Implementing an application for communication and quality measurements over UMTS networks

Examensarbete utfört i Kommunikationssystem vid Tekniska Högskolan i Linköping
av

Kenth Fredholm, Kristian Nilsson

Reg nr: LiTH-ISY-EX-3369–2003
Implementing an application for communication and quality measurements over UMTS networks

Examensarbete utfört i Kommunikationssystem vid Tekniska Högskolan i Linköping

av

Kenth Fredholm, Kristian Nilsson

Reg nr: LiTH-ISY-EX-3369–2003

Supervisors: Per Elmdahl, Frida Gunnarsson, Jonas Olsson, Frans Unosson

Examiner: Fredrik Gunnarsson

Linköping, 14th March 2003.
The interest for various multimedia services accessed via the Internet has been growing immensely along with the bandwidth available. A similar development has emerged in the 3G mobile network. The focus of this master thesis is on the speech/audio part of a 3G multimedia application. The purpose has been to implement a traffic generating tool that can measure QoS (Quality of Service) in 3G networks. The application is compliant to the 3G standards, i.e. it uses AMR (Adaptive Multi Rate), SIP (Session Initiation Protocol) and RTP (Real Time Transport Protocol). AMR is a speech compression algorithm with the special feature that it can compress speech into several different bit-rates. SIP signalling is used so that different applications can agree on how to communicate. RTP carries the speech frames over the network, in order to provide features that are necessary for media/multimedia applications. Issues like perception of audio and QoS related parameters is also discussed, from the perspective of users and developers.

Keywords: VoIP, QoS, AMR, audio, RTP, RTCP, SIP, UMTS, 3G, multimedia.
Abstract

The interest for various multimedia services accessed via the Internet has been growing immensely along with the bandwidth available. A similar development has emerged in the 3G mobile network. The focus of this master thesis is on the speech/audio part of a 3G multimedia application. The purpose has been to implement a traffic generating tool that can measure QoS (Quality of Service) in 3G networks. The application is compliant to the 3G standards, i.e. it uses AMR (Adaptive Multi Rate), SIP (Session Initiation Protocol) and RTP (Real Time Transport Protocol). AMR is a speech compression algorithm with the special feature that it can compress speech into several different bit-rates. SIP signalling is used so that different applications can agree on how to communicate. RTP carries the speech frames over the network, in order to provide features that are necessary for media/multimedia applications. Issues like perception of audio and QoS related parameters is also discussed, from the perspective of users and developers.

Keywords: VoIP, QoS, AMR, audio, RTP, RTCP, SIP, UMTS, 3G, multimedia.
Acknowledgment

To begin with, we want to show our appreciation to Mikael Gustafsson, Per Norlander and Martin Rantzer for making this master thesis project possible and for spending your valuable time on us. Especially we want thank Mikael for sharing his knowledge in project management and Per for reassuring that the project was going to be finished even though LINLAB was shutting down; it was a great comfort.

We also want thank our supervisors and examiner for their engagement and support on various levels. Frida Gunnarsson, thank you for giving us good comments in a rapid speed, it really helped us to improve our essay. Jonas Olsson, your guidance and support in several issues especially the theoretical questions have been invaluable. Per Elmdahl, without your assistance in setting up our system we would not have been able to finish in time. Frans Unosson, we want to thank you for helping structuring our code and for being supportive in every possible way (our visit at your mansion was real nice). Mattias Rönnblom we have never met anybody with so good knowledge in programming as you have, thank you for helping us out with some tricky stuff. Tomas Bigun, without you the time spent in the office would not have been as enjoyable as it was, we also want to thank you for contributing with your expertise in some programming matters. Fredrik Gunnarsson, thanks for your criticism and for reading the essay in so short time before your journey.

We also want to thank Magnus Westerlund at Ericsson Research in Stockholm for giving our questions good answers, fast!

To finish with, we want to thank TietoEnator, Ericsson and the people that work there for making us feel welcome; it has really been a good period in both our lives.

Kristian Nilsson and Kenth Fredholm

To succeed in the world it is not enough to be stupid, you must also be well-mannered.
--Voltaire [Francois-Marie Arouet] (1694 - 1778)
Notation

Abbreviations

3GPP 3rd Generation Partnership Project.
AMR Adaptive Multi-Rate.
BFI Bad Frame Indication from Access Network.
CRC Cyclic redundancy check.
CS Circuit Switched.
CSCF Call Session Control Function.
CSRC Contributing Source Count.
DFI Degraded Frame Indication.
ETSI European Telecommunications Standards Institute.
GPRS General Packet Radio Service.
GUI Graphical User Interface.
IANA Internet Assigned Numbers Authority.
IETF Internet Engineering Task Force.
MR–ACELP Multi–Rate Algebraic Code Excited Linear Prediction.
MT Mobile Termination.
NTP Network Time Protocol.
PCM Pulse Coded Modulated.
PS Packet Switched.
QoS Quality of Service.
RTCP Real–Time Transport Control Protocol.
RTP Real–Time Transport Protocol.
SCR Source Controlled Rate operation.
SDP Session Description Protocol.
SGSN Serving GPRS Supporting Node.
SID Silence Descriptor.
SIP Session Initiation Protocol.
SSRC Synchronizations source.
UDP User Datagram Protocol.
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMTS</td>
<td>Universal Mobile Telecommunication System.</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Indicator.</td>
</tr>
<tr>
<td>VAD</td>
<td>Voice Activity Detection.</td>
</tr>
<tr>
<td>VoIP</td>
<td>Voice over IP.</td>
</tr>
</tbody>
</table>
# Contents

1 Introduction .............................................. 1
   1.1 Background .......................................... 1
   1.2 Idea ................................................. 2
   1.3 Objectives ........................................... 2
   1.4 Delimitations ......................................... 3
   1.5 Disposition ........................................... 3

2 Theoretical Background ................................. 5
   2.1 Introduction ......................................... 5
   2.2 Audio Programming ................................... 5
   2.3 Socket Programming .................................. 7
   2.4 Adaptive Multi–Rate (AMR) ............................. 7
       Introduction ........................................ 7
       General Principles .................................. 8
       Error Concealment of Lost Frames ................. 10
       Source Controlled Rate Operation ................ 10
   2.5 Real–Time Transport Protocol (RTP) ................. 12
       Introduction ........................................ 12
       RTP Header ......................................... 13
       Real–Time Transport Control Protocol (RTCP) .... 14
   2.6 Quality of Service (QoS) for Real Time Audio .... 17
   2.7 Session Initiation Protocol (SIP) ..................... 17
       Introduction ........................................ 17
       Session Setup using SIP ........................... 17
       SIP Messages ........................................ 20

3 The Application .......................................... 25
   3.1 Introduction ......................................... 25
   3.2 Network Overview .................................... 25
   3.3 Structure .............................................. 26
       Application Startup ................................ 26
       Media Session ...................................... 30
       Control Information from RTCP .................. 36
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4 Perceptual Sound Quality</td>
<td>36</td>
</tr>
<tr>
<td>4 Conclusions and Discussion</td>
<td>37</td>
</tr>
<tr>
<td>4.1 Results</td>
<td>37</td>
</tr>
<tr>
<td>4.2 Implementation Difficulties</td>
<td>37</td>
</tr>
<tr>
<td>4.3 Areas of Usage</td>
<td>38</td>
</tr>
<tr>
<td>4.4 Possible Extensions and Improvements</td>
<td>38</td>
</tr>
<tr>
<td>A AMR Coding Scheme</td>
<td>41</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Background

The interest for various multimedia services accessed via the Internet has been growing immensely along with the bandwidth available. In a near future it will be possible to get enough amount of bandwidth using a cellular phone to be able to use multimedia services similar to those available on the Internet. This fact, among others, has led to research and development of how multimedia shall be handled in the 3G network. The focus in this thesis is on the speech/audio part of a multimedia application.

In general, when one implements an application, it is of most importance that it follows the rules and specifications that is specified for that kind of applications. Otherwise the application will be totally meaningless, especially if it is supposed to interact with its environment. A speech/audio application is no exception. In order to be able to communicate with its environment it has to follow the chosen standards. The mandatory standards, for speech/audio applications in 3G networks, upon which our application is based are presented here:

3GPP (3rd Generation Partnership Project) is a collaboration agreement that brings together telecommunications standards. They have chosen Adaptive Multi–Rate (AMR) as the standard codec for compression of audio in the circuit–switched and packet–switched domains in the 3G network. In the circuit–switched domain all data has the same predefined route from source to destination. In the packet–switched domain the data is packed in packets which are routed individually from source to destination.

Also, the Session Initiation Protocol (SIP) is chosen be used for setting up a connection between two or more communicating parties, in the packet–switched domain. To carry the compressed speech between the participants a transport protocol called RTP (Real–Time Transport Protocol) is chosen. This protocol is, among other things, used to obtain the short time delays that oral communication between humans require.
Ericsson (LINLAB, Linköping) has since March 2001 worked on the End–to–end Quality of Service\(^1\) Testbed (E2E QoS testbed) project, which explores solutions for making end–to–end quality of service work in an envisioned 3G network. This master thesis is an important part of this project.

These areas of research are important both for Ericsson as the leading 3G radio network system developer, and for TietoEnator Wireless R&D (Research and Development) as the main supplier of telecom R&D services in the Nordic market. Ericsson and TietoEnator has decided to finance and mentor this masters thesis work cooperatively, enabling the masters thesis students to benefit from both companies organisations, expertise and strengths in diverse areas.

1.2 Idea

In the development of 3G it is of most importance to investigate how different parts in the 3G network should work without having to build them for real and in full scale. In order to achieve this artificial environment, Ericsson has created a QoS (Quality of Service) testbed. One important part in using this testbed is to be able to generate traffic load to the testbed. When emulating 3G traffic in the testbed, it is necessary that the traffic is identical with the traffic that will be used in the future 3G network. No such traffic generating tool or measuring equipment existed within Ericsson before this master thesis project started.

The purpose of this master thesis project has been to implement a “3G phone” for Ericsson to use for measuring QoS in emulated 3G networks. It can also be used for internal and external demonstrations and it can form a basis for further prototype development (e.g., 3GPP PS (3G Partnership Project)(Packet Switched) conversational multimedia and 3GPP PS streaming applications), etc. Since this master thesis is carried out in conjunction between TietoEnator and Ericsson one important part has been to enhance TietoEnator’s knowledge within this area.

The purpose for us is to enhance our knowledge in this area and to get experience from working in industrial life.

1.3 Objectives

The objectives of the master thesis project are listed below:

- Deliver a fully operational AMR phone implementation for measurement of QoS in 3G networks.
- The application will be able to handle automatic testing over a QoS testbed.
- The application will support both subjective and objective (=statistics logging) tests.
- The application will be compliant to the 3GPP standardization.

\(^1\)QoS can have many different meanings; in this case the QoS can be seen as the perceptual audio quality in conversations between humans.
1.4 Delimitations

The original implementation only handles speech, not multi-media applications like music and video. Only AMR and not AMR–WB (Adaptive Multi Rate – Wide Band) is explored. Neither is the original implementation delivered with a Graphic User Interface (GUI). The users interface is text based and operated from an elective “Linux terminal window”.

Optimizing the code have not been a priority. An old saying reads: first make it work, then make it right, then make it fast. We have accomplished at least the two first phrases and maybe lightly touched the third.

The sound quality has only been analyzed manually by hearing, but the application forms a basis for more advanced quality analysis.

To clarify what we mean with QoS in the scope of this thesis we must say that the application is not a tool that can be used to provide QoS (though there are several possible settings available that affects the quality of the audio). It is mainly a tool that can be used to measure it.

The application is implemented as a software running on a Linux/UNIX platform, thereby it is said that it is not a cellular phone. However, in the future, a cellular phone will contain an application similar to ours.

1.5 Disposition

Chapter 1 gives the background and idea behind this master thesis project. The objectives and some delimitations are also presented here.

Chapter 2 contains a survey of the theoretical foundation on which the application lies. It starts with basic knowledge on audio and socket programming and continues with a more thorough presentation of the AMR (Adaptive Multi Rate) codec and the two protocols RTP (Real Time Transport Protocol) and SIP (Session Initiation Protocol).

The application is described in detail in chapter 3. First an introduction and a network overview is presented. Thereafter the application structure is discussed and the functionalities of main parts in the application are described.

Chapter 4 contains the conclusions and different aspects of the application, such as areas of usage, is discussed.
Chapter 2

Theoretical Background

2.1 Introduction

The purpose with this chapter is to give the reader an opportunity acquire the knowledge needed to fully understand our application, that will be described in chapter 3. The speech compression algorithm (AMR) and the connection and transport protocols (SIP and RTP) will be discussed here. But first an introduction, on how the speech/audio is handled by the computer soundcard and the network device, is given in the sections audio and socket programming.

2.2 Audio Programming

When doing an application that somehow involves digital audio and using the sound card one have to consider the possibility to do specific settings of the sound card in order to get the wanted result. It is important to explicitly set all parameters the application depend on. There are default values for all parameters, but it is most likely so that other (future) devices do not support them.

The most fundamental parameter to be set is the sampling rate. The sampling rate limits the highest frequency that can be recorded, according to the Nyquist Sampling Theorem. The highest frequency that can be recorded when sampling a signal is at most half of the sampling frequency.

To decide the dynamic range of the recorded signal, the sample encoding i.e., how many bits each sample contains, has to be set. The dynamic range is the difference between the faintest and the loudest signal that can be recorded. In theory the maximum value of the dynamic range is obtained by \( \text{number of bits per sample} \times 6 \) dB.

The number of channels (mono or stereo) is also a fundamental parameter that has to be set. On some sound cards the number of channels cannot be set, they can therefore only record in either mono or stereo. To solve this problem, if it is a problem, the application has to contain an algorithm that either duplicates or
decimates the signal.

It is of most importance to always set sampling parameters so that the number of channels (mono or stereo) is set before the sampling rate. Failing to do this can result in wrong sample rate. If the application first sets sampling rate to 44.1 kHz and then sets the device to stereo mode the sampling rate can be decreased to 22.05 kHz\(^1\) (while the programmer still believes that the device is in 44.1 kHz mode.)

When the fundamental parameters described above have been decided and set there are others somewhat more tricky parts that have to be considered. The parts that will be handled here involves setting the device to full duplex and configuring the buffer. Such operations are sometimes referred to as advanced audio programming. However they do not automatically make your application better but are necessary in some cases. They should not be used when they are not necessary, because they make the application less compatible to other devices and operating systems.

Full duplex means that the audio device is capable of handling input and output in parallel. To have an application running in full duplex mode is not complicated in theory but in practice it is not simple. There are a number things that must be considered. A device can either be half duplex, full duplex or both. If the device is half duplex it is hard to implement applications which do both playback and recording simultaneously. If the device is full duplex the application does not need to turn on full duplex, it is always on. When running in full duplex the application should be able to handle both input and output simultaneously without blocking on writes or reads (all audio read and writes are nonblocking as long there are enough space in the DMA (Direct Memory Access) buffer, see below, when the application makes the call). Usually this means that the application must use synchronization methods. When using a device that handles both, the application must turn on full duplex. This must be done immediately after opening the device.

It is important to know that full duplex does not mean that different applications can access the same device (output or input) simultaneously.

Most applications “talk” with the audio device via a kernel DMA (Direct Memory Access) buffer. The method used is often some kind of multi buffering, see below. There are two or more buffers, one is being accessed by the device and another by the application. The fact that the device processes the buffers in turn gives the application time to do some processing at the same time. This is a requirement to be able to record and playback without pausing.

When configuring the buffer it is an advantage to use so called multi buffering (if available). This method increases the control of how the application reads from and writes to the DMA buffer. It enables the possibility to divide the buffer into fragments. Since the time the device spends processing the buffer half depends on the buffer size and half on the data rate, multi buffering gives the opportunity to be more precise in determining the device process time. This is e.g. necessary to avoid latency when the application needs a quick response from the device.

\(^1\)many older sound cards support 44.1 kHz in mono but only 22.05 in stereo
2.3 Socket Programming

To be able to transmit and receive information between networked computers, some sort of connection has to be established. In practice it would be difficult to send information on a network since the networking hardware and communicating software is difficult to develop by every person (who wants to create a simple application which transmit something). But this is already taken care of and simplified by the operating system for the users convenience. The resulting function to establish a connection is hereby called socket.

A socket is a bidirectional pipe for incoming and outgoing data between networked computers. On each host, it is likely that more than a single application will be running at a given time. This explains the use of ports, because each application can be associated with a unique port number. To reduce the potential for confusion, many popular applications are assigned well-known port numbers by the Internet Assigned Numbers Authority (IANA).

When creating a socket, the type of socket must be specified. The are different options in the addressing format, the common schemes are AF_UNIX and AF_INET. The AF_UNIX type is a UNIX pathname for identification of sockets, this type is very useful for communication between processes on the same machine.

The type AF_INET uses the Internet address range, which is separated in two versions. There are two standards, these are called IPV4 and IPV6. IPV4 is most commonly used in the Internet, but IPV6 is the newer standard and it has a larger address range to satisfy the upcoming amount of users. The port numbers is used to allow each machine to have multiple AF_INET sockets.

There are also other options which must be set when creating sockets. Two types of AF_INET sockets exists, SOCK_STREAM and SOCK_DGRAM. SOCK_STREAM is used to stream data via the TCP protocol (with automatic error correction and retransmission) and SOCK_DGRAM is used to send data in packets via the UDP protocol (without any error control). This is why real time applications (like ours) must use SOCK_DGRAM since error correction and retransmission will convey unacceptable time delays.

2.4 Adaptive Multi–Rate (AMR)

Introduction

AMR was originally developed to be used in GSM by the European Telecommunications Standards Institute (ETSI). In early 1999, 3GPP standardisation approved AMR as mandatory codec for circuit– and packet–switched speech in 3G–networks. The purpose with AMR is to have speech codec that can be set to different bit rates. AMR can operate at 8 distinct bit rates: 12.2; 10.2; 7.95; 7.4; 6.7; 5.9; 5.15; or 4.75 kbit/s. It also contains a low rate encoding mode, called Silence Descriptor (SID) which operates at 1.80 kbit/s\(^2\), to produce background noise and a

\(^2\)Assuming SID frames are continuously transmitted
NO_TRANSMISSION mode. The SID frames contain a set of parameters that is used to describe the background noise that is heard during a conversation, e.g. when one is in a car the background noise is significant. How the SID frames are used is discussed in detail in subsection Source controlled rate operation.

The codec is designed to allow seamless switching between different modes from one frame to another. Therefore is an application that uses the codec able to do a trade–off between quality and capacity as function of the network load.

Since each AMR speech frame consist of 160 encoded speech samples (13–bit uniform Pulse Coded Modulated (PCM)) and the sample rate is 8 kHz, the duration of each frame is 20 ms. This means that the codec can switch mode, i.e. bit rate, every 20 ms.

In the next section the AMR general principles will be discussed. No speech coding is discussed, though it might be relevant here. For those who are interested please check out appendix A.

The facts in the above section is taken from [3].

General Principles

Audio In and Out

The speech encoder takes as its input a 13–bit uniform Pulse Coded Modulated (PCM) signal. This signal is a result of analogue–to–digital conversion of the speech input. This is either done by direct conversion to 13–bit PCM format or by conversion to 8–bit A–law or µ–law compounded format, followed by 8–bit to 13–bit conversion.

The output from the decoder is converted from 13–bit/8 kHz uniform PCM to analogue. The digital–to–analogue conversion is done by inverting either of the two analogue–to–digital conversion methods.

Just before the encoding process, the input speech is down–scaled and high-pass filtered. The high–pass filter has a cut off frequency of 80 Hz and is used to keep unwanted low frequency components away. At the output of the speech decoder the speech is high–pass filtered and up–scaled. The filter has a cut–off frequency off 60 Hz and its purpose is the same as the purpose of filter on the encoder side. The down–scaling on the encoder side is to reduce the possibility of overflows, the up–scaling on the decoder side is to compensate for the down–scaling.

Those who wants to know more about how AMR handles inputs and outputs of audio can find the specifications [3] and [4] interesting.

Frame Structure

The generic structure of the AMR frame is depicted in Figure 2.1. As the figure shows, the frame is divided into a header, auxiliary information and core frame.

3A–law and µ–law is the two most common coding techniques used as input for speech coding algorithms. To preserve the quality of the speech, typically 12 or 13 bit linear quantization is required but due to logarithmic quantization these techniques only uses 8 bits and still retain the same quality.
2.4 Adaptive Multi–Rate (AMR)

The AMR header contains the Frame Type and Frame Quality indicator fields. The Frame Type is used to indicate the quality of the data carried in the AMR Core Frame. If it is a speech frame the Frame Type contains the AMR codec mode for that particular frame otherwise it indicates a SID frame or a NO DATA frame. The Frame Quality Indicator indicates if the frame is good or bad.

The AMR Auxiliary Information includes the Mode Indication, Mode Request and Codec CRC fields. The CRC (Cyclic redundancy check) field in the AMR Auxiliary Information is used in the purpose of error–detection. The 8 bits contained in this field is parity check bits generated by a cyclic generator polynomial which is computed over all Class A bits in the AMR Core frame.

![Figure 2.1. Generic AMR frame structure](c⃝ETSI 2002. Further use, modification, redistribution is strictly prohibited. ETSI standards are available from http://pda.etsi.org/pda/ and http://www.etsi.org/eds/)

The AMR Core Frame field is used to carry the encoded speech bits divided into the A, B and C classes. In the case of comfort noise, the Class A field contains the comfort noise parameters i.e. a SID frame, Class B and C are not used.

To be able to provide error protection when, for example, the speech bits are carried over a radio interface, bit ordering is used. The speech bits are packed in the AMR Core Frame in an order corresponding to their subjective importance. After this the speech bits are divided into the three classes A, B and C, where Class A contains the most important speech bits i.e. the bits that are most sensitive to errors. The reason for dividing the speech bits into classes is that they then can be subjected to different error protection in the network. Since the Class A bits are the most important bits, an error in any of this bits will result in that the frame will be considered to be corrupted and will therefore not be decoded without applying

---

4Comfort noise is usually the background noise that is transmitted along with the speech or alone if there is no speech. Without this background noise the participants, in a conversation, might think that their connection is broken.
appropriate error concealment (error concealment is discussed below). Therefore are the Class A bits protected with Codec CRC to detect errors. There is no step-wise change in importance between the speech bits in the different classes, the speech bits lose importance continuously from Class A to C (from speech bit to speech bit within each class). When errors occur in Class B or C the speech quality is gradually reduced according to the error rate and the importance of the erroneous speech bits. Decoding of a erroneous speech frame is usually possible without any annoying artifacts.

The number of speech bits in each class is dependent on which codec mode that is being used. It is not always that all the three classes is being used, for example when the AMR codec mode is 4.75 no speech bits is packed as Class C bits.

For further reading, [5] is recommended.

Error Concealment of Lost Frames

Error concealment in AMR is used to conceal the effect of lost AMR speech or SID frames, or if error(s) is detected in the Class A bits. If several frames is lost AMR will mute the output. This is to avoid the error concealment procedure to generate annoying sounds and to indicate the breakdown of the channel to the user.

It is the network that is responsible for indicating the different kind of errors. This is done by setting flags in the AMR frame, more specific, by setting the received Frame Type. The decoder can receive nine different Frame Type modes, indicating the quality of the received frame. If, for example, a speech frame or a SID frame is lost the received Frame Type shall have the values SPEECH_BAD or SID_BAD. These flags tell the speech decoder to perform parameter substitution to conceal errors. In both of these cases the BFI (Bad Frame Indication from Access Network) flag is set. If the frame is not lost but distorted the received Frame Type shall contain the value SPEECH_DEGRADED. In this case the DFI (Degraded Frame Indication) flag is set.

If a lost speech frame would be decoded it normally generates unpleasant sounds. This inconvenience is prevented by letting the coder substitute lost frames with either repetition or extrapolation of the previous good speech frame(s). The substitution decreases the output level gradually, resulting in silence at the output. If the frame is not lost but degraded, the corrupted data can be used to assist error concealment. A lost SID frame is substituted using the information from previously received valid SID frame(s). For many subsequent lost SID frames the output, the comfort noise, will be gradually silenced.


Source Controlled Rate Operation

In order to lower the average bit rate and to save power in the User Equipment the Source Controlled Rate operation (SCR) is used. SCR for AMR in UMTS systems is mandatory in all UMTS equipment. Basically the SCR accomplishes its
2.4 Adaptive Multi-Rate (AMR)

purpose by detecting the speech inactivity during a conversation to regulate the data transmission. When nothing is said nothing should be encoded and sent.

To detect inactivity in the speech, Voice Activity Detection (VAD) (see below) is used on the transmitting (TX) side. A problem when using SCR is that if the transmission is switched off when speech inactivity is detected the acoustic background noise, that is transmitted along with the speech, is also switched off. This is very annoying for the listener, especially when these switches between active and inactive mode take place rapidly. The listener cannot know whether the connection is down or if it is just temporarily silence. The introduction of comfort noise generation on the receiver (RX) side has solved this problem. The background noise on the TX side is evaluated and the characteristic parameters is packed in a SID (Silence Descriptor) frame and sent to the RX side at a regular rate when the transmission of speech data is switched off. In this way the comfort noise generated on the RX will be updated for the changes in the background noise on the TX side.

If a SID frame is seriously corrupted, at the RX side, the SID frame should be substituted by using information from previously received SID frames. This to avoid unpleasant effects for the listener.

![Diagram of VAD flag and SID frames]

**Figure 2.2.** Hangover procedure for AMR © ETSI 2003. Further use, modification, redistribution is strictly prohibited. ETSI standards are available from [http://pda.etsi.org/pda/] and [http://www.etsi.org/eds/]

The first seven frames after that the encoder has switched from active to inactive mode (active mode is when speech is detected on TX side) is always marked as GOOD_SPEECH i.e. a speech frame. This is done even though the VAD flag is
set to “0” which indicates that no speech is detected. The period of these seven frames (7 * 20 ms) is called a hangover period (see Figure 2.2) and is used by the decoder on the RX side to compute a SID frame. Therefore no information about the background sound is sent in the first SID (SID\_FIRST) after active speech. This information is sent, for the first time, in the third frame after the SID\_FIRST frame in a SID\_UPDATE frame. However, if the coder switches rapidly to active mode and then back again to inactive mode no hangover period is used. Then the last SID\_UPDATE frame that was computed should be passed to the network. This is to reduce the network load.

It is recommended to read [6], [2] and [1] for more profound knowledge about AMR source controlled rate.

**Voice Activity Detection (VAD)**

The VAD algorithm is used to detect if each 20 ms frame contains speech or not. The input signal is the presumptive speech along with a set of parameters computed by the AMR speech encoder. The output of the VAD is a Boolean flag (VAD flag) indicating the presence of speech signals.

The synthesis done in the speech decoder is different whether the received frame contains normal speech or not.

The comfort noise generation process is as follows:

- the evaluation of the acoustic background noise in the transmitter
- the noise parameter encoding (SID) and decoding
- the generation of comfort noise in the receiver

The VAD algorithm will not be discussed further here, the interested reader is directed to [3].

### 2.5 Real–Time Transport Protocol (RTP)

**Introduction**

When the usage of multimedia applications started, such as audio- and videoconferencing, each application had its own transport protocol. But since most of these applications had the same requirements, it finally led to the development of the Real–Time Transport Protocol (RTP). RTP became the universal protocol for use by multimedia applications. It is developed by IETF\(^6\).

When different protocols are discussed it is common to place them in different layers such as application, transport, network and physical layer, see Figure 2.3.

\(^5\)The SID\_FIRST frame is always sent at the end of a talk spurt and is used to initiate the generation of comfort noise on the RX side.

\(^6\)Internet Engineering Task Force
The application layer is the interface to the user. The transport layer contains protocols like TCP and UDP (RTP is also seen as a transport layer protocol). This layer typically handles end-to-end communication between applications over a network. The well known IP protocol is located in the network layer that handles addressing and routing. The physical layer is where the raw data bits are transmitted over a physical cable.

It is not totally obvious why RTP is called a transport protocol. Since a not insignificant part of the functionalities that is specific to multimedia applications is implemented in RTP. Another reason is that it runs on top of transport-layer protocols such as UDP. It is therefore close at hand to call it an application-layer protocol. However since its primal purpose is to provide end-to-end network functions for data with real-time characteristics it is called a transport protocol.

RTP is implemented to run over many different lower-layer protocols but it typically runs over UDP. UDP has the, for real-time applications, favorable characteristic that no retransmission of lost data is done. This is necessary for an application that is dependent on short packet delays, if a packet is lost and then retransmitted it most certainly will arrive too late to be useful. The RTP protocol stack is usually used in multimedia applications as depicted in Figure 2.3.

<table>
<thead>
<tr>
<th>Application</th>
<th>RTP</th>
<th>UDP</th>
<th>IP</th>
</tr>
</thead>
</table>

Figure 2.3. Application protocol stack

It is important to notice that RTP does not provide any functions that guarantees quality-of-service (QoS) which may be implied in the name. However it provides information (by using RTCP, see below) that makes it easy for applications to manage its QoS requirements. RTP provides sequence numbers for each RTP packet but does not guarantee delivery or that the packets are delivered in correct order. The sequence numbers is used to sort the RTP packets on the receiver side and to locate the proper location for a packet without decoding the packets in a sequence.

The RTP standard consists of two protocols working together. Besides the RTP protocol there is the Real-Time Transport Control Protocol (RTCP). The main task for RTCP is to monitor the quality of service in an on-going session.

RTP Header

The RTP header format is depicted in Figure 2.4. It is important that the header has been designed to be as short as possible. The reason for this is that audio
packets, which are a very common types of multimedia data, is kept short to avoid latency due to packetization. If the packets were short and the headers were long the transmission would become very bandwidth inefficient. The RTP header is always at least 12 bytes long, the first three rows in Figure 2.4. The rest is optional, as described below.

The first two bits in the header identifies the current version of the RTP protocol. The next bit is used to indicate if padding is employed. If the padding bit, P, is set the payload ends with one or more padding octets which are not part of the payload. The reason for using padding is application dependent (see below), however it is commonly needed by encryption algorithms. The next bit, X, is set when the fixed header is followed by exactly one header extension. The extension header is seldom used, but it is meant to be used by applications that need to put additional information in the header. The four bits in the Contributing Source (CSRC) count, CC, contains the number of CSRC identifiers (for CSRC see below).

The purpose of the markerbit, M, is dependent on the payload format carried in the RTP packet; how it is used for AMR is described in section 3.3 subsection Packing AMR frames into RTP packets. The payload format, e.g. AMR, is identified in the payload type field, PT. The 16 bits that follows contains the sequence number. The sequence number is incremented by one for each RTP packet that is sent. The receiver side can use it to detect packet loss or if a packet is received in wrong order. If RTP detects a lost packet it is then up to the application to take action. This is in accordance with the principle by which RTP is created, namely that each application understands its own needs best.

The timestamp, that follows next, corresponds to the sample instant of the first bit in the RTP payload. The timestamp is used on the receiver side to enable that the samples are played back in correct intervals and to synchronize different media streams, therefore it has to be incremented monotonically and linear in time. How much the timestamp is incremented is not defined in RTP, it is dependent on the format carried in the payload. The resolution of the clock, that clocks the timestamp, must be sufficient to enable the receiver to play back the samples in appropriate intervals and achieve desired synchronization accuracy.

To ensure that every synchronization source of an RTP stream, within the same RTP session, is uniquely identified the synchronizations source (SSRC) identifiers field is used. It contains a random number which identifies the source. The contributing source (CSRC) is used when RTP are passed through a mixer.

Most of the information about RTP is collected from [10] and [8]. Those who are eager to learn more is strongly recommended to read, especially, [10].

Real–Time Transport Control Protocol (RTCP)

The primary function of Real–Time Transport Control Protocol (RTCP) is to provide feedback on the performance of the data transmission handled by RTP. This

---

7 A microphone or a web camera are examples of different possible sources.

8 A mixer reduces the bandwidth requirements by receiving data from many sources and sending it as a single stream.
function is particularly useful for applications designed to adapt their data rate according to network conditions. For example an application using the AMR codec can compress the data more to avoid congestion, or less if there is little congestion to enhance the quality. The feedback, given from RTCP, can also be used to analyze network problems. If IP multicasting is used, a third–party that is not otherwise involved in the session such as a network service provider, can receive the feedback given from all the participants in the session and use it to diagnose network problems. The feedback is given in so called RTCP sender and receiver reports. These reports will be discussed below.

RTCP also provides an identifier for each RTP source called canonical name (CNAME). The canonical name often looks like an email address e.g. “rubber-duck@tulsa.18wheelers.com”. It may seem that this should not be necessary since the SSRC identifier in the RTP header is used to identify a source. But the value of the SSRC identifiers can collide and therefore the receiver needs the CNAME to keep track of each participant. Every receiver needs to receive the CNAME as soon as possible in order to identify the source and start synchronizing operations such as lip–sync.

For both of these functions, the sender/receiver reports and the canonical name, it is a must for each participant to send RTCP packets. It is obvious that this control traffic can consume significant amount of bandwidth. For example, an audioconference can have several hundreds participants, only two or three of these are talking at the same time and therefore consuming bandwidth sending audio packets. However, all participants are sending the RTCP control packets. This traffic must somehow be regulated to keep down the bandwidth consumed. The problem is solved by letting every participant send its control packets to all the others and thereby each participant finds out the total number of participants in the session. The number is often an approximation but is sufficient information for every participant to calculate the rate at which they should send the control

---

9Synchronization between speech, received from an audio source, and lip movements, received from a video source.
packets. The goal is to limit the total amount of RTCP traffic to approx. 5 percentage of the RTP data traffic. The active senders sends at a higher rate since their reports are the most interesting.

Sender and Receiver Reports

As mentioned above the sender report (SR) and the receiver report (RR) gives feedback on the network performance. The information given in the SR and the RR is similar, the only thing that differs is that information about the sender is omitted in the RR. Not all the information in the SR packet will be discussed here, only some parts that relates to QoS monitoring (which, by the way, the major part does). The following information, given in different fields in the SR packet, can be used to monitor QoS:

Network Time Protocol (NTP) timestamp: indicates the wallclock time when a report is sent.

Fraction lost: contains the number of RTP data packets lost, since the last SR or RR packet was sent, divided with the expected number of packets.

Cumulative number of packets lost: indicates the total number of RTP data packets lost from a certain source since the beginning of reception. The number is defined as the difference between the total number of packets expected and the total number of packets actually received.

Interarrival jitter: contains an estimate of the statistical variance of the RTP data packet interarrival time. This is the same as the difference in the relative transit time between the two packets that are compared.

The NTP timestamp can, in combination with the timestamps given in the reception reports from receivers, be used to calculate the round-trip time to those receivers.

Both SR and RR reports contain the cumulative number of packets lost. Thereby, can the difference be calculated between any two reports. In this way both short and long term quality measurements can be accomplished.

Just like the fraction lost the interarrival jitter are a measure of the short term network congestion. The fraction lost is a measure of the persistent congestion while the jitter tracks the transient congestion. The jitter may indicate congestion before packet loss occur.

The SR also contains the RTP timestamp (this is the same kind of timestamp as in the RTP data packets). This corresponds to the same time as the NTP timestamp. Together they can be used to synchronize different media streams from the same source since they give the key to convert wallclock time to the RTP timestamps.

Further information about RTCP can be found in [10] and [8].
2.6 Quality of Service (QoS) for Real Time Audio

The End–to–End (E2E) Quality of Service (QoS) requirements for real time audio is very strict. Studies have shown that the human ear is far more sensitive to erroneous audio than the eye is to errors in video. The error rate must be under 5 percent during a conversation to be acceptable.

The aim for a Voice over IP (VoIP) application is to provide a voice quality that is at least as good as the quality in a regular phone, at a lower cost. Since a VoIP application is used for conversation between humans, the perceptual audio quality is used to measure the QoS provided. The voice delay, rate variation and voice lost are the measurable quality parameters. As explained above, these parameters are all carried in the RTCP sender and receiver reports, it is then up to the application to analyze these parameters to get a measure of the QoS.

This is just a brief introduction to QoS for real time audio; the facts are picked from [13] and [12].

2.7 Session Initiation Protocol (SIP)

Introduction

The Session Initiation Protocol (SIP) is used to create and terminate sessions. SIP has become the dominant protocol for establishing calls like Voice over IP, video conferencing and instant messaging. But it is becoming the dominant protocol for establishing calls of any kind. It is intended to be used as a control part for other connections. This means that it does not transfer any data payload between the communicating entities, only information about how they should communicate with each other. Often both sides support many different protocols and modes. SIP solves the communication problem by negotiating a protocol and a mode that both sides can use. With this information given they can start their session and transmit data to each other. Usually when the connection is up between the machines, the SIP signalling is silent. But if anything needs to be changed during the session additional SIP communication can be employed.

When the communicating participants are finished with their conversation, SIP signalling is used again to close their connections gracefully.

The Session Initiation Protocol can also be expanded to get the benefits from other protocols. The most usual case is that SIP lives in symbiosis with SDP (Session Description Protocol) to be able to transmit and receive specific network or application related information. The SIP protocol has also many features that will come in handy in a few years. An example of this is the possibility to implement payment services.

Session Setup using SIP

This section will describe more verbose how the session is started. To describe this better let us use two participants which we call Rubberduck and Spider Mike.
In the first scenario Spider Mike wants to call Rubberduck on their IP-phones. Spider Mike also have the information about exactly where Rubberduck is, so he inputs “rubberduck@18wheelers.com” in his phone and presses the call button. First Spider Mike sends an INVITE message, with his phone, directly to Rubberduck by using the IP-address or DNS (Domain Name Server) name (e.g. 18wheelers.com) see Figure 2.5. When Rubberducks phone receives the INVITE message, it starts to process it and responds with a TRYING message. When the INVITE message is processed the phone starts ringing to catch Rubberducks attention and responds with a RINGING message to Spider Mike’s phone. After a while when Rubberduck lifts his receiver, the phone transmits an OK message to Spider Mike and the session is established so they can talk to each other.

Later on, they are finished with their discussion and Rubberduck hangs up. His phone then sends a BYE message to Spider Mike’s phone and it responds back with an OK message. With this transaction the session is closed.

Some messages like TRYING probably seems unnecessary, but it is easier to understand if we use a little more complex picture like Figure 2.6.

In the second scenario we introduce the Proxy servers. These are used to help route requests to the correct location, authenticate and authorize users for e.g. different services. In the figure it is easy to see that it might take a while before an INVITE message reaches its destination. If an INVITE message is lost somewhere
on the way to the destination, it is easy to locate where it was lost since the previous instance did not receive any TRYING message. If some instance does not receive an expected TRYING, then only that instance needs to resend the original message. Otherwise, the originator would have had to send a new message. This is important, because if many messages, that traverses many proxy servers, has to be retransmitted from the originator it consumes a lot of bandwidth.

In the third scenario, described in Figure 2.7, we use a service called Registrar server. The main idea with Registrar servers is to be able to contact machines somehow closer to the target. When Rubberduck puts his phone or UA (User Agent) online, it registers itself in a registrar server maybe under the name “rubberduck@tulsa.18wheelers.com”. This registrar server stores this name in a location service server, which might contain a registry for all people under the domain name “18wheelers.com”. Lets say that Spider Mike wants to call Rubberduck again. Spider Mike does not know or care that Rubberduck is in the town of Tulsa, he just wants to talk to him. In this case Spider Mike inputs “rubberduck@18wheelers.com” in his phone. Now his phone just sends an INVITE message to “rubberduck@18wheelers.com”. This message delivered to a proxy server which
has been placed under the address “18wheelers.com”. This proxy then contacts the location service server and responds with the exact location where Rubberduck can be found, namely “tulsa.18wheelers.com”. The proxy now have information about where to forward the original INVITE message, so it changes the address to what it received from the location service and forwards it.

Note that the information in the location service can be either inputed administratively or created from user agents sending REGISTER messages. REGISTER requests can add, remove and query bindings in the location service.

**SIP Messages**

SIP messages are always in clear text. This implies that the Session Initiation Protocol cannot be independent from other protocols. Usually, on the Internet, SIP messages are contained as the payload in UDP (User Datagram Protocol).

An INVITE message might contain the following fields:

```
INVITE sip:rubberduck@18wheelers.com SIP/2.0
Via: SIP/2.0/UDP traders.com:5060
To: Mr. Rubberduck <sip:rubberduck@18wheelers.com>
```
From: Mr. Spider Mike <sip:spider_mike@trucks.traders.com>
Call-ID: 123456789@traders.com
Contact: sip:spider_mike@trucks.traders.com
CSeq: 1 INVITE
Subject: Drive faster!
Content-Type: application/sdp
Content-Length: 208

v=0
o=spider_mike 17450917453215 17450917453215 IN IP4 traders.com
s=Phone Call
c=IN IP4 10.20.30.40
t=0 0
m=audio 49170 RTP/AVP 0 6 8
a=rtpmap:0 PCMU/8000
a=rtpmap:6 DVI4/16000
a=rtpmap:8 PCMA/8000

If the fields in the message have the form Header: ValueCRLF (Carriage Return+Line Feed), they are called headers.

The first line contains INVITE, this is called the method. The line also contains the destination called request–URI (Uniform Resource Indicator) and a SIP version number.

The first header is on the second line. It is the Via header. Every proxy or UA that creates or forwards SIP messages adds its own address in a new Via header.

The next headers are To and From, they show the originator and the destination of the message. Name labels can be used, much like in e–mails, and that is why the SIP address is enclosed in brackets.

The Call–ID is an identifier to make every SIP session unique. It is created by a locally unique string, then the "@" and the host name is added to make it globally unique. It is important that this field is unique because both ends might have multiple calls and signalling.

CSeq is a command sequence number which contains the sequence number and the method, in this case INVITE. It makes it easy for each UA to separate each SIP conversation. Every answers to this method contains exactly this line ("CSeq: 1 INVITE"). This is to be able to identify the answers with corresponding method. The number is increased with one each time a new message (method) is sent. The sequence number can start at any number, in this example it is set to 1.

There are many other headers that can be added to each message, but only those discussed above are necessary in all SIP messages.

Optional fields in this example is the Contact and Subject headers. The Contact header can be used to route messages directly to the originator (Spider Mike in the example). The Subject header contains a message, in this case: Drive faster!. This message can be displayed when an INVITE is received and has with same function as the subject line in an e–mail. That gives the option to the
receiver not to accept the call.

The header named \textit{Content - Type} indicates that this message has a body attached to its end. It also contains the type of the body attached, here the body is a Session Description Protocol (SDP).

\section*{Using SDP as Payload in SIP}

When looking at the message described in the section before, you see the body that is attached to the end of the SIP message. The body contains all media attributes that the caller wants for the session. Nothing is assumed by SIP, so it will be exactly as the caller has described it. Each line is a separate field and it always starts with a letter and then a ". Likewise SIP, many SDP fields are optional. Each line in SDP will now be discussed.

The v= field contains the version number of SDP. But since the current version of SDP is 0 all valid messages always begin with v=0.

The o= field have information about the originator of the session, username and a session id. The field is a unique identifier of each session. It contains the following information:

\begin{verbatim}
  o=username session-id version network-type address-type address
\end{verbatim}

The \textit{username} can either be originators name or host. \textit{Session - id} is either a random number or a timestamp using NTP (Network Time Protocol). The \textit{version} is also a timestamp using NTP or a number which is increased each time the session is changed. The \textit{network - type} is always IN, which indicates Internet, and \textit{address - type} can either be IP4 or IP6 which are short for IP (Internet Protocol) version 4 or IP version 6.

The s= field is the name on the session. The name can be any type of ASCII characters. This might seem unnecessary since the o= field makes the session unique, but it is mandatory.

The c= field have the media connection info. It looks like this:

\begin{verbatim}
  c=network-type address-type connection-address
\end{verbatim}

or for multicasting:

\begin{verbatim}
  connection-address=base-multicast-address/ttl/number-of-addresses
\end{verbatim}

The \textit{network - type} is as earlier IN for the Internet and \textit{address - type} is IP4. The \textit{connection - address} is the IP address of the receiver. And as shown above, the address can also be a multicast address.

The t= field indicates start and stop time for the session. The times are defined with NTP timestamps. But as shown in the example, start and stop times can be set to zero to indicate infinite time. A start time and a stop time with value zero indicates that the session is permanent until changed.

Then there are many optional fields, but non are unimportant.
The m= field contains information about the type of the media session. It contains:

\[ m=\text{media port transport format-list} \]

The media parameter can be audio, video, application, data or control. The port is the port number and transport is the type of transport protocol that should be used. Often the transport protocol RTP/AVP (Real Time Protocol/Audio Video Profiles) is used.

Format-list contains more information about the media. Often it is information about the payload types contained in RTP.

a= fields are also optional, they contain characteristics of the starting media session. In this example the sender has explained the payload types 0 6 8. This can be important when using payload type numbers between 96 and 107. This numbers is allocated dynamically and the explanation is used to avoid misunderstandings between clients. The example values 8000 and 16000 are sample rates that ensures that sound and video will played at the same rates as they were recorded.

For more exhaustive knowledge about SIP, [7] and [9] can be appropriate.
Chapter 3

The Application

3.1 Introduction

The application is basically a tool for generating traffic in order to analyze audio quality when packet losses or delays occur. To be compliant with the 3GPP standardization the application has to use the AMR codec, SIP signalling and the RTP/RTCP protocol. Also very important parts were to learn how to handle sound, and how to master network buffering which brings delays.

The user controls the application via a text based interface in a terminal window. It may not look much to the world, however the interesting parts are under the surface. There is a lot going on during an execution of the program (this will be discussed in detail below). The application talks to several protocol stacks, compresses/uncompresses data and sends it to the soundcard or other devices.

3.2 Network Overview

This section will give an insight in how the application interacts with 3G network; how the communication between end nodes is setup and carried out; which entities that is involved in the communication and their locations in the network.

The protocol stack for the application is depicted in Figure 3.1. The figure shows an overview of the communication between the different layers within each end node. The details of the inter-layer communication will be discussed below.

Figure 3.2 describes the communication between two end nodes, equipped with the application, over the UMTS network. Do not pay too much attention to the details in the figure, they are there to make the picture complete (and maybe as something extra for the interested reader) but are really not in the scope of this thesis. The communication starts when a AMR phone sends a SIP INVITE (described in detail in section 2.7) to a SIP proxy, MT (Mobile Termination). The MT, located in the access network, sends on the INVITE to a Serving GPRS Supporting Node (SGSN) that forwards it to a Gateway GPRS Supporting Node (GGSN). Both the
SGSN and GGSN is located in the Packet Switched (PS) domain. The INVITE message then leaves the PS domain and travels into the IP multimedia domain to a Call Session Control Function (CSCF). The CSCF is as a SIP server and is considered as the primary SIP node in the network. After the CSCF the INVITE message is sent to the receiving AMR phone. The way through the network, for this last transport, is different depending on where in the network the receiving AMR phone is located; for example the way is different if it is a mobile entity or if it is an entity located in the IP network.

To get an overview of where the different nodes (TE, MT, SGSN...) is located in the 3G network see Figure 3.3. Likewise Figure 3.2 the details in the picture is of no importance in this scope and is preferably overlooked.

### 3.3 Structure

The program structure can be described as a module based hierarchy, so it is quite easy to add new functions or to develop a nice GUI (Graphical User Interface). We have created as much code as possible in object oriented C++ and therefore the structure is mostly class based. A schematic view over the functionality of the application is depicted in Figure 3.4. The figure shows how a session is created and how the session is handled between two clients.

#### Application Startup

First everything is initialized from our main-control function. This main-control function fulfills all necessary preconditions and activates all objects in the correct order.

The preconditions are a lot of variables that has to be set before any signalling or sessions can be created. The variables are stored in a file called “config”, and everything in this file is then stored into the memory when the program is started.

When these things are finished, the user is presented with a menu, containing sub-menus, showing how to navigate the application. During this, a batch file called
“runfirst” is read and executed. This file is created mostly for the users convenience. The file is not necessary, but it can be convenient since it may contain commands and settings that are executed automatically so that the user does not have to input them manually every time he/she starts the application. There is also a possibility for the user to create and run other, different, batch files manually from the command prompt since different settings might be useful for different types of sessions.

From a prompt in the menu the user can input commands that will be executed each time enter is depressed. And an example might look like this:

Amrphone:> run conf/settings1
Amrphone:> set dest sip:rubberduck@e2eQos.ericsson.se
Amrphone:> call

The first command line “run ...” will execute all commands listed in the batch file called “config1” located in a folder called “conf”.

**Figure 3.2.** Communication overview (End–to–End)
The second line sets the destination address, similar to an e-mail address. The sip URI (Uniform Resource Indicator), the address, may contain very advanced settings for the sip protocol that are out of the scope of this thesis. But some simple URI destinations might be `sip:Mr. F.Rubberduck@e2eqos.ericsson.se` which works like in emails with “Mr. F.Rubberduck” as the receiving user. Another URI might be `sip:rubberduck@e2eqos.ericsson.se; maddr=239.255.255.1; ttl=15` which indicates multicast to 239.255.255.1 with `ttl=15` (Time To Live).

The last line activates the call. The “call” command will hence activate the communication and the initiation signalling which will be discussed below.

**Session Initiation Signalling**

By activation the application sends orders to the SIP protocol to transmit an INVITE message to the destination address. It will also put in a template of SDP fields into the message, which are adjusted to fit our application. See section 2.7 for information on how this works. Parts of this SDP template can be changed from the command interface.

An example of our SDP field can be:
Here we have chosen port 17000 for the RTP traffic. The RTP payload type (PT) is 97. The PT 97 is not reserved, by IANA, for a certain kind of payload and can therefore be allocated dynamically to be used by payloads that does not have a reserved number. Here it is associated with the AMR codec with the sample-rate 8000. In this case only the AMR codec modes 4.75, 5.9, 7.95 and 12.2 kbit/s may be selected by the receiver side in this session. The receiver side is also constrained to use octet-aligned mode. See section 3.3 subsection Packeting AMR Frames into RTP Packets about this.
On the receiver side this SIP message will be received and interpreted. It will be examined for different types of sessions. All session types that are proposed in the INVITE message will be tested if they can be handled by the receiving client. Those that are supported are put into a new SIP OK message and transmitted back to the originator of the INVITE message together with their port numbers and modes.

The RTP stack can now be initialized with the values from the SDP fields on both sides, since they now both know what codecs and modes they support. Also information about how the codecs are “packed” is given to the RTP packer objects, and the speech session starts.

There are also messages called REINVITE and they are treated exactly like new INVITE messages. They are used when an existing session should be modified in some way. They contain the same information as the original INVITE messages but with the new settings instead. So an existing session will be changed to whatever is defined in the REINVITE.

**Media Session**

**Sound Recording**

Both the sending and receiving sides are now in a kind of looping stage. Sound are recorded from the sound card using the UNIX/Linux dsp (Digital Signal Processor) device. The soundcard settings have been changed to minimize read and write delays. This involved recalculating the DMA (Direct Memory Access) buffer utilized by the soundcard and activating the multi fragment settings to improve DMA performance. The buffer consist of an amount of fragments with a specified size, and the soundcard will block access to each memory fragment until it is filled. Therefore many fragments of small size is preferred by an application which needs fast access to the data. See section 2.2.

The user also might have an interest in using other devices than the soundcard to input or output the sound streams. But that will be described later in section 3.3 subsection Using other Devices for Input and Output.

**AMR Encoding**

When sound is recorded, it is compressed using the AMR codec and our application stores it into a circular buffer which is used for redundancy frames in RTP packets. This circular buffer is shifted around for each AMR frame that is inserted. The point with this is to retransmit old speech frames together with new speech frames inside each RTP packet. For example, one frame that previously have not been transmitted is sent in a RTP packet along with three “old” frames. Since the application runs over UDP, no retransmission is done (as mentioned in section 2.5 subsection Introduction retransmission is often meaningless in a real time application). This redundancy method is used to cover up for lost packets. It means that some RTP packets can be lost, without loosing any speech frame. It will just increase the delay with 20ms. The same occurs for network jitter, it also builds up
a delay almost like lost frames. This delay is only kept for the whole sentence for which the user is talking. So nobody will notice temporary network jitter. What happens is actually that the receiving application starts to buffer speech frames and plays them in order. And when the sending side stops transmitting packets, the receiver will in time catch up. The redundancy level can be set by the user in the application user interface.

It may sound strange that the clients stop transmitting packets, but this is a feature from the AMR encoder and is called Voice Activity Detection (VAD). The transmission is actually not stopped, there is only less frames sent. Since there is no speech it is possible to remove all transmission. But this is not an option because people react on the total silence in the phone and thinks that the connection is lost. That is why some frames are sent all the time. These frames are not ordinary compressed speech. They are actually created by the encoder using 7 ordinary speech frames and filtered so that only background noise levels are recognized. This means that the only thing that is sent is background noise. This single noise frame is extracted so it covers 7 real frames by the decoder. The decoder will therefore always feed the soundcard with sound.

If packets are lost which contains these noise frames, the sound output will after a while slowly fade to silence. See section 2.4 subsections Source Controlled Rate Operation and Voice Activity Detection (VAD) for more detailed information.

As mentioned above the VAD algorithm is a feature in AMR. When no speech is recorded the encoder produces NO_DATA frames. To keep the synchronization, in order to be able to generate comfort noise, the decoder has to be fed with NO_DATA frames in exact the same time intervals as they are produced by the encoder. But in order the reduce the network load our application does not send the NO_DATA frames, produced by the encoder, over the network. It is therefore up to our application to calculate when a not sent NO_DATA frame should be generated and fed to the decoder on the receiving side. Without doing this, no comfort noise will be generated. This part was a bit tricky, especially to understand that this was the way it was supposed to work.

Packing AMR Frames into RTP Packets

Packing AMR frames into RTP packets may sound easy to do. The thing that has to be accomplished is just to copy the speech frames into the RTP packet. But there are other things that require more attention. The fact is that the AMR speech frames consists of several parts which have different durability against errors. Therefore each speech frame are separated bitwise into three classes called A, B and C classes. Frames that have errors in class A bits will create the most severe distortion.

The class A, B and C bits are later re-packed into different positions in each AMR speech frame according to the bit-ordering defined in [2]. See chapter 2.4 subsection Frame Structure. Some other options like cyclic redundancy check (CRC) and interleaving may also be selected in the packer, these options can be set from the application user interface.
All frames are handled separately by the packetizing algorithm. They are handled differently according to different types, see section 2.4 subsection Source Controlled Rate Operation. Also there were some interpreting issues about the rules how a certain bit should be set in the RTP packet. The markerbit, M, in the RTP header has to be set correctly. It marks the beginning of each talkspurt\(^1\). This enables the receiver to identify silence between talkspurts. Once identified, the receiver can lengthen or shorten these silence periods in order to regulate the playback. It is a fact that small differences in the length of silence in between words is not perceptible for the human ear. However, if the words them self is of different length it is perceptible and annoying.

The rules for when this bit should be set is described in [11]. This description is not very verbose, though. At first it was a problem to understand the purpose of the marker bit (this is not explained in [11]) and thereby it was hard to understand how it should be set. When we finally understood the purpose, described above, it was easier but still there was the problem to recognize a talkspurt on a frame level. We figured out that the beginning of a talkspurt could be recognized by checking if the current speech frame was preceded by a NO\_DATA frame. If it was, the markerbit should be set. Thereafter we also had to check if the current speech frame was carried as the first frame in a RTP packet; if it was not, the markerbit should not be set.

When the desired amount of frames are packed into the RTP packet utilizing the previously discussed circular buffer, it will be sent via UDP etc. to the set destination.

Receiving RTP Data

The application will check if there are any packets to receive at least every 20\(\text{ms}\). If there is a packet then it will be received and depacked. The depacking works almost like packing of AMR frames into an RTP packet, but backwards. A schematic picture of a RTP packet containing AMR frames with timestamps\(^2\) can be seen in Figure 3.5. If there is no new packet to receive then there might be frames that can be depacked from the previous RTP packet. Frames extracted by any of these two methods are treated equally. The usual case is that there is a new packet to receive every 20\(\text{ms}\) if no network jitter or packetloss occur. If one or more packets are lost then error concealing actions will be taken by the decoder (see section 2.4 subsection Error Concealment of Lost Frames) until a new packet can be received. From this new packet some amount of previous lost frames can be extracted which can be seen in Figure 3.5. Now since the application is playing old frames, it is delayed in time against the sender. This delay will only continue until next time the encoder detects speech inactivity. Then source controlled rate operations takes action which gives the decoder time to catch up.

This delay is also utilized by us to buffer packets. If the delay is, for example

\(^1\)A talkspurt is a collection of sounds followed by silence.

\(^2\)This is the AMR timestamp in the RTP header i.e. it has a resolution of 160 samples per 20\(\text{ms}\)
20\text{ms} \text{ then the application can allow loosing packets again, as long as at least one packet is received in between, without adding any delay or loosing any quality of the speech. The same procedure is applied to network jitter.}

As seen in the Figure 3.4 we have separated the sections “depack” and “depacked”. They are actually totally different in our implementation. The “depack” will only be called upon when a new RTP packet is received. It will then extract the AMR frames using the table of contents carried in the RTP packet if the cyclic redundancy check (CRC) are correct. It will also deinterleave the whole packet if that option has been selected.

The “depacked” on the other hand first selects a new AMR frame (that has been extracted from the RTP packet by “depack”) or discards old frames according to the AMR timestamp. Then it takes care of moving the class A, B, and C bits to the original positions. After this the speech frame has been fully reconstructed (it now looks exactly like the encoder output) and it will be sent to the decoder.

**AMR Decoding**

Most of this section has already been discussed in the earlier section AMR Encoding. The theory that is related to the network is explained in section 2.4 and the part that is related to speech encoding/decoding is explained in appendix A. What happens, in the AMR decoding phase, is that the decoder decompresses and rebuilds the data representing the sound. This data can later be processed directly by the soundcard.

**Playing Sound**

When arriving at this section, we have raw data bytes that we want to play.
Before the first time we access the soundcard, we need to tell it what type of data we want to read from— or write to it. This has been done in the initialization phase. These settings are for example the rate that sound will be played, what type of data our sound is and mono/stereo settings. For further information on audio programming see section 2.2.

Strangely many modern soundcards do not support mono, and telephones are seldom in stereo. So we had to strip and rebuild these byte streams manually in our recording and playing phases. Stereo sound consist of 2 channels often called left and right. The two channels are “plaited” together in a way so that first comes two bytes from one channel then two from the other and so on. When we strip the sound to mono we simply removes every two bytes using a simple algorithm we implemented. Another similar algorithm is used to rebuild the channel before sending the sound to the soundcard.

Also the rate of the sound cannot be set exactly to 8000 samples per second on all soundcards, but we have left that problem to the user of the application. It sounded very good anyhow. If we would fix the problem in our code we would have to change the rate of speech by Fourier transforming all sound to the frequency plane, and then inverse transform it back to the adjusted time plane. This would have taken too much time from our primary goal.

The sound will now be written to the soundcard. It is strictly important that this writing is nonblocking, because we want to do other things while we are playing the sound. For example we want to extract and decode a new sound frame during this sound is playing, so we do not get any stuttering i.e. milliseconds of silence between each frame. The ear is very sensitive and it needs an even flow of sound.

Using other Devices for Input and Output

The user might be interested in doing something else with the sound than playing it, or recording from something else than the soundcard. This has been implemented in a way that makes it easy to use.

Most users are interested in saving the sound to a file, or reading from a file. Some examples below show some settings:

Amrphone:> set output /home/username/outfile.raw
Amrphone:> set input /home/username/somesound.raw

Here we send all output from the phone to a file called outfile.raw, and all input is taken from somesound.raw. We prefer using the “.raw” ending or “.au” for all file types. This is because these files slightly differ from most ordinary sound files.

The difference is that they do not contain information about how they should be played, like samplerate or stereo/mono modes contained in conventional file types like “.wav”. Of course the user also can use both soundcard and files:

Amrphone:> set output /home/username/outfile.raw
Amrphone:> set input /dev/dsp
By the availability of choosing something else than the soundcard for input, some things were affected. Using the soundcard as input will create frames with exactly 160 samples and 20ms separation since it also needs 20ms to record the sound itself. Choosing a source that do not create this natural delay by itself will result in flooding of the network with packets. This is done since the application is created to work as fast as possible (as is customary). The UMTS carrier is created for a constant bandwidth and therefore the rate of frames or packets must be controlled. For this we had to create a bandwidth throttling algorithm, and it will strive towards an exact rate of frames at 20ms. This was a bit tricky to implement since there is not any C functions like sleep and similar that can be set to so low time intervals. Another thing that complicated things were that we did not want to tie up valuable CPU time.

Anyway the bandwidth throttling algorithm will be auto selected if the user chooses not to use the /dev/dsp for input. It is up to the user to experiment with other devices, but it may be interesting to use some devices shown below:

- /dev/dsp1 (soundcard2)
- /dev/dsp2 (soundcard3)
- /dev/null (trash device)
- /dev/ttyS0 (serial port)
- /dev/parport0 (printer port)

Being able to select files or devices freely like this also have other benefits. Since the application is an instrument created for measurement, it might not always be the sound that is interesting. If the user has the ability to understand underlying layers in network and signalling, an option can be selected to create all from simple to extensive log-files.

**Session Termination Signalling**

When a user wants to hang up, we will arrive to this section and SIP becomes active again. The client that wants to hang up sends a message named BYE to the other side. It looks like this:

```
BYE sip:rubberduck@18wheelers.com SIP/2.0
Via: SIP/2.0/UDP traders.com:5060
To: Mr. Rubberduck <sip:rubberduck@18wheelers.com>
From: Mr. Spider Mike <sip:spider_mike@trucks.traders.com>
Call-ID: 123456789@traders.com
CSeq: 2 BYE
Content-Length: 0
```

As we can see the message contains less information than INVITE messages. It can also only be sent when a session already exists. The message will be acknowledged by an OK message and both sides will stop the conversation.
Nothing may be sent over the previously defined session after the BYE message has been transmitted. If more information has to be sent to the other side another new session has to be established using INVITE again. Section 2.7 gives further detailed information about the SIP protocol.

Control Information from RTCP

During the media session RTCP sends control information as described in section 2.5 subsections Real-Time Transport Control Protocol (RTCP) and Sender and Receiver Reports. This information is written, nonblocking, to a log-file. It is necessary to write nonblocking otherwise there is a risk that speech starts stuttering. The sender and the receiver reports are the most interesting parts that are written to the log-file. Our application does not use this information to take any actions it just provides it, however it can be used for thorough analysis of the QoS.

3.4 Perceptual Sound Quality

Experimental Setup

In order to evaluate the application sound quality and to assess how sensitive the speech is to disturbance, a machine were connected in between the communicating entities. This machine contained a program called Mate. The Mate program is used to filter network traffic. It can drop a certain percentage of the RTP packages, delay them a certain amount time or inflict a certain bit error rate. A fourth machine connected to the network were running the well known network analyzing program, Ethereal, to monitor the traffic generated from the application.

The tests were performed as ordinary telephone conversations. The application were also tested by sending sound recorded in files over the network and recording it on the receiver side. These tests are only subjective (not objective) and have been performed by laymen in a non scientific manner.

Speech Evaluation

Manual evaluation of the sound (using the human ear), both from the recorded files and from the phone-calls, shows that the quality is very good. Using the Mate program to filter the data sent over the network, resulted in the discovery that smaller amounts of packet loss do not imply any noticeable sound defects. Neither do small amounts of packet time delays affect the users perception of the sound quality. An acceptable delay between sender and receiver can be as much as 200ms without the users noticing it. Delays higher than this conveys that people will begin to continually interrupt each other.
Chapter 4

Conclusions and Discussion

4.1 Results

All the predefined objectives with this master thesis specified in section 1.3 are accomplished. These are as follows:

- a fully operational AMR phone implementation that can be used for QoS measurements in 3G networks.
- an application capable of handle automatic testing using batch files.
- an application that supports both subjective and objective test.
- an application compliant to the 3GPP standardization

The application functions as a tool to generate traffic over the End–to–End QoS testbed and over the UMTS network in general. It is implemented so it can be used for internal and external demonstrations and it forms a basis for further prototype development, see section 4.4.

Future presentations and (possible) co–operation with TietoEnator will fulfill the goal to enhance their knowledge within this area.

Another result is that manual evaluation of the speech in a conversation proved that the quality is very good.

4.2 Implementation Difficulties

The communication between the application and the different protocol stacks has been extremely difficult to implement and has conveyed a lot of work. The fact that we, partly, used code implemented by others resulted in lot of troubles due to minimal documentation or total absence of it. Some times even erroneous behavior occurred. But producing the protocol stacks and the codec ourselves would not have been an option. The amount of time and knowledge required was not available.
Another thing that caused a lot of problems were that the code we used (programmed by others) often were fitted to another operating system and another compiler than the ones we were using. This meant that we had to rewrite make files, configure files and similar. This was an, for us, completely new area of knowledge.

4.3 Areas of Usage

In general the application can be seen as a tool for generating traffic in order to analyze the sound for packet loss, packet delays and bit errors in a packet. It could also be used to test new products to see if they are compliant to the 3GPP standardization. With this in consideration it is obvious that the application can be useful in various contexts and places. For example everybody that develops mobile phones (Sony Ericsson, Nokia, Siemens etc.) and 3G system developers (Ericsson, Siemens, Motorola, Nokia etc.) can benefit a lot from the application. Also telecom operators, like Vodafone, Orange and Deutche Telekom should be most interested in possibilities the application supplies.

The possibilities are not limited to 3G systems; the application can be used in similar ways in Internet applications. Where it, for example, can be used as a tool for analyzing sound quality depending on different levels of congestion in the network. It can also be used to analyze network specific problems, both locally and globally.

4.4 Possible Extensions and Improvements

Though the sound quality is very good as it is (as discussed above), it could be enhanced without any major changes in the application. This could be done by refining some of the already existing functions. Another action that could be taken to improve the quality of the sound is to employ AMR-WB (Adaptive Multi Rate – Wide Band) instead of AMR. However, this will approximately double the bandwidth used.

As mentioned in the introduction, our application is not a complete multimedia application, it is only capable to handle speech/audio. However it is designed so that additional modules easily can be added. Typically an additional module can contain code that enables the application to communicate with a new data source, such as web camera and/or mp3 player. An elective movie player is another possible source, although the bandwidth required will then be immense.
Bibliography


Appendix A

AMR Coding Scheme

Though this thesis is not about speech coding in particular, speech coding is used frequently. In this scope it is interesting to understand how the amount of speech data can be reduced so significantly by coding. This is very important in a network load point of view. Here it is, among other things, explained what is sent over the network (representing the speech). This explains how it is possible to reduce the amount data significantly.

The coding scheme used in AMR is Multi–Rate Algebraic Code Excited Linear Prediction (MR–ACELP). It is based on two different synthesis filters. One filter is used to model the short–term spectrum of the speech signal. The short–term synthesis filter function is given by:

\[ H(z) = \frac{1}{A(z)} = \frac{1}{1 + \sum_{i=1}^{m} \hat{a}_i z^{-1}} \tag{A.1} \]

It is a 10th order linear prediction (LP) where \( \hat{a}_i, i = 1, ..., m \), are the quantified linear prediction parameters and \( m \) is the order of the filter (in this case \( m = 10 \)). The other filter is used to model the long–term periodicity also called the pitch. The long–term synthesis filter function is given by:

\[ \frac{1}{B(z)} = \frac{1}{1 - g_p z^{-T}} \tag{A.2} \]

where \( T \) is the pitch delay and \( g_p \) is the pitch gain. This filter is using the so-called adaptive codebook\(^1\) approach.

Every 20 ms speech frame is analyzed to extract the parameters of the CELP model (linear prediction filter coefficients, adaptive and fixed-codebook indices and gains). These parameters are encoded and transmitted over the network. The parameters are the encoded speech data. At the decoder, these parameters are decoded and used to retrieve the excitation and synthesis filter parameters. Then

\(^1\)The adaptive codebook contains excitation vectors that are adapted for every subframe. It is derived from the long–term filter state. The long–term filter delay, typically the true pitch period, can be viewed as an index into the adaptive codebook.
the speech is reconstructed by filtering the excitation signal through the short–term LP synthesis filter. The CELP speech model is shown in Figure A.1.

\[
A(z) s(n) + v(n) c(n) u(n) \]

\[
g_c
\]

\[
g_f \]

\[
L_P s(n) \]

\[
A(z) \]

\[
\frac{1}{A(z)} \]

\[
\hat{s}(n) \]

\[
\hat{x}(n) \]

Figure A.1. Simplified block diagram of the CELP synthesis model “© ETSI 2002. Further use, modification, redistribution is strictly prohibited. ETSI standards are available from http://pda.etsi.org/pda/ and http://www.etsi.org/eds/”

As shown in Figure A.1 the speech is synthesized by adding excitation vectors from the adaptive and fixed codebooks and then feeding the sum (the excitation signal) to the short–term synthesis filter. The synthesized speech is then compared with the original speech in order to choose the optimum excitation sequence in a codebook. This selection is done by using an analysis–by–synthesis search procedure. This procedure uses a perceptual weighting filter to choose the synthesis in which the error between the original speech is minimized. The perceptual weighting filter is given by:

\[
W(z) = \frac{A(z)}{A(z)} \]

where \(A(z)\) is the unquantized LP filter and \(0 < \gamma_2 < \gamma_1 \leq 1\). For the 12.2 and 10.2 kbit/s mode \(\gamma_1 = 0.9\) and \(\gamma_2 = 0.6\) for all other modes \(\gamma_1 = 0.94\). All LP parameters used are unquantized.

The LP analysis is performed twice per frame for the 12.2 kbit/s mode and once for all other modes. A speech frame is divided into four subframes of 40 samples each (5 ms long). For every subframe, both the adaptive and fixed codebook parameters are transmitted. On basis of the perceptually weighted speech frame an open–loop pitch lag is estimated for every other subframe (except from the 5.15 and 4.75 kbit/s modes for which it is done once per frame). This estimation results in the best possible pitch delay and pitch gain.

At the decoder side the coder parameters are retrieved at each frame. These parameters are used to obtain LP filters at each subframe. For each subframe the
excitation is constructed by adding the adaptive and fixed code-vectors (the code-
vectors are scaled with their respective gains. Then the excitation, corresponding
to one subframe, is feed to the short–term LP synthesis filter and there by the
speech is reconstructed. Finally, the reconstructed speech is passed through an
adaptive post-filter\textsuperscript{2}.

For further reading \cite{4} is recommended.

\textsuperscript{2}The adaptive post–filter is used to enhance the perceptual quality of the reconstructed speech.