2006

Video Quality Measurement For 3G Handsets

Enjapuri, Praveen

http://hdl.handle.net/10026.1/9324

http://dx.doi.org/10.24382/896
University of Plymouth

University of Plymouth

School of Computing, Communications & Electronics

Video Quality Measurement For 3G Handsets

M.Res Thesis

A dissertation submitted by Mr. Praveen Enjapuri **in partial fulfilment of the requirements of University of Plymouth for the Degree of Master of Research in Network Systems Engineering.**

Supervisors:

Dr.Lingfen Sun
Professor.Emmanuel Ifeachor

September 2006.

UNIVERSITY OF PLYMOUTH

CERTIFICATE OF COMPLETION OF THESIS/RESEARCH PAPER

Faculty in which student was registered **Computing, Communications & Electronics, Faculty of Technology**

Degree Followed **Master of Research**

This is to certify that Mr **Praveen Enjapuri**
(Name in Full)
a registered student of this university has completed his M.Res Project Report and three identical copies of the work have been produced in accordance with the requirements of the University and are acceptable for examination.

Title of the M.Res Thesis Report **Video Quality Measurement for 3G Handsets**

Name of Supervisor LINGFEN SUN Signature Lingfen Sun

Date: 22/9/2006

Name of Supervisor E.C. IFEACHOR Signature [illegible]

Date: 22/9/06

VIDEO QUALITY MEASUREMENT FOR 3G HANDSETS

Praveen Enjapuri

Abstract

Internet provides many services. VOIP (Voice over IP) is one such service also known as Internet Telephony or IP Telephony. Using VOIP we can make voice telephony calls, participate in video conferences, etc over data networks (WAN'S and LAN'S) or internet. VOIP operates by first converting voice data into digital form, organizing them into packets, transmitting them through the most convenient route to their destination and finally reassembling them at the destination. Protocols like SIP/RTP, H.323, MGCP are designed which perform all the above steps.

This project aims to make a video call from a 3G Mobile to an IP phone via Asterisk Gateway. Asterisk to act as bridge for video call between 3G-IP network must capture the audio/video stream from 3G mobile, convert captured stream into an IP compatible stream and send stream to an IP client and vice-versa. Asterisk needs to support AMR codec for audio and MPEG-4 codec for video and H.324M protocol stack for capturing audio/video streams from 3G Mobile. Asterisk currently supports audio codec's like GSM, G.729, A-law, and U-law. It allows H.261, H.263 video streams as pass-through. It supports VOIP protocols like SIP/RTP, MGCP, and H.323 which allows it to interface with other devices. This project aims to implement AMR codec, H.324M protocol stack, MPEG-4, bridging functions between SIP/RTP-ISDN and 3G Mobile in Asterisk which allows a 3G phone to call a SIP client via Asterisk. This thesis discusses the implementation of AMR in asterisk as well as SIP protocol and SIP soft phones.

Acknowledgements

While I worked on my M.Res project, several people had a positive influence on my progress and I would like to thank them in general. Some of them I would like to thank in particular. First, I thank my supervisors Dr.Lingfen Sun & Professor. Emmanuel Ifeachor. They initiated this project and gave me the opportunity to be a part of this project. They also spend their valuable time, in commenting and structuring the publication of my thesis report. I would also like to thank Mr. Zhuoqun Li for his help. I also take this opportunity to thank Motorola team Mr. Tim Robinson and Mr. Zizhi Qiao (William) for their help and belief that I can contribute something to this project. I also thank my friend Mr. Tajuddin for creating perfect ambiance for working on my project and thesis. Finally, I would like to thank my parents, for giving me the opportunity to study at the university and supporting me on sidelines.

Mr Praveen Enjapuri.

This work is supported in part by Motorola.

Table of Contents

List of Figures

List of Tables

Abbreviations

ACELP	Algebraic Code Exited Linear Prediction
AMR	Adaptive Multi Rate
API	Application Program Interface
AVI	Audio Video Interleave
BRI	Binary Rate Interface
CNG	Comfort Noise Generation
ETSI	European Telecommunications Standards Institute
GSM	Global System for Mobile communications
IETF	Internet Engineering Task Force
IP	Internet Protocol/Intellectual Property
IAX	Inter Asterisk Exchange Protocol
ISDN	Integrated Services Digital Network
ITU	International Telecommunications Union
LAN	Local Area Network
MGCP	Media Gateway Control Protocol
NAT	Network Address Translation.
PBX	Private Branch Exchange
PEVQ	Perpetual Evaluation of Video Quality
PESQ	Perceptual Evaluation of Speech Quality
PEAQ	Perceptual Evaluation of Audio Quality
POTS	Plain Old Telephone Service
PRI	Primary Rate Interface
PSNR	Pulse Signal to Noise Ratio
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RTCP	Real-Time Control Protocol
RTP	Real-time Transport Protocol
SIP	Session Initiation Protocol
SDP	Session Description Protocol
UA	User Agent
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
URI	Universal Resource Indicator
VAD	Voice Activity Detection
VOIP	Voice over IP
WAN	Wide Area Network
WCDMA	Wireless Code Division Multiple Access

1. Introduction

Implementing AMR codec in Asterisk and study of implementing bridging functions between ISDN B-channel and RTP/SIP in asterisk are the outcomes of this project thesis. This report elaborates on the implementation of these functionalities. Section 1.1 presents an introduction to VOIP. Section 1.2 states motivation for this M.Res project. Section 1.3 gives the precise project definition. Section 1.4 describes project goals and approaches for achieving these goals. Section 1.5 gives the outline of the thesis.

1.1 Introduction

VOIP converts analogue signals to a digital format (across a network) and then back into analogue at the receiver when using a phone.In simple words it is a technology that enables the transmission of voice over IP network.VoIP technology use packet switching to transmit only voice packets to the destination.It is very cheap when comparted to PSTN which uses complex circuit switching technology.Internet usage is increasing day by day and the connection cost is decreasing .With VOIP you can make a call from anywhere you have broadband connectivity.

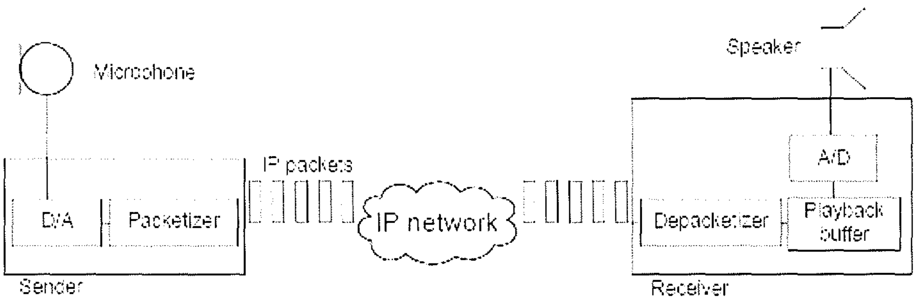


Figure 1.1 VOIP System

Figure 1.1 depicts a typical VOIP System and below is the explanation for each of its parts. At the sender side the voice data is sampled using a microphone. This sampled signal is translated into Digital representation by Digital to Analogue Converter. This digital representation is packed into IP packets and is sent along IP network. At the receiver end these packets are paketized (headers are removed) and then placed in playback buffer to compensate for the variation of delay over the network, called as jitter. Finally, an A/D (analogue-to-digital) converter converts the voice data back into an analogue signal.

1.2 Motivation:

Asterisk PBX is an Open Source VOIP product. It's a software implementation of hardware PBX and is very flexible. A PBX has features and capabilities that are not available with PSTN. Asterisk as already said is an open source product distributed under GPL Licence and features are being added to it all the time by the developers all over the world. A new Feature can be added to it any time and made available to other users. But Asterisk currently doesn't have all the features of PBX system. We wish to implement new functionalities in asterisk which enables it to act as H324M-SIP Gateway. The successful implementation of this functionality makes it possible for a video call from 3G Mobile phone to IP Phone through Asterisk. We succeeded in implementing couple of functionalities.

1.3 Project Definition:

The project goal is to make Asterisk a gateway between ISDN and IP network. My role in the project is to implement AMR Codec, bridging functions between ISDN B-Channel and RTP/SIP channel in Asterisk .This thesis only gives background information required to implement bridging functions between ISDN B-Channel and RTP/SIP channel.

1.4 Project Goals and Approach:

This project thesis aims to successfully implement AMR Codec in asterisk and give the basic information for implementing the bridging functions between ISDN B-Channel and RTP/SIP channel. As already mentioned the whole project actually intends to make Asterisk a gateway for video call. This requires the implementation of 4 other functionalities which are as below

1) Recording of video and audio streams from ISDN H.223 channel into Asterisk using H.324M protocol stack ;
2) Loop back of audio and video to 3G handsets, sending the video/audio streams to Web/Test server for quality testing
3) Implementing AMR Codec & MPEG-4 into Asterisk
4) Implementing bridging functions between RTP/SIP channel in Asterisk and ISDN B-Channel

Below is the schematic diagram of our project

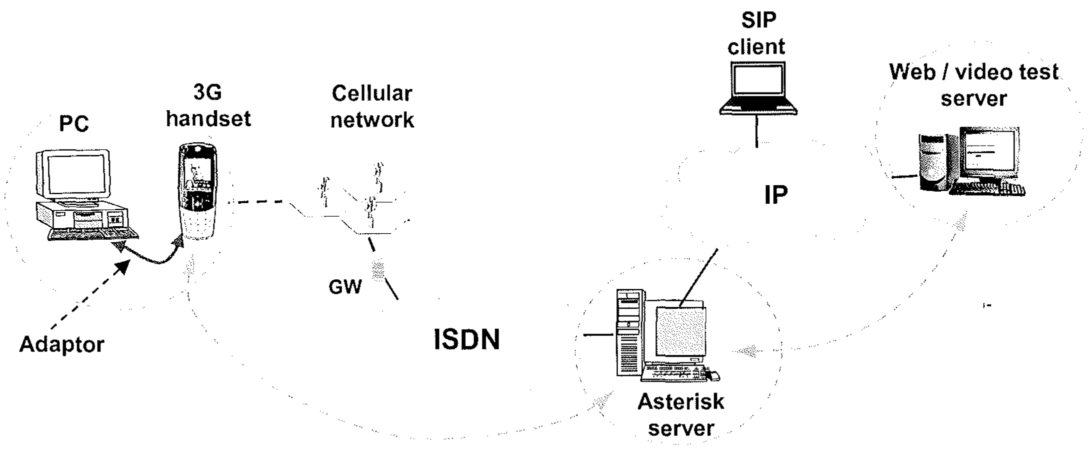


Figure 1.2: Schematic Diagram for 'Video Quality Measurement for 3G Handset' Project [1]

We took the following approach to achieve our goal

i) Understanding AMR speech codec.
ii) Understanding SIP/RTP/RTCP protocol Structure.
iii) Implementing AMR codec support in Asterisk Channels.
iv) Researched for the techniques to test the AMR implementation in asterisk.

1.5 Asterisk as 3G-H.324M & SIP Gateway

This chapter elaborates on how asterisk can act as 3G-H324M and SIP Gateway. Figure 2.1 depicts Asterisk as an ISDN – IP Gateway.

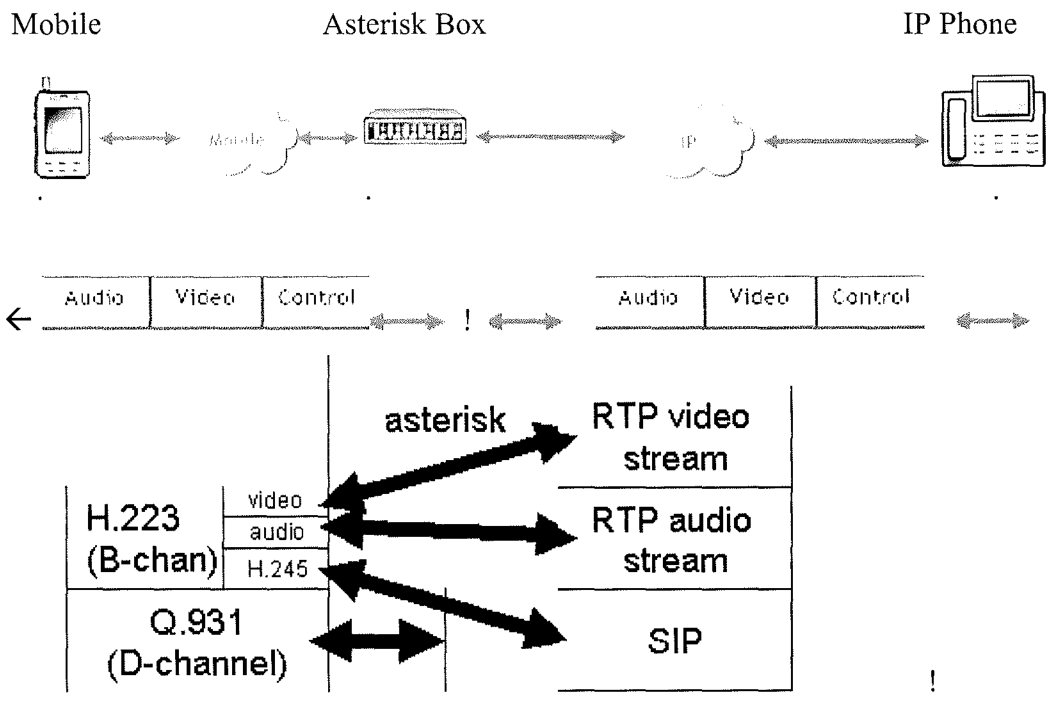


Figure 1.3: 3G-H324M/ISDN – SIP Video Call [1]

<u>3G Network</u> -

3GPP and 3GPP2 have adopted the 3G-324M protocol as standard for transporting conversational video over mobile network. The 3G-324M protocol combines voice, video, data and control into a single 64kbps stream of circuit-switched data. 3G-324M based video content is carried by a single H.223 64kbps stream that multiplexes audio, video, data and control information. The video portion of H.223 stream uses MPEG-4 codec, whereas audio content is based on AMR compression technique. The control portion of the H223 stream is based on H.245 protocol which controls the session and takes care of channel parameter exchanges.

IP Network -

The transportation of the real-time video in Voice over IP networks is done in a different way using three different UDP/IP Streams. RTP/RTCP is used to transport voice, video and control information on different streams. Two streams are used for voice and video and an RTCP stream for control. The video stream uses H.264/H.263/H.261 or MPEG4 compression technique and audio stream uses G.711, G.729, G.723 or any other voice compression scheme. The compressed audio and video streams are encapsulated in an RTP stream which is then placed in UDP/IP Packet. Signaling and control information is carried using either SIP (uses SDP for controlling info) or H.323 (which uses H.245 for control info).

Asterisk should do the following to act as a video gateway

i) Sip must be translated to H.245 and vice-versa

ii) RTP stream must be translated into corresponding AMR, MPEG-4 streams and vice-versa. (Converting one form of audio/video codec to another audio/video codec called as audio/video Transcoding)

iii) All of them must be multiplexed into a B-channel if a sip client is communicating with 3G mobile

To perform the above functions Asterisk PBX needs to support AMR, MPEG-4 and H324M. But currently it doesn’t. As already mentioned Implementing AMR codec and study for the implementation of bridging functions between ISDN B-channel and RTP/SIP in asterisk is the outcome of this project thesis

1.6 Thesis Organization

This thesis is organized as follows. Chapter 2 gives an overview of Asterisk PBX. Chapter 3 describes the AMR Codec and procedure followed to implement it in Asterisk. Chapter 4 gives an overview of SIP, RTP, and RTCP Protocols. It also gives information on configuring various Linux based SIP Soft phones. Chapter 5 describes audio/video quality measurement techniques for measuring audio and video quality using PESQ and PEVQ. Chapter 6 states the conclusions and gives recommendations for future research.

2 Asterisk –Open Source PBX

2.1 Introduction

This chapter describes the general architecture of Asterisk PBX, File System Organization, channels and various VOIP protocols it supports.

Section 3.2 gives the information about downloading, installing, loading Asterisk. Section 3.3 discusses its general architecture including its file organization. Later various VOIP Protocols supported by Asterisk are explored. Asterisk PBX is a open source software Licensed under GPL. Mark Spencer of Digium created Asterisk. It is completely written in c. It allows telephones (IP phones or hard phones) connected to it to call each other and also to PSTN. Asterisk thus can act as a registrar for IP phones, gateway between IP phones and PSTN. Asterisk supports VOIP protocols like SIP, H.323 and has its own protocol called as IAX used for efficient switching of call among Asterisk servers.

2.2 Software Installation

2.2.1 Requirements

Asterisk works with most Linux distributions like REDHAT, Fedora, Debian, Mandrake etc and some non-Linux Operating Systems like Solaris, BSD and OS X. This project "Video Quality Measurement for 3G Handsets" is implemented on SUSE Linux Distribution, Version 10.0. Asterisk is designed to work on kernel Version 2.4 of Linux but it works on version 2.6 also. We must install below packages before installing asterisk.

i) Linux2.4 kernel sources

ii) Bison and Bison-devel packages

iii) Open-ssl and Open-ssl devel packages.

iv) Z-lib and Z-lib devel packages

v) ncurses and ncurses devel packages used for CLI functionality.

Although Asterisk package is the only real requirement for VOIP network it requires Zaptel Telephony drivers and PRI libraries. Zaptel drivers are required if we are using analogue or digital drivers or if we are using ztdummy as a timing interface. Libpri library is optional unless we use ISDN PRI interfaces.

2.2.2 Downloading

Asterisk software is made available by DIGIUM in two categories. The one which is used in called stable and the other which is used for testing new features and bugs called as head release. Stable releases can be obtained via FTP. We will be using only stable branch for our project. Stable and head branches can also be obtained from CVS but the stable release from CVS may be buggy.
Downloading from CVS - Stable release can be downloaded from CVS with following set of commands on Linux console.[2]
Export the CVSROOT path:

```
# cd /usr/src/
# export CVSROOT=:pserver:anoncvs:anoncvs@cvs.digium.com:/usr/cvsroot
```

Downloading from FTP - Asterisk stable release can be obtained from DIGIUM FTP server located at ftp://ftp.digium .com using wget program. We need to enter the following commands on Linux konsole

```
# cd /usr/src/
# wget --passive-ftp ftp.digium.com/pub/asterisk/asterisk-1.*.tar.gz
# wget --passive-ftp ftp.digium.com/pub/asterisk/asterisk-sounds-*.tar.gz
# wget --passive-ftp ftp.digium.com/pub/zaptel/zaptel-*.tar.gz
# wget --passive-ftp ftp.digium.com/pub/libpri/libpri-*.tar.gz
```

2.2.3 Compiling Asterisk:

Asterisk uses three main packages the main Asterisk Program, Zapata Telephony Drivers, and the PRI Libraries. Asterisk is the only requirement for a VOIP System. But it is recommended to install libpri and zaptel packages before installing asterisk. Asterisk doesn't require a special hardware like a soundcard. Asterisk can be compiled by using GNU make program. We need to execute following commands in sequence on the Linux console. We need to have root privileges to compile and run asterisk. [2]

```
cd /usr/src/asterisk-version no
make clean
make
make install
make samples
```

Executing make clean command removes the compiled binaries from within the source directory. This command should be executed before we attempt to compile asterisk. Running make samples command will install default configuration files. If we already have configuration files installed in /etc/asterisk, running the above command will append .old to the end of current configuration files.

2.2.4 Loading Asterisk:

Asterisk can be started in different ways. Asterisk binary file is located at /usr/sbin/asterisk by default .Asterisk can be loaded by running this binary file directly from Linux console (CLI). "Asterisk" Command helps us to connect to asterisk CLI, set the verbosity of CLI output and allow core dumps if asterisk crashes. Below are the most commonly used options with this command.

-c:	This option allows us to connect to CLI
-v:	Verbosity. This option sets the amount of output for CLI debugging.
-g:	This is used for Core dump in case asterisk crashes.
-rx:	Restart now. Allows us to execute a CLI command without having to connect to CLI.

Ex: asterisk –vvvgc: Executing this command starts asterisk and connects to CLI with debugging level of 3.

2.3 Asterisk Structure

Asterisk connects telephony technologies (like SIP, H323, IAX, MGCP) with telephony applications (include bridging, conferencing, voicemail etc).

2.3.1 Asterisk Architecture:

The Figure 3.1 below is the general architecture of asterisk.

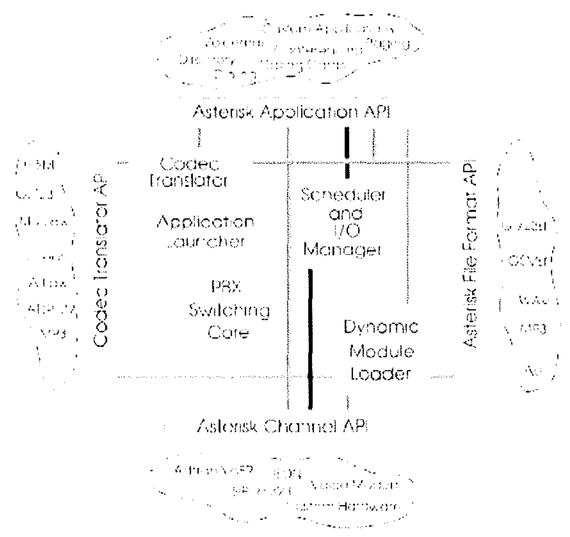


Figure 2.1: Asterisk Architecture [4]

API's like channel API, Application API, Codec Translator API, File Format API are defined around Asterisk Core system. Asterisk Core handles internal interconnection of PBX whereas Loadable Module API's take care of protocols, hardware interfaces. Whenever asterisk starts Dynamic Module Loader is loaded which initializes the channels drivers, file format, codec, and applications.

2.3.1.1 Asterisk Core:

PBX Switching, Application Launcher, Codec Translator and Scheduler and I/O manager constitute central PBX core system for asterisk.

PBX switching - PBX switching core accepts calls arriving on various software and hardware interfaces. It then handles the calls according to the dial plan. PBX Switching Core also uses Application Launcher to play ring tones, to connect to voicemail etc.
Application Launcher - Application Launcher starts various applications which perform services for users, such as voicemail, file playback etc.
Codec Translator- Codec Translator connects different channels which are compressed with different codec's.
Scheduler and I/O Manager - Asterisk core includes a scheduler I/O manager which handles low-level tasks for optimal performance of the system.

2.3.1.2 Loadable Module API's:

Asterisk is highly modularized. Various API'S facilitate hardware and protocol abstraction. Below four API's allow asterisk to integrate with any telephony hardware.

Channel API - A phone call made through asterisk consists of incoming connection and outgoing connection. Asterisk has drivers that support a particular technology like SIP, ZAP, and IAX2. These Loadable module API's are called channel drivers. The call comes to asterisk through a channel driver. Asterisk loads these channel drivers dynamically.

Application API - Application API provides the flexibility for application modules to perform functions on demand, and allows for open development of new applications.

Codec Translator API-It loads various codec modules defined in /usr/lib/asterisk/modules directory. They support various audio and video decoding formats such as GSM, A-law, Mu-Law and even mp3.

File Format API- File format API handles reading and writing of various file formats for storage of data in file system.

2.3.2 File System Organization:

This section discusses the necessary directories which are created during asterisk installation. The following table describes where Asterisk related files are stored.[3]

Table 2.1: Asterisk File Organization

Directory	Description
/etc/asterisk	It has all the asterisk configuration files except /etc/zapatel.conf
/var/run	Runtime named pipes and PID files
/var/run/asterisk/ctl	Named pipe used by asterisk to enable remote operation
/var/spool/asterisk/	Directory contains Files for voicemail, out calls
/var/run/asterisk.pid	Primary process identifier (PID) of the running Asterisk process
/var/lib/asterisk/sounds	Contains audio files ,prompts used by Asterisk
/var/lib/asterisk	Variable data used by asterisk during normal operation.
/var/include/asterisk	Header files required for building asterisk applications,

	channel drivers and other loadable modules.
/usr/lib/asterisk/	Contains Asterisk architecture specific binary objects
/var/lib/asterisk/agi-bin	AGI scripts used by dial plan AGI applications

Files of the form app_ represent applications, chan_ represent channel drivers, res_ represent resources.

2.3.3 Configuration Files

Configuration files in asterisk contain channel definitions, information describing internal services, information regarding location of other modules, or dialplan. Asterisk contains a configuration file for each module. A configuration file needs to be reloaded whenever changes are made to it. Below are some important configuration files in asterisk.

modules.conf - Whenever asterisk is started Dynamic Module Loader loads modules defined in modules.conf file. This file controls which module should be loaded and which should be not. The statement format to do this is
load => or noload=> constructs

This file always starts with [modules] header. Below is the sample modules.conf file
Ex :

```
[modules]
autoload=yes
;noload => pbx_gtkconsole.so
;load => pbx_gtkconsole.so
;Load either OSS or ALSA, not both
;By default, load OSS only (automatically) and do not load ALSA
noload => chan_alsa.so
;noload => chan_oss.so
```

The autoload statement tells Asterisk whether to automatically load all modules directly or to load only selected modules.

sip.conf - **SIP** clients are configured in this file. A client can be configured in this file to call another sip client or to receive a call from other sip client via Asterisk. A sip client can be configured to accept or receive calls by defining some basic parameters shown below.

username:	This parameter sets the username for the client. Asterisk uses this parameter to connect when it receives a call.
canreinvite:	This option tells asterisk to never send a reinvite message to client
context:	This parameter if used sets the default context for this client only .
type:	type parameter defines the type of connection class for the sip client. This parameter has three options (peer/user/friend).
peer:	A peer type receives calls from Asterisk server.
user:	A user type makes calls through Asterisk server.
friend:	A friend type can receive and make calls through Asterisk .
secret:	This parameter sets the password for the client. The client should supply this password when registering with Asterisk.
host:	This parameter can be set to ip address or resolvable host name of the device. If this is set to *'dynamic'* it means that connection request is expected to come from any IP address.
defaultip:	This parameter is used when the host parameter is set to 'dynamic'. If set Asterisk will sends calls to this ipaddress whenever it receives a call for a sip client that is not yet registered.

Below is an example where we configured sip.conf file so that sipclients (Kphones) can communicate with each other.

```
[general]
port=5060
```

```
 bindaddr=192.168.0.8
 context=default


[praveen]
type=friend
secret=uop
username=neo
host=dynamic
defaultip=141.163.8.211
context = trusted


[zeeshan]
type=friend
username=waheed
secret=uop
host=141.163.8.216
canreinvite=no
context=untrusted.
```

With the above configuration a sip client with username either neo/waheed with password uop can register with Asterisk and can call another sip client registered with asterisk

extensions.conf – This configuration files contain dialplan logic of Asterisk. Asterisk routes the calls by reading this file. We edited this file so that asterisk handles the call between two sip phones. This file is divided into sections called as contexts. A context is a group of extensions that route calls to specific channels, applications, or other extensions with in local or remote dialplans. An extension is broken into four parts called as number, priority, application, special argument that is passed to application. Below is the general syntax of an extension

```
exten => number,priority,application[,arguments]
```

Ex: exten => 6161,2,Voicemail,u6161

Below is the general syntax of a context.

```
              [context1]
exten => number,priority,application[,arguments]
exten => number,priority,application[,arguments]
```

Below is the syntax of a diaplan.

```
             [general]
 -->settings
            [globals]
--> declaration of gloabal variables

          [context1]
-->extension1,priority1,application
 -->extension1,priority2,application
          [context2]
 -->extension999,priority1,application
 --> extension 999, priority1,application
```

Below is a sample extensions.conf file

Ex:

```
    [default]
exten=>steve,1,Dial(SIP/praveen);
exten=>mark,2,Dial(SIP/zeeshan);
    [uop]
exten=>s,1,Answer
exten=>s,n,Background(welcome)
exten=>2,1,Hangup
```

```
   [test]
exten=>s,1,Ringing
exten=>s,n,Wait,2
exten=>s,n,Background(options)                                    ;
exten=>1,1,Goto(default,praveen,1)
exten => 2,1,Goto(default,zeeshan,2)
```

iax.conf – IAX is used to pass calls between Asterisk servers. The clients which connect to asterisk using IAX protocol are configured here. Like sip.conf this configuration file is also divided into contexts. Each context defines general parameters like port Asterisk will listen, to use jitter buffer, which audio codec's are allowed and which are disallowed, etc for each connection. Below is a sample iax.conf file

```
   [general]
bandwidth=low
disallow=lpc10
jitterbuffer=no
forcejitterbuffer=no
tos=lowdelay
autokill=yes


[zeeshan]
type=user
context=test
callerid="GuestIAXUser"

 [iax-test]
type=peer
username=praveen
```

```
host=dynamic
trunk=yes
context=test
```

We made the above modifications to the iax.conf file so that two iax clients(here Praveen & Zeeshan) can communicate with each other.

2.4 Asterisk and VOIP

2.4.1 VOIP Asterisk Channels

Asterisk Channel is a form of interface between Asterisk and Outside Resource (VOIP provider, IP Telephone, phone line). It simply represents connection to the protocol Asterisk supports. Asterisk channel has the name corresponding to the VOIP protocol technology it supports. Ex: SIP, ZAP, IAX2. Each channel has its own features and asterisk can convert from one type of channel to another. A phone call made to asterisk comes in via a channel that supports one of the technologies like SIP, ZAP, IAX2. They each have their own structure which is defined in channel.h file and handled by channel.c file.

2.4.2 VOIP Protocols

VOIP call starts with signalling exchanges between the participants (includes gateways in between), ends with transportation of media streams (in one or both directions that carry the actual conversation). Several VOIP protocols available today can handle all this mechanism. Asterisk supports VOIP protocols such as SIP, IAX, H.323, MGCP.

IAX - IAX stands for Inter-Asterisk exchange Protocol. This protocol can be used to establish communication between two Asterisk Servers. Its current version is 2. It's a peer-peer (state maintenance) transport protocol which uses a single UDP port (4569) (unlike SIP and H.323) for transmitting both signaling and RTP streams [3]. Trunking multiple sessions is the unique feature of IAX resulting in minimum bandwidth. IAX carries media, sequencing and timing information in IAX frames. It uses single UDP port to convey this information so it does not have any NAT problems.

IAX is a binary protocol and the messages transmitted are called frames. There are two types of frames that are transmitted. They are Full-Frame, Mini-Frame. A Full-Frame is used to send signaling, audio, video information reliably which means the recipient returns a acknowledgement upon receiving the message.

```
                     1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------+-+-+-+-----------+-+-+-+---------+-+-+-+-----------+-+
|F|     Source Call Number      |R|   Destination Call Number   |
+---------+-+-+-+-----------+-+-+-+---------+-+-+-+-----------+-+
|                           Timestamp                           |
+---------+-+-+-+-----------+-+-+-+---------+-+-+-+-----------+-+
|     OSeqno    |    ISeqno     |   Frame Type  |C|  Subclass   |
+---------+-+-+-+-----------+-+-+-+---------+-+-+-+-----------+-+
|                                                               |
|                             Data                              |
|                                                               |
```

Figure 2.2 Full Frame Binary Format [3]

Figure 3.2 depicts the binary format of the Full-Frame. Below Table describes the each field in the Full-Frame.

Table 2.2: Field Descriptions for Full Frame

FIELD	**DESCRIPTION**
F	Set to 1 indicating that this is a full frame
Source Call Number	Call Number of the transmitting side of the full frame
R	Set to the value 1 if this frame is being retransmitted and the value 0 for the initial transmission.
Destination |Call Number	Call number of the receiving side of the Full Frame
Time Stamp	Full 32 bit Time stamp

OSeqno	Outbound stream sequence number
ISeqno	Inbound stream sequence number
Frame Type	Indicates Frame Type
C	Sub class value format
Sub class	Sub class

A Mini Frame is used to send media with a minimal protocol overhead. Below Figure 3.3 is the binary format of the Mini Frame.

```
                     1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-----------+-+-+-------------+-+-+-----------+-+-+-----------+-+
|F|    Source Call Number       |           Timestamp             |
+-+-----------+-+-+-------------+-+-+-----------+-+-+-----------+-+
|                                                                 |
|                              Data                               |
|                                                                 |
+-+-----------+-+-+-------------+-+-+-----------+-+-+-----------+-+
```

Figure 3.3: Mini-Frame Binary Format[4]

Field value F is set to zero indicating that the frame transmitted is not a full frame. Source Call Number field contains call number of the transmitting side of the mini frame. Time stamp field contains 16-bit time stamp.

SIP- SIP, or the Session Initiation Protocol, is specified by IETF. Its text based signalling protocol. It has syntax similar to that of HTTP and SMTP. SIP along with RTP is discussed in detail in section 5. Asterisk supports sip through chan_sip.so module, and the sip clients are defined in the configuration file sip.conf file.

MGCP - The Media Gateway Control Protocol (MGCP) is also specified by IETF. Asterisk supports MGCP through chan_mgcp.so module, and the clients are defined in the configuration file mgcp.conf file. But the current version of asterisk cannot register to a remote MGCP server.

H.323 - H.323 is specified by the ITU-T (International Telecommunication Union Standardization). H.323 uses the RTP protocol for media communications. H.323 has the same NAT problem as the SIP. There are two implementations for H.323 that can be used with Asterisk: asterisk-oh323 and chan_h32. chan_h323 in asterisk acts as a gateway. On the other hand asterisk-oh323 driver in asterisk is complete c implementation which helps to setup only H.323 signalling channels and asterisk handles all the media transportation.

2.4.3 Codec's & File Formats

A codec compresses analogue voice into digital data and decompresses the digital data into analogue signal. Most of the codec's which Asterisk supports don't require any license agreement but some like G.729 required to be licensed before they can be used. Asterisk currently supports the following codec's [3]

Audio:

- G.711 u-law as used in US
- ADPCM
- Slin
- GSM
- G.711 a-law as used in Europe
- G.723.1 - pass-thru for people who need a license , free for other people
- G.726
- G.729 - may require a license unless using pass-thru, free version available for use in countries without patents or for educational use only
- iLBC
- LPC10 (not recommended!)
- Speex - configurable 4-48kbps

Image

- Jpeg
- Png

Asterisk can translate between any of the above codec's. Asterisk doesn't support any video codec's. Asterisk passes video from either file to an IP port or from one IP port to another file.

File formats are related to files used to store audio data. File formats are used when user leaves a voicemail or listening to Music on Hold. Thus File formats deal with files and codec's deal with live media streams in Asterisk. Formats do not have any associated configuration files. Below Table 3.3 gives the information about the formats supported by asterisk and their corresponding files.

Table 2.3: Asterisk Supported Formats

Format Supported	Corresponding File
g723sf	*format_g723.so*
g726-40, g726-32, g726-24, g726-16	*format_g726.so*
g729	*format_g729.so*
h263	*format_h263.so*
ilbc	*format_ilbc.so*
jpg	*format_jpeg.so*
pcm	*format_pcm.so*
alaw	*format_pcm_alaw.so*
sln	*format_sln.so*
vox	*format_vox.so*
wav	*format_wav.so*
wav49	*format_wav_gsm.so*
gsm	*format_gsm.so*

2.5 Summary

Asterisk with its configuration files, extensions, dialplans, codec's, VOIP protocol support provides features and capabilities to connect to any external component. But Asterisk with its complex architecture needs to be configured according to the requirement. We need to know the structure of Asterisk, its File System organization (configuration files, channels) which is discussed in this chapter. For a SIP call between two SIP clients via Asterisk sip.conf file has to be modified. For any new functionality in the asterisk a module needs to be implemented in asterisk (not in all cases). This project intends to implement AMR codec, MPEG-4 codec, H.324M protocol stack into Asterisk, SIP-ISDN bridging functions for the video call. Next chapter gives the overview of AMR codec and its implementation in Asterisk.

3 Adaptive Multi-Rate Speech Codec

3.1 Introduction

AMR stands for Adaptive Multi Rate Codec. AMR is a popular speech codec used in VOIP applications. Adaptive Multi-Rate is originally standardized by ETSI for the GSM cellular system .It is now adopted as the mandatory standard speech codec by 3GPP.

The 13 kbit/s Full-Rate (FR) codec was the first voice codec defined for GSM. It has a bit rate of 22.8kbits/sec. The codec was standardised in 1989. Later Half-Rate (HR) codec was chosen to provide channel capacity savings through operation in the half rate channel. It was standardized in 1995. It has a bit rate of 11.4 kbits/sec. Both FR and HR Codec provided same level of speech quality. Enhanced Full-Rate (EFR) codec with a bit rate of 12.2 kbits/sec jointly developed by Nokia and University of Sherbrooke brought a substantial speech quality (equivalent to that of a wireline telephony). [5]

Later in 1999 GSM standardized AMR NB codec. The AMR codec was jointly developed by Ericsson, Nokia and Siemens. The AMR codec offers better error robustness in the full-rate channel by adapting speech and channel coding depending on prevailing channel conditions. The AMR wideband speech codec, jointly developed by Nokia and Voice Age is the recent standardization for GSM and WCDMA 3G systems. Nokia/Voice Age codec was selected as the 3GPP/ETSI AMR-WB codec in December 2000.
We used AMR-NB for implementation in Asterisk.

3.1.1 GSM –AMR

GSM-AMR or simply AMR is just another name for AMR-NB. The AMR codec is a multi-mode codec that supports 8 narrow band speech encoding modes with bit rates between 4.75 and 12.2 kbps. These bit-rate modes are also denoted with indices from 0 to 7 where 0 maps to the 4.75 kbits/s mode and 7 maps to the 12.2 kbits/s. Frequency used in AMR is 8000 Hz and the speech encoding is performed on 20 ms speech frames.

Therefore, each encoded AMR speech frame represents 160 samples of the original speech. AMR uses different techniques, such as Algebraic Code Excited Linear Prediction (ACELP), Discontinuous Transmission (DTX), Voice Activity Detection (VAD) and Comfort Noise Generation (CNG).

In a typical telephone conversation, voice transmission alternates regularly between both sides, leaving long pauses of silence. These can be more efficiently represented as background noise that is transmitted at a much lower bit rate. The discontinuous transmission mode (also called source controlled rate operation) is used to encode frames that contain only background noise. When operating in DTX mode, a voice activity detector (VAD) on the TX side evaluates whether a 20 ms frame contains any voice data. In the absence of speech, a silence identifier (SID) frame is transmitted, which contains characteristics describing the background noise. On the RX side, a "comfort noise generator" is used to synthesize background noise based on the SID frame parameters.[6]

AMR Narrowband / NB uses the ACELP compression format in the 3GP container among other coding techniques, and QCELP is used for AMR Wideband in the 3GPP2 standard. The AMR-NB bandwidth is 200-3400 Hz, its sample rate is 8 kHz. Typical bit rates are 4.75-12.2 kbps/mono (including second generation cellular standards). [8]
AMR codec was designed to allow seamless switching on a frame by frame basis between the different modes.

Table 3.1: AMR Codec Modes

Channel	Source codec bit-rate
TCH/FS/AMR (TCH/AFS)	12.2 kbit/s (GSM EFR)
	10.2 kbit/s
	7.95 kbit/s
	7.40 kbit/s (IS136 EFR)
	6.70 kbit/s
	5.90 kbit/s
	5.15 kbit/s
	4.75 kbit/s
TCH/HS/AMR (TCH/AHS)	7.95 kbit/s
	7.40 kbit/s (IS136 EFR)
	6.70 kbit/s
	5.90 kbit/s
	5.15 kbit/s
	4.75 kbit/s

3.1.2 AMR-WB

Adaptive Multi Rate - WideBand or AMR-WB standard is developed using ACELP technology.The AMR-WB Codec has been standardized by the ITU-T standards body and is referred to as G.722.2.This speech coder is mainly used for speech compression in the 3rd generation mobile telephony. It is the first codec to be adopted for both wireless and wire line services. This resulted in a single wideband codec for GSM, WCDMA 3G and ITU-T. This codec has been standardized as the default codec for "speech" media type at 16 kHz sampling frequency for Packet Switched Streaming Service (PSS) [15] and Multimedia Messaging Service (MMS).[8]

AMR-WB codec operates with nine basic bit rates, 23.85, 23.05, 19.85, 18.25, 15.85, 14.25, 12.65, 8.85 and 6.6 kbit/s. AMR-WB provides a speech bandwidth of 50 - 7000 Hz with sampling rate of 16khz [10]. AMR-WB provides an excellent speech quality compared to narrowband speech codecs which are optimized for POTS and they provide high quality speech with modes 12.5kbps or higher.The two lowest modes 8.85 and 6.6 kb/s are intended to be used only temporarily during severe radio channel conditions or during network congestion.

This codec works on the principle of Algebraic Code Excited Linear Prediction (ACELP) for all bit rates. To reduce average bit rate, this codec supports the discontinuous transmission (DTX), using Voice Activity Detection (VAD) and Comfort Noise Generation (CNG) algorithms. In addition to fixed rate speech and channel codec modes AMR-WB also includes a background noise mode that is designed to be used in discontinuous transmission (DTX) operation in GSM and as a low bit rate source-dependent mode for coding background noise in other systems. In GSM the bit rate of this mode is 1.75 kbps [14].

The coder works on a frame of 320 speech samples (20 msec), and a look ahead of 5 msec is required. So the algorithmic delay for the coder is 25 msec. Figures 4.1 and 4.2 describe the encoder and decoder of GSM-AMR .The output file from the AMR-NB encoder has either .AMR extension or .COD extension. .AMR files are stored in "AMR File Storage Format1" (discussed in section 3.3.1). These types of files are handled by the Ericsson AMR tool, as well as by Nokia Series 60 phones. They have a header of "#!AMR\n". Others have ".COD" extension and are stored in "AMR Interface Format 2" (Discussed in section 3.3.2). This is specified in 3GPP TS 26.101, Appendix A. These are coded and decoded as per 3GPP TS 26.104 floating point reference codec source package. AMR IF1 and IF2 are discussed in the later sections. Below are the differences between these two file types.

- '.AMR' files have a 6 byte header: "#!AMR\n" where as '.COD' files don't have a header.

- .AMR files take two nibbles (8 bits) to express the frame flags [Format: "PTTTTVPP", where P is "Pad" (0), T is "Frame Type", and V is "Valid"]. Where as '.COD' files require a single nibble to express the frame flags. The actual speech frame bits begin directly after this, in the 2nd nibble of the first byte of the frame.

- '.AMR' files pack the bits of the frame into bytes in big-endian order, that is, the most significant bit (0x80) of each byte is actually the first bit of the byte, and the least significant bit (0x1) is the last bit. '.COD' files pack the bits of the frame into bytes in little-endian order. The least significant bit is the first bit of the byte, followed by the 0x2 bit, the 0x4 bit, etc.

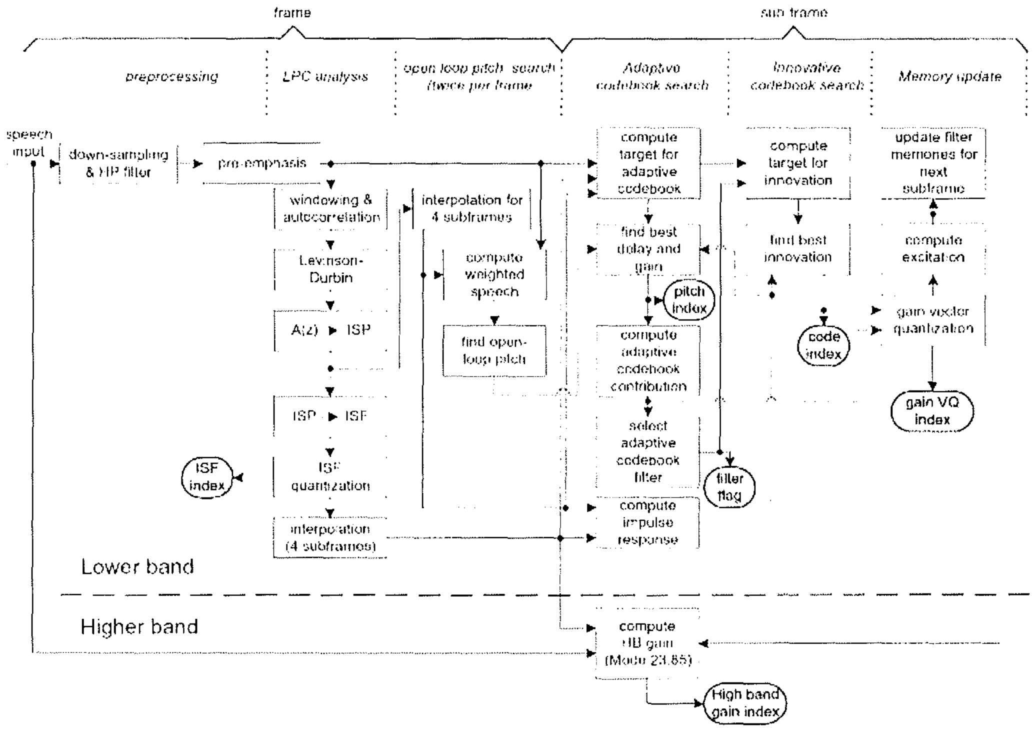


Figure 3.1: Block diagram of the AMR-WB ACELP encoder [4]

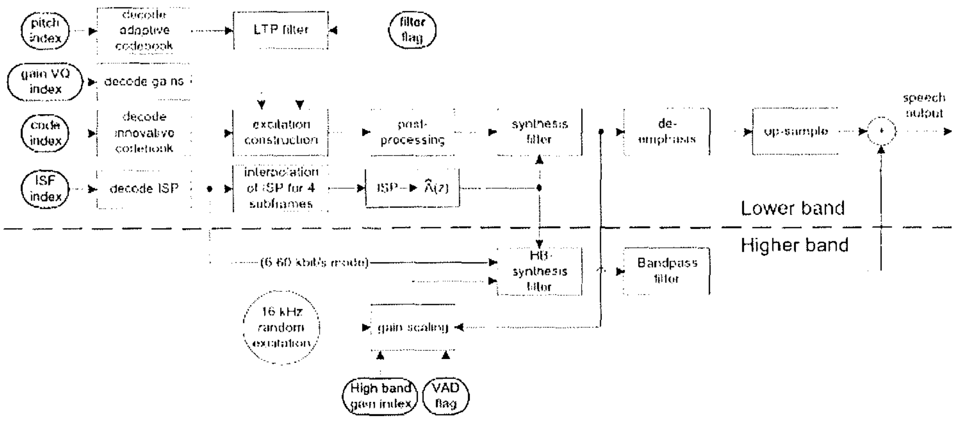


Figure 3.2: Block diagram of the AMR-WB ACELP decoder [4]

3.1.3 AMR-WB+

AMR-WB+ is an audio codec that extends AMR-WB. AMR WB+ codec is standardized in February 2004 by ETSI/3GPP. AMR-WB+ utilizes a hybrid of two technologies: ACELP® and TCX, respectively Algebraic Code Excited Linear Prediction® and Transform Coded Excitation. The AMR-WB+ codec has a wide bit-rate range, from 6-48 kbps. Mono rates are scalable from 6-36 kbps, and stereo rates are scalable from 7-48 kbps. It is also compatible with AMR-WB [10].

This audio codec is recommended by 3GPP for MMS, PSS and MBMS services over GSM and WCDMA networks.

3.2 ETSI AMR VS 3GPP AMR

There is no algorithmic difference between ETSI-AMR and 3GPP-AMR for AMR-NB and AMR-WB. 3GPP-AMR support packing of coded bits and unpacking of bits before decoding. This packing and unpacking is done as per RFC3267.

In ETSI AMR DTX ON, 245th word in coded bit stream is always value of mode i.e. from 0 to 8, where 8 for DTX mode. In 3GPP AMR for DTX ON, 245th word is value of mode for non DTX frame but it is equal to -1, if DTX frame is found.

In ETSI one flag called NSYNC is used as input from the n/w. In 3GPP this is not used. The NSYNC is used in mobile communication when switching from one Base station to next base station. This alters the VAD operation. When ever NSYNC is set, the mode will not be NODTX. [17]

AMR is generally referred as GSM AMR or AMR-NB

3.3 AMR Frame Structure for AMR-NB:

3.3.1 AMR Interface Format 1 (AMR IF1):

The AMR frame consists of three parts: AMR Header, AMR Auxiliary Information, and AMR Core Frame. Table below is the frame composition called as interface format1 for AMR codec.

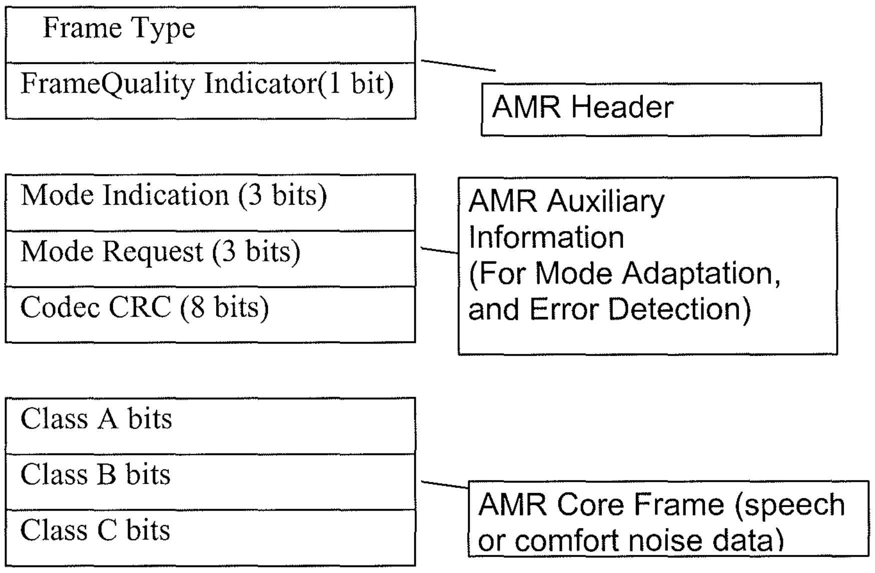

Frame Type
FrameQuality Indicator(1 bit)
AMR Header
Mode Indication (3 bits)
Mode Request (3 bits)
Codec CRC (8 bits)
AMR Auxiliary Information (For Mode Adaptation, and Error Detection)
Class A bits
Class B bits
Class C bits
AMR Core Frame (speech or comfort noise data)

Figure 3.3: Generic AMR frame structure [5]

The AMR Header part includes the Frame Type and the Frame Quality Indicator fields. The AMR auxiliary information part includes the Mode Indication, Mode Request, and Codec CRC fields. The AMR Core Frame part consists of the speech parameter bits or, in case of a comfort noise frame, the comfort noise parameter bits. In case of a comfort noise frame, the comfort noise parameters replace Class A bits of AMR Core Frame while Class B and C bits are omitted.

3.3.1.1 AMR Header

The length of the AMR header field is 4 bits. The AMR Header part includes the Frame Type and the Frame Quality Indicator fields. Frame Type indicates the use of one of the eight AMR codec modes, one of four different comfort noise frames or an empty frame. The remaining three indices are reserved for future use.

Table 3.2: Frame Type Field Contents [5]

Frame Type	Frame content (AMR mode, comfort noise, or other)
0	AMR 4,75 kbit/s
1	AMR 5,15 kbit/s
2	AMR 5,90 kbit/s
3	AMR 6,70 kbit/s (PDC-EFR)
4	AMR 7,40 kbit/s (TDMA-EFR)
5	AMR 7,95 kbit/s
6	AMR 10,2 kbit/s
7	AMR 12,2 kbit/s (GSM-EFR)
8	AMR SID
9	GSM-EFR SID
10	TDMA-EFR SID
11	PDC-EFR SID
12-14	For future use
15	No Data (No transmission/No reception)

Frame Quality Indicator: The field length of the frame quality indicator is one bit. This indicates whether the data in the frame contains errors. Table defines the frame quality indicator

Table3.3: Frame Quality Indicator Definition [5]

FrameQuality Indicator (FQI)	Quality of data
0	Bad frame or Corrupted frame (bits may be used to assist error concealment)
1	Good Frame

3.3.1.2 AMR Auxiliary Information

The AMR auxiliary information part includes the Mode Indication, Mode Request, and Codec CRC fields.
Mode Indication and Mode Request: Mode Indication and Mode Request fields which are 3-bit fields each and are defined only in the range 0…7 to specify one of the eight AMR codec modes.

Table3.4: Frame Type, Mode Indication and Mode Request field Contents [5]

Mode Indicatio n	Mode Request	Frame content (AMR mode, comfort noise, or other)
0	0	AMR 4,75 kbit/s
1	1	AMR 5,15 kbit/s
2	2	AMR 5,90 kbit/s
3	3	AMR 6,70 kbit/s (PDC-EFR)
4	4	AMR 7,40 kbit/s (TDMA-EFR)
5	5	AMR 7,95 kbit/s
6	6	AMR 10,2 kbit/s
7	7	AMR 12,2 kbit/s (GSM-EFR)

Table3.5: Mapping of FQI and FT to TX_TYPE and RX_TYPE

Frame Quality Indicator	Frame Type Index	TX_TYPE or RX_TYPE	Comment
1	0-7	SPEECH_GOOD	FTI depends on the bit-rate Being used.
0	0-7	SPEECH_BAD	Friedens on the bit-rate being used. The corrupted data may be used to assist error concealment.
1	8 8	SID_FIRST or SID_UPDATE	SID_FIRST and SID_UPDATE are differentiated using One Class A bit: STI.
0	8	SID_BAD	
1	9-11	SID_UPDATE	
0	9-11	SID_BAD	
1	15	NO_DATA	Typically a non-transmitted frame or an erased or stolen frame with no data usable To assist error concealment.

Codec CRC: Codec CRC is an 8 bit value used for error-detection purposes. The codec CRC field of AMR Auxiliary Information block contains this value. This value is generated by the cyclic generator polynomial:-$G(x)=D^8 + D^6 + D^5 + D^4 + 1$ [5]
and is computed over all Class A bits of AMR Core Frame. This Codec CRC value is generated for Generic AMR codec frames with Frame Type 0 to 11.This Codec CRC value is not included when the Frame type is 15.

3.3.1.3 AMR Core Frame

The AMR Core Frame part consists of the speech parameter bits or comfort noise parameter bits (in case of a comfort noise frame). In case of a comfort noise frame, the comfort noise parameters replace Class A bits of AMR Core Frame while Class B and C bits are omitted.

AMR Core Frame Speech Bits: AMR Core Frame carries the coded speech data. These are generated for each of the Frame type indices from 0-7. The encoder generates bits which are denoted as {s(1),s(2),...,s(K)}, where K refers to the number of bits produced by the speech encoder. These speech bits are ordered according to their subjective importance. Below Table define AMR IF1 bit ordering for all the eight AMR codec modes. These tables should be read line by line from left to right. The first element of the table has the index 0. Speech bits are numbered in the order they are produced by the corresponding speech encoder

The ordering algorithm is described in pseudo code as:

```
- for j = 0 to K-1
- d(j) := s(table_m(j)+1);
```

where $\text{table}_m(j)$ refers to the relevant table below on the AMR mode m=0..7 [5]

Table3.6: Ordering of the speech encoder bits for the 4.75 kbit/s mode: $table_0(j)$ [5]

j=0	j=1	j=2	...	...	...	...	...	...	...
0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	23	24	25	26
27	28	48	49	61	62	82	83	47	46
45	44	81	80	79	78	17	18	20	22
77	76	75	74	29	30	43	42	41	40
38	39	16	19	21	50	51	59	60	63
64	72	73	84	85	93	94	32	33	35
36	53	54	56	57	66	67	69	70	87
88	90	91	34	55	68	89	37	58	71
92	31	52	65	86					

The reordered bits are further divided into three indicative classes according to their subjective importance. They are Class A, Class B, and Class C. These classes are then subjected to different error protection techniques.

Class A: Class A contains the bits most sensitive to errors and any error in these bits typically results in a corrupted speech frame which should not be decoded without applying appropriate error concealment. This class is protected by the Codec CRC in AMR Auxiliary Information.

Class B & Class C: Classes B and C contain bits where increasing error rates gradually reduce the speech quality, but where decoding of an erroneous speech frame is usually possible. Class B bits are more sensitive to errors than Class C bits. Below Table4.7 indicates number of speech bits in each class for each AMR mode.

Table 3.7: No of bits in Class A, B, and C for each AMR Codec Mode [5]

Frame Type	AMR codec mode	Total number of bits	Class A	Class B	Class C
0	4,75	95	42	53	0
1	5,15	103	49	54	0
2	5,90	118	55	63	0
3	6,70	134	58	76	0
4	7,40	148	61	87	0
5	7,95	159	75	84	0
6	10,2	204	65	99	40
7	12,2	244	81	103	60

<u>AMR Core Frame Comfort Noise Bits</u>--These are the bits generated by the encoder for frame types ranging from 8-15.These comfort noise bits are all mapped to Class A of AMR Core Frame and Classes B and C are not used.

The comfort noise parameter bits produced by the AMR speech encoder are denoted as s(i) = {s(1),s(2),...,and s (35)}. These bits are numbered in the order they are produced by the AMR encoder. Comfort Noise bits are followed by the SID Type Indicator STI and the Mode Indication mi(i) ={mi(0), mi(1), and mi (2)} ={LSB .. MLB}. Thus, the AMR SID or comfort noise bits {d(0),d(1),...,and d (38)}are formed as defined by the pseudo code below.

- for j = 0 to 34;
- d(j) := s(j+1);
- d(35) := STI;
- for j = 36 to 38;
- d(j) := smi(j-36). Note: This mapping is different to the usual mapping: LSB first.

Note: The alternative would be: d(j) := mi(38-j): MSB first [5]

3.3.2 AMR Interface Format 2

It is also called as octet-aligned frame format for the AMR codec. The frame is composed of 4-bit Frame Type field (similar to IF1), AMR Core Frame (similar to IF1). The total number of bits in the AMR IF2 speech frames in the different modes is typically not a multiple of eight and bit stuffing is needed to achieve an octet structure.

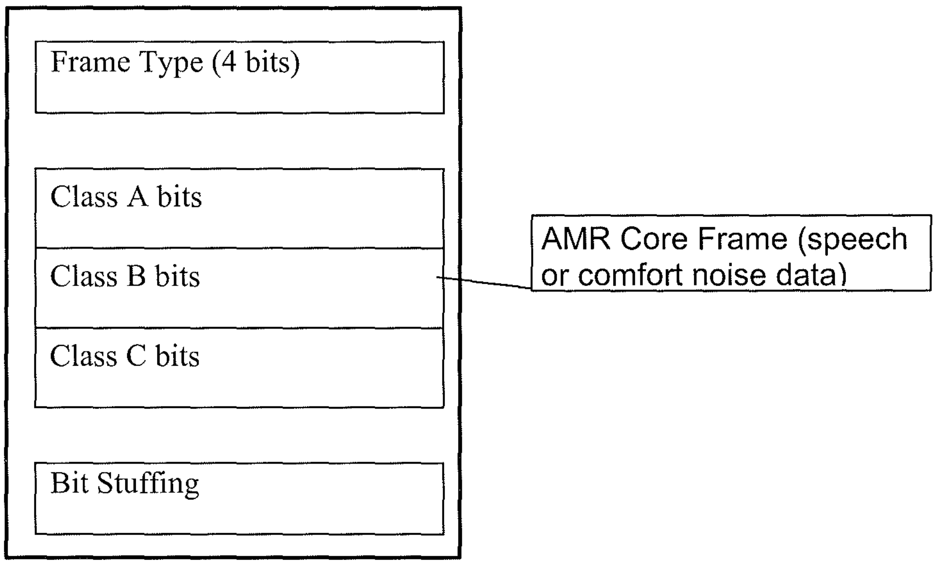


Figure 3.4: AMR Frame Structure for IF2 [12]

3.4 AMR Implementation in Asterisk:

In the above sections we presented the basic knowledge that is required to understand AMR Codec and Asterisk. This section will elaborate on the actual implementation. Codec is encoding and compression algorithm utilized by applications. All the source files for the codecs currently supported by asterisk are placed under /codecs/ directory of the source package.

The implementation is based on the concept that the core encoding method in asterisk is AST_FORMAT_SLINEAR, in which non-compressed audio is sampled at 8000 times a second, with 16-bit signed samples. Asterisk basically translates voice frames from SLINEAR to other compression techniques and vice versa.

codec_amr.c :So we first started writing AMR as a standalone program which will be loaded as a module. The name of this file is codec_amr.c. As per the Asterisk standard the AMR should have the following functions to successfully load it as a module.
int load_module():- In this module asterisk checks if it can load the AMR module successfully.It checks whether it can translate the voice frames from SLINEAR to AMR format and vice-versa. If it is the case then it returns zero, otherwise it returns non-zero.

int unload_module():- This function will unload the AMR module. This function returns zero if it successfully unloads the module otherwise it returns non-zero. Usually module can only be unloaded if the no of channels using the module indicated by the variable usecount is zero.

char *description():- This function returns the codec functionality.

int usecnt():- This function returns the no of channels using this AMR module.

Then various functions like *lintoamr_new, *amrtolin_new, *lintoamr_sample are implemented by using file codec_ilbc as a template.
The implementation of this file is explained in detail in Appendix A.1. The corresponding Makefile in the same directory has to be edited so that asterisk can recognize this codec(The Makefile is described in detail in Appendix A.2).

format_amr.c: Also format_amr.c is created in /formats/ directory to handle .amr extension sound files. The corresponding Make File in the same directory is edited to load this format_amr.c file. The implementation of this file is explained in detail in Appendix A.3.

The corresponding Makefile in the same directory has to be edited so that asterisk can recognize this format.(The Makefile is described in detail in Appendix A.4).

The following files in the asterisk have to be modified for AMR codec implementation to work.

Frame.c:- AST_FORMAT_AMR should be added to list of media format supported.

Ex :
```
/*! AMR codec */
#define AST_FORMAT_AMR       (1 << 11)
```

Rtp.c:- AST_FORMAT_AMR should be added to the list of RTP Payload types supported.

Ex :
```
/*! AMR codec */
#define AST_FORMAT_AMR       (1 << 11)
```

Channel.c :- In this file also AST_FORMAT_AMR should be added to the list of preferred codec formats.

Ex :
```
/*! AMR codec */
#define AST_FORMAT_AMR       (1 << 11)
```

/include/frame.h:- Here also AST_FORMAT_AMR should be added to the Asterisk Frame Definition List.

Ex:
```
  /*! AMR codec */
#define AST_FORMAT_AMR          (1 << 11)
```

3.5 Summary

This chapter described the basic information about Adaptive Multi Rate Speech Codec, differences between the versions of AMR by ETSI/3GPP. We also discussed the frame Structure for the AMR-NB which is the focus of our implementation in Asterisk. Finally in the last Section we discussed how exactly it is implemented in Asterisk. Here In addition to adding new source files in Asterisk existing source files need to be modified so that Asterisk can identify AMR. Although AMR-NB is implemented in asterisk it is not tested for its functionality. The other feature implemented in Asterisk is bridging function between SIP/RTP – ISDN. Next chapter discusses the SIP/RTP protocols.

4 Session Initiation Protocol and Real-Time protocol

4.1 Introduction:

Session Initiation Protocol (SIP) is IETF's standard for multimedia conferencing over IP. SIP is ASCII-based that can be used for forming, maintaining, and terminating sessions that involves multimedia elements such as video, audio etc. It is most popular protocol for VOIP applications today. It works in the Application layer of the (OSI) communications model; the key functions of SIP in packet telephony network are signaling and session management.

SIP was originally intended to create a mechanism for inviting people to large-scale multipoint conferences on the Internet Multicast Backbone (Mbone).Later it was realised that SIP could be used to set up point-to-point conferences. Signalling carries call information across network, where as Session management provides the ability to control the attributes of an end-to-end call[25]. Sip uses suitable protocol to convey the information in the session. The key functions of sip include[26]

i) SIP supports address resolution, name mapping, and call redirection. Thus it can easily locate target end point.

ii) Establishing a session between the originating and target end point.

iii) Determining the availability of the target end point.

iv) Establishing a session between the originating and target end point.

v) Handling the transfer and termination of calls.

SIP makes the communication between the devices possible with the help of two other protocols namely RTP/RTCP and SDP. RTP Protocol is used to transport data in real time, whereas SDP protocol is a format for describing the communication session like session name and purpose, times the session is active, media comprising the session etc.

4.2 SIP Elements:

SIP architecture has two basic components. SIP User Agent and SIP Network Server.

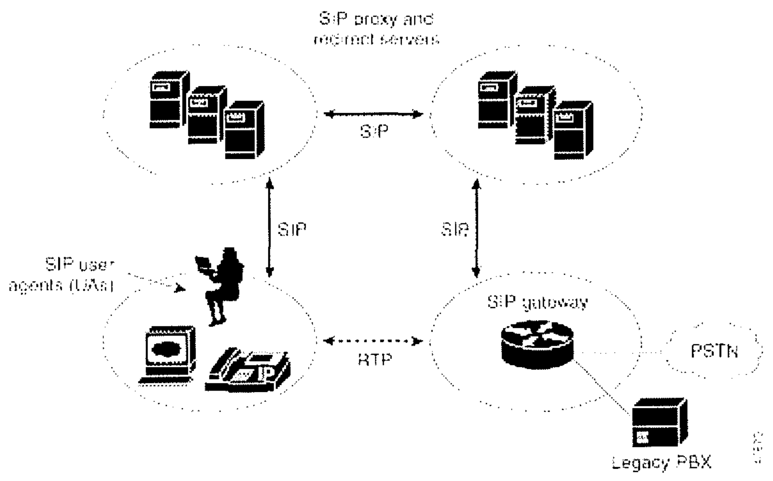


Figure 4.1: SIP Architecture

4.2.1 SIP User Agent (UA)

The user agent is the end system component that initiates the call, which can be represented by a hardware or software device implementing SIP (e.g., an IP phone), and consists of two main components: User Agent Client (UAC) and User Agent Server (UAS) .UAC component is a client application that initiates the SIP request. Where as UAS Server application that contacts the user when a SIP request is received and that returns a response on behalf of the user.

4.2.2 SIP Network Server (NS)

It's a network device that handles the signalling associated with multiple calls.
It consists of SIP Register Server, SIP Proxy Server, and SIP Redirect Server.

SIP Proxy server acts as mediator serving the requests or forwarding them to other UAS's or UACs for servicing. There are two different operating modes for proxy server. Stateless mode and Stateful mode. In stateless mode, server forgets all the information once the request is sent but in stateful mode server save previous routing information and is able to use it to speed up the message transfer).[27]
Proxy servers can also be used for name mapping. For this proxy server questions a location service to map an external SIP identity to an internal SIP identity.

Note: Proxy servers are not Firewalls they are independent servers on the internet that proxy the request on behalf of the user.
SIP Register Server: It receives registration messages from endpoints regarding current user location and maps the SIP addresses with the physical location(s) in the domain where the endpoint is located. These mapping data are stored in a database, which can reside on the same machine or on a remote server.

Users can alter the address at which they are contactable through registrar server. SIP clients can contact the registrar server by utilizing information that is configured into the client or by multicasting the address to contact the registrar server.

SIP Redirect Server helps endpoints to find the desired address by redirecting them to try another server. Thus users can temporarily change geographic location and still be contactable through the same SIP identity through redirection.

4.3 SIP Messages:

SIP messages are exchanged when a call is set up, modified, and terminated. SIP messages are very much like HTTP, the Web protocol, or SMTP. SIP messages are formatted according to RFC822.

4.3.1 SIP Message Format

Sip message can either be a request or response message.. Below is the general format for sip message (request or response)

- A start line
- One or more header fields
- An empty line
- A message body (optional)

Each line must end with a carriage return-line feed (CRLF).

Ex

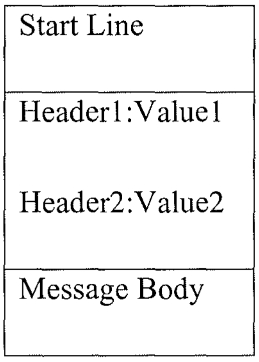


Figure 4.2: SIP Message Format

A sample sip message captured during a sip call via Asterisk is in page 46.

Start line contains the SIP version and the method, address in case of request messages and the status line in response messages.• Headers contain the information related to the call like the initiator, the recipient of request etc in text lines. Below are the some of the main fields in headers:

- Via: shows the transport protocol used and the request route, each proxy adds a line To this field
- From: shows the address of the caller.
- To: show the address called user of the request.
- Call-Id: Unique identifier for each call and contains the host address. It must be the same for all the messages within a transaction.
- Cseq : begins with a random number and it identifies in a sequential each message.
- Contact: shows one (or more) address than can be used to contact the user
- User Agent: The client agent which deals the communication.

Message body (payload) generally carries SDP message. Sometimes it can also carry ISUP (in case of interworking with PSTN)

4.3.2 SIP Request:

The SIP request methods that are commonly used in call set-up and release are OPTIONS, REFER, INVITE, BYE, ACK, REGISTER, CANCEL, NOTIFY, SUBSCRIBE.

- OPTIONS Is used to query a server about its capabilities.
- REFER, to transfer calls and to contact any external resource
- INVITE, to initiate a dialog between two participants
- BYE, to terminate a connection between two users in a session
- ACK, for reliable message exchange following a successful INVITE
- REGISTER, to convey location information to a SIP server
- CANCEL, to cancel an initiated session not yet finalized

- NOTIFY, to provide information about a state change that is not related to a specific session
- SUBSCRIBE, to indicate a desire for NOTIFY requests.

4.3.3 SIP Response:

The response messages sent by the server contain Status Codes and Reason Phrases that indicate the current condition of this request. The status code values are divided into six general categories:

- 1xx: Provisional: The request has been received and processing is continuing.
- 2xx: Success: An ACK, to indicate that the action was successfully received, understood, and accepted.
- 3xx: Redirection: Further action is required to process this request.
- 4xx: Client Error: The request contains bad syntax and cannot be fulfilled at this server.
- 5xx: Server Error: The server failed to fulfil an apparently valid request.
- 6xx: Global Failure: The request cannot be fulfilled at any server.

Below is a sample sip message

Ex:

```
User-Agent: Asterisk PBX
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY
Contact: <sip:1498@141.163.8.211>
Content-Length: 0

SipCall: Incoming response
SipTransaction: Incoming Response
SipCallMember: localStatusUpdated: 100
SipClient: Receiving message...

SipClient: Received: 12:48:47.074
--------------------------------
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP 141.163.8.194;branch=z9hG4bK12C31968;received=141.163.8.194
From: "praveen" <sip:1499@141.163.8.211>;tag=5553C38
To: <sip:1498@141.163.8.211>;tag=as6589950a
```

```
Call-ID: 1339886467@141.163.8.194
CSeq: 4603 INVITE
User-Agent: Asterisk PBX
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY
Contact: <sip:1498@141.163.8.211>
Content-Length: 0
```

4.3.4 Session Description Protocol (SDP)

SIP acts as a carrier for the Session Description Protocol (SDP), which describes the media content for the multimedia session.SDP is described in RFC 2327.SDP was originally designed to announce necessary information for the users and multicast MBONE (Multicast Backbone) applications. A SDP message is made up of lines, called fields, where names are identified by a single letter

The SDP message format includes a number of lines of text called fields, of the form <type> = <value>, where the <type> defines a unique session parameter identified by a single letter, and the <value> provides a specific value for that parameter. These SDP messages can be transported by other protocols, like SAP, RTSP, and electronic mail with MIME applications or HTTP-like protocols

SDP messages include

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

Below are some of the fields of an SDP message

v= (protocol version)

o= (owner/creator and session identifier).

s= (session name)

i=* (session information)

u=* (URI of description)

e=* (email address)

p=* (phone number)

c=* (connection information - not required if included in all media)

b=* (bandwidth information)

z=* (time zone adjustments)

k=* (encryption key)

a=* (zero or more session attribute lines)

Time description

t= (time the session is active)

r=* (zero or more repeat times)

Media description

m= (media name and transport address)

i=* (media title)

c=* (connection information - optional if included at session-level)

b=* (bandwidth information)

k=* (encryption key)

a=* (zero or more media attribute lines)

4.3.5 Signalling & Call Mechanism:

SIP establishes communication between two or more users. Users in a SIP network are identified by unique SIP addresses. A SIP address is similar to an e-mail address and is in the format of sip:userID@gateway.com. The user ID can be either a user name or an E.164 address.[26]

Users register with a registrar server using their assigned SIP addresses. Registration occurs when a client needs to inform a proxy or redirect server of its location. During this process, the client sends a REGISTER request to the proxy or redirect server and includes the address (or addresses) at which it can be reached. Not all REGISTER requests are accepted by the server because it may be the service which has to be paid for. Only the users which are known to the system will be registered after authentication by using username and password. For this four messages has to be exchanged between SIP client and SIP server. The first is a REGISTER request and the answer of the SIP server will be a 401 unauthorized message, which contains WWW-Authenticate field. Then the sip client will send the username and password. The sip server then checks the response and if it is correct responds with a 200 OK response and the user agent is registered at the SIP server. Message sequences for the above authentication process is shown in the below figure 5.3

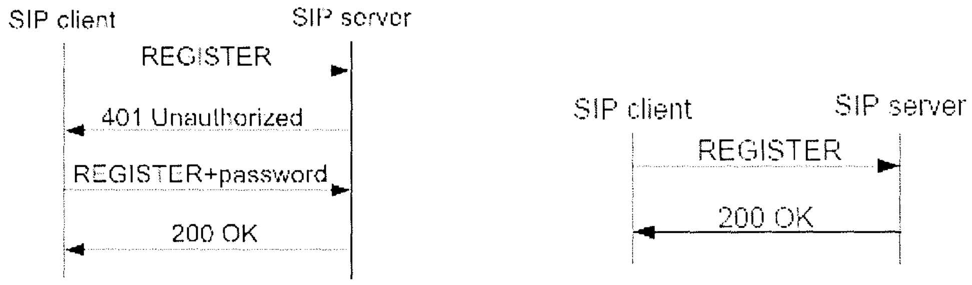


Figure: 4.3 SIP Message sequences for authentication.[27]

A sip client registered at a registrar server is identified with a URI. This sip client can be found by any other sip client with this URI. Below figure5.4 describes the call setup procedure between two sip clients that want to communicate.

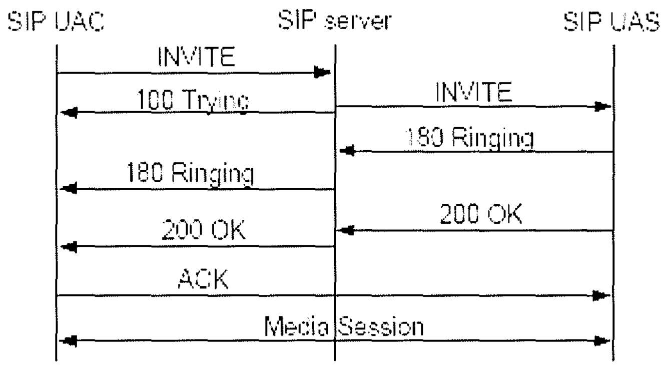


Figure4.4: Call Setup procedure between two SIP clients [27]

A sip client who initiates call is inviter and the client that is being called is invitee. The initial invite message is sent to the invitee via a proxy server as it doesn't know the URI of the invitee. The inviter then receives the OK message from the invite which contains URI of the invitee. Then inviter send ACK message directly to the invitee. Then the two sip clients communicate with each other as they have each others Uri's.

4.4 RTP:

SIP VOIP calls use RTP/RTCP to send real time data such audio, video or simulation data. Even H.323 uses RTP to send media streams. RTP defines a standard format for delivering audio/video content in IP networks. It was developed by the Audio-Video Transport Working Group in IETF and defined in RFC 3550.

RTP actually is a combination of RTP and RTCP. Real-time data transfer protocol (RTP) manages delivery of data that has real-time properties. But it does not guarentee timely delivery or provide quality-of-service. For that it relies on lower-layer services like TCP and UDP. RTCP monitors the quality of service and conveys information about the participants in an on-going session.[28]

In a typical RTP session audio and video data is transmitted using separate channels.

4.4.1 RTP Data Transfer Protocol

Real Time media is exchanged by means of RTP data Transfer Protocol. RTP runs on top of UDP. Generally audio or video chunks of data, generated by the sending side of a multimedia application, are encapsulated in RTP packets. Then the application sending these packets sends them to UDP socket interface. At the receiver end also these packets are received at UDP socket interface.

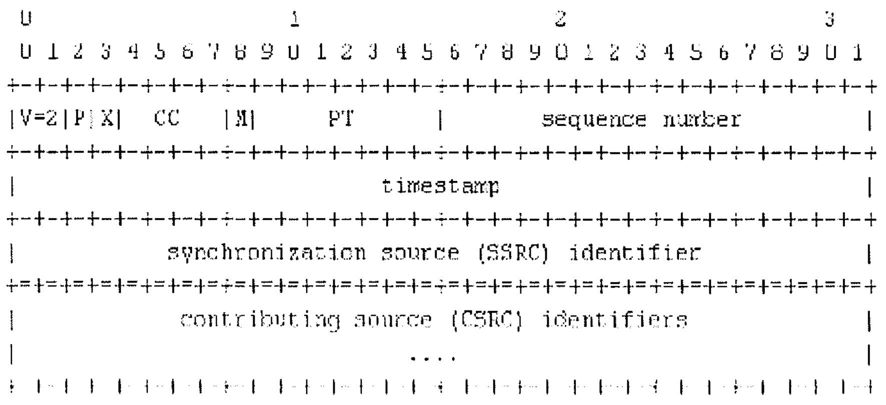
If we consider a scenario where an application wants to transport voice using RTP it precedes each chunk of the audio data with an RTP header, which includes the type of audio encoding, a sequence number and a timestamp. This audio chunk along with the RTP header forms the RTP packet. The RTP packet is then sent into the UDP socket interface, where it is encapsulated in a UDP packet. At the receiver side, the application receives the RTP packet from its socket interface. The application extracts the audio chunk from the RTP packet, and uses the header fields of the RTP packet to properly decode and playback the audio chunk.

Below is the structure of RTP packet.

IP Header	UDP Header	RTP Header	RTP Payload

Figure4.5: RTP Packet Structure [28]

RTP payload is the real time media that is being transferred. RTP payload contains the information related to payload.
Below is the header structure of RTP.



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|X|  CC   |M|     PT      |       sequence number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           synchronization source (SSRC) identifier            |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|            contributing source (CSRC) identifiers             |
|                             ....                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4.6: RTP Header Structure[28]

- Version(V):2bits

  This field identifies the version of RTP. This is always set to 2.
- Padding(P):1bit

  If set, this packet contains one or more additional padding bytes at the end which are not part of the payload. The last byte of the padding contains a count of how many padding bytes should be ignored. Padding may be needed by some encryption algorithms with fixed block sizes or for carrying several RTP packets in a lower-layer protocol data unit.
- extension(X):1bit

  If the extension bit is set, the fixed header is followed by exactly one header extension.
- CSRCcount(CC):4bits

  The CSRC count contains the number of CSRC identifiers that follow the fixed header.
- Marker(M):1bit

  Marker bit is used by specific applications to serve a purpose of its own. We will discuss this in more detail when we study Application Level Framing.
- PayloadType(PT):7bits

  This field identifies the format (e.g. encoding) of the RTP payload and determines its interpretation by the application. This field is not intended for multiplexing separate media.

- SequenceNumber:16bits
  The sequence number increments by one for each RTP data packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence. The initial value of the sequence number is random (unpredictable).
- TimeStamp:32bits
  The timestamp reflects the sampling instant of the first octet in the RTP data packet. The sampling instant must be derived from a clock that increments monotonically and linearly in time to allow synchronization and jitter calculations.
- SSRC:32bits
  The SSRC field identifies the synchronization source. This identifier is chosen randomly, with the intent that no two synchronization sources within the same RTP session will have the same SSRC identifier. The receiver may be receiving data from several sources. So for proper arrangement it needs to identify the source of individual packets which is possible from the SSRC field.

- CSRC
  The CSRC list identifies the contributing sources for the payload contained in this packet. The number of identifiers is given by the CC field. If there are more than 15 contributing sources, only 15 may be identified. CSRC identifiers are inserted by mixers, using the SSRC identifiers of contributing sources.

The synchronization of RTP packets at the receiver end is done with the help of synchronization source, sequence number, and sampling instant of packets which it gets from the RTP header. Receiver may receiver data packets from several sources so it needs SSRC to identify the source of individual packet. Receiving application determine the order of the packet using the sequence number field. Thus it detects packet loss and restores packet sequence. Time stamp only is not enough synchronizing the packets as RTP packets may have equal timestamps if they are generated at once.

4.4.2 RTP Control Protocol

RTCP doesn't transport any data. It periodically carries control packets(RTCP Packets) to all the participants in the session. RTCP packet is a combination of header and data .Below figure is the basic structure of a RTCP packet

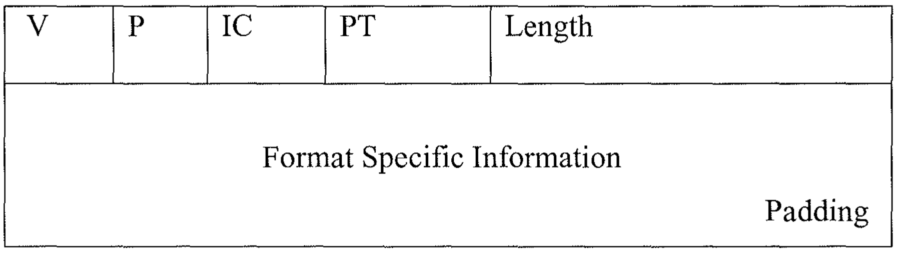


Figure4.7: RTCP Packet Format[29]

All of the above packet types have the same header format which has five fields.

Version number (V) 2 bits : This field identifies the version of RTP which is currently 2.

Padding(P) 1 bit : If the padding bit is set, the packet contains one or more additional padding octets at the end which are not part of the payload. The last octet of the padding is a count of how many padding octets should be ignored. Padding may be needed by some encryption algorithms with fixed block size or for carrying several RTP packets in a lower-layer protocol data unit.

Item Count (IC) : This field is used by the packet types to indicate the number of items included in the packet. Maximum 31 items may be included in each RTCP packet. If an application wants to transmit more items it has to send multiple packets. This field may be used for other purposes if packet types don't need an item count.

Packet type (PT) 8 bits : It indicates the type of information carried in the packet . Five standard packet types are defined in the RTP

Length (16 bits) : The length field denotes the length of the packet contents following the common header.

RTCP header is followed by packet data and optional padding. There are several type of RTCP packets. Some of them are Sender report packet which is for for transmission and reception statistics from participants that are active senders. Receiver report packet, for reception statistics from participants that are not active senders. Source Description RTCP Packet like CNAME. Goodbye RTCP Packet which indicates end of participation.Application Specific RTCP packets indicating various application specific functions.

These RTCP packets are always grouped together for transmission forming compound packets. Each compound packet is then encapsulated in a UDP/IP packet for transport.

4.5 SIP SoftPhones:

SIP establishes communication between two or more users. Users in a SIP network are identified by unique SIP addresses. We have configured and tested some of the softphones with Asterisk as part of this project. Asterisk allows SIP softphones to communicate with each other using internet. The calls made through Asterisk using SIP softphones pass through SIP channel in Asterisk. SIP soft phones need a username and password which will allow them to call other softphones which are already registered with Asterisk. Below is the procedure for configuring a SIP softphone so that it can call other softphones. This procedure contains modifying sip.conf, extensions.conf files in Asterisk as well as adjusting softphone. (Kphone, a Linux based softphone will be quoted as an example)

i) edit sip.conf file in /etc/asterisk directory as below for users (softphones) to register

```
 [praveen]
type=friend
secret=uop
username=praveen
host=dynamic
defaultip=141.163.8.211
context = trusted
```

[praveen] → this means that user ‘praveen’ will be registered.

type=friend→ this user can place or receive calls. We use ‘peer’ for type for INBOUND calls only. For outbound calls only use ‘user’ as type.

secret=uop→ This creates ‘uop’ as password for the user to login/authenticate on Asterisk.

host=dynamic-→ It sets dynamic IP for the host. We can also define static IP for this user.

context=trusted →This defines the dial context for the. In Asterisk, outgoing numbers are divided in groups called contexts in order to separate/define different needs for different user types.

```
 [zeeshan]
type=friend
secret=uop
username=zeeshan
host=dynamic
defaultip=141.163.8.216
context = trusted
```

The above statements register user ‘zeeshan’ with asterisk.

ii) Then we need to edit the extensions.conf file in /etc/asterisk/ directory. The extensions.conf file decides how Asterisk handles incoming calls. It should be modified as follows

```
                [trusted]

        exten => 1234,1,Dial(SIP/praveen)
```

For a call between two Kphones we need to do the following steps in sequence.

i) Start the Asterisk.

ii) Start Kphone.

iii) To configure audio/ video codecs in the main window click preferences >

To configure it to register with Asterisk in main window click File > Identity ... in the menu bar fill the full name field as praveen, user part of url as praveen, in the host part of url give the ipaddress on which the Asterisk you want to connect to is running. This can be found out by 'ifconfig' command on the Linux console. Leave the remaining fields blank and press register button at the bottom. If the softphone is registered successfully with asterisk then it will display the window saying registration successful otherwise it display a window asking to try again. Another SIP softphone with the name zeeshan is registered with the same Asterisk following the same procedure from different machine. For convenience the softphone registered with name Praveen is called as Praveen and with name zeeshan is called as zeeshan below.

If the registration of the SIP softphones (named as praveen and zeeshan above) is successful, we can call from one SIP softphone to another (here Praveen to zeeshan and vice-versa). In this example for example to make a call from Praveen to zeeshan type zeeshan in the main window of praveen sip phone and press audio call button next to the window. If the zeeshan sip softphone accepts the call then Asterisk bridges the call and the two softphones can communicate. (The same procedure is repeated if zeeshan SIP phone wants to call Praveen).For Kphone we have found that the quality of the audio signal is good only when GSM is chosen as the audio codec in both the softphones. If both the softphones are using different codec's we found that the quality of the audio during call is bad.

Below is the list of few softphones with their features that work with asterisk.

Table 4.1: SIP softphones

Soft Phone Name	**Features**	**Codec's Supported**	**Comments**
Eyebeam	Windows Audio/Video	**Audio**-G722, G711 (A-lawful-law, G723, G726, ILBC, Speex. **Video**-H.263,H.264	
Kphone	Linux Audio/Video(with external application)	**Audio** – **Video-**	The quality of the Speech is good when two sip clients are configured to use GSM as audio

		Also support ALSA,OSS for backend	codec.
Linphone	Linux, FreeBSD, Open BSD Audio/Video	**Audio :** Speed(NB & WB), G711(Ulaw,Ulaw),GSM,ILBC **Video :** H.263, Theora	
Twinkle	Linux Audio Only	G711(Alaw, U law), Speex(NB & LB),GSM, iLBC	

4.6 Summary

In this chapter we discussed the SIP protocol functionalities, SIP components(UA and SIP Network Sever) . Later the general format of the SIP message (request/response) is dealt with example. Later how SIP components exchange signalling information for establishing a session is discussed. SIP transfers only signalling information whereas SDP along with RTP transports the actual media to the destination. RTP itself has two sub-components RTDP and RTCP. RDTP transmits RTP Packets which contain the real time media. The format of the RTP packet is discussed in this chapter. RTCP periodically transmits the RTCP 1 packets to the other participants in the session. The structure of RTCP is also discussed in section 5.5.2. Later some of the SIP IP phones tested with Asterisk are explored. Next chapter discusses the audio/video quality measurement which is required in this project for the evaluation of quality of audio/video data in asterisk captured from 3G Mobile.

5 Audio/Video Quality Measurement:

5.1 Introduction

Quality is the characteristic of audio/video passed through a processing system.In the ages of analog systems it was possible to evaluate the quality of processing system by calculating the response of test signal.But now Digital systems are replacing all existing analogy systems.
Audio/Video/Image processing deals with signals meant for human consumption. Audio/Video signal needs to go through different processing stages before being presented to a human observer and each stage of processing may introduce distortions that can reduce the quality of the final output signal. Compression techniques being employed these days reduce the bandwidth for storage or transmission but some times they may distort the signal.

Assessing the quality of signal streamed over IP networks is complicated task. Network level impairments such as packet losses or delay and the availability of resources such as bandwidth are the factors that determine the quality of the signal at the receiving end. Standard metrics that are used to evalute the quality of a signal (audio/video) are categorized as [30]

i) Objective Evaluation Metrics
ii) Subjective Evaluation Metrics

The main goal of many objective quality metric is to automatically estimate user's opinion on a signal processed by the system. Objective quality measurement (as opposed to subjective evaluation by human observers) seeks to determine the quality of signal algorithmically.[32]

It is concerned with how signal is perceived by a user and designates his or her opinion on a particular signal.It is the technique used to find out user's opinion[35].The methods used for the subjective evaluation of signal and audio have been standardized and recommmended by International Telecommunication Union (ITU).The results obtained by using these methods are generally expressed in terms of Mean Opinion Score (MOS).

This chapter discusses the PESQ and PEVQ techniques which are standard objective techniques to evaluate the quality of audio and video signal respectively.

5.2 Speech Quality Measurement using PESQ:

Speech and Audio Quality Measurements have been standardized by ITU-T as PESQ and PEAQ. PESQ predicts subjective quality with good correlation in a very wide range of conditions that may include coding distortions, errors, noise, filtering, delay and variable delay. Early models for audio quality perception didn't include the method for identifying delay which is the major drawback as the current packet transmission based communication may introduce delay. Earlier models like PSQM and MNB is based on concept of quality prediction by comparing perceptual representations of reference and degraded signal. PESQ is also based on the same concept. Below Figure6.1 gives the basic overview of the concept

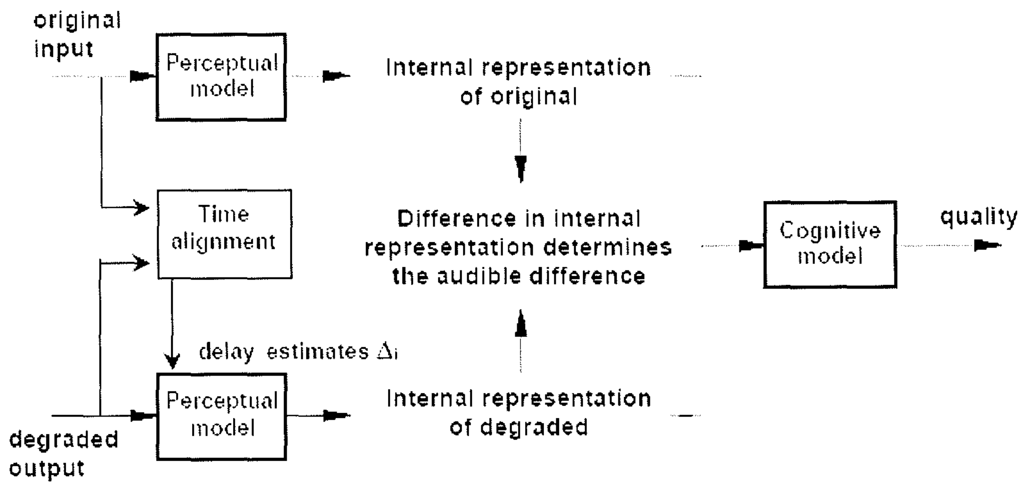

original input
Perceptual model
Internal representation of original
Time alignment
Difference in internal representation determines the audible difference
Cognitive model
quality
delay estimates Δ_i
Perceptual model
Internal representation of degraded
degraded output

Figure 5.1: Block diagram of the basic model of the PESQ algorithm

Original and degraded signals are fed onto an internal representation using a Perceptual Model. A cognitive model uses difference in this representation is used by cognitive model to predict the perceived speech quality of the degraded signal. This perceived listening quality is expressed in terms of Mean Opinion Score, an average quality score. PESQ uses this opinion scale called as ACR (Absolute Category Rating) scale shown in the below table.

Quality of the speech	**Score**
Excellent	5
Good	4
Fair	3
Poor	2
Bad	1

Table 5.1: ACR listening quality opinion scale [37]

The quality of the systems carrying speech in the presence of background or environmental noise can be evaluated by using PESQ. For this the original signal that is passed into PESQ should be clean but the noise should be added before the signals are passed into the system under test and then fed to PESQ system.

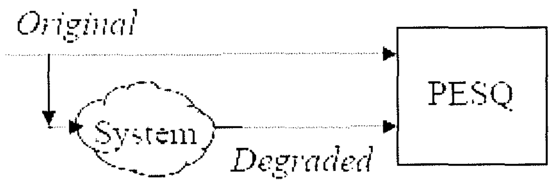


Figure 5.2: PESQ testing using clean Speech[37]

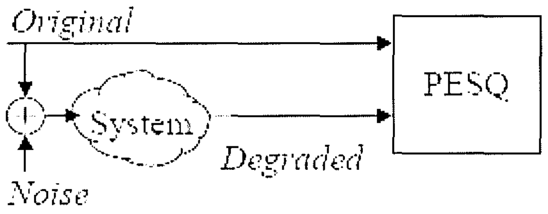


Figure5.3: PESQ testing using noisy speech [37]

Algorithm Structure- Below is a brief overview of Psychoacoustic model of PESQ algorithm

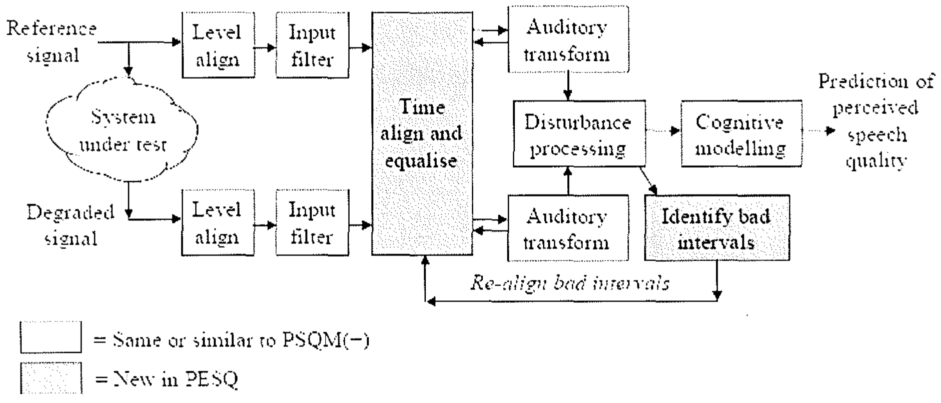


Figure5.4: Structure of PESQ Algorithm from OPTICOM [37]

i) PESQ first compensates for overall gain of the system under test. This is done along with scaling of the signals to a correct overall level. The PESQ level alignment is carried out based on the power of bandpass filtered versions (300 - 3000 Hz) of the original and degraded signals. The original and degraded signals are scaled to same power level.

ii) It is assumed that listening is carried out using a handset with a Frequency response that follows an IRS receive. PESQ computes IRS after receiving filtered versions of the original speech signal and degraded speech signals. In PESQ this is implemented by an FFT over the length of the file, filtering in the frequency domain with a piecewise linear response. The result is the filtered versions XIRSS(t) and YIRSS(t) of the Scaled input and output signals XS(t) and YS(t). [37]

iii) Active Speech Time interval is defined as the interval between start and end of the speech. The presence of the large silent intervals in the original and degraded signals could effect the computation of average distortion values. So estimation is made of the silent parts at the beginning and ending of the intervals.

iv) Time-Frequency Transformation is done by a short term FFT. Then the start points of the frames in the degraded signal are shifted over the delay Observed by the variable delay estimator. The time axis of the original speech signal is untouched.

v) Pitch power density signals PPXWIRSS(f)n and PPYWIRSS(f)n are Obtained by summing up the powers of FFT bands and then normalizing. the power spectrum of the original and degraded pitch power densities are averaged over time. The original pitch power density PPXWIRSS(f)n of each frame n is then multiplied with this partial compensation factor to equalise the original to the degraded signal. This results in a filtered version of the original pitch power density PPX'WIRSS(f)n.[37]

vi) Short-term gain variations are partially compensated by processing the pitch power densities frame by frame. the original and the degraded pitch power densities, the sum in each frame n of all values that exceed the absolute hearing threshold is computed. The distorted pitch power density in each frame, n, is then multiplied by this ratio, resulting in the partially gain compensated distorted pitch power density PPY'WIRSS(f)n.

viii) The original and degraded pitch power densities are transformed to a Sone loudness scale using Zwicker's law.

ix) Then the distributed density is calculated. First signed difference between the distorted and original loudness density is computed. If this difference is Positive, components such as noise have been added. If this difference is negative, components have been omitted from the original signal. This difference array is called the raw disturbance density.

x) Normally if codec distorts the input signal it will be difficult to introduce a new time-frequency component that integrates with the input signal, and the resulting output signal can be decomposed into two different precepts, the input signal and the distortion, leading to clearly audible . When the codec leaves out a time-frequency component the resulting output signal cannot be decomposed in the same way and the distortion is less . This effect is modelled in PESQ by calculating an asymmetrical disturbance density DA(f)n per frame by multiplication of the disturbance density D(f)n with an asymmetry factor. This asymmetry factor equals the ratio of the distorted and original pitch power densities raised to the power of 1.2.

xi) In this step bad intervals are realigned. Bad intervals are the consecutive frames with a frame disturbance above the threshold. bad intervals a new delay value is estimated by locating the maximum of the cross correlation between the absolute original signal and absolute degraded signal precompensated with the delays observed by the pre-processing. When the maximal cross correlation is below a threshold, it is concluded that the interval is matching noise against noise and the interval is no longer called bad, and the processing for that interval is halted. Otherwise, the frame disturbance for the frames during the bad intervals is recomputed and, if it is smaller, replaces the original frame disturbance. The result is the final frame disturbances D''n and DA''n that are used to calculate the perceived overall speech quality.

xi) Next all disturbances are aggregated over time.

xii) At last the final PESQ MOS score is computed. Final PESQ MOS score is a linear combination of the average disturbance value and the average asymmetrical disturbance value. Normal output will be a MOS-like score between 1.0 and 4.5 but in some cases the PESQ score is between -0.5 and 4.5.

5.3 Video Quality Measurement using PEVQ:

Video signal when passed through network is influenced by many factors. Some of them are bit rate(data needed to represent it), the complexity of the encoding and decoding algorithms, robustness to data losses and errors, ease of editing, end-end delay , the compression algorithms used and the number of other factors.The Subjective Quality Measurement for video is done through the following sequence of steps[36].

- Choose video sequences for testing (they are often named SRC).
- Choose settings of system that you want to evaluate (often named HRC).
- Choose test method (how sequences are presented to experts and how their opinion is collected).
- Invite sufficient number of experts (not less than 15 are recommended).
- Carry out testing.
- Calculate average marks for each HRC based on experts' opinion.

There are many ways in which you can show video sequences to expert and to record his or her opinion, and a few of them have been standardized. They are thoroughly described in ITU-T recommendation BT.500. One of standardized methods is DSIS - Double Stimulus Impairment Scale: expert is presented with unimpaired reference video, than with the same video impaired, and after that he is asked to vote on the second video using impairment scale (scale from "impairments are imperceptible" to "impairments are very annoying")[32]

The Objective and Subjective quality metrics for video are furthur sub-classified. Objective metrics are sub- divided as Full Reference Methods, Reduced Reference Methods and No-Reference Methods. No-Reference algorithm has access only to the distorted signal and must estimate the quality of the signal without any knowledge of the 'perfect version' of the signal. Reduced-Reference Algorithm has only partial information regarding the 'perfect version'. A side-channel (called an RR channel) exists through which some information regarding the reference can be made available to the algorithm. RR algorithms use this partial reference information to judge the quality of the distorted signal. Full Reference algorithm has access to a 'perfect version' of the image or video against which it can compare a 'distorted version'. The 'perfect version' generally comes from a high-quality acquisition device, before it is distorted.

PEVQ is one such full reference intrusive algorithm for video quality. We got an evaluation version of PEVQ from OPTICOM and used it to test the video quality of some sample clips. Below is the block diagram of PEVQ from OPTICOM.

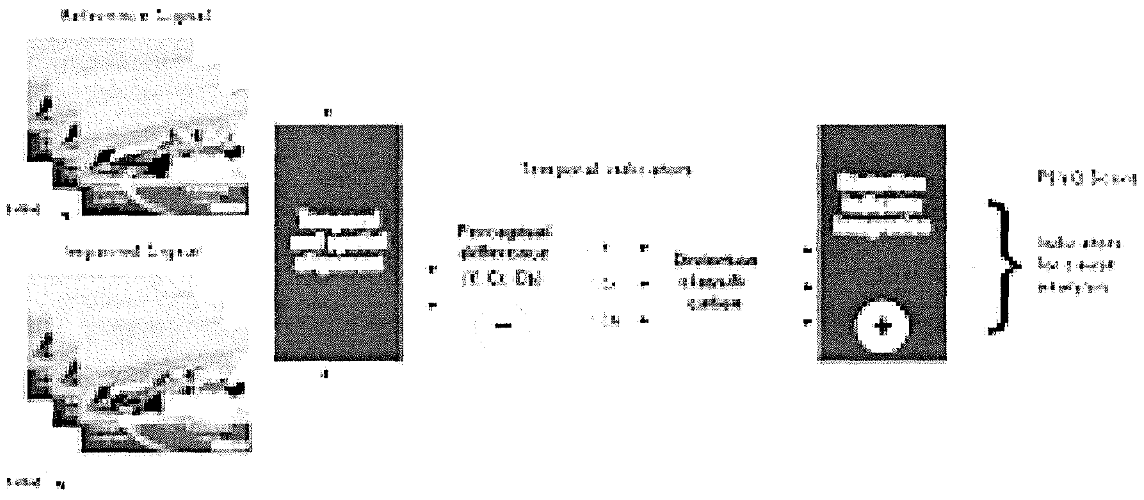


Figure 5.5: PEVQ block Diagram. [36]

<u>Features</u>: PEVQ takes two input signals one is the source sequence and another one is the distortion version of the source. Both these inputs should be AVI type files containing RGB24 data. Both of these should also have constant frame rate. Currently PEVQ from OPTICOM supports frame rates of 25 to 30 frames per second. PEVQ works only with

video streams of CIF, VGA, and QCIF format and also it gives the output score less than 5.0.

Mechanism:

- In the pre-processing stage the spatial and the temporal frames alignment is done and only those frames are compared.
- The second stage calculates the difference of the two aligned signals.
- In the third stage the classification of the signals are done and the distortions are detected.
- In the final stage depending on the detected distortions the final MOS value is generated.

PEVQ measures below parameters in addition to measuring the overall quality of the video.

Delay, Frame Freeze, Frame Skip, Effective Frame Rate, Jerkiness, Blur, Blockiness, Temporal Activity, Temporal Distortions, Brightness, Contrast, PSNR, Distortion Indicators.

The PEVQ demo program from OPTICOM contains source code, libraries and make file. This demo program can be generated by executing 'run' command on the konsole in the main directory of the package. The program consists of PevqOEMMain.cpp file which contains the main function, helper library for command line in IOLib folder of the main directory and PEVQ OEM library in the PEVQLib sub-directory. OEM library in linux system has two files PevqInterface.h and LibPevqOem.a. PevqInterface.h is standard c++ header file which defines the API. LibPevqOem.a contains PEVQ library which must be linked to our own code.

The software package is installed as below:

1 Copy the contents of the OPTICOM PEVQ OEM zipped tar-ball to PEVQ OEM folder on the hard drive.

2 Change into PEVQ OEM directory on the Linux konsole and run the following command.

```
./install_sh –i
```

3 Running the above script automatically copies licence file into /usr/bin folder of linux. If it is not copied then manually copy the file into the folder.

The PEVQ from OPTICOM has couple of sample clips for testing located in /PEVQOem-win0.9.1-Demo/ directory. The demo program on sample .AVI clips by typing the following command on the Linux konsole. The command passes the reference signal tile as well as degraded signal file to the program as parameters to the program.

```
PevqOEMMain –Ref \PevqRef.avi –Test \PevqTest.avi
```

5.4 Summary

This chapter discussed speech quality measurement using PESQ and video quality measurement using PEVQ. We intend to use PESQ and PEVQ to measure the quality of the audio/video streams captured from 3G mobile into Asterisk. In this chapter initially standard audio/video quality metrics like objective evaluation , subjective evaluation are explained. Working of the PESQ algorithm which is standard objective speech quality measurement technique is then explained. In the next section the PEVQ, an objective video quality measurement technique is explained. The installation of the OPTICOM demo PEVQ software, the executing the demo program on some sample clips is then explained.

`6 Discussions, Conclusions & Future Work

The video call between 3G and SIP softphone is commercially an innovative idea for many reasons. This idea can be made a reality by making Asterisk PBX a gateway between 3G and IP network. Asterisk as a gateway directs the media (audio + video + control) from 3G Mobile to soft phone and vice-versa. Asterisk needs to have the necessary features act as a path for the media. This project aims to implement such features in asterisk. We implemented only couple of features. For example we succeeded in implementing AMR codec in asterisk but we couldn't test it for its functionality. We then studied the internals of sip protocol, tested SIP softphones with Asterisk but couldn't pinpoint my research to implement the bridging functions. This chapter section 7.1 presents a summary of conclusions and section 7.2 presents the recommendations for future research.

6.1 Conclusions

Earlier in this thesis we presented the basic idea of this project and it is to make video call from 3G Mobile to SIP Client via Asterisk. Asterisk acts as a path for a call between two sip clients. But to make a video call between a Motorola 3G Mobile and SIP Client via Asterisk we need to implement new features in asterisk such that it acts as a gateway between them.
Section 1.5 discusses the features that need to be implemented in asterisk to make it a gateway. Motorola 3G mobiles use AMR as audio codec and MPEG-4 as video codec. The problem is Asterisk currently doesn't support AMR as well as MPEG-4 codec's and we need to implement them to make it as Gateway. It doesn't even have H324M support.

The project started with a literature review on Asterisk PBX. Asterisk PBX is downloaded and is configured to handle incoming and outgoing calls where we tested various SIP soft phones with Asterisk. In chapter 3 we presented the basic knowledge of

Asterisk. Its file organization and its basic components like channels, protocols and codec's. The basic call handling mechanism in Asterisk is explored.

We also discussed that asterisk currently handle various audio compression algorithms but not AMR. In chapter 4 we presented the basic information about AMR Codec. Here we got familiar with frame structure of AMR-NB. Later the implementation details of this codec in asterisk are provided. The configuration of existing codec is used as a template in AMR implementation.

In chapter 5 we tried to understand how signalling and media transportation is done by studying the SIP/RTP protocols. As already discussed SIP is one of the VOIP protocols supported by asterisk. It transports only signaling messages. It uses RTP to transport the media. SIP is configured as a channel driver that brings a call into Asterisk. A call made from sip client arrives on channel driver interface. This driver executes the dial plan and here if asterisk needs to call the destination client it simply creates a outbound PBX thread and makes a outbound call. If the destination answers the call it bridges the call between calling and called party. If both sip clients agree on both channels and codec's channel driver simply forwards the media stream without passing it through asterisk. Otherwise Asterisk stays in the middle taking care of transcoding and other issues.

Other functionalities that are implemented in Asterisk are H.324M protocol stack which can record audio and video streams from mobile and MPEG-4 codec (These implementations are not discussed in this report. See Zeeshan's report). H.324M stack can record a call (video + audio) made from 3G Mobile into Asterisk. The quality of the video and audio recorded into Asterisk needs to be measured. Chapter 6 presents the standard techniques for the measurement of audio and video quality like PESQ and PEVQ respectively. An evaluation PEVQ copy from OPTICOM is used to evaluate the quality of video of some sample clips.

6.2 Recommendations for Future Research:

This section will discuss the possibilities for future improvements. Currently the new features that are developed are centred on Asterisk. Firstly the current implementation of the AMR Codec in asterisk should be thoroughly tested for its functionality for bugs. Also it will be interesting to explore the possibility of developing a sip client with AMR codec to implement a full 3G mobile to a SIP client call.

As far as Asterisk is concerned it doesn't yet have all the features of a traditional PBX. More research can be done to implement other features into it which are not yet implemented.

The implementation of H.324M protocol stack in asterisk as part of this project can only handle a call from 3G mobile. The code can still be optimized to handle calls from multiple mobiles at the same time. Also the quality of the video and audio clips recorded by the asterisk needs to be evaluated.

Bibliography

[1] Avi Fisher White paper on 'Video-GateWay Using 3G-324M-over –IP'

[2] Jim Van Meggelen, Jared Smith, and Leif Madsen, 2003 *Asterisk:The Future of Telephony.*

[3] Paul Mahler 2003 *VoIP Telephony with Asterisk.*

[4] Mark Spencer, Mack Allison and Cristopher Rhodes. 2003
' *The Asterisk Handbook Version 2* '

[5] "Performance Characterization of the AMR Speech Codec" 3GPP TR 26.975

[6] "Mandatory Speech Codec speech processing functions AMR Speech Codec" 3GPP TS 26.071

[7] "AMR Wideband Speech Codec; General Description,", 3GPP TS 26.171.

[8] "Wideband Coding of Speech at Around 16 kbit/s Using Adaptive Multi-Rate Wideband (AMR-WB),", Geneva, ITU-T Recommend. G.722.2, 2002.

[9] *AMR Wideband Speech Codec; Source Controlled Rate Operation*, 3GPP TS 26.193.

[10] *Adaptive Multi-Rate Wideband Speech Transcoding*, 3GPP TS 26.190.

[11] *AMR Wideband Speech Codec; ANSI-C Code*, 3GPP TS 26.173.

[12] *AMR Wideband Speech Codec; Test Sequences*, 3GPP TS 26.174.

[13] *AMRWideband Speech Codec; Voice Activity Detector (VAD)*, 3GPP TS 26.194.

[14] *AMR Wideband Speech Codec; Comfort Noise Aspects*, 3GPP TS 26.192.

[15] *AMR Wideband Speech Codec; Error Concealment of Lost Frames*, 3GPP TS 26.191.

[16] *AMR Wideband Speech Codec; Frame Structure*, 3GPP TS 26.201.

[17] *AMR-WB Speech Codec Performance Characterization*, 3GPP TR 26.976.

[18] *ANSI C-Code for the Floating-Point Adaptive Multi-Rate (AMR) Wideband Speech Codec*, 3GPP TS 26.204.

[19] *AMR-WB White Paper v07-07-2003*, "Wideband speech coding standards and applications", VoiceAge Corp., http://www.voiceage.com, (Acc. 06 08 24)

[20] Stefan Bruhn, "Bridging the gap between speech and audio coding – AMR-WB+ - the codec for mobile audio", *Ericsson Research, Multimedia Technologies* http://www.s3.kth.se/radio/COURSES/S3_SEMINAR_2E1380_2004 /presentations/EricssonAudio-040506.pdf, (Acc. 20 08 06).

[21] *Packet-Switched Streaming Services (PSS); Protocols and Codecs*, 3GPP TS 26.234.

[22] *Multimedia Messaging Service (MMS); Media Formats and Codecs*, 3GPP TS 26.140.

[23] B. Besette, R. Salami, R. Lefebvre, M. Jelinek, J. Rotola-Pukkila, J. Vainio, H. Mikkola, and K.Jarvinen, "The Adaptive Multirate Wideband Speech Codec (AMR-WB)," *IEEE Trans. on Speech and Audio Processing*, vol. 10, no. 8, 2002, pp. 620 –636.

[24] K. Järvinen, "Standardisation of the Adaptive Multi-rate Codec," *European Signal Processing Conference* (*EUSIPCO*), Tampere, Finland, 4–8 Sept. 2000.

[25] M. Handley and V. Jacobson, Rfc 2327: Session description protocol, April 1998, http://www.ietf.org/rfc/rfc2327.txt.

[26] M. Handley, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, E. Schooler, and J. Rosenberg, Rfc 3261: Session initiation protocol, June 2002, http://www.ietf.org/rfc/rfc3261.txt.

[27] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, Rfc 2543: Session initiation protocol, March 1999, http://www.ietf.org/rfc/rfc2543.txt.

[28] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, Rfc 3550: Real-time transport protocol, July 2003, http://www.ietf.org/rfc/rfc3550.txt.

[29] Colin Perkins *'RTP Audio and Video for the Internet'* June 2003

[30] P. Corriveau, *et al.*, "Video quality experts group: Current results and future directions," *Proc. SPIE Visual Comm. And Image Processing*, vol. 4067, June 2000

[31] Wang and L. Lu, "Objective image and video quality assessment," *Technical Report: IBM T. J. Watson Research Center*, Aug. 2001.

[32] A.B. Watson, J. Hu, and J. F. III. McGowan, "Digital video quality metric based on human vision," *Journal of Electronic Imaging*, vol. 10, no. 1, pp. 20–29, 2001.

[33] ANSI T1.801.03 – 1996, "American National Standard for Telecommunications - Digital Transport of One-Way Video Signals – Parameters for Objective Performance Assessment,"American National Standards Institute

[34] ITU-T Recommendation P.910, "Subjective video quality assessment methods for multimedia applications,"Recommendations of the ITU, Telecommunications Standardization Sector.

[35] ITU-R Recommendation BS. 1535-1, "Method for the Subjective Assessment of Intermediate Sound Quality (MUSHRA)", *International Telecommunications Union*, Geneva, Switzerland, 2001.

[36] PEVQOPTICOM whitepaper 'http://www.opticom.de/download/SpecSheet_PEVQ_06-02-03.pdf' accessed on 24-08-06

[37] OPTICOM white paper on "State-of-the-Art Voice Quality Testing", 2000 'http://www.opticom.de/download/STATEO1.PDF' accessed on 24-08-06

APPENDIX

A . Code

The codec_amr. File and make file should be placed under the /codecs/ directory in the source code of asterisk.

A.1 Codec_amr.c

```
/*
 * Asterisk -- An open source telephony toolkit.
 *
 * ( This is for AMR codec, added by lfsun, )
 *
 * Copyright (C) 1999 - 2005, Digium, Inc.
 *
 * Mark Spencer <markster@digium.com>
 *
 * See http://www.asterisk.org for more information about
 * the Asterisk project. Please do not directly contact
 * any of the maintainers of this project for assistance;
 * the project provides a web site, mailing lists and IRC
 * channels for your use.
 *
 * This program is free software, distributed under the terms of
 * the GNU General Public License Version 2. See the LICENSE file
 * at the top of the source tree.
 */

/*! \file
 *
 * \brief Translate between signed linear and AMR codec
 *
 * \ingroup codecs
 */

// codec_amr.c is modified from codec_ilbc.c, by lfsun, 21/6/06
//

//#define TYPE_MASK        0x3


#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <netinet/in.h>
#include <string.h>
#include <stdio.h>
```

```
#include "asterisk.h"

ASTERISK_FILE_VERSION(__FILE__, "$Revision: 7221 $")

#include "asterisk/lock.h"
#include "asterisk/translate.h"
#include "asterisk/module.h"
#include "asterisk/logger.h"
#include "asterisk/channel.h"

//#include "amr_float/sp_enc.h"
//#include "amr_float/sp_dec.h"
#include "amr_float/interf_enc.h"
//#include "amr_float/interf_dec.h"
//#include "amr_float/rom_enc.h"
//#include "amr_float/rom_dec.h"
#include "amr_float/typedef.h"

/* Sample frame data */
#include "slin_amr_ex.h"
#include "amr_slin_ex.h"

//static int rxframetypeMode = 0;           /* use RX frame type codes
*/
//static enum Mode mode = (enum Mode) MR122;    //default for MR122

AST_MUTEX_DEFINE_STATIC(localuser_lock);
static int localusecnt=0;

static int dtx = 0; //added by lfsun, enable encoder DTX?
enum Mode mode = MR122;

static char *tdesc = "AMR /PCM16 (signed linear) Codec Translator, for
testing testing";

struct ast_translator_pvt {
//      enc_interface_State *enc;
//      dec_interface_State *dec;
        struct ast_frame f;
          /* Space to build offset */
          char offset[AST_FRIENDLY_OFFSET];
          /* Buffer for our outgoing frame */
          char outbuf[8000];
          /* Enough to store a full second */
          short buf[8000];
          int tail;
        int *enc;
        int *dec;
};

#define amr_coder_pvt ast_translator_pvt

static struct ast_translator_pvt *lintoamr_new(void) //for encoder
{
        struct amr_coder_pvt *tmp;

        tmp = malloc(sizeof(struct amr_coder_pvt));
```

```
      if (tmp) {
            /* Shut valgrind up */
            memset(&tmp->enc, 0, sizeof(tmp->enc));
//          enstate = Encoder_Interface_init(dtx);
            tmp->enc = Encoder_Interface_init(dtx);
            tmp->tail = 0;
            localusecnt++;
      }
      return tmp;
}

static struct ast_translator_pvt *amrtolin_new(void)
// for decoder relevant initialisation
{
      struct amr_coder_pvt *tmp;

      tmp = malloc(sizeof(struct amr_coder_pvt));
      if (tmp) {
            /* Shut valgrind up */
            memset(&tmp->dec, 0, sizeof(tmp->dec));
//          Speech_Decode_Frame_init(&tmp->dec, "Decoder");
         /* init decoder */
//          destate = Decoder_Interface_init(); //(int * destatee;)
            tmp->dec = Decoder_Interface_init();
            tmp->tail = 0;
            localusecnt++;
      }
      return tmp;
}

static struct ast_frame *lintoamr_sample(void)
{
      static struct ast_frame f;
      f.frametype = AST_FRAME_VOICE;
      f.subclass = AST_FORMAT_SLINEAR;
//    f.subclass = AST_FORMAT_AMR;
      f.datalen = sizeof(slin_amr_ex); // this is 320 in  char
      /* Assume 8000 Hz */
      f.samples = sizeof(slin_amr_ex)/2; //160?
      f.mallocd = 0;
      f.offset = 0;
      f.src = __PRETTY_FUNCTION__;
      f.data = slin_amr_ex; // 160 samples in signed short ?
      return &f;
}

static struct ast_frame *amrtolin_sample(void)
{
      static struct ast_frame f;
      f.frametype = AST_FRAME_VOICE;
      f.subclass = AST_FORMAT_AMR;
      f.datalen = sizeof(amr_slin_ex); //32 in char
      /* All frames are 20 ms long */
      f.samples = 160;
      f.mallocd = 0;
      f.offset = 0;
      f.src = __PRETTY_FUNCTION__;
      f.data = amr_slin_ex;  //this is char now?
      return &f;
}

static struct ast_frame *amrtolin_frameout(struct ast_translator_pvt
*tmp)
{
      if (!tmp->tail)
            return NULL;
      /* Signed linear is no particular frame size, so just send
whatever
         we have in the buffer in one lump sum */
      tmp->f.frametype = AST_FRAME_VOICE;
      tmp->f.subclass = AST_FORMAT_SLINEAR;
      tmp->f.datalen = tmp->tail * 2;
      /* Assume 8000 Hz */
      tmp->f.samples = tmp->tail;
      tmp->f.mallocd = 0;
      tmp->f.offset = AST_FRIENDLY_OFFSET;
      tmp->f.src = __PRETTY_FUNCTION__;
      tmp->f.data = tmp->buf;
      /* Reset tail pointer */
      tmp->tail = 0;

      return &tmp->f;
}

static int amrtolin_framein(struct ast_translator_pvt *tmp, struct
ast_frame *f)
{
      /* Assuming there's space left, decode into the current buffer at
         the tail location.  Read in as many frames as there are */
      int x, i;
      Word16 tmpf[160]; //for source samples //float in
ilbctolin_framein???
//    unsigned char serial[32];
//    short block_size[16]={ 12, 13, 15, 17, 19, 20, 26, 31, 5, 0, 0,
0, 0, 0, 0, 0 };v

      if (f->datalen % 32) {
                ast_log(LOG_WARNING, "Huh?  An AMR frame that isn't a
multiple of 32 bytes long from %s (%d)?\n", f->src, f->datalen);
                return -1;
        }


        for (x=0;x<f->datalen;x+=32) {
                if (tmp->tail + 160 < sizeof(tmp->buf)/2) {
                  Decoder_Interface_Decode(tmp->dec, f->data + x, tmpf,
0);

                        for (i=0;i<160;i++)
                            tmp->buf[tmp->tail + i] = tmpf[i];
                        tmp->tail+=160;
                } else {
                        ast_log(LOG_WARNING, "Out of buffer space\n");
                        return -1;
                }
        }
        return 0;
}


static int lintoamr_framein(struct ast_translator_pvt *tmp, struct
ast_frame *f)
{
      /* Just add the frames to our stream */
      /* XXX We should look at how old the rest of our stream is, and
if it
         is too old, then we should overwrite it entirely, otherwise we
can
         get artifacts of earlier talk that do not belong */

      if (tmp->tail + f->datalen/2 < sizeof(tmp->buf) / 2) {
            memcpy((tmp->buf + tmp->tail), f->data, f->datalen);
            tmp->tail += f->datalen/2;
      } else {
            ast_log(LOG_WARNING, "Out of buffer space\n");
            return -1;
      }
      return 0;
}

static struct ast_frame *lintoamr_frameout(struct ast_translator_pvt
*tmp)
{
      int x=0, i;
      int byte_counter;
      Word16 tmpf[160];
//    unsigned char serial[32]; //for serial bit streams in bits

      /* We can't work on anything less than a frame in size */
      if (tmp->tail < 160)
            return NULL;
      tmp->f.frametype = AST_FRAME_VOICE;
      tmp->f.subclass = AST_FORMAT_AMR;
      tmp->f.mallocd = 0;
      tmp->f.offset = AST_FRIENDLY_OFFSET;
      tmp->f.src = __PRETTY_FUNCTION__;
      tmp->f.data = tmp->outbuf;
      while(tmp->tail >= 160) {
               /* Encode a frame of data */
                if ((x+1)*32  >= sizeof(tmp->outbuf)) {
                        ast_log(LOG_WARNING, "Out of buffer space\n");
                        break;
                }

            for (i=0; i<160; i++)
                  tmpf[i] = tmp->buf[i];

            byte_counter = Encoder_Interface_Encode(tmp->enc, mode,
tmpf, ((unsigned char *)(tmp->outbuf)) + (x * 32), 0);
            /* Assume 8000 Hz -- 20 ms */
            tmp->tail -= 160;
```

```
			/* Move the data at the end of the buffer to the front */
			if (tmp->tail)
				memmove(tmp->buf, tmp->buf + 160, tmp->tail * 2);
			x++;
		}
		tmp->f.datalen = x *32;
		tmp->f.samples = x*160;

#if 0
		/* Save to a AMR sample output file...*/
		{
			static int fd = -1;
			if (fd == -1) {
				fd = open("amr.out", O_CREAT|O_TRUNC|O_WRONLY, 0666);
				write(fd, tmp->f.data, tmp->f.datalen);
				close(fd);
			}
		}
#endif
		return &tmp->f;
}

static void amr_destroy_stuff(struct ast_translator_pvt *pvt)
{
		free(pvt);
		localusecnt--;
		ast_update_use_count();
}

static struct ast_translator amrtolin =
		{ "amrtolin",
		   AST_FORMAT_AMR, AST_FORMAT_SLINEAR,
		   amrtolin_new,
		   amrtolin_framein,
		   amrtolin_frameout,
		   amr_destroy_stuff,
		   amrtolin_sample
		   };

static struct ast_translator lintoamr =
		{ "lintoamr",
		   AST_FORMAT_SLINEAR, AST_FORMAT_AMR,
		   lintoamr_new,
		   lintoamr_framein,
		   lintoamr_frameout,
		   amr_destroy_stuff,
		   lintoamr_sample
		   };

int unload_module(void)
{
		int res;
		ast_mutex_lock(&localuser_lock);
		res = ast_unregister_translator(&lintoamr);
		if (!res)
			res = ast_unregister_translator(&amrtolin);
		if (localusecnt)
```

```
            res = -1;
      ast_mutex_unlock(&localuser_lock);
      return res;
}

int load_module(void)
{
      int res;
      res=ast_register_translator(&amrtolin);

      if (!res)
            res=ast_register_translator(&lintoamr);
      else
            ast_unregister_translator(&amrtolin);
      return res;
}

char *description(void)
{
      return tdesc;
}

int usecount(void)
{
      int res;
      STANDARD_USECOUNT(res);
      return res;
}

char *key()
{
      return ASTERISK_GPL_KEY;
}
```

A.2 Make File

```
# Asterisk -- A telephony toolkit for Linux.
# Makefile for codec modules
# Copyright (C) 1999-2005, Digium
# Mark Spencer <markster@digium.com>
# This program is free software, distributed under the terms of
# the GNU General Public License
ifeq (${OSARCH},CYGWIN)
CYGSOLINK=-Wl,--out-implib=lib$@.a -Wl,--export-all-symbols
CYGSOLIB=-L.. -L. -lasterisk.dll
else
CFLAGS+=-fPIC
endif
ifneq ($(wildcard g723.1/coder.c),)
  MODG723=codec_g723_1.so
  LIBG723=g723.1/libg723.a
```

```
endif

ifneq ($(wildcard g723.1b/coder2.c),)
 MODG723+=codec_g723_1b.so
 LIBG723B=g723.1b/libg723b.a
endif

ifneq ($(wildcard amr_float/sp_enc.c),)
 MODAMR=codec_amr.so
 LIBAMR=amr_float/libamrfloat.a
endif

UI_SPEEX=$(wildcard $(CROSS_COMPILE_TARGET)/usr/include/speex.h)
UIS_SPEEX=$(wildcard $(CROSS_COMPILE_TARGET)/usr/include/speex/speex.h)
ULI_SPEEX=$(wildcard $(CROSS_COMPILE_TARGET)/usr/local/include/speex.h)
ULIS_SPEEX=$(wildcard
$(CROSS_COMPILE_TARGET)/usr/local/include/speex/speex.h)
ifneq (${UI_SPEEX},)
 MODSPEEX=codec_speex.so
 LIBSPEEX+=-lspeex -lm
endif
ifneq (${UIS_SPEEX},)
 MODSPEEX=codec_speex.so
 CFLAGS+=-I$(CROSS_COMPILE_TARGET)/usr/include/speex
 LIBSPEEX+=-lspeex -lm
endif
ifneq (${ULI_SPEEX},)
 MODSPEEX=codec_speex.so
 CFLAGS+=-I$(CROSS_COMPILE_TARGET)/usr/local/include
 LIBSPEEX=-L$(CROSS_COMPILE_TARGET)/usr/local/lib
 LIBSPEEX+=-lspeex -lm
endif
ifneq (${ULIS_SPEEX},)
 MODSPEEX=codec_speex.so
 CFLAGS+=-I$(CROSS_COMPILE_TARGET)/usr/local/include/speex
 LIBSPEEX=-L$(CROSS_COMPILE_TARGET)/usr/local/lib
 LIBSPEEX+=-lspeex -lm
endif

ifneq ($(wildcard ilbc/iLBC_decode.h),)
 MODILBC=codec_ilbc.so
 LIBILBC=ilbc/libilbc.a
endif


LIBGSM=gsm/lib/libgsm.a
```

```
LIBGSMT=gsm/lib/libgsm.a
LIBLPC10=lpc10/liblpc10.a
#added by lfsun
LIBAMR=amr_float/libamrfloat.a

ifeq ($(findstring BSD,${OSARCH}),BSD)
  CFLAGS+=-I$(CROSS_COMPILE_TARGET)/usr/local/include                          -
L$(CROSS_COMPILE_TARGET)/usr/local/lib
endif

CODECS+=$(MODG723) $(MODSPEEX) $(MODILBC) codec_gsm.so codec_lpc10.so
\
      codec_adpcm.so codec_ulaw.so codec_alaw.so codec_a_mu.so \
         codec_g726.so codec_amr.so

all: depend $(CODECS)

clean:
         rm -f *.so *.o .depend
         [ ! -d g723.1 ] || $(MAKE) -C g723.1 clean
         [ ! -d g723.1b ] || $(MAKE) -C g723.1b clean
         $(MAKE) -C gsm clean
         $(MAKE) -C lpc10 clean
         $(MAKE) -C ilbc clean

$(LIBG723):
         $(MAKE) -C g723.1 all

$(LIBGSM):
         $(MAKE) -C gsm lib/libgsm.a

$(LIBG723B):
         $(MAKE) -C g723.1b all

$(LIBLPC10):
         $(MAKE) -C lpc10 all

$(LIBILBC):
         $(MAKE) -C ilbc all

$(LIBAMR):
         $(MAKE) -C amr_float libamrfloat.a

$(MODILBC): codec_ilbc.o $(LIBILBC)
         $(CC) $(SOLINK) -o $@ ${CYGSOLINK} $< ${CYGSOLIB} $(LIBILBC)
```

```
codec_g723_1.so : codec_g723_1.o $(LIBG723)
	$(CC) $(SOLINK) -o $@ ${CYGSOLINK} $< ${CYGSOLIB} $(LIBG723)

codec_g723_1b.o : codec_g723_1.c
	$(CC) -c -o $@ $(CFLAGS) -DANNEX_B -Dsingle $<

codec_g723_1b.so : codec_g723_1b.o $(LIBG723B)
	$(CC) $(SOLINK) -o $@ ${CYGSOLINK} $< ${CYGSOLIB} $(LIBG723B) -
lm

codec_gsm.so: codec_gsm.o $(LIBGSMT)
	$(CC) $(SOLINK) -o $@ ${CYGSOLINK} $< ${CYGSOLIB} $(LIBGSM)

codec_amr.so: codec_amr.o $(LIBAMR)
	$(CC) $(SOLINK) -o $@ ${CYGSOLINK} $< ${CYGSOLIB} $(LIBAMR) -lm

codec_amr.o: codec_amr.c
	$(CC) -c -o $@ $(CFLAGS) -Dsingle $<

$(MODSPEEX): codec_speex.o
	$(CC) $(SOLINK) -o $@ ${CYGSOLINK} $< ${CYGSOLIB} $(LIBSPEEX)

codec_lpc10.so: codec_lpc10.o $(LIBLPC10)
	$(CC) $(SOLINK) -o $@ ${CYGSOLINK} $< ${CYGSOLIB} $(LIBLPC10) -
lm

%.so : %.o
	$(CC) $(SOLINK) -o $@ ${CYGSOLINK} $< ${CYGSOLIB}

ifneq ($(wildcard .depend),)
 include .depend
endif

install: all
	for x in $(CODECS); do $(INSTALL) -m 755 $$x
$(DESTDIR)$(MODULES_DIR) ; done

depend: .depend

.depend:
	../build_tools/mkdep $(CFLAGS) `ls *.c`
```

A.3 Format_amr.c

```
/*
 * Asterisk -- An open source telephony toolkit.
 *
 * Brian K. West <brian@bkw.org>
 *
 * Copyright (C) 1999 - 2005, Digium, Inc.
 *
 * Mark Spencer <markster@digium.com>
 *
 * See http://www.asterisk.org for more information about
 * the Asterisk project. Please do not directly contact
 * any of the maintainers of this project for assistance;
 * the project provides a web site, mailing lists and IRC
 * channels for your use.
 *
 * This program is free software, distributed under the terms of
 * the GNU General Public License Version 2. See the LICENSE file
 * at the top of the source tree.
 */

/*! \file
 *
 * \brief Save to raw, headerless AMR data.
 * \arg File name extension: amr
 * \ingroup formats
 */
/*
 * modified from format_ilbc.c *
*/
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <sys/time.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>

#include "asterisk.h"

ASTERISK_FILE_VERSION(__FILE__, "$Revision: 7221 $")

#include "asterisk/lock.h"
#include "asterisk/channel.h"
#include "asterisk/file.h"
#include "asterisk/logger.h"
#include "asterisk/sched.h"
#include "asterisk/module.h"
#include "asterisk/endian.h"

/* Some Ideas for this code came from makeg729e.c by Jeffrey Chilton */

/* Portions of the conversion code are by guido@sienanet.it */

struct ast_filestream {
      void *reserved[AST_RESERVED_POINTERS];
      /* Believe it or not, we must decode/recode to account for the
```

```
	   weird MS format */
	/* This is what a filestream means to us */
	FILE *f; /* Descriptor */
	struct ast_frame fr;				/* Frame information */
	char waste[AST_FRIENDLY_OFFSET];	/* Buffer for sending frames,
etc */
	char empty;						/* Empty
character */
	unsigned char amr[32];				/* One Real iLBC Frame
*/
};


AST_MUTEX_DEFINE_STATIC(amr_lock);
static int glistcnt = 0;

static char *name = "AMR";
static char *desc = "Raw AMR data";
static char *exts = "AMR";

static struct ast_filestream *amr_open(FILE *f)
{
	/* We don't have any header to read or anything really, but
	   if we did, it would go here.  We also might want to check
	   and be sure it's a valid file.  */
	struct ast_filestream *tmp;
	if ((tmp = malloc(sizeof(struct ast_filestream)))) {
		memset(tmp, 0, sizeof(struct ast_filestream));
		if (ast_mutex_lock(&amr_lock)) {
			ast_log(LOG_WARNING, "Unable to lock AMR list\n");
			free(tmp);
			return NULL;
		}
		tmp->f = f;
		tmp->fr.data = tmp->amr;
		tmp->fr.frametype = AST_FRAME_VOICE;
		tmp->fr.subclass = AST_FORMAT_AMR;
		/* datalen will vary for each frame */
		tmp->fr.src = name;
		tmp->fr.mallocd = 0;
		glistcnt++;
		ast_mutex_unlock(&amr_lock);
		ast_update_use_count();
	}
	return tmp;
}

static struct ast_filestream *amr_rewrite(FILE *f, const char *comment)
{
	/* We don't have any header to read or anything really, but
	   if we did, it would go here.  We also might want to check
	   and be sure it's a valid file.  */
	struct ast_filestream *tmp;
	if ((tmp = malloc(sizeof(struct ast_filestream)))) {
		memset(tmp, 0, sizeof(struct ast_filestream));
		if (ast_mutex_lock(&amr_lock)) {
			ast_log(LOG_WARNING, "Unable to lock AMR list\n");
```

```
                free(tmp);
                return NULL;
            }
            tmp->f = f;
            glistcnt++;
            ast_mutex_unlock(&amr_lock);
            ast_update_use_count();
        } else
            ast_log(LOG_WARNING, "Out of memory\n");
        return tmp;
}

static void amr_close(struct ast_filestream *s)
{
        if (ast_mutex_lock(&amr_lock)) {
            ast_log(LOG_WARNING, "Unable to lock AMR list\n");
            return;
        }
        glistcnt--;
        ast_mutex_unlock(&amr_lock);
        ast_update_use_count();
        fclose(s->f);
        free(s);
        s = NULL;
}

static struct ast_frame *amr_read(struct ast_filestream *s, int
*whennext)
{
        int res;
        /* Send a frame from the file to the appropriate channel */
        s->fr.frametype = AST_FRAME_VOICE;
        s->fr.subclass = AST_FORMAT_AMR;
        s->fr.offset = AST_FRIENDLY_OFFSET;
        s->fr.samples = 160;
        s->fr.datalen = 32;
        s->fr.mallocd = 0;
        s->fr.data = s->amr;
        if ((res = fread(s->amr, 1, 32, s->f)) != 32) {
            if (res)
                ast_log(LOG_WARNING, "Short read (%d) (%s)!\n", res,
strerror(errno));
            return NULL;
        }
        *whennext = s->fr.samples;
        return &s->fr;
}

static int amr_write(struct ast_filestream *fs, struct ast_frame *f)
{
        int res;
        if (f->frametype != AST_FRAME_VOICE) {
            ast_log(LOG_WARNING, "Asked to write non-voice frame!\n");
            return -1;
        }
        if (f->subclass != AST_FORMAT_AMR) {
            ast_log(LOG_WARNING, "Asked to write non-AMR frame
(%d)!\n", f->subclass);
            return -1;
      }
      if (f->datalen % 32) {
            ast_log(LOG_WARNING, "Invalid data length, %d, should be
multiple of 32\n", f->datalen);
            return -1;
      }
      if ((res = fwrite(f->data, 1, f->datalen, fs->f)) != f->datalen)
{
                  ast_log(LOG_WARNING, "Bad write (%d/32): %s\n", res,
strerror(errno));
                  return -1;
      }
      return 0;
}

static char *amr_getcomment(struct ast_filestream *s)
{
      return NULL;
}

static int amr_seek(struct ast_filestream *fs, long sample_offset, int
whence)
{
      long bytes;
      off_t min,cur,max,offset=0;
      min = 0;
      cur = ftell(fs->f);
      fseek(fs->f, 0, SEEK_END);
      max = ftell(fs->f);

      bytes = 32 * (sample_offset / 160);
      if (whence == SEEK_SET)
            offset = bytes;
      else if (whence == SEEK_CUR || whence == SEEK_FORCECUR)
            offset = cur + bytes;
      else if (whence == SEEK_END)
            offset = max - bytes;
      if (whence != SEEK_FORCECUR) {
            offset = (offset > max)?max:offset;
      }
      /* protect against seeking beyond begining. */
      offset = (offset < min)?min:offset;
      if (fseek(fs->f, offset, SEEK_SET) < 0)
            return -1;
      return 0;
}

static int amr_trunc(struct ast_filestream *fs)
{
      /* Truncate file to current length */
      if (ftruncate(fileno(fs->f), ftell(fs->f)) < 0)
            return -1;
      return 0;
}
```

```
static long amr_tell(struct ast_filestream *fs)
{
      off_t offset;
      offset = ftell(fs->f);
      return (offset/32)*160;
}

int load_module()
{
      return ast_format_register(name, exts, AST_FORMAT_AMR,
                                                    amr_open,
                                                    amr_rewrite,
                                                    amr_write,
                                                    amr_seek,
                                                    amr_trunc,
                                                    amr_tell,
                                                    amr_read,
                                                    amr_close,
                                                    amr_getcomment);


}

int unload_module()
{
      return ast_format_unregister(name);
}

int usecount()
{
      return glistcnt;
}

char *description()
{
      return desc;
}


char *key()
{
      return ASTERISK_GPL_KEY;
}
```

A.4 MakeFile

```
#
# Asterisk -- A telephony toolkit for Linux.
#
# Makefile for file format modules
```

```
#
# Copyright (C) 1999-2005, Digium
#
# Mark Spencer <markster@digium.com>
#
# This program is free software, distributed under the terms of
# the GNU General Public License
#

FORMAT_LIBS=format_gsm.so format_wav.so \
        format_wav_gsm.so format_vox.so format_pcm.so format_g729.so \
        format_pcm_alaw.so format_h263.so format_g726.so format_ilbc.so \
        format_sln.so format_au.so format_amr.so
FORMAT_LIBS+=format_jpeg.so

#
# G723 simple frame is deprecated
#
FORMAT_LIBS+=format_g723.so

#
# OGG/Vorbis format
#
ifneq ($(wildcard $(CROSS_COMPILE_TARGET)/usr/include/vorbis/codec.h),)
  FORMAT_LIBS+=format_ogg_vorbis.so
endif

ifeq ($(findstring BSD,${OSARCH}),BSD)
  CFLAGS+=-I$(CROSS_COMPILE_TARGET)/usr/local/include -
L$(CROSS_COMPILE_TARGET)/usr/local/lib
endif

GSMLIB=../codecs/gsm/lib/libgsm.a

ifeq (${OSARCH},CYGWIN)
CYGSOLINK=-Wl,--out-implib=lib$@.a -Wl,--export-all-symbols
CYGSOLIB=-L.. -L. -lasterisk.dll
else
CFLAGS+=-fPIC
endif
all: depend $(FORMAT_LIBS)

clean:
        rm -f *.so *.o .depend

%.so : %.o
```

```
	$(CC) $(SOLINK) -o $@ ${CYGSOLINK} $< ${CYGSOLIB}

ifneq ($(wildcard .depend),)
 include .depend
endif

format_mp3.so : format_mp3.o
	$(CC) $(SOLINK) -o $@ ${CYGSOLINK} $< ${CYGSOLIB} -lm

format_amr.so : format_amr.o
	$(CC) $(SOLINK) -o $@ ${CYGSOLINK} $< ${CYGSOLIB} -lm

format_ogg_vorbis.so : format_ogg_vorbis.o
	$(CC) $(SOLINK) -o $@ ${CYGSOLINK} $< ${CYGSOLIB} -logg -lvorbis -
lvorbisenc -lm

install: all
	for x in $(FORMAT_LIBS); do $(INSTALL) -m 755 $$x
$(DESTDIR)$(MODULES_DIR) ; done

depend: .depend

.depend:
	../build_tools/mkdep $(CFLAGS) `ls *.c`
```

A.5 slin_amr_ex.h

```
/*! \file
 * \brief Signed 16-bit audio data
 *
 * Source: gsm.example
 *
 * Copyright (C) 1999-2005, Digium Inc
 *
 * Distributed under the terms of the GNU General Public License
 *
 */

static signed short slin_amr_ex[] = {
000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000,
000000,
000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000,
000000,
000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000,
000000,
000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000,
000000,
000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000,
000000,
000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000,
000000,
000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000, 0xfff8,
000000,
000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000,
000000,
000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000,
000000,
000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000,
000000,
000000, 0x0008, 000000, 000000, 000000, 000000, 000000, 000000, 000000,
000000,
000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000,
000000,
0x0008, 000000, 000000, 000000, 0xfff8, 000000, 000000, 000000, 000000,
000000,
000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000,
000000,
000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000, 000000,
000000,
000000, 000000, 000000, 000000, 000000, 000000, 0x0008, 000000, 000000,
000000 };
```

A.6

The following code should be added to the following files

frame.c file in the /include/asterisk/ directory
rtp.c
channel.c
frame.c in the main source directory of the Asterisk Package

```
/*! AMR codec */
#define AST_FORMAT_AMR        (1 << 11)
```

M.Res   Research Paper

How to call from a 3G Mobile to a SIP phone - Implementation based on Asterisk

P.Enjapuri, L.Sun, Emmanuel C. Ifeachor

University of Plymouth, Plymouth, PL4 8AA, UK
Praveen.enjapuri@postgrad.plymouth.ac.uk.

Abstract

This paper describes how a 3G Mobile can call a SIP phone. Asterisk is made to act as a bridge for video call between 3G-IP network. Asterisk to act as a gateway must capture the audio/video stream from 3G mobile, convert it to a stream compatible for IP client and pass it to it and vice-versa. Asterisk needs to support AMR codec for audio and MPEG-4 Codec for video and H.324M protocol stack for capturing audio/streams from 3G Mobile. This paper describes the implementation of AMR codec into Asterisk, SIP signaling and SIP softphones.

Keywords

3G, Asterisk, Gateway, AMR-NB, SIP, IP

1 Introduction

Technology allowed a mobile phone to call with another mobile phone, a land line phone to call another landline phone, and a mobile phone to communicate with landline phone. Later VOIP technology brought soft phones which are a piece of software for making telephone calls over the internet using a PC. Softphones allow two PC's to communiate with each other through internet or they can even be connected to a PBX. This paper introduces the innovative idea of a 3G mobile calling a SIP soft phone. The project aim is to make a video call from 3G mobile to a SIP client via Asterisk. Thus Asterisk acts as a gateway between 3G and IP network. We intend to implement AMR codec, H.324M protocol stack, MPEG-4, bridging functions between SIP/RTP and ISDN in Asterisk so that it can act as path for the video call between 3G-IP. This paper discusses only the implementation of AMR codec .

Section 2 describes the general architecture of the 3G-Asterisk-IP system in detail. Section 3 describes Asterisk and AMR implementation in Aserisk. Section 4 covers SIP signaling and configuring various SIP clients with Asterisk.

2. 3G-H324M-SIP System Structure

Below figure 1 is the structure for 3G-Asterisk-IP System

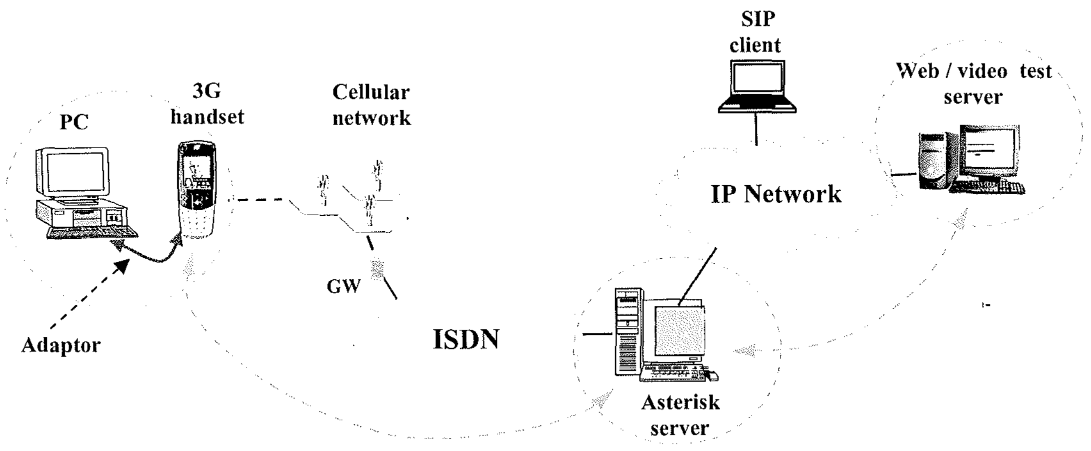


Figure 1: 3G-Asterisk-IP System (Lingfen Sun 2006)

The system has two block 3G-Asterisk & Asterisk-IP. So for a successful video call Asterisk should act as a path for exchanging of media between 3G and SIP softphone. 3G mobiles use AMR as audio codec and MPEG-4 as video codec. The problem is Asterisk currently doesn't support AMR as well as MPEG-4 codec's which we need to implement in it. Their needs to be a feature in Asterisk so that it can capture the audio/video streams from 3G mobile and pass on to the IP block, for this H324M protocol stack needs to be implemented in Asterisk(Not discussed in this paper. See Zeeshan's report). Also bridging functions need to be implemented in Asterisk which converts SIP/RTP signaling to H.324M audio/video signaling. Also the quality of the audio/video streams captured from 3G mobile into Asterisk need to be evaluated using standard speech/video quality measurement techniques like PESQ, PEVQ.

Asterisk functionality as a gateway for a video between 3G-SIP IP client can be understood in detail from the figure 2

<u>3G Network</u> -

3GPP and 3GPP2 have adopted the 3G-324M protocol as standard for transporting conversational video over mobile network. The 3G-324M protocol combines voice, video, data and control into a single 64kbps stream of circuit-switched data. 3G-324M based video content is carried by a single H.223 64kbps stream that multiplexes audio, video, data and control information. The video portion of H.223 stream uses MPEG-4 codec, whereas audio content is based on AMR compression technique. The control portion of the H223 stream is based on H.245 protocol which controls the session and takes care of channel parameter exchanges.

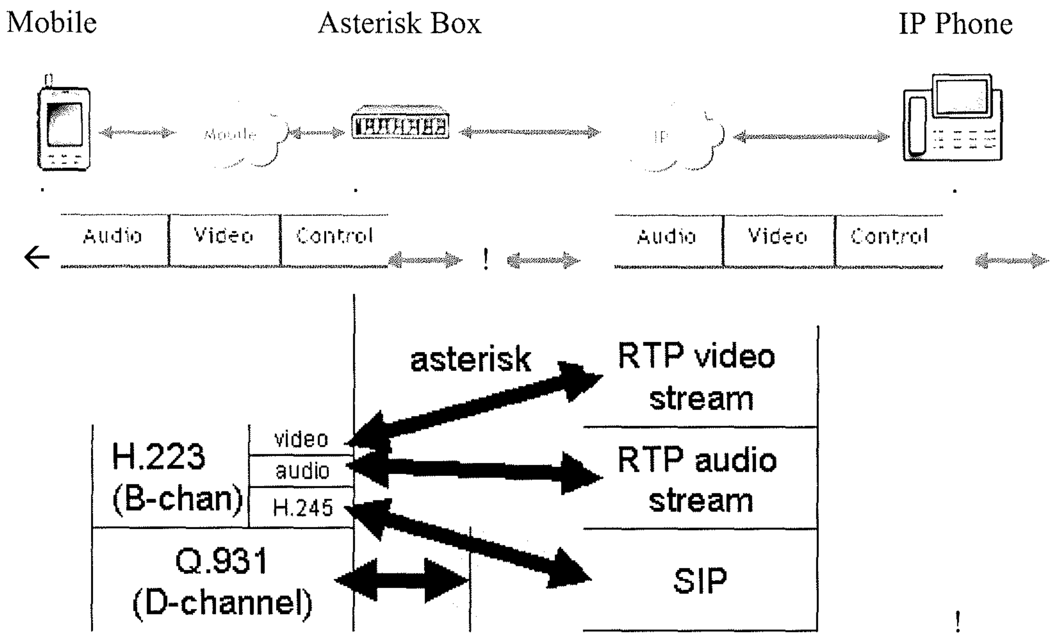


Figure 2 : 3G-H324M/ISDN – SIP Video Call

IP Network -

The transportation of the real-time video in Voice over IP networks is done in a different way using three different UDP/IP Streams. RTP/RTCP is used to transport voice, video and control information on different streams. Two streams are used for voice and video and an RTCP stream for control. The video stream uses H.264/H.263/H.261 or MPEG4 compression technique and audio stream uses G.711, G.729, G.723 or any other voice compression scheme. The compressed audio and video streams are encapsulated in an RTP stream which is then placed in UDP/IP Packet. Signaling and control information is carried using either SIP (uses SDP for controlling info) or H.323 (which uses H.245 for control info).

Asterisk should do the following to act as a video gateway

i) SIP must be translated to H.245 and vice-versa

ii) RTP stream must be translated into corresponding AMR, MPEG-4 streams and vice-versa. (Converting one form of audio/video codec to another audio/video codec called as audio/video Transcoding)

iii) All of them must be multiplexed into a B-channel if a SIP client is communicating with 3G mobile.

Asterisk PBX to perform the above functions needs to support AMR, MPEG-4 and H324M Protocol Stack.

3 Asterisk & AMR codec Implementation

Asterisk PBX is an open source software Licensed under GPL. It's completely written in c. Asterisk connects telephony technologies (like SIP, H323, IAX, MGCP) with telephony applications (include bridging, conferencing, voicemail etc). It allows telephones (IP phones or hard phones) connected to it to call each other and also to PSTN. Asterisk thus can act as a registrar for IP phones, gateway between IP phones and PSTN.

Configuration files, dialplans, applications, modules, codec's are some of the main components of Asterisk Software. Asterisk supports VOIP protocols such as SIP, IAX, and H.323, MGCP (Jim Van Meggelen, Jared Smith, and Leif Madsen 2003)

3.1 Adaptive Multi Rate Speech Codec

AMR is a popular speech codec used in VOIP applications. Adaptive Multi-Rate is standardized by ETSI, 3GPP, as well as ITU-T. We used AMR-NB from 3GPP for implementation in Asterisk.

There is no algorithmic difference between ETSI AMR and 3GPP AMR standards. 3GPP-AMR support packing of coded bits and unpacking of bits before decoding. This packing and unpacking is done as per RFC3267.In ETSI AMR DTX ON, 245th word in coded bit stream is always value of mode i.e. from 0 to 8, where 8 for DTX mode. In 3GPP AMR for DTX ON, 245th word is value of mode for non DTX frame but it is equal to -1, if DTX frame is found. In ETSI one flag called NSYNC is used as input from the n/w. In 3GPP this is not used. The NSYNC is used in mobile communication when switching from one Base station to next base station. This alters the VAD operation. When ever NSYNC is set, the mode will not be NODTX. (B. Besette, R. Salami et al 2002)
GSM-AMR or simply AMR is just another name for AMR-NB. The AMR-NB codec is a multi-mode codec that supports 8 narrow band speech encoding modes with bit rates between 4.75 and 12.2 kbps. These bit-rate modes are also denoted with indices from 0 to 7 where 0 maps to the 4.75 kbits/s mode and 7 maps to the 12.2 kbits/s. Frequency used in AMR-NB is 8000 Hz and the speech encoding is performed on 20 ms speech frames. Therefore, each encoded AMR-NB speech frame represents 160 samples of the original speech. AMR uses different techniques, such as Algebraic Code Excited Linear Prediction (ACELP), Discontinuous Transmission (DTX), Voice Activity Detection (VAD) and Comfort Noise Generation (CNG).

3.1.1 AMR Frame Structure for AMR-NB:

This AMR codec works on the principle of Algebraic Code Excited Linear Prediction (ACELP) for all bit rates. To reduce average bit rate, this codec supports the discontinuous transmission (DTX), using Voice Activity Detection (VAD) and Comfort Noise Generation (CNG) algorithms.

The output file from the AMR-NB encoder has either .AMR extension or .COD extension. .AMR files are stored in "AMR File Storage Format1" (discussed in section 3.3.1). Others have ".COD" extension and are stored in "AMR Interface Format 2." (discussed below). This is specified in 3GPP TS 26.101, Appendix A. These are coded and decoded as per 3GPP TS 26.104 floating point reference codec source package. AMR IF1 and IF2 are discussed in the later sections. Below are the differences between these two file types.

i)'.AMR' files have a 6 byte header: "#!AMR\n" where as '.COD' files don't have a header.

ii).AMR files take two nibbles (8 bits) to express the frame flags [Format: "PTTTTVPP", where P is "Pad" (0), T is "Frame Type", and V is "Valid"]. Where as '.COD' files require a single nibble to express the frame flags. The actual speech frame bits begin directly after this, in the 2nd nibble of the first byte of the frame.

iii)'.AMR' files pack the bits of the frame into bytes in big-endian order, that is, the most significant bit (0x80) of each byte is actually the first bit of the byte, and the least significant bit (0x1) is the last bit. '.COD' files pack the bits of the frame into bytes in little-endian order. The least significant bit is the first bit of the byte, followed by the 0x2 bit, the 0x4 bit, etc. (3GPP TS 26.071)

The Frame Structure for AMR Interface Format1(AMR IF1) consists of three parts: AMR Header, AMR Auxiliary Information, and AMR Core Frame. The AMR Header part includes the Frame Type and the Frame Quality Indicator fields. The AMR auxiliary information part includes the Mode Indication, Mode Request, and Codec CRC fields. The AMR Core Frame part consists of the speech parameter bits or, in case of a comfort noise frame, the comfort noise parameter bits. In case of a comfort noise frame, the comfort noise parameters replace Class A bits of AMR Core Frame while Class B and C bits are omitted.

The Frame Structure for AMR Interface Format2 (AMR IF2) consists of 4-bit Frame Type field (similar to IF1), AMR Core Frame (similar to IF1). The total number of bits in the AMR IF2 speech frames in the different modes is typically not a multiple of eight and bit stuffing is needed to achieve an octet structure.(3GPP TS 26.193)

3.2 AMR Implementation in Asterisk

This section will elaborate on the actual implementation. Codec is encoding and compression algorithm utilized by applications. All the source files for the codec's currently supported by asterisk are placed under /codecs/ directory of the source package.

The implementation is based on the concept that the core encoding method in asterisk is AST_FORMAT_SLINEAR, in which non-compressed audio is sampled at 8000 times a second, with 16-bit signed samples. Asterisk basically translates voice frames from SLINEAR to other compression techniques and vice versa.

codec_amr.c: So we first started writing AMR as a standalone program which will be loaded as a module. The name of this file is codec_amr.c. As per the Asterisk standard the AMR should have the following functions to successfully load it as a module.
int load_module():- In this module asterisk checks if it can load the AMR module successfully. It checks whether it can translate the voice frames from SLINEAR to AMR format and vice-versa. If it is the case then it returns zero, otherwise it returns non-zero.

int unload_module ():- This function will unload the AMR module. This function returns zero if it successfully unloads the module otherwise it returns non-zero. Usually module can only be unloaded if the no of channels using the module indicated by the variable usecount is zero.

char *description():- This function returns the codec functionality.

int usecnt():- This function return the no of channels using this AMR module.

Then various functions like *lintoamr_new, *amrtolin_new, *lintoamr_sample are implemented by using file codec_ilbc as a template.
The implementation of this file is explained in detail in Appendix A.1. The corresponding Makefile in the same directory has to be edited so that asterisk can recognize this codec(The Makefile is described in detail in Appendix A.2).

format_amr.c :Also format_amr.c is created in /formats/ directory to handle .amr extension sound files. The corresponding MakeFile in the same directory is edited to load this format_amr.c file. The implementation of this file is explained in detail in Appendix A.3.
The corresponding Makefile in the same directory has to be edited so that asterisk can recognize this format.(The Makefile is described in detail in Appendix A.4).

The following files in the asterisk have to be modified for AMR codec implementation to work.

Frame.c:- AST_FORMAT_AMR should be added to list of media format supported.

Ex :
```
/*! AMR codec */
#define AST_FORMAT_AMR        (1 << 11)
```

Rtp.c:- AST_FORMAT_AMR should be added to the list of RTP Payload types supported.

Ex :
```
/*! AMR codec */
#define AST_FORMAT_AMR        (1 << 11)
```

Channel.c :- In this file also AST_FORMAT_AMR should be added to the list of preferred codec formats.

```
Ex :     /*! AMR codec */
         #define AST_FORMAT_AMR        (1 << 11)
```

/include/frame.h:- Here also AST_FORMAT_AMR should be added to the Asterisk Frame Definition List.

```
Ex:    /*! AMR codec */
    #define AST_FORMAT_AMR           (1 << 11)
```

4 SIP Protocol & Issues

4.1 SIP Protocol

Session Initiation Protocol (SIP) is IETF's standard for multimedia conferencing over IP. SIP is ASCII-based that can be used for forming, maintaining, and terminating sessions that involves multimedia elements such as video, audio etc. SIP architecture has two basic components. SIP User Agent and SIP Network Server. The user agent is the end system component that initiates the call, which can be represented by a hardware or software device implementing SIP (e.g., an IP phone), and consists of two main components: User Agent Client (UAC) and User Agent Server (UAS) .UAC component is a client application that initiates the SIP request. Where as UAS Server application that contacts the user when a SIP request is received and that returns a response on behalf of the user.
SIP Network server is a network device that handles the signaling associated with multiple calls. It consists of SIP Register Server, SIP Proxy Server, and SIP Redirect Server. SIP messages are exchanged when a call is set up, modified, and terminated. SIP message can either be a request or response message. SIP message (request or response) consists of start line, One or more header fields, an empty line, A message body (optional).Each line must end with a carriage return-line feed (CRLF).
SIP acts as a carrier for the Session Description Protocol (SDP), which describes the media content for the multimedia session. SIP carries only signalling information whereas RTP actually carries the media content to the destination.
SIP establishes communication between two or more users. Users in a SIP network are identified by unique SIP addresses. A SIP address is similar to an e-mail address and is in the format of SIP:userID@gateway.com. The user ID can be either a user name or an E.164 address.

As part of this project we configured no of SIP soft phones with asterisk Kphone is one such linux based softphone and its configuration in asterisk so that it can call another Kphone. The procedure is discussed below.
i) Edit SIP.conf file in /etc/asterisk directory as below for users (softphones) to register

```
[praveen]
type=friend
secret=uop
```

```
username=praveen
host=dynamic
defaultip=141.163.8.211
context = trusted
```

ii) Then we need to edit the extension.conf file in /etc/asterisk/ directory. The extension.conf file decides how Asterisk handles incoming calls. It should be modified as below [trusted]

```
exten => 1xxx,1,Dial(SIP/praveen) exten=>1xxx,2,Hungup()
```

For a call between two kphones we need to do the following steps in sequence. i) Start the Asterisk.ii) Start Kphones on machines. iii) To configure audio/ video codecs in the main window click preferences >iv)To configure it to register with Asterisk in main window click File > Identity ... in the menu bar fill the full name field as praveen, user part of url as praveen, in the host part of url give the ipaddress on which the Asterisk you want to connect to is running. This can be found out by 'ifconfig' command on the linux console.Leave the remaining fields blank and press register button at the bottom. If the softphone is registered successfully with asterisk then it will display the window saying registration successful otherwise it display a window asking to try again.

. 4 Conclusions

This paper discusses the concept of 3G- SIP softphone video call via Asterisk. Asterisk as a gateway directs the media (audio + video + control) from 3G Mobile to soft phone and vice-versa. Asterisk needs to have the necessary features act as a path for the media. H324M protocol is implemented in asterisk so that it can capture the streams from 3G mobile but it needs to be tested properly for bugs. And also we implemented AMR Codec in Asterisk but it needs to be tested properly for the functionality. The implementation of H.324M protocol stack in asterisk as part of this project can only handle a call from 3G mobile. The code can still be optimized to handle calls from multiple mobiles at the same time. Also the quality of the video and audio clips recorded by the asterisk from 3G mobile needs to be evaluated.

References

1 Lingfen Sun 2006 'Video Quality Measurement for 3G Handsets-Project Proposal'- University Of plymouth
2 Jim Van Meggelen, Jared Smith, and Leif Madsen, 2003 *Asterisk:The Future of Telephony.*
3 3GPP TS 26.071 "Mandatory Speech Codec speech processing functions AMR Speech Codec".
4 3GPP TS 26.193 "*AMR Wideband Speech Codec; Source Controlled Rate Operation*"
5 3GPP TS 26.192 "*AMR Wideband Speech Codec; Comfort Noise Aspects*"
6 *AMR-WB White Paper v07-07-2003*, "Wideband speech coding standards and applications", VoiceAge Corp., http://www.voiceage.com, (Acc. 06 08 24).