MOBILE DISTRIBUTED WIRELESS STEREO

Author:
Elisabet Berbel Gonzalez

Supervisors:
Frank Fitzek
Janus Heide
Morten V. Pedersen

AALBORG UNIVERSITY

Department of Electronic Systems
Section of Antennas, Propagation and Radio Networking (APNet)
Fredriks Bajers Vej 7
http://www.es.aau.dk/sections/

Title:

Mobile distributed wireless stereo

Project period:

Autumn semester 2009

Project group:

gr 991

Participants:

Elisabet Berbel Gonzalez

Supervisors:

Frank Fitzek
Janus Heide
Morten V. Pedersen

Copies: 5

Date of completion:

January 6^{th}, 2010

Abstract:

Nowadays mobile phones are an essential part of our life. Due to ever increasing demands from the consumer, mobile devices incorporates more and more functionalities.

The main aim of this project is develop an application able to transmit audio files in real time using a Internet Tablet from Nokia (N810) and analyze all the problems that can come up as the synchronization between devices.

Declaration of Authorship

I, ELISABET BERBEL GONZALEZ, declare that this Master thesis titled, 'Mobile distributed wireless stereo' and the work presented in it are my own and has been generated by me as the result of my own original research. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at Aalborg University.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed: ELISABET BERBEL GONZALEZ

Date:January 6^{th}, 2010

Acknowledgements

First and foremost I would like to thank my thesis advisor Janus Heide and Morten V. Pedersen, for their support and guidance throughout the project, for the meetings during the development of the project and for helpful comments with refining the finished document.

Also, I would like to thank my thesis coordinator Frank Fitzek that has allowed me to make the project into his research group and has given me a lot of facilities to carry out the project.

I thank Aalborg University, the Erasmus Program and my university (EPSC-UPC) to give me the opportunity to do my final Master Thesis in other country as well as the experience to know other culture and language.

I thank Miguel Angel Botella, Fabian Molina, Javier Pablos and Raúl Riesco for listen during the long afternoons when the project development was not in the correct way. Also I want to express my grateful to all people that helped to perform the test.

Finally, I thank my family, Trinidad Gonzalez and David Berbel, and Albert Fuero. They always supports my dreams and aspirations without they I could not carry out this experience. I would like to thank their for all they are, and all they have done for me.

"Many of life's failures are people who did not realize how close they where to success when they gave up"

Thomas A. Edison

Contents

List of Figures

List of Tables

Abbreviations

UI	**U**ser **I**nterface
GTK	**G**imp **T**ool**K**it
GNU	**G**nu is **N**ot **U**nix
LISP	**LIS**t **P**rocessing
BASIC	**B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode
ARM	**A**dvanced **R**ISK **M**achines
MIPS	**M**icroprocessor without **I**nterlocked **P**ipeline **S**tages
PC	**P**ersonal **C**omputer
SPARC	**S**calable **P**rocessor **ARC**hitecture
CD	**C**ompact **D**isc
IP	**I**nternet **P**rotocol
FTP	**F**ile **T**ransfer **P**rotocol
TELNET	**TEL**elecommunications **NET**work
NFS	**N**etwork **F**ile **S**ystem
ISO	**I**nternational **O**rganization of **S**tandardization
DCT	**D**iscrete **C**osine **T**ransform
GSM	**G**lobal **S**ystem for **M**obile Communications
GPS	**G**lobal **P**ositioning **S**ystem
MPEG	**M**oving **P**ictures **E**xperts **G**roup
MP2	**MP**EG Audio Layer**2**
MP3	**MP**EG Audio Layer**3**
AAC	**A**dvanced **A**udio **C**oding
WMA	**W**indows **M**edia **A**udio
WAV	**WAV**eform Audio Format
AMR	**A**daptive **M**ulti-**R**ate

RA	**R**eal **A**udio
HTTP	**H**yper**T**ext **T**ranfer **P**rotocol
RTSP	**R**eal **T**ime **S**treaming **P**rotocol
RTP	**R**eal **T**ime **P**rotocol
RTCP	**R**eal **T**ime **C**ontrol **P**rotocol
TCP	**T**ransmission **C**ontrol **P**rotocol
UDP	**U**ser **D**atagram **P**rotocol
DNS	**D**omain **N**ame **S**ystem
WI-FI	**WI**reless **FI**delity
IANA	**I**nternet **A**ssigned **N**umbers **A**uthority
RFC	**R**equest **F**or **C**omments
LDU	**L**ogical **D**ata **U**nit

Chapter 1

INTRODUCTION

Nowadays, in the market is found a wide range of portable devices to fulfill all the customers requirements. It can found devices from Smartphones to tablets PC which are able to provide all the features of a personal computer but with a smaller size.

Among the different devices that are in the market it can found Tablets PC. A Tablet PC is a device between a Smartphone and a laptop, and it refers to a slate-shape mobile computer equipped with a touchscreen that used the fingers or a stylus to operate the computer, and provide all the features of a PC. There are two different types of Tablet PC: the UMPC (Ultra Mobile Personal Computer) and the MID (Mobile Internet Device). The UMPC is addressed to business people since it is better for computing applications but, on the other hand, it is more expensive than the MID. The MID is addressed to all people due to it reasonable price and lower size, and uses operative systems like Ubuntu Mobile, Windows Mobile or Linux. The main functionalities of MID are multimedia applications.

To enter in this market Nokia has developed a mobile Internet appliance between a Personal Digital Assistant (PDA) and an UMPC. The Internet Tablet made by Nokia is mainly focused on Internet and media features.

The main aim of this project is to provide an improvement in the multimedia features in this type of devices, more specifically, in the audio area. In order to be able to give the possibility to the customers to distribute audio files to several devices in real time.

To achieve the aim is used streaming technology that allows to transmit continuous information, in this case audio files, through a network and reproduce the playback simultaneously to the download, without saving the file information in the hard disk. In this way, the streaming technology, fragments all the information and send the different LDU through the network to be join in the client. So, in this project it is done a study of the architecture that must have a streaming application, an analysis of the different protocols that must to be used to develop the application, the server and the client necessary to transmit and receive audio in real time and a study of several real scenarios analyzing the main problems that can come up and how to solve it. In Figure 1.1 it is shown a scheme of the scenario simulated in this project.

Finally, the report is structured following the next guideline:

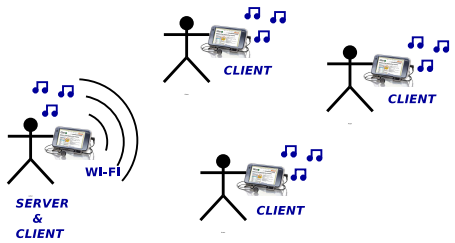

FIGURE 1.1: Audio Streaming application scheme

Firstly, in Chapter 2 is described a briefly description about the device where is added the application, as well as, it main features and operative system. Next, in Chapter 3, it is studied the structured that must follow an application to transmit file in real time, analyzing all the Internet protocols available to carry out the functionalities of the different layers. Finally, it is explained what protocols are the most suitable for the aim of the project and the decision taken.

In Chapter 4, it is explained the real-time server and client developed. It is described the different modules that make up the application as well as the different functionalities of each one. Once it is developed the application, it is necessary to explain some acoustics problems that can come up. So in Chapter 5 and 6 it is explained the main acoustic problem that can appear in the project scenario and it is described the test done to

confirm the effect over the audio perception of the file, respectively.

Then, in Chapter 7 it is analyzed three different real scenarios and the main problems that come up. Moreover, it is discussed how to solved the problems studied in the different scenarios. Finally, in the conclusions, it is discussed the main points as well as the problems that are appear during the achievement of the project. Finally, in Chapter 9 and 10, it is described the future works that are needed to improve the application and the timing diagram of the project, respectively.

Chapter 2

DEVELOPMENT ENVIRONMENT

This chapter presents the specifications of the device used to develop the real scenario of the application. Moreover is explained briefly a description about the platform that use these type of devices and the operative system over which the implementation is executed. Is important to know the different platforms used to chose correctly the programming language as well as the libraries that will be used.

2.1 Nokia N810

Nokia N810 has been designed to allow the user access to all Internet application like a PC but with a small and portable device.

The device provides a quality Internet connection as well as an integrated GPS, a QWERTY keyboard and a wide range of protocols that allow the user send e-mails, make Internet calls (like skype), instant messaging and playback a lot of types of video and audio files, between others.

In the next table, some of the main technical features of the tablet is shown.

N810 Specifications	
Processor:	TI OMAP 2420, 400Mhz
Memory:	DDR RAM 128MB Flash 256MB
Connectivity:	WLAN standard: IEEE 802.11 b/g Bluetooth specification v.2.0 USB high speed for PC connectivity 3.5 mm stereo headphone plug (Nokia AV Connector)
Web Browsing:	Browser based on Mozilla technology with state-of-the-art web standard support including AJAX Full desktop Adobe Flash 9 plug-in, including video and audio streaming
Media:	In-built media player for viewing and listening to downloaded, transfered or streamed media content and easy-on-device management of media library Direct access to shared media over Universal Plug and Play (UPnP) Supported video formats: 3GP, AVI, WMV, MP4, H263, H.264, MPEG-1, MPEG-4, RV (RealVideo) Supported audio formats: MP3, WMA, AAC, AMR, AWB, M4A, MP2, RA (RealAudio), WAV Supported playlist formats: M3U, PLS, ASX, WAX, WVX, WPL
E-mail:	Browser access to familiar webmail services Get an easy start to desktop email with pre-configured email services E-mail application for personal e-mail usage with IMAP, STMP, and POP3 support

TABLE 2.1: N810 Specifications [1]

2.2 Maemo

Maemo is a platform implemented by Nokia, in collaboration with several open source projects, developed to run on mobile devices such as smartphones and Internet Tablets. The platform uses open source code based on a Linux kernel and the Debian distribution package management.

In Figure 2.1 the key components of a Maemo platform can be seen.

- **Linux kernel:** the kernel of the platform is based on a Linux system. It provides support for the hardware system like memory, CPU and I/O devices.

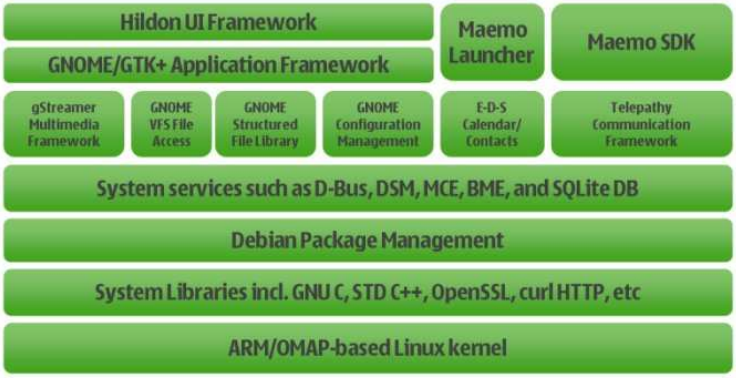

FIGURE 2.1: Maemo Structure [2]

- **System libraries:** It is based on the standard GNU C library.
- **Debian Package Management:** It uses all the main features, such as the file system hierarchy and the design policies, about package management from Debian distribution.
- **System Services:** The D-Bus system is the method used to do the communications between desktop applications which are running simultaneously, furthermore it also provides the channel for the communication between the desktop session and the operating system.
- **Maemo launcher:** It is the responsible for launching all applications. It is composed of two parts: the Maemo-invoker, which is the responsible to execute the script to start the service, and the Maemo-launcher, which is a server that initialize the applications' data.
- **Hildon UI Framework:** Is responsible for providing finger interface. It consist on a set of GTK extensions that provide functionalities for the mobile devices and also provides the desktop environment.

2.3 Linux

Linux is an operative system from free distribution, developed by Linus Torvalds. The system is formed by the system kernel plus a large number of programs and libraries. Linux is distributed under the GNU General Public License, it means that the source

code must be always available and it can be freely distributed, copied and modified.

There are a lot of programming languages available to develop in Linux, the main is C/C++ but it is also possible to used Java, Objective-C, Pascal, LISP, BASIC, Perl, between others.

Next, some of the main features of a Linux system is listed:

- Multitasking: Is the ability to execute several programs at the same time.
- Multi user: A lot of users using the same machine in the same time.
- Multi-platform: It can be used in different platform such as 386-, 486-, Pentium, Pentium Pro, Pentium II, Amiga and Atari it is also version to used in other platforms like Alpha, ARM, MIPS, Power PC and SPARC.
- Multiprocessor: Supports system with more than one processor.
- Memory protection between process, it means that one of these process can not fail the system.
- Multiple virtual console.
- CD-ROM files system that read all the CD-ROM standard formats.
- TCP/IP, including ftp, telnet, NFS, etc.
- Several network protocols included in the kernel: TCP, Ipv4, Ipv6.

Chapter 3

AUDIO STREAMING

This chapter explains a briefly description about what is audio streaming and the general architecture that should have a system which the main purpose is to transmit multimedia content, audio or video, in real time. It is described the different protocols and audio codecs that it can be used to transmit and encode the audio files, respectively.

Finally, the conclusions section explains the protocols and audio codec chosen and why they are more appropriate for the aim of the project.

3.1 Audio Streaming Definition

Audio streaming is a method of delivering audio directly from the source to the player in real-time, that means that data is transmitted at the same rate as it is consumed. This is a continuous process, with no intermediate storage of the media.

There are two types of streaming files, live or on demand. Live streams are also called true streaming and consist of start the transmission without any request by the clients. When the client make a request to the server, he has to connect to this transmission and receive the information that is transmitted at that moment. In this type of streaming clients can not interact with the multimedia contents, the only action allowed is pause the content. On the other hand, on demand streaming or progressive streaming, the client requests the multimedia content that is wanted in each moment and the server

sends a specifically information for each client. In this case, the client can control and interact with the multimedia content since the information is found previously stored in the hard disk of the server.

Downloading vs Streaming

Downloading consist of saving the content in a server and providing users with the content. The user downloads the file to the local hard disk, and then playback the content.

The main drawback to downloading the files is the need to copy the entire files in the computer before play the content, it has an effect on the users' resources since they consume time and disk space. This problem is due to that the user has to wait until the whole file download is complete before playing the file. Furthermore, downloading does not make efficient use of bandwidth in comparison with streaming technology, since all the bandwidth's resources available are used to transfer the data as soon as possible.

Conversely, the bandwidth use for the streaming is more efficient than downloading, since the speed of the transmission only depends on the bitrate of the client. Furthermore, since the audio file is not stored in the local disk it does not consume disk space in the user device. On the other hand, streaming needs a buffer to store some data since during the connection it could be some problems like delay or gaps in the stream.

3.2 Streaming Architecture

There are four components to a streaming architecture [3] :

- Capture and encoding: take the raw audio and video from the microphone and camera and process them into a compressed computer file.
- Serving: the encoded file is uploaded to a server for delivery out to the network. A streaming server is more than just a fancy file server, it controls in real-time the stream delivery.

- Distribution and delivery: the distribution channel connects the server to the player.
- Media player: receives the stream and decompresses back to regular audio and video, where it can be displayed on the device.

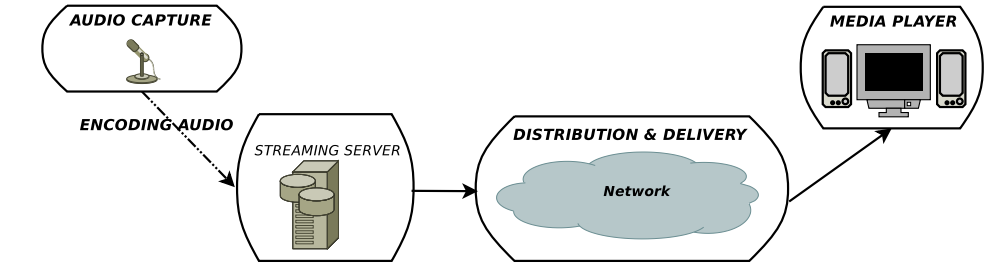

FIGURE 3.1: Streaming Architecture

3.2.1 Encoding Audio

The audio compression consist of a reduction of the quantity information, specifically in reduce the size of the audio files. The main aim is to transport the same information, but reducing the storage size of the audio files and the transmission bandwidth required.

As it is said before, the main aim is to reduce the file size, trying that this reduction does not affect the quality of the content. However, the compression can affect to the quality of the file, depending on the desired quality it is possible to chose between two different audio compression methods:

- Lossless audio compression: this method compresses the information so that the data before and after being compressed are exactly the same. In this case, a higher compression only implies a higher processing time. The bit rate is this type of compression is always variable and the main uses of this technique is in the text file compression.
- Lossy audio compression: Unlike the lossless method the lossy audio compression consist of removing data that the human is not able to hear, to reduce the

size of the file. With this type of methods when the compression is done, it is impossible to obtain the original signal. In the lossy audio compression the bit rate can be constant or variable. It is mainly used in video and audio compression.

One of the main features of lossy audio codecs is the bit rate, it is used for measuring the quality of the file that is coded. When a high bit rate is used it means that less information has been discarded.

Next, it has been listed and described the different audio codecs that are supported by the device.

- **MPEG-1**: MPEG is the acronym of *Moving Picture Expert Group* and it is a group of persons who are part of the *International Organization for Standardization* (ISO) to design standards for digital video and audio compression. MPEG-1 is standard responsible of coding the video and audio for the digital storage environment down to 1,5 Mbits/s.

  The method used to compress the audio is based on the psychoacoustics principles, basically in the auditory masking [1] and simultaneous masking [2]. This procedure is based on removing the irrelevant parts, like the redundant sound, of the audio. This is possible since the parts of the sound that are inaudible for the human auditory it can be removed without any degradation for the human perception. The main features of MPEG-1 are:

  - The audio sampling rate can be 32, 44.1, or 48 kHz.
  - The compressed bit stream can support one or two audio channels.
  - The compressed bit stream can have one of several predefined fixed bit rates ranging from 32 to 224 kbits/s per channel. Depending on the audio sampling rate, this translates to compression factors ranging from 2.7 to 24. MPEG/audio offers a choice of three independent layers of compression:
    * Layer I is the simplest and is best suited for bit rates above 128 kbits/s per channel.

[1] Auditory masking occurs when the perception of one sound is affected by the presence of another sound [4]

[2] Simultaneous masking is when a sound is made inaudible by a "masker", a noise of unwanted sound of the same duration as the original sound [5]

  - Layer II has an intermediate complexity and is targeted for bit rates around 128 kbits/s per channel.
  - Layer III is the most complex but offers the best audio quality, particularly for bit rates around 64 kbits/s per channel.

The features list above have been obtained from "A Tutorial on MPEG/Audio Compression" [6].

Next, it is listed the two audio codecs from MPEG standard that are supported by the device.

- *MP2 (MPEG-1 Audio Layer 2)*: The compression in MP2 takes place in the time domain using a low-delay 32 sub-band polyphased filter bank achieving overlapping ranges to prevent the aliasing.
  The higher quality is achieve at bit rates of 256 kbits/s, although at 64 kbits/s is also acceptable. The main applications for this type of compression is in the audio broadcast, professional recordings and multimedia.

- *MP3 (MPEG-1 Audio Layer 3)*: MPEG-1 layer 3 (MP3) uses a more sophisticated hybrid filter bank with a modified DCT following each polyphase filter [3].

  The quality of the audio file depends on the compress ratio; the standard for an acceptable quality is 128 kbits/s, that is the bit rate used in Internet. The use of this type of codec in Internet is very common due to its combination of high quality and high compression ratio.

	Bit rate range (Kbit/s)	Target bit rate (Kbit/s)	Typical compression ratio
Layer 1	32-448	192 (mono)	1:4
Layer 2	32-484	128 (mono)	1:6 to 1:8
Layer 3	32-448	64 (mono)	1:10 to 1:12
		128 (stereo)	

TABLE 3.1: Summary of MPEG-1 audio [3]

- **AAC**: *Advanced Audio Coding* is a audio compression format developed by the Fraunhofer Institute with AT& T, Nokia, Sony and Dolby. Like the MP3 format, AAC used lossy audio compression, removing some audio data to achieve a higher

level compression without changing the output perception of the audio.

AAC uses variable bit rate, adapting the number of bits depending on the complexity of the audio transmission in a specifics instant. The algorithm used has a higher performance than the MP3, achieving higher quality in smallest files and requiring less system resources to code and decode the files. Moreover, AAC allows polyphonic sounds with a 48 channels and sample frequencies between 8 Hz and 96 kHz.

- **WMA (Windows Media Audio)**: is an lossy audio compression format although recently it has been developed like a lossless audio compression. Is the version of Windows to compress audio, and it allows to reduce the size of big files apart from adapt the codec to different connection speeds in case that it is necessary to reproduce in real time over Internet.

- **WAV**: *Waveform Audio Format* is an lossless audio compression that is used to store audio file in the PC with the operative system windows.

  It is able to support almost all the audio codec, but it is used mainly with PCM (Pulse code Modulation). To obtain a CD quality it is needed record sound at 44100 Hz and with 16 bit, and for each minute of recording consumes 10 Mbytes of hard disc. Its main limitation is that it is only allows record files of 4 GB that are approximately 6 hours and a half in CD quality.

  It is not used to share music by Internet since it does not reduce a lot the size of the file.

- **AMR (Adaptive Multi-Rate)**: is an audio compression format developed for the voice codification. In October 1998 AMR was chosen like the audio method compression for the 3GPP (*3rd Generation Partnership Project*) and now is the most used in GSM.

  This format is been developed to compress voice, so it is not able to achieve good result compressing audio files like music.

- **Real Audio**: Real Audio is an audio format developed by the Real Network company, which use a variety of audio codecs that provide a high compression rate and reduce considerably the size of the audio or video file.

  The Real Audio format allows to adapt the reception capacity of the client depending on the connection speed to Internet. If the user can receive audio packages with a high quality without interruptions, it will send the audio with this speed, in case of a low speed connection the codec decrease the sample frequency to adapt the compression to the reception capacity, in this case the audio quality will be worst.

3.2.2 Serving

As it can be explained before, a streaming server is a server that send files to a device and this files can be reproduce in real-time, in other words, the client does not have to wait for download the file completely. Due to this reason, the streaming server must have several additional functions over a standard web server:

- Real-time flow control.
- Intelligent stream switching.
- Interactive clip navigation.

Taking into account that HTTP does not support any of this new functionality, it must be developed new protocols for streaming media:

- **Real-Time Streaming Protocol** (RTSP), is an application-level protocol that simplifies the distribution of multimedia content in Internet. It was designed to controls the data delivery in real time properties. RTSP is a connection-less protocol, instead of this the server maintain a session assigning an identification session number, almost in all the cases RTSP uses TCP to control the data and UDP to transmit the audio and video data, although it is possible to use TCP to transmit the data. During a RTSP session, a client can open and close several transport connections between the server if it is needed.

Due to RTSP is an application layer protocol, it was designed to be compatible with HTTP providing a similar syntax and operation and being able add some extension mechanisms from HTTP.
Nevertheless there are some differences between the two protocols [7]:

- RTSP introduces a number of new methods and has a different protocol identifier.
- An RTSP server needs to maintain state by default in almost all cases, as opposed to the stateless nature of HTTP.
- Both an RTSP server and client can issue requests.

- **Real-Time Protocol** (RTP), the main aim of RTP protocol is to provide an uniform environment over the transport protocols to deliver data, like audio and video, that have to be transmitted in real time. The main function is to implement the sequence numbers of the packages to reconstruct the information in the client. The main three functions of RTP are:
  - Identify the type of the transmitted information.
  - Add temporal marker and sequence number into the transmitted information.
  - Controls the packets in the destination.

- **Real-Time Control Protocol** (RTCP), is the protocol that is used to control the RTP flow, it allows to transmit fundamental information about the different participants and the quality of the service. Its functionality is based on periodic control package transmission.
  RTCP performs four functions[8]:
  1. The primary function is to provide feedback on the quality of the data distribution.
  2. RTCP carries a persistent transport-level identifier for an RTP source called the canonical name or CNAME.

3. The first two functions require that all participants send RTCP packets, therefore the rate must be controlled in order for RTP to scale up to a large number of participants. By having each participant send its control packets to all the others, each can independently observe the number of participants. This number is used to calculate the rate at which the packets are sent.
4. A fourth, optional function is to convey minimal session control information, for example participant identification to be displayed in the user interface. This is most likely to be useful in "loosely controlled" sessions where participants enter and leave without membership control or parameter negotiation.

3.2.3 Distribution And Delivery

The main functionality of the transport layer is to provide the communication between two applications, it means that this layer is the responsible of controlling the data flow between devices. Depending on the type of network and their requirements it will choose TCP or UDP. TCP is an oriented-protocol connection and guarantee the delivery of the packages in the destination. Conversely, UDP is an unreliable no oriented-connection protocol.

1. TCP protocol

   The Transmission Control Protocol (TCP) is a connection-oriented protocol, it means that two devices connected between them can control the state of the transmission. The use of this type of protocols allows the applications to be able to have a reliable connection independently of the lower layers. During a TCP connection the devices must to establish a connection where will take place the bidirectional connection. Other of the main functions of the TCP protocol is the capability to controls the speed of the data using its capacity of transmit messages with variable size.

   TCP provides five key services to higher-layer applications[9]:

   - Virtual circuits: virtual circuit provides the reliability, flow control, and I/O management features that distinguish it from UDP.

- Application I/O management: TCP provides an I/O buffer for applications to use, allowing them to send and receive data as contiguous streams.
- Network I/O management: TCP has to provide network I/O management services to IP, building segments that can travel efficiently over the IP network, and turning individual segments back into a data-stream appropriate for the applications.
- Flow control: TCP must be able to deal with variations in the send and receive data rate for the different hosts. Furthermore, it has to do all of this seamlessly, without any action being required from the applications in use.
- Reliability: TCP provides a reliable transport services by monitoring the data that it sends.

2. UDP protocol

   UDP is defined in RFC 768, which states that UDP is a stateless, unreliable transport protocol that does not guarantee delivery. Thus, UDP is meant to provide a low-overhead transport for applications to use when they do not need guarantee delivery[9].

   In addition, there are some applications that can not use a TCP connection, since TCP requires a virtual end-to-end circuit, in some applications like streaming or the DNS services is more important the fast reception of the data instead of the verification of data.

3.3 Project Specifications

Finally, and after evaluated the possibilities it has been decided the next specifications for the project implementation.

The connection between devices will be done through a WI-FI connection. Specifically it will be used the IEEE 802.11 g standard which works in the 2.4 GHz band and provides a maximum bit rate of 54 Mbits/s and about a 22 Mbit/s of average throughput.

Due to that the connection must be multicast and in real-time it has been decided to used the UDP transport protocol, since it provides a connection without do a previous reserve of bandwidth. Also in case of packet loss it does not send the packet other time, that in case of audio streaming is the best solution since the loss of one packet it is not perceptible for the human ear. In addition, in a real-time‘s download it is not allowed to stop the transmission to send a packet that it is lost.

For the application layer, it will be used the Real-Time Streaming Protocol since it allow us to delivery data in real-time as well as an interactive control of the playback (play, pause, etc.) over the RTP protocol used to encapsulated the data that must to be transmitted in a UDP socket.

Moreover, it is important decided what audio compressor will be used to code and decode the audio file. In point 2.2.1 it has been analyzed all the audio codecs formats that supports the device. The best audio format for this type of application is the Real Audio format which has been develop to transmit over Internet in real time, nevertheless due to that this format is proprietary and it is a container not a codec it has been decided to used as codec the Advanced Audio Coding (AAC) that is the codec used in Real Audio 9 and 10 and is supported in almost all the libraries since it is a MPEG standard.

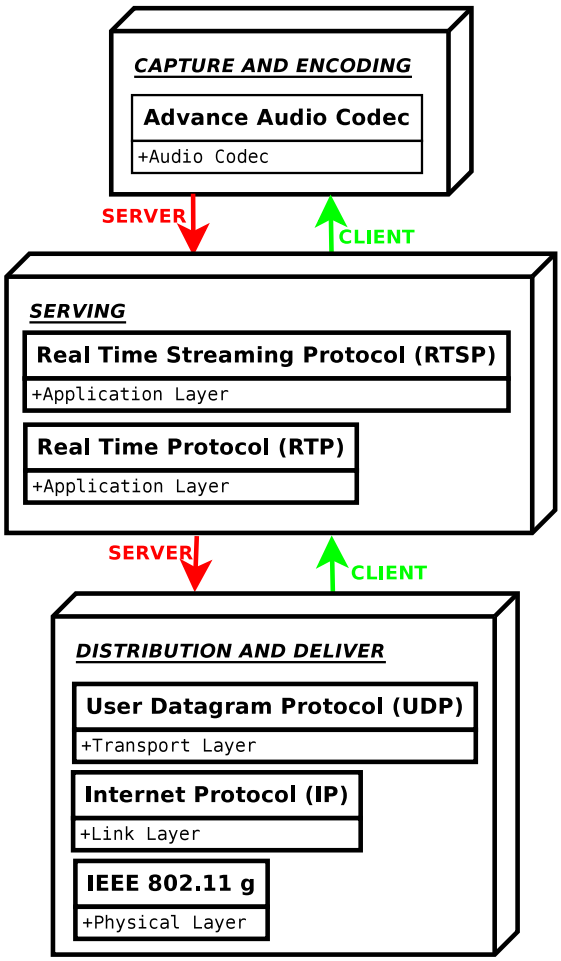

FIGURE 3.2: Project Specifications

Chapter 4

IMPLEMENTATION

In this chapter it is explained the implementation of the server and client that it has been developed to transmitted audio between different devices. The programming language chosen to implement it is C, and it is used the Gstreamer library to do all the procedures that it is necessary to transmit audio in real time.

Moreover, it is included the flow diagram and the pipeline diagram that show, in a visual way, all the steps carried out to achieve the aim of the project that is to transmit audio.

4.1 Audio Streaming Server

The main aim of this project is to be able to transmit an audio file in real time. To do that it has been divided the aim in two parts: the first one consist on adapt the audio files that have the people in a mobile device, in this case in an Internet Tablet, to be suitable for transmit it in real time. The second part is based on encapsulated the data correctly and open all the connections necessary to transmit it over the network.

As it has been explained before, the library used to do the server is Gstreamer, that provides a great number of high-level components to carry out the implementation in a simple and intuitive way.

In the first part it have been created all the elements that is needed to transform the mp3 data, considering that almost all the audio files that have the people nowadays are mp3, to aac data, that is the codec chose to carry out the transmission. To do this it is necessary to create a pipeline, which is the container where it is stored all the elements and where takes place the execution of them. The elements necessary to transform the mp3 data to aac data are: file source, mp3 decode, audio convert, aac encoder and RTP payload. An element is like a black box where there are an input and an output and between them the data are transformed. Next it is listed the functionality of the different elements:

- FILE SOURCE: The file source only generate data, it is the element that reads the audio file and creates a data buffer that it is used in the pipeline.
- MP3 DECODE: The mp3 decode is the responsible for decode the mp3 data.
- AUDIO CONVERTER: This element is the responsible for adapt the data to encode into aac format.
- AAC ENCODER: The aac encoder, encodes the output data of the audio converter into aac audio data.
- RTP PAYLOAD: Finally this element is the responsible for encapsulate the data into a RTP package.The container used to encapsulate the aac file is MP4, that is the audio container for this type of audio codecs. Specifically, it is used an element that payload MPEG-4 elementary streams as RTP packets according with the RFC 3640 [10]. This specification defines a payload structure to transport MPEG-4 streams, focused on audio and video streams.

In Figure 4.1, it is shown the flow diagram of the first part of the program where it can be seen how, as the different elements are created, pass through the error control to check that it has been created correctly and then are added into the pipeline.
The aim of the second part of the program is to create all the elements needed to transmit the data and link all the elements into the pipeline to make sure that the data flow through all the elements.

Firstly, it is necessary to create an RTP bin that consist on an element that allows

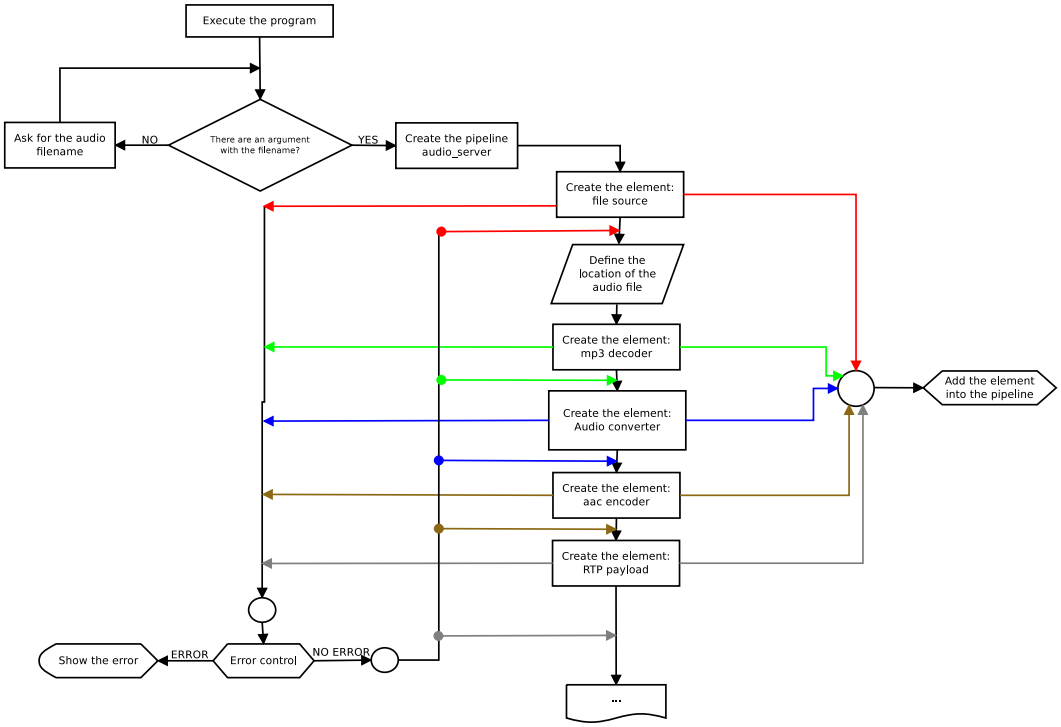

FIGURE 4.1: Flow diagram audio streaming server: Part 1

to combine different linked elements into only one logical element. This type of element is needed to link the data encapsulated in the RTP package with the UDP element. Then it is necessary to create a multicast UDP sink that is the element that allow transmit the RTP package over the network. With the multicast UDP sink is possible to transmit the data for the same port number to several devices, to do that it is needed to define pairs of host:port in the connection configure. The UDP protocol uses a numerical identifier to allow the end-to-end communication between hosts, the numerical identifier is known as a port number. A port number is a 16-bits unsigned integer, and its values can range from 0 to 65535. The ports between 1 to 1023 are well-known ports and they are assigned by the IANA (Internet Assigned Numbers Authority), the ports between 1024 to 49151 are registered ports and they are assigned to a certain use and the ports between 49152 to 65535 are dynamic and/or private ports that are the ports available to use by an application.

Once the elements are created and configured, it is needed to link all the elements. It is important to link all the elements to be able to transport the data through the different elements in the pipeline. The last step, consist on create the sink pad to send

the data over the network and set up the pipeline. It is necessary to set up the pipeline since one element can not perform any action until the state of the element change. There are four different state: null, ready, paused and playing [11]:

- GST_STATE_NULL: this is the default state. This state will deallocate all resources held by the element.
- GST_STATE_READY: in the ready state, an element assign all the resources that it needs.
- GST_STATE_PAUSED: when an element is in the paused state it has opened the stream but it does not execute.
- GST_STATE_PLAYING: in this state the stream is opened and it is processing.

In Figure 4.2 it is shown the flow diagram of the second part of the server.

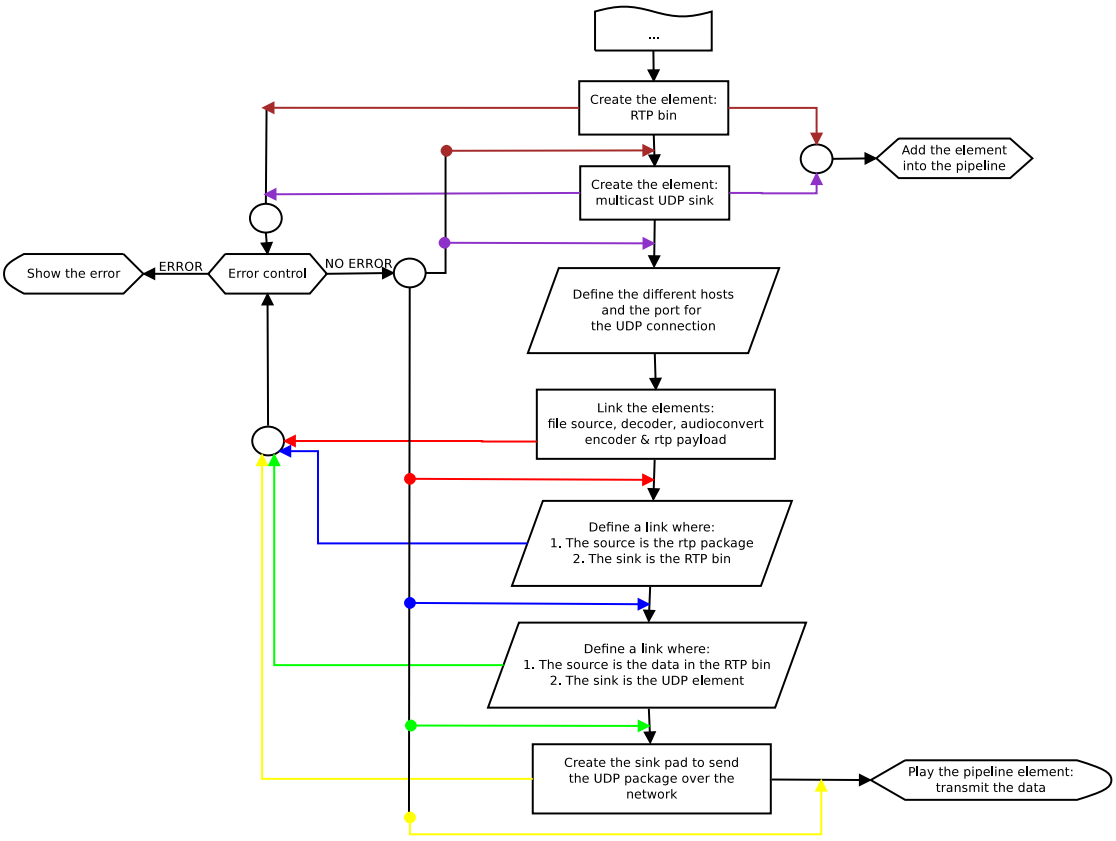

FIGURE 4.2: Flow diagram audio streaming server: Part 2

Next, in Figure 4.3 it is shown the pipeline diagram, where it can see the different elements that are used to implement it. It is important to chosen the port number correctly, according the RFC 3550 [12] for UDP protocol the RTP should use an even destination port number and the corresponding RTCP stream should use the next higher odd destination port number. As it can see in the figure the port used to transmit the data is 55555 and the port used to transmit the control package is 55556.

Moreover, there are two elements which are not explained before, the RTCP sink and source. The RTCP is based on the transmission of control packages to all the users, informing of the data quality distributed by the source. The RTCP sink is the responsible to send the information about the source to the client and the RTCP source is in charge of reception of RTCP packages sent by the client. These elements will be useful in the network synchronization.

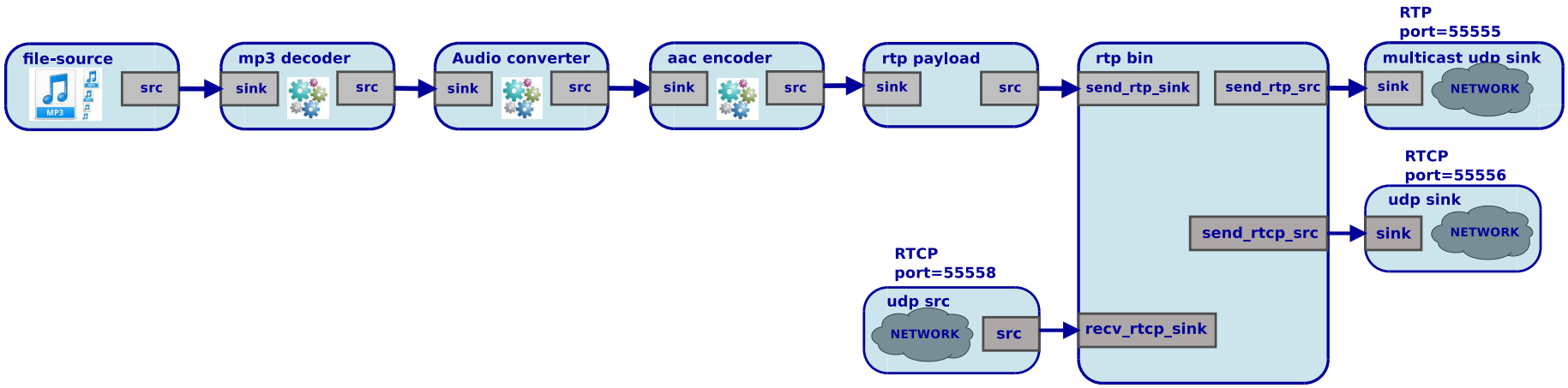

FIGURE 4.3: Server pipeline diagram

4.2 Audio Streaming Client

In the part of the client the method used to received the data and playback it is the same. It is defined a pipeline where it has added all the elements necessary to play the audio file. In this case, the UDP element is the first to be created, as it is needed an element able to receive data. It has been created a source UDP element, this type of element only generate data to used in the pipeline. For the UDP source it is needed to define the port number and the capabilities, which are the type of data that it is transmitted over the network. Then, it is needed a RTP bin, as in the server this element is the responsible for link the UDP element and the RTP package. In Figure 4.4 , it can see the flow diagram of the first part of the client.

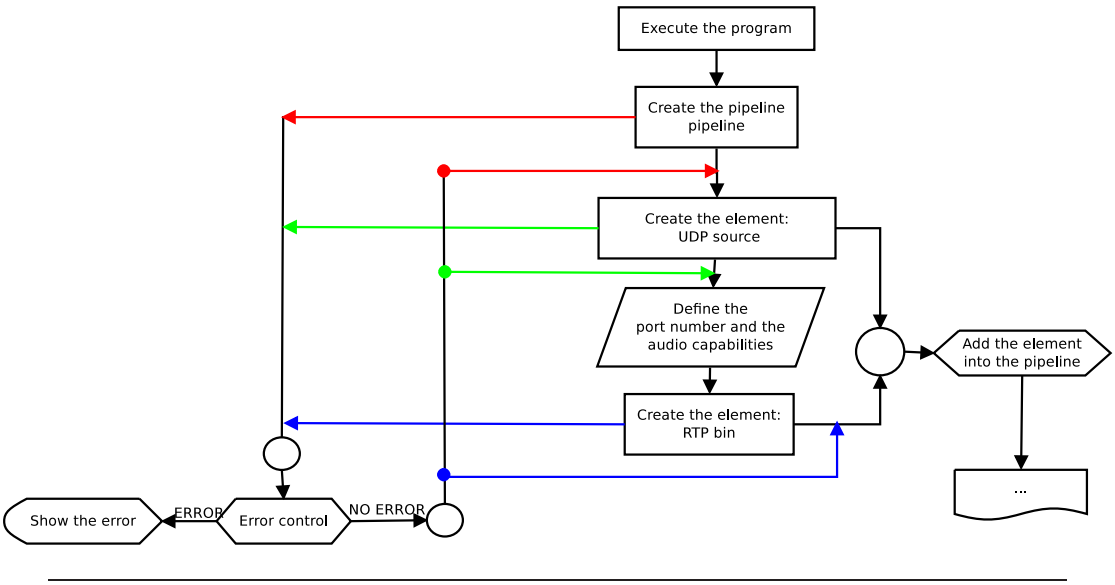

FIGURE 4.4: Flow diagram audio streaming client: Part 1

Once, the client is configured to received the data, it is needed to create the elements that deal with transform the data into data available to reproduce in an audio player. To do that, it is needed an element responsible for depayload the data. On the other hand, and as the data sent are encode in aac it is needed an element that decode the aac data.

After decode the data it is needed to resample the raw audio in the best sample rate to enhance quality before play it in the audio player. Finally, it is used the auto audio sink

element that detects the best audio sink to use. Thus, is the element the responsible for search between all the elements that have “sink” and “audio” in the class field of the element information and chose the best one.

Finally, it is needed to add all the elements into the pipeline, created the sink pad needed to received the data, link all the elements and activated the pipeline changing the state from NULL to PLAYING, as it has been explained in the server. Figure 4.5 shown the second part of the flow diagram.

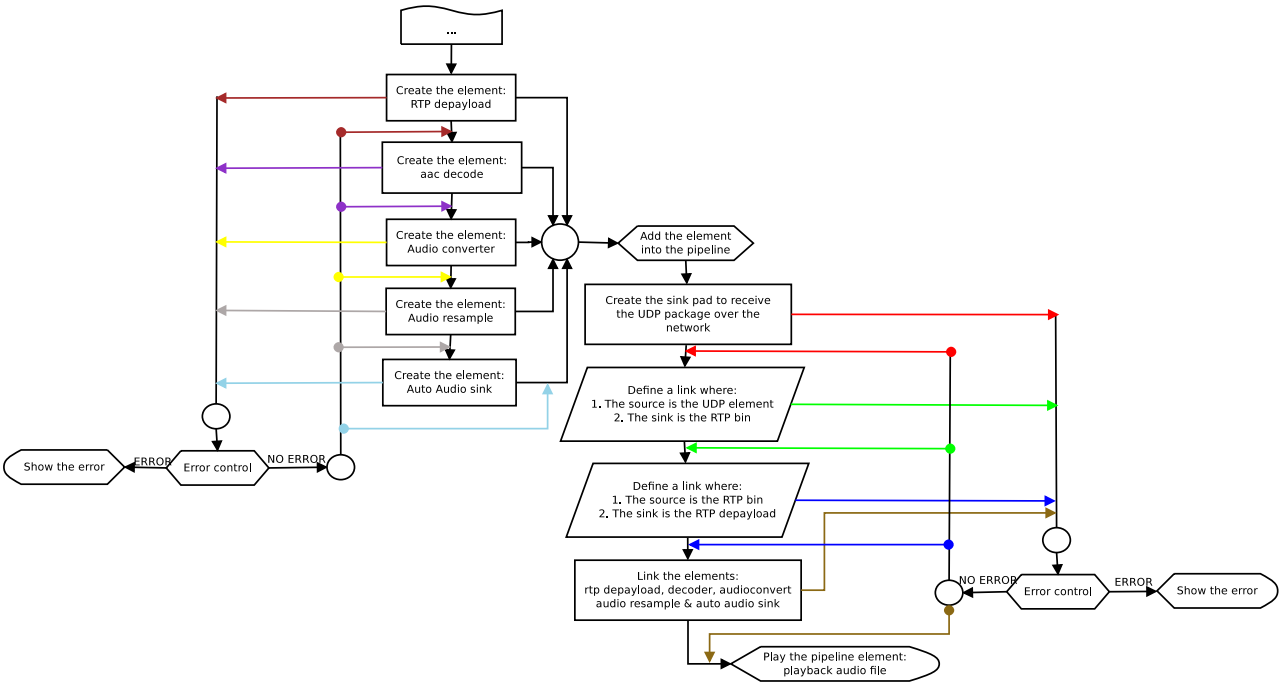

FIGURE 4.5: Flow diagram audio streaming client: Part 2

In Figure 4.6 it is shown the client pipeline diagram. It can see how the UDP port number to receive the data is the same as the port number in the server, and also there are two elements for the RTCP packets one for receive and other to send control data. The element to depayload the data is also based on RFC 3640.

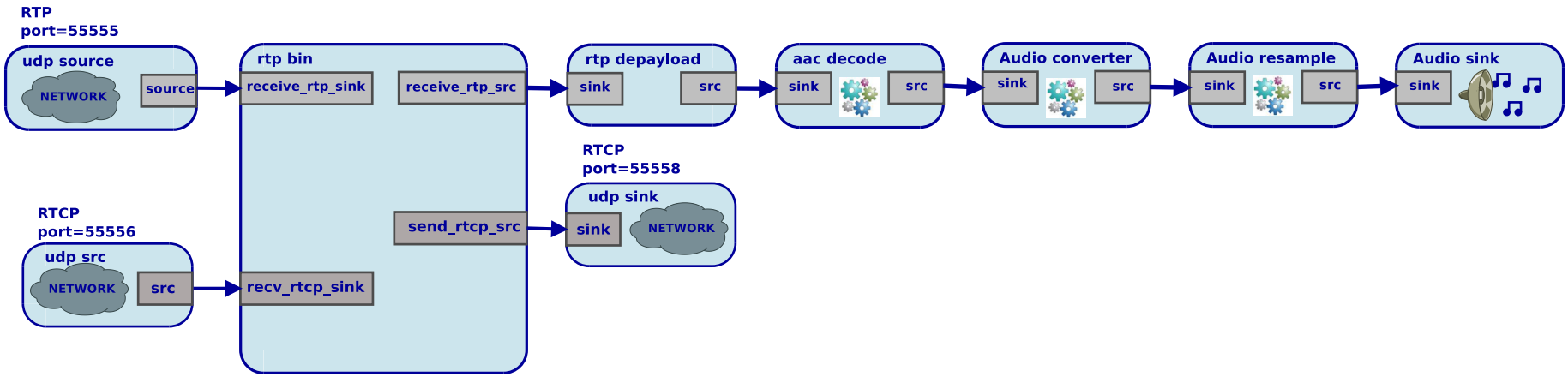

FIGURE 4.6: Client pipeline diagram

Chapter 5

ACOUSTICS PROPERTIES

The main aim of this project is to transmit audio in real time, but during the transmission can appear some problems related with the acoustics properties since in the environment where the project is developed, there are several users involve who their audio perception take an important role in the quality of the application. The aim of this chapter is to investigated about what are the sound properties and the problems that appear when it is needed play audio in a mobile device, such as the distance between several devices.

5.1 The Sound

The sound is an audible wave which consist an oscillation of pressure that are turned, by the human ear, into mechanic waves that are perceived by the brain. The sound propagation involves energy transport without matter transport using mechanic waves that are propagated through the solid, liquid and gaseous matter.

It can be describe some qualities about the sound:

- Pitch: is determined by the fundamental frequency of the sound waves. It allows to distinguish between low, high and medium sounds. The human ear is able to perceive the sounds of which their frequency are between 20 Hz and 20 kHz.
- Intensity: is the quantity of acoustic energy that contain the sound. The intensity is determined by the power which depends of the amplitude and it allows to

distinguish if the sound is strong or faint. The auditory threshold is 0 dB and the pain threshold is 140 dB.

- Duration: is the time that pass between the moment that start the sound until it stops. It determines the vibration time of an object.
- Timbre: is the quality that assign to the sound the harmonics that go with the fundamental frequency. This quality allows to distinguish between two sounds.
- Source: it determines where is the origin of the source.

5.1.1 The Source Quality And The Haas Effect

If two sounds are joined only in one, the localization of the sound depends of the location of the first sound that arrives. This is called 'The Precedence Effect' or 'The Haas Effect'.

The Haas Effect was described by the German doctor Helmut Haas and explain how the sound affects to the human perception.

The Haas Effect describe how if the human ear are hearing several independent sounds, the ear and the brain have the ability to gather all reflections arriving within about 50 ms after the direct sound and combine them to give the impression that all this sound is from the direction of the original source [13]. This effect is due to that the brain stops to perceive the direction and interpret the later sounds like a echo or reverberation of the first sound.

However, in order to the brain understand the origin of the sound in the middle of the two sounds, during the delay interval between 5 ms and 35 ms the second sound must have, more or less, an amplitude of 10 dB higher than the first.

To illustrated this effect it is created the 'Haas curve' that shows the intensity, expressed in dB, necessary to obtain an equivalent delay (expressed in milliseconds) between two sounds. In Figure 5.1 it is shown the 'Haas curve' and how in an interval between 5 ms and 35 ms it is needed more or less an amplitude of 10 dB higher to understand the two sounds like one.

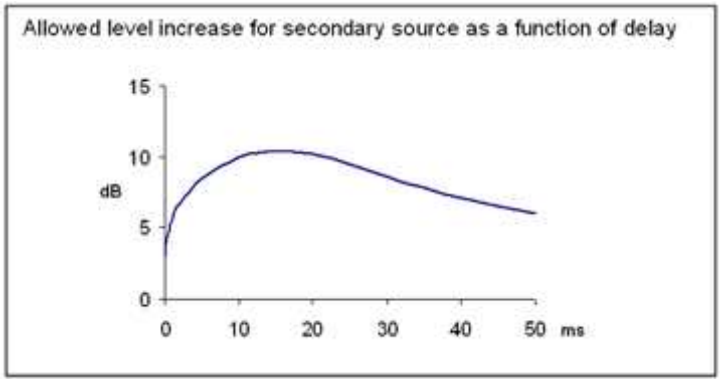

FIGURE 5.1: Haas Curve [14]

5.1.2 Speed Of Sound

To know the equivalent among the delay and the distance between two sounds it is needed to know some relation between distance and time as the sound propagation speed. The speed of the sound changes according to the environment since it depends on the properties through of which the wave is traveling. In solids, the propagation depends on the stiffness to tensile stress, and the density of the medium. In fluids, the more important factors are the compressibility of the medium and the density. And in gases, the main properties that it is needed to take into account when calculated the speed of sound are the compressibility and the density of the gas, properties such as the temperature and the molecular composition are very important.

Next, it is shown the relation between distance and time in air, that is the environment where the sound is propagated in the project:

$$V = \sqrt{\frac{\gamma \times R \times T}{M}} \tag{5.1}$$

Where the typical values for a standard atmosphere to the sea level are:

γ= Adiabatic constant = 1.4 in the air

R = Gas constant = 8.31 $JK^{-1}mole^{-1}$

T = Temperature in Kelvins

M = Molecular weight = 2.87 $\times 10^{-2} Kgmole^{-1}$

Therefore, the sound speed in the air according to the temperature in kelvin is:

$$V = 20.1 \times \sqrt{T} \tag{5.2}$$

Finally, if it is considered an ambient temperature of 20 °C it is obtain that the sound, in the air, is propagated with a speed of 344 m/s.

$$V = 20.1 \times \sqrt{T} = 20.1 \times \sqrt{(273\,^{\circ}\mathrm{K} + T_{\circ\mathrm{C}})} = \{T = 20\,^{\circ}\mathrm{C}\} \Longrightarrow V = 344 m/s \tag{5.3}$$

Chapter 6

HAAS EFFECT TEST

The following chapter analyzed experimentally the Haas effect studied in Chapter 5. The aim is to obtain the maximum distance allowed between devices that are playing audio for the tablets PC used in this project. In the different sections is explained the test that has been performed, the protocol that is carried out to do the test, the results obtained and finally it is analyzed and discussed the result according to the theoretical solutions.

6.1 Abstract

Chapter 5 explains the big role of acoustics properties in the audio streaming. Specifically, there are two main parameters that influences the perception of the sound: the Haas effect, an the delay introduced by the network.

This test studies the effect in the perception of the playback audio due to the distance between two different sound sources (Haas Effect). In order to achieve the test it has been studied the sound perception of a group of persons based on the distance between sources or more specifically depending on the delay between the sound signals.

6.2 Test Description And Specifications

The test consist on two different devices, one of them working like server and client and the other device working only as client. As it is said before, the main aim of this test is to analyze the human perception when they are hearing two identical sounds from different sources, taking into account the delay between the sounds. However, due to that the synchronization is not developed, it is decided no to do the audio streaming and playback the audio file with the Mplayer. In this way, it is possible to reproduce the signals at the same time. Figure 6.1 shows the initial scenario, considering that the worst case is when one individual have the device in his hand and the other individual is moving away progressively.

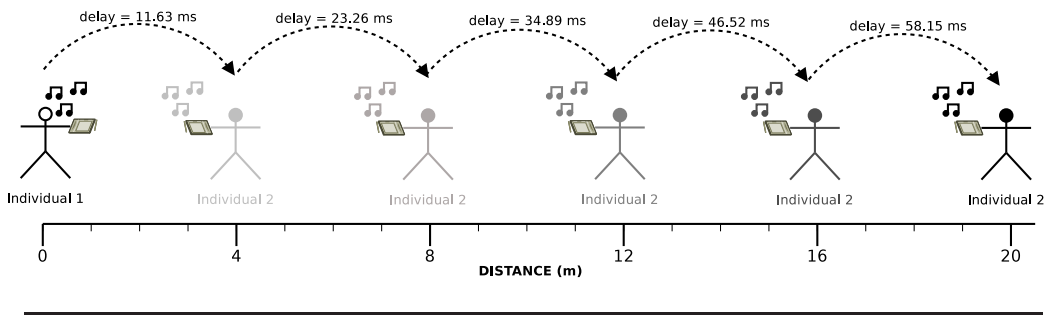

FIGURE 6.1: Test Scenario

As it is explained in Section 5.1.1, the maximum delay tolerable for the human ear to consider that a sound is an echo of the close sound is 50 ms, from this value the human perception is able to distinguish between the origin of the sound and start to hear two sounds delayed. In Figure 6.1 it is shown the scenario of the test, considering 20 °C of temperature since the test has been done inside of a building. To evaluated the perception it has been defined five different intervals, spaced between them 4 m that are equivalent to 11.63 ms and evaluating a maximum distance of 20 m or 58.15 ms of delay. The maximum distance has been decided taking into account that 50 ms correspond to 17.2 m.

Other parameter to take into account in the test is the outcome of the different individuals. During the test, in each interval, the individual is going to be asked about their perception of the audio file. In the next table it is described the three different

outcome that is analyzed.

OUTCOME	DESCRIPTION
1 Device	The individual only hear one sound and their brain interprets the delayed sound like a echo
2 Devices	The individual hear two sounds differentiated
Almost 2 devices	The individual hear one main sound but start to hear another sound coming from second source

TABLE 6.1: Outcomes of the test

Moreover, it is needed to decide the audio file that must be played and the duration of it, in this case it is chosen the *Sleep Away* audio file written by *Bob Acri*, and it has been defined a playback duration of 20 s. Also, it has been defined that the volume of the devices must be always the maximum volume. Finally, it has been decided to carry out the test in a room due to weather conditions, being conscious that in a room there are echo and reverberation problems.

6.3 Test Protocol Definition

In this part of the chapter it has been explained the procedure that it has been carried out during the test. It is important to define step by step all the process to do not forget anything and to realize the test in the most rigorous way as possible.

1. Define the reference point, this location is considering the initial point and all the distances must be calculated from this point.
2. Locate the two individuals and start the test with the first interval.
3. Move the second individual towards the second interval and start the test again.
4. Repeat the point 3 again until do the test in the 5 intervals.

6.4 Results

In Figure 6.2 it is shown the result obtained in the test. It has been studied twelve different individuals.

In the bar diagram can see the results obtained during the test, in the X-axis it is shown the five intervals evaluated and the Y-axis represents the number of individuals that respond each outcome, in total there are twelve individuals for each group of three bars.

The results obtained in the test are:

- For a distance of 4 meters, all the study group perceive only one device.
- In 8 meter the 75 % perceive one device but there are some individuals that start to perceive some echo.
- In 12 meters almost all the study group perceive a slight echo, but still their perception of the sound is satisfactory.
- In 16 meters the 75 % perceive two sound with different origin.
- Finally, in 20 meters, all the individuals hear two sounds and perceive an asynchronism between devices.

6.5 Discussion And Conclusions

As it has been discussed during this chapter and in Chapter 5, the Haas effect determines which are the maximum distance that is possible to cover since from a certain threshold it is impossible to achieve synchronization between devices due to the limitations of the human hearing. The main purpose of this test has been obtain the maximum distance allowed to do audio streaming with Nokia N810 as devices. Thus, if the user is inside the allowed zone, the synchronization problem is due to for the network and not for limitations of the human hear.

Once, analyzed the data obtained in the test it can be said that the results are more or less the expected result. It has to take into account that the this test is a subjective test and measure the perception of the human that it is different for each one, also it has to take into account other parameters like hearing ability, the place where it has been

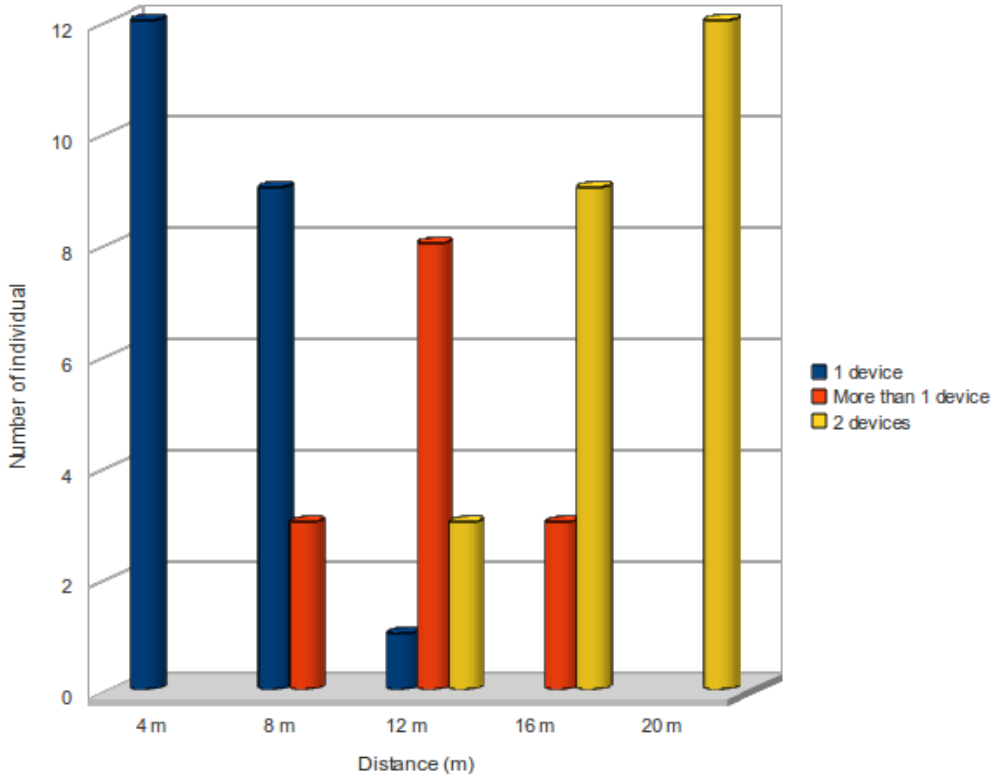

FIGURE 6.2: Test results

done the test, the devices power, between others.

Finally, it can been said that the maximum distance allowed, in this project, between devices where it is possible to controls the synchronization is 12 meters, more far of this distance the synchronization algorithm is useless due to the limitations in the hearing.

Chapter 7

REAL SCENARIOS

In this chapter it is described and analyzed three different real scenarios and the problems that it can come up. It has discussed about the two main problems that appears in a real time application and how they affect in the different scenarios. With this chapter the main idea is to do a briefly overview about the problems that are in real scenarios when it is needed to transmit multimedia files in real time, not only with the transmission but with the synchronization.

Firstly, it is done a description about the network synchronization in applications in real time, how many types of network synchronization there are, and a classification of the different techniques to solve it. Then, it has been explained the synchronization problems introduced by some acoustics properties and one possible solution to avoid this type of problems. Finally, it is described the three different scenarios and the different problems that can come up.

7.1 Network Synchronization

One of the most important aim when transmit multimedia in real time is the synchronization of the different Logical Data Unit(LDU) that is send over the network. Due to the temporary dependency between the different LDU it is necessary some techniques or algorithms to ensure the coordination between devices and the correct order in the streams when they are reconstructed in the receiver. Taking into account the temporary

dependency between the different streams is defined three different types of network synchronization: inter-stream synchronization, intra-stream synchronization and group synchronization.

The inter-stream synchronization is the simultaneous reproduction between sample of different streams. This synchronization takes places during the presentation and is referred to the synchronization between two different streams. The most important example of this type of synchronization is the lip-synchronization that is referred to the simultaneous reproduction of the audio and the lip motion. This type of synchronization is not important for the aim of the project since the only purpose is transmit audio.

The intra-stream synchronization is the responsible for maintain the correct order between samples from the same stream. This is more important in audio and video streaming due to the strong temporary restrictions between samples. To solve this type of synchronization is used two different fields of the RTP protocol, the "sequence number" and the "timestamp". The sequence number is used to number the LDU and to order the audio data into the receiver buffer. The timestamp identifies the sampling instant for the first byte of the LDU.

Finally the group synchronization appears when there is a multicast connection and the server send the data to several receivers, in this case it is needed to reproduce the stream simultaneously in all the devices. Due to the different bitrates a resynchronization during the transmission to ensure the correct synchronization between devices is needed. This point is more important when it is wanted to transmit multimedia data in real time to more than one device and there are a lot of people working in this type of simultaneity. This section shows a briefly classification of the main techniques, it can be found more information about the techniques in [15].

The techniques can be classified into four different categories according to [15]:

1. Basic control techniques: these type of techniques are essential to protect the temporary restrictions in the architecture.

2. Preventive control techniques: these techniques are used to prevent the asynchrony, this type of algorithms are used avoid the synchronization problems.

3. Reactive control techniques: in case of use this type of techniques it means that there are some problem. These techniques are used to solved the synchronization problems.

4. Common control techniques: these techniques can be used in both case. They can be used to avoid and to solved the synchronization problems.

All of these techniques can be used in the transmitter and in the receiver.

7.2 Acoustics Synchronization

Once solved the problem with the network synchronization there are other problem that have to be taken into account, the Haas effect. As it is explained in Chapter 5, from a certain temporary threshold the human brain is incapable to merge two identical sound of two sources in only one, perceiving the two sound as two sound of different sources.

One possible solution for this problem is to identify the position of the different devices and calculated the time between speakers, thus it is possible to delay the playback of the user to achieve a difference between sounds no higher than the maximum allowed. To calculated the location of the different devices some location algorithm can be used. Nowadays, these type of algorithms are researched due to the great number of applications that need to know the position of mobile devices in indoor scenarios. The location algorithms used different sensors like infrared, Bluetooth or Wi-Fi, between others, to calculate by means of several techniques the position of a mobile device. There are three main techniques [16]:

- Proximity based: in this technique the position of the mobile device is the position of the nearest beacon in the environment. In these types of techniques the accuracy depends on the number of beacons that there are in the environment, more beacons implies more accuracy.

- Time based: the time based techniques used the signal propagation between the mobile device and the beacon to calculated the distance and then calculated the position of the device. To have good accuracy it is needed expensive installations.

- Signal strength based: this approach is based on used the received signal strength index (rssi) that applying several algorithms it is able to determine the location of the user.

7.3 Scenarios Description

Next it has been explained the three different scenarios, which are the main elements and which are the main problems that it must to be solved.

- ***Scenario 1***

  The first scenario is composed by one server, several speaker and one user. In this case it has been considered that the users are so close that they can be considered as only one user, it has been defined as "so close" a difference between users not more than 1.72 m that is equivalent to 5 ms of delay in the sound between users. This delay is the maximum that it is accepted in order that a group of people do not differ in the perception of the sound.

  Next, in Figure 7.1 is shown a diagram of the second scenario described.

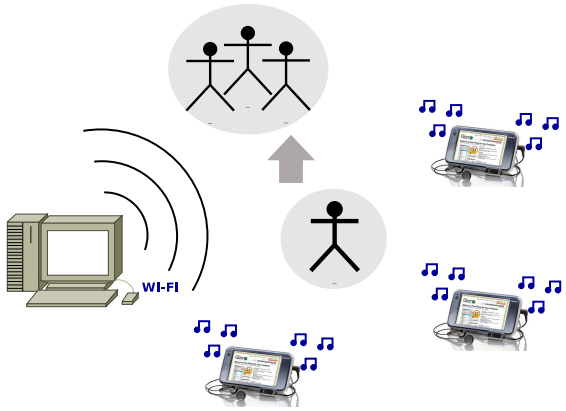

FIGURE 7.1: Scenario 1

In this type of scenarios are found both problems, the network synchronization

and the problems with the acoustics properties. As it is explained before, there are three different problems related with network synchronization, inter-stream, intra-stream and group synchronization, in this case there are intra and group synchronization problems. The intra-stream synchronization is not a problem using the RTP protocol since it provides some fields in the header to solved it, the group synchronization is more complicated and it is required some technique, like the techniques explained in Section 7.1, to solve it.

In the case of the acoustics properties, it can be use the location algorithms to calculated where are all the speakers respect the user, and decrease the time between speakers to achieve that all the sounds will arrive with a difference between them lower than the threshold allowed, that in the project case are 35 ms.

- ***Scenario 2***

The second scenario that is analyzed is composed by one user with one speaker working as a server and as client and several users, with their respective speaker, playing the audio. Nowadays, this scenario is impossible since the operative system of Nokia N810 is not able to code the data. Nevertheless, it is possible to simulated it with a PC or with the new version of Maemo. This type of scenario is a variant of the scenario 1 considering the worst case that is when all the user has also a speaker.

Figure 7.2 shown a scheme of the scenario number three.

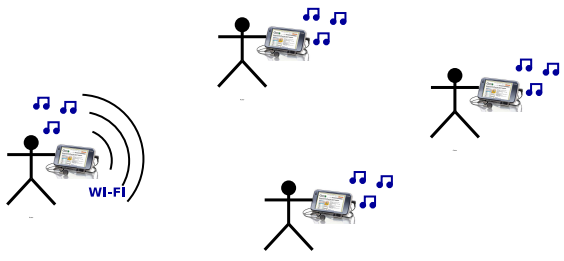

FIGURE 7.2: Scenario 2

As a variant of the scenario 1 this scenario has the same problem with the network synchronization. On the other hand, in this type of scenarios where there are involve several users it is impossible to solved the Haas effect since the fact to improve the perception of one user it implies worse perception for other user. The only solution is to keep a distance between users no higher than the maximum allowed.

- ***Scenario 3***

Finally, the third scenario consist on one server to transmit the audio, several users with their respective speaker playing the audio and also a main speaker playing the audio file. This kind of scenario can take place in a concert, between others, where the idea is to provide the possibility to play the music that are playing in the loudspeaker in your own device. Figure 7.3 shown the scenario described previously.

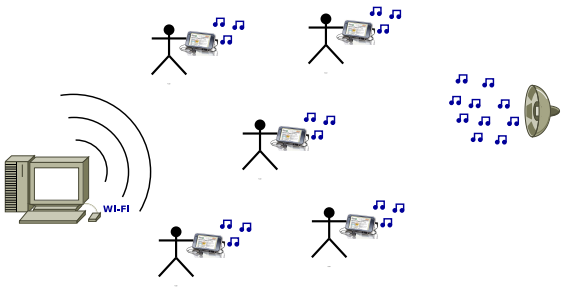

FIGURE 7.3: Scenario 3

As it is said, currently, the server must be a pc or other device able to encode the data since the operative system used by the tablet (Maemo Diablo) does not provide the libraries necessary to do that. Nevertheless, the new version of Maemo (Maemo Fremantle) provides all the libraries necessary to run the server in the device.

The main problems that it have to be solved in this scenario are the same that in the other two scenarios, the network synchronization and the Haas effect. In this case, as in the scenario 2 it is impossible to solve the Haas effect problems since there are several users in the scenario.

Chapter 8

CONCLUSIONS

Transmit audio files between several devices (Nokia N810) in real time has been the main aim of the project during the last three months. The idea has been improve the features of the Internet Tablet made by Nokia to give more facilities to customers.

Firstly, it was done a study about some features of the device such as the operative system, the audio codecs supported and the language needed to develop an application. In this way, it was defined the environment in which the project was going to be developed.

Once it is known the initial specifications, the next step was study the structure and the properties that must follow an application that used streaming technology. It was studied all the parts necessary to carry out the transmission in real time, describing all the audio encoders supported by the devices as well as the different Internet protocols that take part in the process. Finally, after evaluated all the possibilities it was decided use WI-FI connection in the physical layer, UDP as a transport protocol and RTP protocol to adapt the package to a real-time transmission.

The most important and complex part of the project was develop a server able to encode audio, encapsulate the data with all headers necessary and transmit the package over the network. To carry out the server it was decided use the GStreamer library, which provides us all the elements necessary to develop the server. Moreover, it was needed implement a client able to receive the packages and process it to the playback.

To develop the client it also was used some elements from the GStreamer library.

During the development appeared some difficulties when encode the audio data. First, it was decided implement an algorithm able to encode mp3 files to aac. The main problem was the algorithm complexity, so it was decided to used a library that included this functionalities due to encode data was not the aim of this project. The library used was Gstreamer, which is supported by the Nokia N810 development platform.

However, after develop the server and the client and during the installation of each one in the devices, it was detect that the currently version of Maemo do not support all the library functionalities. Specifically, the version of Maemo in the device is not able to encode any type of audio data. Due to this reason it was decided to used a PC as a server to simulated the project scenario.

When the server and the client was developed and the transmission could take place, it was analyzed the main problems that could come up. There were two main problems: the delay introduced by the network and an acoustics problems due to the human heard properties.

The acoustic problem is known as the Hass effect and it was studied by Helmut Haas. To test this type of acoustic effect it was performed a test to measure the maximum delay between several devices that the ear human can accept without perceive any distortion in playback. With this test it was obtained the maximum distance between devices that the human can accept, further from that distance is impossible that a human hears the playback from several devices synchronized.

Moreover it was studied the different problems that can appear with the synchronization of packages sent in real time to several devices. To study these problems it was analyzed three different real scenarios and what kind of network synchronization problems appear in each one. Finally, as future works it is propose some techniques to solved the synchronization problems that appear in the transmission.

Finally,nowadays it was developed a server and a client application able to transmit

audio files in real time and playback the data, respectively, using the Nokia N810 as a client and a PC or other device with the next version of Maemo as a server.

Chapter 9

FUTURE WORK

The main aim of this chapter is to explain the next step once the transmission of audio information in real time is carried out successfully. This chapter is focused in the synchronization problem studied in Chapter 7, that comes up when it is transmitted information in real time to several devices.

Next, it is explained some techniques that have been studied [15] to solved this type of synchronization problems in similar scenarios and can being implemented in the scenario simulated in this project. As it is explained in Section 7.1, the techniques used to solved the synchronization problems introduced by the network can be classified in four different categories: Basic control techniques, Preventive control techniques, Reactive control techniques and Common control techniques.

9.1 Basic Control Techniques

This type of techniques can be found in the transmitter and in the receiver, depending on where the techniques are developed it is needed to implement one thing or other, and the main aim is to maintain the temporary structure. If it is wanted to implement basic control techniques in the source the more common method is to introduce synchronization information, like timestamps or sequence numbers, in the different LDU that have to be sent. In the case of this project, this type of information are included in the RTP protocol that has two fields in the header for this purpose.

On the other hand, it can be created buffers in the receiver to control the synchronization problems. The buffers store the LDU received until its playback, according with some synchronization information. In this way, it can achieve to reduce the jitter effect in the network.

9.2 Preventive Control Techniques

This type of techniques are used to prevent problems with the synchronization. In this case, there are several methods to implement in the transmitter, the more common is to calculated the initial moment when the LDU must be reproduce and send this information to the several receivers before send the data. With this technique is achieved that all the receivers start the playback at the same time.

In the receiver, the method more used is to remove or insert some LDU in the buffer to adapt the time of the playback in all the receivers. In some codecs, like MPEG, it is possible to rule out some LDU less important according to the buffer occupation. Other technique commonly used to prevent problems with synchronization is to estimate the delay introduced by the network, if it is possible, and change the hold time in the buffer according with this time.

9.3 Reactive Control Techniques

In case that the synchronization appears the source can resynchronize the different data flows adjusting the transmission time. If the source is able to know the synchronization problem between data flows it can change the transmission period of the LDU. This type of techniques are more useful when it is needed to synchronize audio and video, in this project the aim is only to transmit audio so this type of techniques are not profitable for the purpose of the project.

In the receiver the technique most used to recover the synchronization consist on discard or repeat some LDU. For instance, in the case that a receiver detects that the

playback time of a LDU has past there are two solutions: execute the LDU and discard the consecutive LDU received, or discard the LDU directly. On the other hand, in case of underflow in the buffer, the receiver can reproduce the LDU repeatedly until receive the next. The main problem of these techniques is the quality, using these type of techniques, the quality of the playback can be affected. To solved the synchronization problems without affect the quality of the playback the receiver can reduce or extend the playback duration of each LDU until recover the synchronization. Reduce the playback duration implies a faster reproduction and extend the playback consist on decrease the speed of the playback.

9.4 Common Control Techniques

Finally, there are several techniques that can be used in both cases, to prevent or to solve problems with the synchronization. One of the method used in both cases consist on realize "hops" or "pauses" in the transmission, according with the feedback information sent by the receivers the source can send empty LDU ("hops") to adapt the transmission rate and the receiver rate. Moreover, there is a technique known as a Media Scaling that consist on adapt the transmission depending on the network conditions.

In the receiver the most used consist on adjust the playback rate changing the clock frequency of the device according to the synchronization received

Chapter 10

GANTT CHART

In this chapter is shown the Gantt Diagram of the project where it can see the different tasks carried out during the project development as well as a bar chart with the different steps followed.

In Figure 10.1 is described the main tasks of the project and start and end date of each task.

Next, Figure 10.2 shows the Gantt chart of the project, which illustrates the project schedule followed during the last three months. The Gantt chart is a type of bar chart that illustrate the different steps of the project in a visual way in a temporal axis.

Name	Begin date	End date
Mobile Distributed Wireless stereo Project	14/09/09	01/01/10
Study & Install Programming Environment	14/09/09	07/10/09
Study Streaming Architecture	07/10/09	03/11/09
Start Implementation	03/11/09	11/12/09
Develop Encode Audio	03/11/09	14/11/09
Develop Server & Client with GStreamer	16/11/09	11/12/09
Study Location Paper	24/11/09	26/11/09
Do the Test	11/12/09	15/12/09
Study Synchronization Techniques	14/12/09	17/12/09
Write the Report	21/09/09	01/01/10
Write Project Description	21/09/09	23/09/09
Write Programming Environment	07/10/09	10/10/09
Write Audio Streaming Chapter	03/11/09	10/11/09
Define & Write Test Definition	04/12/09	05/12/09
Finish the Report	17/12/09	01/01/10

FIGURE 10.1: Gantt Task

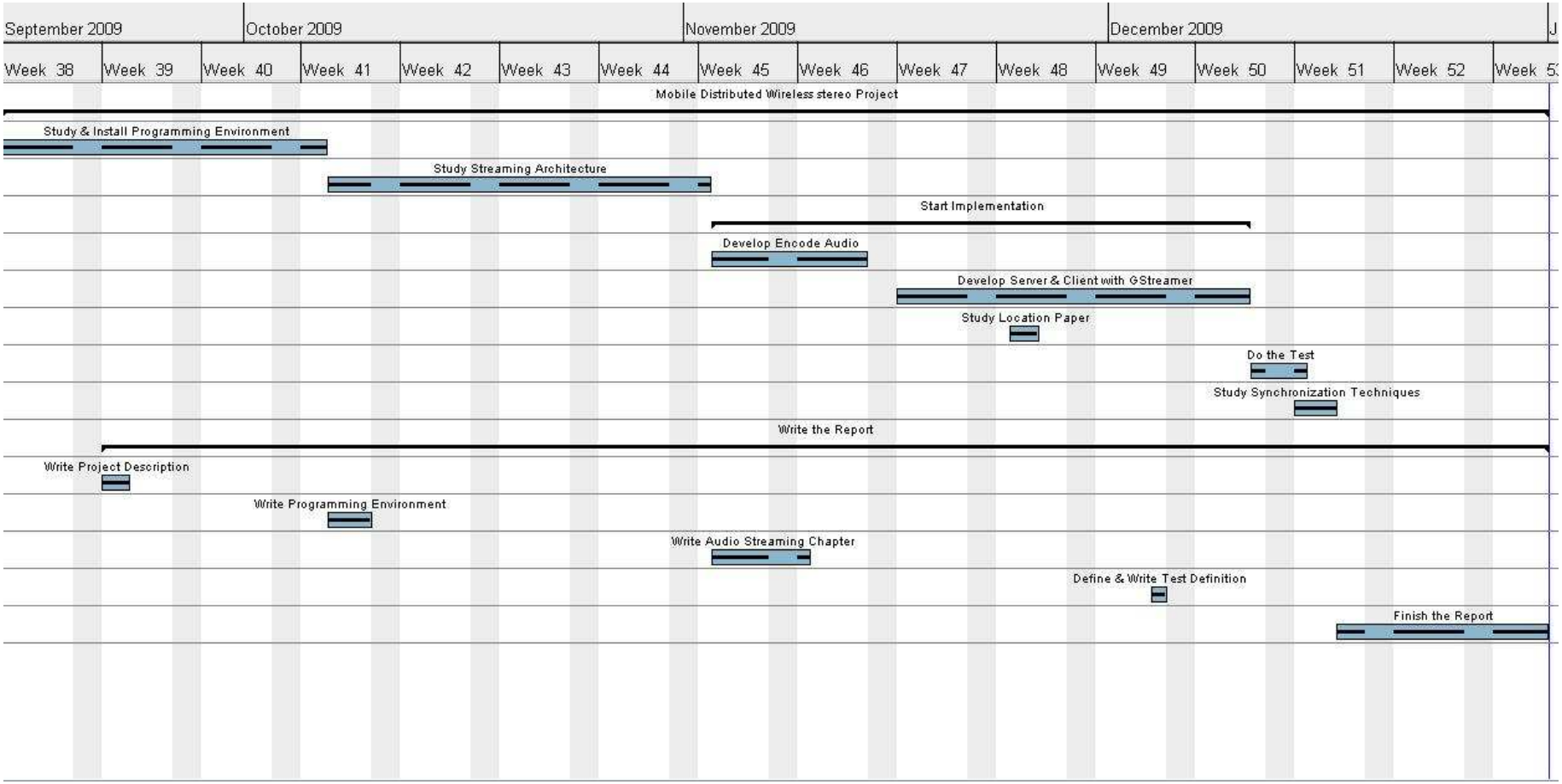

FIGURE 10.2: Gantt Diagram

Bibliography

[1] Nokia Community. Nokia europe - nokia n810 - support, 2009. URL http://europe.nokia.com/get-support-and-software/product-support/nokia-n810/specifications.

[2] Maemo Community. Intro: Software platform, 2009. URL http://maemo.org/intro/platform/.

[3] David Austerberry. *The Technology of Video and Audio Streaming*, pages 13–38, 99–125, 201–225. 1st edition edition, 2002.

[4] Stanley A. Gelfand. *Hearing: An Introduction to Psychological and Physiological Acoustics*. 4th edition edition, 2004.

[5] Brian C.J. Moore. *An Introduction to the Psychology of Hearing*. 5th edition edition, 2004.

[6] Davis Pan. A tutorial on mpeg/audio compression. Technical report, Motorola Inc., 1996. URL http://www.digital-audio.net/res/docs/pdf/mpegaud.pdf.

[7] H. Schulzrinne, A. Rao, and R. Lanphier. Real time streaming protocol (rtsp). Technical report, Network Working Group, 1998. URL http://www.ietf.org/rfc/rfc2326.txt.

[8] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Rtp: A transport protocol for real-time applications. Technical report, Network Working Group, 2003. URL http://www.ietf.org/rfc/rfc2326.txt.

[9] Eric A. Hall. *Internet core Protocols: The definitive guide*. 1st edition edition, 2000.

[10] J. van der Meer, D. Mackie, V. Swaminathan, and D. Signer. Rtp payload format for transport of mpeg-4 elementary streams. Technical report, Network Working Group, 2003. URL http://www.ietf.org/rfc/rfc3640.txt.

[11] Wim Taymans, Steve Baker, Andy Wingo, Ronald S. Bultje, and Stefan Kost. Gstreamer application development manual (0.10.24.3). URL http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/index.html.

[12] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Rtp: A transport protocol for real-time applications. Technical report, Network Working Group, 2003. URL http://www.ietf.org/rfc/rfc2326.txt.

[13] F. Alton Everest. *The master handbook of acoustics*, chapter Chapter 3. The ear and the perception of sound, pages 33–66. 3rd edition edition, 1994.

[14] Bruel & Kjar. Environmental faq - bruel & kjar, 2009. URL http://www.bksv.com/Support/FAQ/FAQ56.aspx.

[15] F. Boronat and J.C. Guerri. Análisis y comparación de algoritmos de sincronización entre flujos y de grupo de flujos multimedia. Technical report, IEEE, 2005. URL http://ewh.ieee.org/reg/9/etrans/ieee/issues/vol3/vol3issue5Dec.2005/3TLA5_04Boronat.pdf.

[16] Antonio Sapuppo, John Aasted Sørensen, and Birger Andersen. Experimental environment for verifications of wi-fi indoor location algorithms. Technical report, Wirelesscenter-CTIF-Copenhagen.

Appendix A

AUDIO STREAMING SERVER CODE

```
#include <stdio.h>
#include <gst/gst.h>
#include <glib.h>
#include <string.h>
#include <math.h>
#include <netinet/in.h>

#define DEST_HOST "10.0.0.28"
//#define UDP_PORT "54783"

#define host1 "127.0.0.1"
#define port1 "54002"


/* print the stats of a source */
static void
print_source_stats (GObject * source)
{
  GstStructure *stats;
  gchar *str;

  /* get the source stats */
  g_object_get (source, "stats", &stats, NULL);

  /* simply dump the stats structure */
  str = gst_structure_to_string (stats);
  g_print ("source stats: %s\n", str);

  gst_structure_free (stats);
```

```
30    g_free (str);
31  }
32
33  /* this function is called every second and dumps the RTP manager stats */
34  static gboolean
35  print_stats (GstElement * rtpbin)
36  {
37    GObject *session;
38    GValueArray *arr;
39    GValue *val;
40    guint i;
41
42    g_print ("**********************************\n");
43
44    /* get session 0 */
45    g_signal_emit_by_name (rtpbin, "get-internal-session", 0, &session);
46
47    /* print all the sources in the session, this includes the internal source */
48    g_object_get (session, "sources", &arr, NULL);
49
50    for (i = 0; i < arr->n_values; i++) {
51      GObject *source;
52
53      val = g_value_array_get_nth (arr, i);
54      source = g_value_get_object (val);
55
56      print_source_stats (source);
57    }
58    g_value_array_free (arr);
59
60    g_object_unref (session);
61
62    return TRUE;
63  }
64
65
66  int main (int argc, char *argv[])
67  {
68          GMainLoop *loop;
69
70          GstElement *pipeline, *source, *decoder, *audio_convert, *encoder, *rtppayload;
71          GstElement *rtpbin, *multiudpsink, *rtcpsink, *rtcpsrc;
72          gboolean link;
73          GstPadLinkReturn link1, link2;
74          GstPad *srcpad, *rtpsinkpad, *udpsinkpad, *sinkpad;
75
76          /* Initialisation */
77          gst_init (&argc, &argv);
```

```
78
79          loop = g_main_loop_new (NULL, FALSE);
80
81          /* Check input arguments */
82          if (argc != 2) {
83                  g_printerr ("Usage: %s <Audio filename>\n", argv[0]);
84                  return -1;
85           }
86
87          /* Create gstreamer elements */
88          pipeline = gst_pipeline_new ("audio-server");
89          source = gst_element_factory_make ("filesrc", "file-source");
90          g_assert(source);
91          g_object_set (G_OBJECT (source), "location", argv[1], NULL);
92
93          /*Add the source element into the pipeline*/
94          gst_bin_add_many (GST_BIN (pipeline), source, NULL);
95
96          /*the audio encoding and payloading*/
97          decoder = gst_element_factory_make ("mad", "mp3-decoder");
98          g_assert(decoder);
99          audio_convert = gst_element_factory_make ("audioconvert", "audioconverter");
100         g_assert(audio_convert);
101         encoder = gst_element_factory_make ("faac", "mp2-encoder");
102         g_assert(encoder);
103         rtppayload = gst_element_factory_make ("rtpmp4gpay", "rtppayload");
104         g_assert(rtppayload);
105
106         if (!decoder || !audio_convert || !encoder || !rtppayload) {
107                 g_printerr ("One element could not be created. Exiting.\n");
108                 return -1;
109         }
110
111         /*Add the audio element into the pipeline*/
112         gst_bin_add_many (GST_BIN (pipeline), decoder, audio_convert, encoder,
113         rtppayload, NULL);
114
115
116         /*the rtpbin element*/
117         rtpbin = gst_element_factory_make ("gstrtpbin", "rtpbin");
118         g_assert(rtpbin);
119
120         /*Add the rtpbin element into the pipeline*/
121         gst_bin_add (GST_BIN (pipeline), rtpbin);
122
123         /*the UDP sink used to transmit the RTP packet*/
124         multiudpsink = gst_element_factory_make ("multiudpsink", "multiudpsink");
125         g_assert (multiudpsink);
```

```
126         g_object_set (multiudpsink, "clients",
127         "192.168.135.215:55555,192.168.134.205:55555", NULL);
128
129
130         rtcpsink = gst_element_factory_make ("udpsink", "rtcpsink");
131         g_assert (rtcpsink);
132         g_object_set (rtcpsink, "port", 55556, "host", DEST_HOST, NULL);
133
134         rtcpsrc = gst_element_factory_make ("udpsrc", "rtcpsrc");
135         g_assert (rtcpsrc);
136         g_object_set (rtcpsrc, "port", 55558, NULL);
137
138         /*Add the UDP and RTCP element into the pipeline*/
139         gst_bin_add_many (GST_BIN (pipeline), multiudpsink, rtcpsink, rtcpsrc, NULL);
140
141         /*Link all the elements*/
142         link = gst_element_link_many (source, decoder, audio_convert, encoder,
143         rtppayload, NULL);
144         g_assert(link);
145
146         link1 = gst_element_link_pads(rtppayload,"src", rtpbin, "send_rtp_sink_0");
147         g_assert(link1);
148         link2 = gst_element_link_pads(rtpbin,"send_rtp_src_0", multiudpsink, "sink");
149         g_assert(link2);
150         udpsinkpad = gst_element_get_pad (multiudpsink, "sink");
151         if(udpsinkpad==NULL){
152                 g_printerr("There are an error in the udp socket\n");
153                 return -1;
154         }
155         g_assert(udpsinkpad);
156
157
158         /* get an RTCP srcpad for sending RTCP to the receiver */
159         srcpad = gst_element_get_request_pad (rtpbin, "send_rtcp_src_0");
160         sinkpad = gst_element_get_static_pad (rtcpsink, "sink");
161         link1 = gst_pad_link (srcpad, sinkpad);
162         g_assert (link1 == GST_PAD_LINK_OK);
163         gst_object_unref (sinkpad);
164
165         /* we also want to receive RTCP, request an RTCP sinkpad for session 0 and
166         * link it to the srcpad of the udpsrc for RTCP */
167         srcpad = gst_element_get_static_pad (rtcpsrc, "src");
168         sinkpad = gst_element_get_request_pad (rtpbin, "recv_rtcp_sink_0");
169         link2 = gst_pad_link (srcpad, sinkpad);
170         g_assert (link2 == GST_PAD_LINK_OK);
171         gst_object_unref (srcpad);
172
173
```

```
174         /* Set the pipeline to "playing" state*/
175         g_print ("Now playing: %s\n", argv[1]);
176         gst_element_set_state (pipeline, GST_STATE_PLAYING);
177 
178         /* print stats every second */
179         g_timeout_add (5000, (GSourceFunc) print_stats, rtpbin);
180 
181         /* Iterate */
182         g_print ("Running...\n");
183         g_main_loop_run (loop);
184 
185         /* Out of the main loop, clean up nicely */
186         g_print ("Returned, stopping playback\n");
187         gst_element_set_state (pipeline, GST_STATE_NULL);
188 
189         g_print ("Deleting pipeline\n");
190         gst_object_unref (GST_OBJECT (pipeline));
191 
192         return 0;
193 }
```

Appendix B

AUDIO STREAMING CLIENT CODE

```
#include <string.h>
#include <math.h>

#include <gst/gst.h>

/* the caps of the sender RTP stream.*/
#define AUDIO_CAPS "application/x-rtp, media=(string)audio, clock-rate=(int)44100,"
"encoding-name=(string)MPEG4-GENERIC,encoding-params=(string)2, streamtype=(string)5,"
"profile-level-id=(string)1, mode=(string)AAC-hbr, config=(string)0a10,"
"sizelength=(string)13,indexlength=(string)3, indexdeltalength=(string)3"

/* the destination machine to send RTCP to.*/
#define DEST_HOST "127.0.0.1"

/* will be called when rtpbin has validated a payload that we can depayload */
static void pad_added_cb (GstElement * rtpbin, GstPad * new_pad, GstElement * depay)
{
  GstPad *sinkpad;
  GstPadLinkReturn lres;

  g_print ("new payload on pad: %s\n", GST_PAD_NAME (new_pad));

  sinkpad = gst_element_get_static_pad (depay, "sink");
  g_assert (sinkpad);

  lres = gst_pad_link (new_pad, sinkpad);
  if ( lres == GST_PAD_LINK_WRONG_HIERARCHY){
        g_printerr ("ERROR: pads have no common grandparent\n");
  }
```

```
30    else if ( lres == GST_PAD_LINK_WAS_LINKED){
31          g_printerr ("ERROR: pad was already linked \n");
32    }
33    else if ( lres == GST_PAD_LINK_WRONG_DIRECTION){
34          g_printerr ("ERROR: pads have wrong direction  \n");
35    }
36    else if ( lres == GST_PAD_LINK_NOFORMAT){
37          g_printerr ("ERROR: pads do not have common format  \n");
38    }
39    else if ( lres == GST_PAD_LINK_NOSCHED){
40          g_printerr ("ERROR: pads cannot cooperate in scheduling \n");
41    }
42    else if ( lres == GST_PAD_LINK_REFUSED){
43          g_printerr ("ERROR: refused for some reason \n");
44    }
45
46    g_assert (lres == GST_PAD_LINK_OK);
47    gst_object_unref (sinkpad);
48  }
49
50  /* The difference between src and sink is that src=receive_data and
51  sink=send_data over the network*/
52
53  int main (int argc, char *argv[])
54  {
55          GstElement *rtpbin, *udpsrc, *rtcpsrc, *rtcpsink;
56          GstElement *audiodepay, *audiodec, *audiores, *audioconv, *audiosink;
57          GstElement *pipeline;
58          GMainLoop *loop;
59          GstCaps *caps;
60          gboolean res, link1, link2;
61          GstPadLinkReturn link;
62          GstPad *udpsrcpad, *sinkpad, *srcpad;
63
64          /* init first */
65          gst_init (&argc, &argv);
66
67          /* the pipeline to hold everything */
68          pipeline = gst_pipeline_new (NULL);
69          g_assert (pipeline);
70
71          /* the udp src and source we will use for RTP*/
72          udpsrc = gst_element_factory_make ("udpsrc", "rtpsrc");
73          g_assert (udpsrc);
74          g_object_set (udpsrc, "port", 55555, NULL);
75
76          /* we need to set caps on the udpsrc for the RTP data */
77          caps = gst_caps_from_string (AUDIO_CAPS);
```

```
78          g_object_set (udpsrc, "caps", caps, NULL);
79          gst_caps_unref (caps);
80
81          rtcpsrc = gst_element_factory_make ("udpsrc", "rtcpsrc");
82          g_assert (rtcpsrc);
83          g_object_set (rtcpsrc, "port", 55556, NULL);
84
85          rtcpsink = gst_element_factory_make ("udpsink", "rtcpsink");
86          g_assert (rtcpsink);
87          g_object_set (rtcpsink, "port", 55558, "host", DEST_HOST, NULL);
88
89          /* the rtpbin element */
90          rtpbin = gst_element_factory_make ("gstrtpbin", "rtpbin");
91          g_assert (rtpbin);
92
93          gst_bin_add_many (GST_BIN (pipeline), rtpbin, udpsrc, rtcpsrc, rtcpsink, NULL);
94
95          /* the depayloading and decoding */
96          audiodepay = gst_element_factory_make ("rtpmp4gdepay", "audiodepay");
97          g_assert (audiodepay);
98          audiodec = gst_element_factory_make ("faad", "audiodec");
99          g_assert (audiodec);
100
101         /* the audio playback and format conversion */
102         audioconv = gst_element_factory_make ("audioconvert", "audioconv");
103         g_assert (audioconv);
104         audiores = gst_element_factory_make ("audioresample", "audiores");
105         g_assert (audiores);
106         audiosink = gst_element_factory_make ("autoaudiosink", "audiosink");
107         g_assert (audiosink);
108
109         /* add depayloading and playback to the pipeline and link */
110         gst_bin_add_many (GST_BIN (pipeline), audiodepay, audiodec, audioconv, audiores,
111         audiosink, NULL);
112
113         /* now link all to the rtpbin, start by getting an RTP sinkpad for session 0 */
114         udpsrcpad = gst_element_get_pad (udpsrc, "src");
115         if(udpsrcpad==NULL){
116                 g_printerr("There are an error in the udp socket\n");
117                 return -1;
118         }
119         g_assert(udpsrcpad);
120         link1 = gst_element_link_pads(udpsrc,"src", rtpbin, "recv_rtp_sink_0");
121         g_assert(link1);
122
123         /* get an RTCP sinkpad in session 0 */
124         srcpad = gst_element_get_static_pad (rtcpsrc, "src");
125         sinkpad = gst_element_get_request_pad (rtpbin, "recv_rtcp_sink_0");
```

```
126         link = gst_pad_link (srcpad, sinkpad);
127         g_assert (link == GST_PAD_LINK_OK);
128         gst_object_unref (srcpad);
129         gst_object_unref (sinkpad);
130
131         /* get an RTCP srcpad for sending RTCP back to the sender */
132         srcpad = gst_element_get_request_pad (rtpbin, "send_rtcp_src_0");
133         sinkpad = gst_element_get_static_pad (rtcpsink, "sink");
134         link = gst_pad_link (srcpad, sinkpad);
135         g_assert (link == GST_PAD_LINK_OK);
136         gst_object_unref (sinkpad);
137
138         /* the RTP pad that we have to connect to the depayloader will be
139         *created dynamically so we connect to the pad-added signal,
140         * pass the depayloader as user_data so that we can link to it. */
141         g_signal_connect (rtpbin, "pad-added", G_CALLBACK (pad_added_cb), audiodepay);
142
143         res = gst_element_link_many (audiodepay, audiodec, audioconv, audiores,
144         audiosink, NULL);
145         g_assert (res == TRUE);
146
147         /* set the pipeline to playing */
148         g_print ("starting receiver pipeline\n");
149         gst_element_set_state (pipeline, GST_STATE_PLAYING);
150
151         /* we need to run a GLib main loop to get the messages */
152         loop = g_main_loop_new (NULL, FALSE);
153         g_main_loop_run (loop);
154
155         g_print ("stopping receiver pipeline\n");
156         gst_element_set_state (pipeline, GST_STATE_NULL);
157
158         gst_object_unref (pipeline);
159
160         return 0;
161 }
```