Distributed Stereo on Mobile Devices

Synchronized audio streaming on Symbian Platform

Mobile Communication/
P9-P10 project, spring semester 2010
Faculty of engineering, health care and natural sciences
Institute for Electronic Systems

Nicandro SCARABEO

AALBORG UNIVERSITET

Aalborg University
Departement of Electronic Systems
Frederik Bajers Vej 7
9220 Aalborg O
Telephone 99 40 86 00
http://es.aau.dk

Title: Distributed Stereo on Mobile Devices - Synchronized audio streaming on Symbian Platform

Semester Theme : Master Thesis

Project term : P9, fall semester 2009 - P10, spring semester 2010

Author: Nicandro SCARABEO

Supervisors : Frank Fitzek
Morten V. Pedersen
Janus Heide

Number of copies : 5

Number of pages: 95

Completed : June 2010

Synopsis:

The purpose of this project was to realize a distributed and synchronized stereo reproduction between more mobile devices: the possibility for a mobile phone to share and stream an audio file with other phones, by starting a synchronized reproduction.

For this purpose an application was built from scratch in order to solve the several problems characterizing this kind of application. It was written a PYTHON code for the management of the Application Layer of ISO/OSI protocols. It allows to stream in real time a PCM file playing on one device by building an ad hoc network. The reliability of the network is discussed and analyzed: the interleaving technique and a coding scheme is suggested to achieve that.

Preface

This report is separated into these two items :

Item 1: The main report on the analysis, simulation and explanation of a proposed antenna structure that includes metamaterials.

Item 2: Appendices in the end of the report

Reading directions

Sources are given by [number], e.g. [10], referring to an index number in the sources list.

Acknowledgements

In connection with this report the author would like to express his thankfulness to Frank Fitzek, Professor at Aalborg University, Morten V. Pedersen and Janus Heide, PhD students at Aalborg University, for the availability, explanations and the invaluable support during all these months. The author would also like to express his appreciation to Peter Vingelmann, for the collaboration, and to Damiano Ciarla and Samantha Caporal Del Barrio who helped during important moments.

Nicandro SCARABEO

.....

Contents

Chapter 1

INTRODUCTION

1.1 Motivation

Since the invention of the radio at the beginning of the twentieth century, continuing 40 years later with the invention of the television, an enormous flux of information started to reach a lot of people in new different ways. Next to newspapers, these new means of communication brought to the people information in the form of voice, images and video. Then, in the end of the century, with the coming of the computer, and the introduction of internet, the streaming of data increased more and more, determining a new virtual world that nowadays is part of our life. The phenomena, that represents the trend, is becoming undoubtedly the peer to peer (P2P), subsequently to a period in which the client – server architecture were bringing these services to the end – users.

Nowadays, entire young generations, and not only, exchange any kind of data by making a portion of their resources (such as processing power, disk storage or network bandwidth) directly available in a distributed network architecture composed by peer nodes. A model in which each component of the network can be a server or a client, by depending on its needs in each moment. Such a phenomena is widely spread between fixed devices, as personal computers or laptops, and overall on fixed and reliable networks. On this big wide architecture many users exchange the last hits of the music parade or the last episodes of their favorite tv-shows, to transfer them after on the mobile devices of last generations. Actually, the quick spreading of MP3 players, IPod or IPad, suggest how the trend of downloading and listening music is involving mobile devices.

Despite that, P2P is not yet widely used on portable/mobile devices. Actually people are using it on laptops, but the mobile phone is a different kind of device which plays a different role in our daily life: it is always on, it is carriable everywhere, etc. Then downloading a song or sharing a video with friends is still a concept related to computers rather than mobile phones. The main obstacle is represented by the lack of a standard in such world and the lack of resources necessary to implement it. This project wants to show how this obstacle can be

broken.

Figure 1.1: By implementing P2P between mobile phones, sharing audio, video and files will become easier, faster and cheaper.

1.2 Goal of the project

The same kind of architecture, created to allow terminal to exchange data, can be build between mobile phones. In this way, new services will be available on them: the possibility to listen the same music on more mobiles without need to have a big loud speaker or sharing a video between more devices without need of a big screen. All of this is possible by using the large potential of a group of phones working together, withouth the help of any kind of architecture but WLAN or Bluetooth. The project wants to follow the principles expressed by the current research that involves the new types of resource-sharing networks called wireless grid [10]. It consists in increasing the potential of the devices by sharing network connected resources via ad hoc wireless networks. A philosophy that allows the vision of a new kind of architecture: not anymore a single mobile device, but a network of those able to interact. In such systems, a wide variety of possible applications can grow up. Actually, they can reach geographic locations and social settings that computers have not traditionally penetrated. In addition to the typical computational resources present on the computers, such as processor power or disk space, new interactive and multimedia applications such as cameras, microphones, GPS receivers, are available on the mobile devices. Those represents the basis on which the fantasy of any programmer can break out. Our prototype wireless grid application demonstrates the practically of this theoretical approach.

The purpose of this project is then to realize a distributed and synchronized stereo reproduction between more mobile devices. A phone, owner of a song file, can decide to share and stream it to other devices by building a reliable network

architecture (Figure 1.2 shows how it should look like). Once the stream will have been started, the music will be played on each device in synchronization. This application wants to eliminate disadvantages such as the impossibility to have a higher-quality stereo sound in a specific background and wants to be the first step of new applications that can follow the same algorithm and logic.

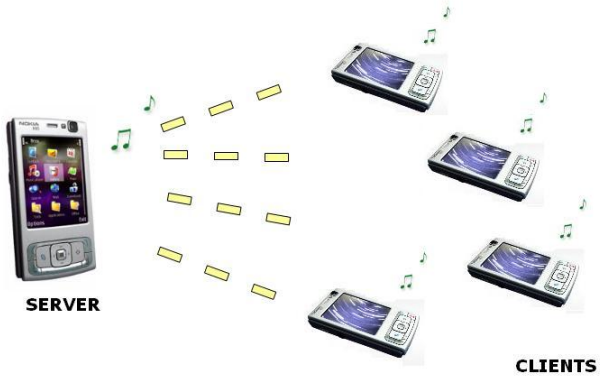


Figure 1.2: The purpose of the project is to manage such a scenario

1.3 Problem definition

In order to build a mobile application, three main requirements are requested:

- the application must be **suitable** for the platform where it has to run;
- the algorithm must have a **low complexity**;
- the communication between the node must be **reliable** enough in order to assure a good quality of the service.

To achieve the first request, it is necessary an investigation about technologies more suitable for such mobile application: from physical layer over network layer to application layer. Several examples about previous mobile implementations must be checked.

To achieve the second request, it is necessary to look for a programming language able to build an audio streaming connection, and at the same time, able to keep low the complexity of the instructions necessary in performing it.

To achieve the third request, an audio format, able to be streamed, has to be found. Besides, to achieve reliability the use of a coding scheme is necessary. Because of that, some existing coding schemes must be analyzed in order to find the best for such implementation.

1.4 Approach

This report introduces a mobile application that is running on the Symbian/S60 platform used on most Nokia smartphones. The application allows audio streaming through an Ad hoc Network established between mobile devices.

After analyzing several existing applications, like Picture Viewer ([19]), a similar architecture has been implemented. Even though it is necessary an investigation about all the ISO/OSI layers, this work will focus mainly on the Transport Layer and Application Layer. In regard to the forth layer, from the bottom, some tests has been performed in order to reduce the packet loss; in regard to the fifth, an algorithm from scratch has been implemented in order to follow the main principles of an RTP application.

The low complexity has been achieved by using a high level programming language and by using an already existing module able to reproduce chunks of sound. In this first implementation, PCM files are streamed. This choice has been done to better knock down possible errors incoming in the connection.

In the end, in order to have a communication as much as possible reliable, some techniques as interleaving and an algorithm using coding schemes, as Erasure Coding and Network Coding, is suggested.

1.5 Problem analysis: summary of the report

The problem analysis of such problem is definitely hard to define and solve in few lines. Because of that, in this first chapter, it is reported just a summary of the topic treated and the problems solved. Actually, each specific problem is deeper treated subsequently in the report.

Chapter 2: Choice of the protocols

One of the first problems necessary to solve is the choice of the protocol to use in the ISO/OSI protocol between the ones available on the phone chosen to test it: Nokia N95.

Which is the Data Link Control layer more valid to implement a multicast connection between many devices? A disquisition about the main characteristics of Bluetooth and Wi-Fi 802.11 b/g is presented in Chapter 2.

Afterwards a similar analysis is presented for the choice of the transport layer. **Which transport layer is more suitable for a real-time application between UDP and TCP?**

Having defined the first four layers of the protocol, the fifth will be proposed and **built from scratch** in order to achieve useful functions typical of **real-time transport layer** by avoiding the complexity of already existing protocols.

Chapter 3: UDP performances in Ad Hoc wireless

Before implementing an application is necessary to describe the platform where is supposed to run, and the programming code suitable for it and easy to use.

Several tests have been performed **to figure out the real capacities of the mobile devices** chosen for testing the application.

It has been proved the good efficiency of such devices by **choosing a right size for the unit packet**, to avoid fragmentation in the network, and in a range demanded by the application.

Chapter 4: Audio engine

In this chapter it will be presented an investigation on the way to reproduce sound from the mobile phone. A first step will involve the choice of the suitable codec and it will be explained why in this first implementation it is necessary to exclude some audio formats.

Aftewards it is presented the module chosen to reproduce sound and why it has been preferred to others. It will be pointed out the use of the player in relation to the fragmented arrival of the packet from the network.

How it is solved the problem of missing packet in the reproduction?

Which is the best strategy to buffer packet to avoid sudden break in the playing time?

How to avoid problems deriving from bursty losses?

Chapter 5: Synchronization

In this chapter the synchronization is discussed and analyzed in details. First theoretically **three scenarios are analyzed** to figure out whether the application can work in such background.

A first scenario is represented by an only person between two speakers.

The second scenario is represented by a couple of person, each one with a mobile phones. **Can we achieve a synchronization in such scenario?**

The third one is representing an evolution of the second: many people owners of their devices in a background characterized by the presence of a big loud speaker.

After discussing those, **an algorithm to assure a synchronization** between an hypothetical infinite number of phone **is presented** to work in a range of 10 meters, as requested from Hass's studies about the capacity of the human hears. The algorithm based on the equality between sending rate and playing rate needs a low latency of the channel to work well.

Because of that, **a test about the latency has been performed.**

Chapter 6: Reliable multicast in ad hoc networks.

In this chapter **a method to achieve a better performance** is figured out.

Previously in the report, it was discussed how the music could sound good by using some linear audio formats even though the presence of losses in the communication. In this chapter coding schemes are analyzed to preserve the audio streaming in case of other audio formats.

ARQ, Erasure Coding and Network Coding are analyzed.

By keeping count of the previous analysis **a new protocol is proposed with the possibility to switch between more coding schemes** depending on the topology of the network.

1.6 A multimedia world: the future has arrived

Only when a "technology" arrives to the masses, it has the duty and the right to be defined like that. The developing of studies and researches about the radio wave transmissions are on since years already, since Guglielmo Marconi started his study in the middle of the nineteenth century . Despite that, many people, nowadays, still consider those something abstract limited to some circumscribed sector. This application wants to prove how that tool, as medium of sharing multimedia contents, can enter strongly in the daily life.

"The progress is real only if the advantages of a technology are available to everybody"

Henry Ford(1863 - 1947)

To achieve that, this application wants to allow people to use their smartphones as tool of new resources without spending other money. Its success is assured by giving a look to the world of today: there is no child that does not ask to his parents to buy the last platform game or the last mobile phone. Many teenagers spend a lot of money to buy IPOD to be able to listen their favorite music everywhere. Besides, the same success of the site Youtube on Internet proves how the way traced from this project is the way to follow in the next ten years by the new technologies. This project represents what the trend suggests to the market.

Chapter 2

CHOICE OF THE PROTOCOLS

Before analyzing in details the project, by considering the ISO/OSI architecture, it is necessary to analyze and describe which protocol should be used on the mobile device on which the application will be tested. Afterwards, each choise in the next sections will be discussed and the protocol stack will be completed.

2.1 Nokia N95 and its available protocols

The application projected in this report will be tested on several NOKIA N95, a mobile phone released by Nokia in March 2007. In the next table the main specifications are shown in order to justify some decisions taken in the implementation of the application.

Figure 2.1: Nokia N95

GENERAL	2G Network	GSM 850 / 900 / 1800 / 1900
	3G Network	HSDPA 2100, HSDPA 850 / 1900
SIZE	Dimensions	99 x 53 x 21 mm, 90 cc
	Weight	120 g
FEATURES	OS	Symbian OS 9.2, S60 rel. 3.1
	Messaging	SMS, MMS, Email, Instant Messaging
	Sound	Yes, with stereo speakers
	Music Player	MP3/WMA/WAV/RA/AAC/M4A
MEMORY	GPRS	Class 10 (4+1/3+2 slots), 32 - 48 kbps
	EDGE	Class 32, 296 kbps; DTM Class 11, 177 kbps
	3G	HSDPA
	WLAN	Wi-Fi 802.11 b/g, UPnP technology
	Bluetooth	Yes, v2.0 with A2DP
	Infrared port	Yes
	USB	Yes, v2.0 miniUSB
BATTERY		Standard battery, Li-Ion 950 mAh (BL-5F)
	Stand-by	Up to 220 h (2G) / 192 h (3G)
	Talk time	Up to 6 h 30 min (2G) / 2 h 42 min (3G)

Table 2.1: Specification of NOKIA N95.

2.2 ISO/OSI

In this section the Open System Interconnection Reference Model is described. It is divided in seven different layers, following a precise structure and strict rules. The layers are listed in the next image.

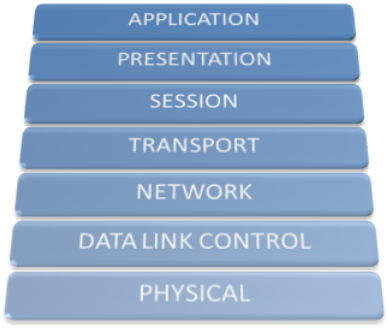


Figure 2.2: The OSI Model

Before describing in details the layers, it is necessary to talk about the principles applied to realize this schema. There is one layer for every different

abstraction or function that is part of the communication. A layer must provide services to the layer above it and receives service from the layer below it. Each level has to be big enough to avoid the presence of sub levels in it, but it has to be not so small, because, anyway, it is necessary a sort of architecture to realize it. The main purpose of this kind of architecture is to have such as "entities", able to communicate, through protocol, with "peer entities" situated in other end-points.

Shortly the main issues for the layers are:

- Addressing
- Error Control
- Flow Control
- Multiplexing
- Routing

After this overview of the ISO/OSI protocol, each layer will be analyzed in detail to explain its role. To do that, it has been refered the [3]

2.2.1 Physical Layer

The main task of the physical layer is to transmit raw bits over a communication channel. Typical questions here are:

- how many volts should be used to represent 1 and 0
- how many microseconds a bit lasts
- whether the transmission may proceed simultaneously in both directions
- how the initial connection is established and how it is turn down
- how many pins the network connector has and what each pin is used for.

The design issues deal with mechanical, electrical, and procedural interfaces, and the physical transmission medium, which lies below the physical layer.

2.2.2 The Data Link Layer

The main task of the data link layer is to take a raw transmission facility and transform it into a line that appears free of undetected transmission errors to the network layer. To accomplish this, the sender breaks the input data into data frames (typically a few hundred or a few thousand bytes), transmits the frames sequentially, and processes the acknowledgment frames sent back by the receiver. The issues that the layer has to solve:

- to create and to recognize frame boundaries - typically by attaching special bit patterns to the beginning and end of the frame
- to solve the problem caused by damaged, lost or duplicate frames (the data link layer may offer several different service classes to the network layer, each with different quality and price)
- to keep a fast transmitter from drowning a slow receiver in data
- if the line is bi-directional, the acknowledgment frames compete for the use of the line with data frames

Broadcast networks have an additional issue in the data link layer: how to control access to the shared channel. A special sub layer of the data link layer (medium access sub layer) deals with the problem.

2.2.3 The Network Layer

The main task of the network layer is to determine how data can be delivered from source to destination. That is, the network layer is concerned with controlling the operation of the subnet. The issues that the layer has to solve:

- To implement the routing mechanism
- To control congestions
- To do accounting
- To allow interconnection of heterogeneous networks
- In broadcast networks, the routing problem is simple, so the network layer is often thin or even nonexistent

2.2.4 The Transport Layer

The basic function of the transport layer is to accept data from the session layer, split it up into smaller units if need be, pass them to the network layer, and ensure that the pieces all arrive correctly at the other end. All this must be done in a way that isolates the upper layers from the inevitable changes in the hardware technology. The issues that the transport layer has to solve:

- To realize a transport connection by several network connections if the session layer requires a high throughput or multiplex several transport connections onto the same network connection if network connections are expensive
- To provide different type of services for the session layer
- To implement a kind of flow control.

The transport layer is a true end-to-end layer, from source to destination. In other words, a program on the source machine carries on a conversation with a similar program on the destination machine. In lower layers, the protocols are between each machine and its immediate neighbors.

2.2.5 The Session Layer

The session layer allows users on different machines to establish sessions between them. A session allows ordinary data transport, as does the transport layer, but it also provides enhanced services useful in some applications.

2.2.6 The Presentation Layer

The presentation layer performs certain functions that are requested sufficiently often to warrant finding a general solution for them, rather than letting each user solve the problem. This layer is, unlike all the lower layers, concerned with the syntax and semantics of the information transmitted. A typical example of a presentation service is encoding data in a standard agreed upon way. Different computers may use different ways of internal coding of characters or numbers. In order to make it possible for computers with different representations to communicate, the data structures to be exchanged can be defined in an abstract way, along with a standard encoding to be used "on the wire". The presentation layer manages these abstract data structures and converts from the representation used inside the computer to the network standard representation and back.

2.2.7 The Application Layer

The application layer contains a variety of protocols that are commonly needed.

For example, there are hundreds of incompatible terminal types in the world. If they have to be used for a work with a full screen editor, many problems arise from their incompatibility. One way to solve this problem is to define network virtual terminal and write editor for this terminal. To handle each terminal type, a piece of software must be written to map the functions of the network virtual terminal onto the real terminal. All the virtual terminal software is in the application layer. Another application layer function is file transfer. It must handle different incompatibilities between file systems on different computers. Further facilities of the application layer are electronic mail, remote job entry, directory lookup ant others.

2.3 Problem analysis: Data Link Control Layer

In order to build our application and to write a protocol to reach the goal of the project, it is necessary to start from the bottom of the ISO/OSI architecture and analyze layer by layer which technology we are going to use. By following

the purpose of the project, to transmit data between phone it is necessary a wireless network. The IEEE 802.11 and the Bluetooth technology are the most common technologies in this field. They define a physical and a MAC (Media Access Control) layer for communications with short distances (0-100m) and low transmitting power (1mW – 800 mW). Usually Bluetooth is oriented to connections between close devices, as a way to substitute cables to transport data, while IEEE 802.11 is oriented to connections between computers (terminals) as a way to substitute older wired LAN. Before making a comparison and an analysis of these, it is necessary to give a look to their main characteristics.

2.3.1 The Bluetooth technology

Bluetooth is 802.15 technology in order with the standard of the IEEE. It works at 2.4 GHz of frequency. By using Bluetooth technology it is possible to realize such a small net. Usually these contain not more than eight elements and are called PICONET. Typical rate for such a net is about 1 Mbps with a maximum output power of 2.5mW for devices like notebook or phones. Considering the GSM's, the power used by this technology is quite low.

Figure 2.3: The Bluetooth technology is spread in several application

The Bluetooth technology is not just a standard for the first two layers of the ISO/OSI architecture, but it is a protocol stack with all the layers described before.

Bluetooth is an important means to realize WPAN (Wireless personal area network). Such a net is bigger not more than few meters and it make link between several kind of devices like notebook, phones, laptop, printer, etc.

Of course, a printer cannot deal with an audio streaming problem, so a protocol is requested to deal with all the different services that a generic device can

offer in a piconet. This kind of protocol is called SDP (Service Discovery Protocol). This protocol uses packets (PDU) to send requests and receive responses. This protocol can ask for all the services of a specific device ("Browsing" mode) or it can look for a specific service in a group of several devices ("Searching" mode). A device can ask and answer, assuming in different time the position of server or client.

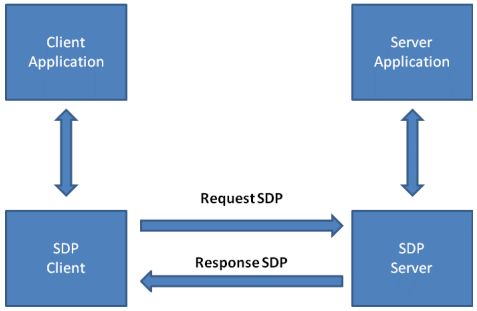


Figure 2.4: Interaction between a server and client in a PICONET

As already mentioned, Bluetooth works in the spectrum 2.402 - 2.480 GHz, subdivided in 79 sub channels of 1 MHz. This technology uses the FHSS technique (Frequency Hopping Spread Spectrum). It consists of a change of the sub channel in regular time space. So, for example, in a communication, a transmitter can change carrier even 1600 times in a second. Next it is explained why Bluetooth uses this technique: working in a free license frequency, a lot of interference can infect the information-signal, so, changing frequency, it is possible to solve this problem. This process, then, gives a kind of stability to all the system.

By using the "Frequency Hopping", it is necessary a device which can synchronize these hopes and manage the whole transmission, choosing the sequence of the use of the frequency. This "special" device is called "Master". Then, a piconet is composed by a master and not more than seven "slaves". An example of that is shown in the next figure. Thinking about the time of communication as divided in several time slots, the master communicates in the even time slot, the slaves in the odd one. In a piconet just eight devices can be activated but it is possible the physically presence of others devices, called "park", that simply do not share the channel, not participating in the communication.

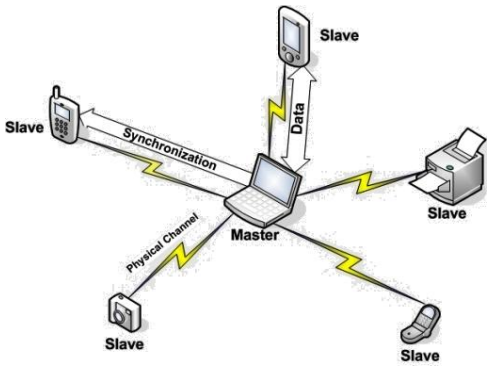


Figure 2.5: A possible example of a PICONET

How to realize a net with more than 8 devices? Is it impossible? No, it is not. "Scatternet" is the name of bigger net, composed by more than one piconet. In this case, every "master" of a piconet, is a "slave" in a higher level net. It is easy to understand that the maximum number of devices which can use the same channel cannot be bigger than 79, because of the dimension of the channel.

As already mentioned, the communication in a Bluetooth technology is divided in time slots, in which the devices exchange information by using packets not bigger than 2871 bits. In order to have a good communication every packet is composed by 3 specific parts: AC (Access Code) of 72 bits, an H (Header) of 54 bits and a Payload (data) that can reach the dimension of 2745 bits. In a higher ISO/OSI level, these packets are managed by the protocol L2CAP (logical link control adaption protocol).

Figure 2.6: The structure of a Bluetooth packet

By looking in details at the stack protocol of Bluetooth, two blocks manage the MAC layer: the Link Manager and the Link Controller. The first one installs the connection and sends commands to the Link Controller, whereas the second one manages the sending of the packets and holds on the connection. Instead, the physical layer is composed by the Baseband and the Radio Channel.

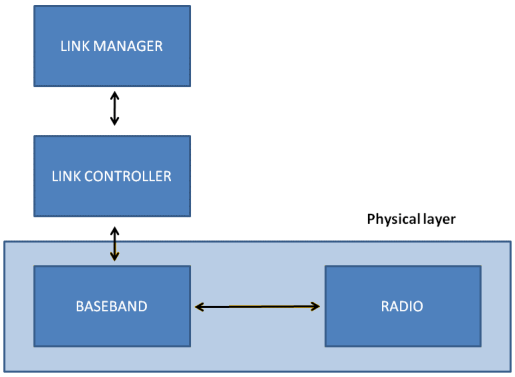


Figure 2.7: The structure of the MAC layer in the Bluetooth packet

Each device, in which Bluetooth technology is installed, has a different MAC address. In this case, we can talk about a BD_ADDR (Bluetooth device address) of 48 bits. It is divided in three parts:

- LAP: 24 bits needed for the sync word and for the frequency hopping
- UAP: 8 bits needed for the error detection and the CRC (cyclic redundancy check)
- NAP: 16 bits needed for giving information about the cryptography.

2.3.2 The IEEE 802.11 technology

The IEEE 802.11 defines the functions and services for devices (often are terminals) that want to use such a wireless network to transmit data. It defines a MAC layer and technique of transmission of signals in a Physical layer by IR (infrared) or RF (Radio Frequency). This technology is "packet oriented". It is important to underline that, above the MAC layer, IEEE 802.11 and a normal IEEE wired LAN are indiscernible. So the mobility of this standard is managed by the MAC layer.

IEEE 802.11 works in the ISM band. It is necessary to specify that several kind of this technology exist. IEEE 802.11b/g is the one available on the phone. 802.11g is the newest and, where available, can offer the best performances: 54Mbps at 2.4Ghz. 802.11b nowadays is a restriction of the "g" one: it can reach nominally at 22Mbps, but usually its real rate is bounded at 11Mbps. This protocol has been known as "Wi-Fi". As much as famous, overall in North America, is the IEEE 802.11a. This standard works at 5 GHz (another unlicensed frequency) and it can reach 54 Mbps. In this report such a protocol cannot be take in consideration because on the phone is not available.

IEEE 802.11 is organized with a cellular architecture. Two or more devices, called STA, can compose a network called BSS (Basic Service Set). They are coordinated by the MAC layer with a CF (Coordination Function). More BSS (called Independent BSS if they are networks ad hoc), can make a bigger network, called ESS (extended service set). An example of that is shown in the next figure. These different BSS are linked by distribution system (DS) and for each BSS, a STA have to be linked with this DS to hold on the communication. This coordinated station is called Access Point (AC).

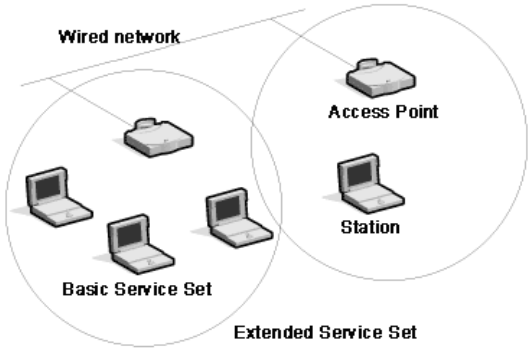


Figure 2.8: A possible example of a Extended Service Set

How can different STAs communicate? Talking about the physical layer, different techniques are known:

- IR (Infrared)
- FHSS (Frequency Hopping Spread Spectrum)
- DSSS (Direct Sequence Spread Spectrum)

The third one is the most used, instead of the first one that it is very useful just for not so long distances (about 10 m without obstacles). DSSS is a wideband transmission technique and it consists of spreading information on bigger part of spectrum to contrast interference. Actually, every single bit is modulated by a particular sequence (Barker's) composed by 11 bits, called chips. Because of this, the modulation is called chipping code.

As already done with the Bluetooth Technology, it is useful to describe how the MAC layer of IEEE 802.11 is organized. The MAC manages and holds the connection between STAs, coordinating the net. The mechanism of access in the medium is called Distributed Coordination Function (DCF), and it is based on the technique of CSMA/CD (Carrier Sense Multiple Access with Collision Detection). Every station, before transmitting, has to check if someone else is doing that. If the medium is free, it can transmit to a different station. This

one, after receiving the packet, will send an ACK (acknowledgment) to inform about the right reception. The PCF (Point Coordination Function) works just in such a net not ad hoc. It establishes a different method to access to the net.

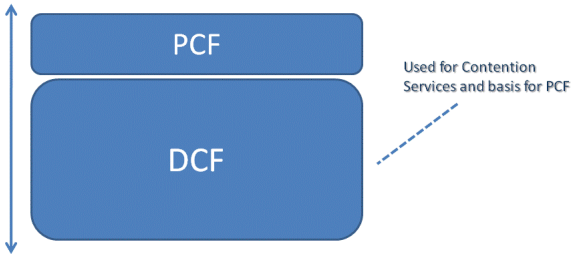


Figure 2.9: The structure of the MAC layer in the IEEE 802.11

An important ability of the MAC IEEE 802.11 is the possibility to fragment a packet if it is too long. That is done to make low the traffic in the net. Every packet will be anyway composed by three parts:

- MAC Header: control information, addresses of the sender and recipient
- Frame Body: if it is a data packet, it is just a payload
- CRC : 32 bits for errors detection.

Figure 2.10: The structure of a IEEE 802.11 packet

Every packet is known through a MAC address of 48 bits. It can be:

- Single address: to refer about one single STA
- Group address: to refer about a multi destination.

2.3.3 Bluetooth versus IEEE 802.11: what should our application use?

After describing the main characteristics of these protocols, in this section a full comparison between these has done. In the end, it will be explained which technology has been chosen and why.

Capacity

With reference to the typical basic net of these two technologies, the Piconet for Bluetooth can be composed by not more than 8 devices whereas the BSS (Basic Service Set) for IEEE 802.11 virtually has an unlimited capacity. It is however possible to link more basic net to have a bigger kind of net: ESS for IEEE 802.11 and Scatternet for Bluetooth.

Arrangement of a network

A parameter quite important is how fast a link between two or more devices is established. In regard to the Bluetooth technology, two procedures are expected: Inquiry and Page. In the "Inquiry" phase, the master scans the environment, looking for MAC address of new potential slaves, whereas in the "Page" phase the master inserts a new slave in the piconet. With reference to the IEEE 802.11 three procedures are planned: Scan, Authentication and Association. The first phase is necessary to know the MAC address of any device, interested to enter in the BSS. This can happen in two different modes: passive or active. In the passive one, it is request a time of 50ms per each channel investigated, whereas in the active one, a new device has to earn the access to the net by following the rules of the CSMA/CD. The phase of authentication and association to the BSS are necessary only if the net is a net of hoc.

Topology

The Piconet and the BSS have a lot of analogies: both are managed by a central entity (master for Bluetooth and access point (AC) for the IEEE 802.11); they make the routing of the packets into the net. The coverage of a piconet is 10m whereas BSS's can reach 150m. Considering the mobility of devices during the transmission, for our project is more indicated the second one. Another point in favor of the IEEE technology is the impossibility of communications between two slaves in the Bluetooth one: actually they cannot communicate directly, but they need the master as a bridge of the communication.

Communication into the net

Bluetooth uses the FHSS, IEEE 802.11 uses the DSSS technique. Certainly, using a frequency diversity, the Bluetooth technology is less weak against the interference in comparison with the time diversity technique of IEEE 802.11. Both technologies can change the data rate, regarding the congestion of the channel. IEEE 802.11 manages this problem in the Physical Layer, instead of Bluetooth that solves this problem in the MAC layer. In regarding to the throughput, IEEE's performance are better, considering the rate of the transmission of 22 Mbps, against 1Mbps of Bluetooth.

	Bluetooth	**IEEE 802.11b**
Physical layer	FHSS	FHSS, DSSS, IR
Hop Frequency	1600 hops per sec	2.5 hops per sec
Max transmitting Power	100mW	800mW
Data Rates	1Mbps	22Mbps
Range	30 ft.	1000 ft. LOS
Cost	Least expensive	Most Expensive

Table 2.2: A comparison of the Bluetooth and Wi-Fi protocols.

Conclusion

After analyzing the main differences between these two protocols, IEEE 802.11 has been prefered because of its theorical unlimited capacity, its possibility to reach further distances and higher rates.

2.4 Problem analysis: Transport layer

Considering that our choice for the lowest layers is gone towards IEEE 802.11b technology, the protocol stack TCP/UDP represents the solution for the transport layer. Transfer Control Protocol (TCP) and User Datagram Protocol (UDP) are typical transport layers of a TCP/IP network. A device, such as phone, can provide and request several services from and to the net. Every service, in such a net, is associated to a "port" of the transport layer that should be specified in every kind of communication between devices. From two ports of different devices, a socket is created to make the streaming on. Before deciding which technique is better, it is necessary to show how they work.

2.4.1 TCP protocol

TCP sockets provide to establish a data pipe between two endpoints, both of which can send and receive streams of bytes. The main characteristic of this protocol is to make on a permanent connection called "session". Such a connection can be compared with a typical client/server model. Actually, a client, creating a socket, connects to the server, and then begins sending and receiving data. On the other side the server has to create a socket as well and listens for incoming connection. The next picture shows up the main functions of the process established between endpoints.

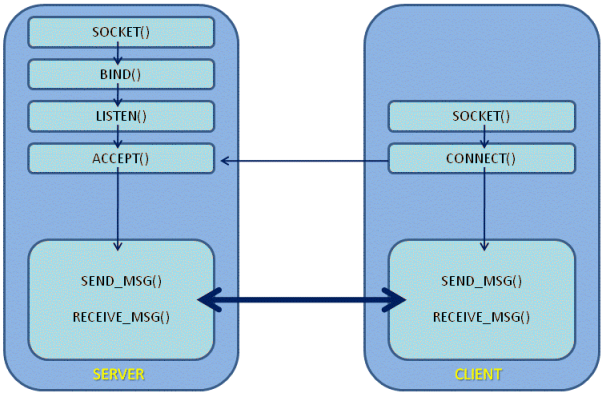


Figure 2.11: The TCP protocol

This protocol makes a bidirectional flux of bits and usually, it cuts the main flux in different parts. Establishing a connection, TCP makes us sure the right reception of data that reach the destination in order and at most once. It assures this using acknowledgments and, in case, retransmission. TCP uses to control the network in every moment, regulating the flux of data, considering the congestions of the buffers.

2.4.2 UDP protocol

Connectionless sockets are the other option for transferring data between two networked devices. By using the same client/server model, actually UDP does not install any permanent connection. It represents a perfect solution for all the applications that do not need a big overhead. The term "overhead" is indicated to any combination of excess or indirect computation time, memory, bandwidth, or other resources that are required to attain a particular goal in a generic communication. UDP, in order to transfer information, uses to send datagrams (message packets). Besides there is no guarantee that the packets will arrive to the destination, neither any guarantee about the order. It is sure anyway that if the packet arrives to the destination, it will arrive entiger. The next picture shows up the main functions of the UDP process.

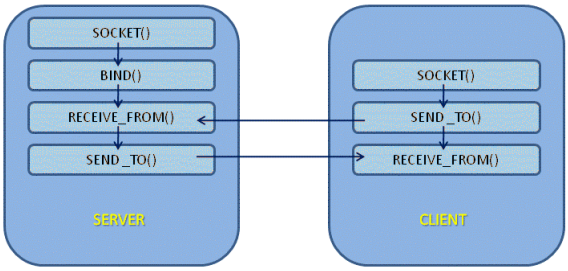


Figure 2.12: The UDP protocol

One of the advantages of this protocol is that it is capable of transmitting data to multiple endpoints, keeping a high data rate. Besides, it can assure a sort of protection of data against errors, with checksum.

2.4.3 TCP versus UDP: what should our application use?

Before deciding which one is the best for our application, it is useful to specify which characteristics are needed. Actually, as mentioned above, the sharing of a multimedia data on more devices in real time is the goal of our project. To reach it, a fast streaming is needed and besides the possibility to send data, at the same time, to more destinations. The protocol that can assure us these characteristics is UDP. Actually, by using TCP, the throughput would quickly decrease as a function of receiving users in order to realize a broadcasting.

Prevention against the fading

The checksum used by UDP are not enough to protect the code transmitted by devices. It is necessary to think about a probably loss of data that should be during the communication. Obviously, it is important to consider the condition of the channel in which the devices are. By using the Wi-Fi standard, the devices can exchange data even for quite long distance (more than 20-30 meters). In order to have a direct and optimum transmission about some receivers and some transmitters, the line of sight should be without any obstacles. This situation is very improbable. It is more common the presence of reflected and scattered waves that arrive to the receiver with different delay, provoking a significant drop of the quality of the signal. It is the phenomena of the fading that actually depends on the channel and must be consider for characterizing the quality of the received signal.

Then, some prevention is necessary to avoid a bad reception of data. Two ways are possible: introducing some algorithm of prevention , or choosing a right audio and video codec.

2.5 Problem analysis: Application layer

By choosing UDP as transport layer, it is necessary to have an application able to manage the multiplexing and the checksum services of it. The most common application used over UDP is RTP (Real-time Transport Protocol). As the standard RFC 3550 3551 says,

"RTP provides end-to-end delivery services for data with real-time characteristics, such as interactive audio and video. Those services include payload type identification, sequence numbering, timestamping and delivery monitoring".

Therefore, RTP can give us the control mechanisms of a multimedia transmission. It is easy to get why it can be very functional just thinking about an example. Actually, as it was already mentioned, UDP/IP cannot assure for the packets the same path and the same order of transmission. So very often some packets can be lost. RTP solve this problem encapsulating the UDP packets in a new numbered payload and knocking down the problem of jitter.

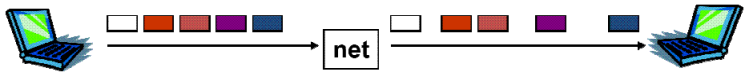


Figure 2.13: The RTP encapsulation of the UDP packets with a new numbered paylod can solve the problem of a bad reception

Jitter and PDV

It is necessary to specify what is jitter for and its main differences with another important parameter: packet delay variation (PDV). Jitter is defined as a variation in the delay of received packets. At the sending side, packets are sent in a continuous stream with the packets spaced evenly apart. Due to network congestion, improper queuing, or configuration errors, this steady stream can become lumpy, or the delay between each packet can vary instead of remaining constant. [4] Usually this problem can be repaired with a playout delay buffer built in the receiver to compensate those delays. Actually RTP does that.

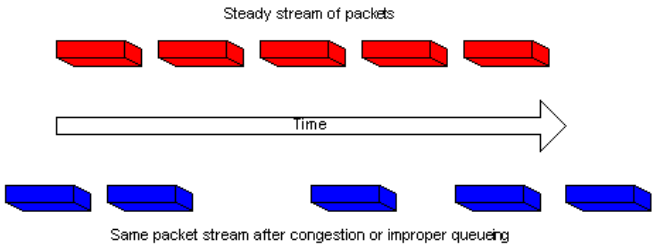


Figure 2.14: A representation of the jitter[4]

In the context of computer networks, the term jitter is often used as a measure of the variability over time of the packet latency across a network. A network with constant latency has no variation (or jitter). Packet jitter is then expressed as an average of the deviation from the network mean latency. This definition can cause confusion with the definition of IP Packet Delay Variation that is, as the RFC 3393 cites, the difference between the one-way-delay of the selected packets. The Instantaneous packet delay variation is the difference between successive packets and this is usually what is loosely termed "jitter", although jitter is also sometimes the term used for the variance of the packet delay. As an example, say packets are transmitted every 5 ms. If the 2nd packet is received 6 ms after the 1st packet, IPDV= -1 ms.

In the next chapter, the latency of the channel will be discussed.

2.5.1 Multimedia streaming: RTSP

By reminding the purpose of the project, a description in general of the streaming is needed. One file (audio or video), safe in the memory of the source (represented from a server), is transmitted to a client. From the definition of streaming, the reproduction of the mediafile in the client starts before the conclusion of the transfer. So, all data should arrive in the receiver (Rx) before the reproduction needs some data to complete itself.

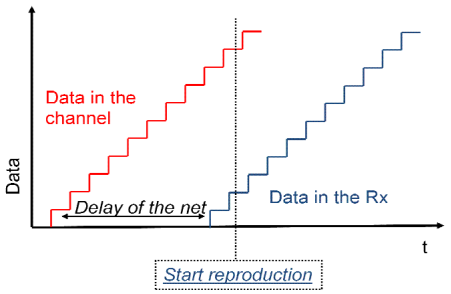


Figure 2.15: Functioning of a multimedia streaming with regard to the time

An easy way to think about streaming is to encapsulate the media file and send it to the client. After his arrival, the file can be move to the player and start the reproduction. Actually that is not streaming, not having in the transfer any pipeline and thinking about the big delay of the transmission.

A different way to think about the streaming is suggested by RTSP (Real Time Streaming Protocol) that actually is the protocol more appropriated for such a problem. The Client receives first a metafile with the description of the file that are going to arrive. Then it passes the file to the player that contact the server to start the streaming. This protocol is very useful because it can

manage the reproduction with commands such "stop", "play", "rewind", typical of a common cassette player.

2.5.2 A new algorithm based on the RTSP

After describing the most spread protocols able to manage the problem of the project, it is necessary to justify why on this report a new protocol will be implemented. First of all, the complexity of the protocols just mentioned is excessive for the one requested by our project. Many functions need to be checked and run and our implementation can easily avoid such a job. Another reason to implement from scratch is the difficulty (sometimes, the impossibility) to find a working protocol available on the OS of the phone. For all these reasons, in the next chapter, a survey of a new protocol will be done.

2.6 Conclusion

After defining the technologies available on the phone, in this chapter an analysis of the main ones has been made. By taking in consideration the ISO/OSI architecture, the IEEE 802.11 has been chosen as physical layer and data link control layer. Over the IP, that manage the Network layer, it has been discussed why the UDP should be use instead of TCP, in the Transport Layer. In the end, analyzing the last layer, after discussing about the main spread protocol for such a problem, RTP and RTSP, it has been explained why for this application a new protocol will be implemented.

Chapter 3

UDP PERFORMANCES IN AD HOC WIRELESS

3.1 Preface

In this section we will discuss which performances we can expect from a network able to make a reliable end-to-end communications. As we mentioned in the previous chapter, we are particularly interested in fast data exchange between multiple mobile devices for real-time interaction situations such as interactive music, in our case, or also exchanges in games. Examples quite important and spread are IP-telephony and VoIP.

By using architecture as IEEE 802.11, the Mobile Ad-hoc NETwork (MANET) represents a natural candidate for ad-hoc device-device wireless interaction. Actually, it does not require coordination by a master node and it has not restriction on the number of nodes which can participate in the exchanging. Because of low control overhead, the transport layer UDP is, as we have already seen, ideal for real-time communication. But it is important to know how to control the possible delays and the packet losses, when the number of nodes, the communications rates and the power increase.

Then, the main purpose of this chapter is to analyze the performance of an UDP MANET in order to find some parameters that can assure a desired Quality of Service (QoS).

3.2 Symbian Smartphone

First of all, it is important to describe the platform that will host the application described previously. As we have already mentioned, such an application is born to run on mobile devices, than can be easily represented by smartphones.

Whereas a simple phone enables voice calls and short messaging (SMS), a feature phone contains some significant additional functionality:

- multimedia messaging support (MMS)
- a colour display
- a digital camera
- support for additional memory through the use of memory cards
- a web or WAP (Wireless Application Protocol) browser
- etc

Smartphones is more advanced than the previously ones, by using high-level operating system with support of multitasking, expandibility, multimedia or convergence features, application interaction and so on. The S60 Platform is the world´s leading smartphone software platform, offering a feature rich software base for phones with advanced data capabilities. It includes the Symbian OS and the Nokia S60 UI (user interface).

Figure 3.1: Logo of S60

S60 consists of numerous architectural units, for example the Symbian OS, the Domestic OS and UICon. The Symbian OS provides several services to the platform and to platform-based devices. Such services are, for example, the User Interface (UI), applications and middleware. The Domestic Operating System (DOS) is the proprietary operating system and no interfaces in it are open to third-party developers. UICon is a graphical user interface library for reference-design (DFRD) independent functions based on EIKON, which is the original graphical user interface library for the Symbian OS. Use of such components guarantees the implementation of the application of the user interface by developers. [1]

Nokia N95, used to test this application, is based on this platform.

3.3 The programming language Python

Different programming languages can run on a Nokia's S60 platform, namely Python, Java, Symbian/C++, etc. The first one was chosen because of its great capabilities and flexibility. Several applications in Python have been written in

the last years because it is easy to learn and, for that, it is pretty simple to find examples and useful modules to make our code.

Figure 3.2: Logo of Python

Python for S60 provides a scripting solution making use of Symbian C++ APIs; it is a dynamic object-oriented open source computer programming language that was created by Guido van Rossum ([7, 8, 9]). Python can be used for many kinds of software development and run on most common platforms. It is pretty spread on smartphones because of its many Python Standard Library modules built-in and the additional mobile device specific modules, e.g., for SMS, telephone features, camera, and system info. The advanced processing power and memory capacity of smartphones allows running an interpreted language like Python on such devices, simply by installing a Python execution environment. On that environment, with few lines of code, it is possible to install an Udp Connection between two or more phones. [2]

3.4 User Datagram Protocol: the code

Having described the code used for this application, it is time to get into the code and realize a simple UDP connection between phones. The main purpose of the script is to establish a connection between two entities: server and client. The server has the duty to broadcast data into a network whereas the clients should receive all data incoming from the server and store it in a buffer.

UDP, that is used for sending very short messages, provides at most one guarantee: that the data received is intact. It does not guarantee that data will actually be received, or that it will be received just once or that different messages will be received in the order that they were sent. So, by creating an object able to establish an UDP connection is not enough to control all those problems: in the final code other instructions are needed to check them out.

In order to make it work, it is necessary that the server phone has defined on it an access point able to establish a public Ad-hoc network. Any kind of protection of the network, such as protocol for security, has been avoided, in order to have a faster application.

Without getting too much in details, the main essential instructions will be described. It is important to underline that Python provides to the programmer a big amount of flexibility and power for operating in system's socket interface. It is easy to notice all the similarities between C and Python that in the network programming offer the same socket services. Having said this, it is time to give a look at the basic client and server written in Python.

By using the module *btsocket*, both clients and server have to define on which network are ready to operate.

```
apid = btsocket.select_access_point()
apo = btsocket.access_point(apid)
btsocket.set_default_access_point(apo)
apo.start()
```

After looking at a list of all the possible network usable by the phones, an user (or it is possible automatically) should choose the same network on both sides. On that one, a UDP connection will established by using the object socket.

```
client = btsocket.socket(btsocket.AF_INET, btsocket.SOCK_DGRAM)
```

The socket represents an extension to the operating system's I/O system that enable communication between processes and machines. In the definition of the socket, as we can see, it is necessary to define the communication type and the protocol family. The communication type specifies the underlying protocol used to transmit data. Example of protocols are IPv4(the most common), IPv6 (latest Internet standard), etc. The protocol family defines how data is transmitted: it is necessary to specify then which transport protocol is used. For our application the communication type is AF_INET (corresponding to IPv4) as defined in the parameters of the Access Point previously installed whereas the protocol family is SOCK_DGRAM, typically used for UDP communications. (SOCK_STREAM is the one for TCP). To use well the socket, it is necessary to provide a tuple containing the remote hostname or the IP adress and the remote port. By reminding the purpose of this script the server transmits on broadcast (255.255.255.255) whereas the clients receive each messages incoming from the network (0.0.0.0). The port, defined on both phones, to make the two transport layers communicating, should be of course the same.

Once defined these parameters, the most is done. No needs to establish a connection means that the server can already start to broadcast data.

```
server.sendto(data, ("255.255.255.255", port))
```

Of course data should be sent with forethought: it should not have a dimension too bigger and in case it is divided in more packets, it should be sent with a rate sustainable from the channel. Next paragraphs will argue that. Anyway for best match with hardware and network realities, the value of data should be a relatively small power of 2, for example, 1024.

A similar script is realized in the client where a loop is establish to check though the net, looking for some incoming packet.

```
client.bind((host, port))
(data, address)=client.recvfrom(buf)
```

After binding a specific host, or all the network, as in our case, the receiver with a simple instruction can manage the connection. Actually this function returns just two pieces of information: the received data and the address and port number of the application that sent the data. Because UDP is connectionless, this is all it is needed to be able to send back a reply.

3.5 Packet loss

By sending packets between devices in order to make a reliable audio streaming, it is important to focus to the spatial correlation of packet losses between nodes in a single hop network, how the performances change increasing the transmit power and which are the differences between a communication in Line of Sight (LOS) and Non Line of Sight (NLOS).

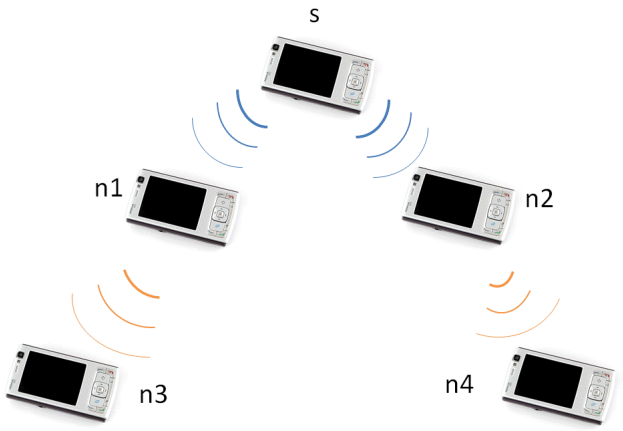


Figure 3.3: An example of the network discussed in the text

Before talking about the problems just described, it is important to see how the correlation packet losses engrave on an application like ours. As done in

[1], we can consider a scenario in which one sending node denoted *s*, multicasts data to two nodes (*n1* and *n2*) that, after receiving, send each one those packets to other different node (*n1* to *n3* and *n2* to *n4*). Let us call *p1* and *p2* the probabilities than *n1* and *n2* do not receive data in a perfect way. If the individual loss probabilities are reasonably low, as we hope, we have a very small joint probability that both receivers loose the same packet given by *p1p2*. Let assume that there is no correlation between the losses of *n1* and *n2*. Then, one protocol possible to make the net efficient is to make *n1*, if the packet does not arrive in it, waiting the time that *n2* spends to transmit the packet to *n4*. Of course, this protocol is completely useless in case there is correlation between the losses of *n1* and *n2*. In this last case, the probability of common loss is greater than *p1p2*.

In past researches, a certain degree of packet loss correlation between devices was proved. But all the papers do not take account of the data lost due to overflow in the sending node buffer.

3.5.1 Experimental results

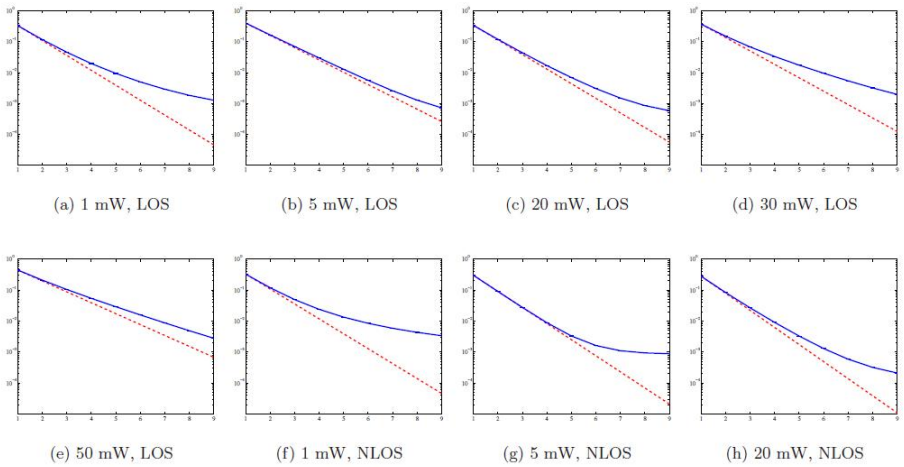
By building a grid of nine receiver devices, NOKIA N95s, running Symbian OS 9.2 (the same we are going to do use in our application), [18]with the Table 3.1 describes how the data rate and the loss probability change increasing the power of the multicast transmitter (an Access Point that transmits 10000 packets of 1400 B) in a LOS and NLOS scenario. In the last row the parameters are shown, after removing the break time between the transmitting of the packets.

P [mW]	Time Space [ms]	**Line of Sight**		**Non Line of Sight**	
		D [Mbps]	L [%]	D [Mbps]	L [%]
1	4	1.82	33.0	1.82	33.0
5	4	1.63	40.0	1.90	30.2
20	4	1.80	33.7	1.95	28.1
30	4	1.71	36.9	2.08	23.4
50	4	1.51	44.4	1.87	31.1
50	0	2.29	88.9	2.29	89.0

Table 3.1: Data rates and packet losses for the receivers in the cluster at different test conditions.

What can we notice from this table? The Data rate in the LOS scenario is less than the NLOS' one whereas the losses in the LOS scenarios are bigger than in NLOS' one. From the last row, we can point out that, by sending packets continuously, the nodes losses approximately three times the number of packets compared to the measurement where the server pauses (4 ms spacing) between each packet. By using the Kullback-Leibler distance, [18]makes a comparison between the Probability density function (pdf) under the assumption

of independent loss probability and the estimated pdf found from the collected data.



(a) 1 mW, LOS (b) 5 mW, LOS (c) 20 mW, LOS (d) 30 mW, LOS

(e) 50 mW, LOS (f) 1 mW, NLOS (g) 5 mW, NLOS (h) 20 mW, NLOS

Figure 3.4: The theoretical and measured probability of different number of erasures for a packet transmitted to the cluster. The punctured line is the theoretical probabilities under the assumption of independent erasure processes, the solid line is the estimated probabilities found from the collected data.

As we can see, if the transmit power is not smaller than 5mW, the correlation packet losses is negligible, in the case in which the nodes are not more than 5. There is no big difference whether there is LOS or not. That is a good result because our kind of application cannot be compulsory to a specific topology. It is completely random. Then, it is possible to notice that by increasing the number of nodes, the correlation increases as well, as we can see from the figure: the curves are actually further. [18]tries afterward to find some confirmation of the previous results going to change the position of the devices in the grid, but it founds out that some devices have considerably worse characteristics than others.

3.6 Testing the performance of the phones

After having looked at a general analysis about the performance of the phones, in this section it will be shown results of some tests made in order to find the best configuration of the network. That one will be used afterward for the implementation of the algorithm able to realize synchronization. Each test will be introduced by a table able to resume all parameters necessary to define the background in which the test has been done.

3.6.1 Testing the packet loss

In this first test, the performances of the phones are shown in a situation of not stressing mode. It has been known that such devices can achieve a good transmission characterized by a rate of 100 kb/s. In this test, the requested rate is just 2 kb/s. Therefore optimum performances are expected.

Number of packets	900
Dimension of packet	2233 bytes
Number of server	1
Number of clients	9
Distance server-clients	4 m
Characteristics channel	LOS
Rate	2168 Bytes/s
Duration test	15'27"

Table 3.2: Description of the parameters necessary to describe the first test

Results

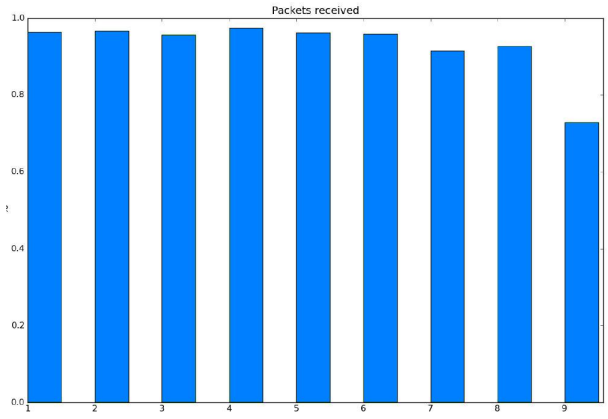


Figure 3.5: Number of packets received by each phone in the first test.

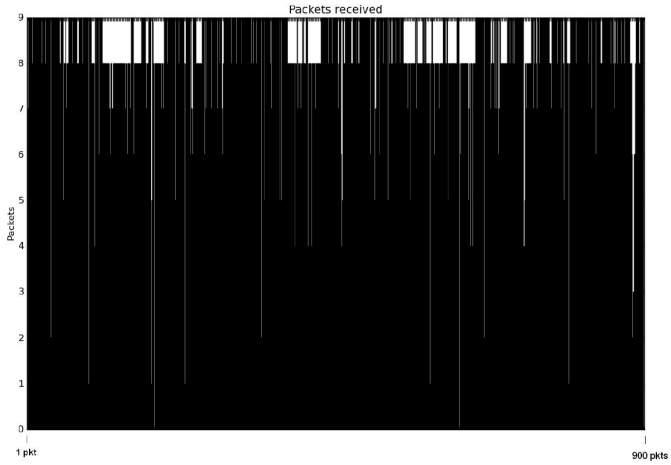


Figure 3.6: The matrix shows how many phones received each single packet in the first test.

As it is shown in the previous graphs, generally all the phones reach very good results by receiving more than 95% of the transmitted packets. Anyway, it has been verified that some phones can have bad performance even in such an easy trial. Some phones in other tests, not shown, did not receive at all. In that case the reset of the phone was necessary to make the phone receiving again. As proved already in the previous section of chapter, the performances of the phones in the network depends on the phone themselves. By giving a look to the second graph, there are packets lost by the most of the phones: the reason can be found in some external interference. It has not been verified any multiple reception of the same packet. As the protocol establishes, no fragmented packets has been received.

3.6.2 Performance as function of the distance

In the second test, the parameters of the previous one are kept by changing the distance between the server and the clients. Actually it has been increased from 4 meters to 20 meters by using a linear shifting.

Number of packets	900
Dimension of packet	2233 bytes
Number of server	1
Number of clients	9
Distance server-clients	4 m - 20 m in linear scale
Characteristics channel	LOS
Rate	2168 Bytes/s
Duration test	15'27"

Table 3.3: Description of the parameters necessary to describe the second test

Results

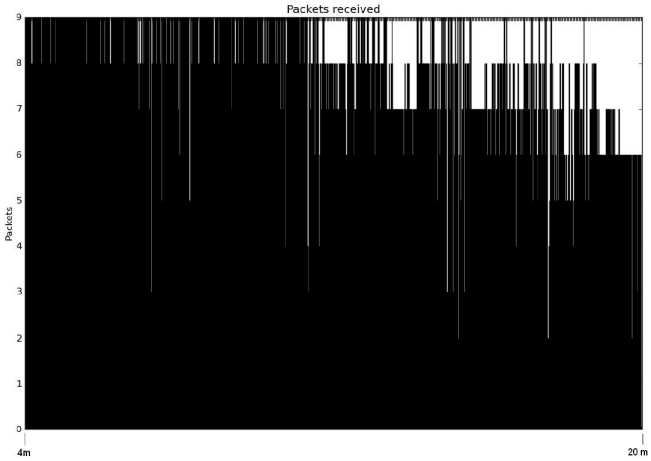


Figure 3.7: The matrix shows how many phones received each single packet in the second test. The x-axis can be associated to the single packet and the distance. For example the packet n.1 has been sent by server far 4 meters from the clients, the packet n.900 at 20 meters.

In this second test, as expected, it is shown the worsening of the performances of the phones by increasing the distance. Anyway, this weights upon just when the distance is already greater than 15 meters. In the next chapters, it will be clarified that the request of a synchronization for this kind of application is requested for a range smaller than 15 meters. It needs to be specified that such results should be referred to very low rate. By increasing the rate, the results gets worse.

3.6.3 Performance as function of the size of the packet

In this subsection, a very delicate problem is brought up: the dependance of the performance of the phone on the size of the packets when a certain rate is requested. This problem is analyzing a crucial situation needed to improve definitely the functionality of the application, goal of this report. As already mentioned, it has been proved that a rate of 100 kb/s is achievable by the phones on which the application is tested. It this context, the following problem is proposed: the research of the best size of the single packet in order to minimize the packet lost by the phones, in order to establish a rate of 44,1 kb/s. To send 44100 bytes each second, three possibilities are proposed:

- 20 packets of 2233 bytes
- 30 packets of 1470 bytes
- 35 packets of 1260 bytes

These sizes have been chosen because they are exactly divisors of the amount of bytes needed to send in one second.

	First case	Second case	Third case
Number of packets	5000	5000	5000
Dimension of packet	2233 bytes	1490 bytes	1280 bytes
Number of server	1	1	1
Number of clients	4	4	4
Distance server-clients	4 m	4 m	4 m
Characteristics channel	LOS	LOS	LOS
Rate	44100 Bytes/s	44100 Bytes/s	44100 Bytes/s
Duration test	2'24"	2'24"	2'24"

Table 3.4: Description of the parameters necessary to describe the third test

Results

Next table shows the results.

	First case	Second case	Third case
Phone #1	58%	30%	98%
Phone #2	78%	43%	99%
Phone #3	39%	42%	97%
Phone #4	35%	26%	96%

Table 3.5: Description of the result of the third test

The test shows ambiguous results that are not easily interpreted. Two factors should be taken in account: the dimension of the packets and the number of loops requested by each phone per minute. It can be figured out that when a phone is increasing the operations, while receiving, generally it drops some incoming packets. However, at the same time, when it is managing less data, it have less problems to receive packets. Having said that, the second case is probably representing the worst case because the packets per second are pretty a lot and, overall, the size of each packet is close to 1500. This critical size creates fragmentation. That is why there is a big difference between the second and third case.

The Maximum transmission unit (MTU) represents the maximum dimension in bytes of the protocol data unit. Each layer of the ISO/OSI stack seen previously can impose a different MTU. These limitations are due to several reasons, dependent on the layer [3]:

- Hardware (e.g., the width of a TDM transmission slot).
- Operating system (e.g., all buffers are 512 bytes).
- Protocols (e.g., the number of bits in the packet length field).
- Compliance with some (inter)national standards.
- Desire to reduce error induced retransmissions to some level.
- Desire to prevent one packet from occupying the channel too long

In our case, the transport layer is needed to be taken up. The server tries to transmit per each fraction of second some data (1490 bytes in the second case, 1280 in the third one, by considering the header added to characterize the UDP packet). The transport layer takes data streams and breaks them up into datagrams. In theory, datagrams can be up to 64kb each but in practise they are usually not more than 1500 bytes. Because the first value is very close to this limit, the probability of some fragmentation is very high. Therefore, each client, to perform a good reception, has to work twice.

3.7 Conclusion

In this chapter an approach to an UDP connection working in MANET has been made. After showing how such a network can be implemented by Python, the code chosen to work out on Symbian smartphones, its performance has been analyzed. The performances of the phones in the network depends on the phone themselves. Some phones can stop to receive suddenly without precise reason, so the reliability is not always guaranteed. Anyway, in the short distances requested by this application (less than 10 meters) and by choosing the right size of the packets for the rate requested, the phones can offer very good performance.

Chapter 4

AUDIO ENGINE

4.1 Preface

In the previous chapters an analysis of the project and the composition of the stack were made. The first chapter does not talk about a specific data and, because of that, it can be associated to several problem of streaming. In this chapter, instead, the report wants to investigate more about the audio streaming and how to use practically all the protocol described before. The choice of the audio codec will be analyzed and how it stores data into a file. Afterward it will be shown a way to stream audio on Python application.

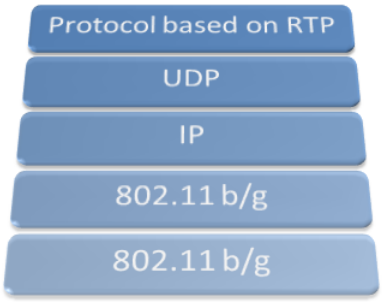


Figure 4.1: The ISO/OSI stack of our application

4.2 Problem analysis: Audio codec

By using network devices such as phones, just the most common audio codec are analyzed in this paragraph. Before going in the description of the standards, it is necessary to specify why we must talk about audio compression. Every audio file is composed by a number of bits that represent the information and these

bits take up space in the memory of such a net device. The compression is then necessary to save space in those memories and save time during a transmission. The composition of the strings of bits that describe a media file are closely related with time. Actually every subsequence of bits, long one second, is able to reproduce a sound. The bitrate is defined as the number of bit transmitted necessary to reproduce a sound for one second. During the reproduction of an audio file the bitrate is not necessarily constant. The purpose of the audio compression is to reduce the bitrate, leaving unchanged, or almost unchanged, the quality of the file. How is this possible? Let's make an example just to make clear that these algorithms are achievable. Every audio file is characterized by a certain band in frequency. By considering the interval of perceptible frequencies from a human being (16 Hz – 20 kHz), it is useless to take account of all the component of frequencies out of this band.

4.3 Setting aside the MP3 format

MP3 format is the most spread audio format because of its low data rate and because of the small dimension of the audio file. Although not a streaming format (it is for download then play), most popular codecs can be used to wrap MP3 files for streaming. The main reason is the architecture on which this kind of format is based. By dividing the sound file in several packet it can happen that an useful information to play a specific istant of time is contained in more than one packet. By using UDP, the right reception of all the packets is not assured. So, in the case of lack of one packet in the receiver, the sound spread by the phone can be affected by noise.

4.4 Digital audio compression: PCM

One of the compressed audio we are going to consider is the PCM, the pulse-code modulation that represents the basis of digital audio. The main characteristic of this format is the possibility of reproducing sounds of any raw chunks even if they are affected by some errors. That is possible because of the linear compression of the sounds.

To obtain a PCM audio file a simple analog to digital conversion is done. The voltage generated by a microphone is sampled at regular intervals, after a processing in a low-pass filter. Then, the sample values with a quantization are turned into a digital code word. This kind of audio format, as already mentioned, uses a fixed step value for the quantization levels.

Sampling

The sampling rate is set by the desired frequency response. The Nyquist criteria dictated that the sampling frequency should be greater than twice the highest frequency captured by the microphone. Telephony has an upper limit of 3.2

kHz; a sampling frequency of 8kHz is used. Broadcast audio is usually limited to 15 kHz; this requires a sampling rate of 32 kHz. By increasing the sampling rates, the reproduction is more similar to the original.

Resolution

The number of the levels used by the process of the quantization determines the resolution. The minimum step value sets the noise floor of the converted signal. Using 8 bits to represents a level of quantization, it is possible to represents just 256 levels. By increasing the number of bits, the resolution will increase. Professional users usually demand resolution of 24 bits, equal to almost 17millions of levels.

Audio channels

Audio can be transmitted in a single channel or can carry spatial information in a multi-channel format. For instance, stereo is very popular; it is enough to think about the two loudspeakers that each PC audio system uses for reproduction. In a compression, it is usually common to remove stereophonic irrelevance that is redundant information in both left and right channels.

4.5 The frame of a PCM audio file

In this paragraph it will be explained in details how is composed the frame of a PCM file that is a specification of the WAVE format. [6]

The Microsoft Waveform Audio (WAVE) file format is a subset of Microsoft's RIFF specification for the storage of multimedia files. A RIFF file starts out with a file header followed by a sequence of data chunks. To be read by CD players directly the data of a wav file should be encoded in PCM; this should help even to reduce the size of the files.

By opening and reading a WAVE file with Python, it is possible to mark all the sub chunks in which it is divided: a WAVE file is often just a RIFF file with a single "WAVE" chunk which consists of two sub-chunks: a "fmt " chunk specifying the data format and a "data" chunk containing the actual sample data.

CODE in hexadecimal	Bytes	Field name
The "RIFF" chunk descriptor		
RIFF	4	ChunkID Contains the letters "RIFF" in ASCII form (0x52494646 big-endian form)
\x1c\xb6\x1c\x00	4	ChunkSize This is the size of the rest of the chunk following this number. This is the size of the entire file in bytes minus 8 bytes for the two fields not included in this count: ChunkID and ChunkSize.
WAVE	4	Format Contains the letters "WAVE" (0x57415645 big-endian form).
The "fmt" sub − chunk		
Fmt	4	Subchunk1ID Contains the letters "fmt " (0x666d7420 big-endian form).
\x12\x00\x00\x00	4	Subchunk1Size 16 for PCM. This is the size of the rest of the Subchunk which follows this number.
\x01\x00	2	AudioFormat PCM = 1 (i.e. Linear quantization) Values other than 1 indicate some form of compression.
\x02\x00	2	NumChannels Mono = 1, Stereo = 2, etc.
D\xac\x00\x00	4	SampleRate 8000, 44100, etc. In this case it is 44132 Hz
\x10\xb1\x02\x00	4	ByteRate SampleRate * NumChannels * BitsPerSample/8 In this case it is 176400 byte/s
\x04\x00	2	BlockAlign NumChannels * BitsPerSample/8. The number of bytes for one sample including all channels. In this case it is 4
\x10\x00	2	BitsPerSample 8 bits = 8, 16 bits = 16, etc. In this case it is 16
\x00\x00Fact\x04\x00 \x00\x00g-\x07\x00	(..)	Extra parameters
The "data" sub − chunk		
Data	4	Subchunk2ID Contains the letters "data" (0x64617461 big-endian form).
\x9c\xb5\x1c\x00	4	Subchunk2Size NumSamples * NumChannels * BitsPerSample/8 This is the number of bytes in the data. You can also think of this as the size of the read of the subchunk following this number
\x00\x00\x00\x00\(..)	(..)	Data The actual sound data.

Table 4.1: How is organized the frame of a WAVE file

4.6 The module Audio Stream

After defining all the main parameters useful to define a PCM file, it is important to think a way to reproduce this sound incoming from the network. By talking just about PCM and generally about WAVE file, Python supports the `wave` module. It provides a convenient interface to the WAV sound format and even if it does not support compression/decompression, it is useful anyway in our case by using PCM, lacking in any kind of compression.

The module just mentioned works by giving to it just the data chunk of the WAVE file before described. Actually it takes all the parameters necessary to read such data by initialing a `Wave_write` object. In that object, it is necessary to set:

- the number of channels
- the sample width
- the frame rate
- the number of frames
- the compression type and description. But at the moment, only compression type NONE is supported, meaning no compression. Probably in the next future, it will be improved.

This kind of module is therefore able to read just frames of sound (even if they are not correct through the function `writeframesraw(data)`).

Though this code was working quite well, in that project it has been used another module located at this web address: `http://mobdevtrac.es.aau.dk/..`
`..symbian-util/browser/trunk/util_audiostream_pyext` .
It was made this choice for two main reasons: for the greater quality of the output of this second option and because of its flexibilty about having in input also other audio format.

The utility Audio stream, based on Symbian C++, is actually able to read:

- MP3
- AAC
- PCM
- AMR

without needed of any remotion of the headers. It can recognize whether the bytes contain audio or just some informations about reproducing the file. It is able to read file characterized by Mono or Stereo Channel and it can accept as input several values for sample rates standard value from 8000 Hz to 96000Hz.

Another important option of that module is the possibility to set the volume of the reproduction by interacting directly with the audio unit, located in the CPU of the S60.

This module is composed by four functions that, interacting, are able to sound. After defining an object by setting all the parameters described before

```
audio=utilaudiostream.AudioStream(...)
```

the audio will be launched by the function

```
audio.open()
```

This call-instruction will awake the other functions, opportunely defined before in the code. The functions `play_open` and `play_data` will manage the function `write_chunk` that is the one designed to send data into the player, having inside the instruction:

```
audio.write(buffer_of_data)
```

It is discussed in the next section the quantity of data that should be put as input of this instruction and how many times the functions should be called to avoid waste of time. It is important to specify that the variable ***buffer_of_data***, representing the chunk of sound next to be played, must be a string of chars.

When all the data has been spread as sound or some error is occurred, the function `play_stop` is launched to close the object audio.

4.7 Disquisition about time of reproduction and time of implementation

In according with the implementation of the player, it is important to analyze how the time of recharge can make slow the reproduction of the sound. Before testing the player for analyzing several situations, it is necessary to specify what time of recharge means. The sound that the player is suppose to reproduce is contained in a string of bytes. Because of the streaming, at the beginning of the reproduction, the player has not got yet all the data that is supposed to play in the next seconds. So, after defining an interval of buffering, the player should get those bytes from a buffer and spread them as sound. Afterwards this operation must be repeated each time that the function `write_chunk` has completed its role. So, in the end of the reproduction of each chunk, the player, in a time called by us "time of recharge", takes back other sound from the buffer, organize it in a string and starts to reproduce it again.

In according on the purpose of this project, it is extremely important that the player located in one of the client works at the same way of the player located in the server. Actually each kind of difference can determinate a delay of some milliseconds, that accumulating, can be noticed by the human ears. In the next sections, it will be analyzed how to prevent such problem by testing some possibilities offered by Python structures.

As already described previously, a player, that has got all the data when is launched, puts all the data in a variable by storing it as a string of char. This situation is, for instance, the one of the server. By the function `write_chunk`, after defining a constant value for the dimension in bytes of the chunk, the player starts to reproduce the stored data, by taking chunk by chunk from the string. So, each time that a chunk has been played, the player takes another one from the string defined at the beginning. It is important to specify that each time of recharge the player does not reproduce any sound. So it is necessary that it is pretty short compared with the time used by the player to reproduce a chunk of sound unless the sound will be characterized by bothered noise and visibly slowed.

4.8 How to manage missed packets in the player

As already mentioned several times, UDP protocol is the one chosen at the transport layer. This protocol does not give any assurance on the right reception of all the packets sent by the server. It is necessary therefore to think about a solution for a certain problem that will be showed up in the application: how to manage the synchronization about more mobile devices if some packets are lost just from few of them? The problem just described is one of the main of this project. This paragraph will try to give a solution for such a problem in the case of constant sample rate. The PCM audio format, described in this chapter, is an example of that; the MP3 instead cannot be involved in this disquisition.

By using an audio format with a constant sample rate, it is sure that by dividing the audio in many packets of the same dimension, each of them carries the same amount of time of sound. Thinking about the possibility to have some losses in the reception of the incoming packets, two strategies should be taken in consideration. Because of the ID header, placed in each packet, it is possible to check whether some of them is got lost. So, the first idea can be to interrupt the reproduction of the sound until the arrival of the next right packet. Of course, this interruption should be measured very carefully to keep on the synchronization with the other audio devices. Another option can be to replace these packets with others with the same dimension and characterized by an array containing just zeros. By inserting such packets in the audio player, it has been proved that no sounds is reproduced as long as the time of the sound contained in the lost packets.

This last option in according with the tests done in the implementation of this application is preferred. It has been proved that by replacing even the 33% of the packets in the original file with these "silence" ones, the music spread by the phone seems to be undistorted. Of course that assumption is true by considering a small dimension for each packet (less than 1.5 Kb) and by assuming this replacing in packets spaced in time. The first option should be abandoned because it depends strongly on the time used by the code to make the player working again: these deviations can severely affect the synchronization. The second option even though assures a good listening by losing a considerable number of packets, cannot avoid a bad reproduction if the losses are too many. But, anyway, it can assure that the synchronization is kept because the player continues to manage chunks of the same kind of before. This strategy is unusable by managing MP3 files, as already told, because of the inconstant sample rate and the organization of the file into the buffer. In according with what has been said, the best idea would be to have a buffer already prepared in the receiver, in which all the silence – packets are placed. By thinking with a reverse philosophy, when a right packet is come, it will replace the fake packet in the buffer, by taking the exact place it is supposed to have.

4.9 Strategies to buffer data

After discussing about the utility of a pre-saved buffer able to interact with the player, it is necessary, to avoid delay, that the server and clients have the same algorithm to manage the player.

Both players should have ready for interacting a pre-built buffer:

- a buffer containing all the file audio into the server one
- a buffer containing silence - packets, ready to be eventually replaced from the incoming packet, in the client ones

Python offers several ways to store data in the memory.

A first option to solve this problem is proposed by taking a char variable as buffer of our application. It is easy to think about that because the function `audio.write()`, as already said, accepts as input just char strings.

After defining this string, the player, by means of a shifting function of Python, is able to take and read just the data chunk of a desired dimension.

```
audio.write(string[counter:counter+dimension_packet])
```

Previously of course, that variable should be updated while receiving new packets from the network. In the next chapter it will be shown how to assure that a packet arrives in a client before that its content is supposed to be sent in the player.

While the player of the server is supposed to take chunks from the original string by a shift, the client must modify the original string each time that a

right packet comes from the network. The problem of such implementation is the weight of this operation in the CPU of the mobile devices.

The char variables in Python are defined as immutable. Because of that, each time the receiver wants to replace a silence - packet previously stored with a new one just arrived, it has to manage three variables of several Mb that are extremely heavy for the execution of the code. Actually the memory of the phone cannot manage such a code.

A second option offered by Python is the use of the list. In such structure, Python allows to store any kind of data (strings, numbers, etc) in a certain position, marked with an index. This kind of structure can be useful in such situation by placing each chunk, carried by a packet, in the list at the index of the same value of the ID indicated in the header of the packet.

It will be easy for the player by manage this situation by calling in the function `audio.write()` an element of the list for time:

```
audio.write(list[i])
```

It has been already discussed in the previous chapter about the dimension of the data that a packet can carry. By considering that assumption, this option must be discarded: the sound contained in each chunk, carried by a packet, is as long as the time of recharge of the player. Because of that, the sound spread by the player is affected by noise.

Fortunately it is easy to solve this problem. It is enough to create a string in the *write_ chunk* function each time and put it as input of `audio.write()`*:*

```
d=list[i]+list[i+1]
audio.write(d)
```

By doing that, the time of execution increases as much as the difference with the time of recharge: the sound is perfect.

4.10 Interleaving as prevention against interference

In previous sections, it has been discussed about the good efficiency of using "silence" packet, in order of a good listening even though lost packets in reception. This assumption, as mentioned, is true if the lost packets are not adjacent. Actually, by replacing two or more packets in a row (bursty losses) with "silence" ones, the quality of the sound gets worse quite quickly. A solution to solve this problem should be found to avoid situations in which some strong interference is able to decrease the instantaneous SNR of the channel.

A way to solve such a problem is suggested by a technique used in several application of the mobile communication world: the interleaving. This tech-

nique, inserted for instance in the protocol of the ADSL, is a "cheap" way to prevent situations described previously. An example can help to describe this technique.

In a network, a transmitter has the duty to transmit the message:

```
A B C D E
```

As first prevention against possible errors of the network, a repetition code is used:

```
AAAAA BBBBB CCCCC DDDDD EEEEE
```

The decoder in the receiver is able to get the sent message by using a majority decision:

```
AAxxA BBxBB CCxxC xDxDD EExxE
```

The same decoder falls in trouble when an interference is concentrated:

```
AAAAA BBBxx xxxxx xDDDD EEEEE
```

The interleaving can avoid such situations. How? It shuffles the characters of the message in a precise order that the receiver knows. Instead of transmitting the normal sequence, it will be transmitted

```
ABCDE ABCDE ABCDE ABCDE ABCDE
```

This time the message is immune to the interference previously described

```
ABCDE ABCxx xxxxx xBCDE ABCDE
```

by replacing the letters in the right order

```
AAxxA BBxBB CCxCC DxxDD ExxEE
```

The previously message is decodable by the decoder.

By following the same approach, the problem of the adjacent lost packets can be decreased. By taking sequence of N sound packets from the buffer located in the server, the block of interleaving will divide them in N subsequences. These ones will constitute new packets that will be recomposed in the receiver.

Even in this case an example can help. A sequence of 10 packets of 1300 bytes each is taken from the data-buffer. Each packets is fragmented in 10

subpackets of 130 bytes. These ones are collected to recompose 10 new packets with this criteria:

- `new packet #1 = [sub#1-packet#1]+[sub#1-packet#2]+...+[sub#1-packet#10]`
- `new packet #2 = [sub#2-packet#1]+[sub#2-packet#2]+...+[sub#2-packet#10]`
- `(...)`
- `new packet #10 = [sub#10-packet#1]+[sub#10-packet#2]+...+[sub#10-packet#10]`

By assuming of losing the packets #5 and #6 for instance, the silence long 1300 x 2 bytes is avoided this time. Actually, by using interleaving, each packet will have in the middle a silence long 130 x 2 bytes, not hearable from human ears.

In conclusion, it is possible to confirm the utility of this technique. Interleaving actually helps the communication against bursty losses, and it is "cheap" because the extra bandwidth requested for this technique is equal to zero. The only disadvantages is the increasing of the latency of the channel because of the time of decoding into the clients.

4.11 Conclusion

In this chapter an Audio engine to broadcast music by the tested mobile phone has been presented. After discussing the module "wave", available on latest versions of Python, the utility Audio stream has been introduced. The possibility of streaming a PCM file has been analyzed by talking about each tricky aspect and managing all the problems presented. A first approach of the problem of the synchronization has been done in regard to the correlation between the player and the receiver of a possible client. This problem will be solved completely in the next chapter with a formulation of an algorithm able to manage a synchronized audio on several mobile devices.

Chapter 5

SYNCHRONIZATION

5.1 Preface

By using wireless devices, it is pretty hard to make an application able to synchrony them, in regard to their possible movements and their several starting positions. As already done in several application, it is easier to manage the problem, by handling with some default scenarios. In these ones, the propagation of the sound is considered and the possible "hearable" echoes.

- The first scenario can be represented by the position of one people in an environment in which two speakers are present. The investigation is done in regard to the distance between the person and the speakers.
- The second scenario is described with couple of people, far each other, but owning each one a speaker, represented for instance, by a Nokia phone. It is examined the delay that one reproduction should have, if it would reach a sync with a different speaker already playing.
- In the third scenario, a complication of the second one is done. Actually at the second scenario a big stronger speaker is added. The investigation is done considering the effect of this new loud speaker on the previous considerations.

In these possible scenarios some assumptions are done. The position of the devices, for example, is found with GPS applications. By transmitting data in streaming, two possible causes of the delays are considered: the delay of the incoming stream to the receiver due to the channel propagation, the adding delay to make the sync when devices are pretty far. In this chapter these situations will be discussed and will be proposed an algorithm in which each delay previously mentioned is supposed to be approximately zero, not hearable by human ears. The first problem is now, how it is possible to define two devices far and when a delay is not hearables from human ears..

5.2 Haas Effect

In the 1946 a German doctor, Helmut Hass, analyzed how the human brain and ears can percept incoming sounds from different sources and the cases in which these sources are heard as only one. He realized several studies on the phenomenon and pointed out how the magnitude and the delay between different incoming sources are related to feel echo.

By thinking about this effect of fusing sounds, we should not be surprised. Actually from the beginning of the 20th century a similar effect is the fortune of the success of the cinema: our eyes fuse a series of sliding pictures, giving us the impression of continuous movement. To avoid to see the flicker 16 pictures per second should be placed (62 ms interval). Haas gave us similar considerations about sounds. Feeling echo is just a perception of independent incoming sounds.

Haas, as resumed in the next graph, found that in the 5 to 35 msec delay range the sound for a delayed loudspeaker has to be increased more than 10dB over the main one before it sounded like an echo [21]. Of course, by increasing the delay, the over-magnitude of the second source can decrease.

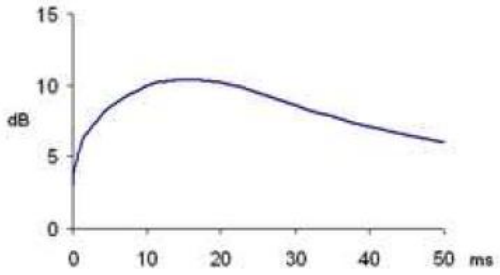


Figure 5.1: Haas curve: allowed level increase for secondary source as a function of delay

5.2.1 Test

In order to analyze the scenarios previously described, from [22] an interesting test is excerpted to prove the Haas effect. The test consists on playing at the same time a music file on two different devices looking for the distances in which the human perception feels two different sounds. To evaluate the perception five different intervals have been defined, far 4 m each. By following the same structure of the previous graph, 20 m are necessary. Actually 4m correspond to 11,63 ms of delay and so, 20m to 58.15ms.

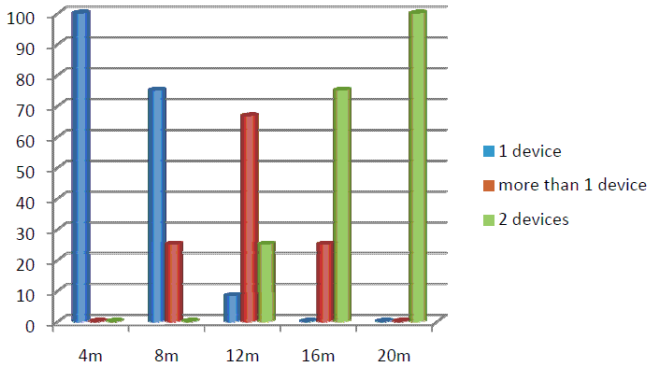


Figure 5.2: Graph of results with regard to the test described in [1]

The previous graph shows the results obtained in the text after experimenting with several individuals. For a distance of 4m, all the group perceives only one device. In 8 meters, some individuals start to perceive some echos that becomes clear at 12 meters. Even though, at this distance the perception of the sound is still satisfactory. By increasing the distance, the asynchronisms gets more perceivable.

5.3 Scenarios

After explaining which acoustic phenomena we can have by talking about synchronization, we can handle with the scenarios previously described.

5.3.1 Double side - scenario

As already mentioned, this scenario is represented by one person and two speakers in an environment.

Figure 5.3: A representation of the first scenario

One parameter very important in a such scenario is the distance between the two speaker and the position of the person between them. As the previous test suggests us, in the hypothesis that the speakers play at the same volume, starting the song at the same time, the position of the person is not relevant if

the distance d is not greater than 8 meters. Actually, even if the person is very close with one device, the echo is just perceivable. So, in the case one device has been already playing, it can send audio streaming make the second device play as if somebody would have pressed the button "play" on the devices at the same time.

An example can make the concept easier. To have synchronization, it should be known the delay τ that data spent to arrive from one device to the other one by crossing the network. Besides, it is important to define a delay τ_e that represents the time of elaboration that a device spends to make the incoming data play, since when it is received. In the end of this chapter it will be analyzed the amount of such parameters. After defining those, it is easy to solve the problem. If, after playing t_0 seconds, a device wants to join on fly another already playing, it should start to play data with a delay of $(\tau + \tau_e)$. (*Solution 1*)

What happens if the distance between the speakers increases? Or to be more precise, when τ increases. If the position of the person is exactly in the middle between the devices, nothing change! But if this doensn't happen, what we have said is not right anymore: the reasons are pretty easy to get. By repeating the same instructions, we will make the same situation has been in the Haas test, previous described. So, if the person is close to one device, even though the devices are exactly playing at the same time, the echo will be evident. How to solve this situation? Hard to answer because, this time, the position of the person is fundamental. One solution can be to increase the volume of the device further from the person, by following the Haas graph; a different way can be to consider even the propagation of the sound until the ears of the person. How?

$\tau_1$represents the time elapsed between the starting of the reproduction of the first device and the beginning of listening that sound from the person.

$\tau_2$represents the time elapsed between the starting of the reproduction of the second device and the beginning of listening that sound from the person.

$\triangle\tau$ is the difference between these delays: of course $\triangle\tau$ can be greater or not than 0, depending on the distance between the devices and the person.

If there is a way to calculate this parameter, our problem is solved. Actually, as discussed previously, to have a precise solution, it is necessary to apply the same system before established, by adding $\tau + \tau_e$ to the value . (*Solution 2*)

5.3.2 Personal speaker - scenario

As already mentioned, the second scenario is described with couple of people, far each other a distance d, owning each one a speaker.

Figure 5.4: A representation of the second scenario

By following the same path done in the scenario n.1, it is easy to point out that if the distance d between the people is in a range not greater than 8 meters, the solution 1, described previously is working, whatever the volume of the speakers. Instead, if d increases, the situation gets worse. Why? As the image shows, "A" and "B" represents the two people, placed near the speakers "a" and "b". If "a" is already playing and wants to install a sync with "b", we can use the solution 2 already used previously: in this way "B" won't have any problem to listen just one sound. What about "A"? He doesn't get any sync, because the delay, established to shift the reproduction of "b" in sync, has seen instead from "A" as a burdened delay between the two sound. Unluckily anything is possible to do to solve the situation, because, by decreasing the delay of the solution 2, we are going to improve the sound listened by "A", but we get worse the one listened by "B". The synchronization looks like impossible.

It can be useful, to hear less the difference between the two sounds, trying to decrease the volume of the two devices: but in this case, each person will just hear his own speaker. Because of that, by following this suggestion, the problem described is solved if the distance d between the two speakers is much greater than 8 meters: the attenuation of the wave of sound will hide the bad sync between the devices.

5.3.3 Concert - scenario

As already said at the beginning of this chapter, the third scenario is just a complication of the second one. Actually, by adding a loud speaker in the second scenario, it appears like a complication but instead, it can represent the solution to our previous problems. It is like that, because the loud speaker, with a stronger volume of the other ones, can hide the problem seen before.

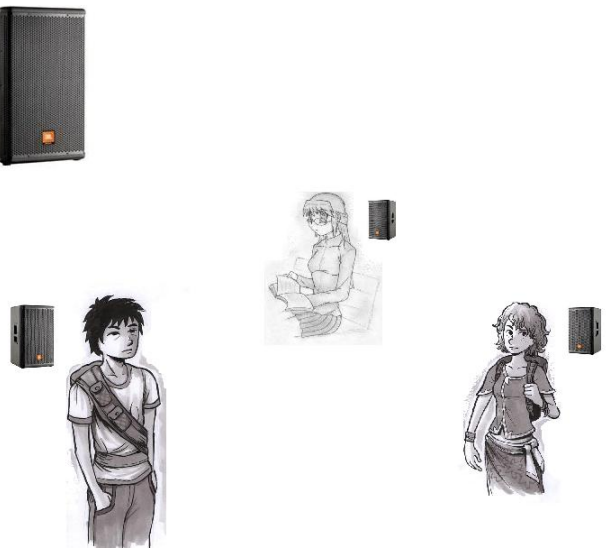


Figure 5.5: A representation of the third scenario

In this new situation each "normal" speaker must try to have sync with the loud one and it is possible by applying the solution 1 and 2, in depending on the distance d_i that separates the "i" speakers from the big one. By considering the scenario n.2, $i = 1, 2$, but the concept can be extended to other situation with more devices.

After installing this sync, for sure possible, between each speaker and the loud one, it doesn't matter that between the speakers "a" and "b", as seen before, cannot be sync: each person should hear just the loud speaker and the one close to them.

The only problem we can consider is this: the device connected to the loud speaker can make a broadcast streaming to install the sync with all the devices, or it should reserve a "private" streaming for each device? Of course, the broadcast is possible if the range that describes the distance between all the devices is less than 8 meters. In the other case, the broadcast is not possible.

5.4 Algorithm for synchronization

As already mentioned in the preface of this chapter, an algorithm of synchronization of an audio streaming is here presented. This algorithm does not take in account any delay caused by the latency of the channel. It results to work with mobile devices closed in a circle of ten meters of diameter. Because of the unidirectional flux of information from the server to the clients, the number of

phones able to sound at the same time is limited just by the dimension of the ad-hoc network.

The main criteria on which is based this algorithm is the equality between the sending rate and the playback rate. Actually by sending packets at the same rate in which they are supposed to be written into the player, the synchronization is achieved just with the help of a pre-buffering at the beginning of the transmission.

It is easier to explain how this algorithm works by taking in consideration an example.

A PCM audio file of 6.3 Mb is handled by the server. Its constant sample rate is equal to 11.025 Bytes/s, so utilizing a stereo channel a second of sound is described by 45.100 bytes. Having said that, the sending rate should be 45,1 Kb/s: a value easily manageable by the mobile devices described in the second chapter. By looking for a right dimension of the single packet and following the consideration stood out in the tests previously described, it has been chosen to fix that at 1280 bytes. In this way the server should send 35 packets per second (the reason of such choice are explained in the Section 3.6.3)

The server's job is therefore very easy. As seen in the server created for managing transmission of messages, it must send data with a loop managed by a timer that assures the sending rate. Each packet is constituted by three parts:

- The ID, used by the clients to check eventually missed packets
- The Total Number of packets, used by the clients to pre-install a buffer of silence – packets in the initialization of the client-player. -
- The 1280 bytes of the file audio.

One of the main differences with the official RTP implementation of an audio streaming is the lack of the clock. Actually the synchronization is assured just by the rate and by the "trick" of the silence-packet, pre-emptively stored in the client. The power of this kind of implementation is even the unidirectional flux of information: actually because of the broadcasting and of the algorithm used, it is not necessary by the clients to inform the server about their status of receiving.

Therefore each client, binding the network, can start to receive packets from the server and store this data into the buffer. As already mentioned, the clients must replace the pre-created silence – packets into the list with the right packets incoming from the network. After receiving one hundred packets, equal to 3 seconds of sound, the player starts to acquire strings from the list as seen in the previous chapter. While the player is on, the receiver will continue to update the same list handled by the player. The information of the 100th packets is

taken by storing the ID of the first arrived packet. When the ID of an incoming packet is shifted one hundred positions from the first one, the player will be called. It is not important if clients start the player in different moment: the synchronization is assured by the timer placed in the server.

5.5 Latency of the channel

As reported in the previous section, the timer plays an important role in the dynamic of the algorithm able to manage the synchronization. It is necessary the right interaction between the server and the client in relation with time.

The timer, manager of the server, assures the launch of an instruction in the code able to send a message. The timer assures the constant repetition of this action each N seconds, where N represents the amount of sound, in time, carried by a packet. At the same time, this timer cannot assure the arrival of the packet to the receiver with the same previous rhythm. It is necessary to check out how often packets arrive to the receiver.

Because of that, it has been made a test in which is measured the time that a packet spend from the launch of the instruction of the server phone to the moment in which the data, carried in the packet, is available in the client phone. This time T can be seen as the sum of three components, $T = T_s + T_p + T_r$, and they represent:

- T_s: time of elaboration of the server phone from the launch of the instruction to the arrival of this data to the physic layer that release it as electromagnetic wave into the air.
- T_p: time of propagation of the electromagnetic waves from the antenna of the server phone to the antenna of the client phone. Considering the characteristics of the background not so different of the vacuum, the velocity of these waves is c, "speed of light", $3x10^8 m/s$. In a range in which the synchronization is requested, distances can arrive to a dozen of meters. This means that Tp is about 10^{-7}, so negligible.
- T_r: time of elaboration of the client phone from the moment in which the antenna receives data to the arrival of the packet in the application layer.

T has been characterized with a distribution of probability shown in the next figure.

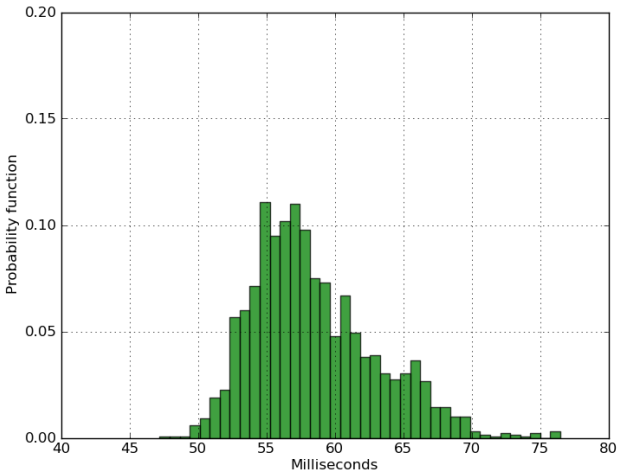


Figure 5.6: Approximation of the probability density function of the parameter T, described in this section

Mean	58.44 ms
Variance	20.52 ms
Standard deviation	4.53 ms

Table 5.1: Main statistical parameters of the Probability density function represented in the previous graph.

The previous results are well accepted since they are proving the right behaviour of the server and the clients in exchanging packets in the proposed algorithm. The variance, associable even to the jitter, is less than the 35 ms, limit found by Haas, in which the human ears start to feel delay. This results can also give us an indication about how big should be the pre-buffering in the clients, before starting the song. A buffer of 100 ms is much bigger than the delay recorded in this test.

5.6 Conclusion

In this chapter, the goal of the whole project has been achieved: an algorithm able to manage a streaming for a synchronized audio between several mobile devices. This chapter wanted to trace a path about all the main problems of such implementation. Besides the proposed algorithm is just an approach of a possible implementation.

Having explained the effect treated by Hass, three possible scenarios have been shown. For each of them, it has been considered whether an implementation of a streaming in synchronization is possible or not. Afterwards the algorithm presents a streaming between mobile phones of a PCM file. In this chapter the reliability of the network has not been analyzed but, however, the proposed algorithm is the one that offers best performances without considering any coding schemes.

Chapter 6

RELIABLE MULTICAST IN AD HOC NETWORKS

6.1 Preface

In the report, an algorithm has been introduced to realize a synchronized audio streaming. As some reader could have noticed, no chapter discusses about the possibility to correct any errors present in the communication, neither a possible interaction between the clients. In the second chapter, it has been proved that UDP performances, in a MANET composed by our specific mobile phones, show low amount of losses, under specific conditions. In the third chapter, some ways to prevent interferences in the audio streaming have been proposed and it was discussed how some losses can be tolerated in streaming PCM files. Then it is necessary to specify how important can be to receive all the transmitted packets when the streaming involves other audio formats: in those cases high reliability in MANET is strongly requested. This chapter wants to investigate in some existing coding schemes and proves the utility of some of them in our context.

As already mentioned, the UDP transport layer cannot avoid the possibility of a significant number of packets lost, even though it has been shown how the efficiency can increase by finding out the right size of the single packet. By thinking to an unicast communication, the automatic repeat request (ARQ) is the way to improve reliability. Actually, in that case, by using acknowledgments and timeout, the error control mechanism can assure the recovering of packet losses. Instead, by thinking to a multicast communication, as proposed in the report, the high number of requests can easily provoke a congestion of the network. Overall because an only server cannot manage the retransmission of different packets to different destinations. Because of that, some coding schemes are needed to manage such a problem. In this chapter the Network Coding (NC) and the Erasure Coding (EC) will be analyzed.

6.2 Network Coding and Erasure Coding

NC and EC are coding schemes able to increase reliability in multicast communication. They allow the server to inject redundant packets into the network so that clients can generally recover original packets without asking for retransmission. By encoding a big amount of original packets in a data stream (potentially infinite), the schemes allow the clients to reconstruct the original data after collecting those and exploding their redundancy. The main difference between these two technologies is the topology of the network they are able to build. The EC uses a "star" topology, with the only server, owner of the source, able to add redudancy to the data before of trasmitting it to the clients. Clients have the only role to broadcast again what they receive. Besides, the NC allows intermediate nodes to broadcast what they receive after pushing data in a new indipendent encoder. The topology in this case seems to look like a "tree" network, sometimes able to assume a fully connected composition.

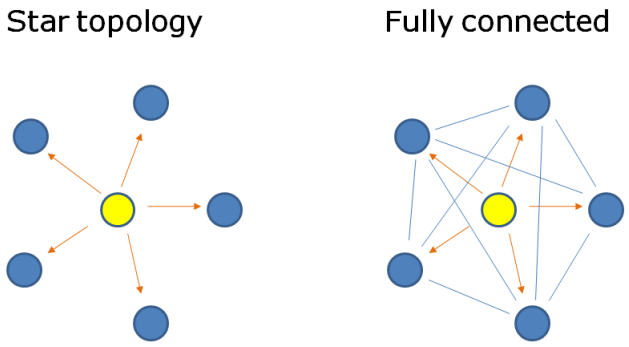


Figure 6.1: Examples of network topologies.

Simulation

To characterize and describe those coding schemes, it has been referred to the simulations made in []. It is assumed that an application generates equal size packets p_0, p_1, p_2. The stream of original packets is slit into "generation" of k packets. Thus, it is possible to define a code rate as $c = k/n$, when k packets in the same generation are encoded into n packets ($n > k$) at the source. In this argumentation, the code rate is considered just for EC, since the NC source does not generate any redundancy, so its code rate is always 1. The server of a NC in this example just encode the data as much as all the clients in the network. Thus each nodes in EC forward packets at most n times while NC at most k times for generation.

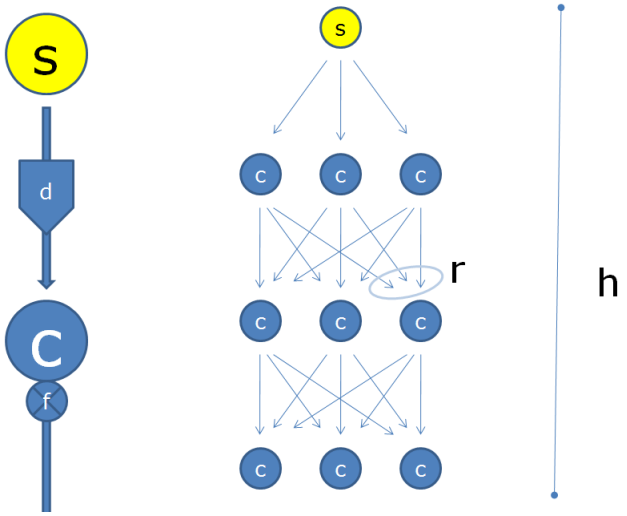


Figure 6.2: Some of the parameters useful to describe the coding schemes.

Each node of the network, except the first hop nodes that receive packets directly from the source, has r redundant paths. The number of hops from a source to receivers is defined by h. Each connection between two nodes can be characterized by the packet drop probability d $(1 > d > 0)$, and each node forward packets with a certain probability f, forwarding probability $(0 < f < 1)$. It is necessary to include the event of "not forwarding" because it is important to prevent excessive transmission overhead in the network.

Having said that, let denote N_{NC} and N_{EC} the number of non-duplicated packets each node can potentially receive.

$$N_{NC}(1) = (1-d) \cdot k$$

$$N_{EC}(1) = (1-d) \cdot n = (1-d) \cdot \frac{k}{c}$$

After defining the number of packets received by the nodes directly connected with the source, it possible to extend to all the cases.

$$N_{NC}(h) = \begin{cases} f \cdot r \cdot (1-d) \cdot N_{NC}(h-1) & if\ N_{NC}(h) \leq k \\ f \cdot r \cdot (1-d) \cdot k & if\ N_{NC}(h) \geq k \end{cases}$$

$$N_{EC}(h) = N_{EC}(h-1) \cdot (1-d) \cdot (1-(1-f)^r)$$

It is necessary to spend some words about these formulas. As seen, theN_{NC} shows how it is important the number of r, since it affects directly the number of packets. The distinction between the two cases keeps count of the possibility

of some losses before the arrival in that specific node. That same important role in the N_{EC} is interpreted by the code rate.

Results

In this subsection, a comparison between these two coding schemes is introduced in order to define which is the most useful for our application. The metrics utilized to compare them are:

- *Packet Delivery Ratio* (PDR) is fraction of recovered packets averaged over all receivers.
- *Normalized Packet Overhead* is the total number of packet transmissions by the network divided by the total number of data packets actually recovered.

In the end, to describe a random topology, utilized by [], is necessary to specify the *node density* as the average number of nodes within the transmission range:

$$node\,density = \frac{\pi \cdot (transmission\,range)^2}{(field\,size)/(number\,of\,nodes)} - 1$$

For each graph, a special attention is given to the case more close to our application.

Delivery ratio as function of the code rate

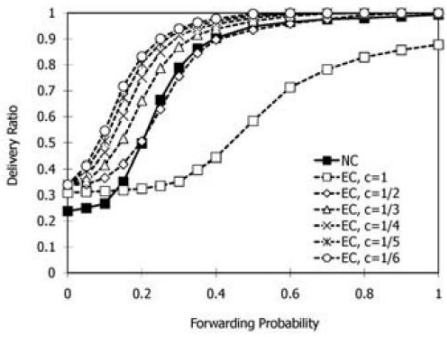


Figure 6.3: Delivery ratio of NC and EC in a random topology with node density = 12, no packet drop probability, and no node mobility. As shown, the EC performances are much better than the NC ones when the code rate is less then 1/4.

Normalized overhead as function of the code rate

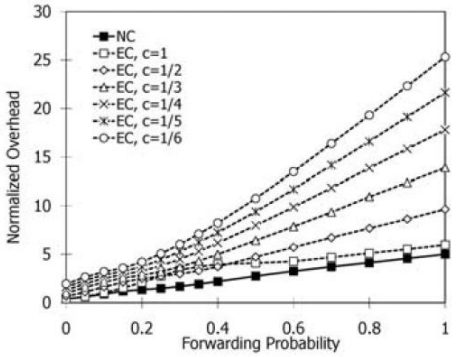


Figure 6.4: Delivery ratio of NC and EC in a random topology with node density = 12, no packet drop probability, and no node mobility. As shown, the overhead of EC is much bigger than the NC as code rate becomes smaller. Unless the EC 's performances are quite better, the increasing of the overhead can be a problem.

Delivery ratio as function of the node density

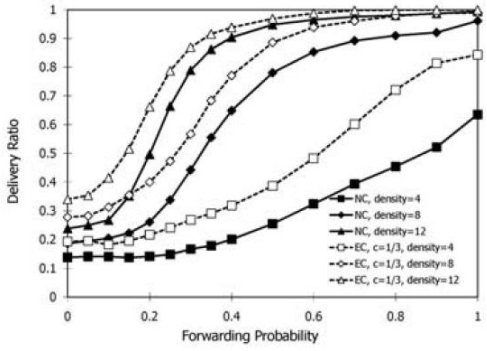


Figure 6.5: Delivery ratio of NC and EC in a random topology varying node density with no packet drop and no mobility. In our application the Node density is equal to the number of clients, so by considering a number of clients quite big (at least 8), the performances of these schemes are similar.

Delivery ratio as function of the packet drop probability

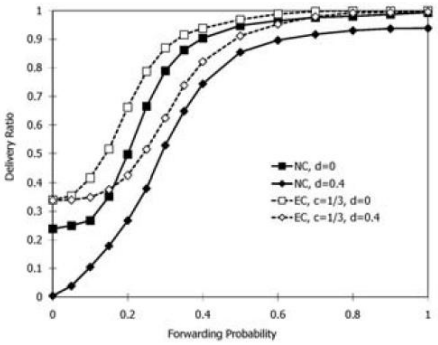


Figure 6.6: Delivery ratio of NC and EC in a random topology varying packet drop probability with node density = 12 and no mobility. In our application the drop probability is greater than zero: the EC, with a code rate = 1/3 has better performances, than NC.

6.3 Analysis

After describing how these coding schemes work and how some parameters can better analyze their performances, in this section it will be done an analysis of our application with regard to those coding schemes. The first consideration that should be done is the difficulty to define right or wrong one of the protocol described. Usually, the Erasure Coding is excluded because of its high overhead, as seen in the previous graph, caused by the high redundancy introduced by the server in the network. Our application cannot really hit this problem. By fixing the sending rate equal to the playback rate, the overhead will never reach tricky values. Actually, it has been proved the necessity of a reliable network once it is changed the audio format of the source file. A PCM file, with its quite big dimension due to its linear quantization, has a size for sure bigger than other possible audio format. So, by replacing PCM with MP3 or AAC, the amount of bit to send each second is less.

At the same time, the previous graphs showed how the Network Coding offers the same performances of the Erasure coding even though the first one does not put any redundancy into the network. Then, why do not use Network Coding as default coding schemes of the network? It has been seen how the performances of receiving of each client is stressed when they are multitasking. It is allowed then to think how a mobile phone can be stressed while it receives, decodes packets, encodes packets, sends again and at the same time it is supposed to play music.

A last consideration is needed before to explain how our protocol works. By assuming the only use of the erasure coding, it can also create problems when one of the clients has very bad connection with the server. Actually if a body is in-between a sender and receiver the data - rate drops significantly. Then, without any "help" of the neighbor, this unlucky client cannot achieve any good audio streaming. Because of all these reasons, a mixed scheme is proposed: a scheme that can assume EC or NC, by depending on the topology of the clients. Before describing how this protocol can work is necessary to make an assumption. The hops in the NC will never be greater than 2 and it is not assumed the presence of devices that a priori cannot receive anything by the server because of the distance. Then two configurations are proposed, in relation to the formula previously proposed in the text:

- ERASURE CODING with 1 hop (server-clients)
- NETWORK CODING with 2 hops (server → clients + "best receivers" clients → clients)

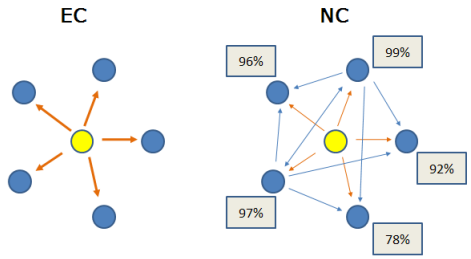


Figure 6.7: Examples of network topologies.

By taking back the formulas used previously, each node can potentially receives:

$N_{NC} = (1-d) \cdot f \cdot r \cdot N_{NC}(1) + N_{NC}(1) = ((1-d) \cdot r \cdot f + 1) \cdot N_{NC}(1) = ((1-d) \cdot r \cdot f + 1) \cdot (1-d) \cdot k$

$N_{EC}(1) = (1-d) \cdot n$

By having a ratio of this two values, it is possible to see when the number of received packets is the same:

$\frac{N_{EC}(1)}{N_{NC}} = \frac{(1-d) \cdot n}{((1-d) \cdot r \cdot f+1) \cdot (1-d) \cdot k} = \frac{n}{((1-d) \cdot r \cdot f+1) \cdot k} = \frac{1}{((1-d) \cdot r \cdot f+1) \cdot c} = 1$

$((1-d) \cdot r \cdot f + 1) \cdot c = 1$

That means that by having a number of 8 clients, with the half candidates to forward packets in Network Coding configuration and by supposing dropped 40 packets over 100 sent by server, the code rate of the Erasure Coding should be 1/2.5.

$$\frac{1}{((1-0.4)\cdot 5\cdot 0.5+1)} = c = \frac{1}{2.5}$$

How to allow the mobile devices to switch between these two schemes? In the normal flux of information spread by the server, some generations of packets, known by the clients, are sent with regular interval. For instance, the first can be sent once the UDP connection is defined. By receiving this generation, each client can know how much the amount of packet losses is. This percentage can be recorded in an internal table and the broadcast this value to all the nodes of the network.

The first node to be interested in these information is the server that, from those, must decide which schemes is more appropriated. The presence of only one client, not able to receive a percentage of data greater than an established threshold, will oblige the server to impose NC as code schemes for the network. Otherwise it starts to broadcast data with a redundancy defined by the code rate c, just calculated by using the received percentages.

If the Network coding has been established in the network, it is necessary to define which are the clients supposed to spread again the information. As done by the server, each client can build a rank of the percentage received by the other clients. If the j-client is in the first m position of the rank, it have to forward packets as soon as it receives them.

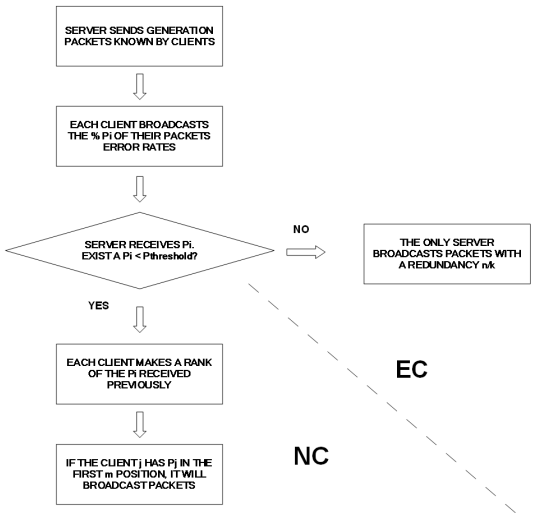


Figure 6.8: Explicanting diagramm of the proposed protocol

The last thing to define in such protocol is how to manage the synchronization even though the increasing of the traffic jam into the network. Previously it has been shown how the synchronization was on by fixing the sending rate equal to the play back rate.

In the EC the server will send all the generation of n packet in a time necessary to play that generation. The client, this time, cannot start to play as soon as it receive all the generation, because it is suppose to encode that data. For sure the time of encoding has to be less than the time of sound contained in one generation. So, each client starts to play the generation n.1 as soon as it receive the first packet of the generation n.3 from the server.

The synchronization is assured for the same reason it works the one implemented in the previous chapter with PCM file.

The arrival of a packet of a new generation from the server is the same principles which assures synchronization even in the case of NC. In this second case it is probably necessary to expand the time of buffering before playing the song.

6.4 Conclusion

In this chapter, the goal was to find a way to give reliability to the working solution found in the previous chapter. Because of that, two common coding schemes have been analyzed to verify which could be the best to solve our problem. By analyzing some tests, it has been proved how Erasure Coding and Network Coding are similar performances in background proposed by us.

Because of that, an algorithm has been proposed to make the ad hoc network more reliable, an algorithm able to switch between the two mentioned coding schemes.

It has been proved actually how the network coding is essential in such background because it is possible a worse reception by one or more clients. However, the application considers also the use of the erasure coding for avoiding to the mobile an excessive effort.

Chapter 7

CONCLUSION

In the first chapter of this report the goal of the project has been defined: Implementation of a stereo distributed between mobile devices.

In order to achieve an audio streaming characterized by a synchronized reproduction three requirements were requested:

- An application suitable for the platform where it has to run;
- An algorithm with a low complexity;
- A communication as much as possible reliable between the nodes of the network.

To achieve the first requirement, an investigation through the ISO/OSI standard has been done in order to figure out which protocols were the most suitable for such implementation. By building a mobile application that is running on the Symbian/S60 platform and tested on NokiaN95, it has been necessary to tighten the choice of the protocol to the ones available on the phone.

The Wi-Fi 802.11 b/g protocol has been preferred at Bluetooth because it can perform broadcast connections and can reach longest distances.

The UDP has been preferred at TCP because it can perform easily and quickly connections with many users.

As Application Layer, instead of using the already existing protocol RTP, a new algorithm from scratch has been implemented in order to avoid complexity. Another important reason is that already existing RTP modules able to run on Symbian Platform are hardly available.

The complexity has been kept low by using a programming language like Python. Actually, even though it is characterized by a pretty high level, it is very useful to implement UDP connection and perform streaming application.

The module UtilityAudioStream has been used in the implementation because of its flexibility in switching audio formats and its easy control functions. In the report, an analysis of the module is presented with a particular attention of its interaction with Python's variable:

- The best way to buffer data in Python
- Which variable is the most suitable to contain the audio file

To achieve the third requirement of the report, it has been decided to stream a PCM file. Its linear quantization and constant sampling allow the reproduction of sound even though some chunks of file are affected by errors. Actually it has been proved how a PCM file can look like undistorted even though the 33% of the chunks in the file are missing. That is possible if these chunks are spaced in time: the Interleaving technique showed how is always possible to have such situation.

In order to stream other audio format a coding scheme has to be implemented in the network. The supremacy of the Network Coding on the Erasure Coding is questioning by a high complexity and the background in which the application is supposed to run. Actually, by avoiding several hops between the sinks of the network, the two schemes have similar performances. The high overhead of the Erasure Coding is balanced by the complexity of the second one. Because of that an algorithm has been performed able to switch between the two ones. When one client has very bad performances in receiving data from the source-server, the Network Coding is used; in the other case, the Erasure Coding is enough (by considering the range of 10 meters where the application is supposed to run).

Having reached those requirements, it is possible to assert that **the prototype built successfully enables users to create a distribute stereo on mobile devices. Further the reproduction of the sound is synchronized between all the devices whether they are starting the application at the same time or one of those is joining on fly the network.**

7.1 Future improvements

Several suggestions are available to make this application able to be on the market.

First of all, it should be implemented an algorithm able to stream MP3 files because they are the most spread on the network. If this proposal is not possible, the MP3 file should be encoded in other audio formats more suitable for such implementation.

The reliability of the network is strictly requested in case of changing audio format. Because of that, the suggestion present in this report should be implemented. Probably its implementation should be done with a lower level programming language.

In order to achieve a good listening for each client it is suggested the use of an audio codec able to repair some eventual errors income in the network. Some of them are already available on the market.

7.2 New Future Applications

Several applications can be derived from this one to import new services on the phones. First of all, it is necessary to remind in which cases or activities this application is already useful like that.

Unless there is no big loud speaker, this application is very useful to make the sound louder.

In a trip made on the bus, young teenagers can decide to listen all together the same song.

In a studying afternoon, students can listen again the conference recorded the previous morning.

All the member of a family can talk with a relative that is calling from another continent.

Those were just example of ways to use this application in funny and smart ways. Of course other ideas can give rise to the creation of new applications based on wireless grid:

- Sharing the screens of the mobile phone to watch a video of a past experience;
- Streaming in real time a concert on the phone because the position is too far from the stage;
- Creating an internal entry videophone for a house.

Appendix A

ENERGY CONSUMPTION IN WIFI APPLICATIONS

A.1 Preface

The purpose of this appendix is to describe the energy consumption characteristics of various aspects and components of WiFi phones. By testing phones to verify the performances of ad hoc networks, the fast loss of power was pointed out. It is important to consider this aspect because phones with low battery life defeat the mobility functionality. Therefore it will be treated which are the main causes of this consumption and how to minimize the consumption while still maintaining the required quality of service. Then it will be shown the energy profile of a WiFi phone, while our application is running on it.

A.2 Evolution and future prediction

In the last years, mobile phones are becoming increasingly popular, but the battery technology has not been improving at the same pace as the power requirements of the devices. Some number can help to describe how the new technology has needed more and more power from the battery. The first generation of the phones in the nineties had a energy consumption relatively low in the range 1-2 W,. It has been increasing by reaching almost the twice value, with the developments of new multimedia devices incorporated in the phone: Camera, multicolor Display, Apps Engine, etc. And this requirement of energy is doomed to increase more. The perspective of transforming a phone in a modern notebook will bring the necessity to have a constant connection activated. The advanced video capabilities will need a powerful display and the spread of audio application (like the one described in this report) will probably need an incorporating stereo loudspeakers on the devices. [13]

A.3 Strategy to save power

As mentioned in the first chapter of this report, our application has been developing in five layers, by setting the ISO/OSI protocol stack. In this paragraph, in each of those layers, a possible solution to decrease the spending of energy will be researched.

A first way, as suggested in [11] is to control the transmission power parameters. By assuming the use of the same phone used in the previous testes (Nokia N95), some forethought can be applied.

- At the mac layer it is possible to reduce the number of retransmission by choosing a different channel/link;
- At the network layer, it is possible to use a topology control, allowing each wireless device to adjust its transmission range and select its appropriate neighbor (the "network coding" already discussed in the previous chapter).
- At the application layer, it is possible to reduce the number and size of transmission by using data compression.

A second way is to reduce the control overhead, that is the energy used to send and receive control packets. By using a transport layer as UDP, this energy saving is already guaranteed .

A.4 Wifi networks performance

For getting closer to the consumption of our application is necessary to describe the performance of IEEE 802.11 b/g in which it is working. To do that we take a look to some tests reported in [20]

Those measurements are performed on Nokia N95, the same phones used to test our audio application. To measure energy consumption, it has been used Nokia's energy profiling application, the Nokia Energy Profiler (NEP). NEP is an application running on mobile device that allows to make measurements without any additional hardware. It provides the values for power, current, temperature, and CPU usage. It has been tested that the measurements done on a common multimeter, as can be AGILENT 66319D, match almost perfectly with the ones of the NEP.

In one of those tests it has been measured the power consumption of WLAN 802.11g/b in infrastructure mode for different states when the distance between the mobile and the AP is 3-5 meters. In such a mode it is possible to distinguish four states:

- Connection: Turn on WLAN and connect to the AP
- Disconnection: Disconnecting from the AP and turning WLAN off.

- Idle: Being connected to the AP and in idle mode.
- Idle in saving mode: Being connected to the AP in idle mode and in saving mode.

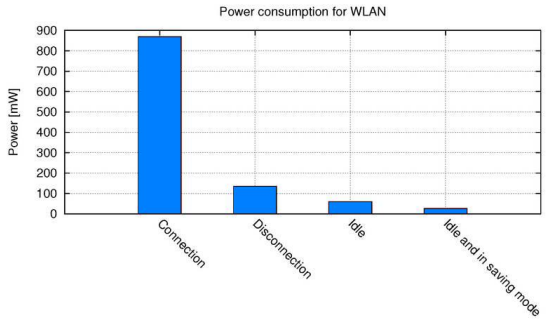


Figure A.1: Power consumption for WLAN in different states when used in infrastructure mode.

As we can see an huge amount of energy is spent on the idle state. In many applications, as for instance Voip, this state is required. When a smartphone is waiting for an incoming call the network interface has to be on all the time. Disconnecting the phone does not help: it will not be able anymore to catch incoming calls. From [13] another strategy to save energy in such cases it has been described. In that research it has been proposed the use of a secondary air interface to wake up the WLAN and different ways to implement it.

A.5 Testing the code

Having analyzed the main characteristics of such a problem, it is time to give a look to the energy consumption of our application by starting from the simple script client/server created in the second chapter in order to exchange messages between two phones. The results will be shown by analyzing the graph of the NEP on either sides: client side and server side.

A.5.1 Client side

In this subsection it will be explained how much the phone spends on the side of the client.

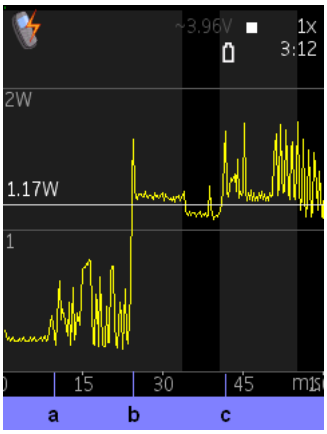


Figure A.2: The NEP shows the power consumption on the Client side while the transmission is starting.

In regard to the previous figure we can distinguish different phases:

- 00.00: on the phone, the NEP has just been activated (we can consider this offset like our hypothetical "zero")
- a: the Python application starts
- b: the phone is trying is connecting to the Ad Hoc Network
- c: the phone starts to receive packets

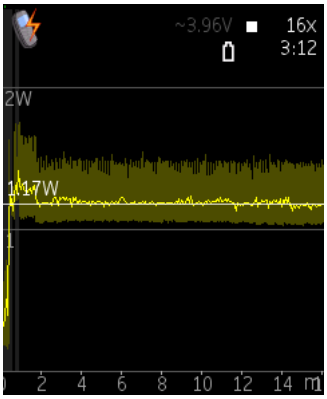


Figure A.3: The NEP shows the power consumption on the Client side.

In this second picture we can observe the constant band of consumption during all the receiving. The average is about 1.17W, one more than the initial phase.

A.5.2 Server side

In this subsection it will be explained how much the phone spends on the side of the server.

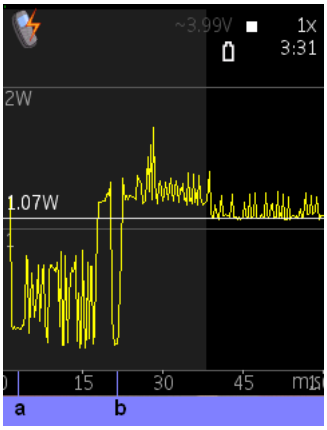


Figure A.4: The NEP shows the power consumption on the Server side while the transmission is starting.

In regard to the previous figure we can distinguish different phases:

- 00.00: on the phone, the NEP has just been activated: we can notice the instability of that phone if we compare it with the client side
- a: the Python application starts
- b: the phone is trying is connecting to the Ad Hoc Network and start to transmit packets

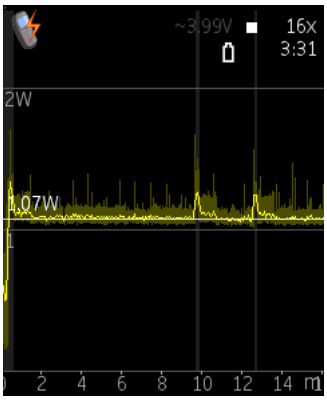


Figure A.5: The NEP shows the power consumption on the Server side.

In this second picture we can observe the band of consumption during all the receiving. The average is about 1.07W. As we can notice, this band is characterized by some fast peak, maybe caused by some interferences occurs in the transmission. As already mentioned, in both case, it is reasonable to think that the average level raises if the size of the packets increase.

Appendix B

CODE TO IMPLEMENT THE APPLICATION

B.1 Testing UDP connection: Server side

```
import btsocket
import e32
import appuifw
import time

f=open("C:\\Data\\python\\Nicola2.wav","rb")
data = f.read()
f.close()

while data[:4]!="data":
  data=data[1:]

data=data[4:]

host = "255.255.255.255"
port = 0
stop = 0
timer = e32.Ao_timer()
counter=10000
time_delta=1/30.0 #30packets for sec --> rate= 44100bytes/s
size_packet=1470
pointer_data=0
size_data=len(data)
server=None
dead_line=15000
```

```
def sender():
  global counter
  global pointer_data
  global server

  #print counter
  if counter%500==0:
    print counter

  msg=str(counter)+str(dead_line)+data[:1470]
  server.sendto(msg, ("255.255.255.255", port))
  pointer_data=pointer_data+size_packet
  counter +=1

def start_server():
  global port
  global server
  global timer
  global counter

if stop == 1:
  print "waiting for server to stop"
  return

  # Select AP
  apid = btsocket.select_access_point()
  apo = btsocket.access_point(apid)
  btsocket.set_default_access_point(apo)
  apo.start()

  print 'AP selected'

  # Create the socket
  server = btsocket.socket(btsocket.AF_INET, btsocket.SOCK_DGRAM)

  port = appuifw.query(u"Port:", "number")
  print "start"
  time_checker=time.clock()
  a=time_checker

while counter<=dead_line:
  while time.clock()>time_checker:
    time_checker=time_checker+time_delta
    sender()
    if counter > dead_line+1:
```

```
        break

    b=time.clock()
    print b-a
    print time_checker-a

    appuifw.app.menu = [(u"Stop", stop_server)]

def stop_server():
    global stop
    global server

    stop = 1
    print 'stopping server'
    server.close()
    appuifw.app.menu = [(u"Start server", start_server)]
    client = None

def quit():
    print 'Closing'
    app_lock.signal()

#### RUN ####
appuifw.app.menu = [(u"Start server", start_server)]
appuifw.app.exit_key_handler = quit

# create an active object
app_lock = e32.Ao_lock()
print "Ready to start"
server = None

# start a scheduler
app_lock.wait()
```

B.2 Testing UDP connection: Client side

```
import e32
import appuifw
import btsocket
from time import*

audio=None
fileSize = 0
musicfile = None
```

```
size_packet=1470
zero=pack('h',0) #2 bit of zeros
empty_packet=zero*(size_packet/2)
list_count=[]
list_double=[]
data=[]
count=0 total_number_packet=5000

while count<total_number_packet:
  data.append(empty_packet)
  count += 1

musicfile=data

# Set the socket parameters
host = "0.0.0.0"
buf = 1700

def udp_callback(d):
  global list_double
  global list_count

  string, addr = d
  count = int(string[:5])-10000
  musicfile[count]=string[10:]
  #print count
  if count%500==0:
    print count

  if count in list_count:
    list_double.append(count)
  else:
    list_count.append(count)

  client.recvfrom(buf, 0, udp_callback)

def start_client():
  global client
  global port
  client = btsocket.socket(btsocket.AF_INET, btsocket.SOCK_DGRAM)

  # Hack to avoid the permission denied on reusing sockets
  port = appuifw.query(u"Port:", "number")
  client.bind((host, port))
  client.recvfrom(buf, 0, udp_callback)
  appuifw.app.menu = [(u"Stop", stop_client)]
```

```
    print 'Client receiving on port ', port

  def stop_client():
    global client

    print 'stopping client'
    client.close()
    appuifw.app.menu = [(u"Start client", start_client)]
    #Create the file for results
    PATH = u"C:\\Data\\MyApp"
    if not os.path.exists(PATH):
      os.makedirs(PATH)

    textfile = file('C:\\Data\\MyApp\\test_35pack.txt','wt')

    print >> textfile, list_count
    print >> textfile, "DOUBLE"
    print >> textfile, list_double
    print >> textfile, "PACKETS RECEIVED"
    print >> textfile, len(list_count)
    textfile.close()
    client = None

  def quit():
    print 'Closing'
    app_lock.signal()

  #### RUN ####
  appuifw.app.menu = [(u"Start client", start_client)]
  appuifw.app.exit_key_handler = quit

  # Create an active object
  app_lock = e32.Ao_lock()

  # Select AP
  apid = btsocket.select_access_point()
  apo = btsocket.access_point(apid)
  btsocket.set_default_access_point(apo)
  apo.start()

  client = None

  # Start a scheduler
  app_lock.wait()
```

B.3 Streaming prototype: Server Side

```
import btsocket
import e32
import appuifw
import time

f=open("C:\\Data\\python\\Nicola2.wav","rb")
data = f.read(7135276)
f.close()

while data[:4]!="data":
  data=data[1:]

data=data[4:]

host = "255.255.255.255"
port = 0
stop = 0
timer = e32.Ao_timer()

counter=10000
time_delta=1/35.0 #35packets for sec --> rate= 44100bytes/s
size_packet=1260
pointer_data=0
size_data=len(data)
server=None
dead_line=len(data)/size_packet+10000

def sender():
  global counter
  global pointer_data
  global server

  #print counter
  msg=str(counter)+str(dead_line)+data[pointer_data:pointer_data+size_packet]

  server.sendto(msg, ("255.255.255.255", port))
  pointer_data=pointer_data+size_packet
  counter +=1

def start_server():
  global port
  global server
  global timer
```

```
    global counter

  if stop == 1:
      print "waiting for server to stop"
      return

    # Select AP
    apid = btsocket.select_access_point()
    apo = btsocket.access_point(apid)
    btsocket.set_default_access_point(apo)
    apo.start()

    print 'AP selected'

    # Create the socket
    server = btsocket.socket(btsocket.AF_INET, btsocket.SOCK_DGRAM)

    port = appuifw.query(u"Port:", "number")
    print "start"
    time_checker=time.clock()
    a=time_checker

    while counter<=dead_line:
      while time.clock()>time_checker:
        time_checker=time_checker+time_delta
        sender()
        if counter > dead_line+1:
          break

    b=time.clock()
    print b-a
    print time_checker-a

    appuifw.app.menu = [(u"Stop", stop_server)]

  def stop_server():
    global stop
    global server
    stop = 1
    print 'stopping server'
    server.close()
    appuifw.app.menu = [(u"Start server", start_server)]
    client = None

  def quit():
    print 'Closing'
```

```
  app_lock.signal()

#### RUN ####
appuifw.app.menu = [(u"Start server", start_server)]
appuifw.app.exit_key_handler = quit

# create an active object
app_lock = e32.Ao_lock()
print "Ready to start"
server = None

# start a scheduler
app_lock.wait()
```

B.4 Streaming prototype: Client Side

```
import utilaudiostream
import e32
import appuifw
import btsocket
import appuifw
import chunk
import wave
import audio
import os, os.path
import thread
import threading
from time import*
from threading import Thread
from threading import Lock
from threading import Condition
from struct import*

audio=None
fileSize = 0
musicfile = None
size_packet=1260
first_time=True
first_save=True
starter=0
zero=pack('h',0) #2 bit of zeros
empty_packet=zero*(size_packet/2)
d=None
data=[]
```

```
count=0
total_number_packet=6000 #6000*1260=7.56Mb

while count<total_number_packet:
  data.append(empty_packet)
  count += 1

musicfile=data

# Set the socket parameters
host = "0.0.0.0"
buf = 1500

def write_chunk():
  global audio
  global fileSize
  global d
  if fileSize >= len(musicfile):
    return
  d = musicfile[fileSize]+musicfile[fileSize+1]
  audio.write(d)
  fileSize += 2

def play_open(error):
  if error != 0:
    print "Error in play open ", error
    return

  #print "Player open"
  write_chunk()

def play_stop(error):
  print "Play stop", error

def play_data(error):
  if error != 0:
    print "Error in play data ", error
    return

  #print "Data written to player, write next chunk"
  write_chunk()

def play_file():
  global audio
  audio.open()
```

```
def udp_callback(d):
  global first_save
  global musicfile
  global starter
  global first_time
  global total_number_packet
  global fileSize

  string, addr = d
  count = int(string[:5])-10000
  if first_save:
    first_save=False
    starter=count
    total_number_packet=int(string[5:10])-10000

  musicfile[count]=string[10:]
  #print count

  if count-starter>100 and first_time:
    first_time=False
    fileSize=(count-100)
    play_file()

  client.recvfrom(buf, 0, udp_callback)

def start_client():
  global client
  global port

  client = btsocket.socket(btsocket.AF_INET, btsocket.SOCK_DGRAM)

  # Hack to avoid the permission denied on reusing sockets
  port = appuifw.query(u"Port:", "number")
  client.bind((host, port))
  client.recvfrom(buf, 0, udp_callback)
  appuifw.app.menu = [(u"Stop", stop_client)]
  print 'Client receiving on port ', port

def stop_client():
  global client
  print 'stopping client'
  client.close()
  appuifw.app.menu = [(u"Start client", start_client)]
  client = None

def quit():
```

```
    print 'Closing'
    app_lock.signal()

  #### RUN ####
  appuifw.app.menu = [(u"Start client", start_client)]
  appuifw.app.exit_key_handler = quit

  # Create an active object
  app_lock = e32.Ao_lock()

  # Select AP
  apid = btsocket.select_access_point()
  apo = btsocket.access_point(apid)
  btsocket.set_default_access_point(apo)
  apo.start()

  client = None
  audio = utilaudiostream.AudioStream(utilaudiostream.DATATYPE_PCM16,
utilaudiostream.SAMPLERATE_11025Hz,utilaudiostream.CHANNELS_STEREO,
play_open, play_stop,play_data)
  audio.setvolume(10)

  # Start a scheduler
  app_lock.wait()
```

Bibliography

[1] : S60 Smartphone Quality Assurance; Laitinen; 2007

[2] : Mobile Phone Programming and its Application to Wireless Networking; Fitzek, Reichert; 2007

[3] : Computer networks; Tanenbaum, A.S.; 2003

[4] : Understanding Jitter in Packet Voice Networks (Cisco IOS Platforms)

[5] : RTP, Audio and Video for Internet; Perkins; 2003

[6] : The technology of Video & Audio Streaming; Austerberry; 2002

[7] : Mobile Python; Scheible,Tuulos; Wiley; 2007

[8] : Foundation of Python Network Programming; Goerzen; Apress; 2004

[9] : Learning Python, Powerful Object-Oriented Programming; Lutz; 2008

[10] : Wireless Grids, Distributed Resource Sharing by Mobile, Nomadic, and Fixed Devices; McKnight, Howison; Bradner; 2004

[11] : Energy Consumption and Conservation in WiFi Based Phones: A Measurement-Based Study. Gupta A. / Mohapatra P. 2007

[12] : Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. Balasubramanian/ Venkataramani /2009

[13] : Energy Saving Strategies for Mobile Devices using Wake-up Signals; Perrucci, Fitzek, Sasso, Katz; 2008

[14] : A Case for Network Musicl Performance; Lazzaro, Wawrzynek; 2001

[15] : Evaluation of packet latency and fluctuation during UDP packet exchange in ad hoc wireless groups; Itaya, Kosuga, Davis; 2004

[16] : Network Coding vs. Erasure Coding: Reliable Multicast in Ad hoc Networks; Fujimura, Oh, Gerla; 2008

[17] CodeCast: Network Coding Based Multicast in MANETs; Gerla, Chen, Lien; 2008

[18] : Know Your Neighbour: Packet Loss Correlation in IEEE 802.11b/g Multicast; Heide, Pedersen; Fitzek; 2008

[19] : PictureViewer - A Mobile Application using Network Coding; Pedersen, Heide, Fitzek, Larsen; 2009

[20] : Wifi: consumption for different sates; http://mobiledevices.kom.aau.dk/research/energy_measurements_on_mobile_phones../../results/data_communication/wi_fi_different_status/

[21] : 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.

[22] : "Mobile distributed Wireless Stereo", report written in 2009 by Elisabeth Berbel Gonzales with the same supervisor of this report.