Implementation and Optimization of H.323 Protocol Application for Embedded VoIP Phone

By

Vipul Patel

(04MCE016)

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

Implementation and Optimization of H.323 Protocol Application for 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 Patel
(04MCE016)

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

Implementation and Optimization of H.323 Protocol Application for Embedded VoIP Phone

Presented by

Vipul 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 Patel** entitled "***Implementation and Optimization of H.323 Protocol Application for Embedded VoIP Phone"*** has been carried out at **Nirma Institute Of Technology** 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: 04/05/06

Abstract

VoIP is an emerging technology. Voice over Internet Protocol (also called VoIP, IP Telephony, Internet Telephony and Digital Phone) is the routing of voice conversations over the Internet or any other IP-based network. The voice data flows over a general purpose packet-switched network instead of traditional dedicated circuit-switched voice transmission lines. Protocols used to carry voice signals over the IP network are commonly referred to as Voice over IP or VoIP Protocols. The examples of VoIP protocols are SIP, H.323, MGCP and Megaco. H.323 is a popular VoIP protocol that was originally designed for multimedia communication. The H.323 standard provides a foundation for audio, video, and data communications across IP-based networks.

My project work has been completed at NirmaLabs, the most recent initiative of Nirma Education and Research Foundation – NERF, an incubator to spawn High Tech Knowledge based Wealth Generation ventures. As the part of Project VoIP which was commenced at NirmaLabs, the task of developing optimized H.323 Protocol Application for ARM based VoIP Phone was assigned to me. Optimization can be performed due to the memory constraints on any Embedded VoIP Phone. The Optimized version of H.323 Protocol Application has been developed for both x86 and ARM platforms. Debugging and profiling have been completed in order to trace the flow of the application. For x86 platforms, the full featured H.323 enabled soft phone, having the Gatekeeper support has been developed. It has been tested and deployed successfully in the LAN environment. For ARM target, two optimized C libraries namely uClibc and Glibc were used. TS-7250 Development Board having ARM920T processor was used for the testing purpose. The signaling of H.323 Protocol Application has been tested on TS-7250 Development Board.

Acknowledgements

I would like to express my gratitude to all those who gave me their kind support to complete my thesis. I want to thank NirmaLabs, who gave me the opportunity to work on this project and provided the necessary resources. 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.

I am deeply indebted to my guide Prof. Jaladhi Joshi from CE Dept, Institute of Technology, Nirma University, who provided his constant technical guidance and encouraged me a lot to go ahead with my work during the project.

I want to thank Mr. Madhukar Pai for his technical support, interest and valuable hints through out my project.

A very special thank to my Program In Charge Dr. S. N. Pradhan of Computer Science and Engineering Dept, Institute of Technology, Nirma University for all his technical guidance in this project.

Table of Contents

List of Tables

List of Figures

Abbreviations

ADC	Analog-to-Digital Converter
ALSA	Advance Linux Sound Architecture
API	Application Programming Interface
ACF	Admission Confirmed
ALU	Arithmetic and Logical Unit
ARQ	Admission Request
ATM	Asynchronous Transfer Mode
B-ISDN	Broadband - Integrated Services Digital Network
BW	BandWidth
CFG	Control Flow Graph
CSE	Common Subexpression Elimination
CODEC	Coder/Decoder
COM Port	Communication Port
CPU	Central Processing Unit
DAC	Digital-to-Analog Converter
DIO	Digital Input Output
DNS	Domain Name Services
DSP	Digital Signal Processor
DTMF	Dual Tone Multi Frequency
EN	Enterprise Network
FPU	Floating Point Unit
GCC	GNU Compiler Collection
GNU	GNU Not UNIX
GK	Gatekeeper
GNOME	GNU Object Model Environment

GRQ	Gatekeeper Request
GSM	Global System for Mobile communication
GUI	Graphical User Interface
HTTP	Hyper Text Transfer Protocol
IDE	Integrated Development Environment
IETF	International Engineering Task Force
I/O	Input/Output
IP	Internet protocol
IP PBX	Internet Protocol Private Branch Exchange
IPX	Internet Packet Exchange
ISDN	Integrated Services Digital Network
ITU-T	International Telecommunications Union - Telecommunications sector
Kbps	Kilo bits per second
KB	KiloBytes
LAN	Local Area Network
MAN	Metropolitan Area Network
MB	MegaByts
MCU	Multipoint Control Unit
MGCP	Media Gateway Control Protocol
MHz	Mega Hertz
MMU	Memory Management Unit
MOS	Mean Opinion Score
ms	mili second
NAND	Not AND
NERF	Nirma Engineering Research Foundation
NSS	Network Security System
OS	Operating System
OSS	Open Sound System
PBX	Private Branch Exchange
PC	Personal Computer

POTS	Plain Old Telephone System
PPS	Packets Per Second
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RAS	Registration, Admissions, and Status
RRQ	Registration Request
RTP	Real-time Transport Protocol
RTCP	Real-time Transport Control Protocol
SBC	Single Board Computer
SCN	Switched Circuit Network
SDP	Session Description Protocol
SDRAM	Synchronous Dynamic Random Access Memory
SIP	Session Initiation Protocol
SMTP	Simple Mail Transfer Protocol
SPARC	Scalable Processor Architecture
T1	Terminal-1
T2	Terminal-2
TCP	Transport Control Protocol
TTL	Time-To-Live
TV	TeleVision
UDP	User Datagram Protocol
URQ	UnRegistration Request
USB	Universal Serial Board
VOP	Voice Over Packet
VoIP	Voice over IP
WAN	Wide Area Network
WWW	World Wide Web

Chapter-1 Introduction

1.1 Project Description:

My project has been completed at NirmaLabs, the most recent initiative of Nirma Education and Research Foundation – NERF, an incubator to spawn High Tech Knowledge based Wealth Generation ventures. The basic idea is to identify individuals with technical knowledge and experience, bring them to NirmaLabs campus, and groom them to evolve project plans for high tech wealth generation ventures. VoIP has been proven to be a very popular Technology. VoIP enables the transmission of voice packets over the IP network. Due to tremendous benefits of VoIP over traditional PSTN phone and very high commercialization possibilities, the Project VoIP was started on 22nd September 2005 at NirmaLabs with the team of 9 students under the guidance of Prof. Jaladhi Joshi and NirmaLabs members. Project VoIP aimed at developing the ARM based embedded VoIP phone. As part of the Project VoIP, the responsibility of Implementing and Optimizing the H.323 Protocol Application for Embedded VoIP Phone was assigned to me.

H.323 Protocol Application has been developed for both x86 and ARM platforms. H.323 Protocol Application requires the H.323 Media Library to provide the real time communication support. The Media Library used in my project was from Objective Open H.323 Media Library for developing H.323 Protocol Application. It is a fact that the cost of any Embedded VoIP Phone depends on the memory it supports. Various optimization techniques need to be applied on the code in order to improve the performance as well as to reduce the code size. For this reason, Optimization has been performed on H.323 Media Library and H.323 Protocol Application. Debugging and profiling are very useful to identify errors and flow of the application. The debugging and profiling for H.323 Protocol Application have been performed in order to provide the bug free and effective Embedded VoIP solution. The H.323 protocol based soft IP Phone has been tested and deployed

successfully in the LAN Environment. For x86 platform, the Gatekeeper support has been added to H.323 Protocol Application. For developing ARM based Embedded VoIP Phone, we need to do cross compilation. The cross compilation for both of H.323 Media Library and H.323 Protocol Application has been completed for ARM target. The ARM based TS-7250 Development Board having ARM920T processor has been used for testing purpose in my project. The H.323 Protocol Application has been tested on TS-7250 Development Board.

1.2 Organization of the Report:

My whole thesis is organized in this way. Chapter-2 is devoted to literature study of VoIP Technology. In this chapter, the topics like the issues and challenges faced by VoIP Technology, advantages of using VoIP and the applications of VoIP are described. Chapter-3 describes literature survey of H.323 Protocol Standard. This section includes all the relevant information to H.323 Standard like the components of H.323 protocol standard, H.323 Protocol Stack, Connection establishment procedure of H.323 Call and benefits/alternatives to H.323 standard. Chapter-4 includes various Optimization techniques and the results obtained after applied them on H.323 Media Library and H.323 Protocol Application. In Chapter-5, the cross compilation tools and techniques are discussed for making the H.323 Protocol Application working on TS-7250 Development Board, having ARM Processor. The results in terms of the reduced size of H.323 Media Library and H.323 Protocol Application after using both the Glibc and uClibc libraries are shown. In Chapter-6, introduction to debugging, tracing and profiling techniques are given. The results of profiling, tracing and debugging for H.323 Media Library and H.323 Protocol Application is shown in form of the table. The analysis of RTP media stream for H.323 Protocol Application is shown in this section. Chapter-7 shows the implementation and configuration of Gatekeeper support to H.323 Protocol Application for x86 platforms. In Chapter-8, the description of various tools and technology used for my project is given. The brief introduction of TS-7250 Development Board and its features is also given here. In the last Chapter-9, the help needed by end users for configuring and installing H.323 Protocol Application is given. Chapter-10 describes the conclusion and scope of the future work for my Project.

Chapter-2 Literature Study of VoIP Technology

VoIP also known as Internet Telephony is the technology that uses data packets to transmit voice over the Internet [1].

2.1 Introduction to VoIP Technology:

The idea behind VoIP is to successfully convert digital and analog communications (such as phone calls and faxes) into IP packets to be sent through one network instead of a separate telephony network. The voice signals first become digitized, then are split up into information packets and sent through an IP network [1].

VoIP consists of 3 steps:

(1) The conversion of the analog voice into IP packets, using a DSP embedded in the phone or using a PC in case of soft phones, which are software programs used with a hand set or headset connected to a PC

(2) Transmitting IP packets over a packet or data network (where the packets travel along with other VoIP packets, data packets, Video over IP packets, etc.), which are sent to appropriate destinations by routers and switches

(3) Conversion of IP packets back to voice (using DSP or computer)

With the use of appropriate gateways (devices that do appropriate conversions),
Telephone calls can be made from a,

- Soft phone to a soft phone
- Soft phone to POTS (plain old telephone system, i.e. regular phone) and vice Versa
- POTS phone to a POTS phone

VoIP technology allows you to make telephone calls using a broadband Internet connection, instead of a regular phone line. The architecture of VoIP is shown in the figure-1 below [5]:

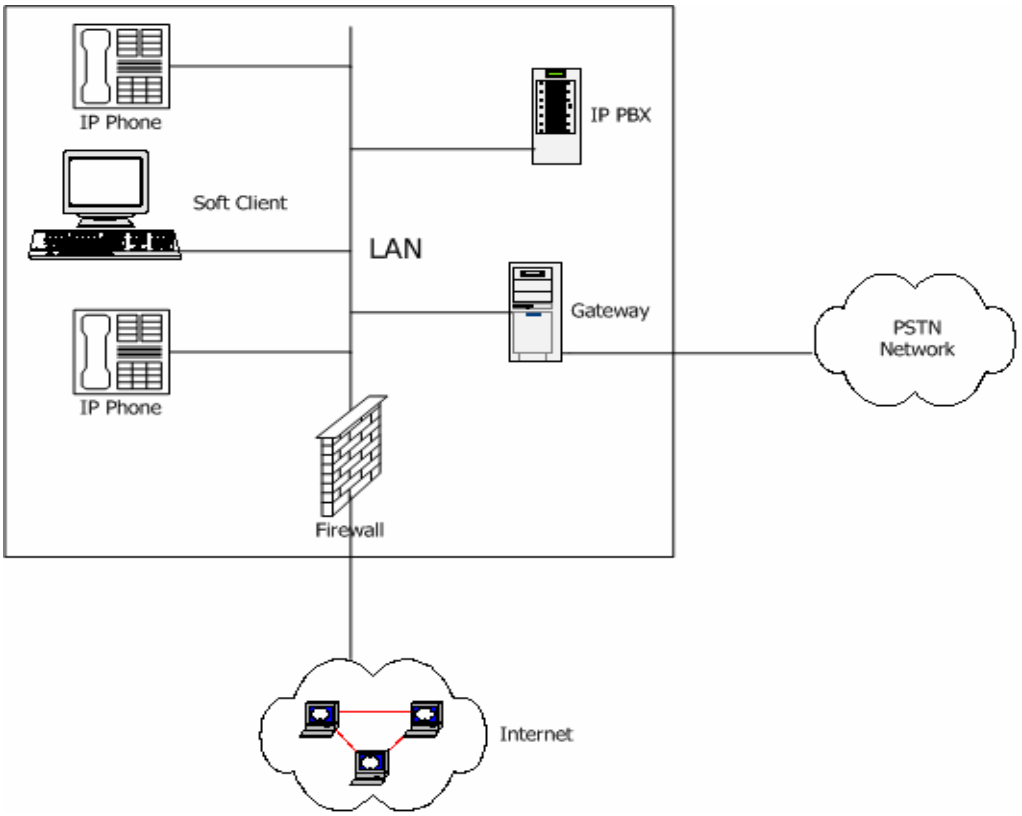

Figure-1 :: Architecture of VoIP

The VoIP architecture includes components like IP PBX, Gateway and IP Phone. The short description of each component is given as below:

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 [5]. 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.

Gateway:

The fundamental role of gateways in networks is to perform protocol conversion allowing intercommunication between different types of networks, such as between IP and PSTN. Particularly, 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 [5].

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 known as VoIP Soft Phones. But now a day's different types of VoIP phones are available [5].

2.2 Issues and Challenges in VoIP:

The biggest problem faced by voice over IP network is that of providing the end users the quality of service that they get in traditional PSTN network, where a dedicated end-to-end connection is established for a call. There are multiple parameters which affects the QoS of the network like delay, jitter and packet loss [2].

Delay:

The delay experienced in a packet network is classified into the following types: packetization delay and the network delay. Each of these adds to the overall delay experienced by a user. This delay depends upon the sample time and the type of voice coder used, areas and the standards. The accumulated voice samples are next encoded into a packet, which leads to the packetization delay. Finally, once this packet is sent through the network, it experiences some delay to reach the destination. Delay in transporting a voice packet through the network leads to two main problems, namely Echo and Talker Overlap. Echo becomes a major problem when the round trip delay through the network becomes greater than 50 milliseconds. To avoid the problems of echo in calls, echo cancellers are required. Since packet networks introduce higher end-to-end delay, and hence have a greater round trip time, echo cancellation is an essential requirement for a voice over packet network. Apart from this, in case the end-to-end delay becomes greater than 250msec, the problem of talker overlaps surfaces. This is experienced by one talker overlapping the other talker, and can be extremely annoying. Also, a delay of more than 250msec feels like a half-duplex connection and cannot be claimed to be an interactive session [2].

Jitter:

In packet-based networks, two packets sent from the same source to the same destination might take different routes through the network. This is because the packets are routed through the network independently. Hence, two packets between the same source and destination might experience different processing delays and different congestion situations in the network, resulting in a variation in the overall delay experienced by the packets. This variation in the delay experienced by packets is measured as delay jitter. Also, this might lead to packets reaching the destination out of order. Unlike data packets, voice packets would be severely affected by the delay jitter. To take care of the jitter, a buffering scheme is used at the destination. Packets at the destination are received and buffered. After the buffer is full to a threshold value, the packets are played in sequence and with a constant delay, minimizing the delay jitter. However, this buffering of packets at the destination leads to an additional delay and also adds up to the other types of delays discussed above [3].

Packet Loss:

Since IP is a best effort protocol, packets routed through an IP network can be lost. To provide reliable transmission of data in an IP network, a retransmission scheme is used at the transport layer, which retransmits any packets for which an acknowledgment is not received from the destination (assuming that the packet got lost) [2]. However, the same scheme cannot be applied to voice, as a retransmitted voice packet might reach the destination much later than when it was expected. To compensate for lost packets one scheme proposes to play the last packet received again. However, this scheme cannot work if there is a burst of lost packets. Another scheme proposes to send redundant information, increasing the bandwidth requirements. Thus, lost packet compensation is an essential requirement for supporting VoIP [3].

2.3 Applications of VoIP:

VoIP service consists of translating the traditional analog line into a digital signal so voice calls can be transferred over the Internet at costs way below traditional PSTN service. Using this fact many VoIP applications have been developed. Some of them are listed below:

- **Internet Telephony:** Traditionally, Internet has been used for data applications. However, with improvements in packet-over-voice, Internet telephony has now become a hot area. Specifically, long-distance telephony has emerged as a killer application because it offers significant cost-savings vis-à-vis PSTN.
- **Internet-based call centers:** VoIP opens exciting opportunities in call-center related activity. A user browsing through the product profile of a company may directly call on line through a VoIP based call center. This facilitates better decision-making. Besides this, call-centers located at different places may be connected through a VoIP-based network.

- **Trunking applications:** Another VoIP application that holds promise is providing trunking facilities between head-office and branch-office using an IP link. The IP link can provide voice-only services, as well as integrated voice/data connectivity.

The latter provides advantages that come along with network consolidation (e.g. reduced network management overheads, lower costs, etc.)[1].

2.4 Advantages of VoIP:

VoIP benefits users/customers, service providers, and equipment makers: Users benefit by new applications based on converged voice and data networks [1]. For example,

- VoIP eliminates Long distance charges over LANs and WANs
- No need to maintain separate voice and data networks in the enterprise
- Enables web call centers
- Supports multi-media conferencing and collaboration
- Provides unified messaging (using a phone or PDA to access both email and voice mail; appropriate conversions are performed using text-to-speech and speech recognition systems)

Service providers benefit with more efficient use of their networks. For example,

- Voice packets can be sent along with data over a single network
- Compression techniques can be applied to send more voice packets over the same bandwidth thus enabling more VoIP connections (normal voice channels require 64 kbps; this bandwidth can be reduced to 6.4kbs by using compression techniques such as G.723; further compression has a noticeable negative effect on voice quality), and a single network can be used for transmitting both voice and data.

Equipment makers will benefit from the new types of equipment and software. For example,

- VoIP gateways for converting analog voice to IP packets and vice versa. This allows the use of PSTN for the last mile connection to the phone users
- IP phones (phones which convert voice to IP packets and vice versa; no IP connection is needed for such phones - a voice connection is not required).

Chapter-3 Literature Study of H.323 Protocol Standard

H.323 is a standard that specifies the components, protocols and procedures that provide multimedia communication services (real-time audio, video, and data communications) over packet networks, including Internet protocol (IP) based networks [8] [9] [10]. H.323 is part of a family of ITU-T recommendations called H.32x that provides multimedia communication services over a variety of networks [8].

3.1 Introduction to H.323 Standard:

The H.323 standard is a cornerstone technology for the transmission of real-time audio, video, and data communications over packet-based networks [9]. Packet-based networks include IP based (including the Internet) or IPX based LANs, ENs, MANs and WANs. H.323 can be applied in a variety of mechanisms: audio only (IP telephony); audio and video (video telephony); audio and data; and audio, video and data (multimedia). H.323 can be applied to multipoint-multimedia communications. H.323 provides myriad services and, therefore, can be applied in a wide variety of areas: consumer, business, and entertainment applications [8].

H.323 Versions:

The H.323 standard is specified by the ITU-T Study Group 16. Version 1 of the H.323 recommendation - visual telephone systems and equipment for LANs that provide a non guaranteed QoS - was accepted in October 1996. It was, as the name suggests, heavily weighted towards multimedia communications in a LAN environment. The emergence of VoIP applications and IP telephony has paved the way for a revision of the H.323 specification. The absence of a standard for VoIP resulted in products that were incompatible. With the development of VoIP, new requirements emerged, such as providing communication between a PC based phone and a phone on a traditional SCN. Such requirements forced the need for a standard

for IP telephony [8]. Version 2 of H.323 (packet-based multimedia communications systems) was defined to accommodate these additional requirements and was accepted in January 1998. The new features are being added to the H.323 standard, which evolved to Version 3. The features being added include fax-over-packet networks, gatekeeper-gatekeeper communications, and fast-connection mechanisms. Recently version 4 is available in the market by ITU-T having advance features like multimedia support, conferencing support, fax support, white board application support and so on. All the standards are maintained by Packetizer website [9].

H.323 in Relation to Other Standards of the H.32x Family:

The H.323 standard is part of the H.32x family of recommendations specified by ITU-T. The other recommendations of the family specify multimedia communication services over different networks like H.324 over SCN, H.320 over ISDN, H.321 and H.310 over B-ISDN, H.322 over LANs that provide guaranteed QoS. One of the primary goals in the development of the H.323 standard was interoperability with other multimedia-services networks [8].

3.2 Components of H.323 standard:

The H.323 standard specifies four kinds of components, which, when networked together, provide the point-to-point and point-to-multipoint multimedia-communication services: There are four main components of the H.323 network called terminals, gateways, gatekeepers and MCUs [9]. The four components are shown in Figure-2. The short description for each component is given below:

Terminals:

Used for real-time bidirectional multimedia communications, an H.323 terminal can either be a PC or a stand-alone device, running an H.323 Protocol Applications. It supports audio communications and can optionally support video or data communications. Because the basic service provided by an H.323 terminal is audio

communications, an H.323 terminal plays a key role in IP telephony services. The primary goal of H.323 is to inter work with other multimedia terminals [8].

Gateways:

A gateway connects two dissimilar networks. An H.323 gateway provides connectivity between an H.323 network and a non-H.323 network. For example, a gateway can connect and provide communication between an H.323 terminal and SCN networks. This connectivity of dissimilar networks is achieved by translating protocols for call setup and release, converting media formats between different networks, and transferring information between the networks connected by the gateway. A gateway is not required for communication between two terminals on an H.323 network [8] [9].

Gatekeepers:

A gatekeeper can be considered as the brain of the H.323 network. It is the focal point for all calls within the H.323 network. Although they are not required, gatekeepers provide important mandatory services such as addressing; call admission, bandwidth management; accounting; billing; and charging [8]. Gatekeepers may also provide optional services like call control signaling, call authorization and call management. A gatekeeper is optional in an H.323 system. H.323 networks that do not have gatekeepers may not have these capabilities, but H.323 networks that contain IP-telephony gateways should also contain a gatekeeper to translate incoming E.164 telephone addresses into transport addresses. A gatekeeper is a logical component of H.323 but can be implemented as part of a gateway or MCU [10].

Multipoint Control Units:

MCUs provide support for conferences of three or more H.323 terminals. All terminals participating in the conference establish a connection with the MCU. The MCU manages conference resources, negotiates between terminals for the purpose of

determining the audio or video CODEC to use, and may handle the media stream. The gatekeepers, gateways, and MCUs are logically separate components of the H.323 standard but can be implemented as a single physical device [9].

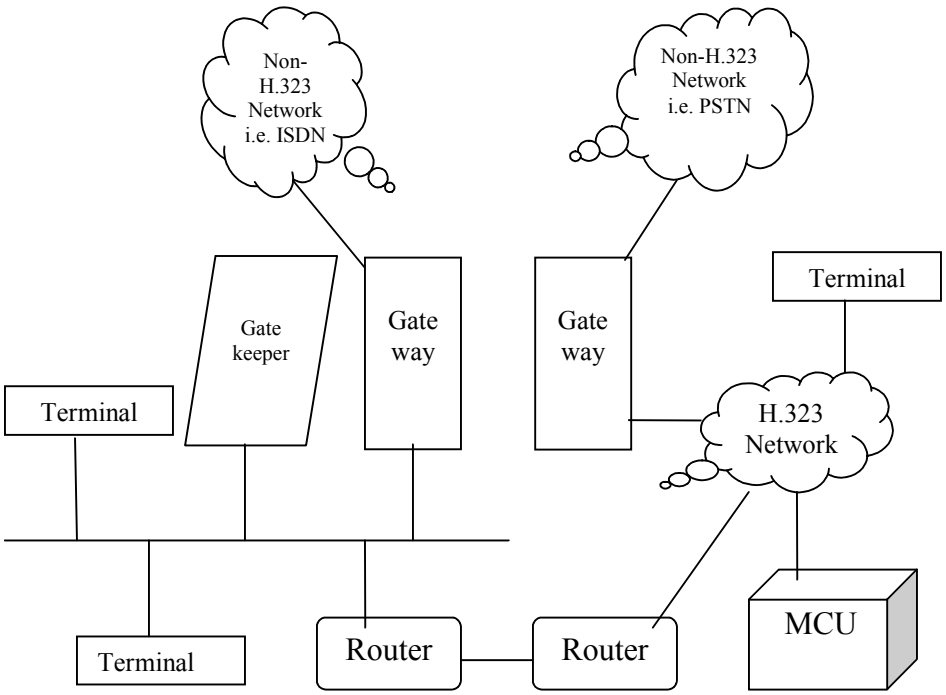

Figure-2 :: Components of H.323 Standard

H.323 Zone:

An H.323 zone is a collection of all terminals, gateways, and MCUs managed by a single gatekeeper. A zone includes at least one terminal and may include gateways or MCUs. A zone has only one gatekeeper. A zone may be independent of network topology and may be comprised of multiple network segments that are connected using routers or other devices [8] [9].

3.3 H.323 Protocol Stack:

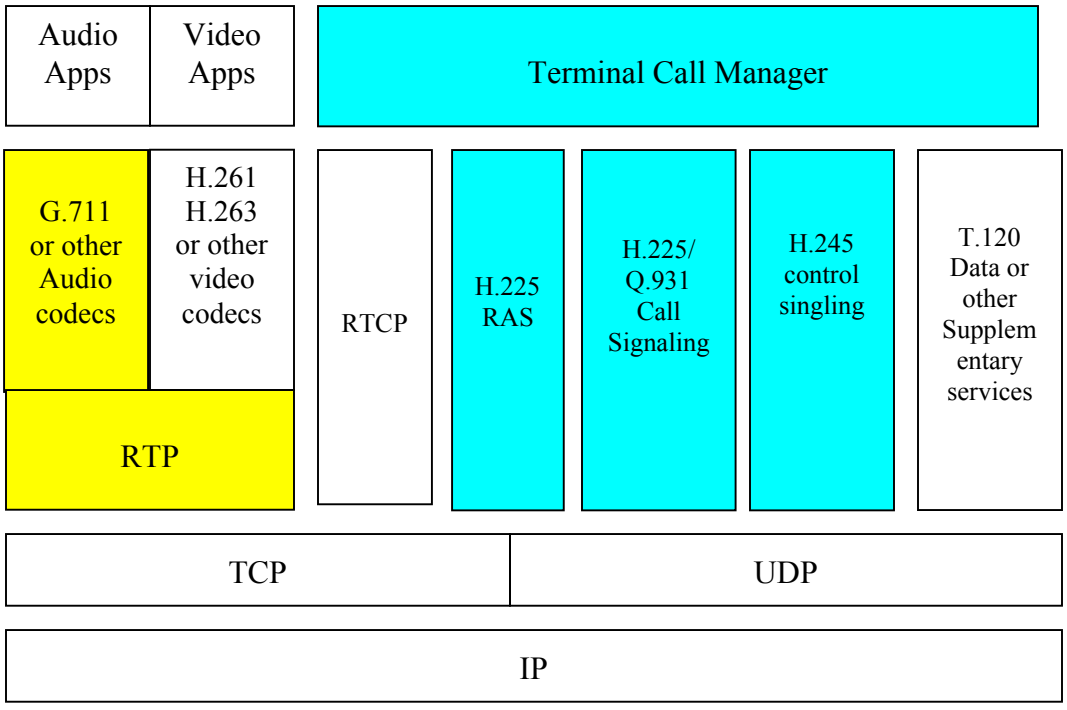

Figure-3 :: H.323 Protocol Stack

The H.323 standard is actually an umbrella of protocols. H.323 protocol stack is shown in Figure-3. H.323 The H.323 Protocol Stack shown in the figure-3 is very similar to that provided by the ITU-T [8]. The part of the protocol stack in light blue color is implemented in H.323 Media Library. The portion in the yellow color is developer specific. The developer can use this H.323 Media Library to implement H.323 Protocol Application [12]. According to ITU-T standard, it may also contain some optional features like H.235, H.450, H.239 and T.38 depending on the developers of the H.323 Media Library.

Audio CODEC:

An audio CODEC takes the audio signal from the microphone for transmission on the transmitting H.323 terminal and decodes the received audio code that is sent to the speaker on the receiving H.323 terminal [8]. Because audio is the minimum service provided by the H.323 standard, all H.323 terminals must have at least one audio CODEC support as ITU-T recommendation (G.711 at 64 kbps) [10]. Additional audio CODEC recommendations such as G.722 (64, 56, and 48 kbps), G.723.1 (5.3 and 6.3 kbps), G.728 (16 kbps), and G.729 (8 kbps) can be supported by the developer [8].

Video CODEC:

A video CODEC takes video from the camera for transmission on the transmitting H.323 terminal and decodes the received video code that is sent to the video display on the receiving H.323 terminal. Because H.323 specifies support of video capabilities as optional, the support of video CODECs is optional as well. However, any H.323 terminal providing video communications must support video encoding and decoding as specified in the ITU-T H.261 recommendation [8].

H.225 Registration, Admission, and Status:

RAS is the protocol between endpoints (terminals and gateways) and gatekeepers. RAS is used to perform registration, admission control, bandwidth changes, status, and disengage procedures between endpoints and gatekeepers. RAS channel is used to exchange RAS messages. This signaling channel is opened prior to the establishment of any other channels [8].

H.225 Call Signaling:

The H.225 call signaling is used to establish a connection between two H.323 endpoints. This is achieved by exchanging H.225 protocol messages on the call-signaling channel. The call-signaling channel is opened between two H.323 endpoints or between an endpoint and the gatekeeper [8].

H.245 Control Signaling:

H.245 control signaling is used to exchange end-to-end control messages governing the operation of the H.323 endpoint. These control messages carry information like capabilities exchange, opening and closing of logical channels used to carry media streams, flow-control messages, general commands and indications [8].

Real-Time Transport Protocol:

RTP provides end-to-end delivery services of real-time audio and video. Whereas H.323 is used to transport data over IP based networks, RTP is typically used to transport data via the UDP. RTP, together with UDP, provides transport-protocol functionality. RTP provides payload-type identification, sequence numbering, time stamping, and delivery monitoring. UDP provides multiplexing and checksum services. RTP can also be used with other transport protocols [9].

Real-Time Transport Control Protocol:

RTCP is the counterpart of RTP that provides control services. The primary function of RTCP is to provide feedback on the quality of the data distribution. Other RTCP functions include carrying a transport-level identifier for an RTP source, called a canonical name, which is used by receivers to synchronize audio and video [8].

T.120: The T.120 standard specifies data conferencing.

T.38: This T.38 standard specifies an IP based fax service.

H.235: The H.235 standard specifies security and authentication mechanism [9].

H.450: The H.450 standard specifies the supplementary services like call transfer, call diversion, call hold, call park, call waiting, call offer, call intrusion, message waiting indication and calling party name identification [9].

3.4 H.323 Connection Establishment Procedure:

This section describes the steps involved in creating an H.323 call, establishing media communication, and releasing the call. The example network contains two H.323 terminals (T1 and T2) connected to a H.323 gatekeeper (GK). Direct call signaling is assumed. It is also assumed that the media stream uses RTP encapsulation [8].

H.323 Call Establishment Procedure:

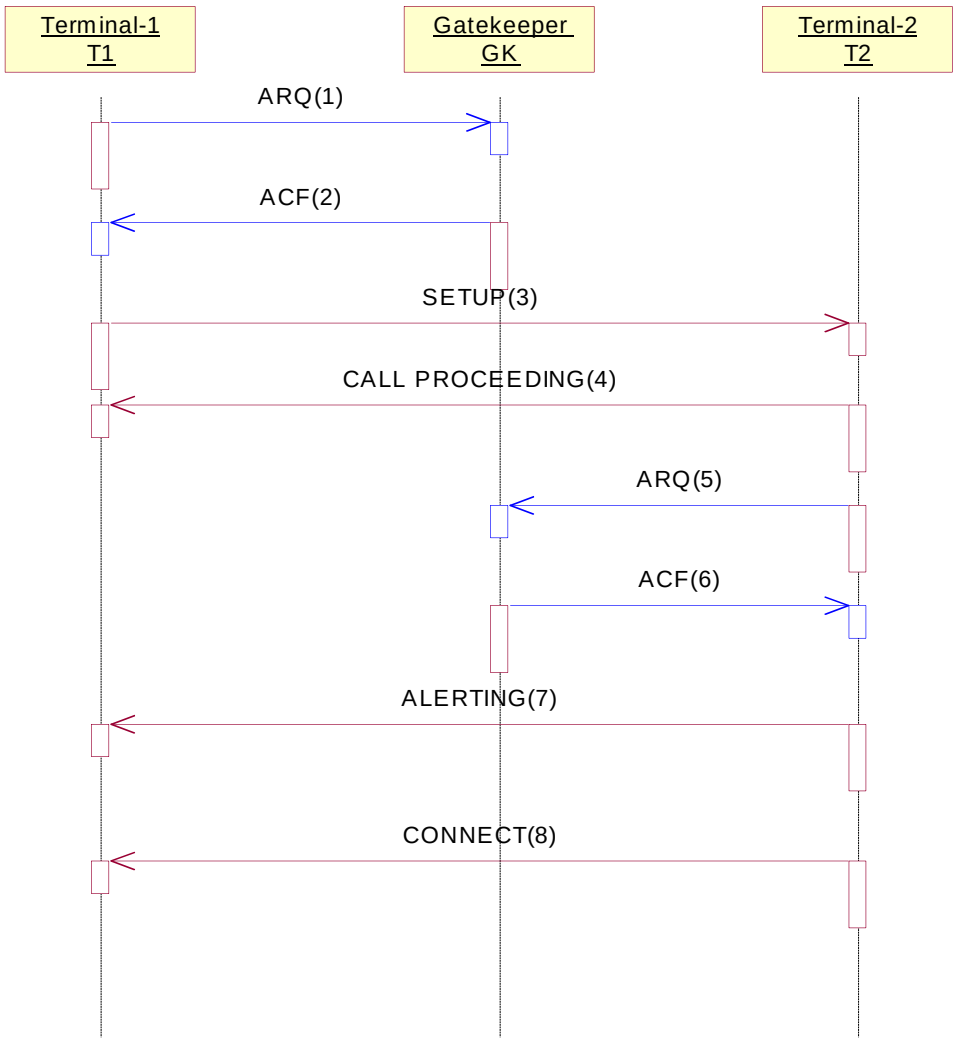

Figure-4 :: H.323 Call Establishment Procedure

Figure-4 illustrates H.323 call establishment procedure [8]. **Note** that here the messages 1, 2, 5 and 6 are H.225 signaling messages and 3, 4, 7 and 8 are RAS messages.

1. T1 sends the RAS ARQ message on the RAS channel to the gatekeeper for registration. T1 requests the use of direct call signaling.

2. The gatekeeper confirms the admission of T1 by sending ACF to T1. The gatekeeper indicates in ACF that T1 can use direct call signaling.

3. T1 sends an H.225 call signaling setup message to T2 requesting a connection.

4. T2 responds with an H.225 call proceeding message to T1.

5. Now T2 has to register with the gatekeeper. It sends an RAS ARQ message to the gatekeeper on the RAS channel.

6. The gatekeeper confirms the registration by sending an RAS ACF message to T2.

7. T2 alerts T1 of the connection establishment by sending an H.225 alerting message.

8. Then T2 confirms the connection establishment by sending an H.225 connect message to T1, and the call is established.

H.323 Control Signaling Flow:

The Steps for H.323 Control Signaling Flow is shown in the Figure-5 [8]. **Note** that here all the messages are H.245 messages.

9. The H.245 control channel is established between T1 and T2. T1 sends an H.245 TerminalCapabilitySet message to T2 to exchange its capabilities.

10. T2 acknowledges T1's capabilities by sending an H.245 TerminalCapabilitySetAck message.

11. T2 exchanges its capabilities with T1 by sending an H.245 TerminalCapabilitySet message.

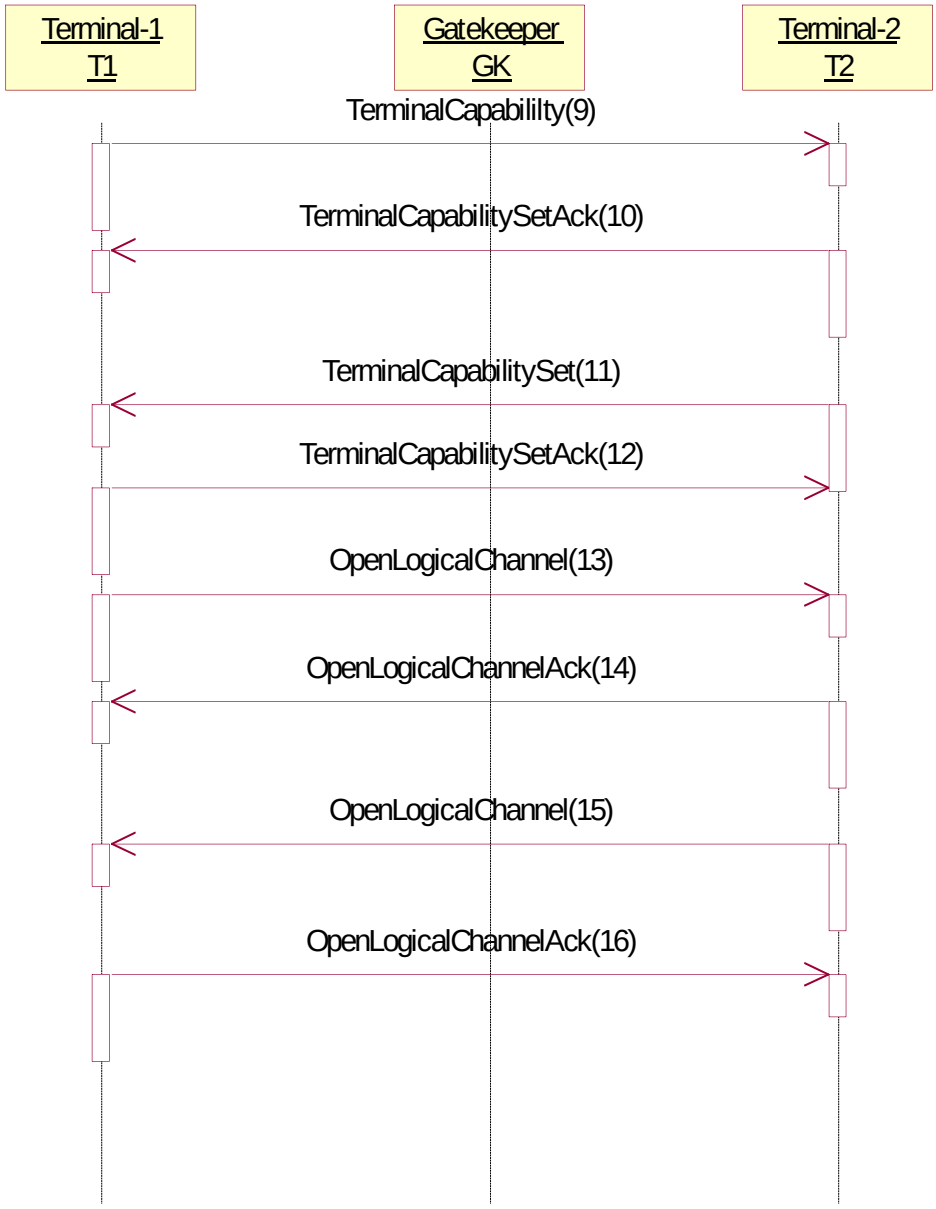

Figure-5 :: H.323 Control Signaling Flows

12. T1 acknowledges T2's capabilities by sending an H.245 TerminalCapabilitySetAck message.

13. T1 opens a media channel with T2 by sending an H.245 openLogicalChannel message. The transport address of the RTCP channel is included in the message.

14. T2 acknowledges the establishment of the unidirectional logical channel from T1 to T2 by sending an H.245 openLogicalChannelAck message. Included in the acknowledge message are the RTP transport address allocated by T2 to be used by the T1 for sending the RTP media stream and the RTCP address received from T1 earlier.

15. Then, T2 opens a media channel with T1 by sending an H.245 openLogicalChannel message. The transport address of the RTCP channel is included in the message.

16. T1 acknowledges the establishment of the unidirectional logical channel from T2 to T1 by sending an H.245 openLogicalChannelAck message. Included in the acknowledging message are the RTP transport address allocated by T1 to be used by the T2 for sending the RTP media stream and the RTCP address received from T2 earlier. Now the bidirectional media stream communication is established

H.323 Media Stream and Media Control Flow:

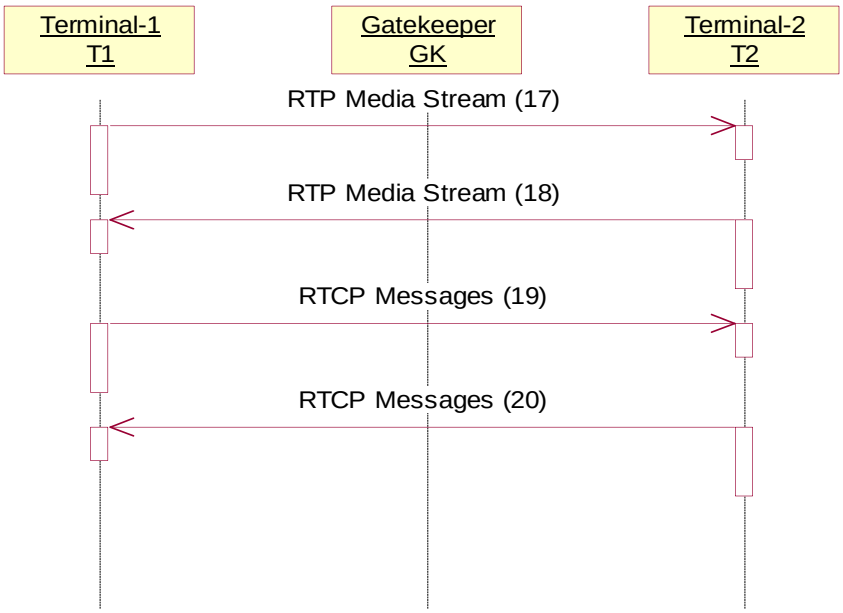

Figure-6 :: H.323 Media Stream and Media Control Flows

Figure-6 shows H.323 Media Stream and Media Control Flows [8]. **Note** that here all the messages are RTP media stream and RTCP messages.

17. T1 sends the RTP encapsulated media stream to T2.

18. T2 sends the RTP encapsulated media stream to T1.

19. T1 sends the RTCP messages to T2.

20. T2 sends the RTCP messages to T1.

H.323 Call Release Procedure:

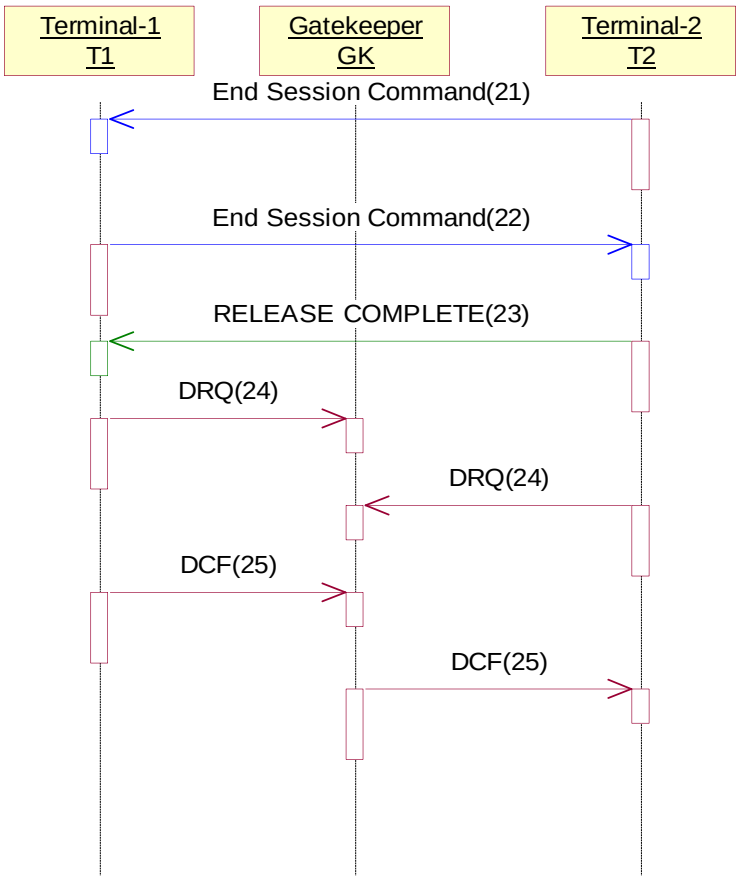

Figure-7 :: H.323 Call Release Procedure

The procedure of H.323 Call Release is shown in the Figure-7 [8]. **Note** the messages with number 21 and 22 are H.225 Signaling messages, the one with number 23 is RAS message and those with 24 and 25 are H.245 messages.

21. T2 initiates the call release. It sends an H.245 EndSessionCommand message to T1.

22. T1 releases the call endpoint and confirms the release by sending an H.245 EndSessionCommand message to T2.

23. T2 completes the call release by sending an H.225 release complete message to T1.

24. T1 and T2 disengage with the gatekeeper by sending an RAS DRQ message to the gatekeeper.

25. The gatekeeper disengages T1 and T2 and confirms by sending DCF messages to T1 and T2.

3.5 Alternatives to H.323 Protocol:

There are several alternatives to H.323 protocol like SIP, MGCP and Megaco. SIP is fairly comparable to H.323. The short description of all of these protocols is given as below:

MGCP and Megaco:

MCGP and Megaco share many similarities. Both operate in a master/slave configuration where a media gateway controller instructs the media gateway to establish, control, and release connections between one or more media streams. The similarities between the two can be traced to the roots of the protocols. Megaco was the final derivative of several draft standards, including MGCP. In general, MGCP and Megaco have constructed to accomplish the same types of tasks. However, because MGCP was a precursor to Megaco, Megaco has refined and extended many of the functionalities. Megaco is a much more complex protocol than MGCP [11].

SIP:

SIP (Session Initiation Protocol) is a lightweight, text-based signaling protocol used for establishing sessions in an IP network. SIP is a request-response protocol that closely resembles two other Internet protocols, HTTP and SMTP (the protocols that power the WWW and email); consequently, SIP sits comfortably alongside Internet applications. Using SIP, telephony becomes another web application and integrates easily into other Internet Services. SIP is a simple toolkit that service providers can use to build converged voice and multimedia services. SIP is widely used in VoIP applications, mobile applications, business applications and conferencing applications .As the name implies SIP deals generically with sessions, which can include voice, video, or data [14].

H.323 Vs SIP:

Both protocols are independent and different in architecture but offering very similar functionality. There are several key differences between these two protocols as mentioned below [15] [16]:

- **Standard:** H.323 is an ITU-T specified standard while SIP is the ITEF specified standard.

- **Architecture:** H.323 covers almost every service, such as capability exchange, conference control, basic signaling, QoS, registration, service discovery, and so on. SIP is modular because it covers basic call signaling, user location, and registration. Advance features are facilitated with SIP in other separate orthogonal protocols like SDP, RTP etc.

- **Components:** The Gateway is not used in SIP and the Gatekeeper is replaced by SIP Server.

- **Protocols:** H.323 is having subset of protocols like RAS, H.225, H.245, and RTP/RTCP while SIP is having protocols like SDP and RTP/RTCP.

- **Call Control Functionality:** Most of the call control functionalities are same in both SIP and H.323 protocol; But SIP missing some of the functionalities like message waiting indication, name identification, call offer and call intrusion facility.

3.6 Key Benefits of H.323 Standard:

H.323 Standard offers a number of advantages. The summary of key benefits of H.323 Standard is listed below [10]:

- **CODEC Standards:** H.323 establishes standards for compression and decompression of audio and video data streams, ensuring that equipment from different vendors will have some area of common support.

- **Interoperability:** Users want to conference without worrying about compatibility at the receiving point. Besides ensuring that the receiver can decompress the information, H.323 establishes methods for receiving clients to communicate capabilities to the sender. The standard also establishes common call setup and control protocols.

- **Network Independence:** H.323 is designed to run on top of common network architectures. As network technology evolves, and as bandwidth-management Techniques improve; H.323-based solutions will be able to take advantage of those enhanced capabilities.

- **Platform and Application Independence:** H.323 is not tied to any hardware or operating system. H.323-compliant platforms will be available in many sizes and shapes, including video-enabled personal computers, dedicated platforms, IP-enabled telephone handsets, cable TV set-top boxes and turnkey boxes.

- **Multipoint Support:** Although H.323 can support conferences of three or more

endpoints without requiring a specialized multipoint control unit, MCU's provide a more powerful and flexible architecture for hosting multipoint conferences. Multipoint capabilities can be included in other components of an H.323 system.

- **Bandwidth Management:** Video and audio traffic is bandwidth-intensive and could clog the corporate network. H.323 addresses this issue by providing bandwidth management. Network managers can limit the number of simultaneous H.323 connections within their network or the amount of bandwidth available to H.323 Protocol Applications. These limits ensure that critical traffic will not be disrupted.

- **Multicast Support:** H.323 supports multicast transport in multipoint conferences. Multicasting is the process of sending a single packet to a subset of destinations on the network without replication. In contrast, unicast sends multiple point-to-point transmissions, while broadcast sends to all destinations. In unicast or broadcast, the network is used inefficiently as packets are replicated throughout the network. Multicast transmission uses bandwidth more efficiently since all stations in the multicast group read a single data stream.

- **Flexibility:** An H.323 conference can include many H.323 endpoints with different capabilities. For example, a terminal with audio-only capabilities can participate in a conference with terminals that have video and/or data capabilities. Furthermore, an H.323 multimedia terminal can share the data portion of a video conference with a T.120 data only terminal, while sharing voice, video, and data with other H.323 terminals.

- **Inter-Network Conferencing:** Many users want to conference from a LAN to a remote site. For example, H.323 establishes a means of linking LAN-based desktop systems with ISDN-based group systems. H.323 uses common CODEC technology from different videoconferencing standards to minimize transcoding delays and to provide optimum performance.

Chapter-4 Optimization of H.323 Protocol Application

For any embedded application, the size of the final application executable is very important, so that Optimization comes into picture. Compiler optimization techniques are very important to reduce the size as well as to improve the performance of the application executable [17]. In this chapter, various compiler-based Optimization techniques will be introduced. Also some Optimization flags will be discussed those Compiler uses for both performance improvement and size reduction.

4.1 Compiler Optimization Techniques:

Compiler Optimization Techniques are optimization techniques that have been programmed into a compiler. These techniques are automatically applied by the compiler whenever they are appropriate. Because programmers no longer needed to manually apply these techniques, programmers are free to write source code in a straightforward manner, expressing their intentions clearly. Then the compiler can choose the most efficient way to handle the implementation details. Techniques in Optimization can be broken up along various scopes which affect anywhere from a single statement to an entire program. Generally locally scoped techniques are easier to implement than global ones but result in lesser gains. Various types of Compiler Optimizations Techniques are broadly categorized as follow [18]:

(1) Peephole Optimizations
(2) Local Optimizations
(3) Loop Optimizations
(4) Data Flow Optimizations
(5) Other Optimizations

(1) Peephole Optimizations:

Peephole Optimizations usually performed late in the compilation process. They examine at most a few instructions. They may eliminate instruction sequences that do nothing, e.g. a memory read and write that end with the same data in the same registers. It also transforms instructions into less expensive one. For example when compiler encounters A*2, it will convert it into A+A as the multiplication is more expensive than the addition operation.

(2) Local Optimization:

Local Optimizations are applied on the single basic block. A basic block is a straight-line piece of code without any jumps or jump targets in the middle; jump targets, if any, start a block, and jumps end a block. If the CPU has multiple execution units, then parts of the block without data dependencies may be executed in parallel.

(3) Loop Optimizations:

Loop Optimizations act on a number of basic blocks which make up a loop. A classic example of this is for loop in high-level code. Loop optimizations are important because many programs spend a large percent of their time inside loops. Example of the loop optimizations are Loop Unrolling and Loop fusion which are described as follow:

Loop Unrolling:

Loop Unrolling Technique duplicates the body of the loop multiple times, in order to decrease the number of times the loop condition is tested and the number of jumps. Loop unrolling is also known as loop unwinding. The idea is to save time by reducing the number of overhead instructions that the computer has to execute in a loop, thus improving the cache hit rate and reducing branching. To achieve this, the instructions that are called in multiple iterations of the loop are combined into a single iteration. This will speed up the program, if the overhead instructions of the loop impair performance.

For example: for (int x = 0; x < 100; x++) { delete(x); } *can be rewritten as*

for (int x = 0; x < 100; x += 5) { delete(x); delete(x+1); delete(x+2); delete(x+3); delete(x+4); }

Loop Fusion:

Loop Fusion is a technique which replaces multiple loops with a single loop in order to minimize the code size.

For ex: for (i = 0; i < 100; i++) { a[i] = 1; } for (i = 0; i < 100; i++) { b[i]= 2; } *can be rewritten as*

for (i = 0; i < 100; i++) { a[i] = 1; b[i] = 2; }

(4) Data Flow Optimizations:

Data flow optimizations, based on Data-flow analysis, primarily depend on how certain properties of data are propagated by control edges in the Control Flow Graph. A control flow graph is a representation, using graph notation, of all paths that might be traversed through a program during its execution. Each node in the graph represents a basic block. Directed edges are used to represent jumps in the control flow. There are two specially designated blocks: the entry block, through which control enters into the flow graph, and the *exit block*, through which all control flow leaves. Some of Data Flow Optimizations are given as follow:

Common subexpression elimination (CSE):

CSE is a compiler optimization that searches for instances of identical expressions (i.e. they all evaluate to the same value), and analyses whether it is worthwhile replacing them with a single variable holding the computed value. For example: (a+b)-(a+b)/4 Compiler realize (a+b) won't change and calculate its value once.

Constant folding and prorogation

It replaces expression consisting constants (e.g. "3 + 5") with their final value ("8") at compile time rather than at run time.

(5) Other Optimizations:

This category includes various Optimizations like dead code elimination and Instruction scheduling.

Dead code elimination:

This is used to reduce program size by removing code which can never be executed (known as dead or unreachable code).

For example: the statement after return in program function will never execute and it will be declared as a dead code.

Instruction Scheduling:

Instruction Scheduling is a compiler optimization used to improve instruction-level parallelism, which improves performance on machines with instruction pipelines. Without changing the meaning of the code, it tries to tolerate pipeline stalls by rearranging the order of instructions to avoid duplicated memory access. The pipeline stalls can caused by structural hazards (processor resource limit), data hazards (output of one instruction needed by another instruction) and control hazards (branching).

Factors affecting the Compiler Optimizations:

The factors that affect the compiler optimization are no of CPU registers, no of functional units and cache size.

No of CPU registers:

To a certain extent, more the registers, easier it is to optimize for performance. Local variables can be allocated in the registers and not on the stack. Temporary /intermediate results can be left in registers without writing to and reading back from memory.

No of functional units:

Some CPUs have several ALUs and FPUs. This allows them to execute multiple instructions simultaneously. But there may be restrictions on which instructions can pair with which other instructions.

Cache Size:

Techniques like inline expansion and loop unrolling may increase the size of the generated code and reduce code locality. The program may slow down drastically if an off-run piece of code (like inner loops in various algorithms) suddenly cannot fit in the cache.

4.2 Effect of various Compiler Optimization flags:

Without any optimization option, the compiler's goal is to reduce the cost of compilation and to make debugging produce the expected results. Turning on optimization flags makes the compiler attempt to improve the performance and/or code size at the expense of compilation time and possibly the ability to debug the program. The compiler performs optimization based on the knowledge it has of the program. Not all optimizations are controlled directly by a flag. The optimizations are divided into certain levels as follow [17]:

-O0 = this will not perform optimization. This is the default compiler option.
-O1 = this option tries to reduce code size and execution time. It takes more time and large memory for large functions.
-O2 = this option optimize even more than -O2 option. It doesn't perform space-speed tradeoff optimization options. It excludes loop optimization and function inlining optimization. It takes grater compilation time but results in greater performance.
-O3 = this option tries to optimize yet more. It turns on all optimizations specified by -O2. It also performs function inlining and loop optimizations.
-Os = this enables all -O2 optimizations that do not typically increase code size. It also performs further optimizations designed to reduce code size. This is designed internally in GCC to enable/disable certain optimization algorithms.

Note: If you use multiple -O options, with or without level numbers, the last such option is the one that is effective. Some of the flags are very important in reducing the size as well as improving the performance. They are enabled or disabled by different optimization levels. Several optimization flags used in my project are given as below:

-ffunction-sections -fdata-sections

Place each function or data item into its own section in the output file if the target supports arbitrary sections. The name of the function or the name of the data item determines the section's name in the output file. Use these options on systems where the linker can perform optimizations to improve locality of reference in the instruction space. When you specify these options, it generates only one function/data per section and this helps the linker to omit the unused function/data from the executable. This won't reflect the object size but they reduce the executable size.

-fforce-mem -fforce-addr

Force memory operands and memory address constants to be copied into registers before doing arithmetic on them. This produces better code by making all memory references potential common sub expressions.

-finline-functions

Integrate all simple functions into their callers. The compiler heuristically decides which functions are simple enough to be worth integrating in this way. This will improve the performance.

-fmerge-constants

Attempt to merge identical constants (string constants and floating point constants) across compilation units.

-fschedule-insns

If supported for the target machine, attempt to reorder instructions to eliminate execution stalls due to required data being unavailable. This helps machines that have slow floating point or memory load instructions by allowing other instructions to be issued until the result of the load or floating point instruction is required.

-floop-optimize

Perform loop optimizations: move constant expressions out of loops, simplify exit test conditions and optionally do strength-reduction and loop unrolling as well.

-fpeephole

This will perform the Peephole optimization. I have discussed the peephole optimization in the previous section.

-mno-apcs-frame

This option is specific to ARM target. It tells the compiler that we are generating code for 32 bit ARM Instruction set.

It is proved that the combination of various compiler optimization flags can improve or degrade the performance [6]. Some of the flags those applied on H.323 Media Library and H.323 Protocol Application resulted in size reduction. **Note** that here the debugging information is also removed form the application. The result is shown in table-1.

	H.323 Media Library (KB)	**H.323 Protocol Application (KB)**	**Stripped Application (KB)**
Original (No Optimization)	2056	1558	1464
-Os --ffunction-sections --fdata-sections --fmerge-constants --fforce-mem --fforce-addr --fschedule-insns --floop-optimize	1708	984	860

Table-1 :: Effect of Optimization on the size of H.323 Media Library and H.323 Protocol Application

4.3 Stripping of various audio and video CODECs Support:

The H.323 Media Library and H.323 Protocol Application provides support for various audio CODECs like G.711, GSM, G.723 and G.729 and video CODECs like H.261 and H.263. According to our project requirement the H.323 Protocol Application didn't require the video support so that it was removed from both of H.323 Media Library and H.323 Protocol Application. Also all the audio CODECs support except G.711 was removed for the optimization purpose. Only support of G.711 A-law and u-law COPDEC is kept. The list of functions for audio and video CODECs support removed from H.323 Media Library and H.323 Protocol Application is shown in Appendix A.

After applying both Stripping and various Optimization techniques, the size of both the H.323 Media Library and H.323 Protocol Application is reduced. The results are shown in table-2.

	H.323 Media Library (KB)	**H.323 Protocol Application (KB)**	**Stripped Application (KB)**
Original	2056	1558	1464
Removing Audio CODEC support except G.711 CODEC	1348	972	848
Removing above audio + video CODEC support	1300	928	804

Table-2 :: Effect of Stripping + Optimization on the size of H.323 Media Library and H.323 Protocol Application

4.3.1 Calculate BW per call requirements:

H.323 Protocol Application has been tested successfully in the LAN environment. The comparison is made for various audio CODECs to calculate the BW per call consumption requirement. It is better to use G.711 CODEC in LAN environment due to its high MOS. The reasons for using G.711 CODEC are (1) G.711 is having the highest MOS (2) In LAN environment BW is not a major concerned and (3) G.711 is easy to implement.

The comparison has been done with various audio CODECs like G.711, G.723, G.726 , GSM and G.729 in terms of the parameters like bit-rate, MOS, complexity of the algorithm it uses, delay, voice payload size, Packets Per Second and BW per call consumption. The comparison is shown in table-3 [4] [5]:

Codec Standard	**Bit - Rate (Kbps)**	**MOS**	**Complexity**	**Delay (ms)**	**Voice Payload Size (Bytes)**	**Packets Per Second (PPS)**	**BW Consume Per Call (Kbps)**
G.711	64	4.3	1	0.125	160	50	87.2
G.726	32	4.0	10	0.125	80	50	55.2
GSM	13	3.7	5	20	40	40	15.4
G.723	6.3	3.8	25	37.5	24	34	21.9
G.729	8	4.0	30	15	20	50	31.2

Table-3 :: Comparison of various audio CODECs for BW per call consumption calculation

The explanation of various audio CODECs parameters is given below:

Codec Bit Rate:

Based on the codec, this is the number of bits per second that need to be transmitted in order to deliver a voice call. (Codec bit rate = codec sample size / codec sample interval).

MOS:

MOS is a system of grading the voice quality of telephone connections. With MOS, a wide range of listeners judge the quality of a voice sample on a scale of one (bad) to five (excellent). The scores are averaged to provide the MOS for the codec.

Complexity:

The complexity is calculated in terms of no of operations performed i.e. addition and multiplication performed. The Comparison is made by taking the base of G.711 having unit complexity.

Delay:

The delay is defined as the time to perform the coding/decoding operation or the processing time of the algorithm.

Voice Payload size:

The voice payload size represents the number of bytes (or bits) that are filled into a packet. The voice payload size must be a multiple of the codec sample size. For example, G.729 packets can use 10, 20, 30, 40, 50, or 60 bytes of voice payload size.

Packets Per Second (PPS):

PPS represents the number of packets that need to be transmitted every second in order to deliver the codec bit rate. For example, for a G.729 call with voice payload size per packet of 20 bytes (160 bits), 50 packets need to be transmitted every second [50 PPS = (8 Kbps) / (160 bits per packet)]

Assumptions taken in the BW per call consumption calculation:

The following protocol header assumptions have been used for the calculation:

→ Compressed IP + TCP/UDP + RTP Header = 2 to 4 Bytes

→ Ethernet Layer2 Header = 6 Bytes

→ Voice Payload Size = not fixed (variable)

BW Per Call calculation Formula with an example of G.729 CODEC:

→ Total packet size = Ethernet Layer2 Header + Compressed IP/UDP/RTP Header
+ Voice Payload Size

→ PPS = (codec bit rate / voice payload size)

→ BW = total packet size * PPS

For Example G.729:

→ Total packet size = Ethernet Layer2 Header of 6 Bytes + Compressed IP/UDP/RTP header of 2 Bytes + Voice Payload size of 20 Bytes = 28 Bytes = 28 * 8 = 224 bits

→ PPS = (8 kbps codec bitrate) / (160 bits of Voice payload) = 50 PPS

→ BW per call = voice packet size (224 bits) * 50 PPS = 11.2 Kbps

Chapter-5 Cross Compilation of H.323 Protocol Application

A Cross Compiler is a tool which runs on one system and produces code for another system. A compiler producing programs which can run on a different system is a cross compilation compiler, or simply a *cross compiler*. Similarly, we speak of cross assemblers, cross linkers, etc. The basic idea of cross-compiling is to use some processor (HOST) to compile software for some other processor (TARGET) that uses different architecture [19]. TS-7250 Development Board [30] has been used in my project for testing cross compiled H.323 Media Library and H.323 Protocol Application. This board consists of ARM920T processor, so we need a Cross Compiler which can generate executables for the ARM platform. These executables are produced on the HOST Debian-Linux machine [31] and then just transferred to the board and executed there. The Cross Compilation tool used for this purpose was Scratchbox toolkit [20].

5.1 Scratchbox- A Cross Compilation Toolkit:

Scratchbox is a cross compilation toolkit designed to make embedded Linux application development faster and easier. It also provides a full set of tools to integrate and cross compile an entire Linux distribution. Scratchbox provides a sandboxed build environment which offers a controlled set of tools and utilities needed for cross compilation. It provides an environment in which it is easy to assure that the intended versions of libraries, headers and other similar files are used during the build. Scratchbox has been designed to allow multiple application developers work simultaneously on a single host machine. Each developer has private user account, and all configurations are developer-specific. Currently, Scratchbox supports cross compiling for ARM and PowerPC targets. Scratchbox is licensed under GNU General Public License (GPL) [20].

Installing the Scratchbox:

→ You will need root privileges for this part of the Scratchbox installation.

(1) Add this line to the /etc/apt/sources.list file in Debian-Linux system:

deb http://scratchbox.org/debian ./

(2) Update the package list with command:

apt-get update

(3) Install packages:

apt-get install <packages>

Note that the apt-get utility is used to get and install the package in Debian-Linux.

→ After downloading the required packages, Scratchbox will be unpacked to '/scratchbox' directory and the installation procedure will ask you some questions about the group and user accounts. Default group to Scratchbox users is 'sbox'. Group can be renamed but default should be fine unless you have LDAP or similar network authentication scheme. If network authentication is used then using an existing group is recommended.

→ Users who will be using Scratchbox should be added using command:

sb-adduser <username>

It will automatically include users to the Scratchbox group, create user directories under '/scratchbox/users' directory and mount several directories (/dev, /proc, /tmp) under user directory. Now it is the time to configure scratchbox in order to run it.

Scratchbox Configuration:

→ To start the Scratchbox type # ./scratchbox

→ For configuring the Scratchbox use # sb-menu . After entering into the scratchbox. The Scratchbox configuration menu enables us to install/remove andselect particular target nvironment like ARM-uClibc or ARM-Glibc. The snapshot of the scratchbox configuration menu is shown in figure-8.

Figure-8 :: Snapshot of scratchbox configuration menu

5.2 uClibc Vs Glibc and the Results:

In this section, the comparison between two optimized C libraries namely uClibc [21] and Glibc is given [22]. The uClibc and Glibc both are the customized C libraries which are widely used in embedded Linux application development. uClibc and Glibc are not the same. The trade of between size and speed always exists. There are a number of differences which may or may not cause you problems. Some key differences between them are highlighted as follow [23]:

- uClibc is smaller and more configurable than Glibc
- uClibc does not even attempt to ensure binary compatibility across releases
- uClibc does not support database library (libdb) and NSS (/lib/libnss*), which allows Glibc to easily support various methods of authentication and DNS resolution. uClibc only supports flat password files and shadow password files for storing authentication information. If you need something more complex than this, you can compile and install them.

- uClibc's Math library only supports double as inline, and even the long double support is quite limited. Also, very few of the float math functions are implemented.
- uClibc's librt library currently lacks all asynchronous input output routines, all clock routines, and all shared memory routines (librt is a small general programming library. It includes lists, hash table, socket functions, notification subsystem, etc. Currently only the timer routines and the message queue routines are implemented.)
- Very limited support for the time functions -- Leap seconds are not supported
- Very limited stdio support -- uClibc's printf function is much stricter than that of Glibc
- uClibc no longer supports 'gcc -fprofile-arcs -pg' style of profiling, which causes your application to generate a 'gmon.out' file that can then be analyzed by 'gprof' that means you are lacking of debugging and profiling in case of uClibc.

H.323 Protocol Application is compiled with both of uClibc and Glibc. It was found out that uClibc results into less size compared to Glibc, but the difference is not much grater for my H.323 Protocol Application. Here various optimization techniques are applied as discussed in section-4.2. The results of application executable size and media library size after applying the optimization techniques for both uClibc and Glibc is shown in table-4.

Target	**H.323 Media Library (KB)**	**H.323 Protocol Application (KB)**	**Stripped Application (KB)**
ARM-Glibc	2608	1860	1736
ARM – Glibc (Optimized)	2000	1128	1000
ARM – uClibc	2536	1792	1668
ARM-uClibc (Optimized)	1944	1076	956

Table-4 :: uClibc Vs Glibc – effect on the size of H.323 Media Library and H.323 Protocol Application

Chapter-6 Debugging and Profiling of H.323 Protocol Application

In this section, two methods namely debugging and profiling are described for analyzing the flow and finding the errors of H.323 Protocol Application. Debugging is a methodical process of finding and reducing the number of bugs and defects. With the help of debugging, we are able to find the error location in our programs [24]. Profiling is the method of investigating program's behavior using information gathered as the program runs. The goal of profiling is to determine which parts of a program to optimize for speed or memory usage [26].

6.1 Debugging of H.323 Protocol Application using LOG file:

LOG file is very important to trace the flow of any application. The LOG file for H.323 Media Library and H.323 Protocol Application has been developed. Various checkpoints have been added to trace the program flow of H.323 Protocol Application. The LOG file shows the date and time when the function was called and the inserted comments. The content of the LOG file generated for H.323 Media Library and H.323 Protocol Application is shown as below:

Content of LOG file for H.323 Media Library:

---------- Date 03/04/06 ----------

22:10:24 Audio device open successfully

22:10:24 Sample size 8 bits, sampling rate 8000

22:10:25 StartOf:CreateReceiveRTPChannel: local 222.222.101.118:5000

22:10:25 EndOf:CreateReceiveRTPChannel

22:10:25 StartOf:StartReceiveAudioAndPlayback

22:10:25 StartOf:ReceiveSpeakerThread

22:10:25 EndOf:StartReceiveAudioAndPlayback

22:10:27 StartOf:CreateTransmitRTPChannel:Destination 222.222.101.126:5000

22:10:27 EndOf:CreateTransmitRTPChannel

22:10:27 StartOf:StartTransmitMic

22:10:27 EndOf:StartTransmitMic

22:10:27 StartOf:TransmitMicThread

22:10:36 StartOf:StopReceiveAudioAndPlayback

22:10:36 EndOf:StopReceiveAudioAndPlayback

22:10:36 StartOf:StopTransmitMic

22:10:36 EndOf:StopTransmitMic

Content of LOG file for H.323 Protocol Application:

---------- Date 03/04/06 ----------

22:10:24 Signaling IP address is set to 222.222.101.118

22:10:24 Listen port number is set to 1720

22:10:24 Creating CMD listener at 222.222.101.118:7575

22:10:24 CMD listener creation - successful

22:10:24 DTMF mode set to H.245 (alphanumeric) for endpoint

22:10:24 DTMF mode set to H.245 (signal) for endpoint

22:10:24 H323 listener creation - successful

22:10:24 H.323 Endpoint Configuration is as follows:

22:10:24 Trace File: h323app.log

22:10:24 FastStart - enabled

22:10:25 H245 Tunneling - enabled

22:10:25 AutoAnswer - disabled

22:10:25 Local signaling IP address - 222.222.101.118

22:10:25 H225 ListenPort - 1720

22:10:25 Call Establishment Timeout - 60 seconds

22:10:25 MasterSlaveDetermination Timeout - 30 seconds

22:10:25 TerminalCapabilityExchange Timeout - 30 seconds

22:10:25 LogicalChannel Timeout - 30 seconds

22:10:25 Session Timeout - 15 seconds

22:10:25 Cmd connection accepted

22:10:25 Processing MakeCall command

22:10:25 Created a new call

22:10:25 Parsing destination 222.222.101.126

22:10:25 Trying to connect to remote endpoint(222.222.101.126:1720) to setup H225 channel

22:10:25 H225 transmitter channel creation - successful

22:10:25 Created new logical channel entry

22:10:25 Receive channel of type audio started

22:10:25 Created new logical channel entry

22:10:25 Sent Message - Setup

22:10:25 H.225 Call Proceeding message received

22:10:25 H.225 Alerting message received

22:10:27 H.225 Connect message received

22:10:27 Sent Tunneled Message - TerminalCapabilitySet

22:10:27 Sent Tunneled Message - MasterSlaveDetermination

22:10:27 H.225 Facility message Received

22:10:27 MasterSlaveDetermination done - Slave

22:10:36 Processing Hang call command

22:10:36 Hanging up call

22:10:36 Processing StopMonitor command

22:10:36 Doing StopMonitorCalls

22:10:36 Clearing call

22:10:36 Cleaning Call-reason : REASON_LOCAL_CLEARED

22:10:36 Clearing all logical channels

22:10:36 Stopped Receive channel

22:10:36 Stopped Transmit channel
22:10:36 Closing H.245 connection
22:10:36 Removed call from call list
22:10:36 Stopping listener for incoming calls
22:10:36 Ending Monitor thread
22:10:37 Destroying H.323 Endpoint

6.2 Tracing of H.323 Protocol Application using Strace Toolkit:

Strace Toolkit [25] is very important to see how your application interacts with OS or Kernel. It intercepts and records all the system calls which are called by a process and the signals which are received by a process. System administrators, diagnosticians and trouble-shooters will find it valuable for solving problems with programs for which the source code is not readily available since they do not need to be recompiled in order to trace them. It is very useful in close examination of the boundary between user and kernel for bug isolation, sanity checking and attempting to capture race conditions.

STRACE USAGE:

Use the following commands for tracing the application executable:
Strace <Strace options> <application executable> <application options>
You can write the standard output to specific file by giving -o <filename> option

Some important System Calls used by H.323 Protocol Application are as follows:

write - send a message to another user
kill - send a terminate signal to a process
mmap, munmap - map or unmap files or devices into memory
access - check user's permissions for a file
close - close a file descriptor

read - read from a file descriptor

socket - create an endpoint for communication

clone - create a child process

connect - initiate a connection on a socket

listen - listen for connections on a socket

gettimeofday, settimeofday - get / set time

uname - print system information

getuid - get user identity

bind - bind a name to a socket

wait, waitpid - wait for process termination

getsockname - get socket name

times - get process times

System Call	No of times	Errors	Seconds	% time
Write	6		0.006161	0.85
Socket	104		0.001106	6.08
Connect	3	1	0.000470	2.58
Gettimeofday	35	7	0.000389	2.14
Accept	146		0.000244	1.34
Bind	7	1	0.000171	0.94
Send	6		0.000144	0.79
Recv	15		0.000143	3.21
Waitpid	24	2	0.000108	0.59
----------	----------	----------	----------	----------
Total	649	42	0.018203	100

Table-5 :: Tracing of system calls for H.323 Protocol Application

The Strace is installed and tested for H.323 Protocol Application. It gives useful information like which system call is executed how many times by your application; percentage of total

execution time spent in particular system call and time spent in each system call. It also shows the number of errors occurred in each system call. H.323 Protocol Application has been traced using Strace and part of the result is shown in table-5:

Note that Strace is useful in case where the source code is not available. The tracing of H.323 Protocol Application is shown to demonstrate the power of Strace. When we run any application on any target machine, we have the executable only and not the source code. In this kind of situation Strace is very useful. Even though there were errors in some system calls for H.323 Protocol Application, it was running fine.

6.3 Profiling of H.323 Protocol Application using Gprof Utility:

Profiling allows you to learn information like where your program spent its time, number of times a particular function is called, which functions are called by a particular function while it was executing etc. This information can show you which pieces of your program are slower than you expected, and might be candidates for rewriting to make your program execution faster. **Gprof Utility** of Linux is used for profiling the H.323 Protocol Application. You can see the functional level flow of your application. Since the profiler uses information collected during the actual execution of the program, it can be used on programs that are too large or too complex to analyze by reading the source [26].

Steps for Profiling:

(1) You must compile and link your program with profiling enabled gcc option like
→ export CFLAGS=-pg
→ Configure your application
→ Install your application
(2) You must execute your program to generate a profile data file
→ ./executable
→ This will create gmon.out file which is having the format similar to a.out

(3) You must run gprof in order to analyze the profiled data

→ gprof <options> <application executable> <gmon.out>

→ If you want to redirect the results in some text file then use

gprof –b exe-name gmon.out > result.txt

→ Open the txt file or open the gmon.out in the **Kprof** utility of Debian-Linux to analyze the profiled data for your application

Note that the Kprof utility [27] of Debian-Linux shows the graphical view of the profiled data as opposed to the Gprof utility which shows unformatted data. Kprof is used because the output of Gprof was very difficult to read.

Name of the Function	**No of times the function is Called**	**Time spent in function (Seconds)**	**Percentage of total execution time**
ooH323EpInitialize	1	0.02	3.03
ooH323SetEpCallbacks	3	0.01	3.67
ooReadAndProcessStackCommands	4	0.01	6.67
ooHandleH225Messeges	2	0.04	4.01
ooHandleH245Messeges	3	0.04	2.05
ooEpStartRecvChannel	1	0.01	1.37
.....	.....	.....	.....
ooH323EpDestroy	1	0.01	4.23

Table-6 :: Profiling results of H.323 Protocol Application

The result of the analysis is the flat profile. The flat profile shows how much time your program spent in each function, and how many times that function was called. If you simply want to know which functions burn most of the cycles, it is stated concisely here. This can

suggest places where you might try to eliminate function calls that use a lot of time or optimize them in order to improve the performance. The profiling of H.323 Protocol Application has been completed and part of the results is shown in table-6.

6.4 Analysis of RTP Packets for H.323 Protocol Application:

For voice communication the analysis of RTP packets stream is very important. The reach ability and format of the voice packets (RTP packets) has been checked. The RTP packets are easily captured and analyzed by ethereal toolkit [28]. The snapshot taken to analyze RTP packet is shown in figure-9.

In voice communication, the analysis of RTP stream is used to measure delay and jitter. The contents found in RTP packets includes the information about padding, extension, contributing sources, , synchronization source, marker, payload type, sequence number, timestamps and the payload data. The delay and jitter for the flow of RTP packets were measured. For x86 architecture, the measured delay was near around 20ms and jitter was between 10ms to 30ms.

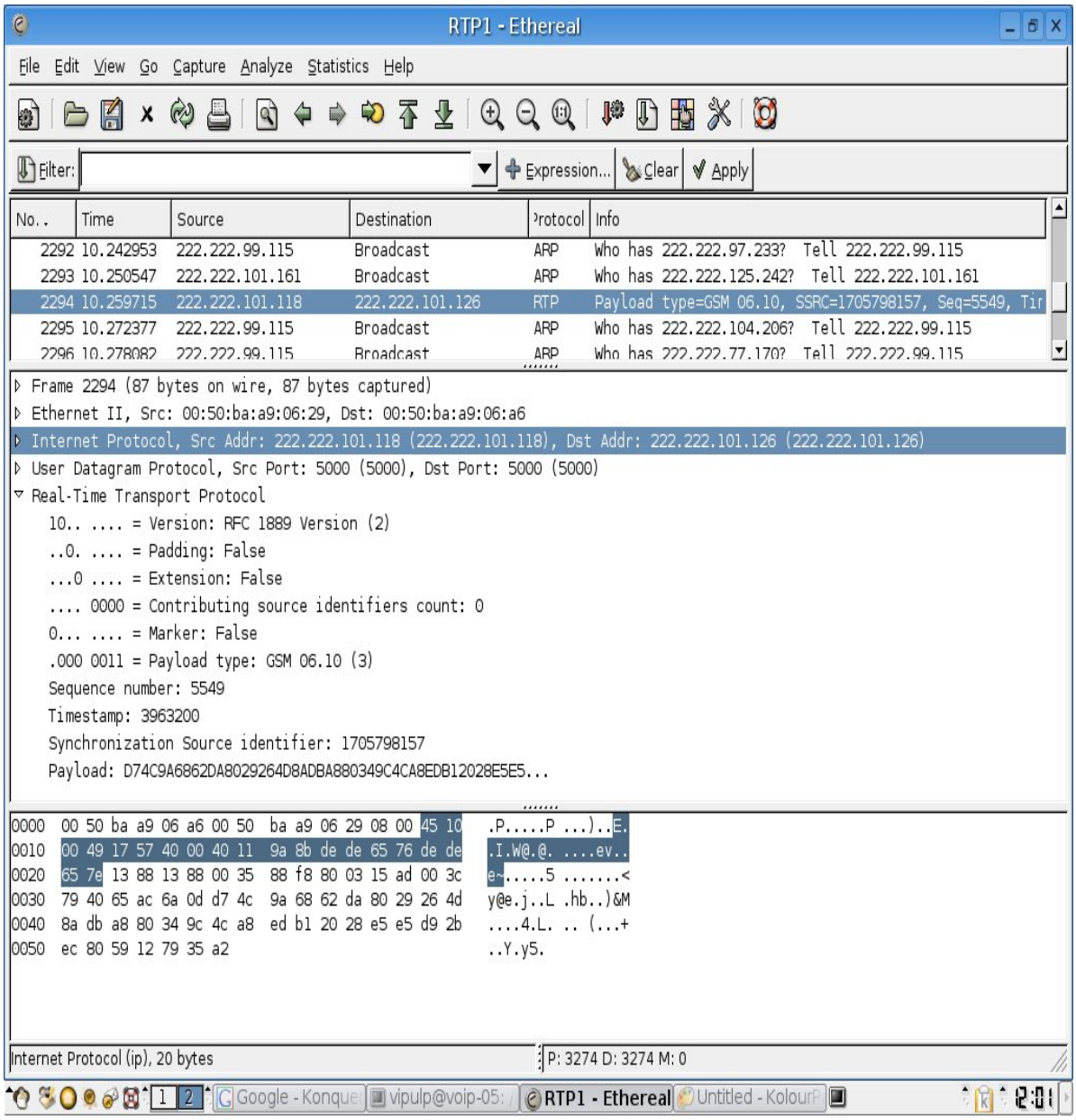

Figure- 9 :: Snapshot of RTP packet captured using ethereal

Chapter-7 Implementing GK Support to H.323 Protocol Application

In this section we will see the Implementation and Configuration of Gatekeeper with H.323 Protocol Application. H.323 Gatekeeper is an entity that manages an H.323 zone, providing address information/translation and other services to the H.323 terminals within the zone [8]. A Gatekeeper provides call control services to the H.323 endpoints. The introduction to H.323 Gatekeeper is given in chapter-3.2. Gatekeeper is an integral part of most useful Internet telephony installations that are based on the H.323 standard. The H.323 gatekeeper used in my project was from GNU organization called GnuGk [29].

7.1 Compiling and Installing GK:

To build the GnuGk H.323 Gatekeeper, PwLib and OpenH323 [13] are needed as prerequisite libraries. The development version of the gatekeeper usually needs the most recent OpenH323 version available. The libraries used are PwLib-1.5.2, OpenH323-1.12.2 and Gnuk-2.2.3. Here the OpenH323 is the H.323 protocol stack written in C++ and PwLib is the class library to support the operations of H.323 protocol [13].

Note that the order of compilation for H.323 Gatekeeper is very important. It is as follow:

(6) PwLib (release + debug version)

(7) OpenH323

(8) OpenH323 test application (not needed, just to make sure everything works so far)

(9) The GnuGk H.323 Gatekeeper

On UNIX operating system, use *`configure'* and *`make debug'* or *`make opt'* in the gatekeeper directory to build debug or release version, respectively. Use *'make both'* to build both versions. Note you have to use GCC 3.3.x or later. The older version may not work. Good practice is to do a *`make debugdepend'* or *`make optdepend'* in the gatekeeper directory before starting actual compilation (make debug or make opt) - these commands build appropriate dependency lists, so after you will update your sources from CVS, all affected files will get recompiled. Otherwise you can finish with the Gatekeeper partially compiled with the older headers and partially with the updated headers which are a not a good practice.

7.2 Examine the LOG file for GK Support:

The purpose of LOG file is discussed in section-6.1. To check whether the GK support has been added successfully or not in H.323 Protocol Application, various checkpoints were added to different functions of the gatekeeper. The content of the LOG file for H.323 Protocol Application after adding the GK support is shown below:

Content of the H.323 Protocol Application after adding the GK support:

---------- Date 10/04/06 ----------

22:10:24 Signaling IP address is set to 222.222.101.118

22:10:24 Listen port number is set to 1720

22:10:24 Creating CMD listener at 222.222.101.118:7575

22:10:24 CMD listener creation - successful

22:10:24 DTMF mode set to H.245(alphanumeric) for endpoint

22:10:24 DTMF mode set to H.245(signal) for endpoint

22:10:24 Gatekeeper Mode - RasUseSpecificGatekeeper

22:10:24 Gatekeeper IP:port set to – 222.222.101.121:1719

22:10:24 H323 listener creation - successful

22:10:24 H.323 Endpoint Configuration is as follows:

22:10:24 Trace File: h323app.log

22:10:24 FastStart - enabled

22:10:25 H245 Tunneling - enabled

22:10:25 AutoAnswer - disabled

22:10:25 Local signaling IP address - 222.222.101.118

22:10:25 H225 ListenPort - 1720

22:10:25 Call Establishment Timeout - 60 seconds

22:10:25 MasterSlaveDetermination Timeout - 30 seconds

22:10:25 TerminalCapabilityExchange Timeout - 30 seconds

22:10:25 LogicalChannel Timeout - 30 seconds

22:10:25 Session Timeout - 15 seconds

22:10:25 Gatekeeper Client Configuration:

22:10:25 Gatekeeper mode - UseSpecificGatekeeper

22:10:25 Gatekeeper To Use – 222.222.101.121:1719

22:10:25 H323 RAS channel creation - successful

22:10:25 Sent GRQ message

22:10:25 Gatekeeper Confirmed (GCF) message received.

22:10:25 Sent RRQ message

22:10:25 Registration Confirm (RCF) message received

22:10:25 Gatekeeper Registration TTL is 60

22:10:25 Cmd connection accepted

22:10:25 Processing MakeCall command

22:10:25 Created a new call

22:10:25 Parsing destination 222.222.101.126

22:10:25 Trying to connect to remote endpoint(222.222.101.126:1720) to setup H225 channel

22:10:25 H225 transmitter channel creation - successful

22:10:25 Created new logical channel entry

22:10:25 Receive channel of type audio started

22:10:25 Created new logical channel entry

22:10:25 Sent Message - Setup

22:10:25 H.225 Call Proceeding message received

22:10:25 H.225 Alerting message received

22:10:27 H.225 Connect message received

22:10:27 Sent Tunneled Message - TerminalCapabilitySet

22:10:27 Sent Tunneled Message - MasterSlaveDetermination

22:10:27 H.225 Facility message Received

22:10:27 MasterSlaveDetermination done - Slave

22:10:36 Processing Hang call command

22:10:36 Hanging up call

22:10:36 Processing StopMonitor command

22:10:36 Doing StopMonitorCalls

22:10:36 Clearing call

22:10:36 Cleaning Call-reason : REASON_LOCAL_CLEARED

22:10:36 Clearing all logical channels

22:10:36 Stopped Receive channel

22:10:36 Stopped Transmit channel

22:10:36 Closing H.245 connection

22:10:36 Removed call from list

22:10:36 Stopping listener for incoming calls

22:10:36 Ending Monitor thread

22:10:37 Destroying H.323 Endpoint

Note: The part of the added gatekeeper support is shown in the Italic format.

Chapter-8 Implementation Issues

In this section, I will include the tools and technology used to implement my project. For developing Embedded VoIP Phone, we need to port the H.323 Media Library and H.323 Protocol Application on the TS-7250 Development Board [30]. The brief introduction about TS-7250 Development Board is given in section-8.2.

8.1 Tools and Technology used:

Debian OS

Debian Operating system was used as my work environment. Debian GNU/Linux provides more than a pure OS: it comes with over 15490 packages, precompiled software bundled up in a nice format for easy installation on your machine. It provides interactive GUI for most of the applications and it is more user friendly than any other Linux flavor like Red Hat, Fedora, Kubuntu etc. Now a days, Debian is used by almost all the developers and professionals because it is open source. I installed and configured the Debian sarge (3.1).

Anjuta-1.2.2

Anjuta is a versatile Integrated Development Environment (IDE) for C and C++ on GNU/Linux. It has been written for GTK/GNOME and features a number of advanced programming facilities. These include project management, application wizards, an on-board interactive debugger, and a powerful source editor with source browsing and syntax highlighting [32].

Ethereal-0.10.10

Ethereal tool is required for capturing the traffic coming to the PC or going out side the PC. It analyzes the packets along with their contents and presents them in a nice format. We can

analyze the TCP/UDP/RTP and other kind of packets. This is an extremely good tool for networking.

Scratchbox-1.0.2

Scratchbox is a Cross Compilation toolkit designed to make Embedded Linux application development easier. It also provides a full set of tools to integrate and cross compile an entire Linux distribution. The more information has been provided in section-4.1.

Gcc-3.3.4

The GNU Compiler Collection, which currently contains front-ends for C, C++, FORTRAN, Java, and Ada, as well as libraries for these languages libstdc++, libc, etc. GCC formerly meant the GNU C compiler, which is a very high quality, very portable compiler for C and C++. The compiler supports multiple front-ends and multiple back-ends by translating first into Register Transfer Language and from there into assembly code for the target architecture.

Glibc-3.4

Glibc is a library of internal subroutines that GCC uses to overcome shortcomings of particular machines, or special needs for some languages like C. It contains all the functionalities in terms of header files in the library.

Strace-4.5.14

Strace is a system call tracer, i.e. a debugging tool which prints out a trace of all the system calls made by another process/program. The program to be traced need not be recompiled for this, so you can use it on binaries for which you don't have source code. The more information has been given in section-6.2.

Gprof-2.9.5

Gprof is a profiler that allows you to learn where your program spent its time and which functions called which other functions while it was executing. This information can show

you which pieces of your program are slower than you expected, and might be candidates for rewriting to make your program execute faster. It can also tell you which functions are being called more or less often than expected. This may help you spot bugs that had otherwise been unnoticed.

Kprof-1.4.3

Kprof is a visual tool for developers that display the execution profiling output generated by code profiling tools. The output of Gprof is usually difficult to read (beyond the flat profile information). Kprof presents the information in list-or-tree views that make the execution profiling information very easy to understand.

ALSA-1.0.11

ALSA - Advanced Linux Sound Architecture provides audio and MIDI functionality to the Linux operating system. ALSA is released under the GPL (GNU General Public license) and the LGPL (GNU Lesser General Public License). ALSA has significant features like:

- Efficient support for all types of audio interfaces
- Fully modularized sound drivers.
- User space library (alsa-lib) to simplify application programming and provide higher level functionality.
- Support for the older OSS API, providing binary compatibility for most OSS programs.

8.2 TS-7250 Development Board – A Target Platform:

For my project, Debian-Linux [31] is used as HOST and TS-7250 [30] as TARGET platform. TS-7250 is provided by the Technologic System Company. The TS-7250 is a compact full-featured Single Board Computer (SBC) based upon the Cirrus EP9302 ARM9 CPU, which provides a standard set of on-board peripherals. The EP9302 features an advanced ARM920T 200 MHz processor design with MMU that allows support for high-level Embedded Operating Systems such as Linux, Windows CE and others. This board includes

NAND flash devices, which can provide up to 128MB of fast flash memory. This allows quick operating system booting and large file system storing at the same memory device. Besides, it can run up to 64MB of high speed SDRAM. The snapshot of the TS-7250 is shown in figure-10 [30]:

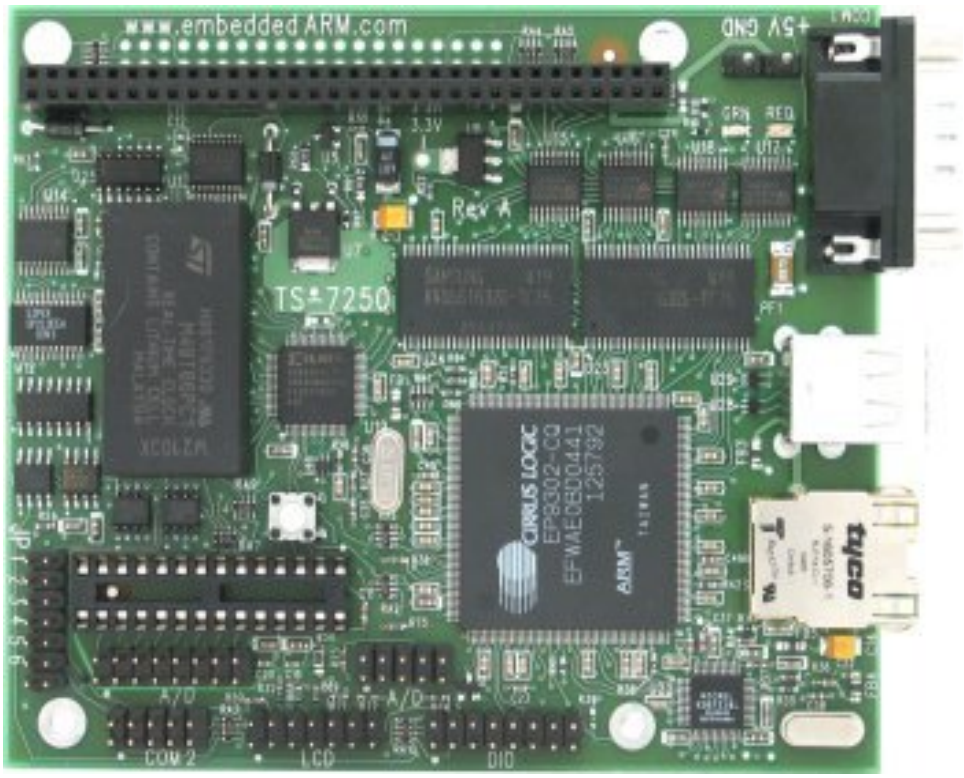

Figure-10 :: Snapshot of TS-7250 Development Board

The list of the features of TS-7250 Development Board is given as below:

- 200 MHz ARM9 processor with MMU
- Boots Linux out of the box
- 32 MB SDRAM (64 MB optional) and 32 MB Flash disk (128MB optional)
- 10/100 Ethernet
- USB Flash drives supported
- 2 USB ports, 2 COM ports, 20 DIO
- 5 channels 12-bit ADC and Optional 8 channel ADC

Chapter-9 User Guide

In this section, I will give you the overview of H.323 Media Library and H.323 Protocol Application for its usage to the end users. I will also describe the compilation and installation of the H.323 Media Library and H.323 Protocol Application.

9.1 Introduction to H.323 Media Library:

The H.323 Media Library used was from Objective Open H.323 Media Library to develop H.323 Protocol Application. H.323 Media Library is used to provide basic H.323 call signaling. The Objective Open H.323 for C (ooH323c) protocol stack is an API for building H.323 based applications [12]. The H.323 Protocol Stack implements Q.931/H.225 call signaling procedures, H.245 logical channel operations, and RAS messaging for GK communications. As shown in Figure-3, the basic functionalities like call signaling and control signaling for H.323 protocol is provided by this media library But other supports like audio-video CODECs support and the RTP support are specific and implemented by the 3rd party software developer [12].

The user functions provided by the H.323 Media Library are broadly categorized as follow:

- **Stack command functions:** These are high level functions used to initiate common H.323 telephony operations (for example, to make a call, to answer a call, to forward a call etc.).
- **H.323 endpoint management functions:** These are function used for managing the global H.323 endpoint (for example, initialize and destroying endpoint, set TCP,UDP and RTP port ranges, set IP address, adding audio and video CODECs capabilities to the endpoint etc.)
- **Call management functions:** These are functions used to manage active calls within the stack (for example, creating a new call, set calling party address, adding various audio

and video CODECs capabilities to the existing call, etc.)

- **Capability management functions:** These functions are used for negotiating capabilities between two different H.323 terminals (for example, adding and removing receive/transmit capability of voice, data and video, DTMF capabilities, etc.)
- **H.225 and H.245 message handling functions:** Functions for creating and handling H.323 standard ASN.1 messages (for example, sending and receiving of H.225 call signaling and H.245 control signaling messages)
- **Q.931 functions:** Functions for the execution of various standard Q.931 operations. These functions are used in conjunction with the H.225 call signaling messages.
- **TCP/IP and UDP socket communication functions:** Low-level functions for writing data to and receiving data from sockets (for example, creating and destroying the socket, socket operations, creating RTP packets, sending and receiving RTP packets, etc.)

Directory tree structure of H.323 Library:

The directory tree structure shows you how the library is organized. The directory tree structure for H.323 Media Library is shown in the figure-11:

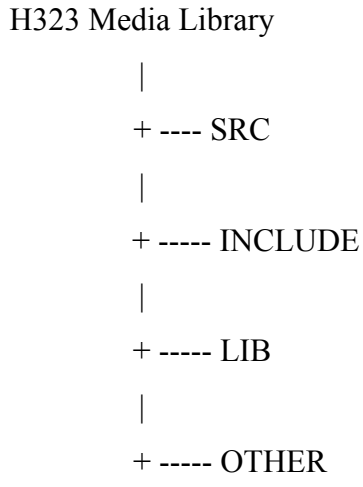

Figure-11 :: Directory tree structure of H.323 Media Library

→ SRC: It contains all the source files (e.g. all .c, .cxx files) which are used to build the library

→ INCLUDE: It contains all the header files required to build the library

→ LIB: After compilation, the library is generated in this sub-directory

→ OTHER: Other files include the Makefile and Configuration files

9.2 Compiling and Installing H.323 Protocol Application:

Before installing the H.323 Protocol Application, we need to install H.323 Media Library in order to provide the H.323 signaling protocol support. The H.323 Media Library can be installed for both Linux and Windows. The H.323 Media Library has been installed for Debian-Linux OS.

For Installing H.323 Media Library and H.323 Protocol Application, we need to tweak some flags and environment variables in order to customize it according to our requirements. For example: the path of where to install the library or if we want to generate code for some other target then set the system flags like CC, CFLAGS, CPPFLAGS, LDFLAGS etc.

Generic Installation Instructions:

The `configure' shell script attempts to guess correct values for various system-dependent variables used during compilation. It uses those values to create a `Makefile' in each directory of the package. It may also create one or more `.h' files containing system-dependent definitions. Finally, it creates a shell script `config.status' that you can run in the future to recreate the current configuration, and a file `config.log' containing compiler output (useful mainly for debugging `configure').

Simplest way to compile the package:

1. Go to the directory containing the package's source code and use `./configure' to configure the package for your system. If you're using `csh' on an old version of System V, you might need to use `sh ./configure' instead to prevent `csh' from trying to execute `configure' itself. Running `configure' takes awhile. While running, it prints some messages telling which features it is checking for.
2. Use `make' to compile the package.
3. Optionally, use `make check' to run any self-tests that come with the package.
4. Use `make install' to install the programs and any data files and documentation. (Note that by default, `make install' will install the package's files in `/usr/local/bin', `/usr/local/man', etc. You can specify an installation prefix other than `/usr/local' by giving `configure' the option `--prefix=PATH'.)
5. Here you have several choices for installing the library and its application. For example, if you want to generate the optimized version of the application then use 'make opt' and to generate debug version of the application use 'make debug' and to do profiling of your application use 'make profile' option.
6. You can remove the program binaries and object files from the source code directory by using `make clean'. To remove the files that `configure' created (so you can compile the package for a different kind of computer), type `make distclean'.

Compilers and Options:

Some systems require unusual options for compilation or linking that the `configure' script does not know about. Run `./configure --help' for details on some of the pertinent environment variables. You can give `configure' initial values for configuration parameters by setting variables in the command line or in the environment. Here is an example:

./configure CC=c89 CFLAGS=-O2

Note that if you are cross compile the code for some other target than you have to explicitly set the --host and --target environment variable while configuring.

9.3 H.323 Protocol Application usage:

This section provides the general guidelines for usage of H.323 Protocol Application to the end users. The name of the H.323 Protocol Application is given to `h323app'.

Note that before running the application, we need to set the H.323 Media Library path (for example, set LD_LIBRARY_PATH=/usr/local/lib). To see help run './h323app --help'. The front end for the H.323 Protocol Application is shown below:

Making Peer to Peer Call:

→ To run the application in listening mode use following command on one terminal
PC1$ ./h323app --use-ip IP_ADDR1 --use-port 1720 --listen

→ To make a call use following command on other terminal
PC2$ ./h323app --use-ip IP_ADDR2 --use-port 1720 IP_ADDR1

This will start transmit and receive channels between two terminals if the remote party accepts the call. Now you will find the command prompt where you can use other commands for facilities like sending the message, forwarding the call, terminating the call etc.

Note that the terminal can be either the PC or the Development Board. PC1 and PC2 are the two terminals. IP_ADDR1 and IP_ADDR2 are the IP addresses of the two terminals

****

H.323 Protocol Application

****

USAGE:

Listening to a Call h323app [options] –listen

Make a call h323app [options] <remote IP address>

USE THE FOLLOWING OPTIONS:

--use-ip <ip> - IP address for the endpoint (default - uses gethostbyname)

--use-port <port> - Port number to use for listening to incoming calls (default-1720)

--user <username> - Name of the user displayed for outbound calls

--user-number <number> - Caller number

--gk-discover - Discover gatekeeper

--gk <ip: port> - Use specific gatekeeper

--auto-answer - Enables auto answer mode

--h323id <h323id> - H323ID to be used for this endpoint

--e164 <number> - E164 number used as caller id for this endpoint

--no-faststart - Disable fast start (default - enabled)

--no-tunneling - Disable H245 Tunneling

--help -Prints this usage message

Note that you can specify options for enable/disable fast start or H.245 tunneling. Also you can specify the alias as your name for the call. The gatekeeper options are discussed in the next topic.

Running the H.323 Gatekeeper:

After the Gatekeeper installed successfully, just copy the executable to the directory you like and create a configuration file for it. In the Configuration File, we need to set the information like GK port, GK bandwidth to be used, TTL for the GK, the GK mode etc. On Linux x86 platforms, the optimized executable gnugk is produced in obj_linux_x86_r/ subdirectory. You may copy it to /usr/sbin/, create a config in /etc/gnugk.ini and start the gatekeeper by

$ /usr/sbin/gnugk -c /etc/gnugk.ini -o /var/log/gnugk.log

Making Peer to Peer Call Using Gatekeeper:

PC1$ ./h323app --use-ip IP_ADDR1 --use-port 1720 --use-gk GK_IP_ADDR --listen

PC2$ ./h323app --use-ip IP_ADDR2 --use-port 1720 --use-gk GK_IP_ADDR

IP_ADDR1

Here the call will be routed by the gatekeeper to the called party.

Note that the GK_IP_ADDR is the IP address where the gatekeeper is running. Before running the application, you have to run the gatekeeper on GK_IP_ADDR in order to provide GK support). **Note** You can also specify other options the current call.

Gatekeeper Options:

-c --config filename	Specify the configuration file to use.
-l --timetolive n	Specify the time-to-live timer (in seconds) for endpoint registration. It overrides the setting Time-to-Live in the configuration file.
-b --bandwidth n	Specify the total bandwidth available for the gatekeeper. Without specifying this option, the bandwidth management is disabled by default.
-u --user name	Run the gatekeeper process as this user. This is valid only for UNIX versions.
-d --direct	Use direct endpoint calls signaling.
-r --routed	Use gatekeeper routed call signaling.
-o --output filename	Write trace log to the specified file.
-h --help	Show all available options and quit the program.

Chapter-10 Conclusion and Scope of future work

10.1 Conclusion:

Project VoIP, commenced at NirmaLabs was aimed at developing the ARM based Embedded VoIP Phone. H.323 Protocol Application has been developed for both x86 and ARM platforms. The cost of any Embedded VoIP Phone is affected by the memory it supports. To minimize the memory requirement, the Optimized version of the H.323 Media Library and H.323 Protocol Application has been developed. Various Optimization Techniques have been applied on my code to achieve greater improvement in terms of both the size as well as the performance. After applying Optimization, the final size (H.323 Media Library + H.323 Protocol Application) has been reduced from 3.5 MB to 2.5 MB (around 1MB). As part of Optimization, the Stripping of some extra audio and video CODECs support, not required by H.323 Protocol Application has been performed. After Stripping, the final size (H.323 Media Library + H.323 Protocol Application) has been reduced from 3.5 MB to 2.1 MB (around 1.4MB). For Stripping purpose the comparison has done between various audio CODECs in terms of various parameters like MOS, delay, BW per Call consumption etc. For x86 platforms, the full featured H.323 Enabled soft VoIP Phone has been successfully tested and deployed in the LAN Environment. The Gatekeeper support has been implemented and tested with H.323 Protocol Application. Debugging and profiling of H.323 Protocol Application and H.323 Media Library have been performed in order to remove the bugs and to check the correct flow. The delay and jitter for the flow of RTP packets has been measured. The delay came near around 20ms and the jitter came between 10ms to 30ms as expected. For ARM platform, the cross compilation of H.323 Media Library and H.323 Protocol Application has been completed. Here also for Optimization purpose, two different optimized C libraries namely uClibc and Glibc were used and also the comparison has been done for the same. For ARM-Glibc target the final size (H.323 Media Library + H.323

Protocol Application) has been reduced from 4.5 MB to 3.0 MB (around 1.5MB) and for ARM-uClibc target the final size (H.323 Media Library + H.323 Protocol Application) has been reduced from 4.3 MB to 2.9 MB (around 1.4MB). The H.323 Protocol Stack and its signaling application have been tested successfully on the TS-7250 Development Board.

10.2 Scope of future work:

The field of VoIP is very vast. Several possibilities for new comers exist in this field. As far as my work is concerned, there are three things that can be enhanced as the future work namely, Application enhancement, More Optimization of H.323 Protocol Application and Solving Real time issues.

[1] Application Enhancement:

The basic H.323 enabled VoIP Application for both x86 and ARM platforms have been developed. In the future, we can develop the enhanced H.323 Protocol Applications like H.323 audio conferencing, H.323 video conferencing and H.323 multimedia application.

[2] More Optimization of Application:

We can develop our own customized C libraries like uClibc, Glibc and customize our application for any particular target platforms like ARM, PowerPC, SPARC etc. In this way we can reduce the size of our enhanced H.323 Protocol Applications. The other way is to find out the combinations of various effective compiler optimization options.

[3] Solving Real time Issues:

All VoIP Application are Real time applications. The problem generally occurs with these types of applications is the quality of voice and the effectiveness of CODEC algorithm used. Sometimes the delay is very high in the voice communication or the voice communication couldn't take place. The quality of Voice depends on the type of CODEC we are using. So some work can be done in this direction to develop the effective Embedded VoIP solutions.

References

[1] Voice over IP: A Technology brief, a white paper by Silicon Press; http://www.silicon-press.com/briefs/brief.voip/brief.pdf

[2] Voice over IP; a white paper; by DSQ software Ltd, 5/24/2004; http://www.dsqsoft.com/library/articles/voip_pap.pdf

[3] Voice over IP solutions, a white paper by Sean Christensen, Juniper Networks; http://cn.juniper.net/solutions/literature/white_papers/200011.pdf

[4] Voice over IP – Per Call Bandwidth Consumption, a white paper by CISCO Press; http://www.cisco.com/warp/public/788/pkt-voice-general/bwidth_consume.pdf

[5] Voice over IP by UYLESS BLACK, 2nd edition, Prentice Hall Series

[6] Arpad Beszedes, Tamas Gergely, "Optimizing for Space: Measurements and Possibilities for Improvements", Research Group on Artificial Intelligence, University of Szeged, Hungary, http://gcc.rgai.hu/

[7] http://www.nirmalabs.org/

[8] H.323 tutorial by IEC, The State University of New Jersey, RUTGERS, 2003; http://www.td.rutgers.edu/documentation/FAQ/H.323_Tutorial/

[9] H.323 Online tutorial; http://www.iec.org/online/tutorials/h323

[10] A Primer on H.323 Series Standard by Packetizer;
http://www.packetizer.com/voip/h323/papers/primer/

[11] http://www.protocols.com/pbook/h323.htm

[12] H.323 Protocol Stack by Objective Open Organization;
http://www.obj-sys.com/open/index.shtml

[13] http://www.openh323.org

[14] Understanding SIP, a white paper by Ubiquity;
http://www.ubiquitysoftware.com/pdf/UnderstandSIP_WP.pdf

[15] SIP Versus H.323; http://www.iptel.org/info/trends/sip.html

[16] SIP Vs H.323 – A comparison; http://microtronix.ca/sip_vs_h323.htm

[17] GCC optimization options;
http://gcc.gnu.org/onlinedocs/gcc-3.3/gcc/Optimize-Options.html

[18] Compiler based Optimization; http://en.wikipedia.org/wiki/Compiler_optimization

[19] Cross Compilation; http://en.wikipedia.org/wiki/Cross-compilation

[20] Scratchbox manual; http://www.scratchbox.org/documentation/user/scratchbox-1.0/

[21] Introduction to uClibc; http://uclibc.org/

[22] http://directory.fsf.org/GNU/glibc.html

[23] Glibc Vs uClibc; http://www.uclibc.org/downloads/Glibc_vs_uClibc_Differences.txt

[24] Degugging; http://en.wikipedia.org/wiki/Debugging

[25] Strace homepage; http://www.liacs.nl/~wichert/strace/

[26] GNU Gprof manual; http://www.delorie.com/gnu/docs/binutils/gprof_toc.html

[27] Kprof help; http://kprof.sourceforge.net/

[28] Ethereal User's Guide; http://www.ethereal.com/docs/eug_html_chunked/

[29] GnuGk Gatekeeper; http://www.gnugk.org/gnugk-manual.html

[30] TS-7250 ARM single Board Computer;
http://www.embeddedarm.com/epc/ts7250-spec-h.html

[31] Getting Started with Debian; http://www.debian.org/

[32] Anjuta help; http://anjuta.sourceforge.net/

Appendix A :: Stripped of audio and video CODEC support functions

As part of the Optimization, various audio and video CODEC related functions have been removed from the H.323 Media Library. The removed support functions are shown as below:

Removed Audio Support Functions:

** int ooCapabilityAddGSMCapability(struct OOH323CallData *call, int cap,*
unsigned framesPerPkt, OOBOOL comfortNoise,
OOBOOL scrambled, int dir,
cb_StartReceiveChannel startReceiveChannel,
cb_StartTransmitChannel startTransmitChannel,
cb_StopReceiveChannel stopReceiveChannel,
cb_StopTransmitChannel stopTransmitChannel,
OOBOOL remote);

→ This is an internal helper function which is used to add a GSM capability to local endpoints capability list or to remote endpoints capability list or to a call's capability list. The 1st argument is the handle to a call. If this is not null, then capability is added to call's remote endpoint capability list, else it is added to local H323 endpoint list. The 2nd argument is Type of GSM capability to be added. The 3rd argument is number of GSM frames per packet. The 4th argument is Comfort noise spec for the capability. The 5th argument is scrambled enabled/disabled for the capability. The 6th argument is the direction of capability like OORX, OOTX, and OORXANDTX. The 7th argument is Callback function to start receive channel. The 8th argument is Callback function to start transmit channel. The 9th argument is Callback function to stop receive channel. The 10th argument is Callback function to stop transmit channel. The 11th argument is TRUE, if adding call's remote capabilities. This function will return OO_OK, on success and OO_FAILED, on failure.

* *ooCallAddGSMCapability*

→ This function is used to add GSM capability for the call. The "ooCallAdd...Capability" functions allow to override the global endpoint capabilities and use specific capabilities for specific calls.

* *ooCallAddG723Capability*

→ This function is used to add G723 capability for the call. The "ooCallAdd...Capability" functions allow to override the global endpoint capabilities and use specific capabilities for specific calls.

* *ooCallAddG729Capability*

→ This function is used to add G729 capability for the call. The "ooCallAdd...Capability" functions allow to override the global endpoint capabilities and use specific capabilities for specific calls.

* *ooH323EpAddGSMCapability*

→ This function is used to add GSM capability for the H.323 Endpoint.

* *ooH323EpAddG723Capability*

→ This function is used to add G7231 capability to the H.323 Endpoint.

* *ooH323EpAddG729Capability*

→ This function is used to add G729 capability for the Endpoint.

* *ooCapabilityCreateGSMFullRateCapability*

→ This function is used to create a GSM Full Rate capability structure.

* *ooCapabilityCheckCompatibility_GSM*

→ This function is used to check the compatibility with GSM codec.

* *ooIsAudioDataTypeGSMSupported*

→ This function is used to check whether the GSM audio data type is supported or not.

Removed Video Support Functions:

** EXTERN int ooCapabilityAddH263VideoCapability(struct OOH323CallData *call,*
unsigned sqcifMPI, unsigned qcifMPI,
unsigned cifMPI, unsigned cif4MPI,
unsigned cif16MPI, unsigned maxBitRate, int dir,
cb_StartReceiveChannel startReceiveChannel,
cb_StartTransmitChannel startTransmitChannel,
cb_StopReceiveChannel stopReceiveChannel,
cb_StopTransmitChannel stopTransmitChannel,
OOBOOL remote);

→ This function is used to add H263 video capability to local endpoints capability list or to remote endpoints capability list or to a call's capability list. The 1st argument is the handle to a call. If this is not null, then capability is added to call's remote endpoint capability list, else it is added to local H323 endpoint list. The 2nd argument is Minimum picture interval for encoding/decoding of SQCIF pictures. The 3rd argument is Minimum picture interval for encoding/decoding of QCIF pictures. The 4th argument is Minimum picture interval for encoding/decoding of CIF pictures. The 5th argument is Minimum picture interval for encoding/decoding of CIF4 pictures. The 6th argument is Minimum picture interval for encoding/decoding of CIF16 pictures. The 7th argument is Maximum bit rate in units of 100 bits/s at which a transmitter can transmit video or a receiver can receive video. The 8th argument is Direction of capability like OORX, OOTX or OORXANDTX. The 9th argument is Callback function to start receive channel. The 10th argument is Callback function to start transmit channel. The 11th argument is Callback function to stop receive channel. The 12th argument is Callback function to stop transmit channel. The last argument is TRUE, if adding call's remote capabilities. This function will return OO_OK, on success and OO_FAILED, on failure.

** ooCapabilityUpdateJointCapabilitiesVideo*

→ This function is used to update joint video capabilities for call. It checks whether remote capability can be supported by local capabilities for the call and if supported makes entry into

the joint capability list for the call.

* *ooCapabilityUpdateJointCapabilitiesVideoH263*
→ This function is used to update joint video H263 capabilities for call. It checks whether remote capability can be supported by local capabilities for the call and if supported makes entry into the joint capability list for the call.

* *ooCapabilityCreateH263VideoCapability*
→ This function is used to create a H263 video capability.

* *ooIsVideoDataTypeSupported*
→ This function is used to determine whether a particular video capability can be supported by the endpoint.

* *ooIsVideoDataTypeH263Supported*
→ This function is used to determine whether a particular H263 capability can be supported by the endpoint.

* *ooCapabilityCheckCompatibility_Video*
→ This function is used to check the video compatibility.

* *ooCallAddH263VideoCapability*
→ This function is used to add H263 video capability for the call. The "ooCallAdd...Capability" functions allow to override the global endpoint capabilities and use specific capabilities for specific calls.

* *ooH323EpAddH263VideoCapability*
→ This function is used to add H263 video capability to the H323 endpoint.

Note that the arguments are almost same except one or two functions, so that they are not described here for all functions. The details of audio and video CODEC support related functions are given on the Objective Open H.323 web site [12].

Appendix B :: Functions for the Gatekeeper support

The functions implemented for the GK support used in H.323 Protocol Application are as follow:

** EXTERN int GkClientInit(enum RasGatekeeperMode eGkMode, char *szGkAddr,*
int iGkPort);

→ This function is used to initialize the Gatekeeper client. If an application wants to use gatekeeper services, it should call this function immediately after initializing the H323 EndPoint. The arguments are the mode of the gatekeeper, followed by the GK's IP address followed by GK's PORT. It will return 0 on failure and 1 otherwise.

→ The gk mode is one of the following:

6. RasNoGatekeeper (DEFAULT), No Gatekeeper
7. RasDiscoverGatekeeper, to discover a gatekeeper automatically
8. RasUseSpecificGatekeeper, to use a specific gatekeeper

** EXTERN int GkClientDestroy(void);*

→ This function is used to destroy Gatekeeper client. It releases all the associated memory.

** EXTERN int GkClientStart(ooGkClient *pGkClient);*

→ This function is used to start the Gatekeeper client functionality. It takes the argument as the handle to the client.

** EXTERN int GkClientSetGkMode(ooGkClient *pGkClient, enum*
*RasGatekeeperMode eGkMode, char *szGkAddr, int iGkPort);*

→ This function is invoked to set a gatekeeper mode. The list of arguments is the handle to the client, gatekeeper mode, IP and PORT on which the GK is running.

* *EXTERN int GkClientCreateChannel(ooGkClient *pGkClient);*

→ This function is used to create a RAS channel for the gatekeeper.

* *EXTERN int GkClientCloseChannel (ooGkClient *pGkClient);*

→ This function is used to close a RAS channel of the gatekeeper client.

* EXTERN int GkClientReceive(ooGkClient *pGkClient);

→ This function is invoked to receive data on Gatekeeper client's RAS channel.

* *EXTERN int GkClientHandleRASMessage(ooGkClient *pGkClient,*
*H225RasMessage *pRasMsg);*

→ This function is used to handle a received RAS message by a gatekeeper client. The 1st argument is the handle to the client. The 2nd argument is the Handle to received RAS message.

* *EXTERN int GkClientSendMsg(ooGkClient *pGkClient, H225RasMessage*
**pRasMsg);*

→ This function is used to send a message on Gatekeeper client's RAS channel. The 1st argument is the handle to the client. The 2nd argument is the handle to RAS message to be sent.

* *EXTERN int GkClientSendGRQ(ooGkClient *pGkClient);*

→ This function is used to send Gatekeeper request message.

* EXTERN int GkClientHandleGatekeeperReject(ooGkClient *pGkClient,
H225GatekeeperReject *pGatekeeperReject);

→ This function is used to handle a received gatekeeper reject message. The 1st argument is the handle to the client. The 2nd argument is the handle to the received reject message.

* *EXTERN int GkClientHandleGatekeeperConfirm(ooGkClient *pGkClient,*
*H225GatekeeperConfirm *pGatekeeperConfirm);*

→ This function is used to handle a received gatekeeper confirm message. The 1st argument is the handle to the client. The 2nd argument is the handle to the received confirm message.

* *EXTERN int GkClientSendRRQ(ooGkClient *pGkClient, ASN1BOOL keepalive);*

→ This function is used to send Registration request message. The 1st argument is the handle to the client. The 2nd argument indicates whether keepalive lightweight registration has to be sent or not.

* *EXTERN int GkClientHandleRegistrationConfirm(ooGkClient *pGkClient,*
*H225RegistrationConfirm *pRegistrationConfirm);*

→ This function is used to handle a received registration confirm message. The 1st argument is the handle to the client. The 2nd argument is the handle to the received confirm message.

* *EXTERN int GkClientHandleRegistrationReject(ooGkClient *pGkClient,*
*H225RegistrationReject *pRegistrationReject);*

→ This function is used to handle a received registration reject message. The 1st argument is the handle to the client. The 2nd argument is the handle to the received reject message.

* *EXTERN int GkClientSendURQ(ooGkClient *pGkClient, struct OOAliases *aliases);*

→ This function is used to send UnRegistration request message. The 1st argument is the handle to the client. The 2nd argument is list of aliases to be unregistered. NULL, if all the aliases have to be unregistered.

* *EXTERN int GkClientHandleUnregistrationRequest(ooGkClient *pGkClient,*
*H225UnregistrationRequest *punregistrationRequest);*

→ This function is used to handle a received unregistration request message. The 1st argument is the handle to the client. The 2nd argument is the handle to received unregistration request.

** EXTERN int GkClientSendUnregistrationConfirm(ooGkClient*

**pGkClient, unsigned reqNo);*

→ This function is used to send an unregistration confirm message to gatekeeper. The 1st argument is the handle to the client. The 2nd argument is Request Sequence number for the confirm message.

** EXTERN int GkClientSendAdmissionRequest(ooGkClient *pGkClient, struct*

*OOH323CallData *call, ASN1BOOL retransmit);*

→ This function is invoked to request BW admission for a call. The 1st argument is the handle to the client. The 2nd argument is the handle to call. The 3rd argument indicates whether new call or retransmitting for existing call.

** EXTERN int GkClientHandleAdmissionConfirm(ooGkClient *pGkClient,*

*H225AdmissionConfirm *pAdmissionConfirm);*

→ This function is used to handle a received Admission confirm message. The 1st argument is the handle to the client. The 2nd argument is the Handle to received confirmed message.

** EXTERN int GkClientHandleAdmissionReject(ooGkClient *pGkClient,*

*H225AdmissionReject *pAdmissionReject);*

→ This function is used to handle a received Admission Reject message. It finds the associated call and marks it for cleaning with appropriate call end reason code. The 1st argument is the handle to the client. The 2nd argument is the handle to received admission reject message.

** EXTERN int GkClientSendDisengageRequest(ooGkClient *pGkClient, struct*

*OOH323CallData *call);*

→ This function is invoked to request call disengage to gatekeeper. The arguments are the handle to the client and the call respectively.

* *EXTERN int GkClientHandleDisengageConfirm(ooGkClient *pGkClient,*
*H225DisengageConfirm *pDCF);*

→ This function is used to handle a received disengage confirm message. The 1st argument is the handle to the client. The 2nd argument is the handle to received confirmed message.

* *EXTERN int GkClientRRQTimerExpired(void);*

→ This function is used to handle an expired registration request timer. The argument is the handle to callback data.

* *EXTERN int GkClientGRQTimerExpired(void);*

→ This function is used to handle an expired gatekeeper request timer.

* *EXTERN int GkClientREGTimerExpired(void);*

→ This function is used to handle an expired registration time-to-live timer.

* *EXTERN int GkClientARQTimerExpired(void);*

→ This function is used to handle an expired admission request timer.

* *EXTERN int GkClientCleanCall(ooGkClient *pGkClient, struct OOH323CallData*
**call);*

→ This function is used to clean call related data from gatekeeper client. The arguments are the handle to the client and the call respectively.

* *EXTERN int GkClientSetCallbacks(ooGkClient *pGkClient,*
OOGKCLIENTCALLBACKS callbacks);

→ This function is used internally to set Gatekeeper Clients callbacks. The 1st argument is the handle to the client. The 2nd argument is Callback structure containing various gatekeeper client callbacks.