.

CC, CSRC count. 4 bits.

The number of CSRC identifiers that follow the fixed header.

M, Marker. 1 bit.

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. A profile may define additional marker bits or specify that there is no marker bit by changing the number of bits in the payload type field.

PT, Payload Type. 7 bits.

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. 16 bits.

The 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. The initial value of the sequence number is random (unpredictable) to make known-plaintext attacks on encryption more difficult, even if the source itself does not encrypt, because the packets may flow through a translator that does.

Timestamp. 32 bits.

The 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. The resolution of the clock must be sufficient for the desired synchronization accuracy and for measuring packet arrival jitter (one tick per video frame is typically not sufficient).

SSRC, Synchronization source. 32 bits.

Identifies the synchronization source. The value is chosen randomly, with the intent that no two synchronization sources within the same RTP session will have the same *SSRC*. Although the probability of multiple sources choosing the same identifier is low, all RTP implementations must be prepared to detect and resolve collisions. If a source changes its source transport address, it must also choose a new *SSRC* to avoid being interpreted as a looped source.

CSRC, Contributing source. 32 bits.

An array of 0 to 15 CSRC elements identifying the contributing sources for the payload contained in this packet. The number of identifiers is given by the *CC* field. If there are more than 15 contributing sources, only 15 may be identified. CSRC identifiers are inserted by mixers, using the SSRC identifiers of contributing sources. For example, for audio packets the SSRC identifiers of all sources that were mixed together to create a packet are listed, allowing correct talker indication at the receiver

Following are the challenges for RTP

- **Multicast address allocation**:
  Eventually, many thousands of multicast sessions may exist concurrently. Currently, the IPv4 multicast address space is very limited and thus requires careful global allocation to avoid collisions.

The IPv6 multicast address space is very much larger, supports administrative scoping and may allow random allocation.

- **Scalable multicast routing**:
  Multicast routing needs to work for both a very large number of small groups and a smaller number of large groups, without routers not on the multicast tree having to know about groups.
- **Compensating for packet loss:**
  For the foreseeable future, the Internet will have areas and times of high packet loss (1% to 10% and higher).
- **Play out delay compensation:**
  End systems need to compensate for network delay variations.
- **Synchronization of different media:**
  Several audio and video streams coming from one or, less commonly, several sources need to be synchronized (lip sync).

2.7 OSS:

The Open Sound System (OSS) is a device driver for sound cards and other sound devices under various UNIX and UNIX-compatible operating systems. OSS was derived from the sound driver written for the Linux operating system kernel. The current version now runs on more than a dozen operating system platforms and supports most popular sound cards and sound devices integrated on computer motherboards. Sound cards normally have several different devices or ports which produce or record sound.

There are differences between various cards, but most have the devices described in this section.

The digitized voice device (also referred to as a codec, PCM, DSP or ADC/DAC device) is used for recording and playback of digitized sound.

The *mixer* device is used to control various input and output volume levels. The mixer device also handles switching of the input sources from microphone, line-level input and CD input.

The *synthesizer* device is used mainly for playing music. It is also used to generate sound effects in games. The OSS driver currently supports two kinds of synthesizer devices. The first is the Yamaha FM synthesizer chip which is available on most sound cards. The second types of synthesizer devices are the so-called wave table synthesizers. These devices produce sound by playing back pre-recorded instrument samples. This method makes it possible to produce extremely realistic instrument timbres. The Gravis Ultrasound (GF1) is an example of a wave table synthesizer.

A MIDI interface is a used to communicate with devices, such as synthesizers, that use the industry standard MIDI protocol. MIDI uses a serial interface running at 31.5 kbps which is similar to (but not compatible with) standard PC serial ports. The MIDI interface is designed to work with on-stage equipment like synthesizers, keyboards, stage props, and lighting controllers. MIDI devices communicate by sending messages through a MIDI cable. Most sound cards also provide a joystick port and some kind of interface (IDE, SCSI, or proprietary) for a CD-ROM drive. These devices are not controlled by OSS but there are typically separate drivers available.

2.7.1 OSS API Basics

The application programming interface (API) of the OSS driver is defined in the C language header file <soundcard.h> .

The OSS software ships with a copy of the header file in the include/sys subdirectory. You may have older versions of the include file that are included with your operating system (Linux distributions typically include the older OSS/Free driver, for example). It usually causes no harm to use the older header file but you will not be able to use some of the newer features only provided in OSS. Very old versions may also cause compatibility problems. To avoid this, you can either point to the OSS header files when compiling applications (use the compile option "-I/usr/lib/oss/include") or install the header file in a standard system header file location (e.g. /usr/include/sys).

If you get compile errors when building an application, verify that you are using the version of <soundcard.h> supplied with OSS.

2.7.2 Device Files Supported by OSS

The OSS driver supports several different types of devices. These are described in the following sections.

/dev/mixer

The mixer device files are used primarily for accessing the built-in mixer circuits of sound cards. A mixer makes it possible to adjust playback and recording levels of various sound sources. This device file is also used for selecting recording sources. Typically a mixer will control the output levels of the digital audio and FM synthesizer and also mix it with the CD input, line level input and microphone input sources. The OSS driver supports several mixers on the same system. The mixer devices are named /dev/mixer0, /dev/mixer1, etc. The device file /dev/mixer is a symbolic link to one of these device files (usually the first mixer, /dev/mixer0).

/dev/sndstat

This device file is provided for diagnostic purposes, and unlike all of the other sound devices, produces its output in human readable format. The device prints out information about all of the ports and devices detected by the OSS driver. Running the command "cat /dev/sndstat" will display useful information about the driver configuration. It should be noted that the output of /dev/sndstat is *not* intended to be machine readable and may change without notice in future Versions of OSS.

/dev/dsp and /dev/audio

These are the main device files for digitized voice applications. Any data written to this device is played back with the DAC/PCM/DSP device of the sound card. Reading the device returns the audio data recorded from the current input source (the default is the microphone input). The /dev/audio and /dev/dsp device files are very similar. The difference is that /dev/audio uses logarithmic mu-law encoding by default while /dev/dsp uses 8-bit unsigned linear encoding. With mu-law encoding a sample recorded with 12 or 16-bit resolution is represented by one 8-bit byte. Note that the initial sample format is the only difference between these device files. Both devices behave similarly after a program selects a specific sample encoding by calling ioctl. .

The OSS driver supports several codec devices on the same system. The audio devices are named /dev/dsp0, /dev/dsp1, etc. The file /dev/dsp is a symbolic link to one of these device files (usually /dev/dsp0). A similar naming scheme is used for /dev/audio devices.

/dev/sequencer

This device file is intended for electronic music applications. It can also be used for producing sound effects in games. The /dev/sequencer device provides access to any internal synthesizer devices of the sound cards. In addition, this device file can be used for accessing any external music synthesizer devices connected to the MIDI port of the sound card as well as General MIDI daughter cards connected to the Wave Blaster connector of many sound cards. The /dev/sequencer interface permits control of up to 15 synthesizer chips and up to 16 MIDI ports at the same time.

/dev/midi

These are low level interfaces to MIDI bus ports that work much like TTY (character terminal) devices in raw mode. The device files are not intended for real-time use % there is no timing capability so everything written to the device file will be sent to the MIDI port immediately. These devices are suitable for use by applications such as MIDI SysEx and sample librarians. There device files are named /dev/midi00, /dev/midi01, etc. (note the two digit device numbering). The device /dev/midi is a symbolic link to one of the actual device files (typically /dev/midi00).

3. Functional Specification:

For getting functional Specification for SIP soft phone in Simputer different open source soft phone were studied.

Minisip:

Minisip is a SIP User Agent ("Internet telephone").
It can be used to make phone calls, instant message and video calls to your buddies connected to the same SIP network.

Features:

- SIP compliant (RFC 3261 and more)
- Multiple lines (users) on the same phone
- Multiple incoming/outgoing calls simultaneously
- Runs on multiple Operating Systems (Linux PC, Linux familiar IPAQ PDA, Windows XP and soon Windows Mobile 2003 SE)
- Instant Messaging
- Video conferencing
- Call Logging

The source code is available as a number of libraries under the GNU Lesser General Public License (LGPL) and applications under the GNU General Public License (GPL).

SipXPhone:

The sipXphone project, formerly known as Pingtel's instant xpressa softphone, is a fully functional SIP softphone that runs on Microsoft Windows and Linux. The sipXphone project includes the Java Application Layer, C++ embedded web server, and a C/C++ JNI API.

Features:

- The SIP softphone leverages a well tested RFC 3261 compliant.
- Offers rich configuration abilities

- A java-based application framework
- Message Waiting Indication to on-phone bridged conferences

Linphone

Linphone is a web phone: it let you phone to your friends anywhere in the whole world, freely, simply by using the internet. The cost of the phone call is the cost that you spend connected to the internet.

Features:

Works with the Gnome Desktop under Linux,

- Works as simply as a cellular phone. Two buttons and one more to chat.
- Linphones includes a large variety of codecs (G711-ulaw, G711-alaw, LPC10-15, GSM, SPEEX and iLBC). The Speex codec it is able to provide high quality talks even with slow internet connections, like 28k modems.
- Understands the SIP protocol.
- Other technical functionalities include DTMF (dial tones) support though RFC2833 and ENUM support (to use SIP numbers instead of SIP addresses).
- Linphone is free software, released under the General Public License.
- Linphone includes a sip test server called "sipomatic" that automatically answers to calls by playing a pre-recorded message.

By above Study I found that Linphone has been better suitable for simputer which is linux based PDA. Linphone is purely in c/c++ and for Linux platform .

The Soft phone should provides the following capabilities:

- **Determines the location of the target endpoint**—SIP supports address resolution, name mapping, and call redirection.
- **Determines the availability of the target endpoint**—If a call cannot be completed because the target endpoint is unavailable, SIP determines whether the called party is connected to a call already or did not answer in the allotted number of rings. SIP then returns a message indicating why the target endpoint was unavailable.
- **Establishes a session between the originating and target endpoints**—If the call can be completed, SIP establishes a session between the endpoints.
- **User Capabilities:** To negotiate and determine the media capabilities, e.g. a voice codec that is supported by both calling party and the called party.

4. Devices and Tools:

4.1. Simputer

I have used Amida Simputer as my target device which has been designed and developed by PicoPeta, and manufactured by Bharat Electronics Limited. Simputer was configured to become part of our LAN.

Technical Specification

Hardware Specification	
processor	Intel StrongArm 206MHz SA-1100
Permanent Storage Memory	Intel Strata Flash 32MB
Synchronous DRAM	64MB RAM
Physical Dimension	142mm*72mm*20mm
Weight	206grms
Display Option	3.8 inch 240*320 color LCD Display
Audio Interface	Interface for headphone/microphone built-in microphone and speaker
Input Interface	3 Functional Key ,touch sensitive LCD display operated with Stylus
USB Interface	2 USB Ports, one in Master mode only, one with Master/Slave option
Connectivity	Optional external dial-up modem over serial port, connect through USB to select Wireless/Wired connectivity options, Bi-directional Infrared
Serial Interface	4-pin serial port interface
Smartcard Interface	Built-in reader/writer supports asynchronous
Power Specification	4.2V built-in lithium-ion battery,built-in charger 100-300V AC adapter with 5V/1A output

Software Specification	
Operation System	Linux Kernel 2.4.18 Device drivers, Customized boot loader
Environment	X Windows
Libraries	C libraries, Alchemy libraries
Simputer Application Interface	Alchemy
Application (optional)	Address Book, Notebook, Calculator, Clock, Calendar, Khatha, Voice recorder, Web browser

Table:3 Simputer Specification

4.2 Wireless Adapter:

Linksys WUSB54G has been used as Wireless Adapter. Linksys Wireless-G USB Network Adapter **(WUSB54G)** connects your USB-equipped desktop, PDA or notebook computer to a wireless network at incredible speeds. By incorporating two new, blazing fast technologies -- USB 2.0 and Wireless-G (802.11g) -- the Adapter delivers data rates up to 54Mbps (5 times as fast as 802.11b), without the trouble of opening up the case of your desktop computer.

The Wireless-G USB Network Adapter's high-gain antenna lets you put your computer almost anywhere in the building, without the cost and hassle of running cables.

WUSB54G Features:

- Standards IEEE 802.11b, 802.11g, USB 1.1, USB 2.0
- Port USB Port
- Channels 802.11b/802.11g
- LEDs Power, Link
- Transmit Power 15-17 dBm (Typical) @ 11Mbps
- Receive Sensitivity -65 dBm @ 54Mbps, -80 dBm @ 11Mbps
- Security Features WEP Encryption
- WEP Key Bits 64, 128-bit
- Operating Temp. 32ºF to 104ºF (0ºC to 40ºC)

4.3 USB to Ethernet Adapter:

It provides the way to add Ethernet network connectivity to your PC with this USB 10/100 Ethernet Adapter. With the simple, plug-and-play design, you don't have to bother with a difficult installation or worry whether or not you have an available expansion slot. It automatically transforms the USB port on your PC into a 10/100 Ethernet port.

General Features:

- Plug and Play Ethernet connectivity through USB port
- Bus/Interface Type: USB 1.1 USB 2.0
- Max USB Operation Speed : Full Speed USB
- Network Type: Ethernet
- LAN Connectors: Fast Ethernet 10/100BaseTX-RJ45
- Form Factor: External
- Native Transmit Speed(s): 10 Mbps
- Native Transmit Speed(s): 100 Mbps
- Single LED
- Bus powered

4.4 Ethereal:

Ethereal is the world's most popular network Analyzer. Ethereal is used by network professionals around the world for troubleshooting, analysis, software and protocol development, and education. Its open source allows talented experts in the networking community to add enhancements. It runs on all popular computing platforms, including UNIX, Linux, and Windows.

Features:

Ethereal has a comprehensive feature as listed below.

- Data can be captured "off the wire" from a live network connection, or read from a capture file.
- Ethereal can read capture files from tcpdump (libpcap), NAI's Sniffer™ (compressed and uncompressed), Sniffer™ Pro, NetXray™, Sun snoop and atmsnoop, Shomiti/Finisar Surveyor,

AIX's iptrace, Microsoft's Network Monitor, Novell's LANalyzer, RADCOM's WAN/LAN Analyzer, Live data can be read from Ethernet, FDDI, PPP, Token-Ring, IEEE 802.11, Classical IP over ATM.

- Captured network data can be browsed via a GUI, or via the TTY-mode "tethereal" program.
- Capture files can be programmatically edited or converted via command-line switches to the "editcap" program.
- 750 protocols can currently be dissected
- Output can be saved or printed as plain text or PostScript.
- Data display can be refined using a display filter.
- Display filters can also be used to selectively highlight and color packet summary information.
- All or part of each captured network trace can be saved to disk.

4.5 Building cross-compiling tool chain

I have been working on the Linux Base Workstation so I need to cross compile everything like libraries, device drivers, sip system code .So first of all I have to build the cross compiler tool chain. We have different types of build-in tool chains available .But in build-in tool chain there are chance of incompatibly with our target system environment . So I have built my own tool chain as follows

Required sources

How to build a cross compiling tool chain from the ground up is explained below. .You need the following source packages to go on:

Packages:

- binutils
- Linux Kernel
- ARM Kernel patch

- gcc
- glibc

binutils:

Unpack the binutils tarball into a temporary directory, change to the unpacked binutils directory and run the following commands:

```
# ./configure --target=arm-linux
# make
# make install
```

You have now some arm-linux-* binaries in /usr/local/bin. These are the binutils used by the cross-compiling tool chain. And you'll find the new directory /usr/local/arm-linux/. This is where the cross-compiling tool chain will be installed.

You can check if the binutils are compiled correctly by calling arm-linux-ar.

Linux Kernel header files:

To compile gcc we need some header files from the linux kernel source. Unpack the kernel source code in a temporary directory and change to the unpacked source directory. You'll need to patch the kernel with the ARM kernel patch. You do this by running this command:

```
# zcat  path-to-arm-patch/patch-2.4.17-rmk4.gz | patch -p1
```

Now you need to configure the kernel by calling this command:

```
# make menuconfig ARCH=arm
```

Notice that you need to specify ARCH=arm otherwise you are going to configure the kernel for your host architecture which maybe a x86 machine.

You don't need to do a complete configuration unless you want to compile the kernel now. Up to now you don't have a cross compiler so you can't compile it anyway. All you need to do is to select the correct processor type. In my case I have selected *SA1100-based* in ARM system type Implementations. Now save the configuration and call the following command to finish the kernel configuration:

```
# make dep
```

Now copy the include files from the kernel source to the tool chain directory:

```
# mkdir /usr/local/arm-linux/include

# cp -dR include/asm-arm /usr/local/arm-linux/include/asm

# cp -dR include/linux /usr/local/arm-linux/include/linux
```

Finally change to the toolchain directory and create a symbolic link from *include* to *sys-include*:

```
# cd /usr/local/arm-linux/

# ln -s include sys-linux
```

gcc, which we will compile now, is searching for the include files in *sys-linux* by default. You can use the --with-headers configure-option to specify an other directory but this results in copying the specified directory to sys-linux. So I think it's better to create a symbolic link to avoid redundant files.

gcc

Unpack the gcc source code and change to the unpacked source directory. We currently don't have a running glibc so we can't compile the whole compiler suite. But for now it is enough to compile only the C compiler. Later we can compile the glibc with this cross compiler and after that we can compile the whole compiler suite.

It may be necessary to modify the gcc source a little bit. I have done this because otherwise I was not able to compile, I got these error messages:

```
./libgcc2.c:41: stdlib.h: No such file or directory

./libgcc2.c:42: unistd.h: No such file or directory
.make[3]: *** [libgcc2.a] Error 1
```

There are rumors that it is not always needed. If you think (or know, or learn) that you need it, edit the file gcc/config/arm/t-linux, search this line:

```
TARGET_LIBGCC2_CFLAGS = -fomit-frame-pointer -fPIC
```

And change it to this:

```
TARGET_LIBGCC2_CFLAGS = -fomit-frame-pointer -fPIC -Dinhibit_libc -
D__gthr_posix_h
```

Now configure the source code, compile and install:

```
# ./configure --target=arm-linux --disable-threads --enable-languages=c

# make

# make install
```

You have now a running cross compiler (/usr/local/bin/arm-linux-gcc) but without glibc it is not really useful. So let's cross-compile glibc.

glibc

Unpack the glibc tarball in a temporary directory as usual. Then switch to the unpacked source directory and unpack the linuxthreads add-on into it:

```
# tar xvfz glibc-2.2.4.tar.gz

# cd glibc-2.2.4

# tar xvfz ../glibc-linuxthreads-2.2.4.tar.gz
```

Now set the environment variable CC to *arm-linux-gcc* because we want the glibc to be cross-compiled for the ARM platform. Then configure, compile and install the beast:

```
# export CC=arm-linux-gcc

# ./configure arm-linux --target=arm-linux --prefix=/usr/local/arm-linux --enable-
add-ons

# make

# make install
```

Be sure you use the --prefix parameter correctly, otherwise you mess up your hosts glibc installation.

You'll now find a lot of new files and directories in /usr/local/arm-linux. These are the glibc headers, libraries and utitilies.

Notice that you can't use this compiled glibc on the target machine because of the specified prefix. If you want to compile a glibc which you can copy to your target machine, use an empty prefix (--prefix=) instead and use the install_root parameter to specify the installation directory:

```
# make install install_root=/path/to/target/root
```

Finally, make sure you unset the CC environment variable (with unset CC), because in the next step we are going to recompile the cross compiler and we don't want to cross-compile the cross-compiler. ;-)

gcc (Second time)

Now we have a cross compiled glibc so we can now go on and compile the whole gcc compiler suite.

You can use the already unpacked source code of gcc but you have to remove the changes you have made and you should call make distclean to clean up the source. To be sure to do it right I suggest you delete the old source directory and unpack the gcc sources again. Whatever, after you have a clean gcc source directory, change into it, configure the source, compile and install it:

```
# ./configure --target=arm-linux –enable-languages=c,c++
# make
# make install
```

If compilation fails because *PATH_MAX* is undeclared in basicio.c then add the following line to the file libchill/basicio.c somewhere between all the other includes at the top of the file:

```
#include <linux/limits.h>
```

Call make again and it should compile fine.
Now We have a working cross-compile tool chain for the ARM platform.

5. Implementation Details:

Host System: i686- based Linux workstation

Target System: SA1100- Based Simputer

The tool chain was tested by simple client-server program but there was a error regarding c++ shared libraries. So the stdlibc++.so was replaced in /usr/lib directory of simputer by compatible version from cross compiled libraries. Now we could run simple client-server program successfully.

For development of sip based system as discussed in background we required signaling, codec, real-time communication session establishment .so for that I have used downloaded, optimized and configure following libraries. First of all set up the build environment:

```
#export PATH=$PATH:/root/usr/local/arm/bin

#export LD=/root/usr/local/arm/bin/arm-linux-ld

#export LDFLAGS=-L/root/usr/local/arm/arm-linux/lib
```

5.1 OSIP LIBRARY:

OSIP stands for Open Session Initiation Protocol .OSIP is an implementation of SIP protocol stack. This library aims to provide multimedia and telecom software developers an easy and powerful interface to initiate and control SIP based sessions. OSIP is little in size and code and thus could be use to implement IP soft-phone as well as embedded SIP software. OSIP can also be used to implement "SIP proxy".

Cross -compiling libosip2-2.2.2

```
 #cd libosip2-2.2.2
 #./configure --prefix=/root/armbuild -host=i686-pc-linux
   --target=arm-linux --disable-static
 #make
 #make install DESTDIR=$ARM_INSTALL
```

1.Initialize Osip2

When using osip, your first task is to initialize the parser and the state machine. This must be done prior to any use of libosip2.

```
#include <sys/time.h>
#include <osip2/osip.h>
int i;
osip_t *osip;
i=osip_init(&osip);
if (i!=0)
  return -1;
```

For the URI parser, the API is documented in osip_uri.h

How to parse URI

Here is the sequence needed to parse a given buffer containing a sip URI:

```
osip_uri_t *uri;
   int i;
   i=osip_uri_init(&uri);
   if (i!=0) { fprintf(stderr, "cannot allocate\n"); return -1; }
   i=osip_uri_parse(uri, buffer);
   if (i!=0) { fprintf(stderr, "cannot parse uri\n"); }
   osip_uri_free(uri);
```

How to Parse SIP message

SIP parser, the API is documented in osip_message.h

Here is the sequence needed to parse a given buffer containing a sip request or response. Because the SIP message can contains binary data in its body part, the length of the buffer must be given to the API.

```
osip_message_t *sip;
   int i;
   i=osip_message_init(&sip);
```

```
if (i!=0) { fprintf(stderr, "cannot allocate\n"); return -1; }
i=osip_message_parse(sip, buffer, length_of_buffer);
if (i!=0) { fprintf(stderr, "cannot parse sip message\n"); }
osip_message_free(sip);
```

How to Manage Trasaction

As soon as you have build the SIP message, you are ready to start a new transaction. Here is the code:

```
osip_t *osip       = your_global_osip_context;
osip_transaction_t *transaction;
osip_message_t     *sip_register_message;
osip_event_t       *sipevent;
application_build_register(&sip_register_message);
osip_transaction_init(&transaction,
          ICT, //a REGISTER is an Invite-Client-Transaction
          osip,
          sip_register_message);
```

How to Manage Dialogs

There is two ways of creating a dialog. In one case, you are the CALLER and in the other case, you will be the CALLEE.

Creating a dialog as a CALLER.

In this case, you have to create a dialog each time you receive an answer with a code between 101 and 299. The best place in OSIP to actually create a dialog is of course in the callback that announce such SIP messages. Of course, each time you receive a response, you have to check for an existing dialog associated to this INVITE that can have been created by earlier SIP answer coming from the same User Agent. The code in the callback will look like the following:

```
void cb_rcv1xx(osip_transaction_t *tr,osip_message_t *sip)
          {
```

```
                    osip_dialog_t *dialog;
                    if (MSG_IS_RESPONSEFOR(sip,
"INVITE")&&!MSG_TEST_CODE(sip, 100))
                    {
                              dialog =
my_application_search_existing_dialog(sip);
                              if (dialog==NULL) //NO EXISTING DIALOG
                              {
                                        i = osip_dialog_init_as_uac(&dialog, sip);

          my_application_add_existing_dialog(dialog);
                              }
                    }
                    else
                    {
                              // no dialog establishment for other REQUEST
                    }
          }
```

Creating a dialog as a CALLEE

In this case, you will have to create a dialog upon receiving the first transmission of the INVITE request. The correct place to do that is inside the callback previously registered to announce new INVITE.

You will build a SIP answer like 180 or 200 and you'll be able to create a dialog by calling the following code:

```
 osip_dialog_t *dialog;

 osip_dialog_init_as_uas(&dialog, original_invite, response_that_you_build);
```

How to Use SDP Negotiator.

Here is the way to initialize the new negotiator:

```
          struct osip_rfc3264 *cnf;
```

```
int i;
i = osip_rfc3264_init(&cnf);
if (i!=0)
{
        fprintf(stderr, "Cannot Initialize Negotiator feature.\n");
        return -1;
}
```

5.3 Speex Library:

Speex is open source speech codec. Speex is not targeted at cell phones but rather at voice over IP (VOIP) and file-based compression. As design goals, we wanted to have a codec that would allow both very good quality speech and low bit-rate (unfortunately not at the same time!), which led us to developing a codec with multiple bit-rates. Of course very good quality also meant we had to do wideband (16 kHz sampling rate) in addition to narrowband (telephone quality, 8 kHz sampling rate).

Designing for VOIP means that Speex must be robust to lost packets, but not to corrupted ones since packets either arrive unaltered or don't arrive at all. Also, the idea was to have a reasonable complexity and memory requirement without compromising too much on the efficiency of the codec.

All this led us to the choice of CELP as the encoding technique to use for Speex. One of the main reasons is that CELP has long proved that it could do the job and scale well to both low bit-rates.

Introduction to CELP Coding

Speex is based on CELP, which stands for Code Excited Linear Prediction..

Linear Prediction (LPC)

Linear prediction is at the base of many speech coding techniques, including CELP. The idea behind it is to predict the signal **x[n]** using a linear combination of its past samples:

$$y[n] = \sum_{i=1}^{N} a_i x[n-i]$$

where **y[n]**is the linear prediction of **y[n]**. The prediction error is thus given by:

$$e[n] = x[n] - y[n] = x[n] - \sum_{i=1}^{N} a_i x[n-i]$$

The main characteristics can be summarized as follows:

Speex Library features description

This section explains the main Speex features, as well as some concepts in speech coding

Sampling rate

Speex is mainly designed for 3 different sampling rates: 8 kHz, 16 kHz, and 32 kHz. These are respectively referred to as narrowband, wideband and ultra-wideband.

Quality

Speex encoding is controlled most of the time by a quality parameter that ranges from 0 to 10. In constant bit-rate (CBR) operation, the quality parameter is an integer, while for variable bit-rate (VBR), the parameter is a float.

Complexity (variable)

With Speex, it is possible to vary the complexity allowed for the encoder. This is done by controlling how the search is performed with an integer ranging from 1 to 10 in a way that's similar to the -1 to -9 options to *gzip* and *bzip2* compression utilities. For normal use, the noise level at complexity 1 is between 1 and 2 dB higher than at complexity 10, but the CPU requirements for complexity 10 is about 5 times higher than for complexity 1. In practice, the best trade-off is between complexity 2 and 4, though higher settings are often useful when encoding non-speech sounds like DTMF tones.

Variable Bit-Rate (VBR)

Variable bit-rate (VBR) allows a codec to change its bit-rate dynamically to adapt to the ``difficulty'' of the audio being encoded. In the example of Speex, sounds like vowels and high-energy transients require a higher bit-rate to achieve good quality, while fricatives (e.g. s,f sounds) can be coded adequately with less bits. For this reason, VBR can achieve lower bit-rate for

the same quality, or a better quality for a certain bit-rate. Despite its advantages, VBR has two main drawbacks: first, by only specifying quality, there's no guaranty about the final average bit-rate. Second, for some real-time applications like voice over IP (VoIP), what counts is the maximum bit-rate, which must be low enough for the communication channel.

Average Bit-Rate (ABR)

Average bit-rate solves one of the problems of VBR, as it dynamically adjusts VBR quality in order to meet a specific target bit-rate. Because the quality/bit-rate is adjusted in real-time (open-loop), the global quality will be slightly lower than that obtained by encoding in VBR with exactly the right quality setting to meet the target average bit-rate.

Voice Activity Detection (VAD)

When enabled, voice activity detection detects whether the audio being encoded is speech or silence/background noise. VAD is always implicitly activated when encoding in VBR, so the option is only useful in non-VBR operation. In this case, Speex detects non-speech periods and encode them with just enough bits to reproduce the background noise. This is called "comfort noise generation'' (CNG).

Discontinuous Transmission (DTX)

Discontinuous transmission is an addition to VAD/VBR operation, that allows to stop transmitting completely when the background noise is stationary. In file-based operation, since we cannot just stop writing to the file, only 5 bits are used for such frames (corresponding to 250 bps).

Perceptual enhancement

Perceptual enhancement is a part of the decoder which, when turned on, tries to reduce (the perception of) the noise produced by the coding/decoding process. In most cases, perceptual enhancement make the sound further from the original *objectively* (if you use SNR), but in the end it still *sounds* better (subjective improvement).

Algorithmic delay

Every speech codec introduces a delay in the transmission. For Speex, this delay is equal to the frame size, plus some amount of ``look-ahead'' required to process each frame. In narrowband operation (8 kHz), the delay is 30 ms, while for wideband (16 kHz), the delay is 34 ms. These values don't account for the CPU time it takes to encode or decode the frames.

speexenc

The *speexenc* utility is used to create Speex files from raw PCM or wave files. It can be used by calling:

```
 speexenc [options] input_file output_file
```

The value '-' for input_file or output_file corresponds respectively to stdin and stdout. The valid options are:

-narrowband (-n)

Tell Speex to treat the input as narrowband (8 kHz). This is the default

-wideband (-w)

Tell Speex to treat the input as wideband (16 kHz)

-ultra-wideband (-u)

Tell Speex to treat the input as ``ultra-wideband'' (32 kHz)

-quality n

Set the encoding quality (0-10), default is 8

-bitrate n

Encoding bit-rate (use bit-rate n or lower)

-vbr

Enable VBR (Variable Bit-Rate), disabled by default

-abr n

Enable ABR (Average Bit-Rate) at n kbps, disabled by default

-vad

Enable VAD (Voice Activity Detection), disabled by default

-dtx

Enable DTX (Discontinuous Transmission), disabled by default

Raw input options

-rate n

Sampling rate for raw input

-le

Raw input is little-endian

-be

Raw input is big-endian

-8bit

Raw input is 8-bit unsigned

-16bit

Raw input is 16-bit signed

speexdec

The *speexdec* utility is used to decode Speex files and can be used by calling:

```
speexdec [options] speex_file [output_file]
```

The value for input_file or output_file corresponds respectively to stdin and stdout. Also, when no output_file is specified, the file is played to the soundcard. The valid options are:

-enh

enable post-filter (default)

-no-enh

disable post-filter

-force-nb

Force decoding in narrowband

-force-wb

Force decoding in wideband

-force-uwb

Force decoding in ultra-wideband

-mono

Force decoding in mono

-stereo

Force decoding in stereo

-rate n

Force decoding at n Hz sampling rate

Here Ogg audio compression library is used with Speex .So it is required to cross compile libogg first.

Cross -compiling libogg-1.1.0

```
#cd ../libogg-1.1.0

#./configure --prefix=/root/armbuild --host=i686-pc-linux
 --target=arm-linux --disable-static --enable-fixed-point

#make

#make install DESTDIR=$ARM_INSTALL
```

Now I needed to cross compiling Speex.

Cross -compiling speex-1.1.11.1

```
#cd ../speex-1.1.11.1

#./configure --prefix=/root/armbuild --host=i686-pc-linux
 --target=arm-linux --disable-static
 --enable-fixed-point --enable-arm-asm
 --with-ogg=/root/armbuild/usr
 --with-ogg-ibraries=/roo/armbuild/usr/lib
 --with-ogg-headers=/root/armbuild/usr/include/ogg
```

```
#make
#make install DESTDIR=$ARM_INSTALL
```

Copy "libspeex.so.2.0.0" to your arm-tools.

```
#cp /root/armbuild/usr/lib/libspeex.so.2.0.0
   /root/usr/local/arm/3.4.1/arm-linux/lib
#cd /root/usr/local/arm/3.4.1/arm-linux/lib
#ln -s libspeex.so.2.0.0 libspeex.so

#ln -s libspeex.so.2.0.0 libspeex.so.2
```

5.4 ORTP Library:

ORTP stands for Open Real Time Protocol. ORTP Implements the RFC3550 (RTP) with easy to use API with high and low level access. It provides blocking and non blocking IO for RTP sessions.

ORTP API:

- Initialize the oRTP library. You should call this function first before using oRTP API

```
void    ortp_init ();
```

- Initialize the oRTP scheduler. You only have to do that if you intend to use the scheduled mode of the RtpSession in your application.

```
void      ortp_scheduler_init ();
```

- Uninitialize the library, including shutdowning the scheduler if it was started.

```
void  ortp_exit ();
```

- for creating new Rtp Session following function is used

```
RtpSession* rtp_session_new (int mode) //create new rtp session;
```

- There are different mode for rtp session as below.

```
enum RtpSessionMode
typedef enum {
          RTP_SESSION_RECVONLY,
          RTP_SESSION_SENDONLY,
          RTP_SESSION_SENDRECV
            } RtpSessionMode
```

- for setting jitter value following function is used.

```
void  rtp_session_set_jitter_compensation (RtpSession *session,
                                                    int milisec );
```

cross compiling the ORTP library.

```
#cd oRTP

#./configure --prefix=/root/armbuild --host=i686-pc-linux
  --target=arm-linux --disable-static --enable-fixed-point

#make

#make install DESTDIR=$ARM_INSTALL
```

5.5 Ncurses Library:

Ncurses library is a library of functions that manage an application's display on character-cell terminals. It provides terminal emulators. Readline library uses Ncureses. The Curses library forms a wrapper over working with raw terminal codes, and provides highly flexible and efficient API. We have to include this library by ncurses.h file. To link the program with ncurses the flag -lncurses should be added.

```
gcc <program file> -lncurses
```

The Ncurses library was downloaded and configured   as follows.

```
#./configure --prefix=/root/armbuild -host=i686-pc-linux --target=arm-linux
--enable-shared


#make

#make install DESTDIR=$ARM_INSTALL
```

Initialization functions

initscr( )

The function initscr() initializes the terminal in curses mode. In some implementations, it clears the screen and presents a blank screen. To do any screen manipulation using curses package this has to be called first. This function initializes the curses system and allocates memory for our present window and some other data-structures

raw( ) and cbreak( )

Normally the terminal driver buffers the characters a user types until a new line or carriage return is encountered. But most programs require that the characters be available as soon as the user types them. The above two functions are used to disable line buffering. The difference between these two functions is in the way control characters like suspend (CTRL-Z), interrupt and quit (CTRL-C) are passed to the program. In the raw() mode these characters are directly passed to the program without generating a signal. In the cbreak() mode these control characters are interpreted as any other character by the terminal driver. I personally prefer to use raw() as I can exercise greater control over what the user does.

echo() and noecho()

These functions control the echoing of characters typed by the user to the terminal. noecho() switches off echoing. The reason you might want to do this is to gain more control over echoing or to suppress unnecessary echoing while taking input from the user through the getch() etc. functions. Most of the interactive programs call noecho() at initialization and do the echoing of characters in a controlled manner. It gives the programmer the flexibility of echoing characters at any place in the window without updating current (y,x) co-ordinates.

keypad( )

This is my favorite initialization function. It enables the reading of function keys like F1, F2, arrow keys etc. Almost every interactive program enables this, as arrow keys are a major part of any User Interface. Do keypad(stdscr, TRUE) to enable this feature for the regular screen (stdscr).

endwin( )

And finally don't forget to end the curses mode. Otherwise your terminal might behave strangely after the program quits. endwin() frees the memory taken by curses sub-system and its data structures and puts the terminal in normal mode. This function must be called after you are done with the curses mode .

5.6 Readline Library:

The GNU Readline library provides a set of functions for use by applications that allow users to edit command lines as they are typed in. Both Emacs and vi editing modes are available. The Readline library includes additional functions to maintain a list of previously-entered command lines, to recall and perhaps reedit those lines, and perform csh-like history expansion on previous commands. The history facilites are also placed into a separate library, the History library, as part of the build process.

There is a support for building shared versions of the Readline and History libraries. The configure script creates a Makefile in the shlib' subdirectory, and typing `make shared'will cause shared versions of the Readline and History libraries to be built on supported platforms.

If `configure' is given the `--enable-shared' option, it will attempt to build the shared libraries by default on supported platforms. I have configured and built as follow.

```
#./configure --prefix=/root/armbuild --host=i686-pc-linux
  --target=arm-linux enable-shared
make
make install DESTDIR=$ARM_INSTALL
```

5.7 SIP Based Application:

Linphone has been used as a sip based softphone. The code has been optimized ,configured and cross complied . The console mode has been used so Gnome supported GUI was remove from it and added console mode support because Standard Gnome based is not supported in Simputer. If we want GUI support for softphone in simputer then Amida Software Development Kit will be required but It is not freely available. I have written the configuration script required for configuring softphone for Simputer. The configured script is simputer-config.site

```
#export CONFIG_SITE=~/simputer-config.site
```

There are two type of sound card device drivers ALSA and OSS .But in simputer OSS device driver is there. SO configured soft phone to use OSS device driver.

```
#cd /root/arm/linphone-1.2.0

#./configure --prefix=/root/armbuild
--host=i686-pc-linux --target=arm-linux
--disable-static --disable-glib
--enable-gnome_ui=no --disable-manual
--enable-oss
```

```
--with-readline=$ARM_INSTALL_TREE/usr
--with-osip=/root/armbuild/usr
--with-speex=/root/armbuild/usr

#make

#make install  DESTDIR=$ARM_INSTALL
```

So this way whole required libraries and sip soft phone were build in Linux work station. After that whole build code was ported in the simputer. Sip soft phones were run between a Simputer and a Linux work station successfully.

6 . Results:

To run the soft phone application between a simputer and a pc.
in Pc give the command:

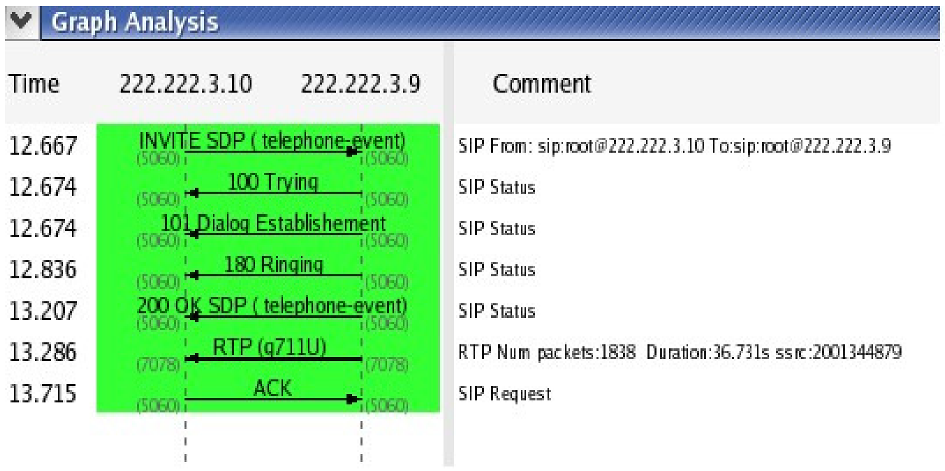
./linphonec answer //application is in receiving mode.

in simputer I gave:

./linphonec –s sip:root@222.222.3.9

Then the call flow is analyzed using Ethereal. SIP by default is running on port 5060.

6.1 SIP Call Flow Analysis



```
Graph Analysis
Time     222.222.3.10                          222.222.3.9    Comment
12.667   INVITE SDP ( telephone-event)         (5060)->(5060)  SIP From: sip:root@222.222.3.10 To:sip:root@222.222.3.9
12.674   100 Trying                            (5060)<-(5060)  SIP Status
12.674   101 Dialog Establishement             (5060)<-(5060)  SIP Status
12.836   180 Ringing                           (5060)<-(5060)  SIP Status
13.207   200 OK SDP ( telephone-event)         (5060)<-(5060)  SIP Status
13.286   RTP (g711U)                           (7078)<-(7078)  RTP Num packets:1838  Duration:36.731s ssrc:2001344879
13.715   ACK                                   (5060)->(5060)  SIP Request
```

Figure:9 Sip Call Flow

INVITE Message Header

Request-Line: INVITE sip:root@222.222.3.9 SIP/2.0

Via: SIP/2.0/UDP 222.222.3.10:5060; branch=z9hG4bK776asdhds

To: sip:root@222.222.3.9

From:sip:root@222.222.3.10;tag=1928301774

Call-ID: 221583024@222.222.3.10

```
CSeq: 20 INVITE
Max-forward:5
Contact: <sip:root@222.222.3.10:5060
Content-Type: application/sdp
Content-Length: 142
User-Agent:Linphone-1.3
```

The first line of the text-encoded message contains the method name(INVITE).

- Via contains the address at which sip:root is expecting to receive responses to this request. It also contains a branch parameter that identifies this transaction

- To contains a display name (Bob) and a SIP or SIPS URI From also contains a display name (Alice) and a SIP or SIPS URI

- Call-ID contains a globally unique identifier for this call, generated by the combination of a random string and the softphone's host name or IP address. The combination of the To tag, From tag, and Call-ID completely defines a peer-to-peer SIP relationship .

- CSeq or Command Sequence contains an integer and a method name. The CSeq number is incremented for each new request within a dialog and is a traditional sequence number.

- Max-Forwards serves to limit the number of hops a request can make on the way to its destination. It consists of an integer that is decremented by one at each hop.

- Contact contains a SIP or SIPS URI that represents a direct route to Sip:root@222.222.3.9, usually composed of a username at a fully qualified domain name (FQDN). While an FQDN is preferred, many end systems do not have registered domain names, so IP addresses are permitted. While the Via header field tells other elements where to send the response, the Contact header field tells other elements where to send future requests.

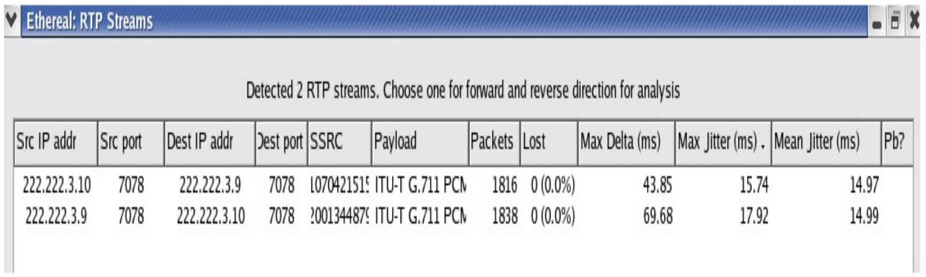
- Content-Type contains a description of the message body (not shown).

- Content-Length contains an octet (byte) count of the message body.

Other messages Trying, Ringing, Dialog establishment, OK are status Messages. So only Status-line instead of Request-line in Message is changed . ACK message is also Request Line type. Others field except content length are rarely changed.

6.2 RTP Streams :

It is two ways communication so here I have detected two RTP streams one from 222.222.3.10 to 222.222.3.9 and other from 222.222.3.9 to 222.222.3.10.I got Mean jitter around 15 ms. Here RTP is running on port 7078.



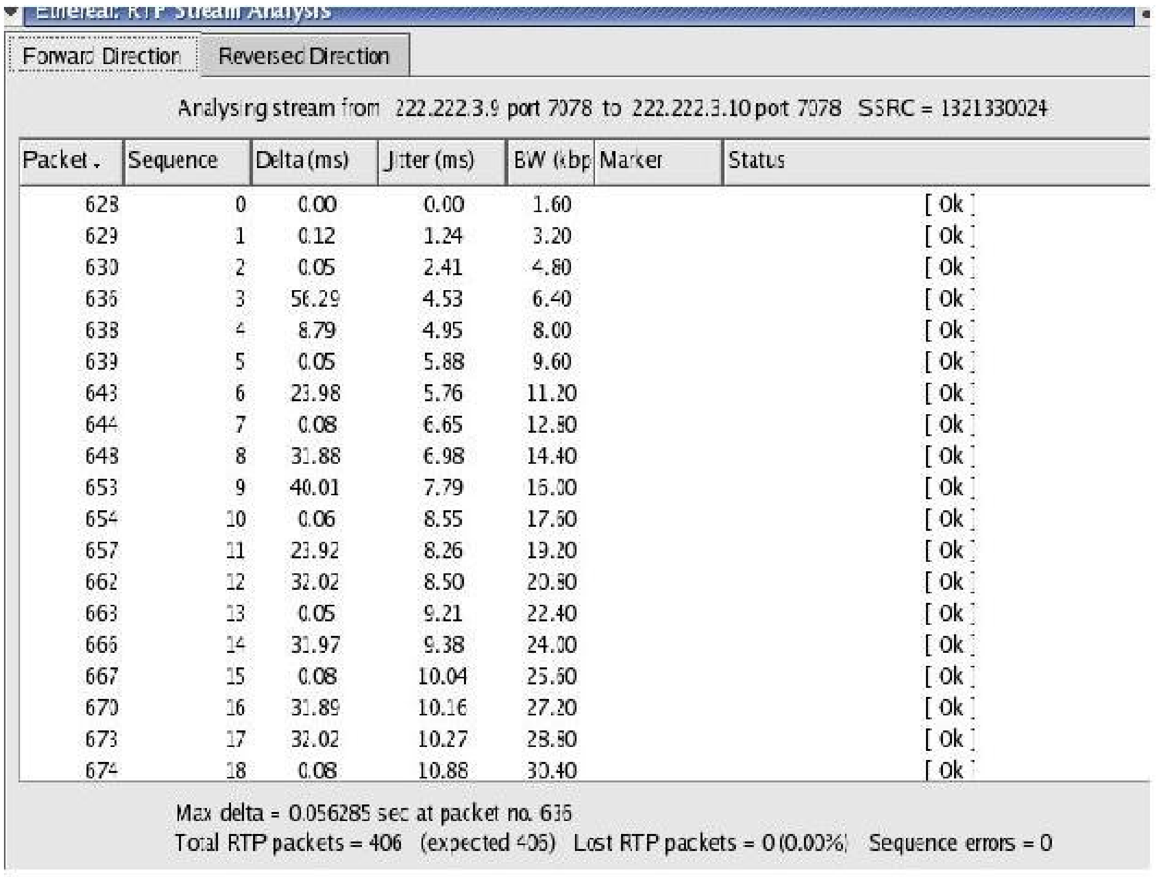
Ethereal: RTP Streams

Detected 2 RTP streams. Choose one for forward and reverse direction for analysis

Src IP addr	Src port	Dest IP addr	Dest port	SSRC	Payload	Packets	Lost	Max Delta (ms)	Max Jitter (ms)	Mean Jitter (ms)	Pb?
222.222.3.10	7078	222.222.3.9	7078	107042151	ITU-T G.711 PCM	1816	0 (0.0%)	43.85	15.74	14.97	
222.222.3.9	7078	222.222.3.10	7078	200134487	ITU-T G.711 PCM	1838	0 (0.0%)	69.68	17.92	14.99	

Figure:10

6.3 RTP Stream Analysis



Analysing stream from 222.222.3.9 port 7078 to 222.222.3.10 port 7078 SSRC = 1321330024

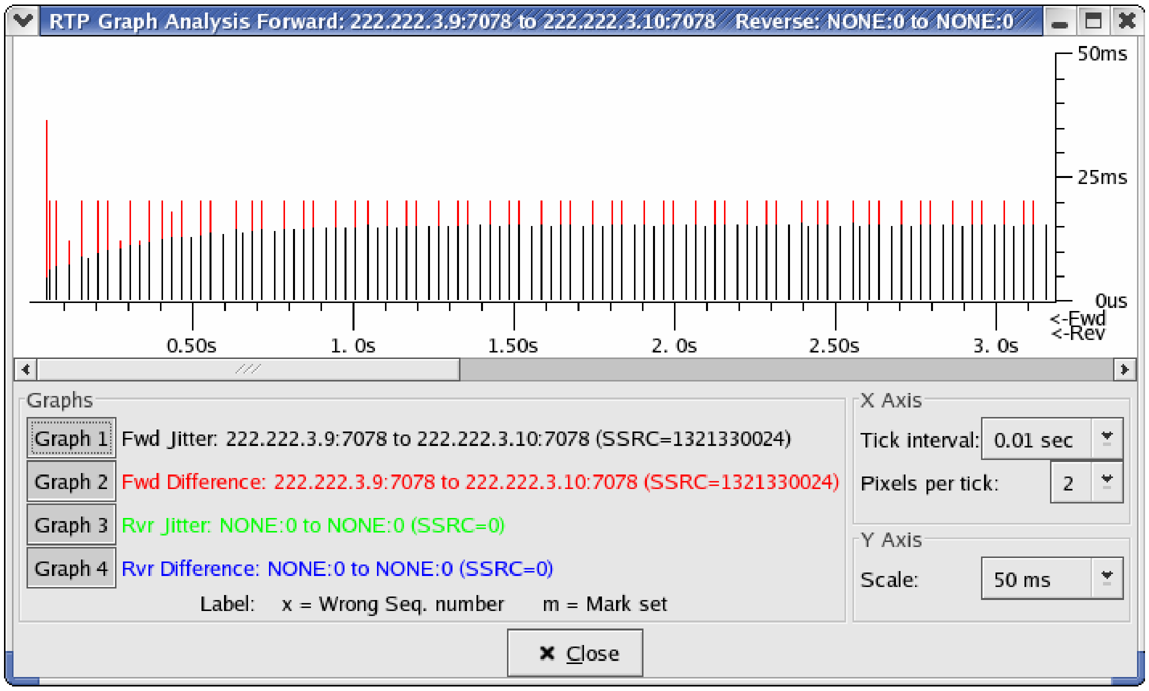
Packet	Sequence	Delta (ms)	Jitter (ms)	BW (kbp	Marker	Status
628	0	0.00	0.00	1.60		[Ok]
629	1	0.12	1.24	3.20		[Ok]
630	2	0.05	2.41	4.80		[Ok]
636	3	56.29	4.53	6.40		[Ok]
638	4	8.79	4.95	8.00		[Ok]
639	5	0.05	5.88	9.60		[Ok]
643	6	23.98	5.76	11.20		[Ok]
644	7	0.08	6.65	12.80		[Ok]
648	8	31.88	6.98	14.40		[Ok]
653	9	40.01	7.79	16.00		[Ok]
654	10	0.06	8.55	17.60		[Ok]
657	11	23.92	8.26	19.20		[Ok]
662	12	32.02	8.50	20.80		[Ok]
663	13	0.05	9.21	22.40		[Ok]
666	14	31.97	9.38	24.00		[Ok]
667	15	0.08	10.04	25.60		[Ok]
670	16	31.89	10.16	27.20		[Ok]
673	17	32.02	10.27	28.80		[Ok]
674	18	0.08	10.88	30.40		[Ok]

Max delta = 0.056285 sec at packet no. 636
Total RTP packets = 406 (expected 406) Lost RTP packets = 0 (0.00%) Sequence errors = 0

Figure:11

In above figure shows the RTP stream analysis from 222.222.3.9 to 222.222.3.10. There is zero percent packet loss. Here Jitter might be reached up to max value set of delay buffer.

6.4 RTP Graph Analysis


RTP Graph Analysis Forward: 222.222.3.9:7078 to 222.222.3.10:7078 Reverse: NONE:0 to NONE:0

Graphs
Graph 1 Fwd Jitter: 222.222.3.9:7078 to 222.222.3.10:7078 (SSRC=1321330024)
Graph 2 Fwd Difference: 222.222.3.9:7078 to 222.222.3.10:7078 (SSRC=1321330024)
Graph 3 Rvr Jitter: NONE:0 to NONE:0 (SSRC=0)
Graph 4 Rvr Difference: NONE:0 to NONE:0 (SSRC=0)
Label: x = Wrong Seq. number m = Mark set

X Axis
Tick interval: 0.01 sec
Pixels per tick: 2

Y Axis
Scale: 50 ms

Close

Above figure shows the jitter values at different time values. Here average jitter experience is 15 msec. Average Fwd Difference is about 4 to 5 msec.

7. Conclusion:

VOIP for wireless LAN brings economical and roaming advantages to the users. The WLAN industry is working hard to enable 802.11-based networks to accommodate the technical characteristics of VOIP. Technical challenges were faced because of different environments. SIP stack, RTP stack and other libraries were cross compiled and ported. The sip soft phone was optimized, configured, cross compiled and ported in the Simputer. There was no device driver for wireless adapter in Simputer having Linux 2.4.18 kernel. Device driver for 2.6.x kernel were configure and installed in linux work station and this way wireless environment was created. The performance was also analyzed. The delay jitter was in range of 14 to 16 ms. It was found that delay jitter between cross platform was 4 to 5 ms greater than between two Workstation .

Future Work:

Latency-induced VOIP performance degrades as users roam. When both the nodes are roaming in wireless network, the network is temporarily unavailable for sometime and no packets are transmitted or received, it will be interesting to monitor the status AP using Simple Network Management Protocol (SNMP) to verify if AP is working properly during this period. So improvement for roaming in VOWAN is required.

8.References :

[1] RFC 3261 'SIP: Session Initiation Protocol'.
http://www.ietf.org/rfc/rfc3261.txt

[2] RFC 3550 'RTP:Real Time Protocol'.
www.ietf.org/rfc/rfc3550.txt

[3] Marco Zibull, André Riedel, Dieter Hogrefe. Voice over wireless LAN: a fine-scalable channel-adaptive speech coding scheme: Proceedings of the 3rd ACM international workshop on Wireless mobile applications and services on WLAN hotspots WMASH '05

[4] **A. Koepsel and A. Wolisz**. Voice transmission in an IEEE 802.11 WLAN based access network. In WOWMOM '01: Proceedings of the 4th ACM international workshop on Wireless mobile multimedia, New York, USA, 2001

[5] Rakesh Arora, 'Voice over IP: Protocols and Standards'.
http://www.cse.ohio-state.edu /~jain/cis788-99/ftp/voip_protocols /index.html#3.-Session-Initiation- Protocol (SIP)> (May 20, 2004)

[6] Kundan Singh and Henning Schulzrinne. Peer-to-Peer Internet Telephony using SIP: Proceedings of the international workshop on Network and operating systems support for digital audio and video NOSSDAV '05

[7] Speex Codec information
http//www.speex.org

[8] Linphone Information
http://www.linphone.org

[9] SIPshare: SIP Beyond Voice and Video.
http//:research.earthlink.net/p2p/

[10] Linksys product information.

http://www.linksys.com

[11] WUSB54Gv4 device driver in linux

http://rt2x00.serialmonkey.com