Escola Politècnica Superior de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

# DIPLOMA THESIS

TITLE: "Linux-Box: DVB and VoD Streaming Over Local Area Networks"

DEGREE: "Enginyeria Tècnica de Telecomunicació"

AUTHORS: Juan José Colmena Pablos, "Comunication Systems"
Óscar Molina Fernández, "Computer Networks"

Director: Gregorio Procesi
Tutors: Michele Pagano
David Rincon Rivera

DATE: May 14, 2007

**Títol:** "Linux-Box: DVB and VoD Streaming Over Local Area Networks"
**Autor:** Juan José Colmena Pablos
Óscar Molina Fernàndez

**Director:** Gregorio Procesi
**Tutors:** Michele Pagano
David Rincon Rivera

**Data:** 14 de Maig de 2007

## Resum

Aquest treball tracta sobre un projecte comú anomenat Linux-Box portat a terme per diferents persones al departament de Telecomunicacions (IET) de la Universitat de Pisa. Linux-Box és un sistema dotat amb targetes TDT (DVB-T) i de televisió per satèl·lit (DVB-S) que permet transmetre aquestes senyals fins a un àmbit domèstic. Més endavant podria ser utilitzat en àmbits privats com les cases de clients o en institucions públiques com escoles, universitats, biblioteques i també seria possible en àmbits empresarials. El projecte està dividit en 4 apartats:

**1. Ubuntu 6.06 LTS.** Explica perquè s'utilitza Ubuntu en el projecte. A més també s'explica de forma breu que és Linux i les distribucions més utilitzades.
**2. Multimedia Network Protocols:** s'expliquen els diferents protocols desde la capa de xarxa fins la capa d'aplicació que s'utilitzen en el projecte Linux-Box. Aquests protocols son utilitzats tant en streaming, com en anunciació, unicast/multicast, encapsulat de vídeo i codecs. Els diversos temes tractats aquí es fan amb el propòsit de comparar i no només com a recerca teòrica. A la fi es veuen els programes utilitzats en el projecte per analitzar el tràfic de la xarxa.
**3. Linux-Box:** s'explica el funcionament i els objectius globals del projecte. Es dedica un sub-apartat a "VideoLan - VLC" part important a nivell de sofware. Més endavant es parla de les característiques de la Linux-Box de forma acurada: streaming de VoD i senyals de TV i s'analitzen els problemes coneguts i les seves solucions proposades. A la fi s'enumeren els llenguatges de programació utilitzats al projecte i en quina part s'utilitzen. Observarem que és una aplicació on diversos llenguatges de programació estan contínuament solapats.
**4. Developed Part:** es posa en pràctica la teoria estudiada a la resta del treball. Està dividida en 4 seccions:

- Desenvolupar una aplicació en codi C per convertir la llista de Canals (tan terrestre com de satèl·lit) en format XML.
- Una secció dedicada al streaming de Canals de TV a la pàgina web principal.
- Un anàlisis profund dels paquets creats per la Linux-Box i la seva activitat a la xarxa.
- Finalment s'analitzen els diferents scripts i les seves configuracions. Alguns són útils per a un futur desenvolupament i d'altres s'utilitzen en seccions prèvies.

**5. Conclusions:** conté les conclusions i línies futures. El projecte compta amb diverses opcions que encara poden ser implementades i estudiades. Aquí exposem les nostres interpretacions i possibles línies futures d'estudi.

**Title:** “Linux-Box: DVB and VoD Streaming Over Local Area Networks”
**Author:** Juan José Colmena Pablos
Oscar Molina Fernàndez

**Director:** Gregorio Procesi
**Tutors:** Michele Pagano
David Rincon Rivera
**Date:** May 14, 2007

## Overview

This work is about a common project carried on by different people in the IET department of Pisa University, called Linux-Box. Linux-Box is a device provided with DTT (DVB-T) and satellite television (DVB-S) sources that permits to transmit these signal in a domestic area. Further on will be reliable in private homes and public places such as schools, universities, libraries and environments like enterprises or offices. The work is divided in the following sections:

**1. Ubuntu 6.06 LTS:** In this section is explained why is used Ubuntu in the project. Also it is explained what is Linux and their most famous distributions.

**2. Multimedia Network Protocols:** in this section diverse protocols are explained starting from the network layer until the application layer that appear in the Linux-Box project related with streaming, announcement, Unicast/Multicast, video encapsulation and codecs. The diverse topics will be treated with a spirit of comparison and not only as a theoretical research. At the end there will be treated each other programs involved in this work to analyze the network traffics.

**3. Linux-Box:** In this section is explained what is their operation way and the objectives of the global project. It is dedicated a sub-section to the main part of the working mode at software level: “VideoLan-VLC”. Later on, the Linux-Box characteristics are treated deeply: streaming of VoD and TV signals, known problems analysis and possible solutions. To finish, programming languages involved in the global thesis are enumerated and put in the global situation. It is interesting to see that it is a truly distributed application where diverse programming languages are constantly interlaced.

**4. Developed Part:** Here the theory studied in the rest of the research is put in practice. It is divided in four sub-sections:
- Developing an application in C code to convert channels lists (as much as Terrestrial as Satellite) to XML format.
- Section dedicated to the TV channels streaming in the main Web Page.
- Deep analysis of the created packets by the Linux-Box and their activity in the net.
- Finally different scripts with different configurations are tested. Some of them are useful for future developments and others used in the previous sub-section.

**5. Conclusions:** This section contains the conclusions and the future lines. The project has many options that can be implemented and studied. Here are explained our interpretations and future possibilities.

# Introduction

This work is about a common project carried on by different people in the IET department of the Pisa University (UNIPI), called Linux-Box.

What is Linux-Box?
Linux-Box is a device provided with Digital Terrestrial Television (DTT) also known as (DVB-T) and satellite television (DVB-S) sources that allows the transmission of these signals in a domestic area. Further on, it could be released in private homes and public places like schools, universities, libraries and environments like companies or offices.

The example that gave us to understand the project at the beginning was the first time that the Linux-Box was used in the University of Pisa (UNIPI). That was to see the world soccer championship final match in 2006 by an IP stream from a laboratory for the teachers of all departments. This was possible thanks to the provided hardware of previous thesis and the creation of a script for the occasion.

The Linux-Box project is still under development. Our objective is to collaborate to the main project with our knowledge, to acquire responsibilities and make specific tasks to the common project. Linux-Box project should be considered as a future commercial product in a middle-long term, always based on open-source code.

The next figure shows a representation of "all the parts" of the whole Linux-Box project. It is organized in three parts: the sources (Hardware), the processing (Software and Web Page) and the destinations (streaming).

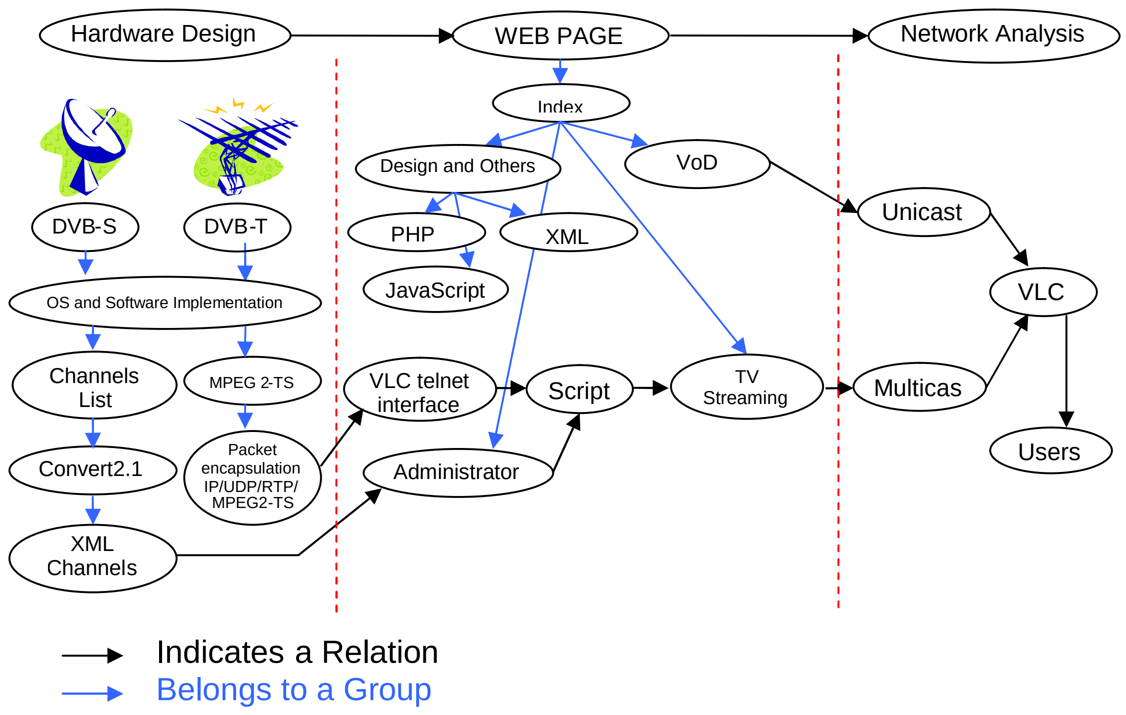


**Figure 0.1: General Parts Scheme of the Linux-Box Project.**

Why Linux?

The Linux-Box runs a Unix operating system, it was highly recommended to make anything related with the thesis in Linux instead of Windows to adapt ourselves easily and understand how it works. This was a great handicap for us because our Linux knowledge was at a very basic level. This change was expensive in time for us and also forced because the computers provided by the university worked on Linux, then the only solution was to adapt as soon as possible.

The thesis is divided on a first theoretical part of research and comparison between protocols. The second part explains what Linux-Box is and the main parts on the common project. To finish, there is a third analytic part in which an application is programmed and plots of net traffics created by the Linux-Box are analyzed.

The are two main parts with the following sections:

## Part 1: THEORETICAL RESEARCH

1. Ubuntu 6.06 LTS: In this section is explained why Ubuntu is used in the project. It is also explained what is Linux and its most famous distributions.

2. Multimedia Network Protocols: in this section there are explained diverse protocols starting from the network layer until the application layer that appears in the Linux-Box project related with streaming, announcement, Unicast/Multicast, video encapsulation and codecs.

The diverse topics will be compared and treated as a theoretical research. At the end will be seen diverse programs used in this work to analyze the network traffics.

## Part 2: APPLICATIVE PART

This part explains the global parts of the Linux-Box. Sections 4 and 5 explain our specific developed parts.

3. Linux-Box: In this section is explained the operation way and the objectives of the thesis. It is dedicated a sub-section for an important part of the working mode at software level: “VideoLan-VLC”. Later, it is treated the Linux-Box characteristics accurately: streaming of VoD and TV signals, known problems analysis and possible solutions. Finally the programming languages used in the thesis are enumerated and put in the global situation. It is interesting to see that it is a truly distributed application where diverse programming languages are constantly interlaced.

4. Developed Part: Here we have put in practice what we have studied in the rest of the work. It is divided in four sub-sections:

- Develope an application in C code to convert channels lists (as much as Terrestrial as Satellite) to XML format.

- Section dedicated to the TV channels streaming in the main Web Page.
- Deep analysis of the created packets by the Linux-Box and their activity in the net.
- At the end are tested different scripts with many configurations. Some of them are useful for future development and others used in the previous sub-section.

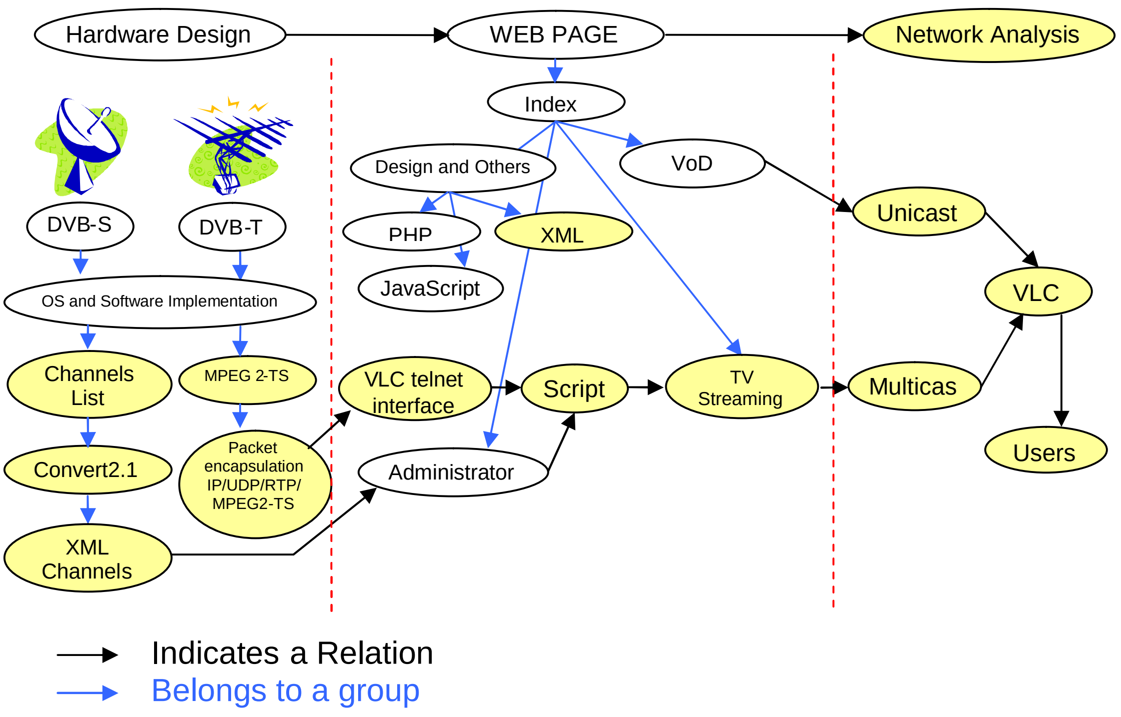


**Figure 0.2: This graph shows the parts developed in this project in yellow.**

<u>5. Conclusions:</u> This section presents the conclusions and the future lines. The thesis is young and has many options that can be implemented and studied in future. Here our interpretations and possibilities are explained.

This work is written in English because was the most common language between the University of Pisa, our tutor in Pisa and our University in Barcelona.

At the same time of this work, it was in process another parallel work with the Linux-Box but in another direction, creating the web interface between the final user, the server and the VoD part. There have been contacts by e-mail and work together in the labs very often to know the status of the global work, and different software adaptations requests.

# Part 1: Theoretical research

# 1. UBUNTU 6.06 LTS

The inclusion of this section in the project is decided for several reasons. The first is that only start the project the tutor's asked us to use any Linux distribution to familiarize with the Linux-Box and with the computers of the laboratory that were only in a Linux environments. Arrived to this point, the whole program code had to be programmed under Linux for strict demand of the tutor. This supposed a superior spend of time to the expected because a previous adaptation could be done to the new environment.

In this section it is also included Linux because it is not part of any topic of any subject of the degree and then it was something new for us, although there were some subjects that treat some Linux anyone has an specific treatment.

Our election together with the tutor was to install the OS Ubuntu 6.06 LTS in our portable computer. It was also installed already in most of computers of the laboratory. The decision was based on its easiness use, its idea of being used with each user's mother language and for its growing popularity.

In this section is explained what is Linux, which are their more famous distributions and later the main different environments that can be used. Also are explained in the Annex basic commands that were good for us to familiarize with the OS. Always making special attention in Ubuntu, which was used by us.

## 1.1. Linux

### 1.1.1. What is Linux?

Linux is an operating system compatible with UNIX. The system core is the kernel, and thanks to the GNU project many software applications have been made. That’s why many people call Linux as GNU/Linux. The first kernel was initially created by Linus Torvalds in 1991, with the first release 0.02, but it is still in developing and now the last stable version is 2.6.

### 1.1.2. Why Linux?

There are many reasons that make Linux suitable, so here only are mentioned those who think are more important:

- Linux is distributed under the GNU General Public License. This means that its source code will always be accessible. This contributed that every time are more free software applications with better quality.
- People can customize the software the way they want. Also this software can support many languages and people with disabilities.
- Many strong software companies like IBM, Sun, HP, etc… collaborate with the project and invest money and time developing the code.
- Support many microchips architecture (Pentium, Amd, etc...) and can also be integrated directly in a process called “embedding”.

- Linux is multitasking, multi-user, multiprocessor and protects memory between processes among its multiple functionalities.

This information is based on [1].

## 1.2. Distributions

The first Linux version basically consisted of its kernel and some GNU tools. Thanks to the effort of individual programmers, university students, companies and others, Linux began to be distributed with many optional software packages around the kernel depending on the own necessity. This is where the “distribution” concept was born.

Today there are many distributions of all types to choose, so the final user can download that who thinks better for his use. There is also Linux versions like Knoppix that is a live CD version to see the capabilities of Linux and has everything you can need on the computer without any installation. If you want to know more see [2].

The purpose of next pages is to explain the most famous distributions, their strong points and philosophy, with special mention to Ubuntu cause is the distribution used in the project:

### 1.2.1. Red Hat

Red Hat is one of the most famous distributions actually. Apart from all benefits that a distribution based on Linux can afford, its success lies in a support all time for companies and users in many languages. For this support should pay a price than can rise from 170€ to 3000€ depending on the distribution type.

So this distribution is overalls recommended to enterprises that need assurance of stability, support for any inconvenient at any time and also good technology integration like the open source provide. Red Hat provides a standard platform where certify their technology and tries to obtain the best cost efficiency for enterprises.

Red Hat has thousand of official distributors around the world and support companies like Amazon.com, Credit Suisse First Boston, DreamWorks, Reuters, etc... It also contributes with its engineers to many open source projects.

To see the differences between editions you can see their web[1].

[1] Red Hat web: http://www.redhat.com

### 1.2.2. Suse

OpenSUSE 10.2 is another possible distribution. The main feature that differs from others is an easy install process on Linux operating system and a complete software package by default that lets that lets common users do all daily tasks in their PC like browse the Web, send e-mail, chat with friends, organize digital photos, play movies and songs, create documents, etc...It also includes all kind applications for network services and applications development. More complete information can be extracted from [4].

This distribution intends to install easily all this software (which include from well-known applications like Firefox or Openoffice to new technologies like Beagle for desktop search).

For these reasons OpenSUSE it is not created specifically for business enterprises, although can be used for this purpose.

### 1.2.3. Fedora

Fedora is another distribution by Red Hat and guided by the Fedora Project Board. It has all the main features provided by a Linux distribution, but its main difference between others is its philosophy.

Fedora has a large community behind that has convert this distribution as the most important when talking about Linux security initiatives (to see exactly these kind of initiatives, you can see [5]) that after become in features implemented in Linux.

### 1.2.4. Mandrake

The purpose of Mandriva Linux (known as Mandrake Linux) distribution was created a Linux distribution easy to use for everyone.

With this initiative, Mandriva offers the best of Linux to any user in an easy-to-use environment. In this way Mandriva can be as easy as a computer which uses Windows or Mac OS. It has an own mechanism called FOSS (Free/Open Source Free Software licensing) that makes possible to collect the best ideas from everybody and make better solutions. It also has interesting features like a little maintenance system that avoids conflicts between applications. See [6] for more additional information.

### 1.2.5. Ubuntu

Before explaining Ubuntu, it is necessary to say that our tutor recommended and provided us this distribution to use it in the project for the characteristics that are explained in the following text, for that reason Ubuntu was selected to use it.

Ubuntu is a complete Linux operating system, freely available with community and professional support and it is developed by a large community. It is suitable for desktop and server stations and supports many architectures [7].

Ubuntu has more than 16,000 applications that can be downloaded, but the default distribution contained on a single CD covers every standard desktop application.

#### *1.2.5.1. Why Ubuntu 6.06 LTS*

LTS means Long Term Support and its purpose is to have a main distribution where everybody can work and develop it to include security updates for the next five years without any subscription. It also includes telephone and online support. That makes this release really interesting, although new versions have been released (Actually 6.10).

Ubuntu 6.06 LTS [7] also has a mechanism to make commercial software available, enabling businesses and individuals to download select software from Independent Software Vendors (ISVs).

Ubuntu is part of the Debian family of distributions. This makes possible to have instantly many software available, but it is also a good choice for us because the Linux-Box is based on a Debian OS and it is easier after to work directly with the Linux-Box and the commands necessary to operate with it.

## 1.3. Environments

As the multiple distribution options, there are also many environments, but the most famous and used are Gnome and KDE. It is introduced as new knowledge an alternative the 3D environments.

### 1.3.1. Gnome

This is the desktop used in Ubuntu. Gnome is used to implement the C application explained later because it supports many programming languages as C, C++, Java, C#, etc…

This project provides two things:

- A desktop environment that makes it an intuitive and attractive desktop.
- A development platform that allows applications to integrate into the rest of the desktop.

GNOME is part of the GNU project with all that a GNU project involves and every six months is released a new version. It is supported by many strong companies as and many industries used it by default. To see some of these go to [8]. Also governments have chosen gnome for their desktops, so its availability and support is assured.

### 1.3.2. KDE

KDE appears as the necessity of easier desktop environment for UNIX workstations, and make it similar to MacOS or Windows.

Of course, KDE is completely free and an open platform available to anyone to modify free of charge including its source code.

KDE Project has developed an application framework, implementing the latest advances in framework technology in order to be a real opponent to the most popular development frameworks as for example Microsoft's MFC/COM/ActiveX technology. This technology makes possible to developers create quickly new and high quality applications [9].

### 1.3.3. Others (3D Environments)

Last years a new environment conception has born based on OpenGL programs to convert virtual desktops in a 3 dimensional on Linux. It is not the purpose to explain how it works this mechanism, only mention that it is still a technology in development and very new yet. It needs more efforts but has many possibilities to improve it. These kinds of environments have different visualization modes and zoom properties. The images above show a possible visualization of these environments.

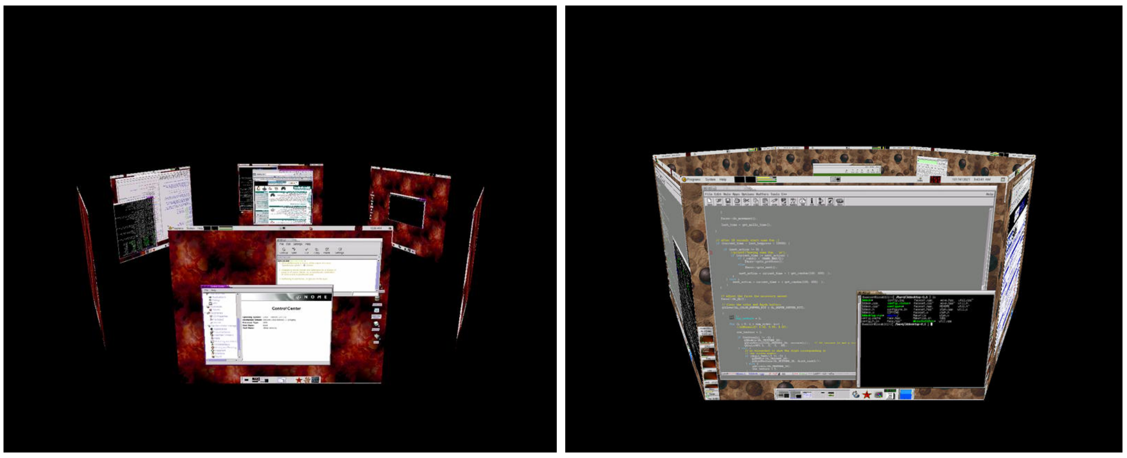


**Figure 1.1: Example view of 3D-Desktop[2]**

[2] Extracted from: http://desk3d.sourceforge.net/screenshots.php

# 2. Multimedia Network & Video Protocols

This section explains protocols and theoretical researches used in all the work. It starts with a brief section for the third network protocol level (IP) and continues in the upper levels until arrives to the video encapsulation protocols and the used software.

Assuming the Physical Layer (1), Link Layer (2) and the Network layer (3), the expected implementation in the Linux-Box will be: IP over Ethernet on standard UTP wire. These layers will not be reconsidered, but in the section "2.3 Headers and efficiency" calculations of efficiency will be made from Ethernet to the application layer. References to Wi-Fi will not be made in any case, although it could be a future implementation.

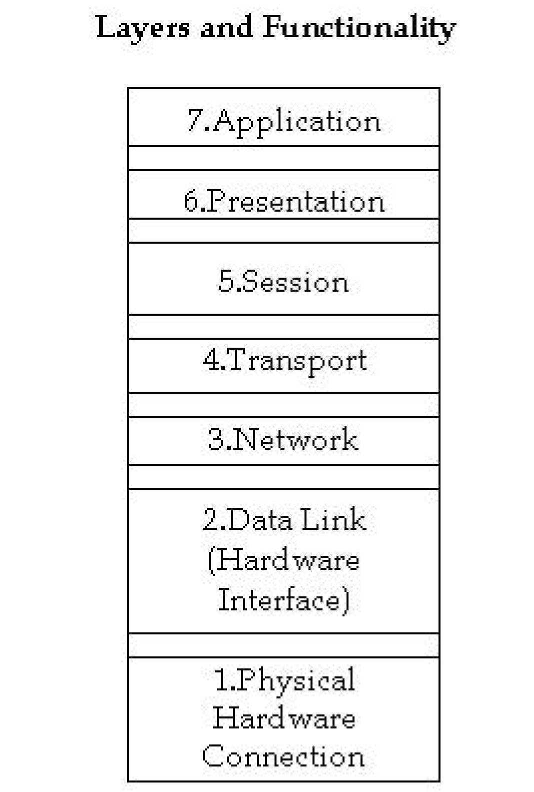


**Figure 2.1: ISO OSI model**[3]

This section does not want to explain explicitly what is IP, TCP or UDP. These protocols are here only with the purpose of comparing them, to explain "when" and "how they are used" and question why they are used. There are some flags usually used in streaming that can help to optimize the streams.

Later is explained the RTP protocol and their dependent protocols RTCP, RTSP that will be used by the Linux-Box and the VLC to encapsulate the audio and video information. There is also an important protocol specialized in the global net environment that implements QoS network functions. This protocol is RSVP

[3] Extracted from: http://www.tutorialsweb.com/networking/tcp-ip/ip.htm

and is explained a bit with its more important features in the annex, because it is not a protocol used in the work.

Later the “announcement and description protocols “are explained (also used in streaming), especially in applications that use broadcast, multicast or videoconference systems where diverse users are implicated.

Section 2.7.1 talks about Unicast, Multicast, and IGMP like protocols related with the network streaming theory that are used by the Linux-Box. In section 2.8 are explained different mechanisms to encapsulate video and audio frame codecs. Overalls are mentioned those more important and those used in the analysis.

The section “2.10. Protocol Analyzer” makes mention of software used in the Linux-Box global project for their analysis and recollect flow and packets information. At the end is made a comparative summary between these protocols.

## 2.1. IP

Internet Protocol (IP) is the main network-level (Layer 3 of the OSI model) communications protocol. Each IP packet has header part (40Bytes) and a payload.

This is the IPv4 Header:

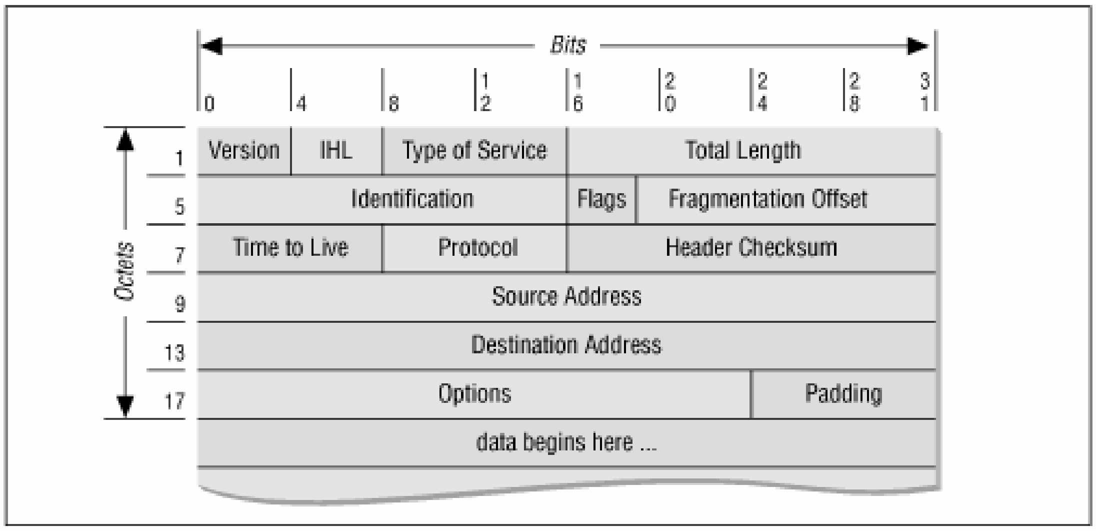


**Figure 2.2: IPv4 Header**[4]

The "Don't Frag." FLAG is essential when using Real-Time applications .Their use avoids to break packets into fragments in diverse routers by those pass through.

[4] Extracted from: http://www.unix.org.ua/orelly/networking/puis/ch16_02.htm

Apart of activating this FLAG, it is important to discover that the MTU of all the nets is for those they pass through. An optimal MTU helps to an effective network resources use and won't create unnecessary delays by not fragmenting packets.

Nowadays is used the IPv4 version, but is rising the concept to use IPv6. All this work is based on IPv4 and IPv6 is mentioned on future lines. The advantages from IPv6 to IPv4 are basically these[5]:

- **Expanded Addressing Capabilities:** IPv6 increases the IP address size from 32 bits to 128 bits, to support more addressing levels hierarchy, much greater number of addressable nodes, and simpler auto-addresses configuration. The scalability of multicast routing is improved by adding a "scope" field to multicast addresses. And a new address type called "anycast address" is defined, used to send a packet to any node of a nodes' group.

- **Header Format Simplification**: Some IPv4 header fields have been deleted or made optional, to reduce the common-case processing cost of packet handling and to limit the bandwidth cost of the IPv6 header.

- **Improved Support for Extensions and Options:** Changes in the way IP header options are encoded allows more efficient forwarding, less stringent limits on the length of options, and greater flexibility for introducing new options in the future.

- **Flow Labelling Capability:** A new capability is added to enable the labelling packets to belong to particular traffic "flows" that can be the sender requests special handling, such as non-default quality of service or "real-time" service.

- **Authentication and Privacy Capabilities**: Extensions to support authentication, data integrity, and (optional) data confidentiality are specified for IPv6.

There are three main kinds of disadvantages with IP on Real-Time proposes:
Variable network latency (Jitter), the packets arrive at their destination in a different order from transmission and may be damaged or lost.

These disadvantages cannot be fixed by IP itself but can be corrected by the higher-layer protocols. Generally TCP provides good transport for general-purpose data but is not suitable for streaming applications. The reason of why UDP is used as standard in this type of applications is explained in the next section.

[5] Advantages extracted from the RFC 2460: “Internet Protocol, Version 6 (IPv6), Specification”

## 2.2. TCP vs. UDP

The TCP header is explained in the figure below. It is important to pay attention on the Checksum. The checksum is calculated on a pseudo-header including all fields. This is because the IP checksum field performs the operations only over the headers. In this way we can assure the integrity the whole of the packet.

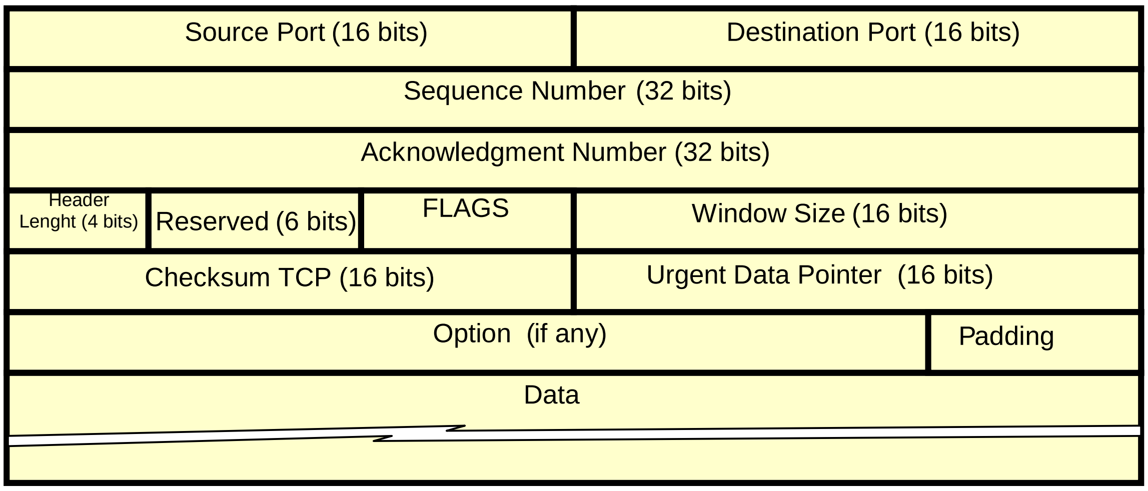


**Figure 2.3: TCP Header**

The UDP Header is smaller than the TCP one:

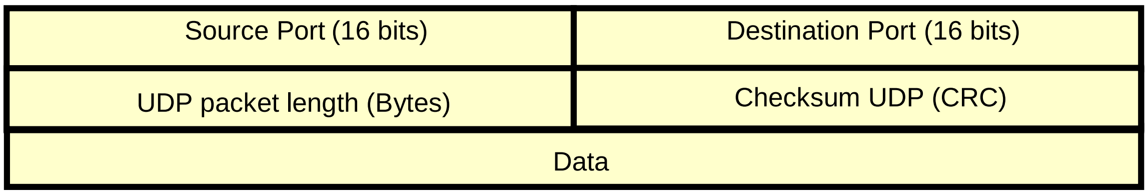


**Figure 2.4: UDP Header**

The checksum UDP is optional. If it is not used its value is ‘0’. For its calculation it is used a pseudo-header including all fields. This, as in TCP, is because the IP checksum field performs the operations only over the headers. In this way we can assure the integrity of the whole packet.

TCP and UDP are two transport-layer (Layer 4) protocols. TCP and UDP are also different between them. We can differentiate them in the way they are used, and how they work inside.

In audio and video streaming aspect is sensed the use of a light Transport Protocol as is UDP. It uses less heavy headers that TCP. This helps to take the best profit of the available Bandwidth to prioritize the useful information and a packet orientation (connectionless). Most of multimedia and streaming applications in the world are based on UDP by these reasons and others that are explained below, too.

TCP transmits a sequence of bytes and informs to the destination the next byte expected to receive. If a byte is not acknowledged in a specified time period, it is retransmitted by the source. This feature allows devices to detect and identify lost packets and request a retransmission. The repeated transmission adds latency to the system, and as we can see below, this is an important problem in general streaming data transmission.

In audio and video, the final user requires a continuous stream in real-time. Retransmission of packets adds delays and uses more bandwidth in the data channel than is needed. Furthermore the delayed information expires in these Real-Time applications. When there is a high transmission error the received buffer in the media player will be emptied and the stream will be interrupted. This is why the strategy for receiving streams is to ignore lost packets, and that is what UDP do. The packets can reduce the quality of the received stream, otherwise streaming media players are often designed to assume these errors.

On the other hand in an environment that does not pass through Internet[6] it could be important to reconsider the use of TCP that offers a connection-orientation that can optimize the results, and several Flags treated in the same section mentioned previously that can be useful when transmitting the same signal to various users. But after all it is possible to see clearly that TCP should not be used by the following main reasons:

- The window-opening mode that implements TCP makes the transmissions don't begin with the maximum possible BW. TCP retransmissions create delay that does not take profit in streaming, which needs constant flows in real-time. These retransfers also close the transmission window that has to grow again. When the BER is high as in Wi-Fi networks, the buffer can be emptied and the stream can be interrupted.

- The characteristic streaming strategy is to ignore the lost packets. This can affect to the subjective video quality but not to its linearity in time.

- Delivered TCP guaranty is carried out through persistent retransmission with a potentially increase wait time between consecutive retransmissions, giving rise to potentially higher delivery time.

- The “Additive Increase Multiplicative Decrease” rule gives rise to a widely varying instantaneous throughput profile in the form of a pattern that is not suitable for streaming media transport.

[6] Internet acts usually as a bottleneck in these kind of applications for its “best-effort” working mode.

| | TCP | UDP | Respect to Streaming |
|---|---|---|---|
| Header | 20 Byte | 8 Bytes | Better UDP with less overhead |
| Connection | Connection Oriented: a connection is set up prior to data transfer | Connectionless: No connection needs to be established | For Multicast is unsuitable a connection oriented communication |
| Reliability | Reliable (ACK) | Unreliable | Reliability is not as important as time delivery. TCP adds delay for retransmission |
| Communication | Two-way, interactivity between server and client. | One way only. No interactivity | In UDP, RTCP implements the feedback |
| Errors | Error Correction FEC on the whole packet | Error Detection only on Header Checksum | UDP less processing time. TCP can correct in the destination host |
| Data flow | Controls data flow to manage the download rate | No flow control | UDP sends to the same data flow as is encoded the media. |
| Re-transmit | Repeat Request | No repeat request | Repeat request adds delay, and is useless for real-time applications |
| Delivery Rate | No predetermined. TCP will increase data rate until packet loss indicates congestion | Delivery rate match the encoded stream rate. | In UDP encoding at several rates must be necessary to adjust to different delivery channels and propagation conditions. Re-encoding it for different network BW costs previous processing and time |
| Client Buffer | Receive buffer overflow: if data arrives too fast, receiver sends messages to the server to slow down. | No local caching. Packets are processed by the media player as they arrive | Client Buffers create delay. |

**Table 2.1: Comparison between TCP and UDP**

As we can see UDP is better than TCP in streaming applications and that enjoys also a total implementation in the world. But, does TCP have some advantage to UDP?
There are a several important advantages using TCP:

- TCP rate control has empirically proven **stability** and **scalability**.
- TCP provides the **guaranteed delivery**, deleting the packet loss efficiently.
- TCP can be useful to **pass over the firewalls**. It can surprise their use in streaming today but it is used like a last resource.
- The **flow control** can be very appropriated, avoiding us the compression from our resources to different bitrates.
- The transmission windows system helps to **optimize the use** of network resources.

Making a modification to the TCP's headers can permit better rate-control and solvent some disadvantages that TCP have in front of UDP to transport streams, or for example to fix the logarithmic windows closing, necessary for mobile networks.

Even with this TCP advantages and/or modifications, **UDP** results obviously better for its use with Real-Time applications, and by extension in the Linux-Box, for the next reasons:

- Minimum overhead
- Sends at maximum data rate from the beginning of the transmission.
- No repeat requests->No retransmissions (The lost of a single packet is not important in a real-time application).
- Low processing time. No buffers

In the next sections we will see the efficiency and the upper protocols.

## 2.3. Headers and efficiency

An advantage of using Ethernet is that has a very low error rate. This is positive because UDP does not incorporate error correction. Let's see deeply the Ethernet Frames:

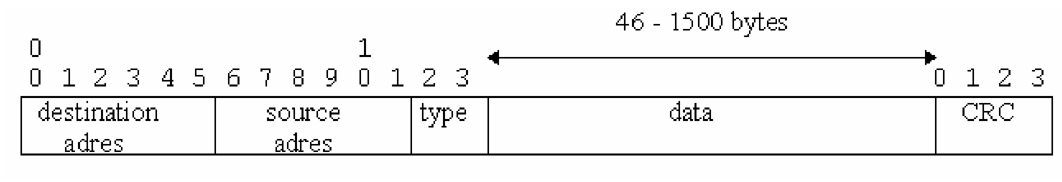


**Figure 2.5: Ethernet Header**[7]

[7] Extracted from: http://web.inter.nl.net/users/Rohiet.Seosahai/ethernet.htm

In this figure we can see that Ethernet implements FCS. The resulting header is of 18 Bytes (14 Bytes+4 Bytes of CRC).

Knowing that the minimum IP header has 20 Bytes and that UDP has 8 Bytes:

| | | | |
|---|---|---|---|
| Ethernet Header (18 bytes) | IP Header (20 byte) | UDP Header (8 byte) | Data |

**Figure 2.6: Packet encapsulation**

Consequently with TCP/IP we have a 28/40 = 40% more header Bytes than with UDP/IP.

The maximum value of "n" is 7. With this value, we obtain the maximum efficiency, because the packets are optimal. A higher value would exceed the MTU.

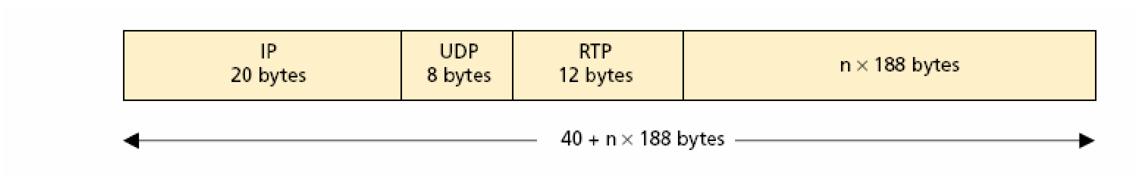


**Figure 2.7: Header RTP with MPEG2-TS encapsulated**

The data length depends on the Codec but in MPEG2-TS the transmission has 188 Bytes with 4 Bytes of payload: (184+4). The theoretical maximum efficiency that offers the studied final environment: (Ethernet/IP/UDP/RTP/MPEG2-TS) is 94%.

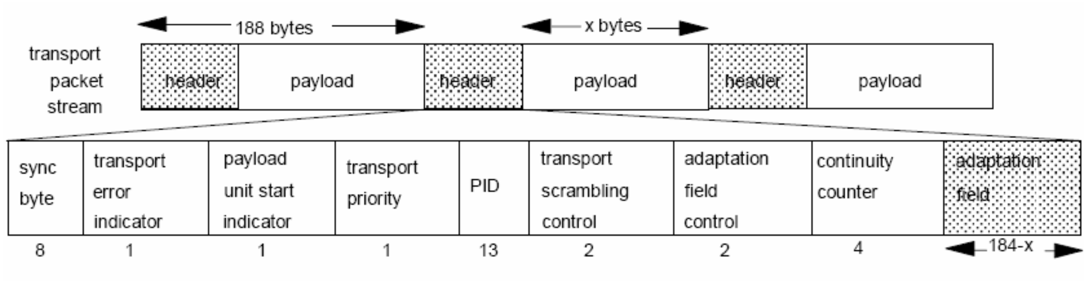


**Figure 2.8: Format of the 4 Bytes payload of every MPEG2-TS**

| | Prot | Suma | Total | N=1 | N=2 | N=3 | N=4 | N=5 | N=6 | N=7 |
|---|---|---|---|---|---|---|---|---|---|---|
| **From IP** | **TCP** | **20 IP + 20 TCP** | **40 B** | 184/ (188+40) =80’70% | 88’46% | 91’39 % | 92’92 % | 93’88 % | 94’52 % | (184x7) / (188x7+40)= **94’98%** |
| | **UDP** | **20 IP + 8 UDP** | **28 B** | 85’18% | 91’09% | 93’24 % | 94’36 % | 95’04 % | 95’50 % | 95’83% |
| **With Eth.** | **TCP** | **18 Eth +20 IP +20 TCP** | **58 B** | 74’79% | 84’79% | 88’74 % | 90’86 % | 92’18 % | 93’09 % | 93’74% |
| | **UDP** | **18 Eth + 20 IP + 8 UDP** | **46 B** | 78’63% | 87’20% | 90’49 % | 92’23 % | 93’31 % | 94’04 % | 94’57% |
| **Eth + RTP** | **TCP** | **18 Eth + 20 IP+ 20 TCP+ 12 RTP** | **70 B** | 71’31% | 82’51% | 87’07 % | 89’54 % | 91’09 % | 92’15 % | 92’93% |
| | **UDP** | **18 Eth + 20 IP + 8 UDP+ 12 RTP** | **58 B** | 184/(188+ 58)= 74’79% | 84’79% | 88’74 % | 90’86 % | 92’18 % | 93’09 % | (184x7)/ (188x7+58)= **93’74%** |

**Table 2.2: Max Theoretically Efficiency in TCP/UDP RTP**

As more accurately the diverse protocols are analyzed and take part in a communication, is observed that the efficiency goes decreasing.

## 2.4. RTP (Real-time Transport Protocol)

### 2.4.1. What's RTP?

RTP is a transport protocol developed specifically for streaming data across IP networks and is the most important streaming standard. All media streams are encapsulated in RTP packets. It is usually encapsulated on UDP/IP as showed in Figure 2.7 and is compatible with other protocols as ATM or IPv6. In section 2.2 is explained why is often encapsulated on UDP and not in TCP. It was primarily designed for multicast real-time data, but it can be also used in Unicast (for example a one-way transport such as video-on-demand).

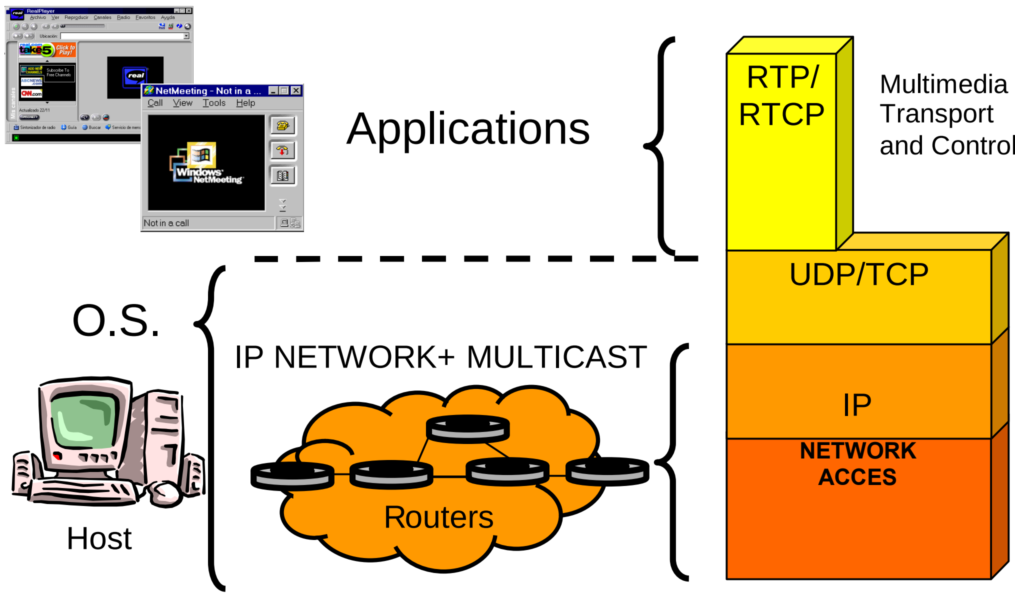


**Figure 2.9: Location of RTP/RTCP in the network**[8]

### 2.4.2. Characteristics

RTP provides information required for multimedia applications to a correct transmission like timestamp, sequence numbering, security, content identification and other mechanisms. These services must be implemented at Application level and it is not a RTP responsibility. So RTP only helps lower layers to have control over resources and add reliability, flow/congestion control and other mechanisms for carry real-time information.

[8] Figure modified and extracted from Systems and Applications notes (an EPSC subject).

RTP transmits packets in real-time over the network, this means that lost or damaged packets are not retransmitted. Client software should solve this problem. Another problem is if the connection speed is lower than the media data rate, the transmission plays poorly or it doesn’t play.

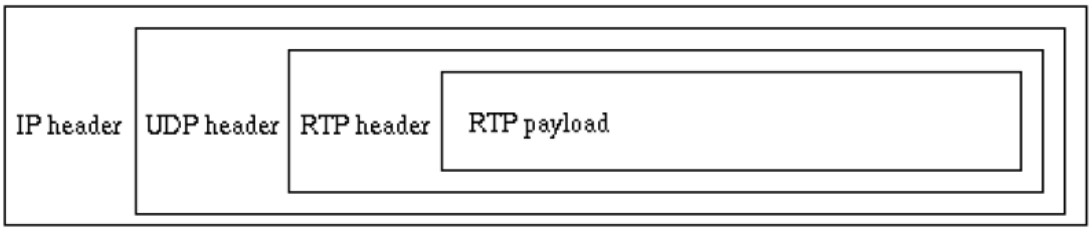


**Figure 2.10: RTP data in an IP packet**[9]

### 2.4.3. Header Format

Formerly the RTP packet has the following format as showed in figure 2.11:

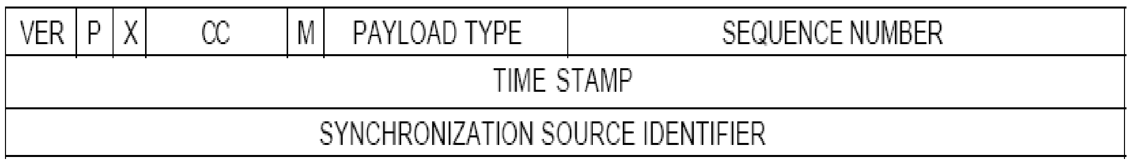


**Figure 2.11: RTP format header**[10]

- **Version (V):** 2 bits. Version of RTP.
- **Padding (P): 1 bit**. If set, the packet can contain additional padding bytes at the end. The last byte contains a count of how many should be ignored.
- **Extension (X): 1 bit**. If set, the header is followed by one extension.
- **CSRC count (CC): 4 bits**. The number of CSRC identifiers.
- **Marker (M): 1 bit**. Allow significant events as frame limits to be marked in the packet stream.
- **Payload type (PT): 7 bits**. Identifies the format of the RTP data.
- **Sequence number: 16 bits**. Increments by one for each RTP data packet sent. Used to detect packet loss and restore packet sequence.

[9] Extracted from: http://www.cs.wustl.edu/~jain/cis788-97/ftp/ip_multimedia/index.htm
[10] Extracted from: “htp://www.ieee802.org/16/tge/contrib/C80216e-04_523r1.pdf”

- **Timestamp: 32 bits**. The instant of the first octet send in the RTP data packet. Used for synchronization and jitter calculations.
- **SSRC: 32 bits**. Used to differentiate sources within the same RTP session.
- **CSRC list: 0 to 15 items, 32 bits each**. Contributing sources for the data contained in this packet.

### 2.4.4. How does it work?

Timestamp is the most important information field for real-time applications. It is incremented when the first byte in the packet is transmitted and increased each time a packet is sent. The receiver uses it to reconstruct the original timing to be able to play out the data correctly rate. Timestamp is also used to synchronize different streams with different timing properties, such as audio and video data in MPEG.

Another important field is the sequence number. UDP does not deliver packets in timely order, so it’s necessary the sequence number to place the incoming data packets in the correct order and for packet loss detection. In addition, for example, with some video format, when a frame is split into several RTP packets, all of them can have the same timestamp.

To specify the payload type and the encoding/compression format is necessary the payload identifier. But an RTP sender can only send one type of payload each transmission, although the payload type may change during transmission, for example, to adjust the network congestion.

It is also an important to consider the MTU’s network to not exceed it. It is not recommended because when a RTP packets exceed the MTU, the router split it in a process called fragmentation, and if one of them is lost, all the remaining will be lost too. Fragmentation puts additional charge on resources also, so always if it is possible the field DF on the IP header must be “1”. The MTU depends on the network type, for example in Ethernet is 1500 bytes.

## 2.5. RTSP (Real Time Streaming Protocol)

### 2.5.1. What is RTSP?

RTSP is an application-level protocol to control the delivery audio and video data with real-time properties. This protocol tries to control multiple data delivery sessions, such as UDP, multicast UDP and TCP, with mechanisms based on RTP.

This means that RTSP acts as a "network remote control" for multimedia servers. The protocol supports the following operations:

- Recovery media from media Server.
- Request of a media server to a conference.
- Add of media to an existing presentation.

## 2.5.2. Characteristics

RTP has the following properties:

- **Extensibility:** Add new methods and parameters is easy.
- **Easy grammar**: Can be recognised by standard HTTP or MIME.
- **Transport-independent**: Can use UDP or TCP.
- **Multi-server capable**: Can control sessions in different servers.
- **Control of recording devices**: Can control recording and playback devices.
- **Separation of stream control and conference initiation.**
- **Presentation description neutral**: The protocol does not impose a particular presentation description or metafile format.
- **Proxy and firewall friendly**: The protocol should be supported by firewalls that understand the setup method.
- **HTTP-like**: RTSP reuses HTTP concepts but with some differences.
- **Appropriate server control**: If a client can start a stream, it must be able to stop it.
- **Capability negotiation:** Client must determine which methods are not going to be implemented.

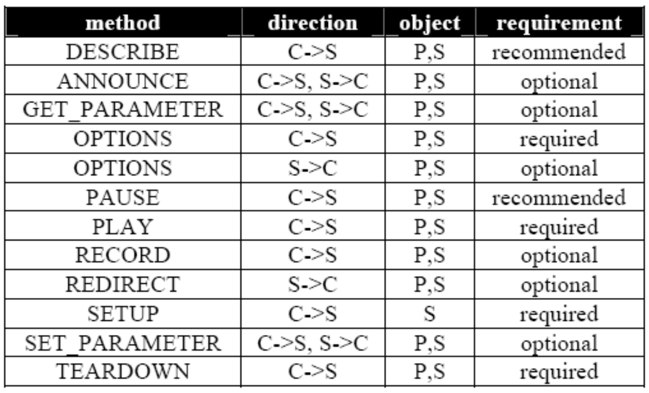
## 2.5.3. Format

The RTSP message formats has a similar syntax to HTTP messages. The general syntax for an RTSP method is:

| |
|---|
| start-line |
| message-header |
| ... |
| message-header |
| CRLF |
| [ message-body ] |

**Figure 2.12: RTSP message format**

These are the RTSP methods:



| method | direction | object | requirement |
|---|---|---|---|
| DESCRIBE | C->S | P,S | recommended |
| ANNOUNCE | C->S, S->C | P,S | optional |
| GET_PARAMETER | C->S, S->C | P,S | optional |
| OPTIONS | C->S | P,S | required |
| OPTIONS | S->C | P,S | optional |
| PAUSE | C->S | P,S | recommended |
| PLAY | C->S | P,S | required |
| RECORD | C->S | P,S | optional |
| REDIRECT | S->C | P,S | optional |
| SETUP | C->S | S | required |
| SET_PARAMETER | C->S, S->C | P,S | optional |
| TEARDOWN | C->S | P,S | required |

**Figure 2.13: RTSP methods**[11]

RTSP offers a VCR-like control to the user: Setup, Play, Stop, Pause, FF and REW, and also random access to any part of the media clip.

It also helps the server to adjust the media bandwidth to the network congestion in order to suit the available capacity. Another important function of RTSP is its ability to choose the optimum delivery channel to the client. For example, if UDP cannot be used (some corporate firewalls will not pass UDP), the streaming server has to offer a choice of delivery protocols – multicast UDP or TCP to suit different clients.

[11] Extracted from: http://www.cs.helsinki.fi/u/jmanner/Courses/seminar_papers/rtsp.pdf

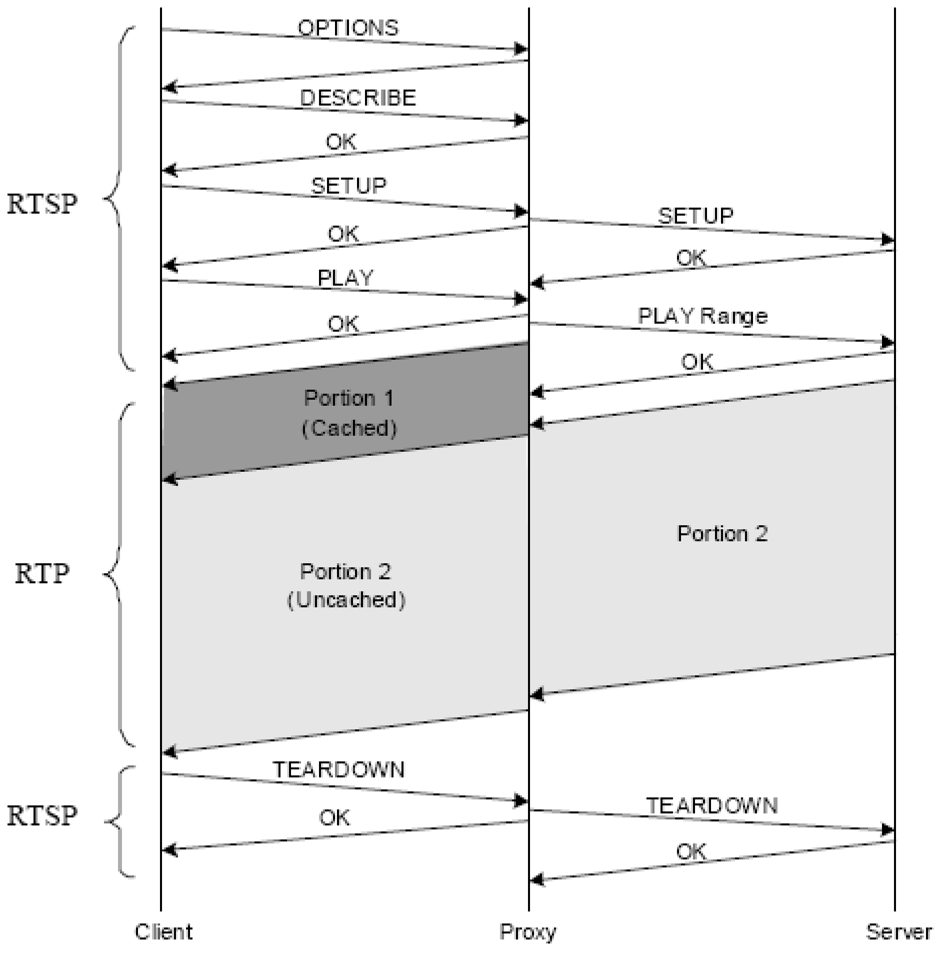


**Figure 2.14: Dialogue between Server and client in a RTSP/RTP communication.**

## 2.6. Announcement and Session Description Protocols

We will study two types of announcement or description protocols. Those two protocols are related, because usually one travels inside the other.

### 2.6.1. SAP

#### *2.6.1.1. What is SAP?*

SAP (Session Announcement Protocol) was created by the necessity to show advertisements of multicast multimedia conferences and other multicast sessions, and communicate the relevant session setup information to its participants.

### *2.6.1.2. Packet Format*

SAP data packets have the format described in figure 2.15.

| 3 | 4 | 5 | 6 | 7 | 8 | 16 | 24 | 32bit |
|---|---|---|---|---|---|---|---|---|
| V | A | R | T | E | C | Auth len | Msg ID hash | |
| Originating source (32 or 128 bits) | | | | | | | | |
| Optional Authentication Data | | | | | | | | |
| Optional timeout | | | | | | | | |
| Optional payload type | | | | | | | | |
| | | | | | | | 0 | |
| Payload | | | | | | | | |

**Figure 2.15: Sap Packet format.** [12]

- **Version Number (V):** The field MUST be set to 1 (SAPv2).
- **Address type (A):** 0 for t IPv4 address and 1 for IPv6 address.
- **Reserved (R):** 0 for SAP announcers and ignored by SAP listeners.
- **Message Type (T):** 0 for a session announcement packet and 1 for a session deletion packet.
- **Encryption Bit (E):** If it's set to 1, the data is encrypted.
- **Compressed Bit (C):** If it's set to 1, the data is compressed using a compression algorithm.
- **Authentication Length**: The header that contain authentication data.
- **Authentication data:** Contains a digital signature with the length marked on the authentication length.
- **Message Identifier Hash:** Provides a unique to identify the precise version of this announcement.

SAP advertisements are received by all the participants and also other session directories, so new participants can use the session description to start the tools required to participate in the session.

The SAP announces periodically an announcement packet by multicast to a well known multicast address and port. Then a SAP possible receiver listens on the well known SAP address and port. That announcement contains a session description and should contain an authentication header.

[12] Extracted from the web: "http://www.protocolbase.net/protocols/protocol_SAP.php"

SAP announcements MUST be sent on port 9875 and should be sent with an IP time-to-live of 255. The bandwidth limit used for SAP announcements is 4 kbps.

## 2.6.2. SDP

### *2.6.2.1. What is SDP?*

SDP (Session Description Protocol) was created in multimedia sessions to transmit information about media streams: session announcement, session invitation, and other forms of multimedia session initiation.

SDP is designed to communicate the conference addresses and information necessary for participation in a multimedia conference and transmit this information to recipients.

### *2.6.2.2. Characteristics*

SDP can be used for many network types and applications but does not incorporate a transport protocol, so it adapts its format to the appropriate protocol SAP, SIP, RTSP, electronic mail using the MIME Extensions or HTTP.

The text payload in an SDP session description should be no greater than 1 Kbyte in length.

An SDP packet includes:

- Session name and purpose
- Time(s) the session is active
- The media comprising the session
- Information to receive those media (addresses, ports, formats and so on)
- The type of media (video, audio, etc)
- The transport protocol (RTP/UDP/IP, H.320, etc)
- The format of the media (H.261 video, MPEG video, etc)

For an IP multicast session, the following are also transmitted:

- Multicast address for media
- Transport Port for media

Where both address and port are the destinations of the multicast stream.

In case of an IP Unicast session, the following are transmitted:

- Remote address for media
- Transport port for contact address

If resources necessary to participate in a session are limited, some additional information should be interesting to transmit:

- Information about the bandwidth to be used by the conference
- Contact information of the responsible person for the session

## 2.7. Other Network Protocols Used In Streaming

### 2.7.1. IP Multicast

**Unicast transmission** means sending one stream to each receiver. Unicast does not represent a particularly efficient use of bandwidth but it allows the users, by using RTSP functionalities, to watch different parts of the media or watch different movies at the same time. Viewers normally open a Unicast media by opening an RTSP URL.

**Broadcast transmission** means sending one copy of the stream over the whole network. A single stream is sent to all hosts in the network. It is possible to do it in small LANs if support *broadcast* but the Internet does not allow it. Broadcast does not allow viewers to control the streams (there is no VCR functionalities) so there is no feedback from the user to the server.

**Multicast transmission** means sending exactly one copy of the stream, not over the whole network (as in *broadcast*), only down the branches of the network where one or more viewers are available or are trying to connect to the server. In this case the available network bandwidth can be used more efficiently. Multicast requires fairly sophisticated router software that allows the server to replicate streams required by the clients. The user of a multicast has no control over the media presented as in *broadcast* (the choice is simply to watch or not to watch). The user's system communicates with the nearest router (rather than directly with the server) to get a copy of the stream. To find information on multicast programmes users must download the Session Description Protocol (SDP) file, usually available on service provider's web page.

For VoD and AoD the Linux-Box uses Unicast. For TV on demand it uses multicast to a specific IP. This is explained in section 4.2.3.
The IP range dedicated to multicast goes from 224.0.0.0 to 239.255.255.255.
There are some special multicast IPv4 addresses:

- Address 224.0.0.1 identifies every host from a subnet. Any host with multicast capabilities within a subnet must join to this group.

- Address is 224.0.0.2 used for identification towards every multicast capable router in a network.
- The address range 224.0.0.0 - 224.0.0.255 is allocated for low level protocols. Datagrams sent to addresses within this range, will never be routed by multicast capable routers.
- The address range 239.0.0.0 - 239.255.255.255 is allocated for administrative purposes. The addresses are locally assigned within each organization, but they could not exist out of the organization. The organization routers should not route any of these addresses out of the corporate network.

| Scope | TTL | Adress range | Description |
|---|---|---|---|
| Node | 0 | | The datagram is restricted to the local host. It will not reach any of the network interfaces. |
| Link | 1 | 224.0.0.0 - 224.0.0.255 | The datagram will be restricted to the sender host subnet, and will not progress beyond any router. |
| Department | < 32 | 239.255.0.0 - 239.255.255.255 | Restricted to one department of the organization. |
| Organization | < 64 | 239.192.0.0 - 239.195.255.255 | Restricted for a specific organization. |
| Global | < 255 | 224.0.1.0 - 238.255.255.255 | No restriction, global application. |

**Table 2.3: Multicast IP directions scope**[13]

### *2.7.1.1. IGMP protocol*

*What is IGMP?*

The Internet Group Management Protocol (IGMP) is used by IP hosts to report their multicast group memberships to the nearest multicast routers. There are three IGMP versions:

- IGMPv1 only has two message types:
  - o Membership Query.
  - o Membership Report.
- IGMPv2 that implements a new message type to allow the hosts to say the router that want to leave the multicast session.
- IGMPv3 give to the multicast routers the capacity to discriminate between different fonts that send traffic to a determined multicast group. To make this, routers must integrate a more intelligent API, so this IGMP version is used in environments where multicast use is extended.

[13] Extracetd form http://www.linuxfocus.org

In our work, as in the department, is used the IGMPv2 for itsr easy implementation and the fact that is useful closing the multicast sessions when no users are listening.

IGMP is an integral part of IP so its messages are encapsulated in IP datagrams and requires to be implemented by all hosts that want to receive IP multicasts.

| 8 | | | 16 | 32 bits |
|---|---|---|---|---|
| Type | | | Max Resp Code | Checksum |
| Resv | S | QRV | QQIC | Number of Sources |
| Group address | | | | |
| Source Address | | | | |

**Figure 2.16: SDP packet format**[14]

- **Type:** There are four types: Membership Query, Membership Report (version 1 o 2) and Leave Group.
- **Max Response Time**: The maximum allowed time before sending a responding report.
- **Checksum:** Is the sum of the whole IGMP message (the entire IP payload).
- **Group Address:** Specifies the group address demanded when sending a Group-Specific or and IP multicast group address of the group being reported or left.

The working mode is simple:

1. Incorporate a process of a new member in a multicast group.
2. The “Designed Router” sends Membership Query periodically (between 60 and 90 seconds).
3. Only is needed that a member of every “Multicast Group” responds to the Membership Query with a Membership Report.

Routers that are members of multicast groups are expected to act as hosts as well as routers, and may even respond to their own demands. So it is correct to say that IGMP can also be used between routers.

### 2.7.2. Protocols comparative

The purpose in this section is to make a summary of the real-time protocols that are used for the delivery and multimedia data control explained previously and to compare them to see with what advantages can contribute each one of them in the environment for which they have been developed.

[14] Extracted from: http://www.protocolbase.net/protocols/protocol_IGMP.php

This comparative don’t enter in the internet protocols that can be used as TCP, UDP and IP since these have been debated at the beginning of this part in the section "TCP vs. UDP" where we saw that the advantages that contribute UDP respect TCP that make it more interesting for real-time applications. Is for this that the real-time protocols are usually encapsulated over UDP.

As UDP it is a simple protocol that does not implement many interesting mechanisms for Real-Time applications, emerges the needing of other upper protocols that can solve the lacks of UDP. With this objective were developed RTP, RTCP and RTSP.

It is also necessary to mention that some of these protocols are complementary between them and they need themselves for the correctly execution of the applications. Each one contributes in something that does not contribute the other one, like we explain below.

RTP is the only protocol that is really strictly necessary, although this statement can be subjective seeing the functionality of the other ones. RTP is very useful to divide the streams in packages that then can be reconstructed before. These streams can be audio or video in the client. Also in the fact that it gives information of the type of data that is being sent to the application, as well as the sequence numbers and the timestamp of the packets, this is vital information for the application to be able to reconstruct and to reorder the received information. The problem is that RTP does not provide feedback with the source even information about the flows that arrive or not to the client. Also it does not implement any mechanism to know the number of members of a session. For this reasons emerge the necessity of another protocol like RTCP that realise these functions. It is necessary to say that none of these protocols assures delivery or quality of service (QoS). They only provide information, so that the source according to their judgement can react to the different problems that can arise of the transmission. The purpose of these protocols is to replace the functions that the inferior layers of the protocols pile aren’t implemented.

To have a continuous feedback between the source and the receiver and that the receiver could control the information that receives, for example stopping, pausing, controls of volume etc... Either audio or video, it will be necessary the transmission of packets over RTSP.
On the other hand, the announcement mechanisms and session description could be necessary in real-time applications, to announce from an automatic way to the users of the availability of a session as well as the description of it and they could be connected to the session if they have the permission or it interests them. Exist other ways more static as for example to enunciate the URL's in a Web page or similar. The advantage of these announcement mechanisms is that they are dynamic in the aspect of the "server" and that they directly describe the announced mechanism.

| | **STREAMING** | **DOWNLOAD/ (PROGRESSIVE)** |
|---|---|---|
| **Server** | Streaming server is required | Standard web server is sufficient |
| **Network layer protocol Used** | UDP/IP | TCP/IP |
| **Application layer protocol Used** | RTP/RTSP | HTTP |
| **Packet loss** | Packet loss acceptable | No packet loss |
| **Time performance** | Real time. The delivered media duration is the same as original | Packets may be retransmitted, leading to slower delivery times |
| **Delivery quality** | Some packets may be discarded, to meet time and/or bandwidth constraints | High-quality delivery guaranteed, no data is lost or discarded |
| **User connection** | Can match the user's bandwidth | File is delivered without regard to the user's Bandwidth |
| **Playback** | File starts playing immediately | Playback begins when all of (in progressive: enough of) the file has been downloaded |
| **Effort** | More burden on service provider(requires server, multiple bit-rate versions and formats) | More burden on the end user (hard drive space, connection speed) |
| **Firewalls** | May not play behind some firewalls | Bypasses most firewalls |
| **Storage** | No files are downloaded to the user's PC | Files are downloaded to the user's PC |
| **VCR functionalities** | Yes (for streaming of pre-recorded material) | No |
| **Zapping of internet radio Channels** | Smooth | Not possible |

**Table 2.4: Streaming server with RTSP vs. HTTP**

This table explains that although http is suitable for transfer web pages, it's not the best option for streaming because is based on TCP, which force reliability regard to timeliness, and this is not suited for use in multimedia presentations with timelines. Also Http doesn't have good mechanisms to access the file as RTSP.

Another streaming alternative very used exist called MMS protocol created for Microsoft to transfer Unicast data over UDP or TCP. By default Windows media will select over UDP, but if it is not possible it will try over TCP or a modified version of HTTP. The problem of this protocol is that only Windows Media has the rights.

## 2.8. Video Codecs and Encapsulation Protocols

### 2.8.1. MPEG2-TS

The MPEG2 signal is very complex, and depending on what part of the MPEG standard we are interested in, the requirements will be different: elementary stream (ES), packetized elementary stream (PES), transport stream (TS), program stream (PS) and program-specific information (PSI).

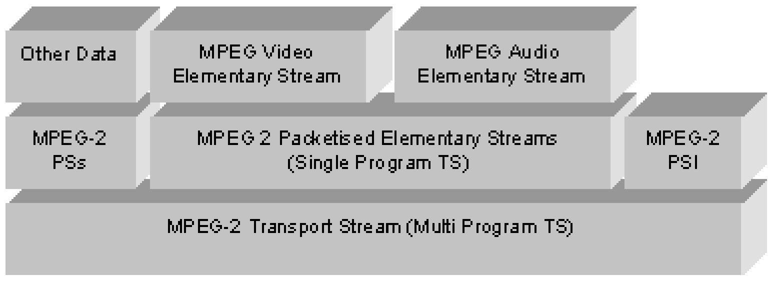

Other Data
MPEG Video Elementary Stream
MPEG Audio Elementary Stream
MPEG-2 PSs
MPEG 2 Packetised Elementary Streams (Single Program TS)
MPEG-2 PSI
MPEG-2 Transport Stream (Multi Program TS)

**Figure 2.17: Streams supported by the MPTS**[15]

To understand how the component parts of the bit stream are multiplexed, first is necessary to first look at each component part. The most basic component is known as an Elementary Stream in MPEG. Any program contains a combination of elementary streams (one for video, one or more for audio, control data, subtitles, etc).

**Elementary Stream (ES):** Each ES output contains a single type of signal. For video and audio, the data is organised into access units that represents a fundamental unit of encoding. For example, in video, an access unit will usually be an encoded video frame.

**Packetized Elementary Stream (PES):** Each ES is input to a processor which accumulates the data into a stream of PES packets. A PES packet may be a fixed (or variable) sized block, till 65536 bytes per block and includes a 6 byte protocol header and usually contains an integral number of ES access units.

[15] All images of this section are extracted from: http://erg.abdn.ac.uk/research/future-net/digital-video/mpeg2-trans.html

The PES header starts with a 3 byte start code, followed by a one byte stream ID and a 2 byte length field.

**Transport Stream (TS):** Each PES packet is broken into more packets making a general purpose for one or more streams, and can have different references. This is recommended in environments with potential packet loss or corruption by noise, or where we need to send more than one program at a time. DVB uses the MPEG-2 TS over many variety networks.

**Program Stream (PS):** Consists of a group of PES packets that are based on the same reference. This is ideal in environments with few errors and it is easy to process for the software. For this reason PS is usually used in digital video storage devices.

**Program:** Transport stream has a concept of programs, which are groups of one or more PIDs that are related to each other. For instance, a transport stream used in digital television might contain three programs, to represent three television channels. Suppose each channel consists of one video stream, one or two audio streams, and any necessary metadata. A receiver wishing to tune to a particular "channel" has to decode the payload of the PIDs associated with its program. It can discard the contents of all other PIDs.

**Program Specific Information (PSI):** Signalling Tables are special streams that help users identify a program. The tables, called PSI, consist of a description of the elementary streams which need to be combined to build programs. Each PSI table is carried in a sequence of PSI *Sections* and is protected by a *CRC* (checksum).

> **PAT** stands for Program Association Table. The PAT lists the PIDs for the programs in the stream. In other words the PAT lists PIDs for all PMTs in the stream. Packets containing PAT information always have PID 0x0.
>
> **PMT**: Program Map Tables, contain information about programs. For each program, there is a PMT, with the PMT for each program appearing on its own PID. The PMTs describe which PIDs contain data relevant to the program. PMTs also provide metadata about the streams in their constituent PIDs. For example, if a program contains an MPEG-2 video stream, the PMT will list this PID, describe it as a video stream, and provide the type of video that it contains (in this case, MPEG-2). The PMT may also contain additional descriptors providing data about its constituent streams.
>
> **PCR:** To assist the decoder in presenting programs on time, at the right speed, and with synchronization, programs usually periodically provide a *Program Clock Reference*, or PCR, on one of the PIDs in the program.

This section is about the TS part because project is based on it. Every transport stream packet has 188 bytes. Each packet contains 184 bytes of data and a 4 bytes header. One of the items in this 4 bytes header is the 13 bit *Packet Identifier (PID)* and the others are for sync, as showed in this figure:

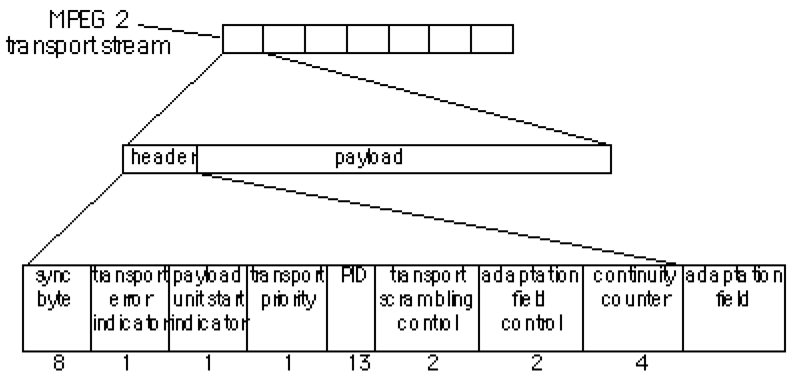


**Figure 2.18: TS Header Structure**

The PID is a unique address identifier. Every video and audio stream as well as each PSI table needs to have a unique PID. The next figure 2-18 shows two elementary streams are sent in the same MPEG-2 transport multiplex. Each packet has its particularly PID value in the header. The audio packets have PID 64, and the video packets PID 51. These values are arbitrary but must be different. Any packet has priority so they have not any correlation. Packets with different PID can be inserted into the TS at any time and if there are not available packets, null packets are inserted (with a PID value of 0x1FFF). The PES are not synchronised, so before this process is necessary another to synchronised the streams.

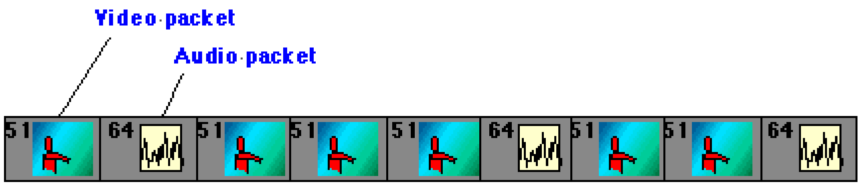


**Figure 2.19: Program Transport Stream (Audio and Video PES).**

Other fields in the TS header include:

- The header starts with a Synchronisation Byte (8 bits).
- The first flag indicates a transport error.
- The second flag indicates the start of a payload.
- The third flag indicates transport priority bit.
- The two scrambling control bits to encrypt the data of some TS packets.
- Two adaptation field control bits which may take four values:
  - 01 – No adaptation field, payload only.
  - 10 – Adaptation field only, no payload.

  - 11 – Adaptation field followed by payload.
  - 00 - RESERVED for future use.
- A half byte Continuity Counter (4 bits) which repeatedly increments zero through 15 for each PID; used to determine if packets are lost or repeated

TS is designed to be robust with short frames. So they need a strong correction mechanism (expects a BER better than $10^{-10}$). TS is a transport protocol but it not ensure us that the data will be transported without problems. This responsibility is for layers under in the OSI. TS needs the under layers to identify the transport packets, and to indicate in when a packet has been transmitted with an error.

The resultant TS output of a multiplexer may be either a single program TS (SPTS) or, more generally, a multiprogram TS (MPTS). A program consists of one or more ESs with the same time reference.

An SPTS contains all the information requires to reproduce the encoded TV channel or multimedia stream.

Sometimes one or more SPTS streams can combine to form a MPTS. It also contains all the necessary to control the transmission, so PID values may be changed cause the need for verifying accurate provisioning. This means also that PSI is required.

An example of TS could be on DVB-S. The DVB-S standard requires the 188 bytes transport packets to be protected by 16 bytes of Reed Solomon (RS) coding.

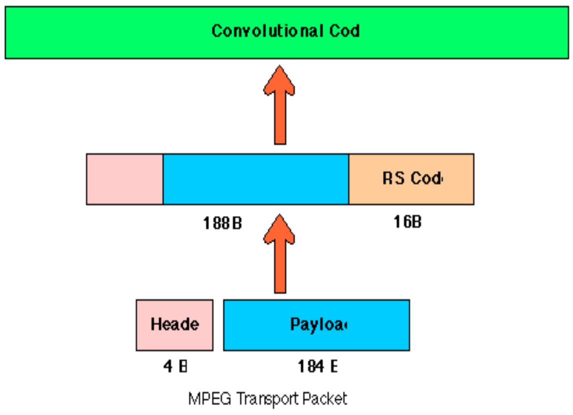


**Figure 2.20: MPEG Transport Service Encoding Specified by DVB-S.**

The coding level can be selected (from 1/2 to 7/8 depending on the intended application and available bandwidth) and the digital bit stream is then modulated using *Quadrature Phase Shift Keying (QPSK)*. A typical satellite channel has a 36 MHz bandwidth, which may support transmission of 35-40 Mbps.

To see the program that Users want, must determine the PID value to filter the correct packets.

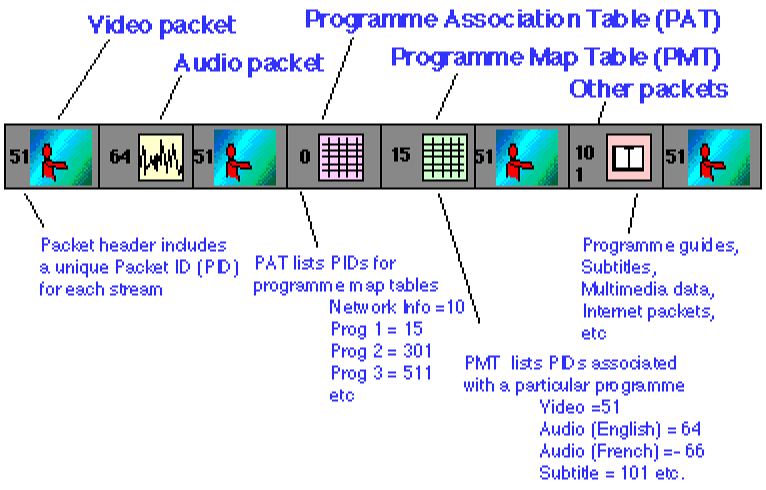

Video packet
Programme Association Table (PAT)
Audio packet
Programme Map Table (PMT)
Other packets
51
64
51
0
15
51
10 1
51
Packet header includes a unique Packet ID (PID) for each stream
PAT lists PIDs for programme map tables
Network Info = 10
Prog 1 = 15
Prog 2 = 301
Prog 3 = 511
etc
Programme guides, Subtitles, Multimedia data, Internet packets, etc
PMT lists PIDs associated with a particular programme
Video = 51
Audio (English) = 64
Audio (French) =- 66
Subtitle = 101 etc.

**Figure 2.21: DVB Signalling Tables and Transport Layer PIDs**

Depending on the program the streams can be synchronised or not. Optionally, to help synchronisation, time stamps can be sent. There are two types of time stamps:

- **Reference time stamp**: Indicates the current time.
- **Decoding Time Stamp (DTS):** Indicate the exact moment where a video or audio frame has to be decoded or presented to the user.

Finally, in the figure below is shown a possible structure of a TS system:

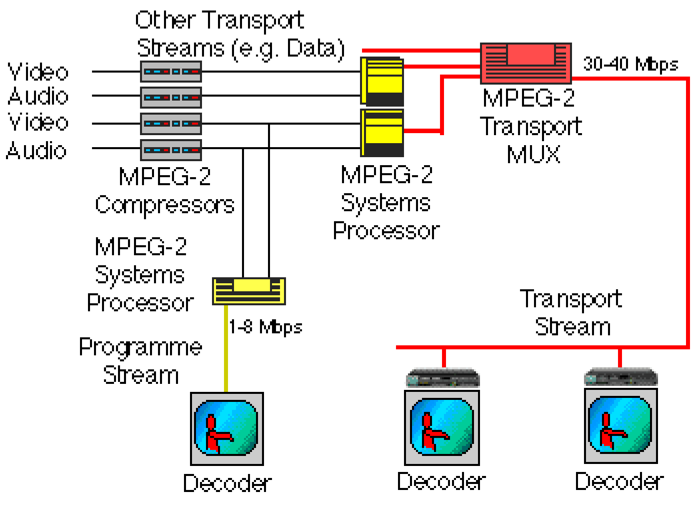


**Figure 2.22: Possible structure of a TS system**

## 2.8.2. Codecs and Payload Types

Here are explained the codecs used on the Linux-Box that are launched by VideoLan. This codecs are identified on the field PT (Payload Type) of RTP and can be for audio (AoD) or video (VoD and Broadcast TV). To see a table of other possible codecs and a brief explanation of the most used see the annex section 9.2.1.2.

### *2.8.2.1. Audio*

#### MPEG (MPA)

Identified as Payload Type 14, MPA denotes MPEG-1 or MPEG-2 audio encapsulated as elementary streams. The encoding may be at any of three levels of complexity, called Layer I, II and III. The selected layer is indicated in the payload. The RTP timestamp clock rate is always 90,000, independent of the sampling rate. MPEG-1 audio supports sampling rates of 32, 44.1, and 48 kHz. MPEG-2 supports sampling rates of 16, 22.05 and 24 kHz. The number of samples per frame is fixed, but the frame size will vary with the sampling rate and bit rate.
This is the common codec used for AoD on the Linux-Box.

### *2.8.2.2. Video*

#### MPEG (MPV)

MPV is identified as Payload Type 32 and designates the use of MPEG-1 and MPEG-2 video encoding elementary Streams.

- The MPEG1 specification is defined in three parts: System, Video and Audio. It is designed primarily for CD-ROM-based applications, and is optimized for approximately 1.5 Mbits/sec combined data rates. Some characteristics:
    - Acquisition: 4:2:0 – Y:Cr:Cb
    - Typical resolutions: 352 x 288, 352 x 240
    - DCT, macro blocks 16 x 16
    - Intra-frame and inter-frame coding
    - Quality comparable to VHS

The MPEG-2 specification is similar to MPEG-1 but it's designed for digital video applications and also high definition applications, so it's optimized approximately for 4 Mbits/sec and 20 Mbits/sec respectively. Using MPEG-2 Video coding tools it is possible to encode efficiently two video sequences transmitted from two different cameras that film the same scene with a small angle between them. Some characteristics:

- For digital video and TV:
    - Acquisition: 4:2:0.

  - Typical Resolutions: 720 x 576 (o 720 x 480) and 30 fps
  - Support to code video interlaced
  - Fields codificated separately or in pairs
  - Define scalability levels
  - Quality comparable to NTSC/PAL for TV
- For High definition applications (HDTV):
  - Acquisition: 4:2:2
  - Typical Resolution: 1920x1080 and 60 fps

MPEG-1 was usually used for VoD (with videoclips for example) and MPEG-2 for Broadcast TV in the laboratory to make the analysis.

**MPEG2-TS (MP2T)**

MP2T designates the use of MPEG-2 transport streams, for either audio or video. To know more see section 2.9.1 of this thesis.

## 2.9. DVB

### 2.9.1. History

In 1991, broadcasters and consumer equipment manufacturers discussed how to form a concerted European platform to develop digital terrestrial TV, and at that time was created a group to watch the development of digital television in Europe. It was called the European Launching Group (ELG) and included the major European media interest groups of all sectors. In September 1993, the Launching Group renamed itself as the Digital Video Broadcasting Project (DVB). At the same time a separate group, the Working Group on Digital Television, prepared a study of the chances and possibilities for digital terrestrial television in Europe. This introduces important new concepts, for example HDTV.

The DVB Project has the following purposes:

- Develop a complete suite of digital satellite, cable, and terrestrial broadcasting technologies all in one standard form.
- The systems should have ‘containers’ where carry any combination of image, audio, or multimedia. They should be ready for SDTV, EDTV, HDTV, surround sound, or any kind of new media.
- The final Standard should go to the ETSI standards for the physical layers, error correction, and transport for each delivery medium.
- To reduce costs all should be common between different delivery platforms, so the DVB Project should only use existing standards if they are available.

The transport for all systems is the MPEG2 transport stream.

### 2.9.2. Types

- **DVB-S:** Developed in 1993 for digital satellite broadcasting was developed in 1993. It uses a system using QPSK. The specification described different tools for channel coding and error protection.

- **DVB-C:** Developed in 1994 for digital cable networks. It uses a system using a 64 QAM, and it is possible to transmit a satellite channel multiplex on a cable channel.

- **DVB-T:** Developed for the digital terrestrial television system. It intends to assemble different noises and bandwidth environments, and multi-path. The receiver is required to adapt its decoding according to signalling. The system uses OFDM. There are two modes: 2K carriers plus QAM, 8K carriers plus QAM. The 8K mode can allow more multi-path protection, but the 2K mode can offer Doppler advantages where the receiver is moving.

- **DVB-S2:** Developed for a better satellite broadcasting system. It has about 30% more data capacity for the same receiving compared to DVB-S. It uses 8-PSK and Turbo coding to increase the efficiency. DVB-S2 will be used for all future new European digital satellite multiplexes, and satellite receivers will be equipped to decode both DVB-S and DVB-S2.

- **DVB-H:** Developed for a more flexible and robust digital terrestrial. Its purpose is to be receivable on handheld receivers systems. This includes features that reduce battery consumption (time slicing) and a 4K OFDM mode, together with other features. DVB-H use more efficient video compression systems such as MPEG4. In the last times, mobile operators are implementing this function their mobile phones to watch TV. Don't confuse it with the UMTS TV streaming implemented by other mobile operators.

## 2.10. Protocol Analyzers

The purpose of this section is only mention and make a brief explanation of the two programs used to do the network protocol analysis in this project.

### 2.10.1. WireShark (Ethereal)

Wireshark is one of the world's most popular network analyzer. This means that read packets from the network, creates captures and analyzes them to show the packets information. It has many features and runs on most platforms as Windows or Linux. The main advantage is that Wireshark is developed under the GNU General Public License, so it's free available and its source code can be modified. Its success comes because it is the successor of Ethereal, a standard software for many companies and public institutions. This means that the main developer team has moved to the WireShark project.

WireShark helps us to see the encapsulation of every packet in the net by setting the net adapter in mode monitor “promiscuous mode”. This means that WireShark captures every packet that pass through the net. For its utilization it is possible to set up filters to reduce the amount of information that isn’t interesting to our network scenery.

### 2.10.2. Commview

This is another network protocol analyzer not as complete as Wireshark but it has an option very interesting to our analysis. This software enables us to see the network bandwidth use every moment. It has the option to determine the bandwidth use for some specific IP also. This is useful for us to know if the streams used overload the net or are acceptable. It is also useful to help us to calculate if the stream is seamed to the theorical expected.

# Part 2: Applicative part

# 3. LINUX-BOX

## 3.1. Introduction

Nowadays, Media Centres have become really known, small devices make their appearance in very homes and with functionalities as CD or DVD player, Console, Internet, e-mail and other functions that only personal computers had before. These devices usually are based on a “Windows XP Media Centre Edition” OS, or easier in an “XP Home Edition”. This efforts the open-source community as they see a potential sector, because we can find this software on Linux/Debian OS prepared on Internet, but they are not enough competitive at a commercial level like Windows can do.

Then Linux-Box is not exactly a Media Centre. It does not have the same functionalities but to make us an idea, has a similar hardware configuration. Basically it is a device provided with television targets (DTT and Satellite) with inputs that allow broadcasting this signals in a local network. In future will be reliable as much to private houses as educational institutions (schools, universities, etc...) and also to environments like companies or offices. The advantage respect other competitors is that runs over existing configurations. Concretely needs IP nets and the most commons and economic in local areas are Ethernet and Wi-Fi.

To work, the Linux-Box needs a digital terrestrial antenna, a satellite antenna and a Network to distribute the desired signal. To do an open-source OS and specific software is provided and can be controlled by the administrator and used easily by any user. It is based on VideoLan (known as VLC) to the streaming engine and the costumers will dispose a Web environment totally clear. The form it works inside will be explained at point “3.2. Operation Way” and will be used briefly: HTML, PHP, Telnet, XML and JavaScript.

As said in the introduction, the Linux-Box project is in development yet. Our objective is, apart from a work documentation and an explanation about its working mode, realise different parts for the common project.

- *We will work in the channels list acquisition, their reading, processing and adequate the specific format of the main application (that is the web page). This could allow to the administrator to select about some available channels and at the same time make them available or not to the clients. The advantage is that these channels lists were made dynamically, that’s because usually there are regional modifications, different frequencies and PID’s. The objective is that when the administrator installs the Linux-Box, executes this module. And later he can execute it to verify and configure new channels automatically.*

- *The second part that we will work is about TV signals Broadcasting from the Linux-Box to the client’s web browser. The part that develops the other team is VoD, AoD and the web page code. All these is currently working right nowadays at the final of our DT, but at the beginning the Linux-Box was only an idea.*

- *Another interesting part in which we work is not only the scalability at the network layer of the Linux-Box resources but also a study of the frames as well as the activity and traffic created. This is an indispensable part for an application that usually works in Ethemet networks (not dedicated exclusively for this end) that also are used for other common applications as the Internet access or other characteristic communications for homes, offices or universities for example.*

The purpose is that the Linux-Box project wants to become a commercial product in a half-long term, always based on open-source code.
The software is based on a main HTML environment programmed in PHP. It uses Scripts and another web protocols to make work VLC like a "contents server" through its Telnet interface that will be directly operated in a transparent way by the web page.

As talked in the introduction, as explanation, the state in that was the Linux-Box project at the beginning of our DT was the following:

Linux-Box was used to watch the Italy matches in the 2005 Soccer World Cup through IP, for teachers and researchers' department. This was possible thanks to that was already prepared hardware of previous thesis and the creation of a special script for this occasion. They called it "vlc-azzurri", to see this go to the section 4.4.1.

We have to understand that it only had been used for this test, and that basically only had been carried out its hardware assembly, a basic drivers installation and the Operating System.

The Scripts in VLC acts like an automatic command line and uses a "relatively easy" owner language.

The "operation way" was previously defined by the other work group with our tutor collaboration, so is explained how it works, but the decisions that justified the design are not explained.

One candidate that seemed to be useful was DVBStream, see [23]. It also has command line to make possible to configure through the Web page, but their limitation only treating DVB flows didn't include all the expected for this project.

Searching other options seems that the VLC election is the best to use it in TV broadcasting, although their functionalities are not the same in VoD and AoD, where could have been found more appropriated programs. The reason to say it is because exists also specially software made for VoD. For example "*Live555*[16]" creates an automatically audio/video repository and streams in a LAN. Anyway if it is necessary to use the same program for the two purposes (TV Broadcasting and VoD), VLC is nowadays, one of the best candidates.

[16] To see more information visit: http://www.live555.com/

## 3.2. Operation Way

To explain the "working mode" it is important to be concerned as told before that another people decided the VLC use (in other previous or parallel projects) from other existing options basically for its great modularity and its Telnet interface.

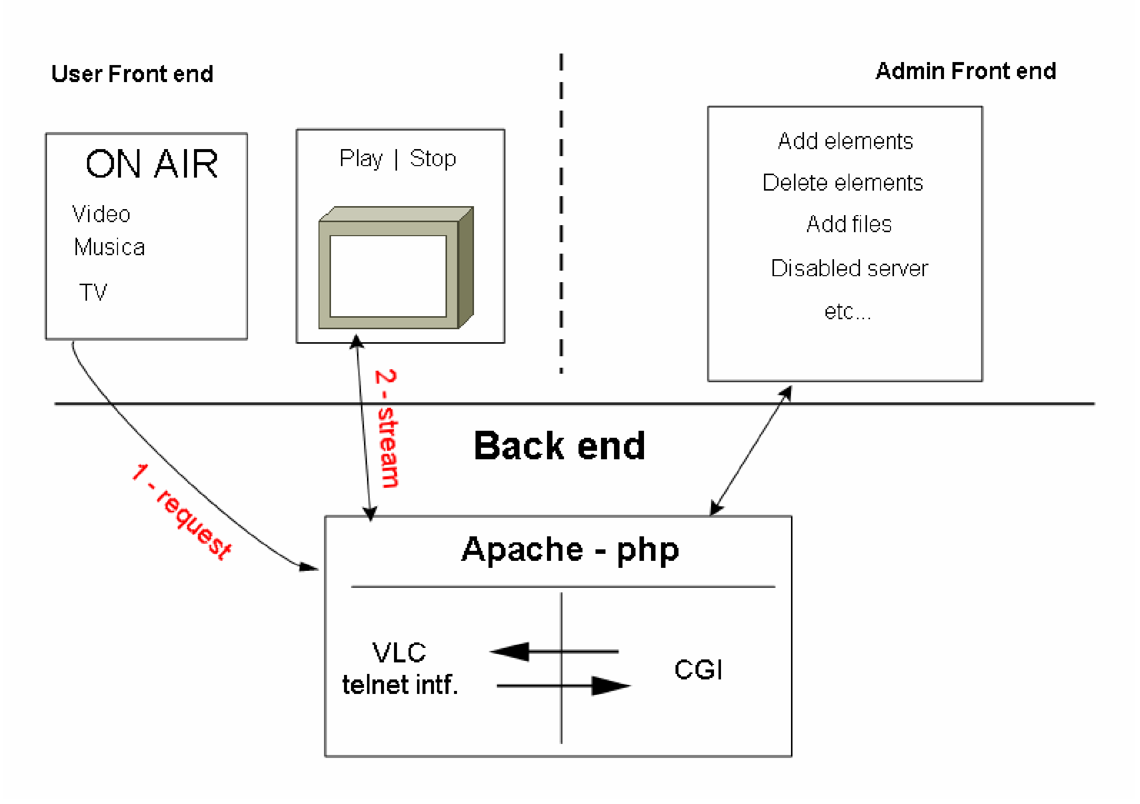
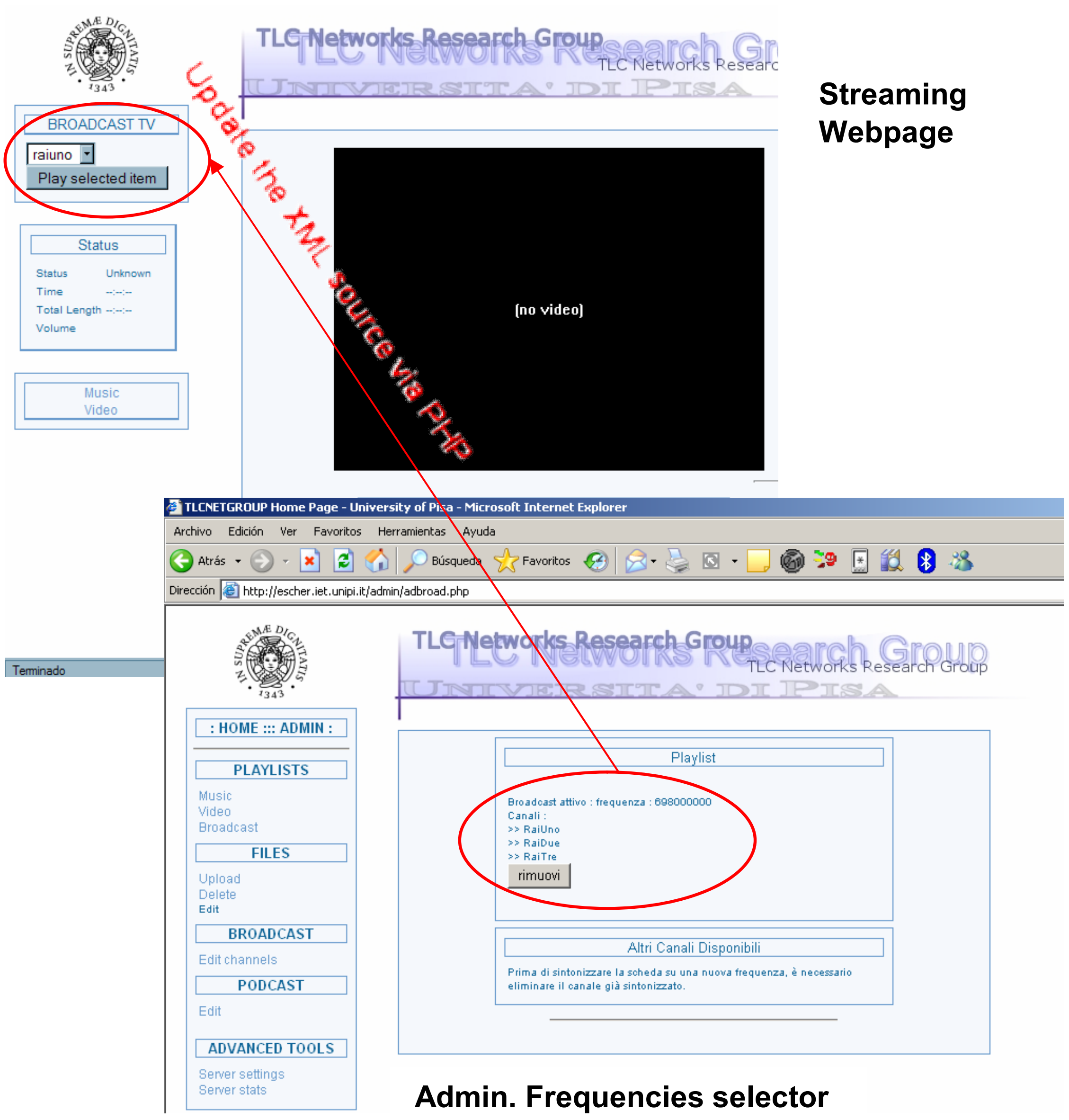


**Figure 3.1: Web Page functions**

The main purpose is to carry out Stream TV and VoD to the final user by web browser and in an easy transparent way. The web browser, through JavaScript's, is implemented all necessary controls and the video window.

For this, it must be installed the VLC plug-in, that is available for various web-browsers. These reproduction options are sent to the Telnet interface of VLC. Then the Linux-Box sends the requested stream to the user.

In the next figure we can se the relation between the administrator page, and the main webpage. The administrator adds the channels and directly it is added to the clients' TV-broadcast page through PHP by editing the appropriated XML file.


Streaming Webpage
Update the XML source via PHP
Admin. Frequencies selector
TLC Networks Research Group
Università di Pisa
BROADCAST TV
raiuno
Play selected item
Status
Music
Video
(no video)
TLCNETGROUP Home Page - University of Pisa - Microsoft Internet Explorer
http://escher.iet.unipi.it/admin/adbroad.php
: HOME ::: ADMIN :
PLAYLISTS
FILES
BROADCAST
PODCAST
ADVANCED TOOLS
Playlist
Altri Canali Disponibili
rimuovi

Status Unknown
Time --:--:--
Total Length --:--:--
Volume

Broadcast attivo : frequenza : 698000000
Canali :
>> RaiUno
>> RaiDue
>> RaiTre

Prima di sintonizzare la scheda su una nuova frequenza, è necessario eliminare il canale già sintonizzato.

Figure 3.2: Frequencies selection

The VLC Telnet commands are explained in the document: "VLC-Streaming-How To [17]". A summary can be seen in the section 3.3.VLC.

[17] Available in the VLC documentation home page: www.videolan.org/documentation.

In the figure 3.3 the process of requesting a source is showed (TV Stream or VoD) from the web page and the actions carried out automatically: Calling VLC and receiving the stream by the JavaScript in the web browser.

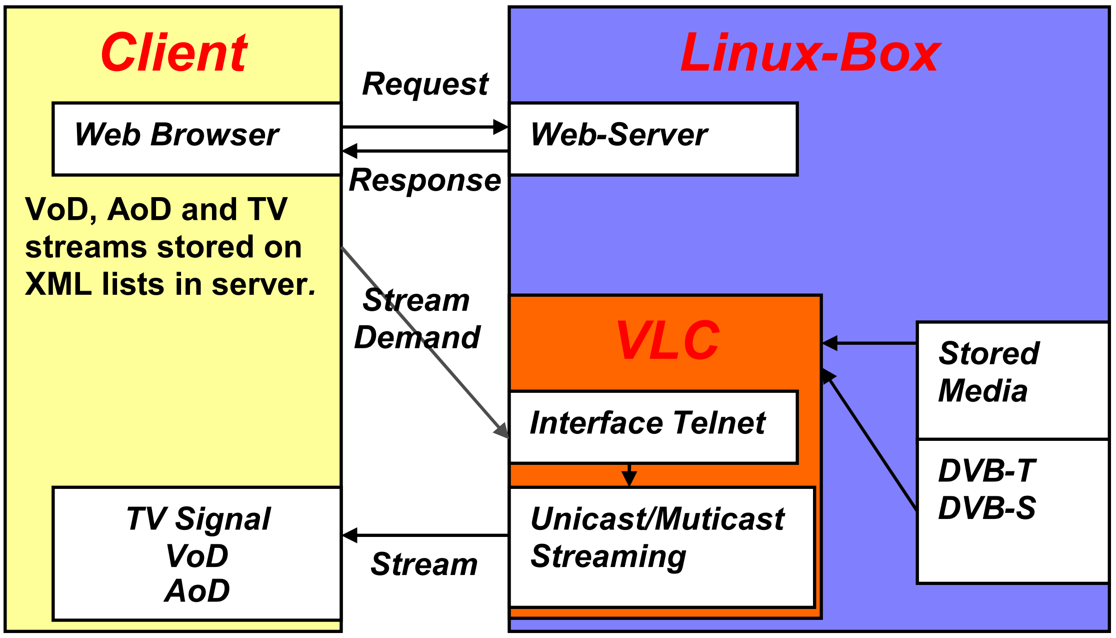


**Figure 3.3: Conceptual figure of the Linux-Box working mode.**

All the stored media and TV signals pass through VLC. It encapsulates the streams in RTP.

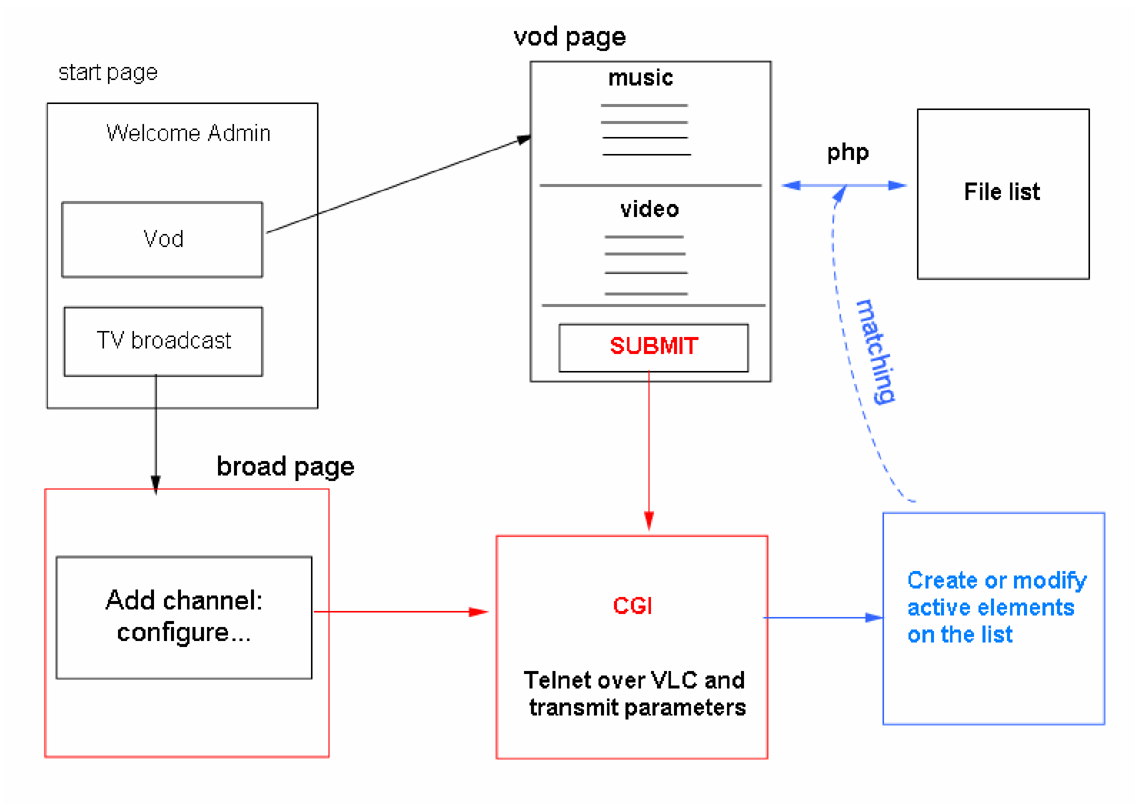

start page
Welcome Admin
Vod
TV broadcast
vod page
music
video
SUBMIT
php
File list
matching
broad page
Add channel: configure...
CGI
Telnet over VLC and transmit parameters
Create or modify active elements on the list

**Figure 3.4: Main Web Page organization**

For the final-users interface there are offered different streams by the Linux-Box and also exists a reserved page for the Administrator that allows adding or removing files and channels of the playlist that appears in the final clients web page.

The administrator's requests modify the playlist file stored in XML in the Linux-Box. From the same page it is also possible to kill the VLC process and start it again. VLC offers a simple playlist option. *In any case it is used the "VLC playlist option" in the Web Page.*

## TV streaming part:

Theoretically like we have said, Linux-Box at the moment only can emit a type of TV signal: DTT or satellite (DVB-T) at the same time, because there is PCI space problems. At this moment, the following rules will be applied:

- The administrator decides what frequencies, DVB-T or DVB-S are streamed.
- If this is not made, will be the first user the one that decides automatically between terrestrial or satellite.
- In case it is DVB-T, they will be able to select the different channels with the same local oscillator (frequency) selected previously. A new stream Multicast is created for each PID.
- A timer will be implemented so when there are not Multicast users, it stops the stream. It is implemented automatically through RTCP. Then the multicast stream leaves streaming. In the case of a Multicast stream will be:
  *UDP://@228.228.228.228:Port*

## VoD/AoD part:

For each Unicast demand a flow is created als each user requests different information in different moments.

At the same time the administrator selects if show or hide the streams stored in the Linux-Box, trough the web interface, which modifies automatically the playlist VoD in XML seen in the final client webpage.

As showed in the next figure RTCP is marked as optional. It is recommended in a RTP stream to use RTCP for the control and feedback implementations. In our work, RTCP is not implemented because VLC does not support it.

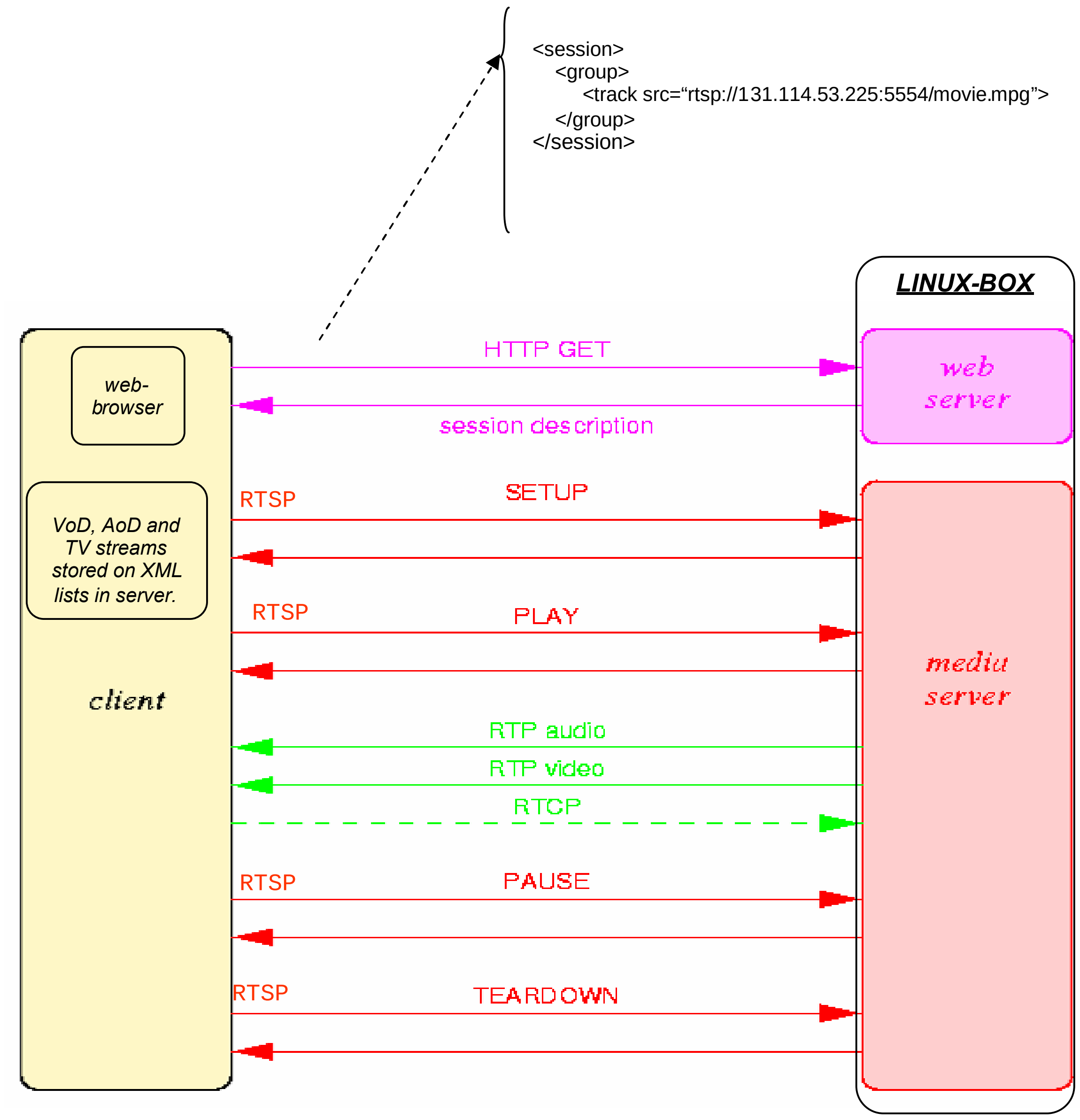


**Figure 3.5: Example of a VoD streaming session in Linux-Box.** [18]

RTSP is only used in VoD.
We have to remember that RTSP is the protocol which implements Play, Pause, Stop... and other functionalities.

[18] Modified figure, extracted from Services and Applications notes (a term on the EPSC).

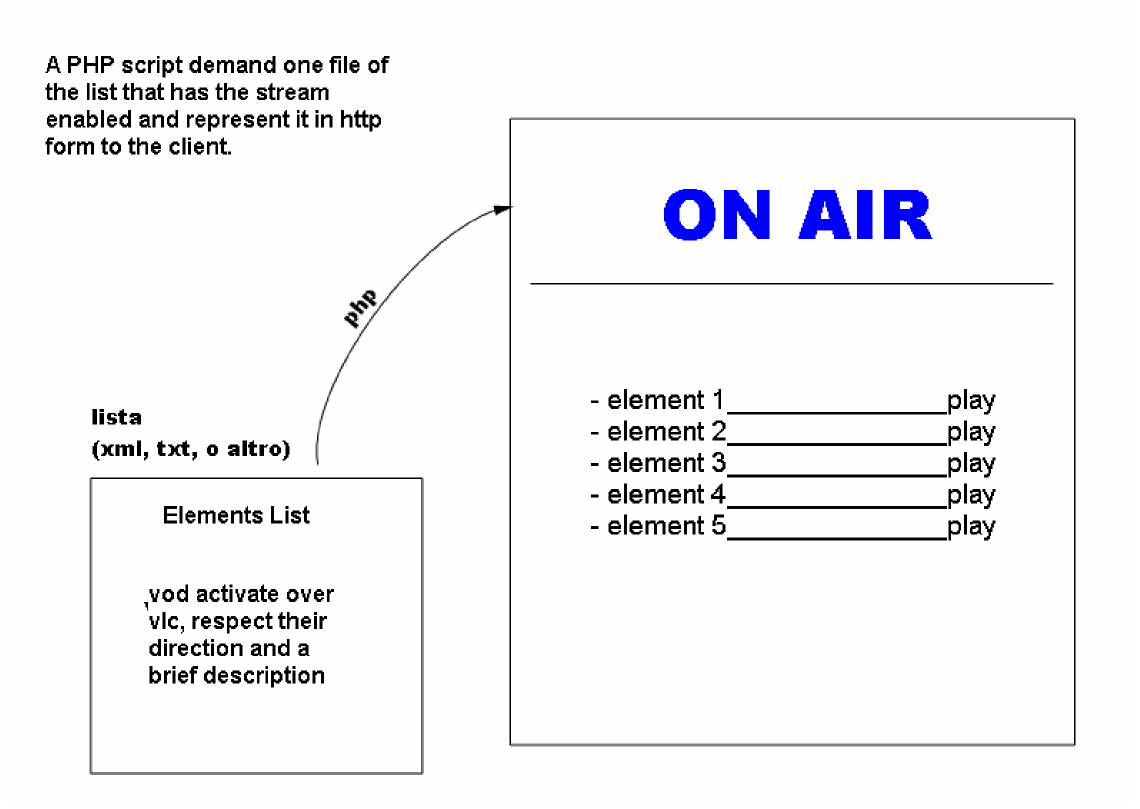

A PHP script demand one file of the list that has the stream enabled and represent it in http form to the client.
php
lista
(xml, txt, o altro)
Elements List
vod activate over vlc, respect their direction and a brief description
ON AIR
- element 1 play
- element 2 play
- element 3 play
- element 4 play
- element 5 play

**Figure 3.6: Webpage Playlist.**

On the other hand each client's requests create a Telnet demand to VLC that creates the requested stream. If is already created it is simply limited to follow the link to the desired stream by the URL direction.

In case of a Unicast stream will be:

```
RTSP ://(ip Linux-Box):Port/ File
```

How we start VLC by only introducing a URL?

VLC listens the port 5554, then when a request is made to this port, VLC stars automatically.

Of course, to do this VLC must be started before. In the operation way section is said that VLC is started before by the Web-Administrator.

*For world-agreement the ports in RTP protocol uses pair ports for the stream and odd for the control information.*

Another important issue about ports number is that each PID is sent in a different port. And will be this the difference to select what PID wants every client.

## 3.3. VLC

VLC is included as a Linux-Box section and not as a software section because it is very important in the project and the different Linux-Box mechanisms are based on it.

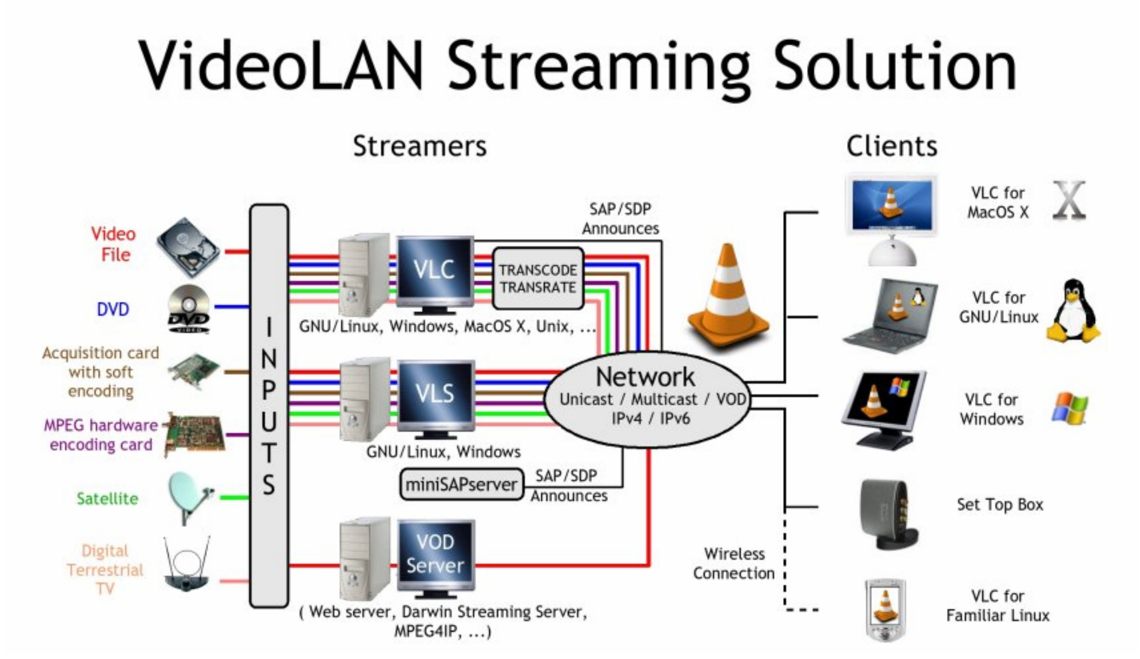


**Figure 3.7: VLC graphic presentation**[19]

### 3.3.1. Telnet Interface

VideoLan uses VLM (VideoLanManager) as a small media manager to control multiple streams by telnet or http interface. In this case it is used the telnet interface that will be executed with php scripts automatically to launch some streaming options that offers the Linux-Box (AoD, VoD).

The purpose of this section is to explain how is executed and the main options used for the Linux-Box.

Once installed VideoLan on the server, user must be in a terminal on the Linux-Box and execute the next command line:

```
vlc --ttl 12 -vvv --color -I telnet --telnet-password videolan --rtsp-
host 0.0.0.0:5554
```

[19] Copyright (c) 1996-2003 VideoLAN. This logo or a modified version may be used or modified by anyone to refer to the VideoLAN project or any product developed by the VideoLAN team, but does not indicate endorsement by the Project.

Where:

- *12* is the value of the TTL (Time To Live).
- T*elnet* launches the telnet interface of the VLC.
- *Videolan* is the password to connect to the telnet interface.
- *0.0.0.0* is the host address.
- *5554* is the port where stream.

Once connect to the vlc telnet interface, It must be created the VoD object:

new Test vod enabled
setup Test input my_video.mpg

“Test” is the object name and “my_video” is the video name to transmit and must be in the same directory that is launched the telnet interface.

To access the stream, user should use this command line on VLC:

```
% vlc rtsp://server:5554/Test
```

Where server is the address of the streaming server (IP or DNS), in our case 131.114.53.225

The telnet interface has the commands control to play, pause, stop and seek. There are also many other options but they are not used in the Linux-Box.

### 3.3.2. Streaming with VLC

VLC offers many codecs compatibility and streaming options (http://www.videolan.org/streaming-features.html). To see how to stream all the other possibilities see [22]. Here is explained the other streaming option in the Linux-Box that is not implemented by telnet, the way to input the DVBT transmission by scripts with VLC. The best way to explain the main options is to do it with an example. The script must be launched in terminal mode too:

```
vlc -vvv --color --ttl 12 --ts-es-id-pid --programs=8508,8505 dvb: \
--dvb-frequency=11739000 --dvb-srate=27500000 --dvb-voltage=13 \
--sout-standard-access=udp --sout-standard-mux=ts --sout \
'#duplicate{dst=std{dst=address1},select="program=8508",dst=std{dst=
address2},select="program=8505"}'
```

Where:

- *ts-es-id-pid* : this option is necessary when is used more than one program because split the multiplex stream into several outputs.

- *programs* : Specifies the programs to select in that frequency. The values are the same that the PIDs. VLC will select all known elementary streams of these programs.
- *dvb-frequency*: specifies the frequency to tune in kHz. This options is extracted from channelsDVBT.conf as can be seen in section 4.1.2.
- *Dst*= This option allow to put the location (ip) where the stream must be sent. Also it is possible to put other options as the access type (access = rtp or udp), the encapsulation method with the command mux or the Sap name (sap, name= "Name").
  There are many other options that can be modified. Here are explained those that are modified later and showed the final script used in section "4.4.1 VLC Scripts".

## 3.4. Hardware

Here is mentioned the main hardware parts of the Linux-Box. The information was extracted from our tutor's knowledge. First of all here is showed how looks like the server:

**Figure 3.8: Linux-Box appearance**

The Linux-Box has the next main components:

- The mainboard is a Via EPIA MII series, specifically the model 12000LVDS, with formati Mini-TX. The processor runs a maximum of 1'2 Ghz. And the RAM memory now intalled is 2 Gb. To know more see[20]:

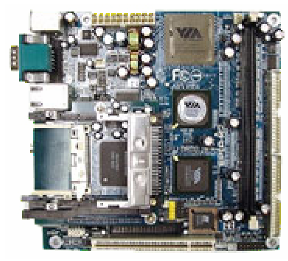


**Figure 3.9: Linux-Box Mainboard**

[20] Extracted from (also image): http://www.via.com.tw/en/products/mainboards/

- A DVP-S board Pinnacle PCTV SAT CI. All characteristics in[21]:

**Figure 3.10: Linux-Box DVB-S card**

- A DVB-T board Pinnacle PCTV 300i. See more in[21]:

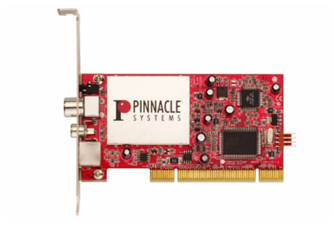


**Figure 3.11: Linux-Box DVB-T card**

## 3.5. Network Streaming

There are two basic services of the Linux-Box, the services on demand (VoD, AoD) and the TV streaming.

The first thing will become under strict demand of the final user. It is for that reason that will have to choose a Unicast communication. This will be the most demanding part in terms of bandwidth. A different flow is created for each client, although they are demanding the same information.

For example, "YOUTUBE[22]" uses Unicast streaming, too. But with a Flash Player Plug-in and a TCP connection.

Ethernet+IP+TCP=58 Bytes.
In a ideal scenario with a optiumum packet size with the maximum 1500 utile Bytes per Ethernet frame, we have:

1500 / 1558= 96% Throughput.

(100Mbps x 96%[23] )/ 4Mpbs or (less[24]) =25 maximum users, if the net is fully available.

[21] Extracted from: http://www.pinnaclesys.com (also images).
[22] http://www.youtube.com
[23] Parameter of maximum efficiency counting headers explained in the section 2.3 Headers and efficiency.
[24] In case we decide to transcode each stream with an inferior bitrate.

Knowing that audio will be in MP3, OGG or similar formats, the average should be from 128 to 192 Kbps in CBR, and 320 Kbps in VBR. It is a tiny traffic compared with video.

On the other hand the TV channels streaming part is made with multicast, because is disposed determined information in a moment for all users. For this objective the maximum multicast flows will be 4.

The first one should be for DTT and is divided in three. If we tune a local oscillator in a certain frequency, it is possible to obtain 3 PID for different channels. For example the RAI frequency has 3 channels depending on its PID: rai1, rai2 and rai3. The fourth flow will be for DVB-S. Exists a problem with the Linux-Box about that it can not reproduce a stream DVB-T or DVB-S at the same time. Three multicast flows do not overload the net as it supposes (3 or 4Mbps) * 3 flows = (9 to 12 Mbps) in total. It could be the same adding the DVB-S flows. In case of resolving the problem in a future version for the simultaneous operation of terrestrial and satellite, it could be possible to have from 18 to 24 Mbps in total. These flows also wouldn't suppose a very high load, but would be possible to offer 6 TV channels simultaneously.

Supposing that always is transmitted the digital television as much terrestrial, the question should be how is the maximum number of users in the net that use VoD or AoD.

These results are only made over what is tested in this project that is DVB-T. Starting from the section “2.3 Headers and efficiency” where it is explained that the maximum use will be approximately of 96%, this means that it could be possible to have about:
(100Mbps x 0'96)- 12Mbps = 84 Mbps for Audio and Video on Demand.

84 Mbps of the net / 4Mbps for each stream means 21 theoretical users or more if the flow is transcoded to a smaller bitrate.

### 3.5.1. VoD (Video on Demand)

The objective is offer Video contents from the Linux-Box. These streamed contents can be stored in the hard disk or in removable mediums (CD/DVD/Others).
The use of VoD was applied to save for example: professors' lessons recorded in video or videoclips and movies. The possibilities are several and depend on the final user. Always respecting, under each ones responsibility, the copyright laws and the royalties (author rights).

Each client demand is demanded in Unicast mode over (UDP/RTP).

By default are encapsulated in MPEG2-TS and they will usually be compressed with diverse codecs[25] that serve to keep the best subjective video quality, and optimize their physical size in memory.

An easy option to implement is reduce the bitrate of each Video through the transcoding option in VLC. In this mode if the demand is higher than the calculated in the measurements of this project and couldn’t use better nets with higher BW like "Gigabit Ethernet", with this solution it could increase the number of users that can use VoD. Obviously there will be a compromise between bitrate and image quality.

In this case LIVE555 implements automatically a VoD system. The problem Is that does not implements nothing about TV signals streaming.

### 3.5.2. AoD (Audio on Demand)

For AoD it is the same thing that for VoD. They are encapsulated in UDP/RTP. Usually they will be compressed in MPEG1 Layer 3 format, although can also be transmitted in Vorbis OGG.

### 3.5.3. TV Streaming (DVB-S & DVB-T)

This is the section our application part is based on. As it has already been explained, the purpose is transmitting TV signals in an IP net.

About the code methods, the primary option studied for scripts maintains the signal format sent in DVB-T and directly encapsulates in MPEG2-TS.

If it is necessary, VLC offers the possibility to transcode the streams coming from the TV capture targets and codes it in other formats or bitrates. But as far as studied in DVB-T and others traffic generated over the net do not seemed necessary. It is not discarded to use it for DVB-S studies because they haven't been made in this work and it is for future lines. The advantage is that these scripts are based on a first input part and another output part. Only modifying the first part for the frequencies and parameters DVB-S the scripts can be profited and realised out in this project.

[25] Codecs are explained in the section “2.8.2.Codecs”.

## 3.6. Known Problems

### 3.6.1. DVB-S + DVB-T

There is a problem with the existent hardware in this first version of the Linux-Box. The fact is that is a mini-box computer type, and only has an available PCI. When the Linux-Box was created, they implemented inside an exchange commutator that commutes between both cards. With this mechanism you can only use one at the same time. This is easily to resolve building a little bigger Linux-Box with 2 or 3 available PCI slots at least and installing each target in one PCI slot. In the other one could be integrated a Wi-Fi card that would cover a great number of extra users.

### 3.6.2. Web-Browsers, plug-ins and others

At the moment it has been able to check that the VLC plug-in for navigators is not compatible with most of them, only works well with Mozilla Firefox.
This limits the range of software possibilities at the moment on the user side and it is supposed that someone in the VLC community will implement in future a better compatibility so that it can access and be compatible with the most number of navigators and OS. Anyway Firefox is a web-browser open-source, powerful, traduced in diverse languages and free, for these reasons should not be a problem for users.

To get the VLC plug-in for Firefox is only necessary to install VLC in your system and select the plug-in option while it is installing it in the menu. This option is not selected by default. Another option is to download the "Mozilla-vlc plug-in" from any Linux software repository.

### 3.6.3. TV Channels Limitations

Another actual problem in the Linux-Box is that only have a tuning DVB-T card with one VCO[26], so that limits a lot their broadcasting usage, because it is only possible to send information at the same time in a specific frequency. This means that if diverse users want to access at the same time by Web to some TV channels they found that in the channels list will be available those that the administrator believes opportune or by defect the one that the first user chooses when he connects to the server. But it is possible to realise one thing, we can visualize those channels that are in the same frequency and have different PID's, transmitting as it has been explained previously each channel by a different port. In final versions they could be possible to install more than one DVB-T card.

[26] "Voltage Control Oscillator" or local oscillator.

### 3.6.4. NAT in the Routers (Operation inside the network)

NAT (Network Address Translation) is a method by which IP addresses are mapped from one group to another, transparent to end users. Network Address Port Translation or NAPT is a method which many network addresses and their TCP/UDP (Transmission Control Protocol/User Datagram Protocol) ports are translated into a single network address and its TCP/UDP ports. These two operations provide a mechanism to connect private addresses to an external net with globally unique registered addresses.

It is not recommended to run a Streaming Server behind a NAT Router, but if there is not an alternative, it must be configured the NAT for the used IP and Ports.

## 3.7. Firewalls and Streaming Servers

A firewall (FW) is a security gateway that enforces certain access control policies between two network administrative domains: often a private domain (intranet) and a public domain (public internet). In this sub section are treated the problems with firewalls and streaming servers, and possible things to do for resolve the problems that may cause the Firewall.

A streaming server uses the IETF RTSP/RTP protocols. RTSP runs on top of TCP, while RTP runs on UDP. Many firewalls are configured to restrict TCP packets by port number, and are very restrictive on UDP. There are three options for streaming through Firewalls with a streaming server. These options are not exclusive and one or more are used to provide the most flexible setup. The three configurations below are for *clients* behind a Firewall.

1. **Stream via Port 80**: This option enables the streaming server to encapsulate all RTSP and RTP traffic inside TCP port 80 packets. Because this is the default port used for HTTP-based web traffic, it will get through most firewalls. But this type of encapsulating the streaming traffic will reduce performance on the network, and require faster client connections to maintain streams. It also increases load on the server.

2. **Open the appropriate ports on the Firewall:** This allows the streaming server to be accessed via RTSP/RTP on the default ports and provides better use of network resources, lower speeds for client connections and less load on the server. The Ports that need to be open for unrestricted streaming include:
   - TCP Port 80: Used for signalling and streaming RTSP/HTTP (if enabled on server)
   - TCP Port 554: Used for RTSP
   - UDP Ports 6970 - 9999: used for UDP streaming (a smaller range of UDP ports can usually be used (typically 6970-6999)).
   - TCP Port 7070: Optionally used for RTSP (this port is used by Real Server, and QuickTime/Darwin can also be configured to use this port)

3. **Set up a Streaming Proxy Server:** The Proxy server is placed in the network - an area on the network that is in between an external firewall to the Internet, and an internal firewall between the DMZ and the internal network. Using firewall rules, packets with the ports defined above are allowed from the Proxy server to clients through the internal firewall, and also between the proxy server and the Internet via the external firewall. However, clients are not allowed to make direct connections to external resource over those ports. This approach insures that all packets bound for the internal network come through the proxy server, providing an additional layer of network security.

**Running Streaming Server behind Firewalls:** Public streaming servers can be placed behind firewalls. However, the ports mentioned before in configuration 2 should be opened to permit clients have open access to the server over HTTP and RTSP/RTP. Alternatively, clients can run behind a restrictive firewall if specify port 80 in references to their stream. For example, if the server stream.example.com was placed behind a restrictive firewall, and we wanted to access a movie named "video.mov" we could use the URL:
rtsp://stream.example.com:80/video.mov

The following table summarizes ports used by a streaming server. The arrows indicate the packet flow between client and server.

| Usage | Ports | Protocols | Notes |
|---|---|---|---|
| Responding to messages from clients (such as Play and Pause) | TCP (client initiates → QTSS) 554,7070, 8000, 8001, 80) | RTSP, RTP, RTCP, MP3 | Main port is 554. 80 is supported in the QT client as an alternative TCP port. These ports also send data to clients. |
| Sending media and receiving client status | • UDP data (QTSS → client): 6970–6999, even numbers<br>• UDP status (QTSS ↔ client): 6971–6999, odd numbers<br>• TCP data & status (QTSS ↔ client): 554, 7070, 8000, 8001, 80 | RTP<br>RTCP<br>RTSP, RTP,RTCP | Status is required to maintain a connection; if blocked, the server disconnects the client.<br>Same ports used to respond to messages. |
| Receiving broadcasts | • UDP data (broadcaster → QTSS): 6972–65535, even numbers<br>• UDP RTCP status (broadcaster ↔ QTSS): 6973–65535, odd numbers<br>• TCP (broadcaster initiates → QTSS): 554, 7070, 8000, 8001, 80 | RTP<br>RTCP<br>RTSP, RTP, RTCP | • Ports depend on the broadcaster configuration.<br>• Status is required to maintain a connection; if blocked, the server disconnects the broadcaster.<br>• Broadcasters can broadcast over their TCP message connection with the server instead of using UDP ports. |
| Streaming through server | TCP (client initiates → QTSS): 554, 7070, 8000, 8001, 80 | RTSP, RTP, RTCP, MP3 | Same ports used to respond to messages and receive TCP broadcasts. |
| MP3 broadcasts (typical default) | TCP (client → QTSS): 8000 | HTTP (Icecast) | |
| Managing QTSS remotely with Server Admin | TCP (admin client initiates → server): 311 | | |
| Managing QTSS remotely with Web Admin | TCP (web browser client initiates → server): 1220 | HTTP | |

**Table 3.1: Usage of ports in broadcasting, utile for Firewalls**[27]

[27] Extracted from: “http://www.soundscreen.com/streaming/firewall.html”

## 3.8. Software

In this section is mentioned the programming languages that take part in the Linux-Box project only as enumeration. It is not the purpose to enter in details in what is or the way each one works. Only it is explained like they are used and applied in the global Linux-Box project.

### 3.8.1. OS

The OS installed running in the Linux-Box is an older Debian version that only has a console environment. This benefits the CPU and memory load. Nobody must use Linux-Box directly. Its access is through SSH or directly through the Telnet interface of VLC.

### 3.8.2. HTTP

HTTP is used to apply the web page format. Known that the page is programmed in PHP language, some HTML tags can be used. To apply styles is used Dreamweaver. This part is not worked on this project, is part of the other workgroup.

### 3.8.3. PHP

How is said in the previous section, the main webpage is programmed in PHP. It implements different options to make automatically and modify files that in HTML is practically impossible. For example modifying dynamically playlists stored in XML is relatively easy with PHP. The inclusion of an Apache server daemon in the Linux-Box does the rest. Note that the IP direction of the Linux-Box is accessible through internet because is a public IP, then it have to implements a bit of security.

### 3.8.4. XML

This code is used to store the different information files. From the channels list offered to the actual available playlists of VoD and TV streaming, are all stores in XML text files. It is important to create always XML well-formed documents, because if only a XML tag is malformed, the rest of the document is null. For this reason is important to treat carefully the XML files.

### 3.8.5. JAVAScript

At the end of the work it is still not decided the final implementation of the video window in the web-browser. First of all it is implemented and working via JavaScript with the play-pause-rew-ff… options. This worked only in Firefox 1.5.

Solving this problem may carry a change of the way it is implement the video window. On the other hand its relation with the plug-ins does not help. It must be studied deeply by the next working groups in the future.
Including the JS code in the PHP web pages reduce errors opening the web. As known JS downloads the code and is executed by the local machine.

# 4. Developed Part

In this section is explained the analytic and application work that we have realised in the Linux-Box global project. It is organized in four sub-sections:

- 4.1 Develop an application in C code called “convert” that makes a DVB channel list in XML format.
- 4.2 talks about the realization of the TV streaming section on the main web-page in the Linux-Box.
- 4.3 is an exhaustive analysis of the Network and the generated frames. Are studied the frames and the occupation of the network to study the modularity and BW requirements. Also are treated the TS from the TV tuner and the TS generated in our network.
- 4.4 explain purposes of Scripts for VLC based on the results of 4.3. This section could belong to future’s section and conclusions cause are not a part we have made an exhaustive work, so it is a way for future projects to continue. Here are explained the way for analyze the TS’s2 and the numerous tests with scripts. Also are analyzed the most important scripts used in the work. The rest of scripts code can be found in the Annexe.

Remembering the Figure 0.2 we can see in yellow, the parts realized in this section. The graphic is divided in a Source part (Hardware Design), a processing part (Web Page) and a last exit part (Network Analysis).

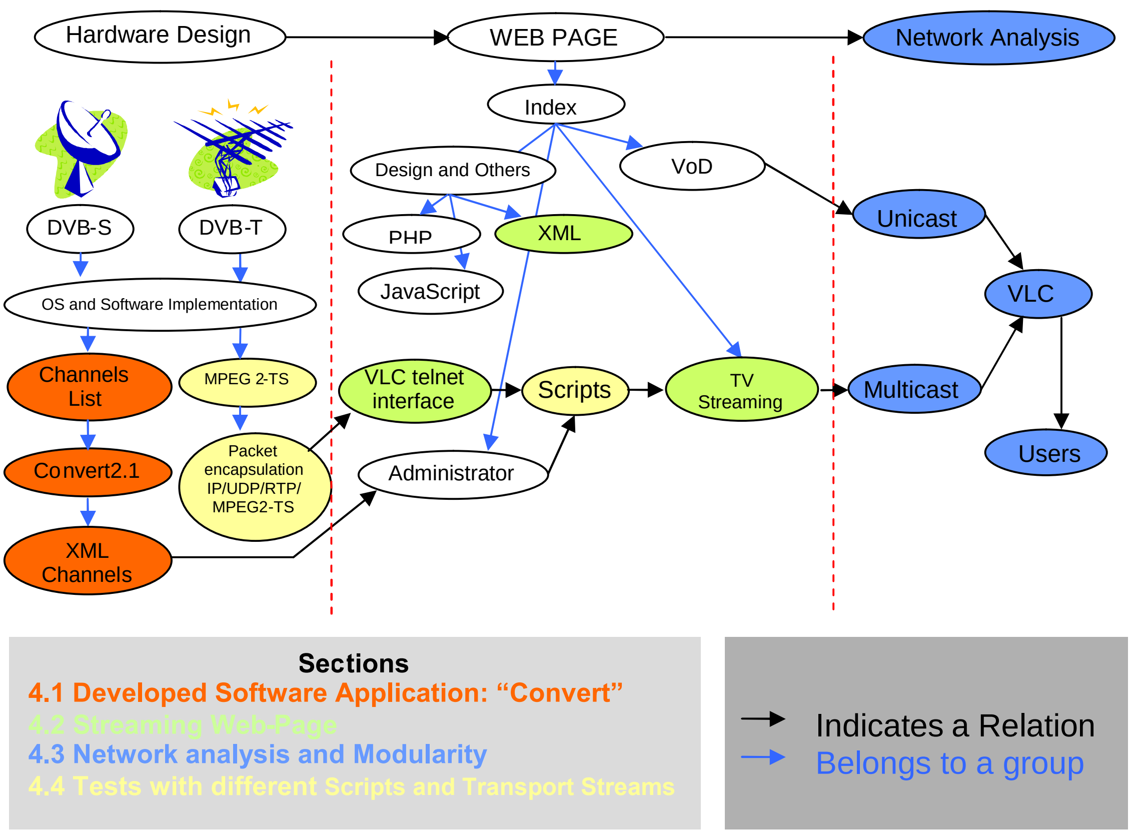


**Figure 4.1: This graph shows by colours the parts developed in this section**

## 4.1. Developed Software Application: “Convert”

### 4.1.1. Introduction

This first half of this work is based on the realization of a program called “Convert”. This program is entirely programmed in “C” code in a Linux environment.

For their realization we have had to act as true engineers. First of all we are assigned of their realization. At high level the program have three differenced parts:

- Entrance of information.
- Processed of this information.
- Exit of information.

The first part is automatic and it comes from the file "channels.conf" and its different versions. It is explained before the origin of the files "*. conf".

The second part is the most complicated. It consists in, once the source information is prepared, transform, filter, control the errors and prepare it for the exit of information.

Finally, the third part consists on the exit of information in two ways, one in the screen to observe the results (this way was eliminated at the end of the program because it wasn't useful for normal execution), only was useful as a revision of the results. The exit of important information is in a XML file form and should be created with correct semantics to introduce the source information there.

It is necessary to convert the final file to XML format to make possible later to treat the channels for their later visualization by web. Also for the administrator to be able to visualize the available channels on the fly in his web interface.

The main problems when planning the realization of this program are two. The first one is the fact that it was not known very well which was the entrance and the exit of information because it was a program that depended on a parallel thesis, which we did not know information because was not yet had been realised. This way is mentioned in the section “*4.1.3 Processing”* when at the beginning only an old “channels.conf” file was the reference. At the half process realization of the program this changed. In that moment the idea was to process diverse file types of "channels*.conf" according to if they were of DVB-T or DVB-S with the different information’s that these contains and its different formats. This forces us to modify all the error controls of the "Process of Information" part.

On the other hand, the exit of information was changing as the parallel thesis was improving the global project. This means that at the beginning, the entrance of information to the program was very simple. But they ask us for the modification of certain parameters regularly until January. This slowed the program final release a lot because when it seemed to be completed, we should make small modifications that sometimes like we will see, were not as small as expected.

There are five Convert versions. They do not respond to anything, only as an orientation and to follow the written versions parameters in commercial programs:

- **Convert 1.0:** The first version. it denotes the beginning of the project. Most part of the programs is based on it.
- **Convert 1.9:** This version appeared when the final program was near. Only needed to see that everything worked well together with the other thesis for the global project.
- **Convert 2.0:** Version appeared in the middle of December and was the last 2006 version. Only one modification was done.
- **Convert 2.1 Stable:** Appeared in the middle of January with another modification (field "available") made after some proves with the other group on the Linux-Box.
- **Convert 2.2 Beta:** This is the last version and its purpose is to simplify the code to improve the speed and a better memory management.

A copy was sent to the other task group for each version and sometimes also some intermediate ones. We have shared six or seven versions, with their corresponding modifications and the unpriceless feedback value between us and the other project.

### 4.1.2. Reading

Previous to the "convert" execution we should obtain the different channels.conf files. To make this, a command included on the TV Card drivers should be executed and a script with the local DVB-T frequencies configuration for this technology. For the "channelsDVBT.conf" acquisition we must write on the Linux terminal the following command line:

```
$ scan T-Pisa –a 0 > channelsDVBT.conf
```

Where "T-Pisa" is the following script. Everyone have to do modifications according to its region and the frequencies that want to explore. This script is needed because the command 'scan' is not a frequencies detector; its function is limited to look for channels inside the frequencies that are specified in this script:

```
# MUX Pisa

# T freq bw fec_hi fec_lo mod transmission-mode guard-interval
hierarchy
# MUX-A RAI
T 698000000 8MHz 2/3 1/2 QAM64 8k 1/32 NONE
# MUX-B RAI
T 658000000 8MHz 2/3 1/2 QAM64 8k 1/32 NONE
# MUX MEDIASET 1
T 618000000 8MHz 2/3 1/2 QAM64 8k 1/32 NONE
# MUX MEDIASET 2
T 754000000 8MHz 2/3 1/2 QAM64 8k 1/32 NONE
# MUX La7/MTV
T 818000000 8MHz 2/3 1/2 QAM64 8k 1/32 NONE
# MUX DFREE
T 586000000 8MHz 2/3 1/2 QAM64 8k 1/32 NONE
```

The way to obtain channelsDVBS.conf is not explained because DVBS finally is not used in the work, only theorically. The tutor gave us a copy of this file, to work on its different format.

Before describe the main characteristics, we show an example of each code line of the file that corresponds with one channel found:

- **For DVB-S:**

```
SKYGATE22:12558:v:0:27500:0:0:9251
```

- **For DVB-T:**

```
RaiUno:698000000:INVERSION_AUTO:BANDWIDTH_8_MHZ:
FEC_2_3:FEC_1_2:QAM_64:TRANSMISSION_MODE_8K:GUARD_INTERVAL_1_32:HIE
RARCHY_NONE:512:650:3401
```

Each file has a lot of channels. Although is obvious that both channels.conf are different, there are some similar fields that are explained below. Each field is separated by ":" and were chosen with the other task group to simplify the xml with the most important information of each channel. But as we see, the resulting DVB-S and DVB-T files are different. For that reason it must be implemented in two different sides on the reading files part too.

Fields are as follows for both files:

- First two fields are from the channel name and its corresponding frequency. Frequency is expressed in Hz.
- Last three fields correspond to the Vpid, Aped and PID for each channel
    - o Vpid is the video value Pid.
    - o Apid is the audio value Pid.
    - o PID is to select different channels transmitted on the same frequency.

If Vpid has a null value but a different Apid means that this channels correspond to a Radio channel. On the other hand if both have a null value means that the channel name is detected but not the signal, so the channels are not available. Those characteristics were discovered when final versions where made as is explained in the next section.

## 4.1.3. Processing[28]

In this section we explain the main structure of the convert code. There are two parts, the main function where we enter the source file to modify and the function file that makes the process.

- **Main function:**

This part programs the main syntax necessary to begin the process.

```
void main (int argc,char *argv[])
{
        int error;
        char file[30], type[5], signal[5];

        printf("\n\n\t+---------+\n\t|\tWelcome Convert 2.2\t  |\n\t+---------+\n\n");
        if (argc!=4)
        {
             printf("Missing Arguments\nUsing: convert [source_file]
             [.destination_type] [Terrestre or Satelitale]\nExample: convert file.dump
             .xml T\n");
        }
```

Here the program read the file channels.conf and compare the syntax with the expected. It made also an error control in case of the argument is not “T” or “S”, corresponding to DVB-S and DVB-T respectively.

```
        else
        {
                 strcpy (file,argv[1]);
                 strcpy (type,argv[2]);
                 strcpy (signal,argv[3]);

                 if ((strcmp(signal,"T")!=0)&&(strcmp(signal,"S")!=0))
                 {
                          printf ("Last Parameter wrong,must be 'S' o 'T'\nTry
                          Again...\n\n");

                          exit;
                 }
```

If all is correct the program send message “Tutto Bene Ciao” on the screen and if not an “Error” message.

```
                 else
                 {
                          if(files (file,type,signal)==0)
                          {
                                  printf("\n\n\t+---------+\n\t|\tConvert 2.2\t\t
                                  |\n\t|\tTutto Bene Ciao\t  |\n\t+---------+\n\n");
```

[28] The complete source-code is shown in the annex section 9.1.1

```
                                    }
                                    else
                                    {
                                            printf("\n\n\t+---------+\n\t|\t   Convert 2.2\t\t
                                            |\n\t|\t      ERROR\t\t  |\n\t+---------+\n\n");
                                    }
                            }
                    }
}
```

- **Files Function:**

The first part corresponds to declare variables, and copies the name of the source file ".conf" to another new to make possible the modifications and leave without modifications the original file. On this part, we create the name of the output file:

```
int files(char f[], char t[], char S[])
{
        int max=190;
        int b,c,n,i;
                …
        comilla=(char)39;
        char f_copia[30];//f_copia sera el arxivo de destino
        FILE *f1;
        if ((f1 = fopen(f,"r")) == NULL ) //el dump
        {
                printf("Can't open Source File: %s\n",f);
                return(1);
        }

        //strcpy (t,destino);
        //destino=".xml";
        strcpy (f_copia,f);//copia f en f copia para no sobreescribirlo
        //falta incluir el punto con algun sprintf
        //sprintf (f_copia,'.';
        strcat (f_copia,t);//anyade t a f_copia
        //printf("destino= %s\n",destino);
        printf("Source = %s \n",f);
        printf("Destination = %s\n\n",f_copia);
        //printf("t = %s\n\n",t);
```

The second part starts with the convert process. The program will read the file while does not arrive to an end of file (EOF). We can see inside the while function two comparisons, one in case of DVB-S and the other for DVB-T. The function compares "strcmp" count every ":" to distinguish the channel fields. Fields most important that must be save are name, frequency, PID and available. The function available is calculated as explained before comparing Vpid and Apid fields. We only put the first compare option for DVB-T cause it is more important in our work, although the other one is also implemented and can be seen on the annex.

```
while (feof(f1)==0)
        {
                fgets (s,max,f1);
                if (s[0]=='\0')
                {
                }
                else
                {
                        //buscar \0
                        tamanyoreal=0;
                        for (i=0;s[i]!='\n';i++)
                        {
                                tamanyoreal++;//tamanyo de cada linia "s"
                        }
                        //iniciar vectores a 0
                        for(z=0;z<40;z++)
                        {
                                nom [z]=0;
                        }
                        for(z=0;z<30;z++)
                        {
                                freq [z]=0;
                                pid [z]=0;
                        }
                        a=0,b=0,c=0,n=0;
                        available=1;//por defecto sera 1. Si no esta disponible
                        sera 0
                        if (strcmp(S,"T")==0)//T de terrestre TDT or DVB-T
                        {
                                …
                        }
                        if (strcmp(S,"S")==0) //S de satelitale o DVB-S
                        {
                                …
                        }
                    }
            }
```

Then the program makes the modifications with the saved fields and creates the XML file with the **xml function** explained in section 4.1.4. After that the function closes.

```
                        if (cont>0)
                        xml (nom,freq,available,pid,f_copia,1);
                        if (cont==0)
                        {
                                xml (nom,freq,available,pid,f_copia,0);
                                cont=1;
                        }
                        numchan++;
                }
                if (feof(f1)!=0)
                {
                        xml (nom,freq,available,pid,f_copia,2);
                }
        }

        printf ("\nNumber of Channels Processed: %d\n",numchan);
        //printf ("%c",comilla);

        //cerrar ficheros i salir OK
        fclose(f1);
        return 0;
}
```

With the convert 2.1 version, is designed the option in the tags <available> that writes if the signal is available or not. This was discovered in January and had to be implemented quickly because some channels written in the exit XML file format have the value VPID and APID similar to 0.

### 4.1.4. Saving (XML Format)

This is the structure of xml function that implements the XML destination file. Apart from the channels’ field variables, the most relevant is the variable control. Depending of this value means the beginning, middle or end of the field and comes from the count variable in the file function.

```
int xml (char nom[40],char freq[30],int available,char pid[30],char filename[30],int
control)
{
          FILE *f2;
          char pid2[30];
          int i;
          ...
          fclose (f2);
}
```

To apply the XML format to the final document it has been realised a meticulous format observation on the XML documents and is observed that they can be summarized in 3 phases:

- The main opening, in our case <streamlist>

```
if (control==0)
{
fprintf (f2,"<streamlist>\n");
control=1;
}
```

- The inside development of each program:

<stream>
/ / different parameters of each channel
</ stream>

```
if (control==1)    //imprimir normal
{
fprintf (f2,"\t<stream>\n");
fprintf (f2,"\t\t<name>%s</name>\n",nom);
fprintf (f2,"\t\t<freq>%s</freq>\n",freq);
fprintf (f2,"\t\t<available>%d</available>\n",available);
fprintf (f2,"\t\t<pid>%s</pid>\n",pid);
fprintf (f2,"\t\t<hidden>disabled</hidden>\n");
fprintf (f2,"\t</stream>\n");
}
```

- Close of the document with </ streamlist>

```
if (control ==2)
{
          fprintf (f2,"</streamlist>\n");
}
```

Playing with these 3 phases the following systems are implemented. At the beginning of the document, a variable is opened to write a <streamlist> only one time, when the function XML is initiated.

The phase two adds the tag <stream> at the beginning and at the end of each channel. For the step three we play with EOF of the source file. When it is detected, the function XML writes </ streamlist>, closes the files and concludes.

Two new useful tags are also added to each <stream> for the web page:
Available and Hidden:

- Available tells to the administrator if the channels can be shown and transmitted in that moment through the Linux-Box.
- Hidden is a field that by default is always “disabled” and changes its value when the administrator selects one available channel and begins its transmission.

It is necessary to mention that we have found tools for Linux that are able to detect errors in the XML format and even solve them, but we find that it is not necessary. It is not also practical to execute several programs to get a single objective, obtain the XML file with the available channels.

This example shows how must be shown finally each channel in the XML file[29]:

```
<streamlist>
        <stream>
                <name>RaiUno</name>
                <freq>698000000</freq>
                <available>1</available>
                <pid>3401</pid>
                <status>disabled</status>
        </stream>
        <stream>
                <name>RADIODUE</name>
                <freq>658000000</freq>
                <available>1</available>
                <pid>3312</pid>
                <status>disabled</status>
        </stream>
        …
</streamlist>
```

### 4.1.5. Thread Test

Apart from the reading error tests, files writing and other useful formalisms while programming in C, the user also can know where the problems or errors are made when being executed. With this feature we implement the following modifications:

- Change strange characters that could be problematic or not recognized by the XML or PHP languages used in the web page:
    - Change all “&” with “i”.
    - Change all spaces “ “ with “_”.
- Make some changes in the XML file organisation and syntax to a better compatibility with the PHP language used by the other work group for a better implementation of the web page.

[29] Complete XML file is shown in the 9.1.2 Annex section

### 4.1.6. Minimal memory spending

The program has been developed for a minimal memory spending. For this reason it does not save all channels' field information before create the XML file. It processes sequentially each channel and its corresponding xml format in the destination file.
Convert version 2.2 also tries to optimize the number of variables and functions and delete many redundancy or unnecessary code found after an exhaustive review. Although there are some similar code that can't be mixed, as the compare functions, that have only few differences but it is necessary its separation depending if it is S or T for DVB-S and DVB-T.
The purpose of these modifications is also increase the process speed in general and a better comprehension for other programmers that could modify it.

## 4.2. Streaming Web-Page

We integrate this part at the middle stages of the work. It becomes because the other work group didn't advanced and we must support them over their already made work. The main work has been to test and make modifications on the code already made, taking advantage of our tests made on the net with VLC and its graphic environment. At the beginning, the video only could be watched in specific streaming programs, such as VLC in graphic environment.

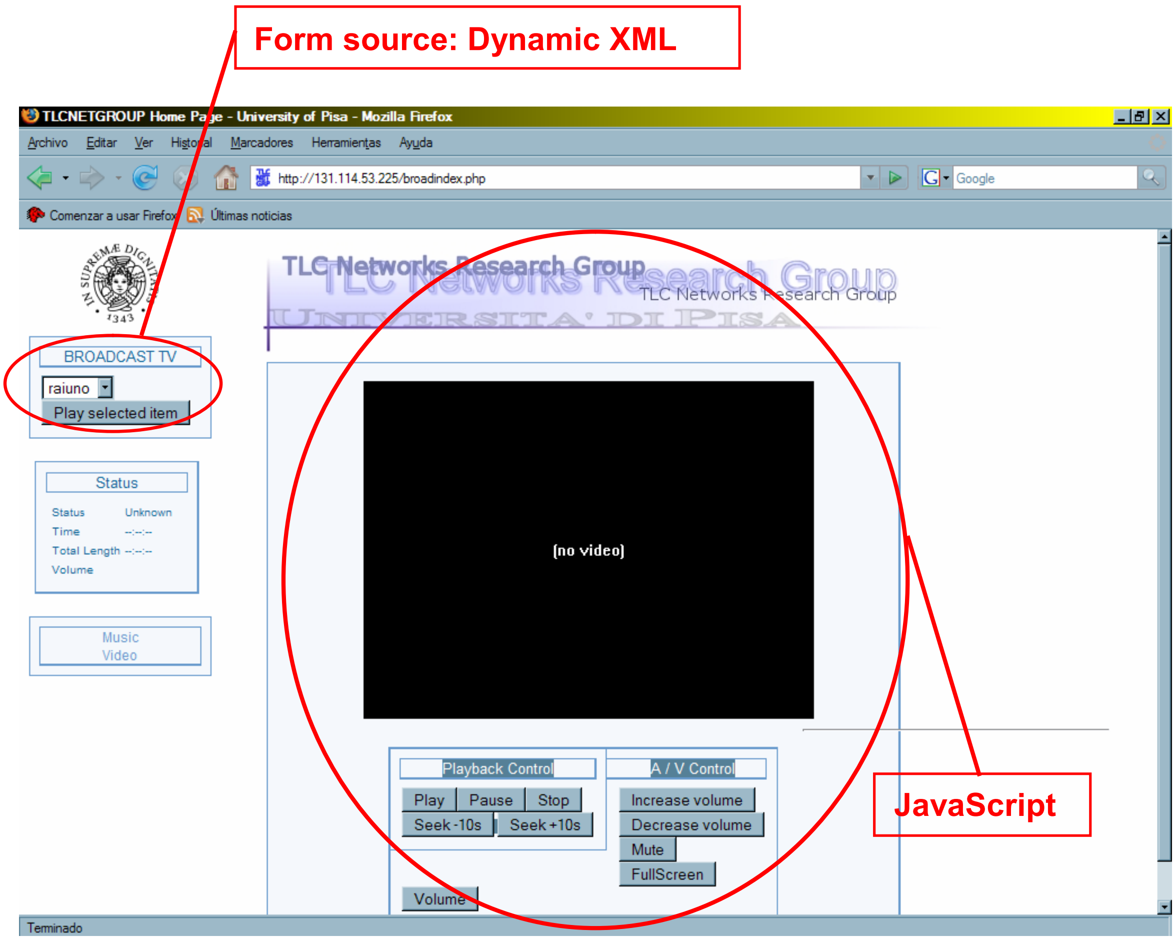


Figure 4.2: TV Broadcasting web-page

At the end, together we have got the broadcasting web page working fine.

The webpage is based on the next parts:

- The format of the main page for the menus.
- A JavaScript that shows that plays the video image and audio, and implements Play, Pause, Stop…and other options.[30] This java script is extracted from the VideoLan forum. It's under open-source license.
- The Form in red shows a XML file information. The file name is: "broadlist.xml" and is edited form the Administrator page, with the available PID's in the selected Frequency.

For playing this medium, the xml must contain a command line like this:

```
<option value="udp://@228.228.228.228:3333">Raitre</option>
```

Where "udp://@228.228.228.228:3333" is the multicast IP and port selected in the scrip and "Raitre" is the name shown in the screen.

For running the web page we must activate first the desired VLC Script. Actually we could make this directly by SSH or via PHP. In the final release, this only will be able by the Administrator. The scripts utilized and their differences are treated in the section 4.4.

## 4.3. Network Analysis and Modularity

To make the modularity tests we should analyze when we lost quality while streaming and how many are the maximum users that the system allows.

To analyze the quality looseness we can make it subjectively through the human eye and qualitatively watching when the FPS begins to reduce.

For the maximum users' number, we prove on the available material in the laboratory, if it is possible to arrive to the maximum users' number supported in the system and under the opportune considerations, explained below.

On the other hand we analyze first if the frames created by Linux-Box has a correct format first of their transmission over the net, and also analyze if their use over the net is what we expect according to our measurements or they are too much, and analyze in general the system behaviour.

[30] For use this JavaScript we must install the "VLC-plug-in". As is said in section 3.6.2.

### 4.3.1. Environment

The environment in which the commercial project of the Linux-Box will be developed is a net Ethernet 100Mbps or in the best in the cases and if the circumstances claim it a net Gigabit Ethernet.

In our case the laboratory tests are made with Cards Ethernet 10/100Mbps configured at 100Mbps. The number of PC's simultaneously available for the modularity tests is 12.

### 4.3.2. Tests carried out

To realise the tests we should keep in mind a key factor, the flow type. As we know the type of the flows can be Unicast, multicast and broadcast. Use broadcast is inutile, because we are using a shared net. One time said this, the Linux-Box uses Unicast for VoD & AoD, and Multicast for the TV streaming. In this point we know that Multicast only arrives to the net cards that demand the multicast address. The maximum numbers of simultaneous multicast Streams will be of 3 or 4 in DTT according to the available PID in the same frequency.

For satellite other simultaneous points. As a result we could have some picks from 6 to 8 simultaneous, but they will usually be 3 Streams. It will be approximately about 12Mbps that means around a 16% of the Ethernet bandwidth (including the headers efficiency). It does not suppose a restriction if all the network devices are from 100Mbps in terms of BW.

For VoD we have the restriction directly from the number of users that use it. It will be there where the modularity tests will give us maximum values.

Regarding AoD we could say the same that with VoD, but in a smaller scale, because the audio streams are not comparable in terms of BW, cause them are around a 1/100 part, in orders of magnitude.

Since we are those in charge of developing the part of Broadcasting TV, we will pay more attention in the analysis of Multicast frames and their activity in the net from DVB-T. Likewise in the part of VoD, as we have already said, the maximum number of users will depend on the bit rate of each flow and the bandwidth in the net. We will study two Unicast stream simultaneous and we will see their frame formats and the activity of the network.

Unicast 0.0.0.0 - 223.255.255.255
Multicast 224.0.0.0 to 239.255.255.255

The tests to realise to observe the frames and their encapsulation will be made with WireShark. To see the occupation of the flows on the net we have several candidates. The main ones are CommView and Network Analyzer. Alter using

both; we have chosen CommView for its clearer environment and for living more information about the net.
The following tests are made as much with Ethereal as CommView:

- **Create a stream with VLC, not in console mode, and analyze the frames with WireShark.**
- **Test of streaming of 3 PID's corresponding to the same frequency DVB-T, with 1 client for each PID.**
- **Test of streaming of 3 PID's corresponding to the same frequency DVB-T, with 3 clients for the same PID.**
- **Test of 2 clients for the same video file, in mode VoD and in mode Unicast.**
- **To finish a test of 3 Streams Multicast corresponding to the same frequency DVB-T and at the same time two clients more of VoD.**

The method for streaming every PID’s is mapping an entire PID directly in a PORT of the multicast direction (as is explained next in the scripts section).

### 4.3.3. Hypothesis of results

Supposing that the whole net is temporarily available for us inside the laboratory subnet, we will have 100Mbps theoretical that in practice will be in (ETH/IP/UDP/RTP) around 100Mbps x 93% =93Mbps of full duplex bidirectional transmission.

On the other hand the DVB-T streaming signal consists of three signals from about 4 Mbps corresponding to the same frequency PID’s.

Theoretical values approximated could be these[31]:
- 0.5 to 4 Mbit/s for a MPEG-4 stream,
- 3 to 4 Mbit/s for an MPEG-2 stream read from a satellite card, a digital television card or a MPEG-2 encoding card,
- 6 to 9 Mbit/s for a DVD.

In this line the expected results were UDP/RTP multicast packets, with 3 PID's, this means 3 streams encapsulated in the same Transport Stream.

This suppose about 12Mbps of constant flow when there are clients connected to the net. We also expect the appearance of IGMP packets for session login in and login out cause they are (connecting to/leaving) a multicast group.

In VoD there are Unicast streams (ETH/IP/UDP/RTP) of 93Mbps/4 = 23Mbps aprox. This is the maximum clients approximately. If we suppose that VoD does

[31] Information extracted from the VideoLAN webpage.

not implements any type of multicast mechanism to make the broadcasting because creates a different stream for each user, it seems obviously to wait for certain collisions in our environment. Therefore we will reduce to about 10 the maximum users' number.

### 4.3.4. Captures

**Test 1. As we have said, first of all we will create a stream with the VLC assistant in a graphic environment, in one pc different from the Linux-Box to see what format has, before using any script.**

The configuration is as follows:

- INPUT: Encapsulate a "avi" video
- OUTPUT: UDP/RTP/Multicast to the direction 228.228.228.228:1234
- Announcement SAP: Roma

In the client it should be configured: UDP://@228.228.228.228:1234
The "@" is for the fact that it is a group multicast.

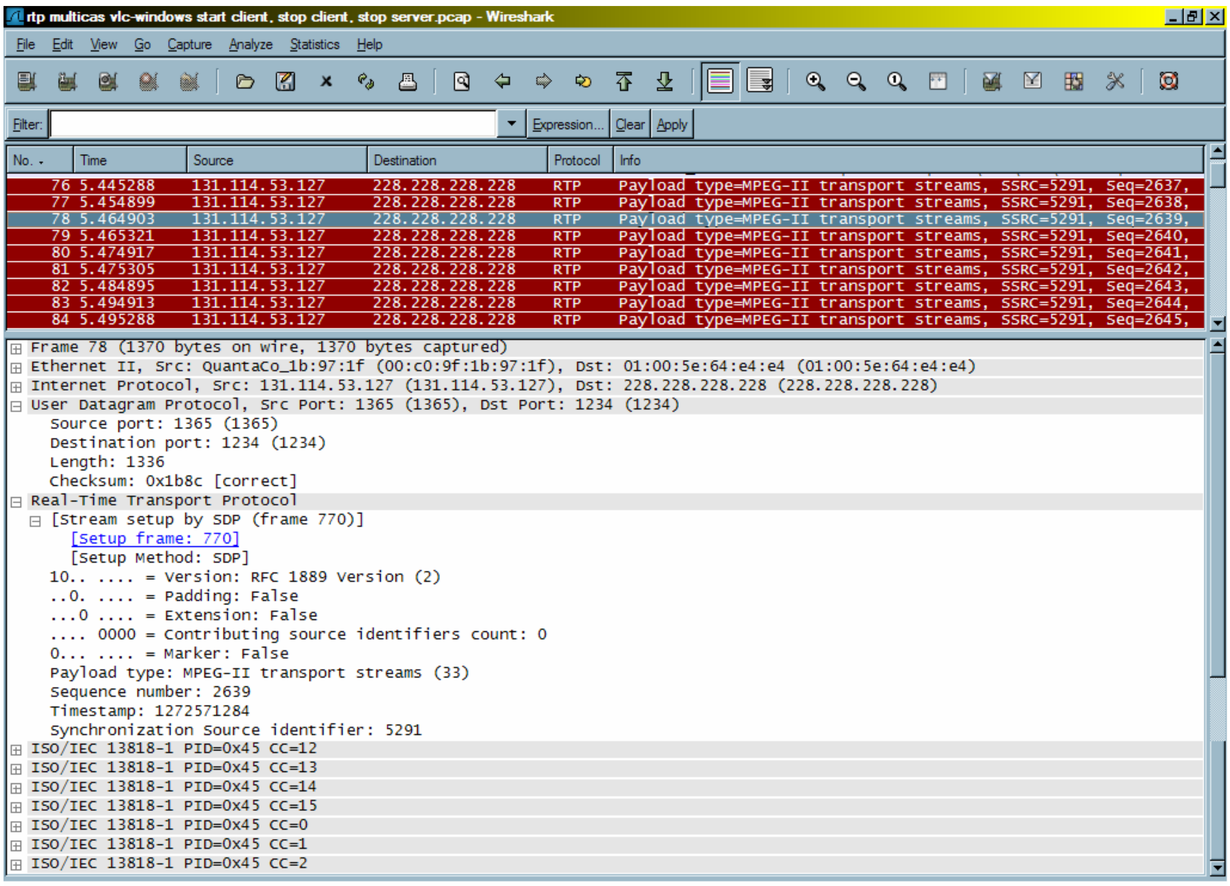
The IP Multicast belongs to a "no restriction" direction. It's used for global application. According to the IANA assignments[32] the IP address 228.228.228.228 belongs to the range (225.000.000.000-231.255.255.255): reserved for IANA and do not used by others. Anyway, for the tests, we must take overall of using the lowest TTL as possible, to do not inundate all the university network with our multicast streams.

The SAP announcement streams de content of the video stream and its configuration in the sap multicast address with the name "Roma". In this form, any video player listening to the sap direction, can add our stream in the playlist and play it automatically (without extra configuration).

This feature will be implemented in final releases of Linux-Box, but will not be used by the main webpage. Only will serve for clients that do not use the web browser.

[32] http://www.iana.org/assignments/multicast-addresses

In this first test, we probed VLC in graphic environment, to create an automatically stream and with the objective of compare it with the other captures, created in Linux text mode.


rtp multicas vlc-windows start client, stop client, stop server.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

```
No. .  Time       Source          Destination      Protocol  Info
    76 5.445288   131.114.53.127  228.228.228.228  RTP       Payload type=MPEG-II transport streams, SSRC=5291, Seq=2637,
    77 5.454899   131.114.53.127  228.228.228.228  RTP       Payload type=MPEG-II transport streams, SSRC=5291, Seq=2638,
    78 5.464903   131.114.53.127  228.228.228.228  RTP       Payload type=MPEG-II transport streams, SSRC=5291, Seq=2639,
    79 5.465321   131.114.53.127  228.228.228.228  RTP       Payload type=MPEG-II transport streams, SSRC=5291, Seq=2640,
    80 5.474917   131.114.53.127  228.228.228.228  RTP       Payload type=MPEG-II transport streams, SSRC=5291, Seq=2641,
    81 5.475305   131.114.53.127  228.228.228.228  RTP       Payload type=MPEG-II transport streams, SSRC=5291, Seq=2642,
    82 5.484895   131.114.53.127  228.228.228.228  RTP       Payload type=MPEG-II transport streams, SSRC=5291, Seq=2643,
    83 5.494913   131.114.53.127  228.228.228.228  RTP       Payload type=MPEG-II transport streams, SSRC=5291, Seq=2644,
    84 5.495288   131.114.53.127  228.228.228.228  RTP       Payload type=MPEG-II transport streams, SSRC=5291, Seq=2645,
```

```
⊞ Frame 78 (1370 bytes on wire, 1370 bytes captured)
⊞ Ethernet II, Src: QuantaCo_1b:97:1f (00:c0:9f:1b:97:1f), Dst: 01:00:5e:64:e4:e4 (01:00:5e:64:e4:e4)
⊞ Internet Protocol, Src: 131.114.53.127 (131.114.53.127), Dst: 228.228.228.228 (228.228.228.228)
⊟ User Datagram Protocol, Src Port: 1365 (1365), Dst Port: 1234 (1234)
    Source port: 1365 (1365)
    Destination port: 1234 (1234)
    Length: 1336
    Checksum: 0x1b8c [correct]
⊟ Real-Time Transport Protocol
  ⊟ [Stream setup by SDP (frame 770)]
      [Setup frame: 770]
      [Setup Method: SDP]
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 .... = Extension: False
    .... 0000 = Contributing source identifiers count: 0
    0... .... = Marker: False
    Payload type: MPEG-II transport streams (33)
    Sequence number: 2639
    Timestamp: 1272571284
    Synchronization Source identifier: 5291
⊞ ISO/IEC 13818-1 PID=0x45 CC=12
⊞ ISO/IEC 13818-1 PID=0x45 CC=13
⊞ ISO/IEC 13818-1 PID=0x45 CC=14
⊞ ISO/IEC 13818-1 PID=0x45 CC=15
⊞ ISO/IEC 13818-1 PID=0x45 CC=0
⊞ ISO/IEC 13818-1 PID=0x45 CC=1
⊞ ISO/IEC 13818-1 PID=0x45 CC=2
```

**Figure 4.3: Test 1.VLC configured with graphic interface, UDP/RTP/Multicast** [33]

IP server: 131.114.53.127
IP client: 131.114.53.110
IP multicast 228.228.228.228:1234

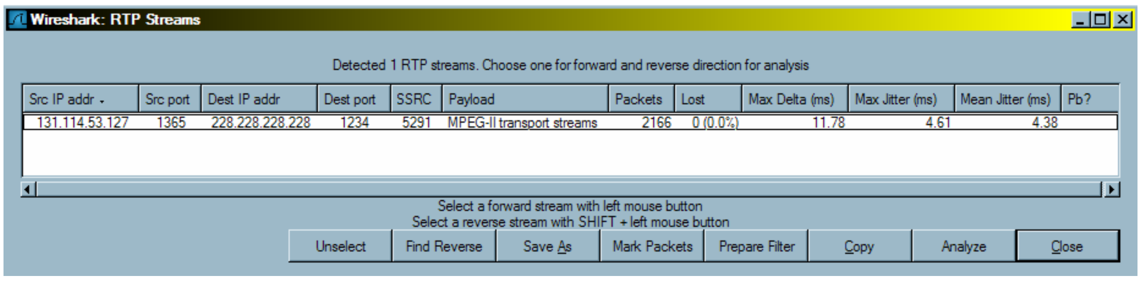
We observe clearly the Multicast flow that goes from the server to the Multicast address. It is encapsulated in RTP, with the payload type (33): MPEG2-TS.

## ISO/IEC 13818-1: Describes audio and video synchronization and multiplexing

Lately we specify some flow types. Really there is only a flow that is a Transport Stream. For that reason we will use a command/tool that WireShark[34]: incorporates the next utility.

[33] Note that red captures in WireShark refers to low TTL frames.
[34] WireShark, successor of the old Ethereal incorporates specific tools for RTP.

Show all RTP streams:



Detected 1 RTP streams. Choose one for forward and reverse direction for analysis

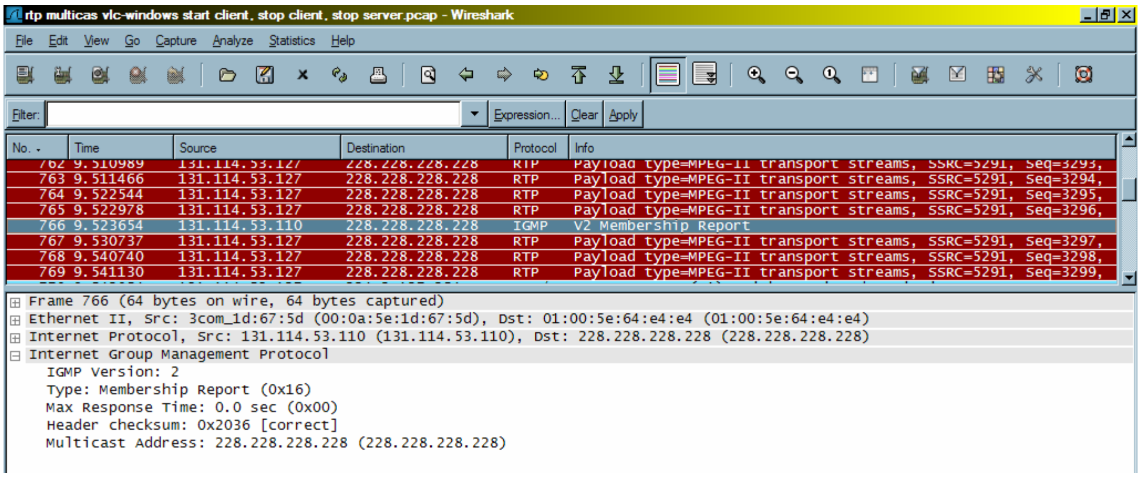
| Src IP addr | Src port | Dest IP addr | Dest port | SSRC | Payload | Packets | Lost | Max Delta (ms) | Max Jitter (ms) | Mean Jitter (ms) | Pb? |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 131.114.53.127 | 1365 | 228.228.228.228 | 1234 | 5291 | MPEG-II transport streams | 2166 | 0 (0.0%) | 11.78 | 4.61 | 4.38 | |

Select a forward stream with left mouse button
Select a reverse stream with SHIFT + left mouse button

**Figure 4.4: WireShark (Show all RTP streams)**

In the “Figure 4” is make an analysis of the RTP packet and show us captured streams. In this case, we only have one MPEG2-TS.

- The lost packets are 0%. This is correct, because we are using a wired Ethernet network, without too much traffic.
- Max Delta represents the maximum interval between two consecutive packets. Its value is 11’78 ms. In a ideal environment would gives the exact packetization interval.
- Max Jitter and Mean Jitter are similar 4’61 ms and 4’38 ms respectively. This indicates that the Jitter remains regular, and low.


rtp multicas vlc-windows start client, stop client, stop server.pcap - Wireshark
File
Edit
View
Go
Capture
Analyze
Statistics
Help
Filter:
Expression...
Clear
Apply

```
No. . Time      Source          Destination     Protocol Info
 762 9.510989  131.114.53.127  228.228.228.228  RTP     Payload type=MPEG-II transport streams, SSRC=5291, Seq=3293,
 763 9.511466  131.114.53.127  228.228.228.228  RTP     Payload type=MPEG-II transport streams, SSRC=5291, Seq=3294,
 764 9.522544  131.114.53.127  228.228.228.228  RTP     Payload type=MPEG-II transport streams, SSRC=5291, Seq=3295,
 765 9.522978  131.114.53.127  228.228.228.228  RTP     Payload type=MPEG-II transport streams, SSRC=5291, Seq=3296,
 766 9.523654  131.114.53.110  228.228.228.228  IGMP    V2 Membership Report
 767 9.530737  131.114.53.127  228.228.228.228  RTP     Payload type=MPEG-II transport streams, SSRC=5291, Seq=3297,
 768 9.540740  131.114.53.127  228.228.228.228  RTP     Payload type=MPEG-II transport streams, SSRC=5291, Seq=3298,
 769 9.541130  131.114.53.127  228.228.228.228  RTP     Payload type=MPEG-II transport streams, SSRC=5291, Seq=3299,
```

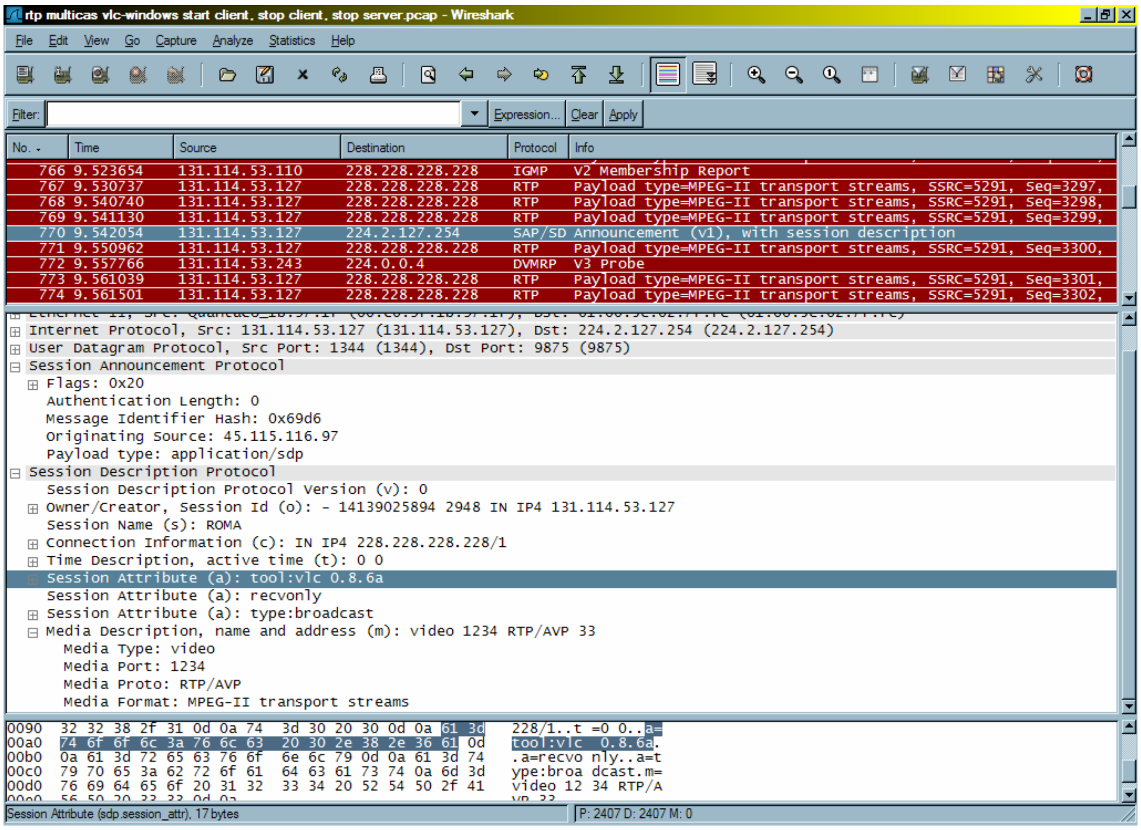
```
⊞ Frame 766 (64 bytes on wire, 64 bytes captured)
⊞ Ethernet II, Src: 3com_1d:67:5d (00:0a:5e:1d:67:5d), Dst: 01:00:5e:64:e4:e4 (01:00:5e:64:e4:e4)
⊞ Internet Protocol, Src: 131.114.53.110 (131.114.53.110), Dst: 228.228.228.228 (228.228.228.228)
⊟ Internet Group Management Protocol
    IGMP Version: 2
    Type: Membership Report (0x16)
    Max Response Time: 0.0 sec (0x00)
    Header checksum: 0x2036 [correct]
    Multicast Address: 228.228.228.228 (228.228.228.228)
```

**Figure 4.5: IGMP**

In this capture we can see the IGMPv2 packet with "Membership Report", from the client, to join access to the Multicast session.

With this packet, the intermediate routers open the multicast tree from the source to the client. As is explained in the section 0. This packet is sent periodically with the finality of do not exceed the timer of the intermediate routers for closing the multicast path.

One time this path is opened, our network adapter receives all the information from the multicast address 228.228.228.228.


rtp multicas vlc-windows start client, stop client, stop server.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression...
Clear
Apply

```
No. -  Time      Source          Destination      Protocol Info
  766 9.523654  131.114.53.110  228.228.228.228  IGMP   V2 Membership Report
  767 9.530737  131.114.53.127  228.228.228.228  RTP    Payload type=MPEG-II transport streams, SSRC=5291, Seq=3297,
  768 9.540740  131.114.53.127  228.228.228.228  RTP    Payload type=MPEG-II transport streams, SSRC=5291, Seq=3298,
  769 9.541130  131.114.53.127  228.228.228.228  RTP    Payload type=MPEG-II transport streams, SSRC=5291, Seq=3299,
  770 9.542054  131.114.53.127  224.2.127.254    SAP/SD Announcement (v1), with session description
  771 9.550962  131.114.53.127  228.228.228.228  RTP    Payload type=MPEG-II transport streams, SSRC=5291, Seq=3300,
  772 9.557766  131.114.53.243  224.0.0.4        DVMRP  V3 Probe
  773 9.561039  131.114.53.127  228.228.228.228  RTP    Payload type=MPEG-II transport streams, SSRC=5291, Seq=3301,
  774 9.561501  131.114.53.127  228.228.228.228  RTP    Payload type=MPEG-II transport streams, SSRC=5291, Seq=3302,

⊞ Internet Protocol, Src: 131.114.53.127 (131.114.53.127), Dst: 224.2.127.254 (224.2.127.254)
⊞ User Datagram Protocol, Src Port: 1344 (1344), Dst Port: 9875 (9875)
⊟ Session Announcement Protocol
  ⊞ Flags: 0x20
    Authentication Length: 0
    Message Identifier Hash: 0x69d6
    Originating Source: 45.115.116.97
    Payload type: application/sdp
⊟ Session Description Protocol
    Session Description Protocol Version (v): 0
  ⊞ Owner/Creator, Session Id (o): - 14139025894 2948 IN IP4 131.114.53.127
    Session Name (s): ROMA
  ⊞ Connection Information (c): IN IP4 228.228.228.228/1
  ⊞ Time Description, active time (t): 0 0
  ⊞ Session Attribute (a): tool:vlc 0.8.6a
    Session Attribute (a): recvonly
  ⊞ Session Attribute (a): type:broadcast
  ⊟ Media Description, name and address (m): video 1234 RTP/AVP 33
      Media Type: video
      Media Port: 1234
      Media Proto: RTP/AVP
      Media Format: MPEG-II transport streams

0090  32 32 38 2f 31 0d 0a 74  3d 30 20 30 0d 0a 61 3d   228/1..t =0 0..a=
00a0  74 6f 6f 6c 3a 76 6c 63  20 30 2e 38 2e 36 61 0d   tool:vlc  0.8.6a.
00b0  0a 61 3d 72 65 63 76 6f  6e 6c 79 0d 0a 61 3d 74   .a=recvo nly..a=t
00c0  79 70 65 3a 62 72 6f 61  64 63 61 73 74 0a 6d 3d   ype:broa dcast.m=
00d0  76 69 64 65 6f 20 31 32  33 34 20 52 54 50 2f 41   video 12 34 RTP/A

Session Attribute (sdp.session_attr), 17 bytes      P: 2407 D: 2407 M: 0
```

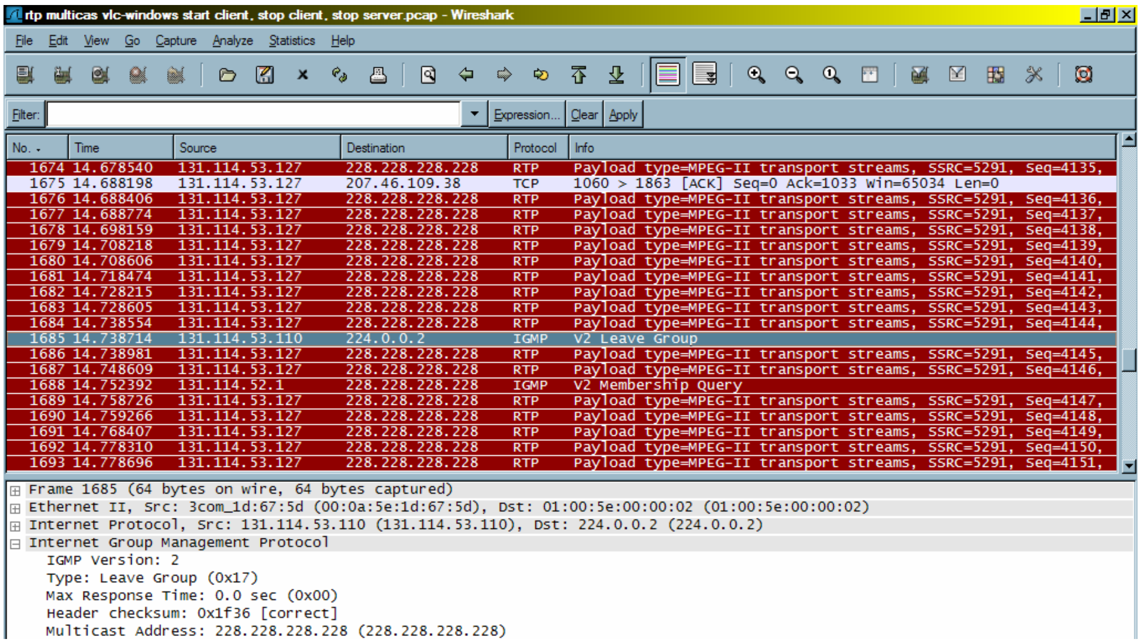
**Figure 4.6: SAP/SDP**

Here we can see the SAP/SDP Announcement packet from the server to the Multicast SAP direction: 224.2.127.254. As is explained in the theoreticall section SAP/SDP this two protocols, always goes together. With more accuracy, we see that the main packet is encapsulated on a SAP packet. Inside of this, comes the SDP. The SAP payload type is “application/SDP”.
We can appreciate in the SDP:

- The session name: Roma
- The used tool: VLC 0.8.6a
- Information about the session creator ID, its IP direction, and the Multicast session IP direction.

In the media description:

- The type of media: MPEG2-TS video over RTP/AV
- Port 1234 of the multicast address (configured by us in the multicast session setup).


rtp multicas vlc-windows start client, stop client, stop server.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

```
No. .  Time       Source          Destination      Protocol Info
 1674 14.678540  131.114.53.127  228.228.228.228  RTP      Payload type=MPEG-II transport streams, SSRC=5291, Seq=4135,
 1675 14.688198  131.114.53.127  207.46.109.38    TCP      1060 > 1863 [ACK] Seq=0 Ack=1033 Win=65034 Len=0
 1676 14.688406  131.114.53.127  228.228.228.228  RTP      Payload type=MPEG-II transport streams, SSRC=5291, Seq=4136,
 1677 14.688774  131.114.53.127  228.228.228.228  RTP      Payload type=MPEG-II transport streams, SSRC=5291, Seq=4137,
 1678 14.698159  131.114.53.127  228.228.228.228  RTP      Payload type=MPEG-II transport streams, SSRC=5291, Seq=4138,
 1679 14.708218  131.114.53.127  228.228.228.228  RTP      Payload type=MPEG-II transport streams, SSRC=5291, Seq=4139,
 1680 14.708606  131.114.53.127  228.228.228.228  RTP      Payload type=MPEG-II transport streams, SSRC=5291, Seq=4140,
 1681 14.718474  131.114.53.127  228.228.228.228  RTP      Payload type=MPEG-II transport streams, SSRC=5291, Seq=4141,
 1682 14.728215  131.114.53.127  228.228.228.228  RTP      Payload type=MPEG-II transport streams, SSRC=5291, Seq=4142,
 1683 14.728605  131.114.53.127  228.228.228.228  RTP      Payload type=MPEG-II transport streams, SSRC=5291, Seq=4143,
 1684 14.738554  131.114.53.127  228.228.228.228  RTP      Payload type=MPEG-II transport streams, SSRC=5291, Seq=4144,
 1685 14.738714  131.114.53.110  224.0.0.2        IGMP     V2 Leave Group
 1686 14.738981  131.114.53.127  228.228.228.228  RTP      Payload type=MPEG-II transport streams, SSRC=5291, Seq=4145,
 1687 14.748609  131.114.53.127  228.228.228.228  RTP      Payload type=MPEG-II transport streams, SSRC=5291, Seq=4146,
 1688 14.752392  131.114.52.1    228.228.228.228  IGMP     V2 Membership Query
 1689 14.758726  131.114.53.127  228.228.228.228  RTP      Payload type=MPEG-II transport streams, SSRC=5291, Seq=4147,
 1690 14.759266  131.114.53.127  228.228.228.228  RTP      Payload type=MPEG-II transport streams, SSRC=5291, Seq=4148,
 1691 14.768407  131.114.53.127  228.228.228.228  RTP      Payload type=MPEG-II transport streams, SSRC=5291, Seq=4149,
 1692 14.778310  131.114.53.127  228.228.228.228  RTP      Payload type=MPEG-II transport streams, SSRC=5291, Seq=4150,
 1693 14.778696  131.114.53.127  228.228.228.228  RTP      Payload type=MPEG-II transport streams, SSRC=5291, Seq=4151,
```

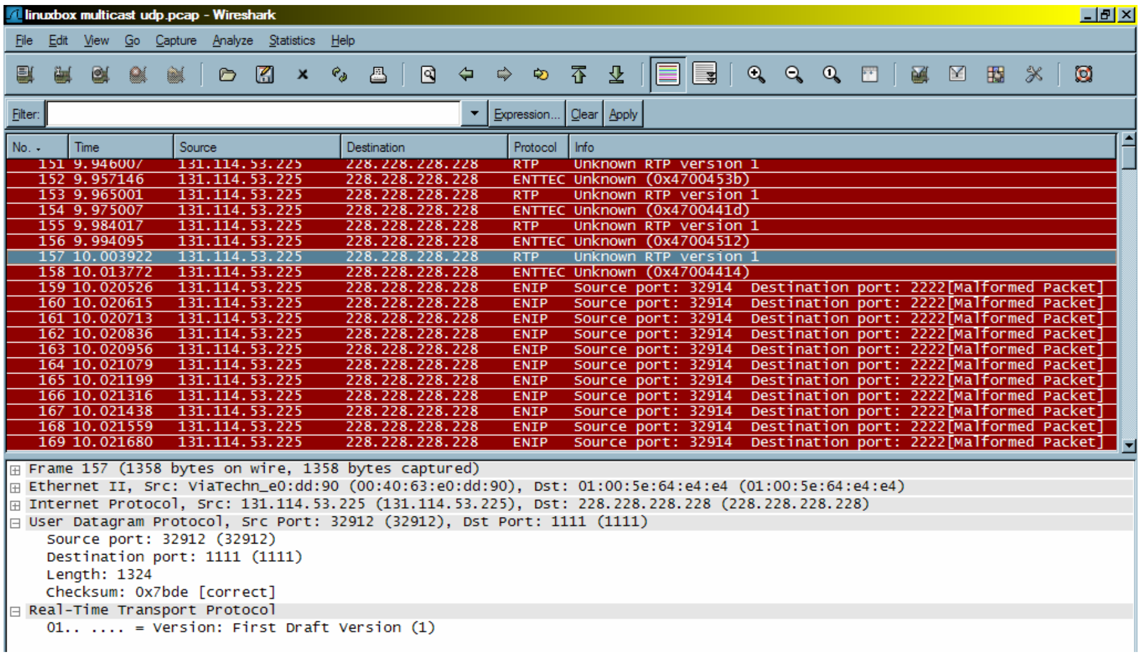
```
⊞ Frame 1685 (64 bytes on wire, 64 bytes captured)
⊞ Ethernet II, Src: 3com_1d:67:5d (00:0a:5e:1d:67:5d), Dst: 01:00:5e:00:00:02 (01:00:5e:00:00:02)
⊞ Internet Protocol, Src: 131.114.53.110 (131.114.53.110), Dst: 224.0.0.2 (224.0.0.2)
⊟ Internet Group Management Protocol
    IGMP Version: 2
    Type: Leave Group (0x17)
    Max Response Time: 0.0 sec (0x00)
    Header checksum: 0x1f36 [correct]
    Multicast Address: 228.228.228.228 (228.228.228.228)
```

**Figure 4.7: IGMP Leave Group**

This is the IGMPv2 packet: "Leave Group" sent by the client to leave out the Multicast session. This type of packets are the key of the IGMP version 2, that implements this, for do not maintaining unnecessary multicast trees opened thought the network. In this capture everything has been as we have expected, we haven’t found problems.

**Test 2. Streaming Test of 3 PID's corresponding to the same frequency DVB-T, with 1 client for each PID.**

This test was made from the Linux-Box with the script called “vlc-rai-sap”.


linuxbox multicast udp.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

```
No. .  Time       Source          Destination      Protocol Info
  151 9.946007   131.114.53.225  228.228.228.228  RTP      Unknown RTP version 1
  152 9.957146   131.114.53.225  228.228.228.228  ENTTEC   Unknown (0x4700453b)
  153 9.965001   131.114.53.225  228.228.228.228  RTP      Unknown RTP version 1
  154 9.975007   131.114.53.225  228.228.228.228  ENTTEC   Unknown (0x4700441d)
  155 9.984017   131.114.53.225  228.228.228.228  RTP      Unknown RTP version 1
  156 9.994095   131.114.53.225  228.228.228.228  ENTTEC   Unknown (0x47004512)
  157 10.003922  131.114.53.225  228.228.228.228  RTP      Unknown RTP version 1
  158 10.013772  131.114.53.225  228.228.228.228  ENTTEC   Unknown (0x47004414)
  159 10.020526  131.114.53.225  228.228.228.228  ENIP     Source port: 32914  Destination port: 2222[Malformed Packet]
  160 10.020615  131.114.53.225  228.228.228.228  ENIP     Source port: 32914  Destination port: 2222[Malformed Packet]
  161 10.020713  131.114.53.225  228.228.228.228  ENIP     Source port: 32914  Destination port: 2222[Malformed Packet]
  162 10.020836  131.114.53.225  228.228.228.228  ENIP     Source port: 32914  Destination port: 2222[Malformed Packet]
  163 10.020956  131.114.53.225  228.228.228.228  ENIP     Source port: 32914  Destination port: 2222[Malformed Packet]
  164 10.021079  131.114.53.225  228.228.228.228  ENIP     Source port: 32914  Destination port: 2222[Malformed Packet]
  165 10.021199  131.114.53.225  228.228.228.228  ENIP     Source port: 32914  Destination port: 2222[Malformed Packet]
  166 10.021316  131.114.53.225  228.228.228.228  ENIP     Source port: 32914  Destination port: 2222[Malformed Packet]
  167 10.021438  131.114.53.225  228.228.228.228  ENIP     Source port: 32914  Destination port: 2222[Malformed Packet]
  168 10.021559  131.114.53.225  228.228.228.228  ENIP     Source port: 32914  Destination port: 2222[Malformed Packet]
  169 10.021680  131.114.53.225  228.228.228.228  ENIP     Source port: 32914  Destination port: 2222[Malformed Packet]
```

```
⊞ Frame 157 (1358 bytes on wire, 1358 bytes captured)
⊞ Ethernet II, Src: ViaTechn_e0:dd:90 (00:40:63:e0:dd:90), Dst: 01:00:5e:64:e4:e4 (01:00:5e:64:e4:e4)
⊞ Internet Protocol, Src: 131.114.53.225 (131.114.53.225), Dst: 228.228.228.228 (228.228.228.228)
⊟ User Datagram Protocol, Src Port: 32912 (32912), Dst Port: 1111 (1111)
    Source port: 32912 (32912)
    Destination port: 1111 (1111)
    Length: 1324
    Checksum: 0x7bde [correct]
⊟ Real-Time Transport Protocol
    01.. .... = Version: First Draft Version (1)
```

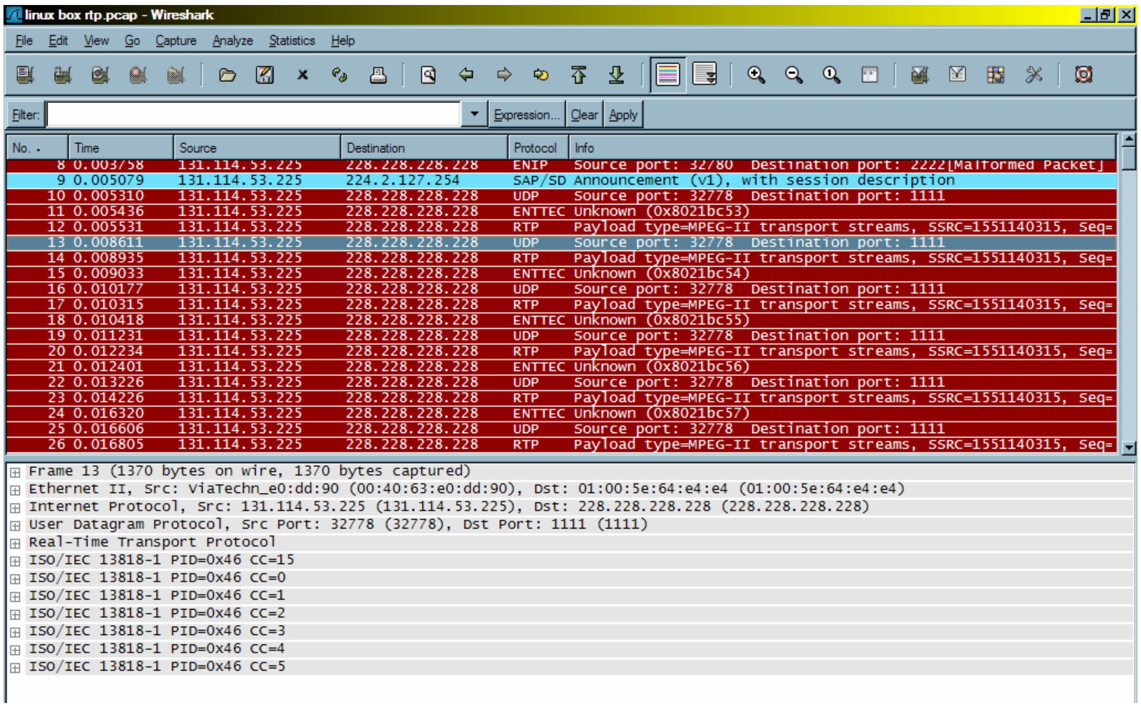
**Figure 4.8: This screenshot shows the *erroneous* packet interpretation of WireShark, when using UDP, without RTP.**

Using the WireShark command “Decode as RTP” we see that it does not really detect the RTP packets because it says “RTP First Draft Version”. This is because the packets do not travel on RTP, they travel over IP/UDP. In this case it is not necessary to use the command “Decode as”.

We see that the packets to the port destination 2222 are erroneous. Probably because the decoding is erroneous while decoding as RTP. Anyway if we make it in a normal way, the application tries to recognize the content of the packets like other protocols that we aren’t really using.

After see this, we reconsider the script because it is sending over UDP. This is not appropriate as we know that with RTP can obtain better results at theoretical level.

After diverse changes and fight with the VLC command line language, we are able to transmit directly over RTP with the script called “vlc-rai-rtp”. At the end the solution we observe that with a small modification in the encapsulate form, but the syntax confuses a lot. Also the server SAP/SDP is incorporated.


linux box rtp.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression...
Clear
Apply

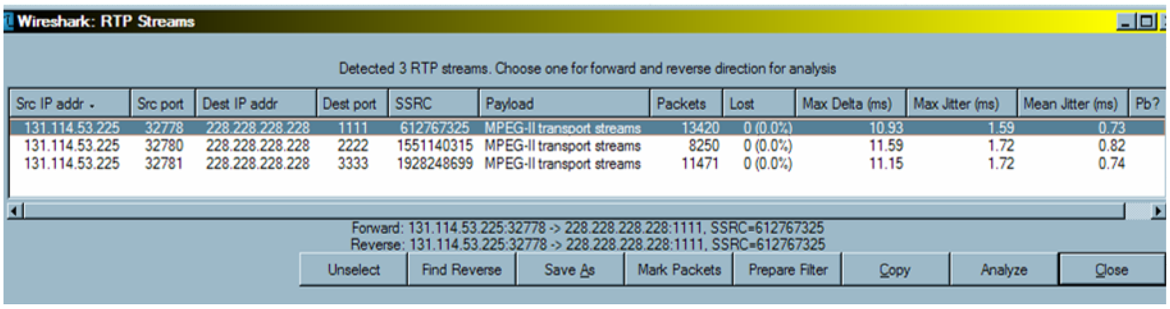
```
No. - Time      Source          Destination     Protocol Info
    8 0.003758  131.114.53.225  228.228.228.228 ENIP   Source port: 32780  Destination port: 2222[Malformed Packet]
    9 0.005079  131.114.53.225  224.2.127.254   SAP/SD Announcement (v1), with session description
   10 0.005310  131.114.53.225  228.228.228.228 UDP    Source port: 32778  Destination port: 1111
   11 0.005436  131.114.53.225  228.228.228.228 ENTTEC Unknown (0x8021bc53)
   12 0.005531  131.114.53.225  228.228.228.228 RTP    Payload type=MPEG-II transport streams, SSRC=1551140315, Seq=
   13 0.008611  131.114.53.225  228.228.228.228 UDP    Source port: 32778  Destination port: 1111
   14 0.008935  131.114.53.225  228.228.228.228 RTP    Payload type=MPEG-II transport streams, SSRC=1551140315, Seq=
   15 0.009033  131.114.53.225  228.228.228.228 ENTTEC Unknown (0x8021bc54)
   16 0.010177  131.114.53.225  228.228.228.228 UDP    Source port: 32778  Destination port: 1111
   17 0.010315  131.114.53.225  228.228.228.228 RTP    Payload type=MPEG-II transport streams, SSRC=1551140315, Seq=
   18 0.010418  131.114.53.225  228.228.228.228 ENTTEC Unknown (0x8021bc55)
   19 0.011231  131.114.53.225  228.228.228.228 UDP    Source port: 32778  Destination port: 1111
   20 0.012234  131.114.53.225  228.228.228.228 RTP    Payload type=MPEG-II transport streams, SSRC=1551140315, Seq=
   21 0.012401  131.114.53.225  228.228.228.228 ENTTEC Unknown (0x8021bc56)
   22 0.013226  131.114.53.225  228.228.228.228 UDP    Source port: 32778  Destination port: 1111
   23 0.014226  131.114.53.225  228.228.228.228 RTP    Payload type=MPEG-II transport streams, SSRC=1551140315, Seq=
   24 0.016320  131.114.53.225  228.228.228.228 ENTTEC Unknown (0x8021bc57)
   25 0.016606  131.114.53.225  228.228.228.228 UDP    Source port: 32778  Destination port: 1111
   26 0.016805  131.114.53.225  228.228.228.228 RTP    Payload type=MPEG-II transport streams, SSRC=1551140315, Seq=
```

```
⊞ Frame 13 (1370 bytes on wire, 1370 bytes captured)
⊞ Ethernet II, Src: ViaTechn_e0:dd:90 (00:40:63:e0:dd:90), Dst: 01:00:5e:64:e4:e4 (01:00:5e:64:e4:e4)
⊞ Internet Protocol, Src: 131.114.53.225 (131.114.53.225), Dst: 228.228.228.228 (228.228.228.228)
⊞ User Datagram Protocol, Src Port: 32778 (32778), Dst Port: 1111 (1111)
⊞ Real-Time Transport Protocol
⊞ ISO/IEC 13818-1 PID=0x46 CC=15
⊞ ISO/IEC 13818-1 PID=0x46 CC=0
⊞ ISO/IEC 13818-1 PID=0x46 CC=1
⊞ ISO/IEC 13818-1 PID=0x46 CC=2
⊞ ISO/IEC 13818-1 PID=0x46 CC=3
⊞ ISO/IEC 13818-1 PID=0x46 CC=4
⊞ ISO/IEC 13818-1 PID=0x46 CC=5
```

**Figure 4.9: Capture from Linux-Box streaming 3 PID’s in RTP/MPEG2-TS directly mapped on the ports 1111, 2222 and 3333 respectively.**

Now we observe the frames well encapsulated in RTP. Inside every RTP packet the MPEG2-TS can be found with each flow. Every flow (TV channel) can be selected by selecting the right port 1111, 2222 or 3333. This in the final version will be changed because by agreement the information travels over even ports and the control over odd ports (RTCP).

If we use the tool "Show all Streams" in WireShark we obtain what we have explained:



Detected 3 RTP streams. Choose one for forward and reverse direction for analysis

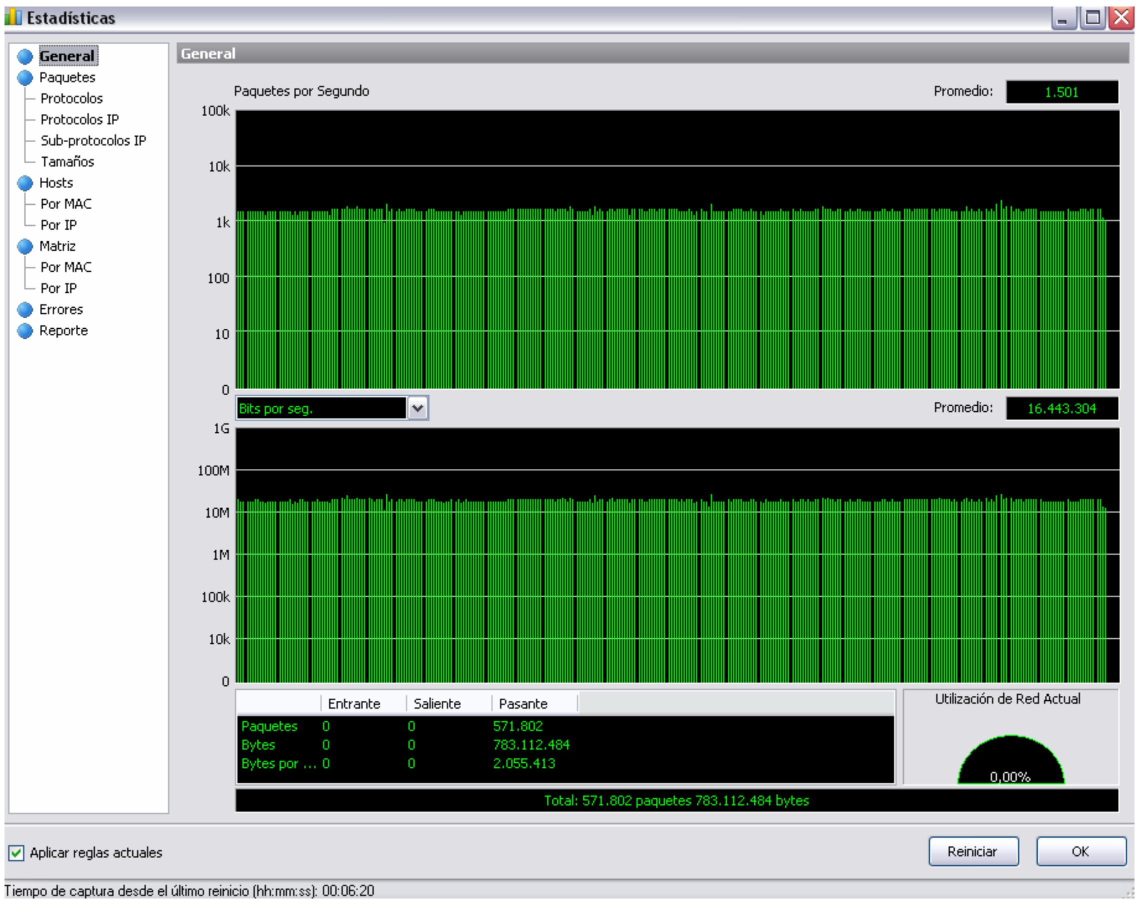
| Src IP addr | Src port | Dest IP addr | Dest port | SSRC | Payload | Packets | Lost | Max Delta (ms) | Max Jitter (ms) | Mean Jitter (ms) | Pb? |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 131.114.53.225 | 32778 | 228.228.228.228 | 1111 | 612767325 | MPEG-II transport streams | 13420 | 0 (0.0%) | 10.93 | 1.59 | 0.73 | |
| 131.114.53.225 | 32780 | 228.228.228.228 | 2222 | 1551140315 | MPEG-II transport streams | 8250 | 0 (0.0%) | 11.59 | 1.72 | 0.82 | |
| 131.114.53.225 | 32781 | 228.228.228.228 | 3333 | 1928248699 | MPEG-II transport streams | 11471 | 0 (0.0%) | 11.15 | 1.72 | 0.74 | |

Forward: 131.114.53.225:32778 -> 228.228.228.228:1111, SSRC=612767325
Reverse: 131.114.53.225:32778 -> 228.228.228.228:1111, SSRC=612767325

**Figure 4.10: The different streams on the 3 Ports in the multicast address.**

Every Port contains a TV channel. These TV channels contain the Video PID, the Audio PID and the Teletext PID or others (if exists).

Changing the analyzer, we study with Open view the total network occupation. We can see it in the graph below that belongs to the net occupation in a logarithmic scale. The first one belongs to packets per second and the second one to the net occupation.



Paquetes por Segundo — Promedio: 1.501

Bits por seg. — Promedio: 16.443.304

| | Entrante | Saliente | Pasante |
|---|---|---|---|
| Paquetes | 0 | 0 | 571.802 |
| Bytes | 0 | 0 | 783.112.484 |
| Bytes por ... | 0 | 0 | 2.055.413 |

Total: 571.802 paquetes 783.112.484 bytes

[✓] Aplicar reglas actuales

Tiempo de captura desde el último reinicio (hh:mm:ss): 00:06:20

**Figure 4.11: Capture with CommView**

The average results come from an average packet side from about 1300 to 1400 bytes. This tells us that if we see the measurements made in section 2 about protocols, we can have an optimal performance because they are closer to the MTU side.

Knowing this, we also observe that the average net occupation is 16.443.304 bps = 16'44 Mbps. it is a little more than we expected because we are transmitting 3 PID's at the same time: 16'44Mbps / 3 = 5'48 Mbps every PID.
We expect a bitrate from 3 to 4Mbps for DVB-T channels encapsulated over MPEG2-TS, so that question us where is the difference between the bitrates.

This can be explained making the measurements with the results in section “2.3.Headers and efficiency” where one keeps in mind the headers efficiency (Ethernet/UDP/RTP/MPEG2-TS): (184x7)/(188x7+58)=93’74~94%

The result of this calculation is:

5’48 x 0’94= 5’1512 Mbps. This value is adequate for the result expected.

**Test 3. Streaming Test of 3 PID's corresponding to the same frequency DVB-T, with 3 clients and the same PID.**

Obviously there aren't changes, because they are Multicast flows. Soon after these tests we observe that VLC interrupts the Multicast flow when does not have any client that demands the file. This is an advantage because we do not inundate the net with unnecessary information all the day, only when somebody uses it. This function comes really implemented from the IGMP protocol. RTCP also does it, but VLC does not implement RTCP now. Anyway VLC implements the fact of finishing the session if there are not users, too.

### 4.3.5. Better Theoretical Protocols (FEC and BER considerations)

In audio and video streaming applications, we usually use certain protocols that help to take advantage of the maximum bandwidth available to prioritize the useful information.
Basing on this our net level will be IP, for its comfort, standardization and easy implementation.

Below IP, we will habitually have Ethernet that implements FEC mechanisms for errors control and correction, but Ethernet is based on a very low BER (Bit Error Rate). We could also have clients using Wi-Fi that has a relatively high BER compared with Ethernet. In this case the quality depending on the SNR signal would be lower. In this case, the level 3 and 4 will start to have problems with the reliability of the sent information.

In the next sections we will study the OSI pile upper layers: Protocols analysis above of IP, on the transport layer. In the audio and video streaming aspect is sensed directly the use of a lighter protocol as is UDP. This protocol as we explain in section "2.2 TCP vs. UDP" basically uses headers less heavy than TCP and a packet orientation. Most of the multimedia applications and streaming applications in the market are based on UDP by these reasons.

### 4.3.6. Encapsulation Analysis of MPEG2-TS

In this section we analyze why in an Ethernet packet usually we have 1370 bytes of data and not 1500 to encapsulate TS packets. The reason is to make the best efficiency and encapsulate the most possible TS packets without fragmenting. We can see also in captures 4-2 and 4-8 of section 4.3.4, that Wireshark recognize each Ethernet packet with payload type MPEG2-TS as a packet with 1370 bytes.

The calculations are simple and we arrive to the fact that why each Ethernet packet contains 7 TS packets. To do the calculations we also use the information in section 2.3 about headers and efficiency:

1370 bytes (Ethernet packet) – 14 bytes (Ethernet Header) – 20 bytes (IP Header) – 8 bytes (UDP header) – 12 bytes (RTP Header) = 1316 bytes.
1316 bytes / 188 bytes (Payload TS (184 bytes) + 4bytes TS header) = 7.

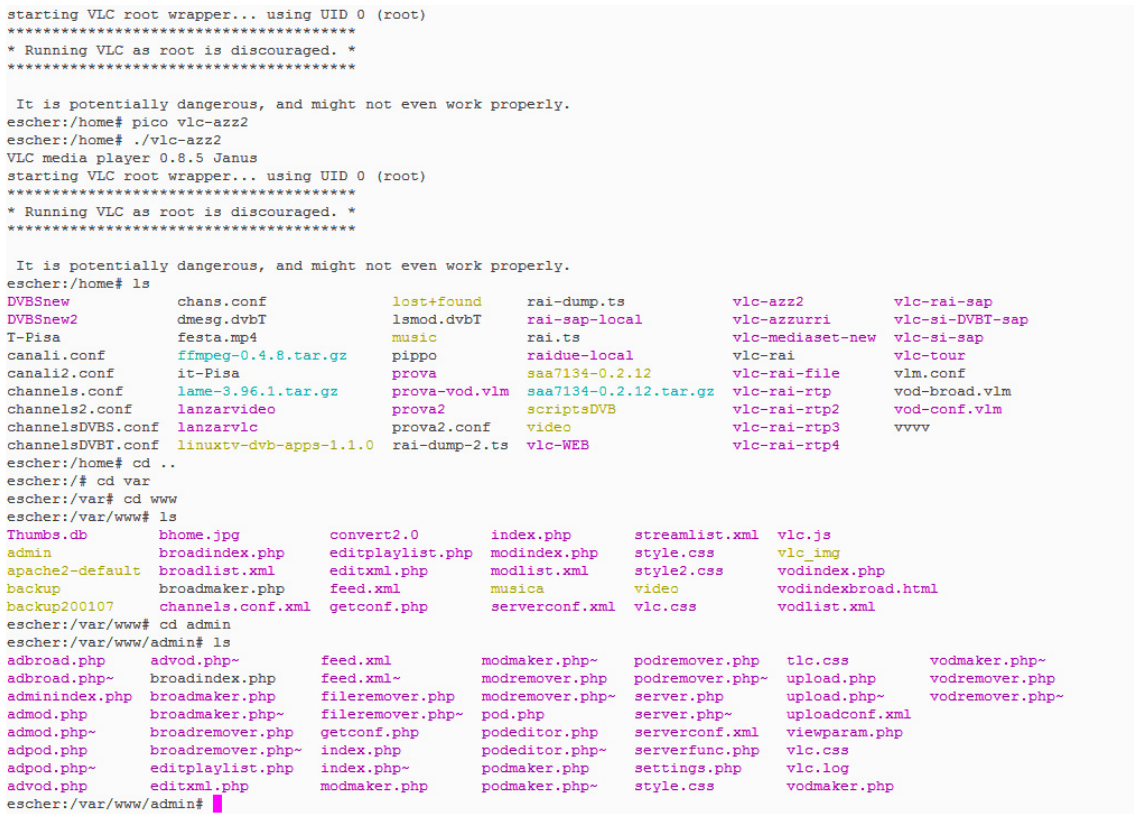
That is the reason why each Ethernet packet contents 7 TS packets identified by a “cc” number as we see in the captures mention before. We can say also that in this case there are not FCS methods included, that is why the Ethernet Header has only 14 bytes and not 18 bytes. If we would use 1500 bytes as the Ethernet MTU the number of TS packets would be 7’69 and we should fragment. As we have explained in section 2.4.4, fragmentation is not a good option with streaming.

## 4.4. Tests with different Scripts and Transport Streams

In this section are treated the different scripts used for the previous sections and other utilized to make different tests. So are studied the TS from the DTT Tuner card and the TS broadcasted in IP received by the clients with the utilization of different scripts.

### 4.4.1. VLC-Script’s

We must say that these tests are not inutile, they serves to make work the TV Streaming part with different parameters. So they are not in Annexes because are active part of our work. The way to access the Linux-Box, as is said in others parts, is through SSH from another computer.



```
starting VLC root wrapper... using UID 0 (root)
****************************************
* Running VLC as root is discouraged. *
****************************************

 It is potentially dangerous, and might not even work properly.
escher:/home# pico vlc-azz2
escher:/home# ./vlc-azz2
VLC media player 0.8.5 Janus
starting VLC root wrapper... using UID 0 (root)
****************************************
* Running VLC as root is discouraged. *
****************************************

 It is potentially dangerous, and might not even work properly.
escher:/home# ls
DVBSnew            chans.conf              lost+found     rai-dump.ts            vlc-azz2          vlc-rai-sap
DVBSnew2           dmesg.dvbT              lsmod.dvbT     rai-sap-local          vlc-azzurri       vlc-si-DVBT-sap
T-Pisa             festa.mp4               music          rai.ts                 vlc-mediaset-new  vlc-si-sap
canali.conf        ffmpeg-0.4.8.tar.gz     pippo          raidue-local           vlc-rai           vlc-tour
canali2.conf       it-Pisa                 prova          saa7134-0.2.12         vlc-rai-file      vlm.conf
channels.conf      lame-3.96.1.tar.gz      prova-vod.vlm  saa7134-0.2.12.tar.gz  vlc-rai-rtp       vod-broad.vlm
channels2.conf     lanzarvideo             prova2         scriptsDVB             vlc-rai-rtp2      vod-conf.vlm
channelsDVBS.conf  lanzarvlc               prova2.conf    video                  vlc-rai-rtp3      vvvv
channelsDVBT.conf  linuxtv-dvb-apps-1.1.0  rai-dump-2.ts  vlc-WEB                vlc-rai-rtp4
escher:/home# cd ..
escher:/# cd var
escher:/var# cd www
escher:/var/www# ls
Thumbs.db        bhome.jpg          convert2.0        index.php       streamlist.xml  vlc.js
admin            broadindex.php     editplaylist.php  modindex.php    style.css       vlc_img
apache2-default  broadlist.xml      editxml.php       modlist.xml     style2.css      vodindex.php
backup           broadmaker.php     feed.xml          musica          video           vodindexbroad.html
backup200107     channels.conf.xml  getconf.php       serverconf.xml  vlc.css         vodlist.xml
escher:/var/www# cd admin
escher:/var/www/admin# ls
adbroad.php     advod.php~         feed.xml          modmaker.php~    podremover.php   tlc.css         vodmaker.php~
adbroad.php~    broadindex.php     feed.xml~         modremover.php   podremover.php~  upload.php      vodremover.php
adminindex.php  broadmaker.php     fileremover.php   modremover.php~  server.php       upload.php~     vodremover.php~
admod.php       broadmaker.php~    fileremover.php~  pod.php          server.php~      uploadconf.xml
admod.php~      broadremover.php   getconf.php       podeditor.php    serverconf.xml   viewparam.php
adpod.php       broadremover.php~  index.php         podeditor.php~   serverfunc.php   vlc.css
adpod.php~      editplaylist.php   index.php~        podmaker.php     settings.php     vlc.log
advod.php       editxml.php        modmaker.php      podmaker.php~    style.css        vodmaker.php
escher:/var/www/admin#
```

**Figure 4.12: Screen capture of accessing Linux-Box (Escher[35]) from an ssh session.**

The VLC scripts are based on two parts, the first part is the input, and the second part the output. Is used the same language as in the VLC telnet interface with a little tag modification.

## VLC-azzurri

```
vlc --intf ncurses --color --ttl 5  dvb: --sout-all --dvb-adapter=0
--dvb-srate=8000  --dvb-budget-mode --dvb-frequency=818000000 --dvb-
bandwidth=8 --dvb-hierarchy=0 --dvb-guard=32 --dvb-transmission=8  -
-sout
'#duplicate{dst=std{access=udp,mux=ts,url=228.228.228.228:1111,sap,n
ame="canale5"},select="program=2"}'
```

This is the already made script used for watch the soccer matches.

## VLC-RAI-SAP

This Script catches all the TS of the **frequency** and selects 3 **PID's**. It broadcast it in mode Multicast to the direction 228.228.228.228 and assigns every PID to a "***:Port***". Also implements a SAP server with the name of the channel.
Important: The output is realized in IP/UDP/MPEG2-TS. Only is sent a TS for every PID. The utilization of this script generates several malformed packets, because of the non utilization of RTP.

[35] "Escher" is the root username of the computer that stores Linux-Box.

```
vlc --intf ncurses --color --ttl 3  dvb: --sout-all --dvb-
adapter=0  --dvb-srate=8000  --dvb-budget-mode --dvb-
frequency=698000000 --dvb-bandwidth=8 --dvb-hierarchy=0 --
dvb-guard=32 --dvb-transmission=8  --sout
'#duplicate{dst=std{access=udp,mux=ts,url=228.228.228.228:111
1,sap,name="RaiUno"},select="program=3401",dst=std{access=udp
,mux=ts,url=228.228.228.228:2222,sap,name="RaiDue"},select="p
rogram=3402",dst=std{access=udp,mux=ts,url=228.228.228.228:33
33,sap,name="RaiTre"},select="program=3403"}'
```

**Input**

**Output**

VLC-RAI-RTP3

This script is the final, utilized on the Streaming webpage. It implements key changes to the performance and standardization of the output flows generated.
The motivation was basically encapsulate over RTP. For this, the VLC language implements various ways to do it. But find one working fine had been hard. The change is marked in **green**.
Looking the code, we see that the script creates a TS for every channel. Then we have three TS in the same Multicast address, but in different ports.

```
vlc --intf ncurses --color --ttl 6  dvb: --sout-all --dvb-adapter=0
--dvb-srate=8000  --dvb-budget-mode --dvb-frequency=698000000 --
dvb-bandwidth=8 --dvb-hierarchy=0 --dvb-guard=32 --dvb-
transmission=8  --sout
'#duplicate{dst=std{access=rtp,mux=ts,dst=228.228.228.228:1111,sap,
name="RaiUno"},select="program=3401",dst=std{access=rtp,mux=ts,dst=
228.228.228.228:2222,sap,name="RaiDue"},select="program=3402",dst=s
td{access=rtp,mux=ts,dst=228.228.228.228:3333,sap,name="RaiTre"},se
lect="program=3403"}'
```

**The encapsulation resulting is like the figure below.**

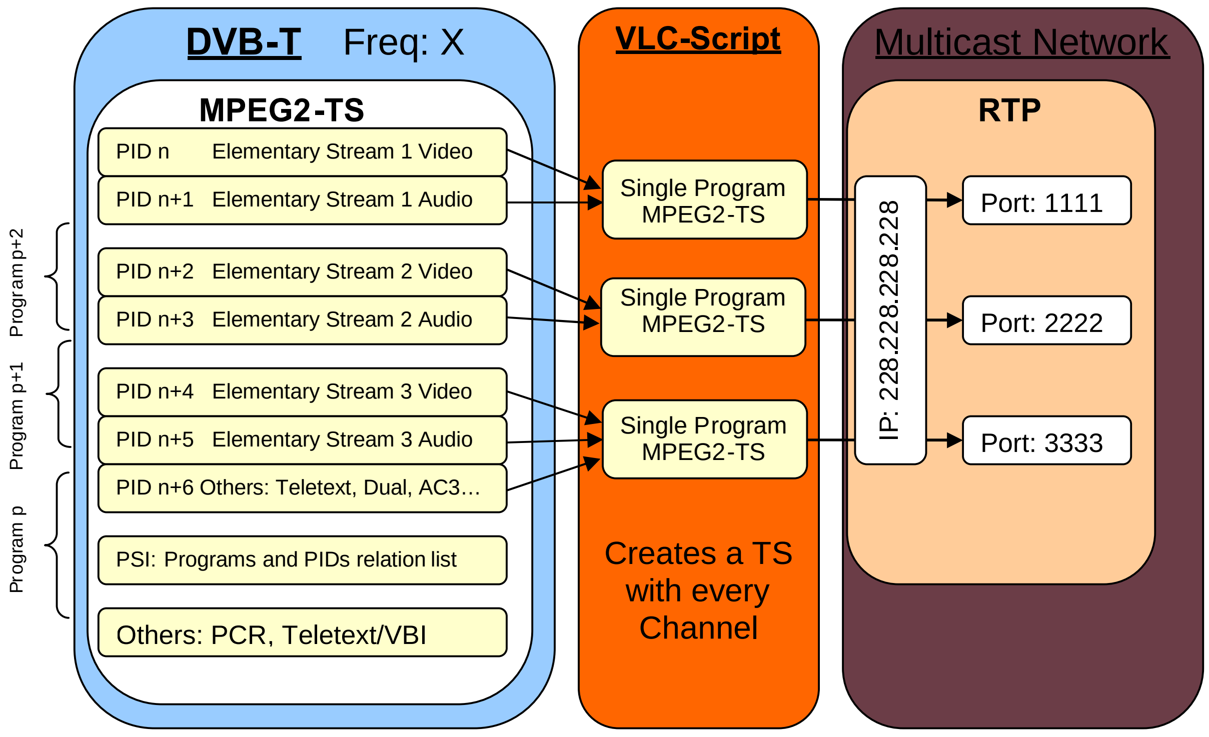

DVB-T Freq: X
MPEG2-TS
PID n Elementary Stream 1 Video
PID n+1 Elementary Stream 1 Audio
PID n+2 Elementary Stream 2 Video
PID n+3 Elementary Stream 2 Audio
PID n+4 Elementary Stream 3 Video
PID n+5 Elementary Stream 3 Audio
PID n+6 Others: Teletext, Dual, AC3…
PSI: Programs and PIDs relation list
Others: PCR, Teletext/VBI
Program p+2
Program p+1
Program p
VLC-Script
Single Program MPEG2-TS
Single Program MPEG2-TS
Single Program MPEG2-TS
Creates a TS with every Channel
Multicast Network
RTP
IP: 228.228.228.228
Port: 1111
Port: 2222
Port: 3333

**Figure 4.13: MPEG2-TS re-encapsulation in IP/RTP by VLC**

VLC-azz3

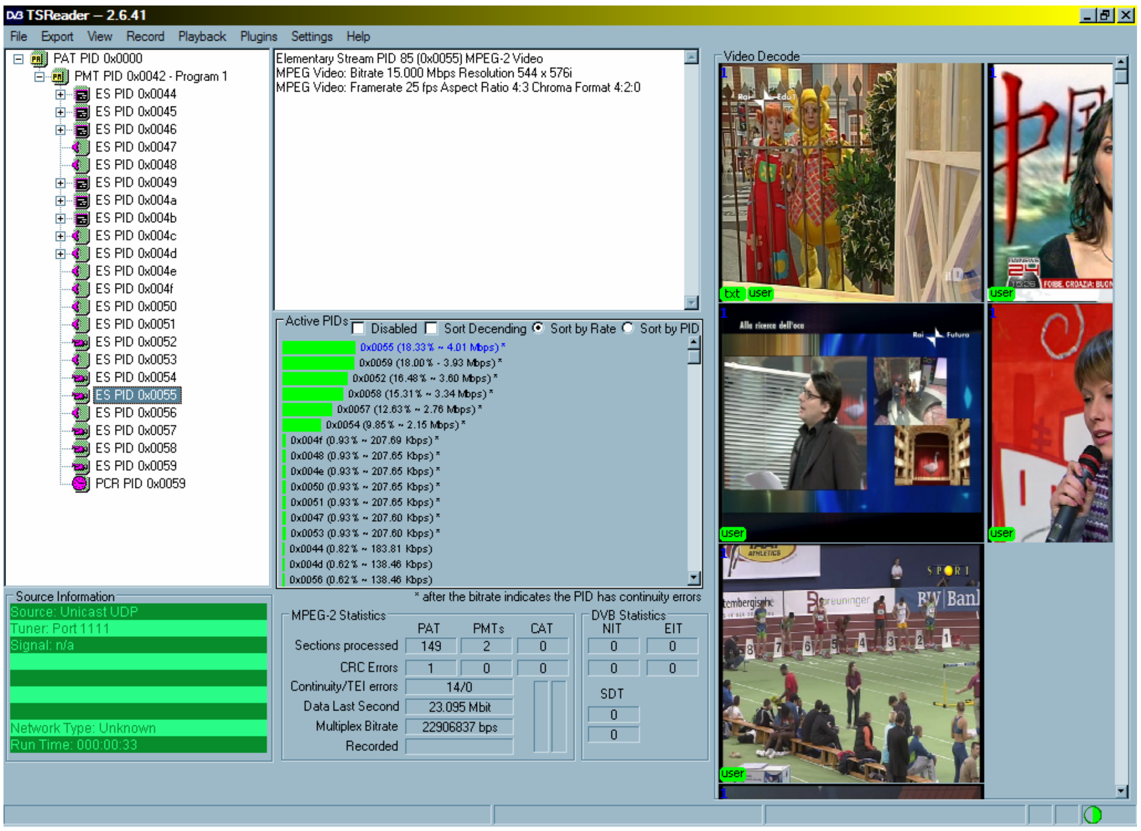
This script is used in the next pages. It serves to stream an entire TS from the DVB-T signal to the IP network over UDP in multicast mode, for analyzing the entire received signal with TSReader.

```
vlc --intf ncurses --color --ttl 6 dvb: --sout-all --dvb-adapter=0
--dvb-srate=8000  --dvb-budget-mode --dvb-frequency=754000000 --dvb-
bandwidth=8 --dvb-hierarchy=0 --dvb-guard=32 --dvb-transmission=8  -
-sout
'#duplicate{dst=std{access=udp,mux=ts,url=131.114.53.128:1111,sap,na
me="canale5"}}'
```

## 4.4.2. MPEG2-TS analysis

With TSReader we have obtained a lot of information about the MPEG2-TS utilized. This sub-section do not uses the Linux-Box software. Here we are only analyzing the streams received by de DVB-T tuner.

First of all, we created a VLC script (VLC-azz3) to re-send an entire MPEG2-TS received from the DVB-T card to our IP direction in Unicast mode, in this form, we can study the whole source signal. We muts select a port to stream and later listen the TS. Now are shown three captures of the inside of the TS received by the DVB-T tuner. Every capture corresponds to a different frequency:



```
PAT PID 0x0000
  PMT PID 0x0042 - Program 1
    ES PID 0x0044
    ES PID 0x0045
    ES PID 0x0046
    ES PID 0x0047
    ES PID 0x0048
    ES PID 0x0049
    ES PID 0x004a
    ES PID 0x004b
    ES PID 0x004c
    ES PID 0x004d
    ES PID 0x004e
    ES PID 0x004f
    ES PID 0x0050
    ES PID 0x0051
    ES PID 0x0052
    ES PID 0x0053
    ES PID 0x0054
    ES PID 0x0055
    ES PID 0x0056
    ES PID 0x0057
    ES PID 0x0058
    ES PID 0x0059
    PCR PID 0x0059
```

```
Elementary Stream PID 85 (0x0055) MPEG-2 Video
MPEG Video: Bitrate 15.000 Mbps Resolution 544 x 576i
MPEG Video: Framerate 25 fps Aspect Ratio 4:3 Chroma Format 4:2:0
```

Active PIDs ☐ Disabled ☐ Sort Decending [✓ Sort by Rate] Sort by PID

```
0x0055 (18.33% ~ 4.01 Mbps) *
0x0059 (18.00% - 3.93 Mbps) *
0x0052 (16.48% ~ 3.60 Mbps) *
0x0058 (15.31% ~ 3.34 Mbps) *
0x0057 (12.63% ~ 2.76 Mbps) *
0x0054 (9.85% ~ 2.15 Mbps) *
0x004f (0.93% ~ 207.69 Kbps) *
0x0048 (0.93% ~ 207.65 Kbps) *
0x004e (0.93% ~ 207.65 Kbps) *
0x0050 (0.93% ~ 207.65 Kbps) *
0x0051 (0.93% ~ 207.65 Kbps) *
0x0047 (0.93% ~ 207.60 Kbps) *
0x0053 (0.93% ~ 207.60 Kbps) *
0x0044 (0.82% ~ 183.81 Kbps)
0x004d (0.62% ~ 138.46 Kbps)
0x0056 (0.62% ~ 138.46 Kbps)
```

* after the bitrate indicates the PID has continuity errors

Source Information

```
Source: Unicast UDP
Tuner: Port 1111
Signal: n/a
Network Type: Unknown
Run Time: 000:00:33
```

MPEG-2 Statistics

| | PAT | PMTs | CAT |
|---|---|---|---|
| Sections processed | 149 | 2 | 0 |
| CRC Errors | 1 | 0 | 0 |
| Continuity/TEI errors | 14/0 | | |
| Data Last Second | 23.095 Mbit | | |
| Multiplex Bitrate | 22906837 bps | | |
| Recorded | | | |

DVB Statistics

| NIT | EIT |
|---|---|
| 0 | 0 |
| 0 | 0 |
| SDT | |
| 0 | |
| 0 | |

**Figure 4.14: TSreader capture in the Freq: 654 MHz of a complete TS, corresponding to RAI channels.**

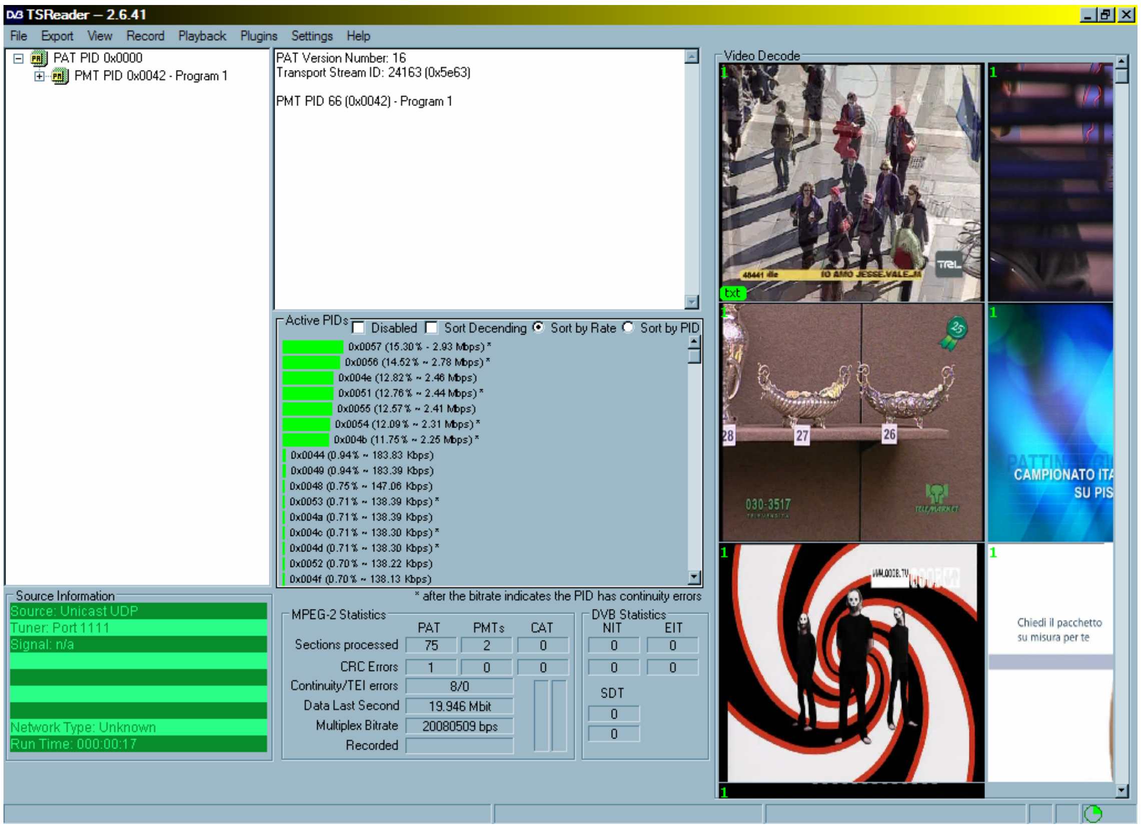
This capture shows the average bitrate of every PID inside the Transport Stream. We can see four types of PID's:

- Video
- Audio
- Teletext
- PCR

Video, Audio and Teletext are Elementary Streams. Te way they are related is in the PSI, specially in the PMT.

The hierarchy starts with PAT (Program Association Table) always with PID=0x00. This contains a PMT (Program Map Tables) for every program. The PMT relates the different ES to crate a TV channel.

In this case the Program 1 (PMT 0x0042) contains all TV channels in an only Program. Another option is to find a specific Program and PMT for each TV channel.



```
PAT PID 0x0000
  PMT PID 0x0042 - Program 1

PAT Version Number: 16
Transport Stream ID: 24163 (0x5e63)

PMT PID 66 (0x0042) - Program 1

Active PIDs  [ ] Disabled  [ ] Sort Decending  (•) Sort by Rate  ( ) Sort by PID
0x0057 (15.30% - 2.93 Mbps)*
0x0056 (14.52% ~ 2.78 Mbps)*
0x004e (12.82% ~ 2.46 Mbps)
0x0051 (12.76% ~ 2.44 Mbps)*
0x0055 (12.57% ~ 2.41 Mbps)
0x0054 (12.09% ~ 2.31 Mbps)*
0x004b (11.75% ~ 2.25 Mbps)*
0x0044 (0.94% ~ 183.83 Kbps)
0x0049 (0.94% ~ 183.39 Kbps)
0x0048 (0.75% ~ 147.06 Kbps)
0x0053 (0.71% ~ 138.39 Kbps)*
0x004a (0.71% ~ 138.39 Kbps)
0x004c (0.71% ~ 138.30 Kbps)*
0x004d (0.71% ~ 138.30 Kbps)*
0x0052 (0.70% ~ 138.22 Kbps)
0x004f (0.70% ~ 138.13 Kbps)
* after the bitrate indicates the PID has continuity errors

Source Information
Source: Unicast UDP
Tuner: Port 1111
Signal: n/a
Network Type: Unknown
Run Time: 000:00:17

MPEG-2 Statistics
                        PAT    PMTs   CAT
Sections processed      75     2      0
CRC Errors              1      0      0
Continuity/TEI errors   8/0
Data Last Second        19.946 Mbit
Multiplex Bitrate       20080509 bps
Recorded

DVB Statistics
NIT    EIT
0      0
0      0
SDT
0
0
```

**Figure 4.15: TSreader capture in the Freq: 818 MHz of a complete TS, corresponding to various channels.**

For this two captures, every video Elementary Stream has a similar Rate to the others. From 2 to 3 or even 4 Mbps every ES.

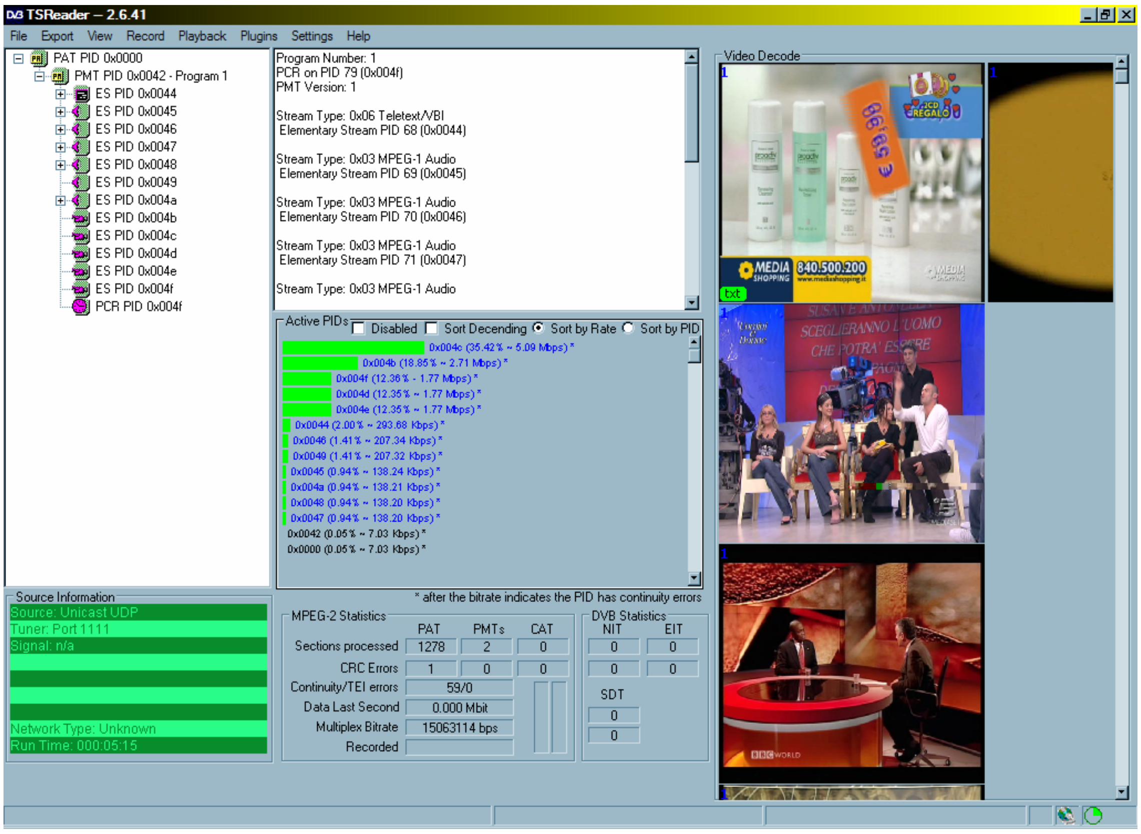
After analyze this two and other frequencies, we have find different video resolutions and Bitrates. The values found are:

```
Resolutions: 720 x 576i , 704 x 576i, 544x576i, 480x576i and 352 x
576i
Header Bitrate Codification (not real bitrate): 15.000 Mbps, 10.000
Mbps, 2.800Mbps
Formats: Framerate 25 fps
         Aspect Ratio 4:3
         Chroma Format 4:2:0
```

Respect to the audio the found values change less:

```
Stream Type: MPEG-1 Audio
MPEG1 Audio: Bitrate 192 Kbps Sample Rate 48 KHz, Bitrate 128 Kbps
Sample Rate 48 KHz
MPEG1 Audio: Layer II Mode Stereo
```

In this capture we have found a peculiar Video ES distribution. It indicates that a channel have very more bit rate and quality than the others. The motives are unknown for us. We can thing in, for example, that the other channels are from news or something like this. In this case the channel with more quality is "Canale 5 from Mediaset ".


TSReader – 2.6.41
File Export View Record Playback Plugins Settings Help
Video Decode
txt
MEDIA SHOPPING 840.500.200
BBC WORLD

```
PAT PID 0x0000
  PMT PID 0x0042 - Program 1
    ES PID 0x0044
    ES PID 0x0045
    ES PID 0x0046
    ES PID 0x0047
    ES PID 0x0048
    ES PID 0x0049
    ES PID 0x004a
    ES PID 0x004b
    ES PID 0x004c
    ES PID 0x004d
    ES PID 0x004e
    ES PID 0x004f
    PCR PID 0x004f
```

```
Program Number: 1
PCR on PID 79 (0x004f)
PMT Version: 1

Stream Type: 0x06 Teletext/VBI
 Elementary Stream PID 68 (0x0044)

Stream Type: 0x03 MPEG-1 Audio
 Elementary Stream PID 69 (0x0045)

Stream Type: 0x03 MPEG-1 Audio
 Elementary Stream PID 70 (0x0046)

Stream Type: 0x03 MPEG-1 Audio
 Elementary Stream PID 71 (0x0047)

Stream Type: 0x03 MPEG-1 Audio
```

Active PIDs ☐ Disabled ☐ Sort Decending [✓ Sort by Rate] Sort by PID

```
0x004c (35.42 % ~ 5.09 Mbps) *
0x004b (18.85 % ~ 2.71 Mbps) *
0x004f (12.36 % - 1.77 Mbps) *
0x004d (12.35 % ~ 1.77 Mbps) *
0x004e (12.35 % ~ 1.77 Mbps) *
0x0044 (2.00 % ~ 293.68 Kbps) *
0x0046 (1.41 % ~ 207.34 Kbps) *
0x0049 (1.41 % ~ 207.32 Kbps) *
0x0045 (0.94% ~ 138.24 Kbps) *
0x004a (0.94% ~ 138.21 Kbps) *
0x0048 (0.94% ~ 138.20 Kbps) *
0x0047 (0.94% ~ 138.20 Kbps) *
0x0042 (0.05 % ~ 7.03 Kbps) *
0x0000 (0.05 % ~ 7.03 Kbps) *
```

* after the bitrate indicates the PID has continuity errors

Source Information

```
Source: Unicast UDP
Tuner: Port 1111
Signal: n/a
Network Type: Unknown
Run Time: 000:05:15
```

MPEG-2 Statistics

| | PAT | PMTs | CAT |
|---|---|---|---|
| Sections processed | 1278 | 2 | 0 |
| CRC Errors | 1 | 0 | 0 |
| Continuity/TEI errors | 59/0 | | |
| Data Last Second | 0.000 Mbit | | |
| Multiplex Bitrate | 15063114 bps | | |
| Recorded | | | |

DVB Statistics

| NIT | EIT |
|---|---|
| 0 | 0 |
| 0 | 0 |
| SDT | |
| 0 | |
| 0 | |

**Figure 4.16: TSReader capture of an entire TS. Freq: 754 MHz**

Here is showed the information obtained of this TS.
Video:

| PID | Resolution | Header Bitrate (not real bitrate) | Compression |
|---|---|---|---|
| 0x004b | 544 x 576i | 15.000 Mbps | MPEG-2 |
| 0x004c | 704 x 576i | 15.000 Mbps | MPEG-2 |
| 0x004d | 544 x 576i | 10.000 Mbps | MPEG-2 |
| 0x004e | 352 x 576i | 10.000 Mbps | MPEG-2 |
| 0x004f | 544 x 576i | 10.000 Mbps | MPEG-2 |

Audio:

| PID | Bitrate | Sample rate | Compression |
|---|---|---|---|
| 0x004a | 192 Kbps | 48 KHz | MPEG1 Audio: Layer II Mode Stereo |
| 0x0049 | 128 Kbps | 48 KHz | MPEG1 Audio: Layer II Mode Stereo |
| 0x0047 | 128 Kbps | 48 KHz | MPEG1 Audio: Layer II Mode Stereo |

Others:

| PID | Type |
|---|---|
| 0x0044 | Teletext/VBI |
| 0x0045 | Teletext/VBI |

This is an example of reported information about PID's by TSReader:

```
Stream Type: 0x02 MPEG-2 Video PID 78 (0x004e)
MPEG Video: Bitrate 10.000 Mbps Resolution 544 x 576i
MPEG Video: Framerate 25 fps Aspect Ratio 4:3 Chroma Format
4:2:0

0x004e (12.37% ~ 1.77 Mbps) *7/0

Stream Type: 0x02 MPEG-2 Video PID 79 (0x004f)
MPEG Video: Bitrate 10.000 Mbps Resolution 352 x 576i
MPEG Video: Framerate 25 fps Aspect Ratio 4:3 Chroma Format
4:2:0

0x004f (12.35% ~ 1.77 Mbps) *4/0
```

On the other hand, we see that video ES's with the same encoded Bitrate and equal transmission rate has different resolutions. This can be due to several reasons, the more logics interpretations are:

- A possible better or more optimal coding of the ES with more resolution.
- That the spare pixels are not used, for example for aspect ratio topics, and are in black.

We can see that the studied TS, and others in Italia are not build with the most common standard format, creating a PMT for every TV channel. We also see differences in the quality between different channels and its respective bitrate.
The fact of find more Audio ES than Video ES if because one channel streams in Dual.

Finally, we can say that DVB-T is a used technology but the dedication of the TV Channels to its well encapsulation is far to be correct. We thing it will be better when the analogical TV leave streaming. Service like complete PCR, HDTV, surround sound, interactive service (with MHP specific receivers) and others can bring to the client a best experience taking advantage of the whole DVB-T technology at a low cost for the TV companies, because the DVB-T technology is already implemented now and working.

# 5. Conclusions

The Linux-Box project is a work in process and even lacks developments and improvements to be completely operative for the final user. Anyway we believe that we have contributed in a satisfactory way in its development in three fundamental parts:

- Definition of what it is, its function and what should be implemented, as well as its operation policies.
- Work and collaboration in several modules for their fine working.
- Analysis of the created network flows.

Our applicative part has been based on the Linux-Box and the study of the streams created by VLC. It belongs to the sections “3.Linux-Box” and “4.Developed Part”:

1. Starting with the developed software, we have created a useful application following the steps required for a good development. The result is a module of the global project. The distributed character of its design allows us to cover a big project, letting many people get involved in the process.

Although our Linux knowledge was reduced: (compiling using “gcc” and configuring interfaces), we think that at the moment we can get by in Unix/Linux and acquired a good base (based on self- learning) that naturally will go on.

2. The web section dedicated to the TV Broadcasting was taken at the middle of the work. We studied the protocols and some other aspects, and since the general project did not advance, we decided to work in its development too, in the sense of making it works. At the end it has been working satisfactorily using JavaScript in the same PHP code.

3. Regarding the network tests, at the beginning they were just some tests about activity, and encapsulation. Later in practice every time it earned importance. At the end we have analyzed the frames with WireShark, the net activity with CommView, and the streams with TSReader. To make this section we have used the scripts of the following section, and then they were overlapped.

- The conclusion we obtain from these tests is that Linux-Box generates the expected traffics if VLC is configured correctly.
- We also see that the flows created can cohabit with the normal traffic of a Local Area Network.
- A problem we considered was the implementation of Firewalls that forced us to use methods explained in the section "3.6.5 Firewalls" to pass through them.

4. The script tests are used in the previous and other sections to configure the Linux-Box. This section links the previous section with the future lines, and means that the scripts proved overlap between the carried out tests and open the door for future implementations.

In the Linux-Box aspect we have seen that it is a good idea, with an appropriated implementation and a real possible commercial success. True alternatives exist for a Multimedia-Centre with a low cost and a possible more popular use. There also exists a great community of users behind, starting with the open-source software used, and the great market where Linux-Box can be introduced.

Its access by Web for the final user is a good idea, which allows him to have a quick, governable and transparent access.

In the hardware aspect, we have seen that the Linux-Box with its actual components installed does not have problems to execute different processes at a memory or CPU level[36] streaming the information explained in section 3.5: Streaming 3 TV programs and a medium of 2 or 3 VoD clients. Otherwise, we see that it is a first version that is already obsolete. It can be used to work in the laboratory, but in future hardware implementations the increment of capacity should be extended according with the market. This is more reasonable in a server that will offer apart from several services, the video streaming capacity.

On the other hand, future versions should solve the local oscillators' problem in DVB-T. We observe two possible solutions, discussed many times with the tutor. The first one is to include several DVB-T cards to offer a complete frequencies service with simultaneous TV. This would have a cost in space and infrastructure but it will not affect in the economical aspect.

The second solution would be to find out using only one card and diverse oscillators. This could be difficult, expensive or impossible. But it is an entire university department, so we think that is possible for a future development.

In the treated protocols aspect we see that as RTP as IGMP are consecrated protocols that make their function solving the deficiencies as far as possible, of other inferior protocols. On the other hand, regarding the video transmission MPEG formats it has a quality guarantee recognised. As way of transport, MPEG2-TS is ideal for several multiplexed flows (Elementary Streams). MPEG-4 seeks to be the successor implementing dozens of improvements and an orientation to new interactive environments, besides the support for portable devices. It is also explained in the Annex.

Regarding to VoD, we have seen that there is written a lot about it, but nothing is standardized for further implementation. Many applications talks about VoD but at the end everything ends in private technologies as Real-Networks, Quick-Time, Windows Media… any global protocol is defined. For the media controls (Play, pause…) it seems that RTSP is the most standardized but not

[36] As is explained in the section 3.4.

indispensable although exists Flash implementations that make the same function.

The intentions of this work in its beginning were ambitious. We wanted to have all the Linux-Box at all, to implement authentication with a Radius Server, in order to make a complete modularity study regarding the maximum users and everything with both technologies DVB-S and DVB-T. These have been the parts that have not been correctly implemented. The security, could not be implemented simply because Linux-Box did not worked fine until the end of the project. For this reason the aspirations of the project had to be reconsidered at the middle of the project.

On the other hand the aspect of treating as much terrestrial television as satellite, although there is not a big difference in its configuration; we have had not enough time for configuring and providing its operation. For this reason, all tests have been made with DVB-T; DVB-S goes to the Future Lines section.

In the personal part, we can conclude that we have deepened in technologies and protocols that we only had heard speak of. We have also learned how DVB works and how the signals are transported by MPEG2-TS. On the other hand, we have acquired a background in the OS Linux/Debian and have made to value many knowledge acquired during the career.

In our case all this is made because of the development of a project on a half-long term that is Linux-Box. In general the elaboration of the work has been a positive and satisfactory experience for us.

## 5.1. Innovation, improving and Future Lines

### 5.1.1. DVB-S Study

The next part that should be studied to continue the project is DVB-S, since we have not studied its development. The basic changes are the frequencies configuration and signal parameters. It should also take as reference the fact that convert2.1 already implements compatibility with DVB-S.

### 5.1.2. Streaming over IPv6

An interesting section for further research would be the Linux/Debian implementation and the corresponding changes to the whole project in order to make it work in the same way as with IPv4. Also, a generated flow analysis should be done for the different changes that implements IPv6.

### 5.1.3. Gigabit Ethernet

The use of Gigabit Ethernet could be interesting for environments with a lot of traffic. If in future it is wanted to implement the Linux-Box in Internet would be enough a potent connection of these characteristics to assume the traffic demanded.

### 5.1.4. POD-casting (discovering protocol)

Pod-casting is a discovery protocol to discover sources for I-POD. It is a technology growing that takes advantage of its actual commercial pull.

### 5.1.5. Transcode the VoD streams

If we use the VLC transcode command we can increase up to where we want the maximum number of VoD users, only taking care of the final image quality. With this it would be unnecessary to change to a new hardware configuration with more capacity, in environments with a lot of VoD demand. Unfortunately, at this time, transcoding in Real-Time with VLC is slow and has a poor quality. Maybe in future versions of VLC this feature is improved. Another solution is to use a specific program for the VoD (for example live555).

### 5.1.6. Streaming over the net

As it has already been commented before, if we play with Transcoding to adapt the flows to the global net, it could be possible to make a VoD and TV server over Internet.

### 5.1.7. Security

The security implementation is basic for the commercial applications. It should be necessary an Apache server, and a web page application with login and password. This part was initially outlined for this project developing a Radius server, which exists in the university department, but at the end it won't be implemented.

### 5.1.8. Economical Aspects

At the beginning the Linux-Box has been designed to be used in local nets, it has been mentioned previously that it could be a good solution as a Media Centre at home, so this possibility should not be rejected. In this case it is convenient a market study on their possible launching, in order to know the impact and the possible economic profit. It would be also necessary to keep in mind the inclusion of new hardware according to the new technologies like HDTV or Blu-Ray.

### 5.1.9. Environmental Aspects

Although it has not been made a widely study on the environment effects derived of using this device's use in this project, it is possible to make the following interesting considerations:

The Linux-Box use can collaborate to use less TVs and take profit of existing monitors to avoid the massive buy of TVs (every time more cheap) but with a difficult recycling process.

For their emission characteristics in packet networks, it can collaborate to re-use and take benefit of the existing ones.

It is important to understand that for the considerations taken in this work with the Open-Source software used and the use of IP communications services, does not affect directly to the natural environment.

## 5.2. Ethical Aspects

### 5.2.1. Social Ambit

The software used in the Linux-Box, applications an Operating Systems are open-source, this implies a less economic spend in licenses and assure that behind the software there is a community that will continue developing the product and improving it. This favours the increasing of free software an as alternative to big company firms of software like Microsoft, as well as the intention of breaking the monopoly that has had till now. It is a hard task indeed but projects as Linux-Box help to change this fact. The rupture of the monopoly in software applications will also allow improvements and advances in new programs and technologies, doing it faster and with more quality.

### 5.2.2. Author Rights, Royalties and Copyrights

All the information emitted by VLC, either AoD, VoD or TV Broadcasting will by the administrator responsibility because everything emitted by VLC is according to the responsibility of each one. The administrator will keep in mind the legal mechanisms about reproduction protection royalties, copyrights, etc…to know if he has rights to do the emissions.

In a first moment the Linux-Box has been made with educational finalities, but if in a moment one would like to commercialize the product, should be made studies about re-streaming media and the TV channels transmission. In this case would be also interesting trying to include some type of parental control to protect young users from a possible mature content.

About the program “convert” created in this thesis, we consider it open-source, so anyone can copy, distribute or to modify it for its own. It will also be published in diverse forums related with DVB, VLC and Linux.

# 6. Bibliography

## For the section Ubuntu 6.06 LTS

[1] About Linux: http://www.linux.org/info

[2] Linux Distributions: http://www.linux.org/dist/

[3] Red Hat: http://www.redhat.com/about/whyredhat/

[4] Suse: http://www.novell.com/products/opensuse

[5] Fedora: http://fedora.redhat.com/About/

[6] Mandrake: http://www.mandriva.com/en/community/resources/about_mandriva_linux

[7] Ubuntu: http://www.ubuntu.com/

[8] Ubuntu 6.06 LTS: http://www.ubuntu.com/download/releasenotes/606

[9] Gnome: http://www.gnome.org/about/

[10] KDE: http://www.kde.org/whatiskde/

[11] 3D-Desktop: http://Desk3d.sourceforge.net

## For the section Multimedia Network Protocols

[12] IP, TCP vs. UDP and headers efficiency: http://www.ieee.org/web/publications/home/index.html

[13] RFCs related: http://www.ietf.org

a. RTP: RFC 3550 “RTP: A Transport Protocol for Real-Time Applications”
b. RTSP: RFC 2326 “Real Time Streaming Protocol (RTSP)”
c. SAP: RFC 2974 “Session Announcement Protocol”
d. SDP: RFC 2327 “SDP: Session Description Protocol”
e. IGMP: RFC 2236 “Internet Group Management Protocol, Version 2”

[14] Chunlei Liu, “Multimedia Over IP: RSVP, RTP, RTCP, RTSP” : http://www.cis.ohio-state.edu/~jain/cis788-97/ip_multimedia/index.htm

[15] How to Enable Real-Time Streaming Protocol (RTSP) Traverse Network Adress Translators (NAT) and Interact with Firewalls, Zeng Westerlund (10-2005). http://tools.ietf.org/pdf/draft-ietf-mmusic-rtsp-nat-04.pdf

[16] Leggio Simone, “Streaming Media over the Internet with the Real Time treaming Protocol”
http://www.cs.helsinki.fi/u/jmanner/Courses/seminar_papers/rtsp.pdf

[17] Testing Digital Video, MPEG2 Testing:
http://www.iec.org/online/tutorials/test_dv/topic01.html
Dr Gorry Fairhurst, MPEG-2 Transmission:
http://erg.abdn.ac.uk/research/future-net/digital-video/mpeg2-trans.html

[18] DVB: http://www.dvb.org/about_dvb/history/index.html

[19] Streaming Servers and firewalls:
http://www.soundscreen.com/streaming/firewall.html

[20] Software
a. WireShark: http://www.wireshark.org/about.html

**For the LINUX-BOX section**

[21] VLC: http://www.videolan.org

[22] How to stream VLC: [http://download.videolan.org/doc/streaming-howto/en/streaming-howto-en.pdf]

[23] DVBStream: Can be found in any Linux/Debian repository.

[24] Live555: http://www.live555.com/

**For the Section Applicative Part**

[25] TS reader: http://www.coolstf.com/tsreader/

[26] VideoLan Forums: http://forum.videolan.org/

**For the Section Annexes:**

[27] RSVP: RFC 2326 “ Resource Reservation Protocol”

[28] MPEG-4:http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.html

[29] RTCP: RFC 3550 “RTP: A Transport Protocol for Real-Time Applications”

[30] Payload Types: RFC 3551 “RTP Profile for Audio and Video Conferences with Minimal Control”

[31] MPEG1/MPEG2: “RTP Payload Format for MPEG1/MPEG2 Video”

[32] H261: RFC 2032 “RTP Payload Format for H.261 Video Streams”

## Others

[26] Panagiotis Papadimitriou, Sofia Tsekeridou, Vassilis Tsaoussidis, “Multimedia Streaming over the Internet”

[27] Danyan Chen, Anjali Agarwal, Michel kandoch, Dr Ahmed K.Elhakem, “Multicast for real-time streaming over Ethernet”

# 7. Glossary of terms

## A

**Application Layer**: The seventh layer in the OSI, defines how applications access to network services.
**AoD:** Audio on Demand.
**AVI**: Audio/Video Interleaved

## B

**Bandwith:** The amount of data that can be transmitted in a fixed time. Usually expressed in bps.
**Bit:** The smallest unit of measure of data in a computer. A bit has a binary value, 0 or 1.
**Bitrate:** The speed at which data travels from one place to another on a network
**Bps:** Bits per second
**Buffer:** Space allocated on a system's RAM where data is stored temporarily until it is transferred to another part of the system.
**Byte:** A common measure data that consist of 8 bits.

## C

**Capture Card:** A device that is used to digitize analogue audio or video and write to a file or write digital audio or video to a file.
**Client:** A software application that receives data from a server.
**CRC:** (Cyclic Redundancy Check)

## D

**DF:** Do not Fragment
**DTT:** Digital Terrestrial Television
**DVB-S:** Digital Video Broadcasting Satellite
**DVB-T:** Digital Video Broadcasting Terrestrial

## E

**Ethernet :** A LAN used to connect devices within a single building or campus at speeds up to 1 Gbps.

## F

**FPS:** Frames per Second

## G

**GSM:** Global System for Mobile Communications

## H

**HTTP:** Hypertext Transfer Protocol
**HTML:** Hypertext Markup Language

## I

**IETF:** Internet Engineering Task Force
**IGMP:** Internet Group Management Protocol
**IP:** Internet Protocol
**ISO:** International Organization for

## K

**Kernel:** Is the core piece of an operating system.

## L

**LAN:** Local Area Network

## M

**MIME:** Multipurpose Internet Mail Extension
**MMS**: Microsoft Media Server
**MPEG:** Moving Picture Experts Group
**MPEG2-TS:**MPEG-Transport Stream
**MTU:** Maximum Transmission Unit
**Multimedia:** The integrated presentation of text, graphics, audio, video and animation.

## N

**NAT:** Network Address Translation
**Network:** Two or more computers connected to each other so they can share resources

## O

**On Demand:** Information that is available when you want it
**OS:** Operating System.
**OSI:** Open Systems Interface

## P

**Packet:** A chunk of data organized in a block for transmission over an IP network.
**Port:** A connection to a computer to enable other devices to interface with the computer.
**Protocol:** A set of rules that enable computers to connect to one another

## Q

**QoS:** Quality of Service

## R

**RAM:** Random Access Memory
**RSVP:** ReSource reserVation Protocol
**RTP**: Real-time Transport Protocol
**RTCP**: Real-Time Control Protocol
**RTSP:** Real-Time Streaming Protocol

## S

**SAP:** Session Announcement Protocol
**Script:** Plain text file used as configuration parameters by a specific program.
**Server:** A software application that sends requested data over a network.
**SDP:** Session Description Protocol
**SSRC:** Synchronization SouRCe
**STB:** Set-Top Box
**Streaming media:** Is the simultaneous transfer of digital media (video, voice and data) so that it is received as a continuous real-time stream.

## T

**TCP:** Transmission Control Protocol
**Transcoding:** The conversion of one digital file format to another digital file format

## U

**UDP** User Datagram Protocol
**URL** Uniform Resource Locator

## V

**VLC:** Video LAN Client
**VoD:** Video on Demand.

## X

**XML:** Extensible Markup Language

# 8. Timing

September

12-Arribal to Pisa
20- First contact with the Italian tutors in the IET (UNIPI)

October

1- Title of the Thesis
1- Install Linux and use it. Study Linux-Box working mode
2- Implement “Convert”

November

15- Possible Applications of the Thesis: Security and Modularity.

December

1- New section: Traffic Analysis. Research about traffic streaming and search ways to analyse it on the Linux-Box.

15- Released “Convert2.0”
27- Star writing the thesis. It loses to do security, modularity and analyze what kind of traffic is better to use.

January

22- Star test with Linux-box. The main part of the other working group does not work. We decide to do the broadcasting webpage
23- Modifications on convert

February

14- Presentation in Pisa
15- Finishes Erasmus in Pisa

March-April
Thesis reconsiderations with the Spanish tutor and format adjustments to the EPSC specific documents format.

May
Expected presentation time.

# 9. Annexes

## 9.1 Surce Codes

### 9.1.1. Convert.c Code

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#define MAX 20

void main (int argc,char *argv[])
{
            int error;
            char file[30], type[5], signal[5];

            printf("\n\n\t+---------------------------------+\n\t|\tWelcome Convert 2.2\t  |\n\t+--------------------
            -------------+\n\n");
            if (argc!=4)
            {
                   printf("Missing Arguments\nUsing: convert [source_file] [.destination_type] [Terrestre or
                   Satelitale]\nExample: convert file.dump .xml T\n");
            }

            else
            {
                        strcpy (file,argv[1]);
                        strcpy (type,argv[2]);
                        strcpy (signal,argv[3]);

                        if ((strcmp(signal,"T")!=0)&&(strcmp(signal,"S")!=0))
                        {
                                    printf ("Last Parameter wrong,must be 'S' o 'T'\nTry Again...\n\n");
                                    exit;
                        }
                        else
                        {
                                    if(files (file,type,signal)==0)
                                    {
                                               printf("\n\n\t+---------------------------------+\n\t|\tConvert
                                               2.2\t\t  |\n\t|\tTutto Bene Ciao\t  |\n\t+--------------------------
                                               -------+\n\n");
                                    }
                                    else
                                    {
                                               printf("\n\n\t+---------------------------------+\n\t|\t   Convert
                                               2.2\t\t  |\n\t|\t      ERROR\t\t  |\n\t+----------------------------
                                               -----+\n\n");
                                    }
                        }
            }
}

int files(char f[], char t[], char S[])
{
            int max=190;
            int b,c,n,i;
            char s[max];
            char filename[30];
            char nom[40];
            char freq[30];
            char pid[30];
            int available;
            int tamanyoreal=0;
            int cont=0,numchan=0;
            char comilla;

            comilla=(char)39;
            char f_copia[30];//*destino; f_copia sera el arxivo de destino
            FILE *f1;
            if ((f1 = fopen(f,"r")) == NULL ) //el dump
            {
                        printf("Can't open Source File: %s\n",f);
                        return(1);
            }
            //strcpy (t,destino);
            //destino=".xml";
            strcpy (f_copia,f);//copia f en f copia para no sobreescribirlo
            //sprintf (f_copia,'.';
            strcat (f_copia,t);//anyade t a f_copia
            //printf("destino= %s\n",destino);
            printf("Source = %s\n",f);
            printf("Destination = %s\n\n",f_copia);
            //printf("t = %s\n\n",t);

            while (feof(f1)==0)
            {
                        fgets (s,max,f1);

                        tamanyoreal=0;
```

```
                for (i=0;s[i]!='\n';i++)
                {
                            tamanyoreal++;//tamaÃ±o de cada linia "s"
                }
                //iniciar vectores a 0
                for(i=0;i<40;i++)
                {
                            nom [i]=0;
                            freq [i]=0;
                            pid [i]=0;
                }
                b=0,c=0,n=0;
                available=1;//por defecto sera 1. Si no esta disponible sera 0

                            if (strcmp(S,"T")==0)//T de terrestre TDT or DVB-T
                            {
                                        tamanyoreal--;
                                        for (i=0; i<tamanyoreal; i++)
                                        {
                                                    if (s[i]==':')
                                                    {
                                                                n++;
                                                    }
                                                    if ((n==0))
                                                    {
                                                                nom [i]=s [i];
                                                                if
                                                                ((nom[i]==comilla)||(nom[i]=='.')||(nom[i
                                                                ]==' '))
                                                                {
                                                                nom[i]='_';//modificacion
                                                                }

                                                                if (nom[i]=='&')
                                                                {
                                                                nom[i]='i';//modificacion
                                                                }

                                                    }
                                                    if ((n==1)&&(s[i]!=':'))
                                                    {
                                                                freq [b]=s [i];
                                                                b++;
                                                    }

                                                    if((n==10)&&(s[i]!=':'))//VPID=0
                                                    {
                                                                if(s[i]=='0')
                                                                            available=0;
                                                    }

                                                    /*if((n==11)&&(s[i]!=':'))// APID=0 Caso radio
                                                    {
                                                                if(s[i]==0)
                                                                            available=0;
                                                    }
                                                    */
                                                    if ((n==12)&&(s[i]!=':'))// &&n==12 en DVBS el pid es
                                                    el 8
                                                    {
                                                                pid [c]=s [i];
                                                                c++;
                                                    }
                                        }
                            }

                            if (strcmp(S,"S")==0) //S de satelitale o DVB-S
                            {

                                        for (i=0; i<tamanyoreal; i++) //-1
                                        {
                                                    if (s[i]==':')
                                                    {
                                                                n++;
                                                    }
                                                    if (n==0)//caracteres del 0 al 30
                                                    {
                                                                nom [a]=s [i];
                                                                if ((nom[a]==comilla)
||(nom[a]=='.')||(nom[a]==' '))//elimina comillas puntos i espacios
                                                                {
                                                                nom[a]='_';//modificacion
                                                                if (nom[a]=='&')//los & son caracteres
especiales en xml
                                                                {
                                                                            nom[a]='i';
                                                                }

                                                                a++;
                                                    }
                                                    if ((n==1)&&(s[i]!=':'))
                                                    {
                                                                freq [b]=s [i];
                                                                b++;
                                                    }

                                                    if ((n==5)&&(s[i]!=':'))//VPID=0
                                                    {
                                                                if(s[i]=='0')
                                                                            available=0;
                                                    }
```

```
                    if ((n==7)&&(s[i]!=':'))//
                    &&n==12 en DVB el pid es el 7
                    {
                        pid [c]=s [i];
                        c++;
                    }
                }
            }
        if (cont>0)
        {
            xml (nom,freq,available,pid,f_copia,1);
        }
        if (cont==0)
        {
            xml (nom,freq,available,pid,f_copia,0);
            cont=1;
        }

        numchan++;

        if (feof(f1)!=0)
        {
            xml (nom,freq,available,pid,f_copia,2);
        }

    }
    printf ("\nNumber of Channels Processed: %d\n",numchan);
    //printf ("%c",comilla);

    //cerrar ficheros i salir OK
    fclose(f1);
    return 0;
}

int xml (char nom[40],char freq[30],int available,char pid[30],char filename[30],int control)
{
    FILE *f2;
    char pid2[30];
    int i;
    //Control=0,1,2. 0=Primera Escritura    1=Segunda hasta penultima 2 =Ultima escritura

    if ((f2 = fopen(filename,"a")) == NULL ) //el xml
    {
    printf("Can't Create Destination File\n");
    return(1);
    }
    if (control==0)
    {
    fprintf (f2,"<streamlist>\n");
    control=1;
    }

    if (control==1)          //imprimir normal
    {
    fprintf (f2,"\t<stream>\n");
    fprintf (f2,"\t\t<name>%s</name>\n",nom);
    fprintf (f2,"\t\t<freq>%s</freq>\n",freq);
    fprintf (f2,"\t\t<available>%d</available>\n",available);
    fprintf (f2,"\t\t<pid>%s</pid>\n",pid);
    fprintf (f2,"\t\t<hidden>disabled</hidden>\n");
    fprintf (f2,"\t</stream>\n");
    }

    if (control ==2) //imprimri nombre i tal i al final lo de stramlist.
    {
        fprintf (f2,"</streamlist>\n");
    }

    fclose (f2);
}
```

## 9.1.2. XML result file

```
<streamlist>
- <stream>
   <name>RaiUno</name>
   <freq>698000000</freq>
   <available>1</available>
   <pid>3401</pid>
   <status>disabled</status>
     </stream>
- <stream>
   <name>RaiDue</name>
   <freq>698000000</freq>
   <available>1</available>
   <pid>3402</pid>
   <status>disabled</status>
     </stream>
- <stream>
   <name>RaiTre</name>
   <freq>698000000</freq>
```

```
  <available>1</available>
  <pid>3403</pid>
  <status>disabled</status>
    </stream>
- <stream>
  <name>RaiUtile</name>
  <freq>698000000</freq>
  <available>1</available>
  <pid>3410</pid>
  <status>disabled</status>
   </stream>
- <stream>
  <name>FD_LEGGERA</name>
  <freq>698000000</freq>
  <available>0</available>
  <pid>3314</pid>
  <status>disabled</status>
   </stream>
- <stream>
  <name>RaiSportSat</name>
  <freq>658000000</freq>
  <available>1</available>
  <pid>3305</pid>
  <status>disabled</status>
   </stream>
- <stream>
  <name>RaiNotizie24</name>
  <freq>658000000</freq>
  <available>1</available>
  <pid>3301</pid>
  <status>disabled</status>
   </stream>
- <stream>
  <name>Rai_Edu1</name>
  <freq>658000000</freq>
  <available>1</available>
  <pid>3307</pid>
  <status>disabled</status>
   </stream>
- <stream>
  <name>Rai_Doc-Futura</name>
  <freq>658000000</freq>
  <available>1</available>
  <pid>3310</pid>
  <status>disabled</status>
   </stream>
- <stream>
  <name>RADIOUNO</name>
  <freq>658000000</freq>
  <available>0</available>
  <pid>3311</pid>
  <status>disabled</status>
   </stream>
- <stream>
  <name>RADIODUE</name>
  <freq>658000000</freq>
  <available>0</available>
  <pid>3312</pid>
  <status>disabled</status>
   </stream>
- <stream>
  <name>RADIOTRE</name>
  <freq>658000000</freq>
  <available>0</available>
  <pid>3313</pid>
  <status>disabled</status>
   </stream>
- <stream>
  <name>CCTV9</name>
  <freq>658000000</freq>
  <available>1</available>
  <pid>3304</pid>
  <status>disabled</status>
   </stream>
- <stream>
  <name>SAT2000</name>
  <freq>658000000</freq>
  <available>1</available>
  <pid>3309</pid>
  <status>disabled</status>
   </stream>
- <stream>
  <name>Servizio_OTA</name>
  <freq>618000000</freq>
  <available>0</available>
  <pid>1</pid>
  <status>disabled</status>
   </stream>
- <stream>
  <name>24ore_tv</name>
  <freq>618000000</freq>
  <available>1</available>
  <pid>2</pid>
  <status>disabled</status>
    </stream>
- <stream>
  <name>Class_News</name>
  <freq>618000000</freq>
  <available>1</available>
  <pid>3</pid>
  <status>disabled</status>
    </stream>
- <stream>
  <name>Coming_Soon</name>
  <freq>618000000</freq>
  <available>1</available>
  <pid>4</pid>
  <status>disabled</status>
    </stream>
- <stream>
  <name>BBC_World</name>
  <freq>618000000</freq>
  <available>1</available>
  <pid>5</pid>
  <status>disabled</status>
    </stream>
- <stream>
  <name>Boing</name>
  <freq>618000000</freq>
  <available>1</available>
  <pid>6</pid>
  <status>disabled</status>
    </stream>
- <stream>
  <name>Mediaset_Premium_1</name>
  <freq>618000000</freq>
  <available>1</available>
  <pid>101</pid>
  <status>disabled</status>
    </stream>
- <stream>
  <name>Mediaset_Premium_2</name>
  <freq>618000000</freq>
  <available>1</available>
  <pid>102</pid>
  <status>disabled</status>
    </stream>
- <stream>
  <name>Mediaset_Premium_3</name>
  <freq>618000000</freq>
  <available>1</available>
  <pid>103</pid>
  <status>disabled</status>
    </stream>
- <stream>
  <name>Mediaset_Premium_4</name>
  <freq>618000000</freq>
  <available>1</available>
  <pid>104</pid>
  <status>disabled</status>
    </stream>
- <stream>
  <name>Mediaset_Premium_5</name>
  <freq>618000000</freq>
  <available>1</available>
  <pid>105</pid>
  <status>disabled</status>
    </stream>
- <stream>
  <name>HUMAX_DOWNLOAD_SVC</name>
  <freq>618000000</freq>
  <available>0</available>
  <pid>8015</pid>
  <status>disabled</status>
    </stream>
- <stream>
  <name>Canale_5</name>
  <freq>754000000</freq>
  <available>1</available>
  <pid>21</pid>
  <status>disabled</status>
    </stream>
- <stream>
  <name>24ore_tv</name>
  <freq>754000000</freq>
  <available>1</available>
  <pid>22</pid>
  <status>disabled</status>
    </stream>
- <stream>
```

```
  <name>Class_News</name>
  <freq>754000000</freq>
  <available>1</available>
  <pid>23</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>Coming_Soon</name>
  <freq>754000000</freq>
  <available>1</available>
  <pid>24</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>BBC_World</name>
  <freq>754000000</freq>
  <available>1</available>
  <pid>25</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>Mediashopping</name>
  <freq>754000000</freq>
  <available>1</available>
  <pid>26</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>s7</name>
  <freq>754000000</freq>
  <available>0</available>
  <pid>27</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>s8</name>
  <freq>754000000</freq>
  <available>0</available>
  <pid>28</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>s9_EI</name>
  <freq>754000000</freq>
  <available>0</available>
  <pid>29</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>s10_EI</name>
  <freq>754000000</freq>
  <available>0</available>
  <pid>30</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>PPlus</name>
  <freq>754000000</freq>
  <available>0</available>
  <pid>998</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>PSV</name>
  <freq>754000000</freq>
  <available>0</available>
  <pid>999</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>LA7_Cartapiu__attivazione</name>
  <freq>818000000</freq>
  <available>0</available>
  <pid>11</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>CANALE_TEST</name>
  <freq>818000000</freq>
  <available>0</available>
  <pid>3</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>LA7_Cartapiu__B</name>
  <freq>818000000</freq>
  <available>0</available>
  <pid>7</pid>
  <status>disabled</status>
  </stream>
```

```
- <stream>
  <name>LA7_Cartapiu__C</name>
  <freq>818000000</freq>
  <available>0</available>
  <pid>8</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>LA7_Cartapiu__D</name>
  <freq>818000000</freq>
  <available>0</available>
  <pid>9</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>LA7_Cartapiu__E</name>
  <freq>818000000</freq>
  <available>0</available>
  <pid>10</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>LA7_Cartapiu__A</name>
  <freq>818000000</freq>
  <available>0</available>
  <pid>6</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>LA7_Cartapiu__F</name>
  <freq>818000000</freq>
  <available>0</available>
  <pid>15</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>LA7</name>
  <freq>818000000</freq>
  <available>1</available>
  <pid>1</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>MTV_ITALIA</name>
  <freq>818000000</freq>
  <available>1</available>
  <pid>2</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>TELEMARKET</name>
  <freq>818000000</freq>
  <available>1</available>
  <pid>12</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>LA7_SPORT</name>
  <freq>818000000</freq>
  <available>1</available>
  <pid>4</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>RETECAPRI</name>
  <freq>818000000</freq>
  <available>0</available>
  <pid>14</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>HUMAX_DOWNLOAD_SVC</name>
  <freq>818000000</freq>
  <available>0</available>
  <pid>8015</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>XXXX</name>
  <freq>818000000</freq>
  <available>0</available>
  <pid>16</pid>
  <status>disabled</status>
  </stream>
- <stream>
  <name>Rete_4</name>
  <freq>586000000</freq>
  <available>1</available>
  <pid>11</pid>
  <status>disabled</status>
```

```
      </stream>
  - <stream>
      <name>Italia_1</name>
      <freq>586000000</freq>
      <available>1</available>
      <pid>12</pid>
      <status>disabled</status>
    </stream>
  - <stream>
      <name>Si_SoloCalcio</name>
      <freq>586000000</freq>
      <available>1</available>
      <pid>13</pid>
      <status>disabled</status>
    </stream>
  - <stream>
      <name>Sportitalia</name>
      <freq>586000000</freq>
      <available>1</available>
      <pid>14</pid>
      <status>disabled</status>
    </stream>
  - <stream>
      <name>Si_Live24</name>
      <freq>586000000</freq>
      <available>1</available>
      <pid>15</pid>
      <status>disabled</status>
    </stream>
  - <stream>
      <name>LCI</name>
      <freq>586000000</freq>
      <available>1</available>
      <pid>16</pid>
      <status>disabled</status>
    </stream>
  - <stream>
      <name>S17</name>
      <freq>586000000</freq>
      <available>0</available>
      <pid>17</pid>
      <status>disabled</status>
    </stream>
  - <stream>
      <name>S18</name>
      <freq>586000000</freq>
      <available>0</available>
      <pid>18</pid>
      <status>disabled</status>
    </stream>
  - <stream>
      <name>DV</name>
      <freq>586000000</freq>
      <available>0</available>
      <pid>999</pid>
      <status>disabled</status>
    </stream>
</streamlist>
```

## 9.1.3. Web-Page Code

The entire code can be found in the direction: http://escher.iet.unipi.it

### *9.1.3.1. Broadcast.php*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html><head>

<meta name="author" content="Stefano Lucetti">
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<meta name="keywords" content="university, Pisa, engineering, telecommunications, networks">
<link rel="stylesheet" type="text/css" href="style2.css">
<link rel="SHORTCUT ICON" href="http://wwwtlc.iet.unipi.it/favicon.ico"><title>TLCNETGROUP Home Page - University of
Pisa</title>
</head><body>
```

```
<script language="javascript">

function volume_up()
      {
      var previous =  document.video1.get_volume();
          var newvolume = previous + 10;
          if( newvolume > 200 )  newvolume = 200;
          document.video1.set_volume( newvolume );
          var volume = document.getElementById("volume_status");
          volume.innerHTML = document.video1.get_volume() + " %";
          }
function volume_down()
          {
          var previous =  document.video1.get_volume();
          var newvolume = previous - 10;
          if( newvolume < 0 )  newvolume = 0;
          document.video1.set_volume( newvolume );
          var volume = document.getElementById("volume_status");
      volume.innerHTML = document.video1.get_volume() + " %";
          }

function status()
          {
                    var play_status = document.getElementById("play_status");
                    var time = document.getElementById("time");
                    var length = document.getElementById("length");
                    play_status.innerHTML = document.video1.isplaying() ?      "Playing"           : "Not playing";
                    if( document.video1.isplaying() == true )
                    {
                              got_time = document.video1.get_time();
                              hours = Math.floor(got_time/ 3600);
                              minutes = Math.floor((got_time/60) % 60);
                              seconds = got_time % 60;
                              if ( hours < 10 ) hours = "0" + hours;
                              if ( minutes < 10 ) minutes = "0" + minutes;
                              if ( seconds < 10 ) seconds = "0" + seconds;
                              time.innerHTML = hours+":"+minutes+":"+seconds;
                              got_length = document.video1.get_length();
                              hours = Math.floor(got_length/ 3600);
                              minutes = Math.floor((got_length/60) % 60);
                              seconds = got_length % 60;
                              if ( hours < 10 ) hours = "0" + hours;
                              if ( minutes < 10 ) minutes = "0" + minutes;
                              if ( seconds < 10 ) seconds = "0" + seconds;
                              length.innerHTML = hours+":"+minutes+":"+seconds;
                    }
                    else
                    {
                              time.innerHTML = "--:--:--";
                              length.innerHTML = "--:--:--";
                    }
                    setTimeout("status()", 1000 );
          }

          function play_selected()
          {
                    select = document.getElementById("item");
                    set_item( select.value );                                            document.video1.play();
          }
          function set_item( name)
          {
                    document.video1.stop();
                    document.video1.clear_playlist();
                    document.video1.add_item( name );
          }

</script>
<!-- ::::::::::::::::::::  MENU :::::::::::::::::::: -->
<div class="floatmenu">
<table border="0" cellpadding="0" cellspacing="3" width="180">
<col width="180">

<tbody><tr>
          <td class="cx">
          <img src="vlc_img/cher_black.jpg" alt="Cherubino" height="70" width="69"></td>
          </tr>

          <tr>
                    <td>
                              <div class="playlist">
                              <div class="head">BROADCAST TV </div>
                              <div class="mini">
                              <select id="item">
                              </select>
                              <input name="chewa" value="Play selected item" onclick="play_selected();" type="button"
                              />
                              </div>
                              </div>
                    </td>
          </tr>

          <tr>
                    <td>
                    <div class="status">
                    <div class="head">Status</div>
                    <div class="content">
                    <table>
                    <tbody>
                    <tr>
                    <td>Status</td>
                    <td><span id="play_status">Unknown</span>
                    </td>
          </tr>

          <tr>
                    <td>Time</td>
                    <td><span id="time">--:--:--</span>
                    </td>
          </tr>

          <tr>
                    <td>Total Length</td>
                    <td><span id="length">--:--:--</span></td>
          </tr>

          <tr>
                    <td>Volume</td>
                    <td><span id="volume_status"></span></td>
          </tr>
</tbody>
</table>
</div>
</div>
</td>
</tr>

<tr>
<td><div class="playlist">
<a href="vodindex.php" target="_self">Video</a><br />
<a href="broadindex.php" target="_self">Broadcast TV</a><br />
<a href="personal.php" target="_self">Your Playlist</a>
</div>

<div class="playlist">
<a href="index.php" target="_self">Home</a><br />
<a href="help.php" target="_self">Help</a>
</div>
</td>
</tr>
</tbody>
</div>
<div class="controls" id="cont">
<div class="playback">
<div class="head">Playback Control</div>
<div class="content"> <input name="chewa" value="Play" onclick="document.video1.play();" type="button" /><input
name="chewa" value="Pause" onclick="document.video1.pause();" type="button" /><input name="chewa" value="Stop"
onclick="document.video1.stop();" type="button" /><br />
<input name="chewa" value="Seek -10s" onclick="document.video1.seek(-10,true);" type="button" />
<input name="chewa" value="Seek +10s" onclick="document.video1.seek(10,true);" type="button" /></div>
</div><div class="av">
<div class="head">AV Control</div><div class="content">
<input name="chewa" value="Vol +" onclick="volume_up()" type="button" /><input name="chewa" value="Vol -"
onclick="volume_down()" type="button" /><input name="chewa" value="Mute" onclick="document.video1.mute()"
type="button" /><input name="chewa" value="FullScreen" onclick="document.video1.fullscreen();" type="button" />
</div></div></div>
<!-- ::::::::::::::::  END OF MENU :::::::::::::::: -->
<div class="mainpage2">
  <embed type="application/x-vlc-plugin" id="video1" autoplay="no" loop="yes" height="300" width="400"
target="video1">
<script language="javascript">
     var volume = document.getElementById("volume_status");
     volume.innerHTML = document.video1.get_volume()+ " %";
     setTimeout("status()", 1 );
  </script>
    <!--
<p class="cx">
<br>
<a class="nohover" href="http://validator.w3.org/check/referer">
<img src="/images/valid-html401.png" style="width:88px;height:31px" alt="Valid HTML 4.01!">
</a>
<a class="nohover" href="http://jigsaw.w3.org/css-validator/">
<img style="width:88px;height:31px" src="/images/vcss.png" alt="Valid CSS!">
</a>
</p>

-->
</div>
</body></html>
```

## 9.2. Protocols related to streaming do not used in the work

### 9.2.1. Video codec

#### *9.2.1.1. MPEG-4*

MPEG-4 is an ISO/IEC standard developed by MPEG (Moving Picture Experts Group). MPEG-4 is designed basically in three fields:

- Digital television
- Interactive graphics applications (synthetic content)
- Interactive multimedia

With these fields MPEG-4 pretends to provide standardized ways for units called “media objects”.

These media objects can have natural or synthetic origin, for example a video camera or a computer. Anyway the objective is to be able to transport this media objects over network channels providing QoS and interact with the scene created.

The following sections illustrate the MPEG-4 functionalities described above, using the audiovisual scene depicted in Figure 9-1.

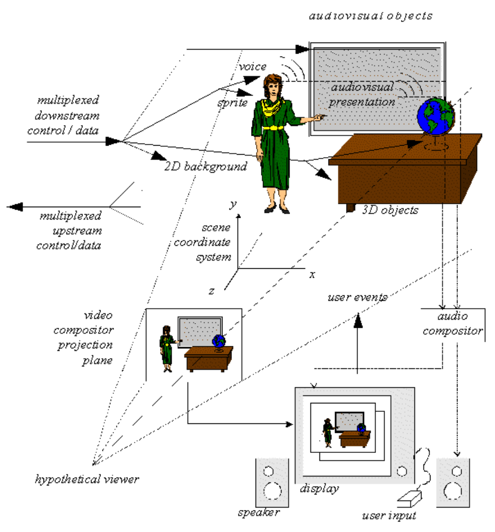


**Figure 9.1:** an example of an MPEG-4 Scene

MPEG-4 makes possible to:

- Identify access units, transport timestamps, clock reference information and data loss.
- Transmit control information to the required QoS for each stream and convert it into network resources.
- Associate elementary streams to media objects.
- Transmit the mapping of elementary streams to the channels.

### *9.2.1.2. Other Codecs*

The purpose of this section is not to enumerate all existing codecs in Internet but a brief explanation of the most used.

As explained in section “2.4.3. Header Format”, depending on the field Payload type it can be different audio or video codecs[37] being implemented.

| PAYLOAD TYPES | ENCODING NAME | AUDIO /VIDEO (A/V) | CLOCK RATE (HZ) | CHANNELS (AUDIO) |
|---|---|---|---|---|
| **0** | PCMU | A | 8.000 | 1 |
| **1** | 1016 | A | 8.000 | 1 |
| **2** | G721 | A | 8.000 | 1 |
| **3** | GSM | A | 8.000 | 1 |
| **4** | unassigned | A | 8.000 | 1 |
| **5** | DVI4 | A | 8.000 | 1 |
| **6** | DVI4 | A | 16.000 | 1 |
| **7** | LPC | A | 8.000 | 1 |
| **8** | PCMA | A | 8.000 | 1 |
| **9** | G722 | A | 8.000 | 1 |
| **10** | L16 | A | 44.100 | 2 |
| **11** | L16 | A | 44.100 | 1 |
| **12** | unassigned | A |  | - |
| **13** | unassigned | A |  | - |
| **14** | MPA | A | 90.000 | - |
| **15** | G728 | A | 8.000 | 1 |
| **16-23** | unassigned | A |  | - |
| **24** | unassigned | V |  | - |
| **25** | CelB | V | 90.000 | - |
| **26** | JPEG | V | 90.000 | - |
| **27** | unassigned | V |  | - |
| **28** | Nv | V | 90.000 | - |
| **29** | unassigned | V |  | - |
| **30** | unassigned | V |  | - |
| **31** | H261 | V | 90.000 | - |
| **32** | MPV | V | 90.000 | - |
| **33** | MP2T | AV | 90.000 | - |
| **3471** | unassigned | - | - | - |
| **72-76** | Reserved | N/A | N/A | N/A |
| **77-95** | unassigned | - | - | - |
| **96-127** | Dynamic | - | - | - |

**Table 9.1** Payload types (PT) for standard audio and video encodings[38]

[37] Extracted from RFC 3551: RTP Profile for Audio and Video Conferences with Minimal Control

*Audio*

### PCM: PCMA and PCMU

PCMA and PCMU audio data is encoded as eight bits per sample, after logarithmic scaling. PCMU denotes mu-law scaling, PCMA A-law scaling. The 56 kb/s and 48kb/s modes are not applicable to RTP, since PCMA and PCMU always be transmitted as 8-bit samples.

### GSM

GSM (Group Special Mobile) denotes the European GSM 06.10 standard for full-rate speech transcoding, ETS 300 961, which is based on RPE/LTP (residual pulse excitation/long term prediction) coding at a rate of 13 kb/s. Blocks of 160 audio samples are compressed into 33 octets, for an effective data rate of 13,200 b/s.

### G722

G722 is specified in ITU-T Recommendation G.722, "7 kHz audio-coding with 64 kbps". The G.722 encoder produces a stream of octets, each of which shall be octet-aligned in an RTP packet. The octet rate or sample-pair rate is 8,000 Hz.

### G723

G723 is specified in ITU Recommendation, "Dual-rate speech coder for multimedia communications transmitting at 5.3 and 6.3 kbit/s". The G.723.1 5.3/6.3 kbit/s codec was defined by the ITU-T as a mandatory codec for ITU-T H.324 GSTN videophone terminal applications. Audio is encoded in 30 ms frames, with an additional delay of 7.5 ms due to look-ahead. A G.723.1 frame can be one of three sizes: 24 octets (6.3 kb/s frame), 20 octets (5.3 kb/s frame), or 4 octets.

*Video*

### H.261

This code is used to video-conference. It is organized in groups. The video stream is composed of a sequence of images, or frames, which are organized as a set of Groups of Blocks (GOB). Each GOB holds a set of 3 lines of 11 macro blocks (MB). Each MB carries information on a group of 16x16 pixels, including luminance information specified for 4 blocks of 8x8 pixels, while chrominance information is given by two "red" and "blue" color difference components at a resolution of only 8x8 pixels. The bits resulting from the Huffman encoding are arranged in 512-bit frames, containing 2 bits of

[38] Extracted from the RFC number 3551: “RTP Profile for Audio and Video Conferences with Minimal Control”

synchronization, 492 bits of data and 18 bits of error correcting code. The 512-bit frames are then interlaced with an audio stream.

Some other characteristics:

- Bitrate: *p* x 64Kbps, where p = 1, 2 … 30. Typically bitrates from 64 to 384 kbps.
- CODEC DCT with motion compensation
- Sampling 4:2:0 Y:Cr:Cb
- Support resolutions
- CIF: 352 x 288
- QCIF: 176 x 144

**H263**

The encoding is specified for "Video coding for low bit rate communication".

## 9.2.2. QoS

### *9.2.2.1. RSVP (Resource ReSerVation Protocol)*

RSVP is a resource reservation setup protocol designed for an integrated Internet service. It is used to request specific QoS from the network to particular application data streams or flows and establish and maintain the provided service.

RSVP has the following attributes:

- Makes resource reservations for unicast and multicast applications and unidirectional data flows.
- Receiver-oriented: The receiver initiates and maintains the resource reservation used for a flow.
- Provides support for dynamic membership changes and automatic adaptation to routing changes.
- Depends on a routing protocol.
- Transports and maintains traffic and policy control parameters.
- Provides transparent operation through routers that do not support it.
- Supports IPv4 and IPv6.

*Header and Objects Format*

An RSVP message consists of a common header, followed by a body consisting of a variable number of variable-length, called “objects".

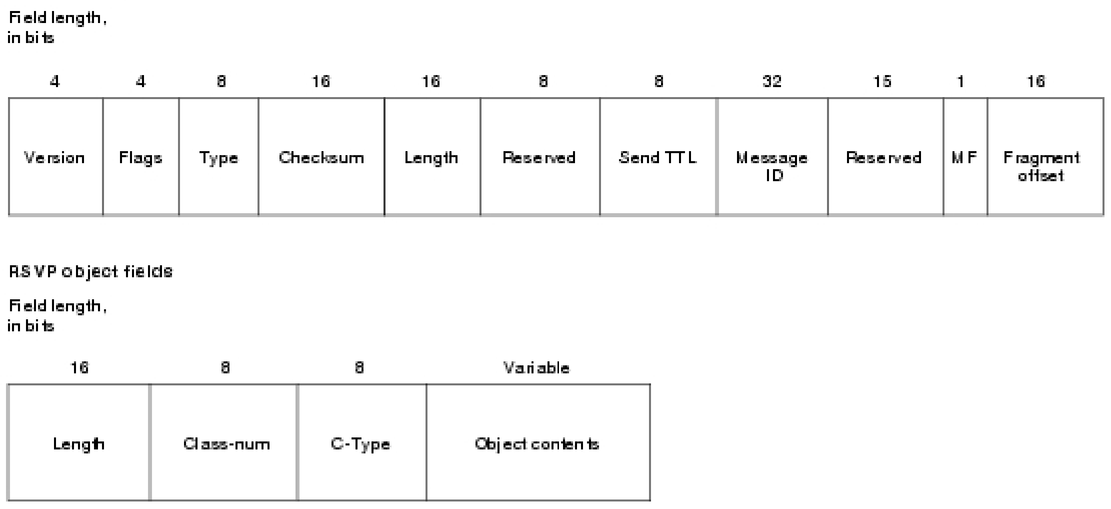


**Figure 9.2** : RSVP message header and object fields[39]

The fields in the common header are as follows:

- **Vers:** 4 bits. Protocol version number.
- **Flags**: 4 bits. 0x01-0x08: Reserved
- **Msg Type:** 8 bits
- **Checksum:** 16 bits.
- **Send_TTL**: 8 bits. The IP TTL value for the message sent.
- **Length**: 16 bits. The total length of this RSVP message.

*Object Formats*

Every object consists of one or more 32-bit words with a one-word header. An object header has the following fields:

- **Length:** A 16-bit field containing the total object length.
- **Class-Num**: Identifies the object class.
- **C-Type:** Identifies the Object-Class.
- **Object contents:** The Length, Class-Num, and C-Type fields specify the form of the object content.

*9.2.2.2. RTCP (Real Transfer Control Protocol)*

*What is RTCP?*

The RTP control protocol (RTCP) is used with RTP and is based on the periodic transmission of control packets to all participants in the session, using the same distribution mechanism as the data packets. The messages include information

[39] Extracted from: http://www.pulsewan.com/

that can be used by higher-level applications to control the session and improve the transmission. RTCP has four primary functions:

- **Provide feedback** on the quality of the data distribution.
- RTCP takes a persistent transport-level to identify the source called the canonical name or **CNAME**. The CNAME is an identifier item required to provide compromise from the SSRC identifier to an identifier for the source that remains constant.
- Each participant send its **control packets** to all the others.
- Optionally, transmits always **minimal session control information.**

*Header Format and Types*

The common header format is this, although depending on packet type the Data field will change:

RTCP packet header format:

| 2 | 3 | 8 | 16 | 32bit |
|---|---|---|---|---|
| Ver | P | Count | Type | Length |
| Data | | | | |

**Figure 9.3: RTCP Packet Format.**[40]

- **Version (V)**: 2 bits. Identifies the version of RTCP.
- **Padding (P)**: 1 bit.
- **Report count (RC):** 5 bits. Contains the number of blocks contained in Data.
- **Packet type (PT):** 8 bits. Identifies the RTCP type packet.
- **Length:** 16 bits. Contains the length of the RTCP packet in 32-bit words minus one, including the header and any padding.
- **SSRC: 32 bits:** Contains the source identifier for the originator of the packet.
- **Data:** depending the kind of the packet the information and fields can be very different.

[40] Extraced from: “http://www.protocolbase.net/protocols/protocol_RTCP.php”

RTCP Packet Types (PT)

| Type | ab. | Description |
|---|---|---|
| 0-191 | | |
| 192 | FIR | Full INTRA-frame request, indicates that a receiver requires a full encoded image. |
| 193 | NACK | Negative acknowledgement. |
| 194-199 | | |
| 200 | SR | Sender report. |
| 201 | RR | Receiver report. |
| 202 | SDES | Source description items, including CNAME. |
| 203 | BYE | Goodbye, Indicates end of participation. |
| 204 | APP | Application-specific functions. |
| 205 | RTPFB | , Generic RTP Feedback. |
| 206 | PSFB | Payload-specific. |
| 207 | XR | RTCP extension. |
| 208-255 | | |

**Table 9.2: RTCP PT**

| SDES TYPES | | |
|---|---|---|
| Abbrev | Name | Value |
| END | End or SDES list | 0 |
| CNAME | Canonical Name | 1 |
| NAME | User Name | 2 |
| EMAIL | User's Electronic mail Address | 3 |
| PHONE | User's Phone Number | 4 |
| LOC | Geographic user Location | 5 |
| TOOL | Name of Application or Tool | 6 |
| NOTE | Notice about the Source | 7 |
| PRIV | Private Extensions | 8 |

**Table 9.3 : Sources Description Types**

## Why RTCP?

The bit-rate of a stream could be changed to combat network congestion. Network congestion varies with the number of users on the net, and this also affects to the available bit-rate on the network for streaming. To optimize this server has to adjust itself dynamically to the highest bit-rate possible. These adjustments are possible by using the RTCP reports from the media player to measure the network congestion and switch the stream rates to multiple bit-rate media files.