



**Tiago Ferreira
Magalhães**

IPTV Open Source

“Tell me and I forget. Teach me and I remember. Involve me and I learn.”

— Benjamin Franklin



**Tiago Ferreira
Magalhães**

IPTV Open Source

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Diogo Nuno Pereira Gomes, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Rui Luís Andrade Aguiar, Professor catedrático do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Professor Doutor Joaquim Arnaldo Carvalho Martins
Professor Catedrático da Universidade de Aveiro

vogais / examiners committee

Doutor João Pedro Brites Ferreira Nogueira
Investigador na Altice Labs

Professor Doutor Diogo Nuno Pereira Gomes
Professor Auxiliar da Universidade de Aveiro

agradecimentos / acknowledgements

Gostava de, em primeiro lugar, agradecer ao meu orientador, Diogo Gomes, pela orientação e apoio prestados na realização deste trabalho. Também quero agradecer a todos os elementos do Instituto de Telecomunicações, local que me acolheu, por todas as condições que me disponibilizou tanto a nível de recursos como a nível humano. Quero deixar também um agradecimento especial aos meus amigos, Ivo Silva, Ricardo Martins, Diogo Cardoso, André Santos e Daniel Silva pelas discussões, momentos de convívio e amizade incondicional durante o curso. Aproveito ainda para agradecer a todos os que me apoiaram ao longo do meu percurso académico, em particular a todos os meus colegas do curso de Mestrado Integrado em Engenharia de Computadores e Telemática, que tornam este curso numa grande família. Finalmente, um agradecimento especial aos meus pais, Joaquim Magalhães e Maria Ferreira, e ao meu irmão, João Magalhães, por toda a força, motivação, apoio e educação que me deram desde sempre.

Palavras Chave

Streaming HTTP, Javascript, Televisão Digital Terrestre, Internet.

Resumo

Os constantes desenvolvimentos a nível de infraestrutura de rede permitem fornecer serviços multimedia de qualidade sobre redes IP. Além disso, tráfego de dados em redes móveis tem crescido com o aumento de capacidades dos dispositivos móveis inteligentes, crescimento esse que motivou a criação de soluções multi-plataforma que podem direccionar não só esses mesmos dispositivos, mas, no caso do mundo IPTV, também as set-top-boxes, televisores inteligentes e computadores. HTTP streaming é uma técnica direccionada a transmissão de dados multimedia sobre redes IP. Várias plataformas a nível global fazem uso de streaming HTTP, sendo Youtube ou Netflix dos exemplos mais conhecidos. Neste trabalho, é proposta uma arquitetura para um servidor de streaming dos canais públicos Portugueses, bem como, uma arquitetura de cliente multiplataforma. O software proposto pode ser implementado em qualquer servidor regular e fornecer funcionalidades extra, como grelha de programação. O cliente é suportado por qualquer navegador Media Source Extension (MSE) sem a necessidade de programas ou extensões adicionais, como o Flash. Finalmente, foram feitos alguns testes nas duas entidades (servidor e cliente) para testar o desempenho do servidor e o comportamento do cliente através de diferentes condições a nível de rede.

Keywords

HTTP Streaming, Javascript, Digital Television Terrestrial, Internet Protocol.

Abstract

The continuous developments made on the network infrastructures allow to provide quality in multimedia services over-the-top. Also, mobile data traffic has been increasingly growing along with smartphone's capabilities. This growth motivates the creation of multi-platform solutions that can target not only these devices but, in the case of the IPTV world, also the set-top-boxes, smart televisions and computers. HTTP streaming is a technique directed to stream over-the-top multimedia. Many global platforms make use of HTTP streaming, with names like Youtube or Netflix. In this work, a server architecture for streaming the public Portuguese channels as well as a multiplatform client architecture are proposed. The proposed software can be deployed on any regular server and provide extra functionalities such as tv listings. The client can run on any Media Source Extension (MSE) browser, without the need of any plugin like Flash. Finally, some tests were made on the two entities (server and client) in order to test the performance of the server and behaviour of the client through different network conditions.

CONTENTS

CONTENTS	i
LIST OF FIGURES	v
LIST OF TABLES	vii
ACRONYMS	ix
1 INTRODUCTION	1
1.1 Motivation	1
1.1.1 Mobile market importance	2
1.1.2 Over-The-Top growth	2
1.1.3 Portugal's available TV platforms	3
1.2 Objectives	4
1.3 Thesis Outline	5
2 STATE OF THE ART	7
2.1 Introduction	7
2.2 The evolution of Television	8
2.3 Digital Video Broadcasting	9
2.4 Conventional Streaming	10
2.5 HTTP Streaming	11
2.5.1 Live Television	12
2.5.2 Dynamic Adaptive Streaming over HTTP	13
2.5.3 HTTP Live Stream (HLS)	16
2.5.4 Microsoft Smooth Streaming	18
2.6 Serving the contents	20
2.6.1 Content Distributed Networks	20
2.6.2 HTTP Servers	21
2.7 Media Preparation	22
2.8 Television Listings	25
2.8.1 XMLTV format	26
2.9 Mobile Devices	28
2.9.1 Android	28
2.9.2 Apple iOS	31
2.9.3 Mobile Devices Overview	31
2.10 HTML frameworks	31

2.10.1	AngularJS	32
2.10.2	ReactJS	32
2.10.3	JavaScript Libraries	34
2.10.4	Overview	35
2.11	IPTV systems available	36
2.11.1	YouTube	36
2.11.2	Kodi	37
2.11.3	RTP Play	38
2.11.4	MEO Go	39
2.11.5	Restreamer	39
2.11.6	Tv Headend	40
2.11.7	Overview	41
3	SOLUTIONS AND SCENARIOS	43
3.1	Ideal Scenario	43
3.2	Requirements	44
3.3	Goals	44
3.4	Possible scenarios	45
4	IMPLEMENTATION	51
4.1	Server side implementation	51
4.1.1	Servers	51
4.1.2	Dvb-t setup	53
4.1.3	Transcoding	62
4.1.4	Tv listings	65
4.1.5	Serve the contents	69
4.2	Client side implementation	73
4.2.1	Player Module	73
4.2.2	EPG Module	74
4.2.3	Help Module	76
4.2.4	Definitions Module	77
4.2.5	Animations Module	77
4.2.6	Application Module	77
4.3	Setup Overview	80
5	EVALUATION AND RESULTS	81
5.1	Dvb	81
5.2	Transcoding	82
5.3	Server	86
5.4	Client	91
5.4.1	EPG	92
6	CONCLUSIONS	95
6.1	Future Work	95
6.2	Collaborations	96
6.3	Contributions	96
6.3.1	A Cloud-based Mobile Video Delivery over Heterogeneous Networks	96
6.3.2	23rd seminar of the Mobile Communications Thematic Network	98
	APPENDIX A	99

APPENDIX B	101
REFERENCES	103

LIST OF FIGURES

1.1	Top 10 peak period applications responsible for Real-time entertainment data, in North America.	3
1.2	Digital Television ready equipments sold monthly. At light green are presented the metrics for the Televisions and at dark green are represented the decoder boxes.	4
2.1	Investments made on different advertisement segments with forecast to 2018, in the United States.	9
2.2	Cisco's forecast of mobile data traffic for the period of 2016-2021[24].	11
2.3	Simple illustration of the DASH MPD format.	13
2.4	Generic behavior of the DASH protocol	14
2.5	HLS global architecture.	16
2.6	HLS master playlist example.	17
2.7	HTTP Live Stream playlist example.	17
2.8	fMP4 file structure used in Smooth Streaming. <i>moof</i> boxes contain fragment metadata and <i>mdat</i> boxes the respective data.	19
2.9	KODI EPG platform.	26
2.10	Android Stack illustration.	29
2.11	React's FLUX Data Flow.	34
2.12	Youtube player statistics.	37
2.13	Kodi Krypton User Interface.	38
2.14	RTP Play m3u8 playlist on a TV channel.	38
2.15	RTP Play m3u8 playlist on a radio station.	39
2.16	Restreamer architecture.	40
3.1	Over-The-Top communication ideal scenario.	43
3.2	Fundamental blocks and workflow of the server architecture.	46
3.3	Server architecture with transcoding using different physical machines.	47
3.4	Client architecture composed of independent modules.	48
4.1	Network diagram of the IPTV server.	52
4.2	Configuration file generated by w_scan tool.	53
4.3	PID scan using dvbsnoop.	55
4.4	Program Association Table in a given moment.	56
4.5	Event Information Table snippet.	57
4.6	Service Description Table snippet.	58
4.7	Program Map Table snippet.	59
4.8	Time Date Table snippet.	60

4.9	Media information captured by FFMPEG tool.	60
4.10	mumudvb configuration file.	61
4.11	Stream properties originated by the MuMuDVB tool.	62
4.12	M3U playlist file with the available channels.	69
4.13	Player component diagram.	73
4.14	Player running as background on the client application.	74
4.15	EPG component diagram.	75
4.16	Restarting program accordingly to the EPG information.	75
4.17	Help component diagram.	76
4.18	Help interface on client's platform.	76
4.19	Increasing volume animation.	77
4.20	Application component diagram.	78
4.21	Main menu interface with the available options.	79
4.22	Layout of Settings menu (at left) and Channels menu (at right).	79
4.23	Setup with representation of the server, API specification and client player.	80
5.1	MuMuDVB impact on the CPU.	82
5.2	FFMPEG's impact on iptvs server, in terms of CPU load, when transcoding 4 and 5 channels.	83
5.3	FFMPEG's impact on CPU and memory, on the iptvs server.	83
5.4	FFMPEG's impact on CPU and memory, on the iptv2 server.	84
5.5	Bitrate variations over time with constant rate frame option.	85
5.6	Bitrate variations over time without constant rate frame option.	85
5.7	Low quality video statistics.	86
5.8	Response times for 50 concurrent users.	87
5.9	Request latency for 50 concurrent users.	88
5.10	Network throughput for 50 concurrent users.	88
5.11	A snippet of the threads in uninterruptible mode during the test.	89
5.12	CPU load during the test made on the server.	90
5.13	Memory usage during the test made on the server.	90
5.14	Network load variation during the test made on the server.	91
5.15	Response time variations when requesting the current program for the RTP1 channel.	92
5.16	Response time variations when requesting the next 10 programs for the TVI channel.	93
6.1	Mobile application architecture.	97
6.2	Headend architecture.	98

LIST OF TABLES

2.1	Some of the tags present on a M3U8 file.	18
2.2	Streaming protocols comparison.	20
2.3	Comparison between RED5's Open Source and PRO versions.	22
2.4	Comparison between transcoding tools.	25
2.5	Android's versions compatibility with some streaming technologies.	29
2.6	Recommended video encoding for playback in H.264 Baseline Profile codec.	30
2.7	List of most known browsers that features MSE and HLS support.	36
4.1	Server machines specifications.	52
4.2	Medium and Low H.264 specifications to increase device compatibility.	64
4.3	Comparison between TV listings data providers.	66

ACRONYMS

AAC	Advanced Audio Coding	DOM	Document Object Model
AC-3	Audio Codec 3	DASH	Dynamic Adaptative Streaming over HTTP
AMP	Akamai Adaptive Media Player	DECA	Departamento de Comunicação e Arte da Universidade de Aveiro
API	Application Programming Interface	EIT	Event Information Table
ATSC	Advanced Television System Committee	EPG	Electronic Programming Guide
AWS	Amazon Web Services	EME	Encrypted Media Extension
AMF	Action Message Format	FastCGI	Fast Common Gateway Interface
CD	Compact Disk	ftype	File Type
CDN	Content Delivery Network	FLV	Flash Live Video
CORS	Cross-Origin Resource Sharing	fMP4	Fragmented MPEG-4 File Segments
CPU	Central Processing Unit	FPS	Frames-per-second
CRF	constant rate frame	FTP	File Transfer Protocol
CSS	Cascading Style Sheets	GB	Giga Byte
CSV	Comma Separated Values	GCC	GNU Compiler Collection
CTV	Cable Television	HAS	HTTP adaptive streaming
DRM	Digital Rights Management	HD	High Definition
DTV	Digital Television	HDMI	High Definition Multimedia Interface
DVB	Digital Video Broadcasting	HDMI-CEC	HDMI- Consumer Electronics Control
DVB-C	Digital Video Broadcasting-Cable	HLS	HTTP Live Stream
DVB-H	Digital Video Broadcasting-Handheld	HTTP	Hyper-Text Transport Protocol
DVB-S	Digital Video Broadcasting-Satellite	HTTPS	Hyper-Text Transport Protocol Secure
DVB-T	Digital Video Broadcasting-Terrestrial	HTML	HiperText Markup Language
DVB-T2	Digital Video Broadcasting-Terrestrial 2	H.264	Advanced Video Coding, MPEG-4 Part 10
DVD	Digital Video Disk	H.265	High Efficiency Video Coding
DVR	Digital Video Recording	ID	Identifier

IIS	Internet Information Services	RTSP	Real Time Streaming Protocol
IoT	Internet of Things	RTMP	Real Time Messaging Protocol
IP	Internet Protocol	RTMPE	Real Time Messaging Protocol Encrypted
IPTV	Television over Internet Protocol	RTMPS	Real Time Messaging Protocol Secure
ISO	International Organization Standardization	RTMPT	Real Time Messaging Protocol Tunneled
J2EE	Java 2 Enterprise Edition	SD	Standard Definition
JSON	JavaScript Object Notation	SDL	Simple DirectMedia Layer
Kbps	Kilobits per second	SDK	Software Development Kit
LAN	Local Area Network	SDT	Service Description Table
LTE	Long Term Evolution	SSL	Secure Socket Layer
MB	Mega Bytes	STB	Set-Top-Boxes
Mbps	Megabits per second	SWF	Shockwave Flash Format
mdat	Movie data	SSH	Secure Shell
moof	Movie Fragment	TCP	Transmission Control Protocol
moov	Movie Metadata	tdt	Time Date Table
MPEG	Moving Pictures Experts Group	TDT	Televisão Digital Terrestre
MPEG-2	Moving Pictures Experts Group-2	TIF	TV Input Framework
MPEG2-TS	MPEG-2 Transport Stream	TLS	Transport Layer Security
MPD	Media Presentation Description	TTML	Timed Text Markup Language
MP3	Motion Picture Experts Group Layer 3	TV	Television
MP4	Motion Picture Experts Group Layer 4	UDP	User Datagram Protocol
MSE	Media Source Extension	UI	User Interface
MVC	Model-View-Controller	UTF-8	8-bit Unicode Transformation Format
ms	milliseconds	URI	Uniform Resource Identifier
NIT	Network Information Table	URL	uniform resource locator
NPAPI	Netscape Plugin Application Programming Interface	USB	Universal Serial Bus
OS	Operative System	UTC	Universal Time Coordinated
OTT	Over-The-Top	VOD	video-on-demand
PAT	Program Association Table	WebRTC	Web Real Time Communication
PMT	Program Map Table	WebVTT	Web Video Text Tracks
QoS	Quality of Service	WWDC	Worldwide Developers Conference
RAM	random access memory	WMV	Windows Media Video
RTP	Real Time Protocol	XML	Extensible Markup Language
RTCM	Rede Temática de Comunicações Móveis	3GP	Third Generation Partnership
RTCP	Real Time Control Protocol	3GPP	Third Generation Partnership Project

INTRODUCTION

Television over Internet Protocol (IPTV) is a method for streaming television channels over the Internet. It makes use of Internet Protocol (IP) networks for content transmission. This chapter presents an overview on the importance of the multimedia transmissions Over-The-Top as well as the motivation behind this dissertation. After presenting the motivation section, the goals and the document structure are also explained.

1.1 MOTIVATION

Nowadays, the Internet is a globally known and used infrastructure, from schools, to people's homes and enterprises, thus it became a commodity and the main way to obtain information.

Xiaojun (2007) concluded that "the current Internet infrastructure is capable of providing the performance requirements of IPTV at low cost and with minimal dedicated infrastructure" [1], which motivated both developers and enterprises to implement solutions to streaming Television (TV) content OTT.

While in the past, the Internet was used mainly for commercial use, nowadays the increasing number of users created benefits for the community. These benefits can be related to the share of knowledge or to the value generated by people's information. Consequently, the number and the value of the Internet services (paid or free) also increased. In video and audio streaming services, many implementations are closed to developers and the services must be paid by users creating an obstacle when defining standard specifications and methods that the community can use to create its own services.

Despite the growth in terms of services and infrastructure, the migration from traditional Television services is not linear. A study performed in the last quarter of 2016 by Nielsen [2], shows a decline in traditional TV among the 18-24 population, but still a mean of 2 hours and 14 minutes per day, in the analysed quarter. The same study provided an analysis of the weekly time spent among the population through different informational services. For older populations, with ages comprised between 35 and 49 years old, the time spent in Live or DVR content performs a total of 30 hours and 55 minutes, a lot more when comparing to the time spent on video on computer, smartphone or other multimedia devices that together represent 6 hours and 3 minutes of time spent. In conclusion, this study shows

that, in the current days, despite the evolution of multimedia services, the television represents a big slice of the communication channels to the population, which motivates the integration of the Live and on demand television services on the Over-The-Top type of delivery.

HTTP streaming is a technique, growing at high pace, that is inserted in the OTT transmission and that enables the use of an Internet connection to sequentially download small media files than can be played continuously by the end players, and can be used in order to provide a modern and reliable IPTV Over-The-Top service.

1.1.1 MOBILE MARKET IMPORTANCE

The fast growth and the developments made on mobile devices, such as smartphones, and other smart devices, like smart Televisions and set-top-boxes, motivates the creation of multi-platform solutions that can target them. These solutions should create familiarity among the devices. In a world where almost everything is connected, it is important to take advantage of these connections to provide users with interactive and updated content, such as multimedia. It is also important to combine them with other entertainment resources that will provide a richer user experience.

1.1.2 OVER-THE-TOP GROWTH

The growth of the OTT services shows the importance of this subject for the future of the Internet and also suggests an increase of the generated data transmissions through this type of content delivery solution. One market study published by *marketsandmarkets* [3], in 2016, estimated that, in 2020, the value of these services will reach approximately 62 billions dollars. An increase of 34 billions dollars when compared to the year of 2015. The same source states that "OTT TV is the digital turning point for the TV industry. It affects business models and success factors along with all steps of the value chain of the TV industry from content creation to distribution. Over-the-top is used for the delivery of audio, video, and other media over the Internet without the involvement of a multiple-system operator to control or distribute the content.", showing that it is highly probable that the future of Television relies on OTT.

Some key players in the market that contribute for this large scale growth are Youtube, Netflix and Hulu, just to name a few [4]. Youtube is one of the most popular video sharing applications globally, with a project created in 2005 and bought by Google in 2006[5]. Netflix is, a paid service that offers users a varied library of movies and television series. Similar to Netflix, Hulu is a service that offers Television shows and movies but with some differences: it offers a small content library, paid and free subscription plans and a limited device compatibility (E.G. doesn't work with Apple devices)[6].

One study provided by Sandvine [7], in North America, shows that the Netflix dominates the downstream traffic with 35,2% and is followed by Youtube with 17,53%. Together they represent more than 50% of the data traffic during the peak periods. The figure 1.1 shows the complete information about this subject. As we can notice, the Real-Time Entertainment is responsible for 72,72% of the traffic during peak periods.

Upstream		Downstream		Aggregate	
BitTorrent	18.37%	Netflix	35.15%	Netflix	32.72%
YouTube	13.13%	YouTube	17.53%	YouTube	17.31%
Netflix	10.33%	Amazon Video	4.26%	HTTP - OTHER	4.14%
SSL - OTHER	8.55%	HTTP - OTHER	4.19%	Amazon Video	3.96%
Google Cloud	6.98%	iTunes	2.91%	SSL - OTHER	3.12%
iCloud	5.98%	Hulu	2.68%	BitTorrent	2.85%
HTTP - OTHER	3.70%	SSL - OTHER	2.53%	iTunes	2.67%
Facebook	3.04%	Xbox One Games Download	2.18%	Hulu	2.47%
FaceTime	2.50%	Facebook	1.89%	Xbox One Games Download	2.15%
Skype	1.75%	BitTorrent	1.73%	Facebook	2.01%
	69.32%		74.33%		72.72%




Figure 1.1: Top 10 peak period applications responsible for Real-time entertainment data, in North America.

1.1.3 PORTUGAL'S AVAILABLE TV PLATFORMS

In the twenty-sixth of April of 2012, the Portuguese analogic television consumers were forced to migrate to the new digital television platform, a service called Televisão Digital Terrestre (TDT). That migration forced users to buy dedicated digital television boxes if their televisions weren't compatible with that technology. The hardware behind the TV must be in conformity with the Digital Video Broadcasting-Terrestrial (DVB-T) technology and the norm Advanced Video Coding, MPEG-4 Part 10 (H.264). Figure 1.2, shows the number of TDT capable devices sold monthly, in the period comprised between June 2009 and February 2013. As we can see, users bought more devices in the last quarter of 2011 and in the first quarter of 2012, close to the transition date, showing the investment made not only by the government to migrate the platform but also by the users to upgrade their hardware [8].

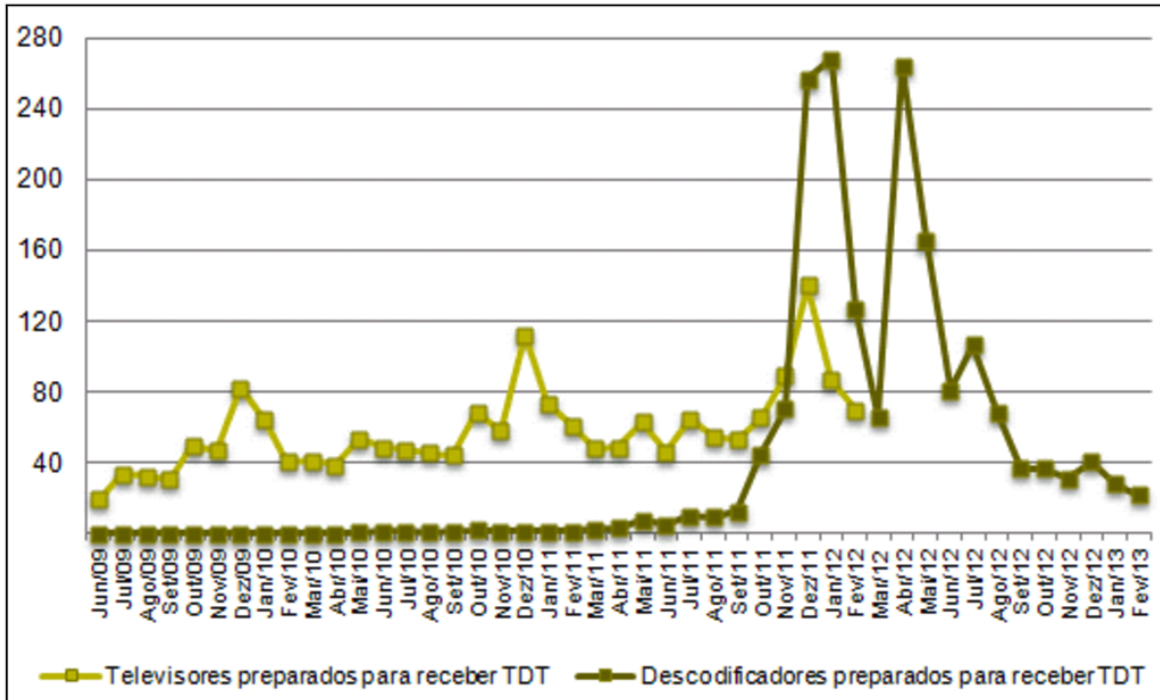


Figure 1.2: Digital Television ready equipments sold monthly. At light green are presented the metrics for the Televisions and at dark green are represented the decoder boxes.

Despite the migration efforts already stated, the number of devices that are able to decode and play with TDT hasn't grown. This is a limitation for users because they are forced to reproduce the public channels on their upgraded Televisions or DVB-T compatible devices and therefore, there is no mobility in that platform that goes along with the growth of the mobile market.

Some channel vendors already identified this problem and created their solutions to move towards the market tendency on the OTT transmissions. RTP Play is one example of the effort made by the state's television company.

The problem behind these implementations is the dispersion between multiple companies with their own websites and their own business model (with advertisements, by example).

1.2 OBJECTIVES

As stated in the previous section, the dispersion between vendor implementations allied to the fact that not all the channels are being broadcasted over the Internet, creates a window for improvement, to **create an infrastructure that serves all the public channels Over-The-Top**. This infrastructure should also be enriched with other multimedia data other than video. Those enrichments should extend the streams with relevant information about what is being broadcasted and other past and future programming information, creating a better user experience. One of those key features that boost the user experience is the seek back of the reproduction of a channel, with the goal of watching video-on-demand (VOD) programs that were already broadcasted.

Electronic Programming Guide (EPG) is the service that provides all the required information about the channels programming and typically is presented by a graphical interface that allows users to navigate through the channel's programming, from start time and duration to the program description. The typical backend of this graphical interface presupposes the share of a file containing all the information for all the channels, present on the EPG. One of the goals is to create an Application Programming Interface (API) that can deliver the information required but, in a customised way, avoiding the share of files. Posteriorly, this graphical interface will support the user interaction with some actions that can be made over the programming like to restart the reproduction of a certain program or to see what is being broadcasted.

With increasing importance of mobile devices and the recent improvements made on the mobile connectivity to Internet, it is important to guarantee that streaming platforms are compatible with as much devices as possible. The goal on this subject is to create a multi-platform application that is able to target as many devices as possible, from Set-Top-Boxes (STB) to smartphones and computers.

1.3 THESIS OUTLINE

The next chapters that compose this thesis are organised in the following sequence:

- **Chapter 2** - Presents a description of the state of the art, with the present standards and specifications, involved in the previously presented problematic, for performing OTT content delivery. This evaluation starts by analysing the Digital Video Broadcasting (DVB) technology and proceeds to the analysis of the Hyper-Text Transport Protocol (HTTP) streaming, with the main focus on the adaptive streaming techniques, standards and specifications.
- **Chapter 3** - Describes the ideal scenario on the delivery OTT multimedia content , the requirements, goals, as well as, the solution proposed to implement; An overview over some implementations and some relevant tools is presented. For tools with the same purpose is also provided a summary comparison between them.
- **Chapter 4** - Presents the implemented solution and provides an analysis of the DVB-T signals received on the server;
- **Chapter 5** - Describes the test challenges and the results of the implementation;
- **Chapter 6** - Supplies the final conclusions about the developed work, as well as, the contributions made on external projects, about the streaming subject.

STATE OF THE ART

This chapter aims to provide a detailed analysis of the state of the art in the IPTV development area. For each domain is firstly described the techniques and methods used nowadays and then is presented some of the most used technologies.

2.1 INTRODUCTION

In order to understand the challenges of Over-The-Top (OTT) content delivery, especially in Live and On-Demand TV scenarios, a detailed analysis and characterization of usage and technologies is presented. The infrastructure to implement is composed of 2 major entities, the server and the client. The first is focused on the delivery of multimedia data and other complementary services. The client is the entity responsible for consuming the data provided by the server and present functionalities to the users, based on that information.

This analysis starts with a view over the Television evolution since Baird's discovery that originated the Mechanical Television Era until our days, on section 2.2.

Next, in section 2.3, is given an explanation of the Digital Video Broadcasting standards and their importance for the television transmission of the public channels of each country.

The most common conventional streaming technologies are presented on section 2.4 and section 2.5 presents an insight of the HTTP streaming as well as the available standard and specifications that follow this technique. In the end of the section is also presented an overview of this analysis, coupled with a comparison between the most relevant and presented solutions.

On section 2.6, the mechanism of delivery of the content through HTTP is explained and are presented some of the implementations available in the market.

In order to create an adaptive streaming solution, it is demanding to create multiple representations of the same multimedia data to be served. On section 2.7 this concept is analysed, as well as some tools that allow performing the media preparation into multiple representations.

The previous sections conclude the analysis of the key themes in the pipeline of an adaptive streaming server over HTTP. It is important to enrich the service provided to clients with data that complements the audio/video streams. Those enrichments, in Live and on-demand streaming, are fundamental to provide user ability to navigate through the channels in terms of time and to gather

more information about what is broadcasted. In that way, the section 2.8 provides insights about this theme and presents the most popular defined format to share this information between entities.

The numerous different types of devices, allied to the improvements made on the Internet mobility creates the need to consider mobile platforms and their operative systems as an important slice of the target devices for audio/video playbacks. The section 2.9 is dedicated to that analysis, providing insights about the limitations in the streaming protocols and codecs, as well as the recommendations to achieve a good quality playback on the devices running those platforms. Additionally, an overview of the Android's TV input framework is presented.

There are many platforms and ways to develop multimedia applications. The recent developments on web platforms, namely the inclusion of Media Source Extension (MSE) on browsers, motivates the development of cross-platform applications over browsers, similarly to the Youtube's implementation (explained in section 2.11.1). On section 2.10 are described some frameworks which are the popular solution to develop web applications, in addition to some players and libraries dedicated to the adaptive HTTP streaming playback.

Finally, in section 2.11 are presented successful solutions in the market like Youtube and Kodi, as well as some server architectures implemented namely Restreamer and Tv Headend.

2.2 THE EVOLUTION OF TELEVISION

The period before 1935, known as the "Mechanical Television Era", introduces the first generation of TV sets where Televisions weren't completely electronic. According to the *tvhistory* website ¹, in the year 1926 was made the first public demonstration of this technology by the Scottish Mr John Baird. In 1930, the Baird "Televisor" arrived at the United Kingdom as the first mass produced "scanning disc television".

Since the first days of this invention, lots of developments and innovations were made. For example, on June 25 1951, the CBS performed the first commercial color broadcast from New York. Nowadays, the evolution and developments on this subject belong to an important technology market where the main question is how to improve the monetization ecosystem and, at the same time, how to provide better services to clients. The evolution of parallel technology like the Internet attracted a massive audience through the years, in a way that this infrastructure allows the television developers and companies to think of new ways to reach audiences across the screens.

In the TV "world" there are three major entities that, combined, compose the Television transmission cycle. The first entity is composed of the channel's owners that can have one or multiple television channels, followed by the distributors or service providers that deliver services to the third entity, the clients or channel consumers. Channel owners and service providers current goal is to provide live and on-demand content anywhere and anytime, in the most different devices as possible.

Despite this innovation being great for the channel's viewers, it results in audience fragmentation for the advertisers that will have to discover new ways to reach their audience, and in a monetization challenge for the channel distributors. The image 2.1 shows the relation of how advertisers spent their budget on different segments of media (TV, radio, digital and others). As we can see, by 2018 the forecasts point to a bigger investment on digital advertising than the traditional television in the United States [9].

¹<http://www.tvhistory.tv/pre-1935.htm>

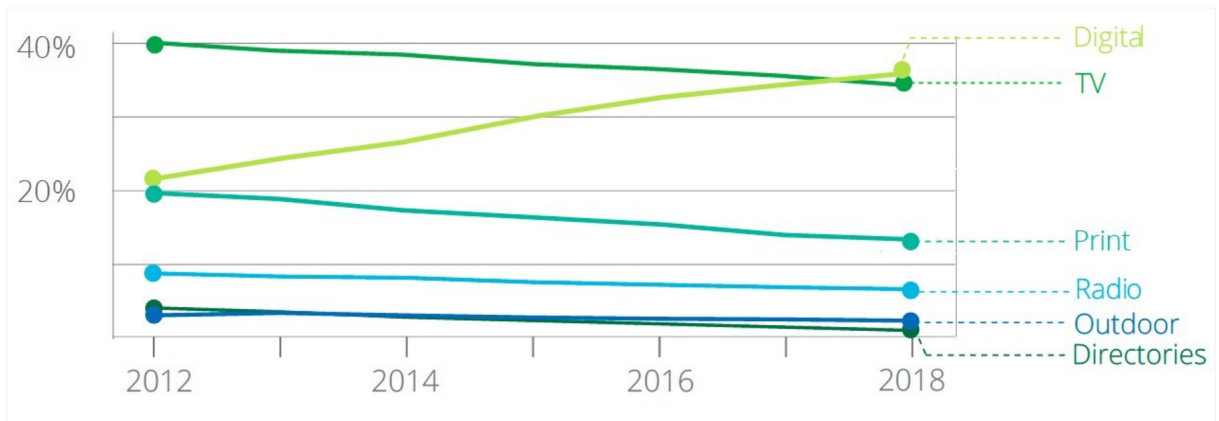


Figure 2.1: Investments made on different advertisement segments with forecast to 2018, in the United States.

With the investments made on digital advertisements, the demand of the target users to watch TV anywhere at anytime and considering the evolution of the Internet infrastructure, it is possible to state that the future of the Television is the Digital TV.

2.3 DIGITAL VIDEO BROADCASTING

As stated in the chapter 1.1, the Television services continue to be of great importance in people's daily lives. With the evolution from analogic Television to digital platforms, the need to define standards that cover the multiple transmission possibilities, from satellite to terrestrial, becomes more evident. DVB is an "industry-led consortium of the world's leading digital TV and technology companies, such as manufacturers, software developers, network operators, broadcasters and regulators, committed to designing open technical standards for the delivery of digital TV." ². And it is of great importance to guarantee quality services of TV transmissions.

Digital Video Broadcasting standardizes a type of digital delivery that uses MPEG media encoding for transporting video. The broadcast can be made over satellite, terrestrial, cable or telecom. Through this method is possible to build streams by multiplexing broadcasting signals from different transmission methods. DVB-T is the terrestrial version of this protocol and the respective board's utility is to receive signals over the air and provide the media to our computational system or other media receptor [10][11]. Actually, this version has an improved specification, known as Digital Video Broadcasting-Terrestrial 2 (DVB-T2), that provide a more robust reception and a significant bitrate increment [12] [13].

There are other types of DVB signals like Digital Video Broadcasting-Cable (DVB-C), Digital Video Broadcasting-Satellite (DVB-S), Digital Video Broadcasting-Handheld (DVB-H) and others [14].

In order to capture the public digital channels we need to choose the appropriate type of board accordingly to the broadcasting type and encoding adopted by each country. For instance, in Portugal it is implemented the DVB-T as the transport protocol with Advanced Video Coding, MPEG-4 Part 10 and AAC, but on the other end, in the United Kingdom the DVB-T and the DVB-T2 are both used to encode and compress the digital satellite communications [15][16][17].

²<https://www.dvb.org/>

2.4 CONVENTIONAL STREAMING

Conventional IPTV services rely mostly on three popular kinds of protocols that can be used for media streaming, Real Time Protocol (RTP)/Real Time Control Protocol (RTCP), Real Time Streaming Protocol (RTSP) and Real Time Messaging Protocol (RTMP).

REAL TIME PROTOCOL

RTP creates a standard format for delivering media over the Internet. It is mainly based on User Datagram Protocol (UDP) although it can work with Transmission Control Protocol (TCP). Using UDP makes this protocol simple and fast but it lacks control over the network (E.G. network congestion issues or Quality of Service (QoS)) [18][19]. RTCP protocol helps to control the network congestion by sending, periodically, control packets to the entities of a connection. These control packets provide basic feedback on the quality of service (E.G. packet counts, round-trip time and Jitter) [20].

REAL TIME STREAMING PROTOCOL

RTSP is an application-level protocol that can make use of RTP and RTCP to deliver data with real-time properties over the Internet, providing control functionalities like play/pause, fast forward, rewind and others, similar to the functionalities present on a DVD player.

RTSP is a client-server protocol that maintains a session between those entities. During an RTSP session, a client may open or close, one or many transport connections to the server, in order to handle RTSP requests[21]. The steps involved using this technology for streaming are as follows:

First, the client establishes a connection (TCP) to the server, typically on TCP port 554. Then, the client will issue a series of RTSP header commands, each of which is recognised and acknowledged by the streaming server. The client will, within these commands, describe the details of the session requirements (E.G. RTSP version support, transport used for the data flow and others). This information is passed using the DESCRIBE and SETUP headers. Also, a Session ID is created, transmitted by the server and used to identify the stream in further data exchanges. After the completion of this initial negotiation, the client will issue a PLAY command to instruct the server to start the media stream. In order to close the stream, the client sends a TEARDOWN command with the Session ID to the server.

REAL TIME MESSAGING PROTOCOL

RTMP is a protocol designed by Adobe, for transmission of data between Adobe Flash technologies (E.G. Flash Player). Nowadays, it is an open specification to create services that enable delivery multimedia data. The delivery is made by using Action Message Format (AMF), Shockwave Flash Format (SWF), Flash Live Video (FLV) or F4V formats. Comparing to RTP and since RTMP relies on TCP, it has better control over congestions on the network[22]. This protocols works as follow:

The player first contacts the server to establish an RTMP connection. After the initial "handshake", the player requests a specific stream, for example – a live video. Once the server receives this request, the media is sent directly over the RTMP connection, in fragments. The same stream can be sent to multiple clients that request it. Additional media servers can be chained together, improving capacity and scalability.

Real Time Messaging Protocol has multiple variations, like the Real Time Messaging Protocol Encrypted (RTMPE) which is RTMP encrypted by an Adobe encryption mechanism or the Real Time Messaging Protocol Secure (RTMPS) which is RTMP over TLS/SSL[23].

2.5 HTTP STREAMING

Internet access is, nowadays, a commodity to most people, in a world where the mobile data traffic is growing at large steps. According to a recent study [24], in the end of 2016, mobile data traffic increased by 63 percent over the same period of 2015, reaching 7.2 exabytes per month of global data traffic. The same study predicts that, in 2021, global mobile data traffic will reach 49 exabytes, per month, and represent 20 percent of total IP traffic.

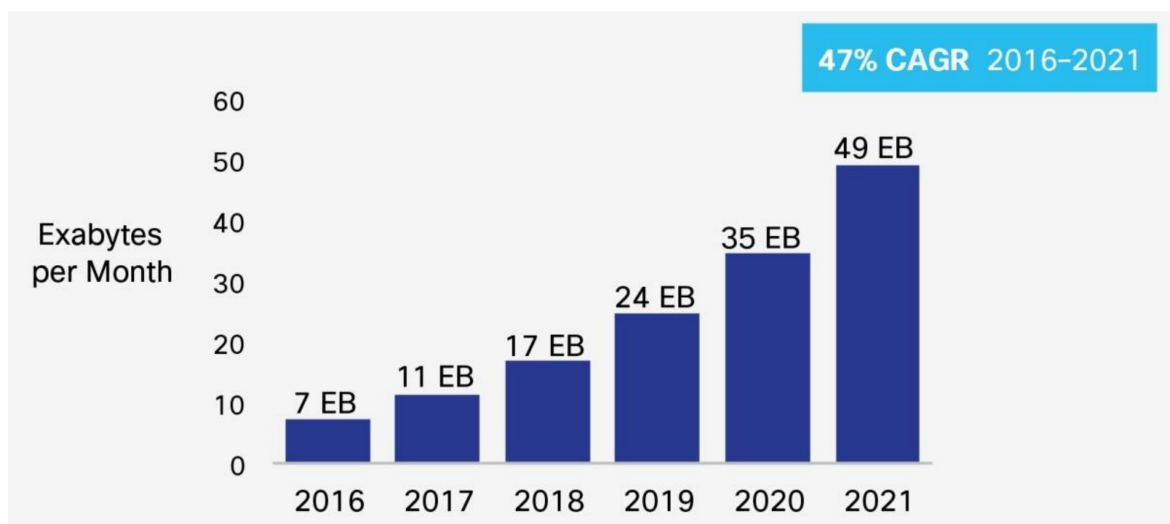


Figure 2.2: Cisco’s forecast of mobile data traffic for the period of 2016-2021[24].

Moreover, cloud services and platforms are expanding quickly and the users lost the sense of downloading media and expect an easier way to reproducing it, using stream technologies. In many cases, download and watch it after is not viable, mainly because it does not allow users to watch media in real time, so streaming live events is an interesting capability, especially for live events or television broadcasts.

For a few years now, Content Delivery Networks (CDNs) have been replacing IP managed networks in the task of delivering media. CDNs improves accessibility and efficiency in the content replication management and simultaneously reduces the load on the origin server. In CDNs architecture, the content is shared between the origin server and cache servers. The cache servers can be located around the globe to be closer to users and provide faster web services to them. To accomplish a good streaming experience is important to have a service that provides a good media quality, avoiding as fewer media losses as possible [25] [26] [27].

Traditional streaming relies on stateful protocols like Real Time Protocol and Real Time Streaming Protocol, that maintain a streaming session and keep track of the clients until they disconnect. This conventional protocol is effective, with low overhead but only works well for managed networks and that is a real problem since nowadays the Internet relies mostly on Content Delivery Networks. On

the other hand, HTTP streaming is stateless, so when a client asks for a media object, the server just sends it and terminates the session. There are some advantages of using HTTP streaming [28] [29]:

- Passing more easily through firewalls;
- HTTP streaming is being adopted massively on open Internet;
- Some implementations, depending on the transmission protocol, can deliver HTTP content on a regular server or CDN without the need of dedicated hardware;
- The client is responsible to control the media requests. The server just has to deliver/send the files and provide the required metadata;
- HTTP streaming provides a way to the client to choose the initial bitrate accordingly to the available bandwidth;
- Server doesn't need to manage a dedicated session for each client;

One study [30], states that, in order to achieve adaptivity to clients and respective network conditions, the server should generate multiple media representations of the original media and generate the appropriate metadata which contains the representation characteristics (bitrate, etc.) Then, is the client's responsibility to check his conditions and request the appropriate media in the appropriate time. It is also important that the client requests an initial amount of data to buffer, that way the media reproduction becomes less sensitive to throughput fluctuations. The problem about the buffering is the initial delay that it causes, especially if in a live scenario.

2.5.1 LIVE TELEVISION

There are two types of streaming services: Live video streaming, that delivers multimedia live content, and VOD streaming, which serves pre-recorded multimedia when clients demand it. Between those two, the live streaming scenario is the one that implies more challenges to the HTTP streaming. The most critical ones are the latency and the delay. High latency or high delay can compromise the user experience because it creates a time slot between when the event occurs and when it is shown to the viewer. One example could be the delay of a soccer game live broadcast.

As Wei stated [31], "In HTTP streaming, the video content is typically divided into multiple segments with predefined fixed intervals, and each video segment is regarded as a separate resource for HTTP requests and responses. As a result, the video content cannot be delivered and played back until the video segment it belongs to is fully generated. Therefore, in the live streaming case, the live latency is at least one segment duration even without considering any encoding/decoding, buffering, and network delays.". Decreasing the segments duration is possible to reduce the delay but, as a counterpart it increases the number of requests to the server and therefore compromises the scalability of the same.

Besides the typical behaviour presented, each standard or specification for HTTP streaming has some particularities that can improve or not the delay on multimedia streaming. For instance, some have the capability to receive byte range requests that enables to download initially part of the segments thus reducing the initial delay. Next, there will be presented the most known and used HTTP streaming technologies.

2.5.2 DYNAMIC ADAPTIVE STREAMING OVER HTTP

The Dynamic Adaptive Streaming over HTTP (DASH) is a standard that enables a way to stream Over-The-Top multimedia data. This standard aims to change the current market, where most of the commercial implementations are closed and with their own formats and protocols, improving interoperability between clients and server vendors [32].

DASH provides a crucial feature to any modern streaming player, the HTTP adaptive streaming (HAS), that enables the clients to dynamically switch between video/audio qualities accordingly to his needs [26]. Figure 2.3 illustrates the metadata file received by the clients that enable this feature.

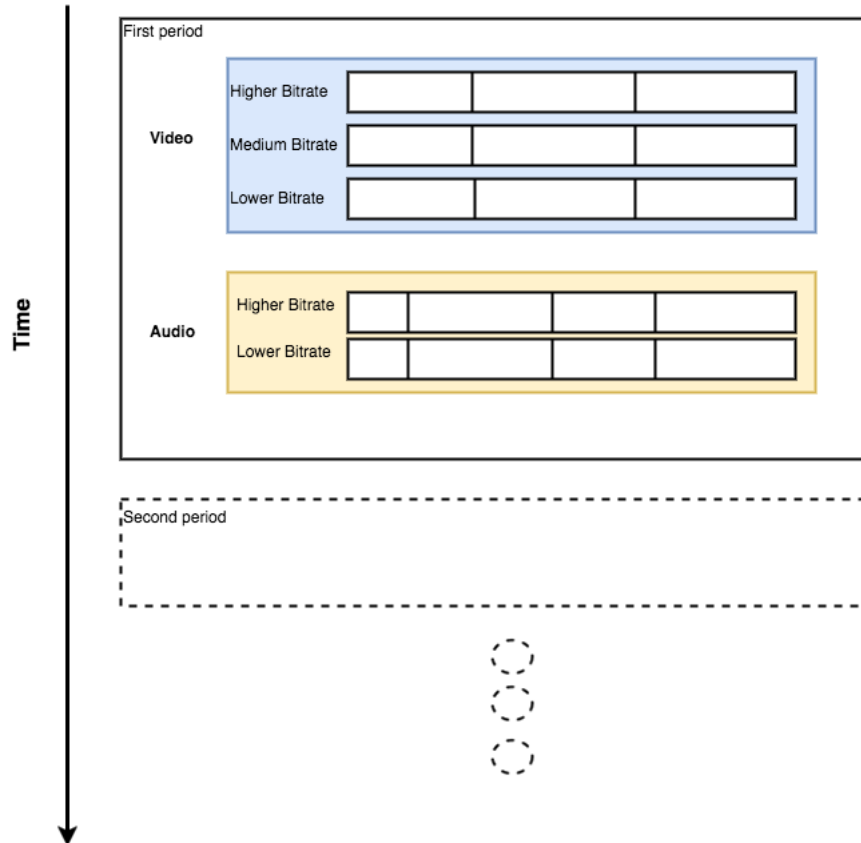


Figure 2.3: Simple illustration of the DASH MPD format.

In this example, the multimedia content consists of both video and audio. The video is encoded at three different bitrates and the audio available in two sound qualities. It is the client's/ device's responsibility to request video and audio levels. Let's assume that the device starts requesting the Medium bitrate video quality and the Higher bitrate audio quality but then realizes that the bandwidth has increased so, at the next request, it switches to the Higher video and audio bitrates. After requesting the Higher video and audio bitrates, a low bandwidth is detected so the client switches, in the next request, to the Lower bitrate video quality and keeps the Higher bitrate audio.

More complex examples can include, 3D media content, Trick-mode (used for playing video in reverse mode), switching between multiple camera views, multiple audio languages each one with his encoding parameters, streams with subtitles and captions, ad insertion, mixed-streaming, low-latency live streaming and a large set of other possible applications [32].

This protocol specification was developed by the Moving Pictures Experts Group (MPEG) with collaboration from other entities, namely the Third Generation Partnership Project (3GPP) [32]. 3GPP Adaptive HTTP Streaming specifies the format of media segments, their delivery protocol and the syntax of the Media Presentation Description (MPD). Although, details on content provisioning (segments size, associated bitrates,...), client behaviour (how to download segments or switching between different representations) or the transport of the MPD file (can be delivered by other means than HTTP) are not covered by this specification. Likewise, the DASH standard doesn't cover the delivery method of the MPD file, media encoding formats or any behaviour on the client side [28] [33].

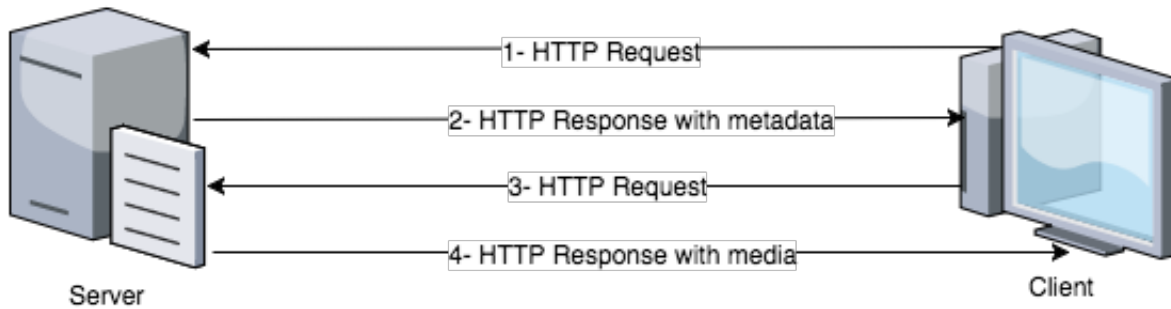


Figure 2.4: Generic behavior of the DASH protocol

As we can see in the figure above, the streaming clients connect to the server in order to obtain information about the media delivery in a file typically called meta-data, or MPD file in DASH scope, that provides information about the streaming profiles and describes the available media content. There are other ways to deliver the MPD file, using email, broadcast, or other file transport. After obtaining and parsing the MPD file, the client can collect information about bandwidth levels, resolutions, media types and locations, accessibility features and Digital Rights Management (DRM). Then, the client can make an HTTP GET to fetch the media segment from the server [32].

```

<MPD xmlns="urn:mpeg:DASH:schema:MPD:2011" xmlns:ytdrm="http://youtube.com/ytdrm"
mediaPresentationDuration="PT0H3M1.63S" minBufferTime="PT1.5S"
  profiles="urn:mpeg:dash:profile:isoff-on-demand:2011" type="static">
<Period duration="PT0H3M1.63S" start="PT0S">
<AdaptationSet>
<ContentComponent contentType="video" id="1" />
<ContentProtection schemeIdUri="com.youtube.clearkey">
<ytdrm:License keyid="60061e017e477e877e57d00d1ed00d1e"
key="1a8a2095e4deb2d29ec816ac7bae2082"/>
</ContentProtection>
<Representation bandwidth="4190760" codecs="avc1.640028" height="1080" id="1"
mimeType="video/mp4" width="1920">
<BaseURL>car_cenc-20120827-89.mp4</BaseURL>
<SegmentBase indexRange="2755-3230">
<Initialization range="0-2754" />
</SegmentBase>
</Representation>
<Representation bandwidth="869460" codecs="avc1.4d401e" height="480" id="3"
mimeType="video/mp4" width="854">
<BaseURL>car_cenc-20120827-87.mp4</BaseURL>
<SegmentBase indexRange="2789-3264">
<Initialization range="0-2788" />
</SegmentBase>
</Representation>
</AdaptationSet>
<AdaptationSet>
<ContentComponent contentType="audio" id="2" />
<ContentProtection schemeIdUri="com.youtube.clearkey">
<ytdrm:License keyid="60061e017e477e877e57d00d1ed00d1e"
key="1a8a2095e4deb2d29ec816ac7bae2082"/>
</ContentProtection>
<Representation bandwidth="127236" codecs="mp4a.40.02" id="6" mimeType="audio/mp4"
numChannels="2" sampleRate="44100">
<BaseURL>car_cenc-20120827-8c.mp4</BaseURL>
<SegmentBase indexRange="2673-2932">
<Initialization range="0-2672" />
</SegmentBase>
</Representation>
<Representation bandwidth="31749" codecs="mp4a.40.03" id="8" mimeType="audio/mp4"
numChannels="1" sampleRate="22050">
<BaseURL>car_cenc-20120827-8b.mp4</BaseURL>
<SegmentBase indexRange="2673-2932">
<Initialization range="0-2672" />
</SegmentBase>
</Representation>
</AdaptationSet>
</Period>
</MPD>

```

Code 1: Example of a MPD file

As we can see in Code 1, the MPD file is an Extensible Markup Language (XML) document that consists of a set of periods. Each period has a start time and duration tags and one or multiple adaptation sets. Each of these adaptation sets provides a reference to multiple media components and its multiple encoding alternatives. In this example, an adaptation set contains various bitrate levels of

the media's video component. The next adaptation set contains the different audio bitrates of the same media. Each of these alternatives within the adaptation set is called representation and gathers a set of parameters (bitrate, resolution and others) and a set of media segments. A segment has an associated uniform resource locator (URL) for the server where the player should download the data [33][32].

Also, Dynamic Adaptive Streaming over HTTP addresses some interesting additional features like the capability for the client to download the manifest in a fragmented way, measure some quality metrics and report back the result to the server, the inclusion of the Universal Time Coordinated (UTC) offset in the segments or signaling the duration of the next segment in the current one (useful for live streaming scenarios where the segments duration can vary) [32].

2.5.3 HLS

HLS is an Apple's open protocol for stream live or on-demand audio/video media. It was originally created to allow streaming data to Apple devices. With the HLS specification it is possible to stream from an ordinary web server, such as a Linux-based machine. HLS supports multiple encodings of the same media (E.G. different bitrates) to provide the client with the ability to adapt to bandwidth changes. In this protocol it is possible to encrypt media and authenticate users over Hyper-Text Transport Protocol Secure (HTTPS), useful when we need to protect the streams [34] [35].

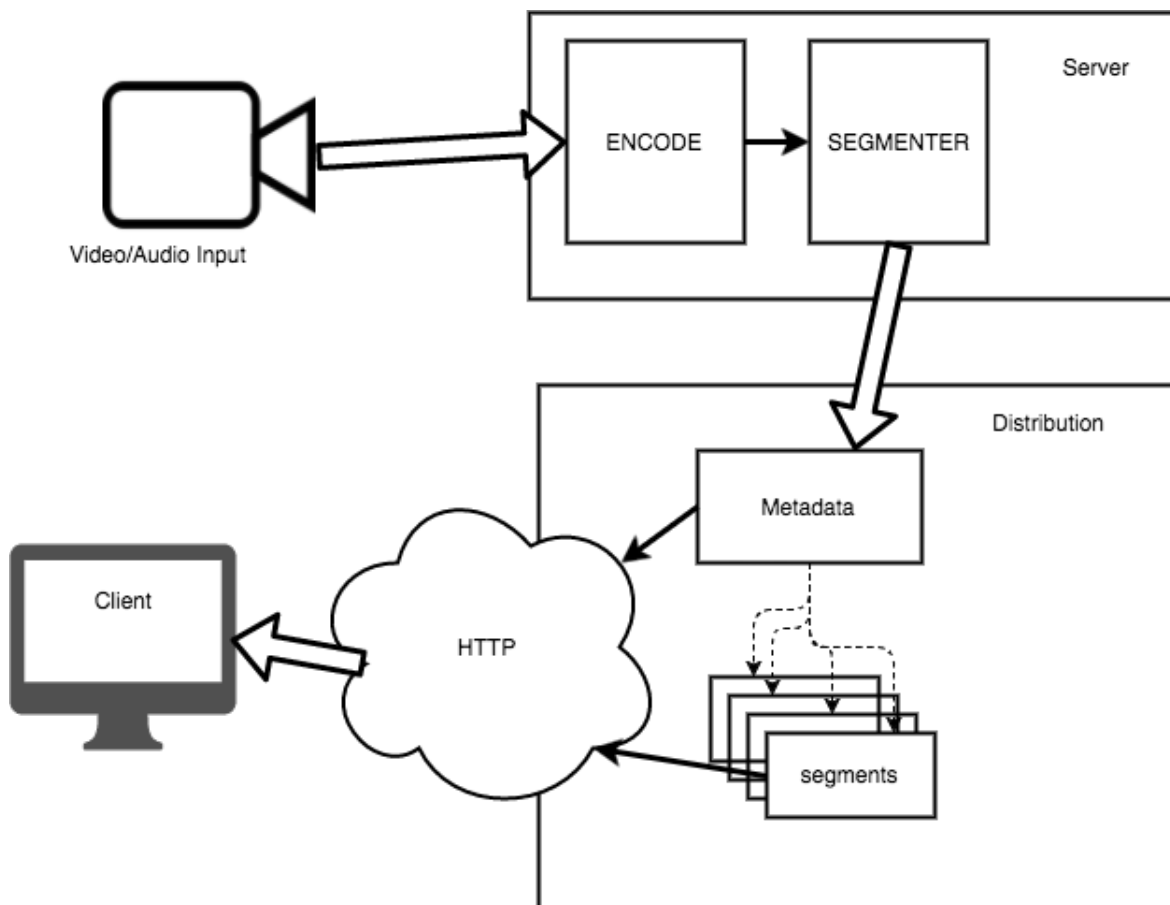


Figure 2.5: HLS global architecture.

As we can see in Figure 2.5, there are three major components on HLS architecture:

- Server Component;
- Distribution Component;
- Client Player

Starting from the server component, its job is to take the source, encode the media, segment it and prepare for delivery. The distribution component takes the media output from the server, generates a metadata file and serves the files over HTTP or HTTPS. The client has the responsibility to fetch the metadata, download the media and serve the user with a continuous stream.

On HLS, the meta-data is represented on an extension of the M3U playlist format(.m3u). An M3U playlist is an UTF-8 text file where each line is an Uniform Resource Identifier (URI), blank line or starts with an '#' to represent a comment or a tag. Each tag must start with "#EXT". An URI identifies the variant playlist on the media segment and the respective tags [36][34] [35].

```
#EXTM3U
#EXT-X-STREAM-INF:BANDWIDTH=1280000
low/audio-video.m3u8
#EXT-X-I-FRAME-STREAM-INF:BANDWIDTH=86000,URI="low/iframe.m3u8"
#EXT-X-STREAM-INF:BANDWIDTH=2560000
mid/audio-video.m3u8
#EXT-X-I-FRAME-STREAM-INF:BANDWIDTH=150000,URI="mid/iframe.m3u8"
#EXT-X-STREAM-INF:BANDWIDTH=7680000
hi/audio-video.m3u8
#EXT-X-I-FRAME-STREAM-INF:BANDWIDTH=550000,URI="hi/iframe.m3u8"
#EXT-X-STREAM-INF:BANDWIDTH=65000,CODECS="mp4a.40.5"
audio-only.m3u8
```

Figure 2.6: HLS master playlist example.

Figure 2.6 shows an example of a master playlist. It references one or more M3U8 playlists and is used to reference different representations of the media. This example has a reference to three audio/video variants and another reference to an audio-only variant [35].

```
#EXTM3U
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:3
#EXTINF:9.009,
http://media.example.com/first.ts
#EXTINF:9.009,
http://media.example.com/second.ts
#EXTINF:3.003,
http://media.example.com/third.ts
#EXT-X-ENDLIST
```

Figure 2.7: HTTP Live Stream playlist example.

In Figure 2.7 we can observe a simple example of a M3U8 playlist with 10 second media files. This file must end in '.m3u8' and have "application/vnd.apple.mpegurl" content-type (if transferred over HTTP) [36].

TAG	Meaning
#EXTM3U	Indicates the file format. Must be on the first line of every Playlist or Master Playlist.
#EXT-X-VERSION	Indicates the compatibility version and must be included in every Playlist that contains incompatible tags with the version 1.
#EXTINF	Indicates the media segment duration.
#EXT-X-TARGETDURATION	Indicates the maximum segments duration.
#EXT-X-MEDIA-SEQUENCE	Indicates the sequence number of the first segment referenced in the Playlist file.
#EXT-X-ENDLIST	Indicates that no more segments will be added to Playlist.
#EXT-X-I-FRAME-STREAM-INF	Indicates a playlist that contains Intra-frames of the media presentation. Intra-frames or I-frames are independent frames and can be used to Trick-Play.
#EXT-X-STREAM-INF	Indicates a Variant Stream
#EXT-X-MEDIA	Indicates media playlists with multiple renditions of the same media. For example, audio with multiple languages

Table 2.1: Some of the tags present on a M3U8 file.

At the beginning of this analysis, HLS only supported MPEG-2 Transport Stream (MPEG2-TS). However, in Worldwide Developers Conference (WWDC) 2016, Apple announced HLS support for Fragmented MPEG-4 File Segments (fMP4). HLS also supports Packed Audio and WebTVV (a format that carries subtitles on the segments). The Packed Audio supported formats are Advanced Audio Coding (AAC), Audio Codec 3 (AC-3) and Motion Picture Experts Group Layer 3 (MP3). This protocol also enables users to play, pause and seek video. The segments duration can vary from a minimum of 1 second (the default is 10 seconds). The delay present on video playback has a heavy impact on delay. Shorter fragment size produces shorter delay but increases the overhead because more segments need to be downloaded [35] [36] [37].

Apple requires, on its app store applications, to use HLS when the video size exceeds 5 Mega Bytes (MB) or 10 minutes of length. Furthermore, a 64 Kilobits per second (Kbps) (or lower) alternate stream must be provided. Despite this requirement, it is fully supported on Safari for iOS and it is possible to run this protocol on almost every MSE browser using Javascript [34][38].

2.5.4 MICROSOFT SMOOTH STREAMING

Smooth Streaming is a Microsoft implementation for streaming media over HTTP. It uses the Motion Picture Experts Group Layer 4 (MP4) Part 14 (ISO/IEC 14496-12) file format to store the data. IIS Smooth Streaming is a hybrid media delivery method since it acts like streaming but is based on a progressive download of a MP4 file in a chunked way [39].

Such as HLS and DASH, this implementation is adaptive, offering clients the possibility to adapt to bandwidth changes. Although, a different approach is followed regarding the way that data is saved since each quality variant is stored in a single MP4 file. When a client requests a specific segment, the server dynamically finds the fragment, on the single stored file, and sends it to the client.

Smooth Streaming uses a wire format for the transport. It defines the structure of the segments sent from the server to the client and is a subset of MP4 since it uses MP4 in a fragmented manner. Figure 2.8 shows an example of an fMP4 file.

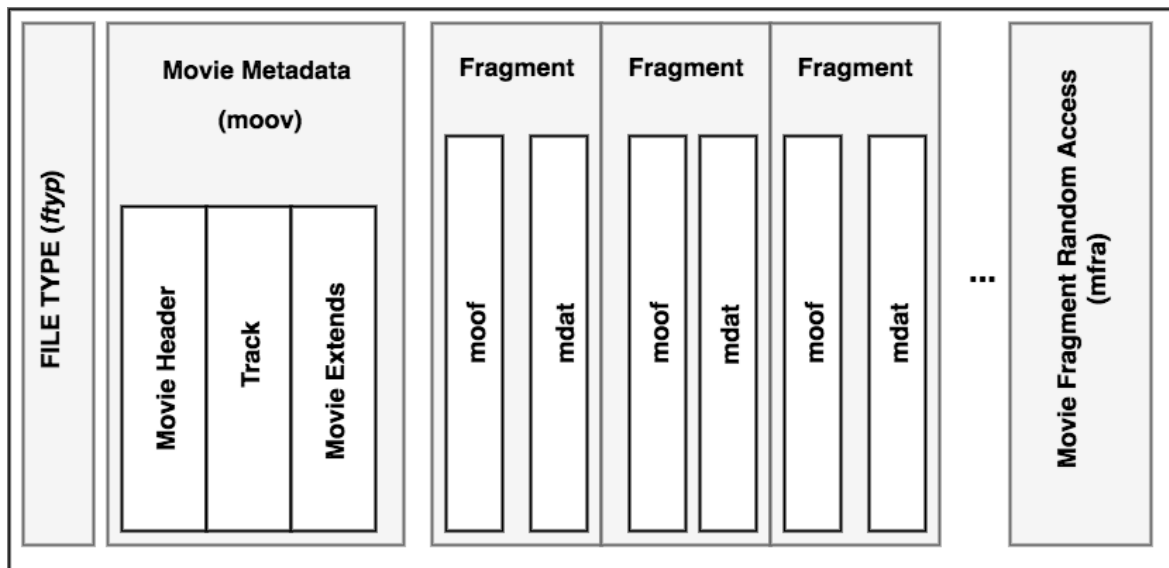


Figure 2.8: fMP4 file structure used in Smooth Streaming. *moof* boxes contain fragment metadata and *mdat* boxes the respective data.

The basic unit of these files is called a "box". Each box can contain audio or video data and metadata. The first box is a File Type (ftyp) box and is used to specify the version information. The next box is a Movie Metadata (moov) box that describes the movie header and media tracks of the file. The audio/video segments data are contained in Movie data (mdat) boxes. Typically, Smooth Streaming has a fragment for each 2 seconds of video/audio. Each of these boxes has a Movie Fragment (moof) box associated that contains the segment metadata and signalling information.

To start requesting the fragments, the client first needs to read the client-side manifest. This manifest is a XML file and should be downloaded at the beginning of the session. It complies information about bitrate levels, codecs, resolutions, captions and other relevant information[40].

Smooth Streaming is part of the IIS Media Services and has the purpose to stream on-demand media over HTTP. Although, IIS Media Services also has the Live Smooth Streaming, the live adaptive streaming for broadcast events. Live Smooth Streaming enables media delivery in multiple formats without re-encoding using, HLS for Apple devices and Smooth Streaming for Microsoft Silverlight clients [41][42].

OVERVIEW

In this section, a comparison is made between the previously stated streaming protocols. As we can see in table 2.2 , the DASH is the standard in the media streaming industry. It benefits from being

codec agnostic and provides the largest set of features comparing to the others presented. Nonetheless, since iOS has restrictions on playing streams that do not use HLS[34], the best solution is to create a solution that gathers both DASH and HLS. Smooth Streaming also makes use of HLS to enable media playback to iOS users and, in terms of performance is the best (comparing with DASH and HLS) in the task of adapting to network conditions [43][44][40][45] .

	DASH	HLS	SMOOTH
Manifest	MPD	M3U8	XML + .ism/ismc extension
Video Codecs	Agnostic	H.264	VC-1/H.264
Audio Codecs	Agnostic	MP3/AAC	AAC/WMA
Media containers	fMP4 MPEG2-TS	MPEG2-TS fMP4 (since v4)	fMP4 (with *.ismv/isma file extension)
Standard?	Yes	No	No
Deployable on a regular server	Yes	Yes	Yes
Fast channel switching	Yes	No	Yes
Support for multiplexed audio/video content	Yes	Yes	No
Full iOS support	No	Yes	Uses HLS

Table 2.2: Streaming protocols comparison.

2.6 SERVING THE CONTENTS

In the previous section, was presented an analysis of the HTTP streaming and the most known standard and specifications that support it. Independently of the method chosen, in order to provide to users any HTTP streaming, a way of serving data must be present. Since the contents to be served must be stored previously, it is necessary to have some entity that receives the requests and takes actions based on them. This entity can be a CDN or a typical HTTP server. In this section, it will be presented a brief description of both Content Delivery Network and HTTP servers. In addition, some examples of the most relevant web servers will also be described.

2.6.1 CONTENT DISTRIBUTED NETWORKS

When serving the contents at a global scale, it is important to guarantee the quality of the service and to do so, the CDNs are becoming increasingly important. The purpose of CDNs is to "maximize bandwidth, improve accessibility and maintain correctness through content replication"[25] To achieve that, a smart management of the cache servers must be done inclusively, bring the cache servers the closest as possible to users. This implementation creates some advantages from reducing the load on origin servers to the improvement of the quality of delivery and speed.

The origin server can be a simple web server and the CDN structure can be "installed" around that [25].

2.6.2 HTTP SERVERS

The HTTP web servers are integrated into a system that consists of a server, one or multiple clients and a network that will support the communication between them. HTTP is the protocol used to communicate. In the beginning, Hyper-Text Transport Protocol wasn't more than a keyword and a document path. Nowadays, HTTP is considered an Internet standard and is the choice of browsers and "virtually every Internet-connected software and hardware application" [46][47].

NGINX

Created by Igor Sysoev, NGINX (pronounced "engine x") is an open-source alternative for one of the most popular open-source web-servers, the Apache. NGINX web server offers the same key functionalities with "a simpler configuration and better performance". It offers: Static file serving; Secure Socket Layer (SSL)/Transport Layer Security (TLS) support; Reverse proxying; Load balancing; Compression; Access controls; Fast Common Gateway Interface (FastCGI); FLV streaming; and others [48][49] [50].

NGINX also features a "module" system. It enables users to create modules (plugins) that expand NGINX core functionality. One popular module is the "nginx_rtmp_module" created by Roman Arutyunyan. This open-source project adds features like RTMP / HLS / DASH live streaming and online transcoding with FFmpeg [51].

Traditional servers rely on threads to handle requests but NGINX has a much more scalable architecture since it handles the requests in an asynchronous way. This web server is the choice of many web sites and services, like:

- Netflix;
- CloudFlare;
- Airbnb;
- Github;
- SoundCloud;
- Heroku;
- MaxCDN;
- and others;

According to the Netcraft's April 2017 Web Server Survey, NGINX served or proxied almost 30% of the busiest sites in that month [52][53].

Currently, NGINX has 2 separate versions, the open source NGINX and the NGINX PLUS. The NGINX PLUS is an enterprise ready version that offers extra features like load balancing, health checks, management, advanced monitoring and full support [54].

RED5

RED5 is an open-source, developed in Java, project started in 2005. This streaming server aimed to replace the Flash Media Server. For curiosity, RED5 name came from the *Star Wars* movies, being Red5 the call name of Luke Skywalker [55].

The first development step was to reverse engineer the RTMP protocol and it took less than a month to successfully connect with a flash client. In the second month, the team behind this project released the version 0.2 of the software.

Nowadays, RED5 is a full-featured Java 2 Enterprise Edition (J2EE) server mainly focused on the flash platform but also supports many protocols like RTMP, Real Time Messaging Protocol Tunneled (RTMPT), RTMPS, RTMPE, as well as HLS, WebSockets and RTSP. The project team is currently working on the Web Real Time Communication (WebRTC) to enable plugin-less browser stream, the integration with the IoT devices and developing the second screen technology (similar to ChromeCast). Companies like Facebook and Amazon use Red5 [56] [57].

There are 2 versions of this project, RED5 Open Source and RED5 PRO. The Table 2.3 shows the main differences between the two [58][59].

	RED5 Open Source	RED5 PRO
HLS support	Via plugin (not maintained anymore)	YES
WebSockets	Via plugin	YES
RTSP	Via plugin	YES
Costumer support	Only Community	YES
Mobile Devices support (SDK)	NO	YES
WebRTC	NO	YES
Second Screen	NO	YES
Clustering	NO	YES
Autoscaling	NO	YES
Open Source	YES	NO

Table 2.3: Comparaision between RED5's Open Source and PRO versions.

OVERVIEW

RED5 supports HLS but misses other relevant adaptive technologies like the standard DASH. Moreover, the open source version has a limited set of features (see table 2.3). On the other hand, the open source NGINX offers support for a large set of features and modules, like the "nginx_rtmp_module" that enables both HLS and DASH. As stated in subsection 2.6.2, NGINX its becoming more popular and offers configuration simplicity without losing performance.

2.7 MEDIA PREPARATION

In this section is described a general view over the encoding and transcoding terms, as well as some state of the art tools that have the capacity to reproduce these concepts in audio/video streams.

For understanding how these tools work it is important to have in mind the main difference between encoding and transcoding. The first term is known as the process of converting an original source (can be a signal, music, text and others) into some well defined format. By example, taking a signal source and produce an H.264 video file with it, is known as video encode. The transcoding is, in short terms, the process of transform a file format into another. For example, taking a video encoded in H.264 and transform it in High Efficiency Video Coding (H.265) [60].

As stated previously in the section 2.5, by using HTTP streaming it is possible to create adaptive streams that will provide different quality rates of the same media, to let clients to adapt to their network conditions. To be able to provide those representations is necessary to perform transcoding of the original media into the multiple desired encodings and quality levels.

Next, it will be presented some of the most used open-source frameworks that are able to handle transcoding procedures of multimedia content.

FFMPEG

FFMPEG is an open-source framework composed by a set of libraries and command-line tools to convert, record or stream media. This solution is able to encode/decode, transcode, stream, filter, mux/demux and even play almost any format created [61][62].

If the goal is to develop an application that relies on FFMPEG, the available libraries are [62]:

- libavcodec;
- libavformat;
- libavfilter;
- libavdevice;
- libswscale;
- libswresample;

If, on the other hand, the goal is to serve end-users, then FFMPEG provides us with these set of command-line tools [62]:

- ffmpeg, media converter between formats;
- ffserver, streaming server for live broadcasts;
- ffplay, simple media player;
- ffprobe, simple multimedia stream analyzer;

There are many software that use this technology like Restreamer (see section 2.11.5) and Blender [63]. In addition, FFMPEG project was forked to Libav project ³ by some developers. This tool accepts a large number of input formats and is capable to perform tasks like extract audio and video streams from a container, mux streams in a new container, cut video or extract track information. FFMPEG also supports RTMP protocol, an important feature for Internet streams since it avoids the complexity and instability of RTP/RTSP in streaming live mode. Using FFMPEG makes it possible to acquire a live or a local video file, transcode it and publish it to an external destination [64].

³<https://libav.org/about/>

VLC

VLC is a free-of-charge, cross-platform, media player and media streamer, created by VideoLAN organization.⁴ By using this player, the client has no need to search and install codecs to play his media. It is possible to input media from the network (RTP/UDP unicast/multicast, HTTP/File Transfer Protocol (FTP), TCP/RTP unicast, ...), DVB(DVB-S, DTV, CTV), local storage or even physical units like Compact Disks (CDs) or Digital Video Disks (DVDs) [65][66].

Besides the media player, VLC can act as a streaming server or client and has transcoding capabilities. For example, the Code 2 shows how its possible to take one media input, transcode it and send it to two different locations [60].

```
% vlc -vvv input_stream \  
%   --sout '#duplicate{dst="transcode{vcodec=mp4v,acodec=mpga,vb=1600,ab=256}:\ \  
%   duplicate{dst=rtp{mux=ts,dst=127.0.0.1},dst=rtp{mux=ts,dst=192.168.1.2}"}'
```

Code 2: Transcoding and streaming to multiple addresses example in VLC.

The supported video formats are: MPEG-1 ; MPEG-2; MPEG-4; DivX 1,2 and 3; Windows Media Video (WMV) 1 and 2; H263; MJPEG and Theora. In terms of audio support, the VLC supports: MPEG Layer 2 audio; MPEG Layer 3 audio; AC-3; AAC; and others. The full list of supported codecs can be found on their official website ⁵.

GPAC

GPac is a multimedia open-source project and has a set of tools:

- MP4Box
- DASHCAST
- OSMO4

These are open-source and cross-platform tools that aim to be used by students, research entities or content creators, in order to test and experiment new standards or to serve multimedia in server streaming applications.

MP4Box is mostly used to manipulate files with main focus on International Organization Standardization (ISO) media files(MP4 or 3GP, by example), encrypt streams and for preparation of DASH content.

DASHCAST allows users to transcode both live and non-live streams, changing its bit-rate and resolution. Also, by using this tool it is possible to segment data streams and pack them for DASH delivery. Currently, DASHCAST is available on Linux and Windows Operative Systems (OSs). The Code 3 shows an example of usage of this tool.

⁴VideoLAN's website: www.videolan.org

⁵<http://www.videolan.org/streaming-features.html>

```
% DashCast -av test.mp4 -seg-dur 100 -live-media
```

Code 3: Transcoding and segmentation in 100-millisecond segments for a live streaming session using DASHCAST.

OSMO4 is the GPAC video player that can play almost any video/audio format and has support for most of the delivery protocols. His focus is on animations and interactivity and is capable to handle 3D multimedia content. OSMO4 is available for windows, Linux, iOS, Android and all platforms compatible with GNU Compiler Collection (GCC) ⁶ and SDL(1.2 or 1.3 version)⁷. This player also offers integration with several browsers like Opera, Firefox or Internet Explorer [67].

OVERVIEW

In this subsection is presented a comparison of the previously referenced tools for transcoding. This comparison takes in account if the tools are open-source, capable to directly transcode into HTTP streaming segmentations and if it supports real-time transcoding.

	FFMPEG	VLC	GPAC
Live Transcoding	Yes	Yes	Yes
Free	Yes	Yes	Yes
Open Source	Yes	No	Yes
HLS ready	Yes	No	No (in development)[68]
DASH ready	Yes (WebM DASH)	No	Yes

Table 2.4: Comparison between transcoding tools.

2.8 TELEVISION LISTINGS

It is important to enrich the service and to provide clients with information about the television multimedia that is available. In that way, this section presents the most known concept behind the share of that complementary information about the television channels programming, as well as the most popular format for delivery TV listings information.

In the past, especially in the analogic television era, channel viewers were informed about channel programming and other complementary information like news, meteorological information and others, using Teletext platform. Nowadays, the concept adopted by the digital television era and the IPTV services is the EPG. EPG, is an application used by digital TV receptors (like set-top-boxes or Smart Televisions) to list and describe programs from one or multiple channels, to the users. This guide provides information about past, current and future programs, start/end times, language, descriptions and other useful information [69]. Figure 2.9 shows an example from the KODI EPG platform⁸.

⁶available at <https://gcc.gnu.org/>

⁷available at <https://www.libsdl.org/>

⁸Image extracted from <http://i.imgur.com/BEa96K3.png>

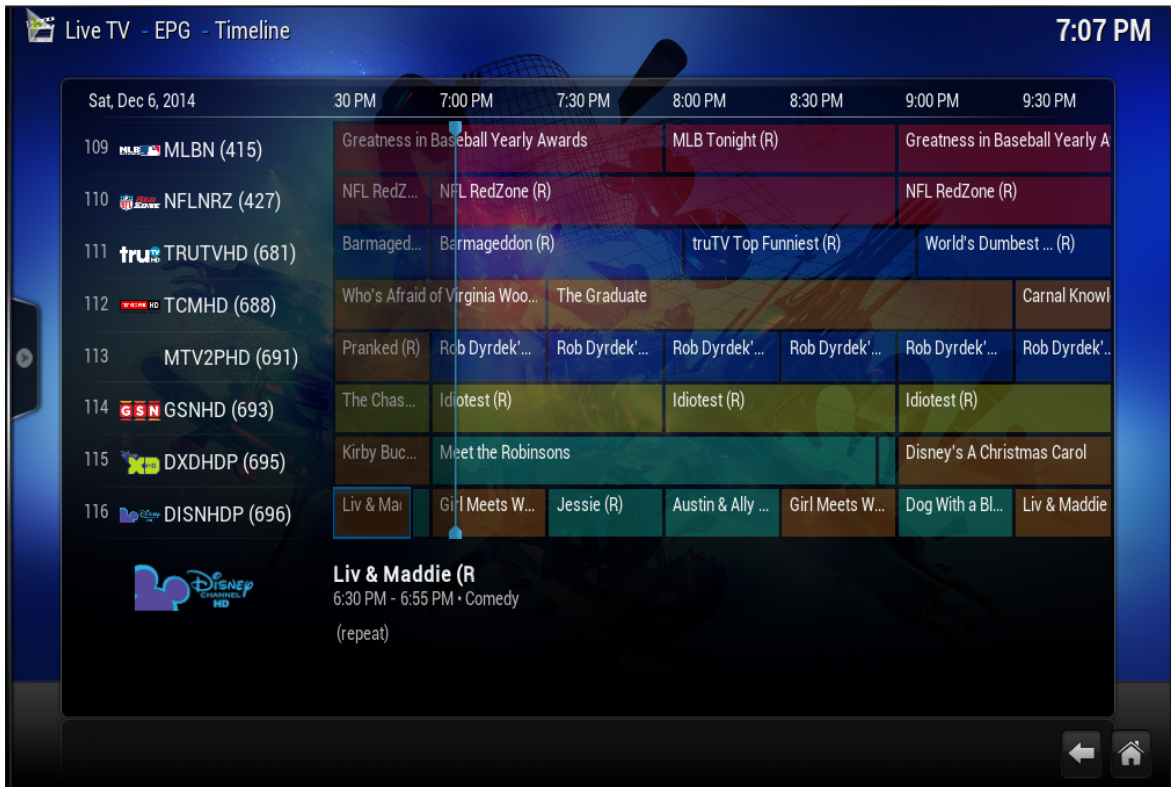


Figure 2.9: KODI EPG platform.

2.8.1 XMLTV FORMAT

This subsection presents the most known format for representing and sharing TV listings, the XMLTV format. As the name suggests, this format is represented in a XML file and contains all the channels programming mixed together.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE tv SYSTEM "xmltv.dtd">
<tv source-info-url="http://www.schedulesdirect.org/"
source-info-name="Schedules Direct"
generator-info-name="XMLTV/$Id: tv_grab_na_dd.in,
v 1.70 2008/03/03 15:21:41 rmeden Exp $"
generator-info-url="http://www.xmltv.org/">
<channel id="I10436.labs.zap2it.com">
<display-name>13 KERA</display-name>
<display-name>13</display-name>
<display-name>13 KERA fcc</display-name>
<display-name>KERA</display-name>
<icon src="file://C:\Perl\site\share\xmltv\icons\KERA.gif" />
</channel>
<programme start="20080715003000 -0600" stop="20080715010000 -0600"
channel="I10436.labs.zap2it.com">
<title lang="en">NOW on PBS</title>
<desc lang="en">Jordan's Queen Rania has made job creation a priority to help
curb the staggering unemployment rates among youths in the Middle East.
</desc>
<date>20080711</date>
<category lang="en">Newsmagazine</category>
<category lang="en">Series</category>
<episode-num system="onscreen">427</episode-num>
<audio>
<stereo>stereo</stereo>
</audio>
<previously-shown start="20080711000000" />
<subtitles type="teletext" />
</programme>
<programme start="20080715010000 -0600" stop="20080715023000 -0600"
channel="I10436.labs.zap2it.com">
<title lang="en">Mystery!</title>
<sub-title lang="en">Foyle's War, Series IV: Bleak Midwinter</sub-title>
<desc lang="en">Foyle investigates an explosion at a munitions factory,
which he comes to believe may have been premeditated.</desc>
<date>20070701</date>
<category lang="en">Series</category>
<episode-num system="dd_progid">EP00003026.0665</episode-num>
<audio>
<stereo>stereo</stereo>
</audio>
</programme>
</tv>

```

Code 4: Example of a XMLTV file.

As we can see in Code 4 , an XMLTV file has two distinct types of records: “channel” and “programme” records. “channel” records save information about the channels available whereas “programme” records store data related to the channel’s programs and episodes, such as:

- channel ID;
- date;
- start time;

- stop time;
- language;
- description;
- sub-title;
- and others;

This format is supported and used in platforms like MythTV, DVBLink, Emby, Perfect Player or the popular KODI media center (formerly called XBMC) [70].

2.9 MOBILE DEVICES

The previous sections described each key entity to build a HTTP adaptive streaming server. As stated in Section 2.5, the growth of mobile traffic increases the importance of offering good video services to mobile users. This section aims to describe the most known mobile devices operative systems, frameworks, as well as the multimedia capabilities in terms of reception, encoding and profiles.

According to data provided by the International Data Corporation, in the third quarter of 2016, Android system dominates the Smartphone market with a share of 86,8%. Considering Android and iOS systems together, the share rises to 99,3% of the market [71].

2.9.1 ANDROID

Android is a software kit for mobile devices and is composed by:

- Linux Kernel (OS);
- Middleware (Frameworks, Libraries and Android Runtime);
- Key Applications



Figure 2.10: Android Stack illustration.

As the Figure 2.10⁹ shows, this system is organized in a stack mode, on the bottom stays the Linux Kernel and on the top are placed the applications [72].

The Table 2.5 shows the network protocols supported for media playback on Android mobile devices. Android provides RTSP and HTTP progressive streaming support in all versions. Android also supports, since version 3.0, HTTP Live Stream. All the versions since 3.1 support HTTPS [73].

	<3.0	3.0	>3.1	>4.0
RTSP	Yes	Yes	Yes	Yes
HTTP Progressive Streaming	Yes	Yes	Yes	Yes
HLS	No	Yes	Version 2	Version 3
HTTPS	No	No	Yes	Yes

Table 2.5: Android's versions compatibility with some streaming technologies.

In terms of audio, Android is capable of encode and decode codecs like AAC , High Efficiency - AAC (encoding available after version 4.1) and decode MP3. In terms of video, the codecs H263, H.264, H.265(after version 5.0), VP8 (after version 2.3.3), VP9 (after version 4.4) can be decoded by this system.

⁹Image Source: https://source.android.com/images/android_framework_small.png

	Low	Medium	High
Resolution	176 x 144	480 x 360	1280 x 720
Bitrate	12 fps	30 fps	30 fps
Frame Rate	56 Kbps	500 Kbps	2 Mbps
Audio Codec	AAC-LC	AAC-LC	AAC-LC
Audio Bitrate	24 Kbps	128 Kbps	192 Kbps

Table 2.6: Recommended video encoding for playback in H.264 Baseline Profile codec.

The Table 2.6 shows the Android’s recommended encodings to playback audio/video media on mobile devices, using H.264 Baseline Profile. These parameters are not mandatory but they can provide a better user experience and less playback issues [73].

TV INPUT FRAMEWORK

Android TV Input Framework (TIF), is a framework created to provide a standard API to control and enable live streams on Android TVs. With TIF, it is possible to access to a wide variety of Television input sources, like High Definition Multimedia Interface (HDMI) and built-in tuner, in a single user interface. This tool becomes useful for device manufacturers and developers since it provides TV broadcast standards without the need to re-implement them.

Developers can use, extend or even replace the TV application present on this framework. TV application is a requirement for Android TV devices. The framework is composed by:

- TV application;
- TV Input Manager, to communicate with the Television application and provide parental control;
- TV Inputs, like HDMI or built-in tuner inputs;
- TV Input HAL, that allows TV Inputs to access TV-specific hardware (when implemented);
- HDMI-CEC, technology that enables remote-controlling over HDMI cable.

Android provided to developers an open-source project that represents a TV application example using TIF¹⁰. This project also provides a tutorial on how to get started. It consists basically in a single TV input where the files are served from Google Cloud Storage. The contents are organized by genre. This example uses DASH and HLS streams, playing content using Google’s ExoPlayer¹¹ [74].

EXOPLAYER

ExoPlayer is an extensible application-level media player for Android systems. Comparing to MediaPlayer (standard Android player API) It provides extended features, is easier to customize and extend, adding support to DASH and IIS Smooth Streaming playbacks on custom applications [75].

¹⁰<https://github.com/googleamples/androidtv-sample-inputs>

¹¹<http://google.github.io/ExoPlayer/>

2.9.2 APPLE IOS

iPhone Operative System, or iOS, is Apple's mobile system that runs on his mobile devices. In terms of audio, iOS supports many of the industry-standard and Apple-specific formats like AAC, Apple Lossless Audio Codec and others. For video playback and based on their recommendations, iOS presents support for the following [76]:

- "H.264 video, up to 1.5 Mbps, 640 by 480 pixels, 30 frames per second, Low-Complexity version of the H.264 Baseline Profile with AAC-LC audio up to 160 Kbps, 48 kHz, stereo audio in .m4v, .mp4, and .mov file formats";
- "H.264 video, up to 768 Kbps, 320 by 240 pixels, 30 frames per second, Baseline Profile up to Level 1.3 with AAC-LC audio up to 160 Kbps, 48 kHz, stereo audio in .m4v, .mp4, and .mov file formats";
- "MPEG-4 video, up to 2.5 Mbps, 640 by 480 pixels, 30 frames per second, Simple Profile with AAC-LC audio up to 160 Kbps, 48 kHz, stereo audio in .m4v, .mp4, and .mov file formats";

Additionally and accordingly to the information present on the "Using HTTP Live Streaming" guide [77], there are other supported parameters available for HLS playback. These parameters depend on the target devices and the H.264 profile used. If we want to target e. g. the iPhone4 and later models, all iPads and Apple TV 2 and later, we can choose the H.264 Main Profile, level 3.1. With this profile enabled, a WiFi connection is recommended, meaning that may be not possible to play the streams on cellular networks. The video parameters allowed for this profile are: H.264 video, up to 4.5 Mbps, 1280 x 720 pixels and 90 frames per second. The complete information about the allowed media parameters can be found on the referenced guide.

2.9.3 MOBILE DEVICES OVERVIEW

Comparing Android and iOS media capabilities, it is noticeable that Android presents a large set of codecs and less restrictions on media transmissions over the network. Although it is possible to stream media to iOS using RTSP or HTTP (using DASH, for example), Apple presents a restriction on playing media (in terms of duration and size) if it is not using HLS(as stated in section 2.5.3).

2.10 HTML FRAMEWORKS

After analysing the server's state-of-the-art technologies and key entities in the building of an adaptive Over-The-Top streaming server, is crucial to investigate the state-of-the-art technologies for the client side of the infrastructure.

The evolution of the browsers on both mobile and desktop platforms and the emergent capabilities of the set-top-boxes, motivates the development of web client instead of creating a dedicated application for each operative system. Recently the browsers adopted the Media Source Extension in almost every platform. MSE is a specification that extends HTML5 media elements to allow JavaScript to generate media streams for media playback. The advantages of this specification are presented by W3 organization ¹² which states, "Allowing JavaScript to generate streams facilitates a variety of use cases

¹²<https://www.w3.org/TR/media-source/>

like adaptive streaming and time shifting live streams."

Another advantage is that relying on JavaScript, there is no need to create Flash players to enable media playback on browsers. This section aims to present some of the most used and known HTML frameworks that enable developers to create and present web video applications.

2.10.1 ANGULARJS

Angular is a JavaScript framework that gathers used concepts in other web programming structures. This framework works like AJAX applications (Single-page applications) on the client side, where the template and data are sent to the browser, to be assembled. Then the server only has to serve static resources to the templates and the required data by them.

In Angular the programming model is the Model-View-Controller (MVC), where the Document Object Model (DOM) is the View, the Controllers are the JavaScript objects and the model data is stored in the object properties. Likewise, it is possible to declare which parts of the UI are mapped to a JavaScript property. This process is called data-binding. Angular also features [78] [79]:

- Dependency Injection, where classes request the data they need instead of creating dependencies;
- Directives, which make it possible to create templates as HTML extending HTML syntax through a DOM transformation engine;
- Reusable Components;
- and others;

The Code 5 shows a very elementar example how Angular templates look like. While tipping the name in the text input, Angular automatically updates the "Hello" phrase [80].

```
<!doctype html>
<html ng-app>
<head>
<script src="../../../angular.min.js"></script>
</head>
<body>
<div>
<label>Name:</label>
<input type="text" ng-model="myName" placeholder="Enter a string">
<hr>
<h1>Hello {{myName}}!</h1>
</div>
</body>
</html>
```

Code 5: Simple example of an Angular template.

2.10.2 REACTJS

React is a JavaScript framework created, by the Facebook team, to handle complex user interfaces. They tried to implement an alternative to Model-View-Controller. MVC consists of two-way data

binding and a rendering template. This framework introduces features like virtual Document Object Model, JSX and FLUX.

Conventional JavaScript front end development involves working around with DOM but React introduces a friendly way to interact with it, improving, at the same time, the performance. The React's data model uses standard JavaScript objects as the building blocks of its internal framework. When this framework detects a change in the data model, all the application is re-rendered but then, React only updates the components that have changed.

React components provide isolation and re usability to our User Interface, splitting the UI in pieces (each one is a component). The example Code 6 shows how React components look like in practice and how it is possible to reuse them [81].

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}

ReactDOM.render(
  <App />,
  document.getElementById('root')
);
```

Code 6: Component composing in React framework.

JSX is the language used by React. It works as an "alternative" to JavaScript and is an object-oriented language designed to run on modern browsers. JSX is faster than JavaScript, provides debugging features and errors can be found on the compilation level. Also, JSX provides a class system similar to Java and, at same time, expressions and statements are equal to JavaScript. It is possible to convert JSX into Javascript [82][83][84].

FLUX is an architecture used by Facebook, inside React framework, to provide unidirectional data flow instead of MVC. Figure 2.11 shows the data-flow of FLUX. As we can see it is composed by Actions, Dispatcher, Stores and Views. Every Action, containing the change in the application, is sent to the Stores via the central hub, the Dispatcher. The communication between the Stores and the Dispatcher works as follow: the Store registers callbacks in the Dispatcher and the Actions are sent through this callbacks. When Stores receives Actions and updates themselves, they emit an event signaling a change to update the View(s) [85].

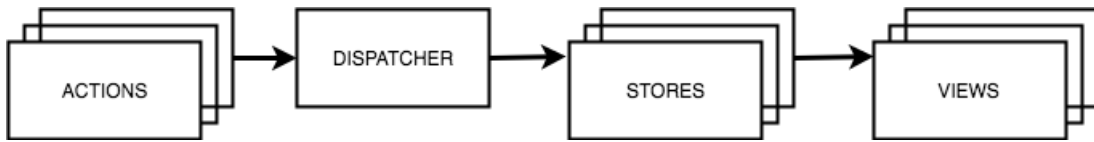


Figure 2.11: React's FLUX Data Flow.

Following the same concept of React, it was created the React Native framework, an open-source library that enables developers to create native user-interfaces.

2.10.3 JAVASCRIPT LIBRARIES

The previous presented frameworks are useful to develop web applications but, in order to create and develop a client platform to run adaptive streams, it is also needed libraries that handle the streams into the HTML 5 elements. This subsection is dedicated to the analysis of some of the most used and known libraries and player implementations that can handle adaptive streaming playback.

HLS.JS

hls.js is a JavaScript library that runs on HTML 5 browsers that support Media Source Extension. The purpose of this library is to play HLS streams on browsers across multiple platforms. A demo page was provided in order to developers or users test their streams. hls.js transmuxes MPEG2-TS and MP3/AAC stream into MP4 fragments (asynchronously, if browser supports Web Worker).

In most recent versions hls.js added support for fMP4 playback (HLS version 4). This library is integrated in players like:

- Akamai Adaptive Media Player (AMP);
- Clappr;
- FlowPlayer;
- and others;

hls.js is also used in production, by companies like Twitter, The New York Times, Fox Sports, Dailymotion and others [86].

DASH.JS

dash.js is an open-source and free for commercial use, JavaScript framework to build players on web browsers, taking advantage of his MSE capabilities. This library was created by DASH Industry Forum and intends to provide a production environment, with the best adaptation algorithms and, at the same time, follow the best practices of MPEG-DASH playback. Another goal is to keep both codec and browser agnostic. This framework also provides a test page that uses its library, for developers and users to test their implementation¹³.

DASH Industry Forum is a non-profit association, composed by many companies from Microsoft, Cisco, Intel, Akamai, Qualcomm to Ericson, and others [87].

¹³<http://dashif.org/reference/players/javascript/>

CLAPPR PLAYER

Clappr is an open-source web player, written in JavaScript, that relies on hls.js to provide users HLS playback. It also supports WebM (Chrome and Firefox browsers) and it is possible to run DASH resorting to an external plugin. There are other plugins, created by community, that enable users to [88]:

- Play YouTube videos;
- Play 360° videos;
- Add markers in the video play bar;
- and others;

This player can also be tested on a demo page provided by Clappr¹⁴.

SHAKA PLAYER

Shaka is a Javascript library, created by Google, that relies on MSE and Encrypted Media Extensions (EMEs) to play adaptive media contents. This player is open-source and free from third-party dependencies. Currently Shaka supports protected content systems based on EME-compliant MSE and both DASH and HLS streaming specification. Supported formats are MP4, Web Video Text Tracks (WebVTT) and Timed Text Markup Language (TTML) but, depending on browser support this player also supports MPEG2-TS and WebM formats [89].

2.10.4 OVERVIEW

FRAMEWORKS

By comparing the Angular 2 and React frameworks it is noticeable that React has implemented some features that diverge on Angular's approach. Both frameworks have handle DOM but, React provides a way to abstract developers and increase performance on this (Virtual DOM). Angular uses the MVC approach with 2-way data binding while React makes use of FLUX to keep data unidirectional. Moreover, React works with JSX, which extends JavaScript functionalities.

PLAYERS & LIBRARIES

Both players and libraries rely on MSE browser capability, that allows JavaScript to generate media streams for playback.

The table 2.7 shows the support for MSE and HLS for the most known browsers. Since Opera Mini offers no support for any of the two features, it is not possible to playback DASH or HLS streams on this browser. The other presented browsers offer, at least, one of the two, currently. For example, on iOS Safari it is possible to run HLS though it is not possible to run DASH or HLS relying on MSE.

¹⁴<http://clappr.io/demo/>

	MSE	HLS support (without MSE)
Chrome	31+	No support
Chrome for Android	40+	40+
Firefox	42+	No support
Safari	8+	6+
Opera	15+	No support
Android Browser	Android 4.4.4+	3+
iOS Safari	No support	3.2+
Internet Explorer	11+ (for windows 8/10)	No support
Edge	12+	12+
Opera Mini	No support	No support

Table 2.7: List of most known browsers that features MSE and HLS support.

Comparing the two players, Clappr and Shaka, the first relies on plugins to run DASH. On the other hand, Shaka doesn't need any external dependency.

2.11 IPTV SYSTEMS AVAILABLE

This section presents some of IPTV platforms used nowadays. Starting by some successful multimedia platforms followed by some live and on-demand television web platforms and finishing with a brief review of IPTV architectures already implemented and available.

2.11.1 YOUTUBE

YouTube is a popular platform to upload and share video, and the most recent statistics show that one third of of the Internet's community has a YouTube's user account [90]. Another interesting statistic is that more than half of the video visualizations come from mobile platforms. As we can see in Figure 2.12, YouTube uses MPEG-DASH to stream media. This platform also makes use of HLS in some cases. YouTube users can view media content on web browser or on the mobile application (available for Android and iOS). Based on their documentation, the player uses an iframe container. By using the iframe API it's possible to control the video player using JavaScript and let YouTube switch to HTML5 on devices that do not support Flash. Although, when analysing the Chrome browser for macOS and Safari for the same operative system, it is noticeable that, by default, YouTube loads in HTML5 mode, avoiding the use of the Flash technology. The page snippets showing that information can be found in appendix A [91].

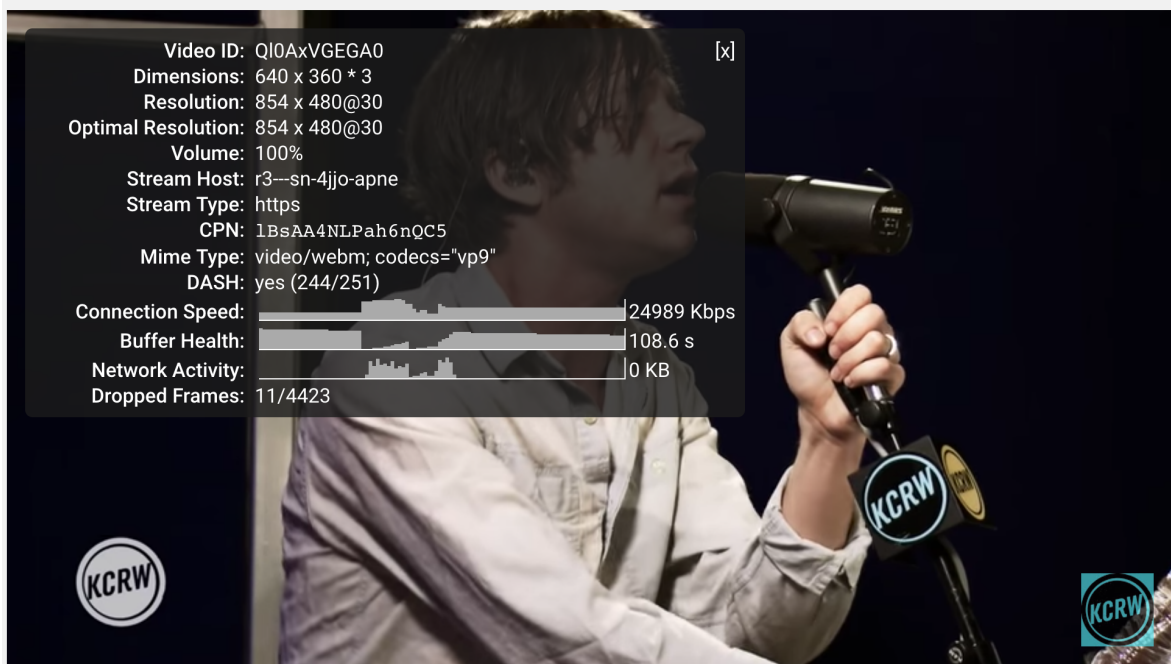


Figure 2.12: Youtube player statistics.

Youtube also features live streaming and analytics to understand the user's interaction with videos [91].

2.11.2 KODI

Kodi, formerly known as XBMC, is an entertainment center that enables users to listen to music, play movies, Television shows and photos. This platform is free, open source and able to run on windows, OSX, iOS, Android, Linux, Rasperry Pi and other platforms. The project was created by the XBMC/Kodi Foundation in 2003 and is sponsored by Nvidia, Alienware, Acquia, Minix, Doghouse, Bytemark, HomeRun and Wetek. Figure 2.13 shows a menu example of Kodi's User Interface. Kodi features an appropriate UI to Televisions and remote controls. Additionally, users can play content from the local storage or through the network [92].

As stated in Kodi's official repository, "Currently Kodi can be used to play almost all popular audio and video formats around. It was designed for network playback, so you can stream your multimedia from anywhere in the house or directly from the Internet using practically any protocol available.". It is possible to run HLS streams using an m3u8 addon available and since version 17 (codename Krypton) , kodi supports DASH, Microsoft Smooth Stream, RTMP streams and others [93].

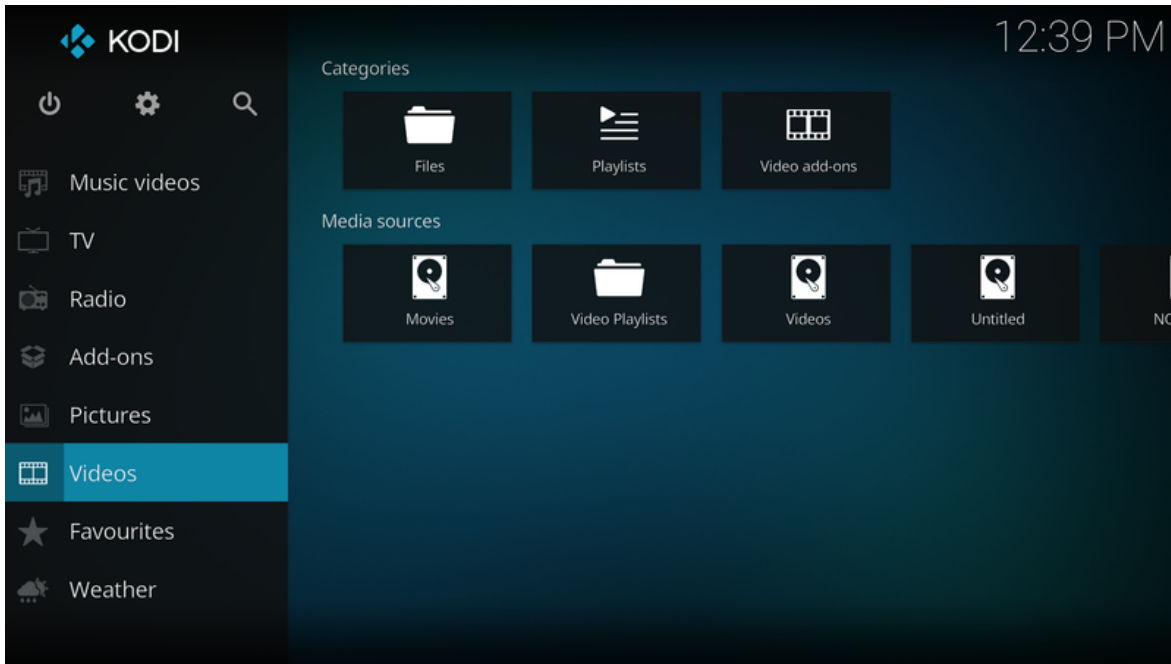


Figure 2.13: Kodi Krypton User Interface.

The biggest downside of Kodi is that its not possible to run it on a web page, so its demanding to install Kodi's software.

2.11.3 RTP PLAY

RTP Play is a website that enables users to watch live or on-demand RTP's Television channels and Radio stations. This platform consists in a web site and a mobile application (available for Android and iOS devices).

Figures 2.14 and 2.15 show, through a page inspection, that RTP uses, both on TV and radio, HLS streaming [94].

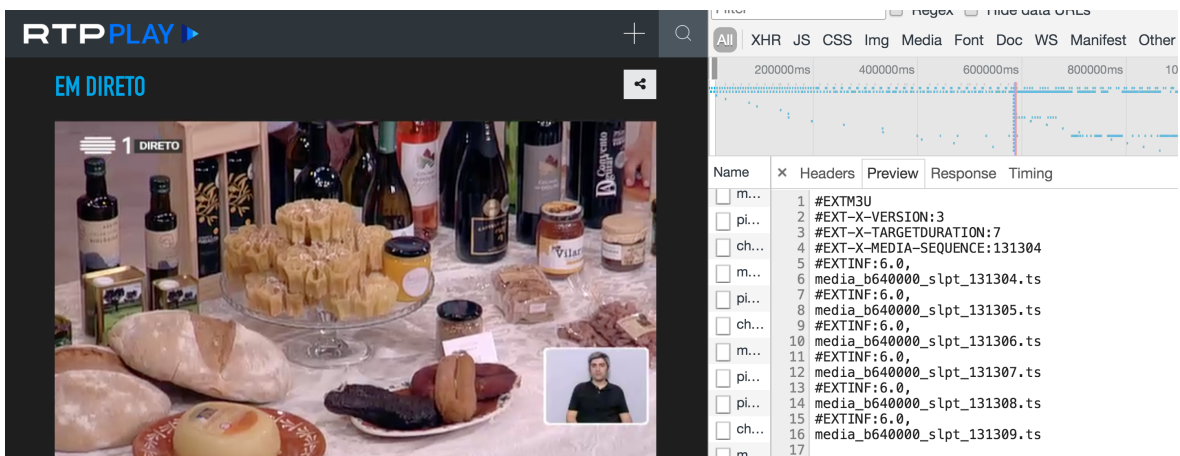


Figure 2.14: RTP Play m3u8 playlist on a TV channel.

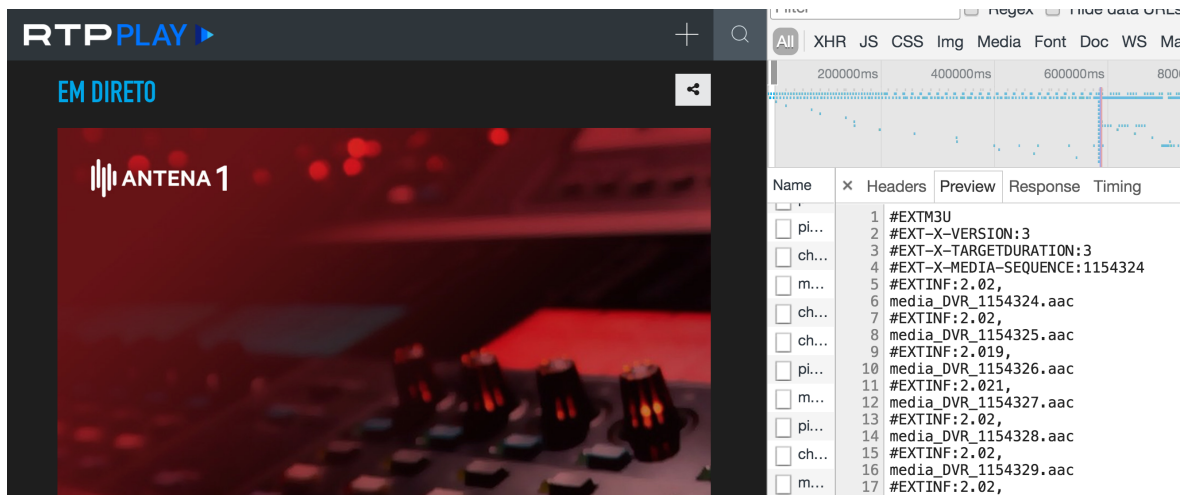


Figure 2.15: RTP Play m3u8 playlist on a radio station.

2.11.4 MEO GO

MEO Go is a solution for watching TV content across multiple devices. This service offers, to its users, a large set of thematic and premium channels. MEO and MOCHE mobile network users can benefit up to 10 GB per month of free data. Moreover, it is possible to watch video store movies, restart or pause the current program channel, access to the TV guide and take advantage of scheduled recordings.

The application is available for iOS, Android and Windows. To run the contents on `meogo.pt`, the browser has to be able to run NPAPI plugins, specifically Microsoft Silverlight, since MEO Go uses Microsoft Smooth Streaming to serve the media contents.

It is recommended, on this platform, a minimum of 1 Megabits per second (Mbps) to run Standard Definition (SD) contents and 3,5 Mbps to High Definition (HD) contents [95].

2.11.5 RESTREAMER

Restreamer is an open-source solution that offers real-time video streaming using HLS. It is possible to stream H.264 video of RTSP/RTMP/RTP sources. Using restreamer enables the user to upload the media on YouTube, Ustream, Twitch, Livestream.com, Wowza or others. This system runs inside a Docker therefore, a Docker image is available to download for free, even for commercial use.

This solution uses NGINX as a server, FFmpeg for transcoding, Angular and Node.js for process management and Clappr as a video player. Figure 2.16 shows the architecture used by Restreamer. The transcoding step pushes the RTMP streams for NGINX and for a external source at the same time (if requested by the user) [96] [97].

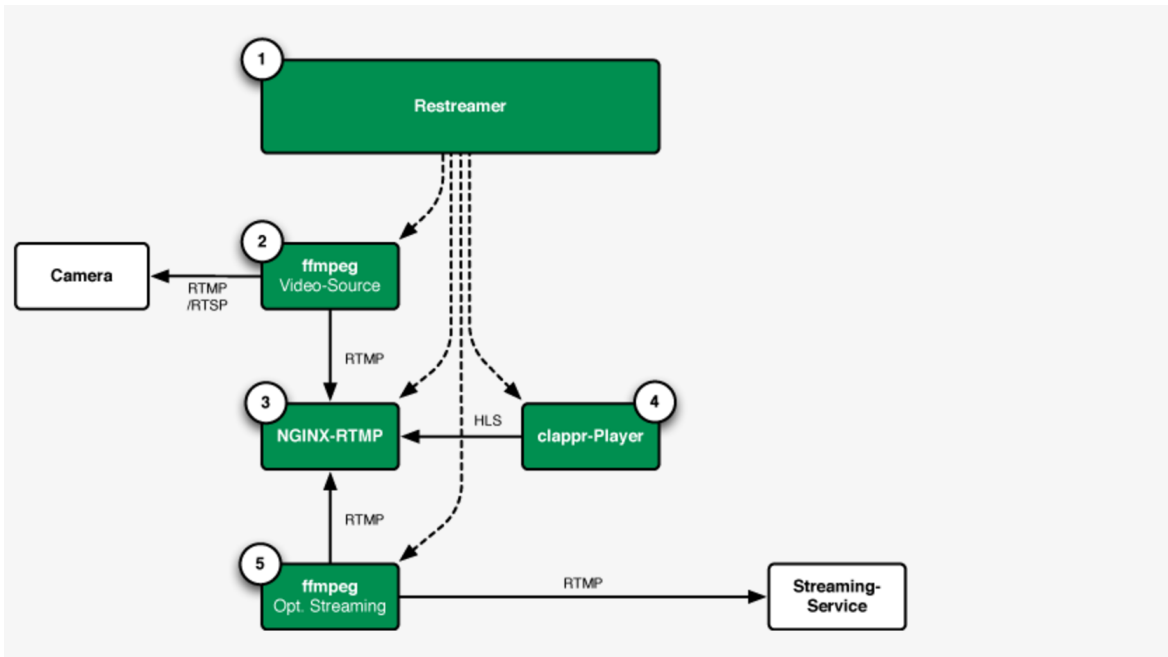


Figure 2.16: Restreamer architecture.

The Restreamer downsides are the input sources limitation already stated and the lack of HTTP streaming solutions. The HLS is the only viable streaming method available at the NGINX_RTMP_MODULE because despite DASH standard is implemented, it currently does not support more than one video representation.

2.11.6 TV HEADEND

Tvheadend is a TV streaming server and Digital Video Recording (DVR) application.

It supports the following inputs:

- DVB-C(2), DVB-T(2), DVB-S(2);
- ATSC;
- SAT>IP
- HDHomeRun
- UDP
- HTTP

And the following outputs:

- HTTP
- HTSP (own protocol)
- SAT>IP

It is possible to input an HLS source using an external application ¹⁵ but it's not possible to output using the same solution or any adaptive alternative like HLS and DASH. TvHeadend provides an EPG interface using multiple input sources from XMLTV to over-the-air DVB [98][99].

2.11.7 OVERVIEW

YouTube is probably the leader of the streaming services, providing solutions across platforms and using state-of-art streaming technologies. It's also possible to stream live events but it was not designed to stream live Television broadcasts so the platform doesn't provide them. The Kodi platform is capable of reproducing the most known adaptive streaming technologies but requires the use of an application (it isn't web based). The RTP Play solution is web based and capable of providing media playback without external plugins but only provides RTP channels data. The Meo Go platform isn't a free system but provides a large set of channels, with playback on multiple devices, nevertheless it requires plugins or the installation of an application. As stated in section 2.11.5, the Restreamer is a streaming solution that specifically uses HLS and has restrictions on input sources. It also doesn't provide any EPG service since it was not concretely designed to stream TV channels. Tv Headend was designed for Television streaming and provides an EPG service although it is not possible to stream using HLS or any adaptive streaming solution.

¹⁵<https://github.com/Jalle19/node-ffmpeg-mpegts-proxy>

SOLUTIONS AND SCENARIOS

This chapter aims to describe the proposed scenarios, goals and requirements as well as the solutions proposed to implement in this work. This analysis starts by describing, in section 3.1, the ideal scenario in an IPTV platform. Next, in section 3.2 are presented the proposed requirements for this work and, in section 3.3, are explained the goals proposed to achieve according to the initial requirements. Keeping in mind these two topics, in section 3.4, are presented the possible scenarios, the main blocks of the proposed architectures and the main advantages of each solution.

3.1 IDEAL SCENARIO

In an ideal approach, an IPTV platform should be composed of two large entities. One entity with the responsibility to serve the contents, that will be called server, and other to receive the contents user request, that will be assumed as the client. Figure 3.1 illustrates this scenario.

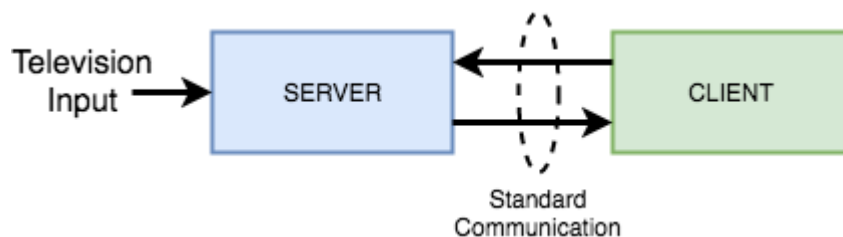


Figure 3.1: Over-The-Top communication ideal scenario.

The channel data should arrive to users with zero delay or, in other words, users should see the digital representation at the exact same time that the real world event occurs, which is an impossible feature, so the goal is to have the less delay as possible. The communication between servers and clients is one of the biggest challenges in the streaming world. First it is important to understand how server and clients are connected since it is this connection that will support the communication between entities. In IPTV the data exchange is made Over-The-Top, meaning that the IP networks are

the base of the audio, video and other data transfers. The biggest limitation about OTT transmissions is the connection's bandwidth.

Poor connections, mobile data connections and other events, can cramp the available bandwidth to serve and/or receive the TV streams so, a server should provide alternatives to adapt to these conditions. One ideal server should also be able to communicate with all its clients in a standard way, enabling them to understand how to receive and display the media.

In a user perspective, it is important to gather information about what they are seeing or what they can see, returning the broadcast to the beginning of the program or to the start of the month. To achieve that, the server should provide an Electronic Programming Guide with relevant information to the client and to the client's user and additionally store the broadcasted data so the clients can request, at any time, past programming media.

One ideal client should concern about providing the best user experience to its clients. It is important to guarantee a quality media playback, with the best video/audio as possible and without any breaks, independently on where the platform is running. To accomplish that the client has to monitor the network connection (checking bandwidth changes) and implement an intelligent buffer that will decide how to avoid playback issues and how to provide the best quality as possible. A good buffering on the client side can also reduce the overhead on the server for VOD situations if it knows how to start and stop requesting data. For example, and similarly to the YouTube's behaviour, the client buffers 20 seconds ahead of the stream instead of buffering indeterminately because users can leave the broadcast after 10 seconds.

3.2 REQUIREMENTS

The purpose of this work is to develop both server and client entities, in order to obtain a full IPTV platform. The main requirements are:

- Create an open source solution;
- The client should be an HTTP-based platform and run in a large set of devices from personal computers or mobile devices to set-top-boxes;
- The server should be deployed on a regular server, without the need of dedicated hardware or non open source Operative Systems;
- The media input (Television data) should be captured using a Universal Serial Bus (USB) DVB-T dongle;

3.3 GOALS

Besides the requirements previously stated, our solution must provide multiple encodings to increase compatibility between devices (see section 2.9). It also should be a distributed solution that enables the use of multiple devices to process the data input. In that way the platform is capable of scale horizontally with ease. Also, as stated in section 2.5, one of the advantages of HTTP streaming is the adaptive capability to network conditions. Our solution should take advantage of

that capability to provide a more flexible and stable service, in both server (that provides the multiple adaptations) and client(that should dynamically adapt accordingly to client's network conditions). In order to complement the streaming service, the server should provide Television listings on the channels broadcasted. The client should then consume these listings to build an Electronic Programming Guide service that will also provide actions like restart the current program or an already broadcasted one.

The interaction with the client platform by users, should be simple with set-top-box menus similarity, with keyboard shortcuts to enable the interface to be controlled by a keyboard or a remote. The client should also be built under a modular, reusable architecture to facilitate the customization by other developers and re-usability in other contexts.

3.4 POSSIBLE SCENARIOS

To fulfill these requirements and achieve the presented goals, there are some possible approaches that can be followed. Starting with a typical HTML server architecture and the global streaming paradigm (which relies on serving media files through the HTML server), the figure 3.2 shows the required data work-flow from the DVB-T input to the distribution server. This scenario has five fundamental blocks:

Transcode: The Transcode block receives the TV data as input and creates or changes the media parameters. These changes can be performed in terms of bit-rate, resolution, codec and others. One input can generate, if needed, multiple outputs.

Segment and meta-data generator: This block has the responsibility to prepare de multimedia data in accordance with the transportation protocol (Section 2.5). This preparation consists in taking the input, that can be composed by one or a set of different encodings for the same channel, generate the media files and the appropriate meta-data that will instruct the players how to manage the requests and other relevant information.

EPG extractor: This extractor uses the DVB-T input to get the broadcasted channel programming, also known as Electronic Programming Guide. The output consists of a file with all the channels programming information.

EPG API: This block takes the result of the EPG extraction and creates a communication interface (Application Programming Interface) that enable users to request the programming information (the last broadcasted program for a specific channel, for instance).

Distribution server: This block is responsible for serving the contents to the clients. It features server locations for serving the metadata, the media files and other contents like relevant static files. It's also responsible to proxy some requests like the EPG related ones, that have to proxied to the EPG API block.

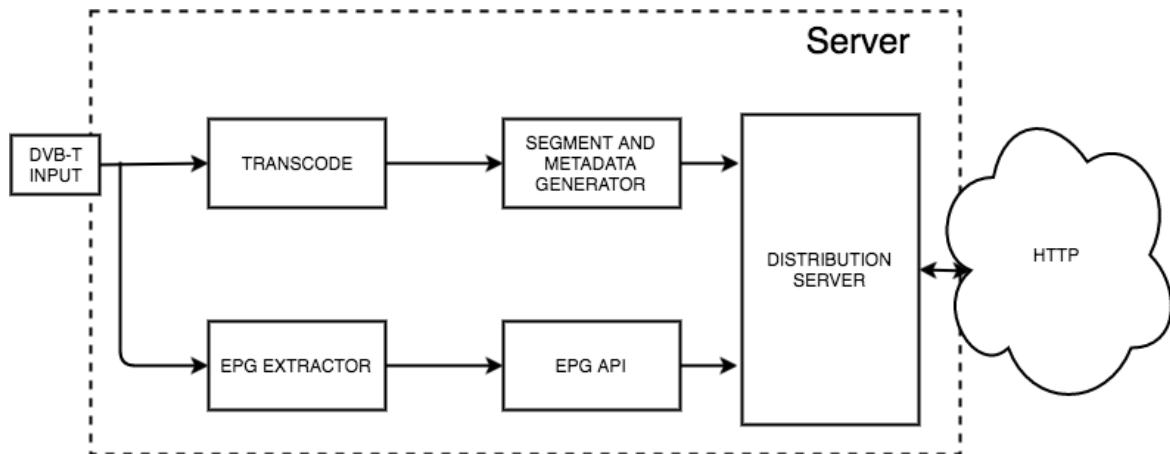


Figure 3.2: Fundamental blocks and workflow of the server architecture.

This scenario satisfies the main server requirements (presented in Section 3.2) but there are some improvements that can be made since in this solution it will be needed one DVB-T USB dongle per each channel. Moreover, due to the fact that one of the presented goals (Section 3.3) is to create a distributed architecture, it is necessary to distribute the Transcode block because it needs more computational resources (for example, CPU). Figure 3.3 shows the obtained solution after these changes and improvements. By using this architecture a new block was added to the scenario:

Unicast channels creator: This block has the purpose of reading the channels from the DVB-T USB dongle and create a multicast address per each channel. These addresses should stay available to the internal network.

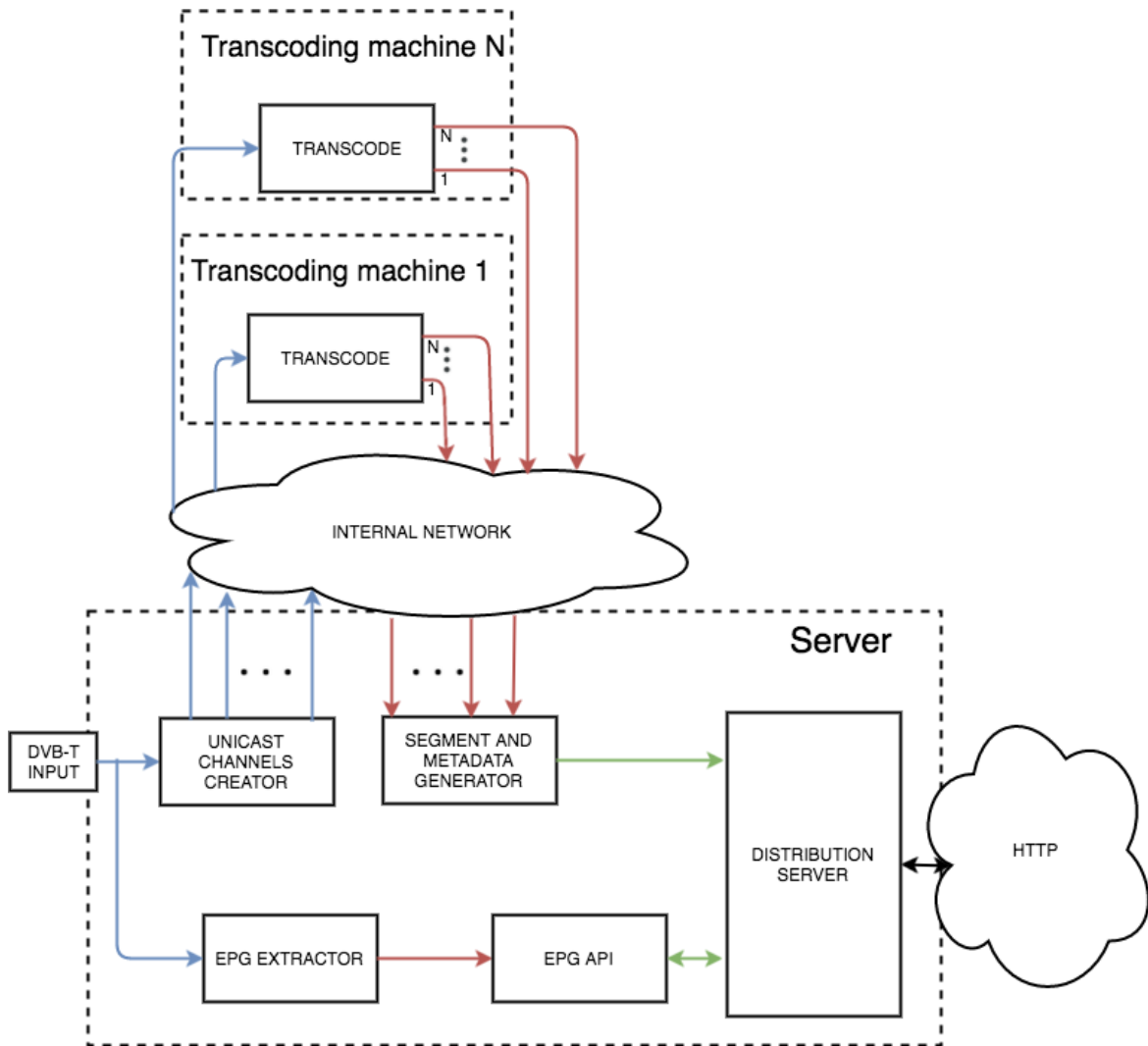


Figure 3.3: Server architecture with transcoding using different physical machines.

With the introduction of this new block, the channels become available, via unicast, to the internal network. This change creates the possibility to multiplex the transcoding through different physical or virtual machines that have connectivity to the server. After this process the result data should be sent through the internal network, using an RTMP stream to the segment and metadata generator.

On the client entity, since one of the goals (Section 3.3) is to develop a modular platform that can be easily customisable and that is able to display not only live multimedia, but also previously broadcasted programs and another channel related information like the EPG, the following solution was created:

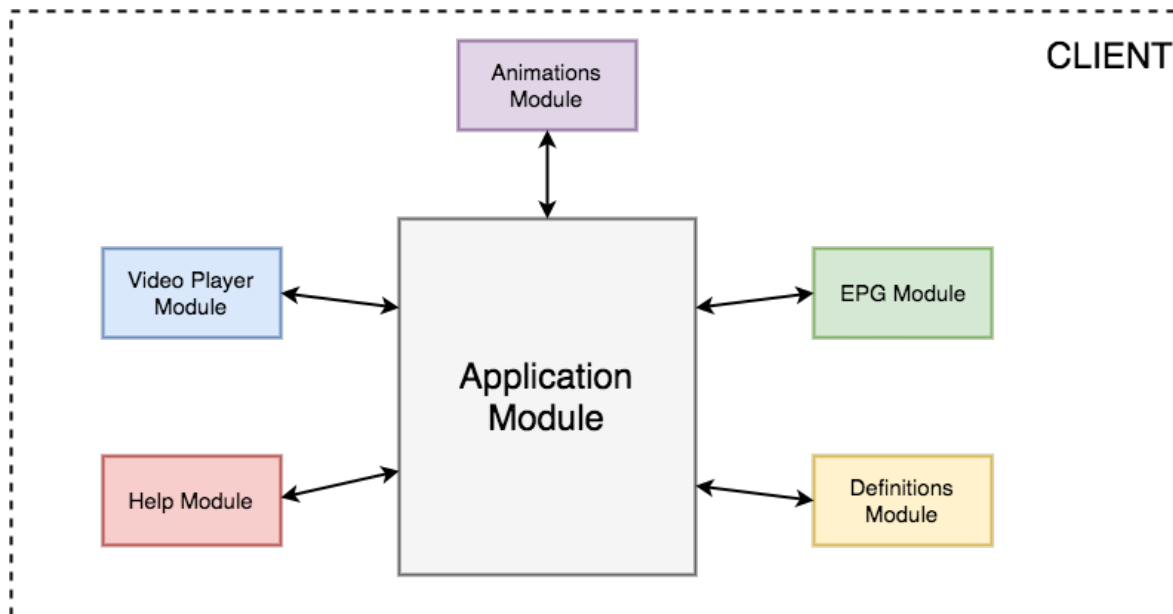


Figure 3.4: Client architecture composed of independent modules.

This architecture is composed of four functional modules plus the main module (Application Module):

Application Module: It is the central component of the client. It knows which module should be rendered in each situation and how to update the module parameters (the video source, in the Video Player module, for example).

Video Player Module: This module is responsible for handling all the multimedia (video and audio) operations from the media source to the video controls and to know how to communicate with the server (communication protocol like HLS or DASH) in order to receive the appropriate media for each available bandwidth level. The Video Player capabilities should also include:

- Set the media source;
- Volume control (increase, decrease and mute);
- Seek the video;
- Control video quality;
- Play and Pause video controls;

EPG Module: This module's responsibility is to present all the relevant information about channels programming and, at the same time, provide control over that information. That control includes taking actions like restart the current program or start playing an already broadcasted one. After the user selects the desired action, the EPG Module communicates with the Application Module to update the Video Player accordingly.

Definitions Module: The Definitions module should provide a set of actions in order to define how to tune the platform to user needs. The actions should clarify if the player must adapt automatically to

bandwidth changes or leave that to the user's responsibility (manually request the desired quality level).

Help Module: The Help module should inform users on how to interact with the client application. That includes the available keyboard shortcuts, the possible actions or how to change the default definitions to adapt the platform to his needs.

Animation Module: This module takes visual actions based on the user interaction. Its purpose is to animate and at the same time provide a feedback of the user interaction with the platform. Changing the channel, mute, change volume, are some of the actions that should be recognised and animated by this module.

All the presented modules will have an associated Cascading Style Sheets (CSS) file that will define the style and animations of each one. In that way, developers will also be able to change and work on these files in order to create new user interfaces independently of the module implementation.

This solution creates independence, reusability and customisation between modules. These characteristics permit modules to be easily integrated into another context or in different platforms.

IMPLEMENTATION

In order to start the implementation of the platform it was developed the HTTP streaming server and in a second step, the client side. The server implementation involved the resources analysis, signal capture from de DVB-T board, the transcoding parameters test, the TV listings API development and the configuration of the HTTP server. The client development involved the implementation and integration of the high order components, as well as, the deploy on a remote infrastructure. Section 4.1 refers to the server implementation and section 4.2 refers to the client player platform.

4.1 SERVER SIDE IMPLEMENTATION

To develop the proposed architecture (see figure 3.3 present on section 3.4), the signals captured from the DVB-T were firstly analysed to understand how the information is transmitted, the subsection 4.1.2 refers to that analysis and to the setup of the MuMuDVB tool, used to read, using one dongle, all the channels and create a unicast address for each one. Next, in section 4.1.3, there are explained the steps made on the transcoding block in order to provide different qualities to the server clients.

The subsection 4.1.4 refers to the explanation of the TV listings process and how the created API is provided to the users. Finally, on section 4.1.5, it is explained how the multimedia channels are prepared to the delivery and how they are served after the preparation.

4.1.1 SERVERS

There are three machines available, connected to the same Local Area Network (LAN), to implement this architecture. One of them has the purpose to serve the contents to the Internet (aquilon) as of the others are for transcoding purposes and aren't exposed to the Internet (iptv-transc-server and iptvs2). The table 4.1 shows some of the hardware and software specifications for all the machines.

	aquilon	iptv-transc-server	iptvs2
CPU	Core2 Duo E8500 2x3.17 GHz	Core i7-4790 8x4 GHz	Core i7-7700 8x4.2 GHz
Memory	4 GB	8 GB	8 GB
Storage	230GB	900 GB	220GB
OS	Ubuntu 16.04.1 LTS	Ubuntu 16.04.2 LTS	Ubuntu 16.04.2 LTS
IP	10.13.1.135 193.136.92.135	10.13.1.18	10.13.1.19
DNS	iptv.atnog.org	iptvs.atnog.av.it.pt	iptv2.atnog.av.it.pt

Table 4.1: Server machines specifications.

The figure above shows the network configuration where the server machines are inserted. The internal network (193.136.93.0/24) is connected to all the machines and the aquilon server has the port 80 open to serve the contents to clients. All other ports (of all machines) are only accessible inside the same LAN.

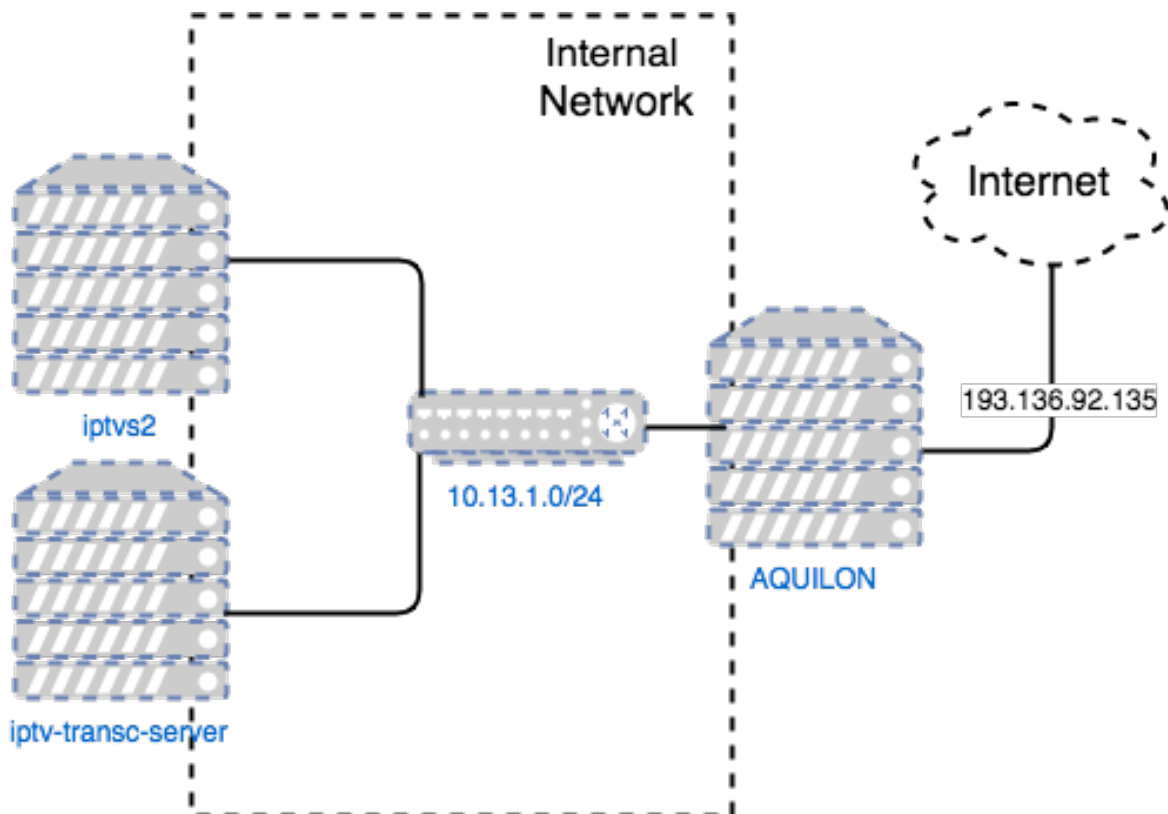


Figure 4.1: Network diagram of the IPTV server.

4.1.2 DVB-T SETUP

The first step, in this task, was to plug into the server the DVB-T USB dongle and check the drivers and signal state. Inside this dongle, there is a Realtek-RTL2832U DVB-T Demodulator with an USB 2.0 interface ¹ and an R820T tuner.

As stated in LinuxTV wiki, on Ubuntu systems, since version 13.10, the drivers for this device are shipped with the kernel. For earlier versions and other linux distributions, the LinuxTV drivers support this demodulator model.[100]

After configuring the proper drivers, some DVB LinuxTV utilities were tested. By using `w_scan` utility it is possible to generate a channels configuration file. The *code 7* shows the command-line interaction with this tool. The `-f` flag specifies the frontend (terrestrial, in this case), the `-c` flag specifies the country (PT for Portugal) and the `-x` flag specifies that the output format will be compatible with zap/xine formats.

```
% sudo w_scan -f t -c PT -X > chtest2.conf
```

Code 7: `w_scan` command to generate the zap configuration file.

The figure 4.2 shows the output file generated by the previous command. Note that the parameters are split by ":". As we can see, all the channels are received in the frequency 754000000 Hz and there are 7 available channels:

- RTP 1;
- RTP 2;
- SIC;
- TVI;
- ARTV;
- RTP 3;
- RTP Memória;

```
tiago@aquilon:~$ cat chtest2.conf
RTP 1:754000000:INVERSION_AUTO:BANDWIDTH_8_MHZ:FEC_2_3:FEC_NONE:QAM_64:TRANSMISSION_MODE_8K:GUARD_INTERVAL_1_4:HIERARCHY_NONE:256:257:1101
RTP 2:754000000:INVERSION_AUTO:BANDWIDTH_8_MHZ:FEC_2_3:FEC_NONE:QAM_64:TRANSMISSION_MODE_8K:GUARD_INTERVAL_1_4:HIERARCHY_NONE:512:513:1102
SIC:754000000:INVERSION_AUTO:BANDWIDTH_8_MHZ:FEC_2_3:FEC_NONE:QAM_64:TRANSMISSION_MODE_8K:GUARD_INTERVAL_1_4:HIERARCHY_NONE:768:769:1103
TVI:754000000:INVERSION_AUTO:BANDWIDTH_8_MHZ:FEC_2_3:FEC_NONE:QAM_64:TRANSMISSION_MODE_8K:GUARD_INTERVAL_1_4:HIERARCHY_NONE:1024:1025:1104
ARTV:754000000:INVERSION_AUTO:BANDWIDTH_8_MHZ:FEC_2_3:FEC_NONE:QAM_64:TRANSMISSION_MODE_8K:GUARD_INTERVAL_1_4:HIERARCHY_NONE:1280:1281:1107
RTP 3:754000000:INVERSION_AUTO:BANDWIDTH_8_MHZ:FEC_2_3:FEC_NONE:QAM_64:TRANSMISSION_MODE_8K:GUARD_INTERVAL_1_4:HIERARCHY_NONE:2048:2049:1108
RTP Memória:754000000:INVERSION_AUTO:BANDWIDTH_8_MHZ:FEC_2_3:FEC_NONE:QAM_64:TRANSMISSION_MODE_8K:GUARD_INTERVAL_1_4:HIERARCHY_NONE:2304:2305:1109
tiago@aquilon:~$ █
```

Figure 4.2: Configuration file generated by `w_scan` tool.

Using the previously generated file it is possible to tune into the desired channel using `tzap` tool. The *code 8* shows the command-line interaction to tune into the "SIC" channel. The `-c` flag specifies the configuration file.

¹<http://www.realtek.com.tw/products/productsView.aspx?Langid=1&PFid=35&Level=4&Conn=3&ProdID=257>

```
% sudo tzap -c chtest2.conf "SIC"
```

Code 8: tzap command to tune into the desired channel based on the configuration file.

To serve the public channels, it is important to check which channels are broadcasted and the media technical information such as, bitrates, resolutions, Frames-per-second (FPS), codecs for audio and video and others. After tuning into the desired channel it is possible to obtain information about the transmitted media and start viewing the tuned channel. At this step, we can use FFMPEG to check the media information and dvbsnoop to analyse the DVB reception. Starting with the dvbsnoop, it is possible to perform a PID scan to see what is being broadcasted.

DVBSNOOP

Dvbsnoop is a command line based program with the purpose to debug, dump or view digital stream information, like digital Television broadcasted via cable, satellite or terrestrial. It is also possible to analyse MPEG streams. This tool can be helpful to read live stream information in a human readable form.[101]

The figure 4.3 shows the obtained result of the performed scan. The PID 0 has the Program Association Table (PAT) information, the PID 16 has the Network Information Table (NIT) information, the PID 17 has the Service Description Table (SDT) information (not present on the exact time of the capture of this example but regularly broadcasted), the PID 18 has the Event Information Table (EIT) information, the PID 20 transports the Time Date Table (tdt) information and each channel has a section dedicated to the Program Map Table (PMT) , for example in PID 272.

```

tiago@aquilon:~$ sudo dvbsnoop -s pidscan
dvbsnoop V1.4.50 -- http://dvbsnoop.sourceforge.net/

-----
Transponder PID-Scan...
-----
PID found:  0 (0x0000) [SECTION: Program Association Table (PAT)]
PID found: 16 (0x0010) [SECTION: Network Information Table (NIT) - actual network]
PID found: 18 (0x0012) [SECTION: Event Information Table (EIT) - actual transport stream, schedule]
PID found: 20 (0x0014) [SECTION: Time Date Table (TDT)]
PID found: 21 (0x0015) [SECTION: ITU-T Rec. H.222.0|ISO/IEC13818 reserved]
PID found: 32 (0x0020) [PS/PES: private_stream_1]
PID found: 34 (0x0022) [PS/PES: private_stream_1]
PID found: 35 (0x0023) [PS/PES: private_stream_1]
PID found: 256 (0x0100) [PS/PES: ITU-T Rec. H.262 | ISO/IEC 13818-2 or ISO/IEC 11172-2 video stream]
PID found: 257 (0x0101) [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio stream]
PID found: 258 (0x0102) [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio stream]
PID found: 272 (0x0110) [SECTION: Program Map Table (PMT)]
PID found: 512 (0x0200) [PS/PES: ITU-T Rec. H.262 | ISO/IEC 13818-2 or ISO/IEC 11172-2 video stream]
PID found: 513 (0x0201) [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio stream]
PID found: 514 (0x0202) [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio stream]
PID found: 528 (0x0210) [SECTION: Program Map Table (PMT)]
PID found: 768 (0x0300) [PS/PES: ITU-T Rec. H.262 | ISO/IEC 13818-2 or ISO/IEC 11172-2 video stream]
PID found: 769 (0x0301) [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio stream]
PID found: 784 (0x0310) [SECTION: Program Map Table (PMT)]
PID found: 1024 (0x0400) [PS/PES: ITU-T Rec. H.262 | ISO/IEC 13818-2 or ISO/IEC 11172-2 video stream]
PID found: 1025 (0x0401) [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio stream]
PID found: 1026 (0x0402) [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio stream]
PID found: 1040 (0x0410) [SECTION: Program Map Table (PMT)]
PID found: 1280 (0x0500) [PS/PES: ITU-T Rec. H.262 | ISO/IEC 13818-2 or ISO/IEC 11172-2 video stream]
PID found: 1281 (0x0501) [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio stream]
PID found: 1296 (0x0510) [SECTION: Program Map Table (PMT)]
PID found: 2048 (0x0800) [PS/PES: ITU-T Rec. H.262 | ISO/IEC 13818-2 or ISO/IEC 11172-2 video stream]
PID found: 2049 (0x0801) [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio stream]
PID found: 2064 (0x0810) [SECTION: Program Map Table (PMT)]
PID found: 2304 (0x0900) [PS/PES: ITU-T Rec. H.262 | ISO/IEC 13818-2 or ISO/IEC 11172-2 video stream]
PID found: 2305 (0x0901) [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio stream]
PID found: 2320 (0x0910) [SECTION: Program Map Table (PMT)]
PID found: 8191 (0x1fff) [stuffing]
tiago@aquilon:~$ █

```

Figure 4.3: PID scan using dvbsnoop.

The PAT must be present in every MPEG2-TS that contains multiple program services. It lists the program number and the location of the PMT, as well as, the location of the NIT.[102] The figure 4.4 shows the contents of the PAT in a given moment, in our scenario.

```

tiago@aquilon:~$ sudo dvbsnoop -n 1 0
dvbsnoop V1.4.50 -- http://dvbsnoop.sourceforge.net/

-----
SECT-Packet: 00000001  PID: 0 (0x0000), Length: 44 (0x002c)
Time received: Mon 2017-05-15 19:18:00.264
-----
0000: 00 b0 29 04 4d f3 00 00 00 00 e0 10 04 4d e1 10  ..).M.....M..
0010: 04 4e e2 10 04 4f e3 10 04 50 e4 10 04 53 e5 10  .N...0...P...S..
0020: 04 54 e8 10 04 55 e9 10 00 9b 8d b1                .T...U.....

PID: 0 (0x0000) [= assigned for: ISO 13818-1 Program Association Table (PAT)]

Guess table from table id...
PAT-decoding...
Table_ID: 0 (0x00) [= Program Association Table (PAT)]
section_syntax_indicator: 1 (0x01)
(fixed): 0 (0x00)
reserved_1: 3 (0x03)
Section_length: 41 (0x0029)
Transport_Stream_ID: 1101 (0x044d)
reserved_2: 3 (0x03)
Version_number: 25 (0x19)
current_next_indicator: 1 (0x01) [= valid now]
Section_number: 0 (0x00)
Last_Section_number: 0 (0x00)

    Program_number: 0 (0x0000)
    reserved: 7 (0x07)
    Network_PID: 16 (0x0010)

    Program_number: 1101 (0x044d)
    reserved: 7 (0x07)
    Program_map_PID: 272 (0x0110)

    Program_number: 1102 (0x044e)
    reserved: 7 (0x07)
    Program_map_PID: 528 (0x0210)

    Program_number: 1103 (0x044f)
    reserved: 7 (0x07)
    Program_map_PID: 784 (0x0310)

    Program_number: 1104 (0x0450)
    reserved: 7 (0x07)
    Program_map_PID: 1040 (0x0410)

    Program_number: 1107 (0x0453)
    reserved: 7 (0x07)
    Program_map_PID: 1296 (0x0510)

    Program_number: 1108 (0x0454)
    reserved: 7 (0x07)
    Program_map_PID: 2064 (0x0810)

    Program_number: 1109 (0x0455)
    reserved: 7 (0x07)
    Program_map_PID: 2320 (0x0910)

CRC: 10194353 (0x009b8db1)

```

Figure 4.4: Program Association Table in a given moment.

The NIT provides information related to the physical multiplex, transports streams carried in the network and the characteristics of the network.[103] The appendix B shows an example of the NIT captured in a given moment, in this scenario. By analysing the example, we can see that we are capturing the streams from the "TDT Continente" network. We can also see that all the seven broadcasted channels format is received in SD digital television service and consecutively not in HD.

The Event Information Table transports the information to build the EPG system. It contains

information about the programs for each broadcasted channel (start/end time, description, title and others). The figure 4.5 shows a snippet of this table. As we can observe in the snippet, the service id refereed is 1109. On this service or channel, it is broadcasted on 20/05/2017 at 15:26 (UTC) the event "Guarda-Factos" in Portuguese.

```

PID: 18 (0x0012) [= assigned for: DVB Event Information Table (EIT)]

Guess table from table id...
EIT-decoding...
Table_ID: 81 (0x51) [= Event Information Table (EIT) - actual transport stream, schedule]
section_syntax_indicator: 1 (0x01)
reserved_1: 1 (0x01)
reserved_2: 3 (0x03)
Section_length: 1171 (0x0493)
Service_ID: 1109 (0x0455) [= --> refers to PMT program_number]
reserved_3: 3 (0x03)
Version_number: 8 (0x08)
current_next_indicator: 1 (0x01) [= valid now]
Section_number: 104 (0x68)
Last_Section_number: 184 (0xb8)
Transport_stream_ID: 1101 (0x044d)
Original_network_ID: 8904 (0x22c8) [= >>ERROR: not (yet) defined... Report!<<]
Segment_last_Section_number: 104 (0x68)
Last_table_id: 81 (0x51) [= Event Information Table (EIT) - actual transport stream, schedule]

Event_ID: 39831 (0x9b97)
Start_time: 0xe225152600 [= 2017-05-20 15:26:00 (UTC)]
Duration: 0x0002500 [= 00:25:00 (UTC)]
Running_status: 0 (0x00) [= undefined]
Free_CA_mode: 0 (0x00) [= unscrambled]
Descriptors_loop_length: 402 (0x192)

DVB-DescriptorTag: 77 (0x4d) [= short_event_descriptor]
descriptor_length: 204 (0xcc)
ISO639_2_language_code: por
event_name_length: 14 (0x0e)
event_name: "Guarda-Factos" -- Charset: reserved
text_length: 185 (0xb9)
text_char: "Crónica Mensal sobre acontecimentos que marcaram o país e o mundo, nesse mês, ao longo dos a

DVB-DescriptorTag: 78 (0x4e) [= extended_event_descriptor]
descriptor_length: 194 (0xc2)
descriptor_number: 0 (0x00)
last_descriptor_number: 0 (0x00)
ISO639_2_language_code: por
length_of_items: 0 (0x00)

text_length: 188 (0xbc)
text: "Crónica Mensal sobre acontecimentos que marcaram o país e o mundo, nesse mês, ao longo dos anos.

```

Figure 4.5: Event Information Table snippet.

The association between the channel number and the service description specifics like channel name and service type is present on the SDT. The figure 4.6 shows the example for the service id 1109, obtained on the analysis of the EIT previously. As we can see, the ID 1109 refers to the "RTP Memória" channel.

```

Service_id: 1109 (0x0455) [= --> refers to PMT program_number]
reserved_1: 63 (0x3f)
EIT_schedule_flag: 1 (0x01)
EIT_present_following_flag: 1 (0x01)
Running_status: 4 (0x04) [= running]
Free_CA_mode: 0 (0x00) [= unscrambled]
Descriptors_loop_length: 27 (0x001b)

    DVB-DescriptorTag: 72 (0x48) [= service_descriptor]
    descriptor_length: 16 (0x10)
    service_type: 22 (0x16) [= advanced codec SD digital television service]
    service_provider_name_length: 0 (0x00)
    service_provider_name: ""
    service_name_length: 13 (0x0d)
    Service_name: "RTP Memória" -- Charset: reserved

    DVB-DescriptorTag: 74 (0x4a) [= linkage_descriptor]
    descriptor_length: 7 (0x07)
    transport_stream_ID: 1101 (0x044d)
    original_network_ID: 8904 (0x22c8) [= >>ERROR: not (yet) defined... Report!<<]
    service_ID: 65535 (0xffff) [= --> refers to PMT program_number]
    linkage_type: 2 (0x02) [= EPG service]

```

Figure 4.6: Service Description Table snippet.

Each channel has a Program Map Table associated. Each section of this document specifies the "characteristics of each elementary stream in the program service".[104]. The figure 4.7 shows an example snippet for our scenario, for the service ID 1109.

```

PID: 2320 (0x0910)

Guess table from table id...
PMT-decoding...
Table_ID: 2 (0x02) [= Program Map Table (PMT)]
section_syntax_indicator: 1 (0x01)
(fixed '0'): 0 (0x00)
reserved_1: 3 (0x03)
Section_length: 41 (0x0029)
Program_number: 1109 (0x0455)
reserved_2: 3 (0x03)
Version_number: 25 (0x19)
current_next_indicator: 1 (0x01) [= valid now]
Section_number: 0 (0x00)
Last_Section_number: 0 (0x00)
reserved_3: 7 (0x07)
PCR_PID: 2304 (0x0900)
reserved_4: 15 (0x0f)
Program_info_length: 0 (0x0000)

Stream_type loop:

Stream_type: 6 (0x06) [= ITU-T Rec. H.222.0 | ISO/IEC 13818-1 PES packets containing private data]
reserved_1: 7 (0x07)
Elementary_PID: 32 (0x0020)
reserved_2: 15 (0x0f)
ES_info_length: 7 (0x0007)

    DVB-DescriptorTag: 86 (0x56) [= teletext_descriptor]
    descriptor_length: 5 (0x05)
        ISO639_language_code: por
        Teletext_type: 1 (0x01) [= initial teletext page]
        Teletext_magazine_number: 1 (0x01)
        Teletext_page_number: 0 (0x00)

Stream_type: 27 (0x1b) [= AVC video stream as defined in ITU-T Rec. H.264 | ISO/IEC 14496-10 Video]
reserved_1: 7 (0x07)
Elementary_PID: 2304 (0x0900)
reserved_2: 15 (0x0f)
ES_info_length: 0 (0x0000)

Stream_type: 17 (0x11) [= ISO/IEC 14496-3 Audio with LATM transport syntax as def. in ISO/IEC 14496-3/AMD1]
reserved_1: 7 (0x07)
Elementary_PID: 2305 (0x0901)
reserved_2: 15 (0x0f)
ES_info_length: 6 (0x0006)

    MPEG-DescriptorTag: 10 (0x0a) [= ISO_639_language_descriptor]
    descriptor_length: 4 (0x04)
        ISO639_language_code: por
        Audio_type: 0 (0x00) [= undefined]

```

Figure 4.7: Program Map Table snippet.

There's also a Time Date Table. This table informs the receptors when will occur a time update and provides the current time and date information. The figure 4.8 exemplifies this table in our scenario, as we can be seen in 29/10/2017 the time will shift in one hour.

```

tiago@aquilon:~$ sudo dvbsnoop -n 1 20
[sudo] password for tiago:
dvbsnoop V1.4.50 -- http://dvbsnoop.sourceforge.net/

-----
SECT-Packet: 00000001  PID: 20 (0x0014), Length: 42 (0x002a)
Time received: Mon 2017-05-15 22:01:08.874
-----
 0000: 73 70 27 e2 20 21 01 07  f0 1c 58 1a 50 52 54 06  sp'. !....X.PRT.
 0010: 01 00 e2 c7 01 00 00 00  00 50 52 54 0b 00 00 e2  .....PRT....
 0020: c7 01 00 00 01 00 b6 07  c8 05  .....

PID: 20 (0x0014) [= assigned for: DVB Time and Date Table (TDT), Time Offset Table (TOT)]

Guess table from table id...
TOT-decoding...
Table_ID: 115 (0x73) [= Time Offset Table (TOT)]
section_syntax_indicator: 0 (0x00)
reserved_1: 1 (0x01)
reserved_2: 3 (0x03)
Section_length: 39 (0x0027)
UTC_time: 0xe220210107 [= 2017-05-15 21:01:07 (UTC)]
reserved_3: 15 (0x0f)
Descriptor_loop_length: 28 (0x001c)

    DVB-DescriptorTag: 88 (0x58) [= local_time_offset_descriptor]
    descriptor_length: 26 (0x1a)
        Country_code: PRT
        Country_region_ID: 1 (0x01)
        reserved_1: 1 (0x01)
        local_time_offset_polarity: 0 [= plus to UTC]
        Local_time_offset: 01:00
        Time_of_change: 0xe2c7010000 [= 2017-10-29 01:00:00 (UTC)]
        Next_time_offset: 00:00

        Country_code: PRT
        Country_region_ID: 2 (0x02)
        reserved_1: 1 (0x01)
        local_time_offset_polarity: 1 [= minus to UTC]
        Local_time_offset: 00:00
        Time_of_change: 0xe2c7010000 [= 2017-10-29 01:00:00 (UTC)]
        Next_time_offset: 01:00

CRC: 3053963269 (0xb607c805)
=====

```

Figure 4.8: Time Date Table snippet.

Figure 4.9 shows the media information, captured by ffmpeg, when the receptor is tuned to the "SIC" channel. In order to obtain the channel, the path must be specified the path to the adapter (/dev/dvb/adaptor0/dvr0, in this case). As can be observed, the broadcasted video is encoded in H.264 profile Main, with yuv420p pixel format at 25 Frames-per-second. The video resolution is 720x576 corresponding to PAL format. In terms of audio, the codec is AAC_latm with a 48000 Hz frequency.

```

Input #0, mpegts, from '/dev/dvb/adaptor0/dvr0':
  Duration: N/A, start: 21314.571089, bitrate: N/A
    Stream #0:0[0x300]: Video: h264 (Main), yuv420p(tv, bt470bg), 720x576 [SAR 16:11 DAR 20:11], 25 fps, 50 tbr, 90k tbn, 50 tbc
    Stream #0:1[0x301]: Audio: aac_latm (HE-AAC), 48000 Hz, stereo, fltp

```

Figure 4.9: Media information captured by FFMPEG tool.

At this stage, we have one board tuned to one channel meaning that following this procedure, we will have to use 7 boards to tune into the seven available channels. Also, since we want to process the media, in the transcoding process and in different machines, we will have to setup physical dongles on each machine. For that reason, we use the `mumudvb`.

MUMUDVB

MuMuDVB (extends for Multi Multicast DVB) is a free linux software created over the linux DVB API and is capable to redistribute DVB sources in multicast or HTTP unicast. This tool is able to decode all channels on a given transport stream and therefore, with one receptor, it is possible to provide all the broadcasted channels to the network.[105][106]

As previously stated, by using MuMuDVB, each channel can be transmitted over unicast or multicast. In our case, we choose unicast because we will only have one client for each channel. Converting the received streams to unicast is also beneficial because we can access everywhere inside our internal network (LAN). Figure 4.10 presents the configuration file used in our scenario. When using the `autoconfiguration=full` option we don't have to specify any channel, the `sap=0` option is set to disable SAP announces (useful for multicast contexts), the `multicast=0` option is set to disable multicast, the `freq=754` is the specification of the frequency of DVB-T in Portugal (as analysed previously with `dvbsnoop`) and at last, the `autoconf_unicast_start_port=1234` specifies which port will start being occupied by `mumudvb`. In our case, since we have seven channels, the ports 1234 to 1240 will be dedicated to each channel.

```
tiago@aquilon:~$ cat mumu.conf
autoconfiguration=full
sap=0
unicast=1
freq=754
autoconf_unicast_start_port=1234
multicast=0
```

Figure 4.10: `mumudvb` configuration file.

To start using this software, we need to run a command similar to the one represented in the *code 9*. The `-d` flag specifies that MuMuDVB should not daemonize and therefore is optional. The `-c` flag exposes the configuration file to the program instructing how it should behaviour.

```
% sudo mumudvb -d -c mumu.conf
```

Code 9: Example of `mumudvb` usage based on a configuration file.

In order to have this software constantly running on the servers, was created an entry on the `linux.rc.local` file. This procedure guarantees that MuMuDVB is started on every boot. The presence of the Linux Screen (command `"screen"`) enables users to run a long running process without maintaining an

active shell session. At anytime it is possible to attach to the screen and analyse the program status. The *code 10* shows this usage.

```
% screen -Sdm "mumu" "mumudvb" "-d" "-c" "/home/tiago/mumu.conf"
```

Code 10: Command that runs on system boot in order to launch mumudvb.

To reduce the load on the internal network, we installed one dongle per each machine and followed the MuMuDVB configuration procedure previously stated, for each one.

4.1.3 TRANSCODING

In this task, we will proceed to the audio/video transcoding, for each channel, into different qualities. Each quality will be sent through an RTMP stream to the streaming server location. First, for testing purposes, we just read the input of one channel ("SIC" channel) and send it to the network to an RTMP location, using the FFmpeg tool.

To install this software, we followed the compilation guide for Ubuntu present on the FFmpeg's wiki.[107] After the installation process, the *code 11* was executed and the result, present on figure 11, was compared to the media information captured from the DVB-T receptor(see figure 4.9).

```
% ffmpeg -i http://iptv.atnog.org:1236
```

Code 11: FFmpeg's command to inspect input stream properties.

As shown in figure 4.11, by using the MuMuDVB tool it is possible to obtain some metadata like service name and an additional stream referring to the teletext (Stream #0:0). In addition the audio and video properties remain the same as the obtained when manually tuned using the tzap tool (see subsection 4.1.2).

```
Input #0, mpegts, from 'http://iptv.atnog.org:1236':
Duration: N/A, start: 93387.951433, bitrate: N/A
Program 1103
Metadata:
  service_name      : ?SIC
  service_provider :
Stream #0:0[0x22](por): Subtitle: dvb_teletext ([6][0][0][0] / 0x0006), 492x250
Stream #0:1[0x300]: Video: h264 (Main) ([27][0][0][0] / 0x001B), yuv420p(tv, bt470bg), 720x576 [SAR 16:11 DAR 20:11], 25 fps, 50 tbr, 90k tbn, 50 tbc
Stream #0:2[0x301](por): Audio: aac_latm (HE-AAC) ([17][0][0][0] / 0x0011), 48000 Hz, stereo, fltp
```

Figure 4.11: Stream properties originated by the MuMuDVB tool.

After the previous test, we added to the FFmpeg command an RTMP source and copied audio and video parameters, and the program showed an error since the audio codec is incompatible with FLV containers present on RTMP streams. The *code 12* shows the command executed in an attempt to only specify the audio codec and bitrate. The *-c:v* and *-c:a* options specifies the codecs for video and audio respectively. In this case, we are using AAC with the bitrate of 48000 Hz. This attempt resulted in an audio only stream (tested with the VLC player and *ffplay*).

```
%ffmpeg -i http://0.0.0.0:1236 -c:v copy -c:a aac -strict -2 -f flv \  
% rtmp://iptv.atnog.org:1247/hls/teste
```

Code 12: FFMPEG's video codec copy mode test.

The next step was to include the libx264 codec, the one recommended by this tool. In their encoding guide, they also refer that for the x264 encode, the systems do not leverage of graphic card. Although, if there is a need to free CPU load, FFMPEG provides a subsystem for hardware acceleration. We also included the constant rate frame (CRF) flag. This method allows achieving a certain quality for the cases when the variance of file size is not extremely important. This provides efficiency with a single pass. The CRF scale varies between 0 and 51, where 0 is lossless video and 51 is the worst quality possible. The default value is 23 but to achieve a visually lossless video we should use 18. The CRF itself doesn't care about the media size however we also have at our disposal a collection of options that combine the compression ratio and the encoding speed. The slower preset will provide the best quality per size ratio (compression). The problem is that for live scenarios we need to use a faster preset. Optionally we can use the `-tune` option that includes a set of default specifics based on the input (zerolatency, fastdecode, film and others). Additionally, we have an optional setting, the video profile (`-profile:v`) which is incompatible with lossless encoding because its purpose is to limit the output to a specific H.264 profile.[108][109]

Based on that information, we started to build our streams. Since the goal is to reach all devices and provide an adaptive streaming experience, we created 3 different transcoding specifications divided by quality level, consequently we will have:

- High level;
- Medium level;
- Low level;

At the first level we will not care about the devices compatibility or about the media bitrate. The goal, at this level, is to provide the source quality to the end users. The *code 13* shows the configuration command for that level.

```
%ffmpeg -i http://0.0.0.0:1236 -vf yadif -c:v libx264 -c:a aac -strict -2 -f flv \  
% rtmp://iptv.atnog.org:1247/hls/teste_high
```

Code 13: FFMPEG command line configuration that generates the High level stream.

The second and third levels should be compatible with Android and iOS devices. Based on the information stated in section 2.9 and in the compatibility guide provided by Apple in the HLS scope [77], we combine both device requirements to create the following specifications:

	Medium	Low
Video resolution	480x360	176x144
FPS	25	12
Bitrate	500 Kbps	128 Kbps
Audio bit rate	128 Kbps	48 Kbps
H264 Profile	Baseline 3.1	Baseline 3.1

Table 4.2: Medium and Low H.264 specifications to increase device compatibility.

At this moment, we need to create 2 additional streams for each channel. In order to read the streaming once and transcode to the multiple variants, we should run FFMPEG with the `-threads` flag. This flag specifies how the tread management should be handled by this tool. If `-threads` is set to 0, FFMPEG will manage it automatically however it is possible to set the number of threads that FFMPEG should use. For example, `-threads 1` will only use one thread to perform the transcoding. The *code 14* shows the test example for this case. The used options are:

- `-i` to specify the input;
- `-vf yadif` to deinterlace the input;
- `-c:v` to specify the video codec;
- `-c:a` to specify the audio codec;
- `-crf` to specify the constant rate frame value;
- `-threads` to tell how FFMPEG should manage the CPU threads;
- `-b:v` to quantify the video bit rate;
- `-b:a` to quantify the audio bit rate;
- `-r` to specify the number of FPS;
- `-profile:v` to specify the video profile;
- `-f` to choose the output format;
- `-pix_fmt` to specify the pixel format;

```
% ffmpeg -i http://iptv.atnog.org:1236 -vf yadif -c:v libx264 -crf 18 -pix_fmt yuv420p -c:a aac \
% -strict -2 -threads 0 -f flv rtmp://iptv.atnog.org:1247/hls/teste_high \
% -s 480*360 -c:v libx264 -b:v 500k -pix_fmt yuv420p -r 25 -c:a aac -strict -2 -b:a 128k \
% -profile:v baseline -level 3.1 -threads 0 -f flv rtmp://iptv.atnog.org:1247/hls/teste_mid \
% -s 176*144 -c:v libx264 -b:v 128k -pix_fmt yuv420p -r 12 -c:a aac -strict -2 -b:a 48k \
% -profile:v baseline -level 3.1 -threads 0 -f flv rtmp://iptv.atnog.org:1247/hls/teste_low
```

Code 14: FFMPEG command line configuration that generates the High, Medium and Low streaming levels.

As stated in section 3.4 and based on figure 3.3, the transcoding process should be made on any machine inside of the same Local Area Network. To accomplish that we decided to run our transcoding block inside Docker containers. The used image tiagofm/phusion_baseimage is a customised image, based on the phusion/baseimage². We have configured the Secure Shell (SSH) access, compiled the FFMPEG and installed other required dependencies (like Linux screen) over it.

While adding more channels, it becomes more complex to manage and monitor all the containers and the machines that powers them. For that reason, we decided to include a Docker manager, the Rancher. This software provides us with a simple User Interface with network and hardware statistics and control over all the containers across the machines. To set up this platform, the first step is to install it on the desired machine and map a network port to access the UI. In our scenario, the rancher is running on the aquilon machine in the port 8080. The *code 15* shows the command executed on the machine.

```
% docker run -d --restart=unless-stopped -p 8080:8080 rancher/server:stable
```

Code 15: Rancher (stable version) platform install on aquilon machine.

Adding a custom host to the platform, in this case the other two machines, that will support the transcoding job, is simple. The Rancher User Interface provides a command to paste on the desired machines where we just need to configure the IP address. Once this step is completed, it is possible to see and control the containers running on all the servers.

As mentioned in section 5.2, and in order to avoid CPU overload, it was decided to assign to the iptvs 3 of the channels and to the iptv2 the remaining four. On the iptvs machine, we transcode the channels RTP1, SIC and TVI. On the iptvs2 machine, we transcode the channels RTP2, RTP3, ARTV and RTP Memória. Each Docker container, related to each channel, has a shell script that ensures that the FFMPEG program is always running. This script is called after the boot and runs inside a Linux Screen. The list of the addresses for access each container, via ssh, is:

- **RTP1**- iptvs.atnog.av.it.pt, port 2201;
- **RTP2**- iptv2.atnog.av.it.pt, port 2213;
- **SIC**- iptvs.atnog.av.it.pt, port 2204;
- **TVI**- iptvs.atnog.av.it.pt, port 2203;
- **RTP3**- iptv2.atnog.av.it.pt, port 2211;
- **ARTV**- iptv2.atnog.av.it.pt, port 2215;
- **RTP Memória**- iptv2.atnog.av.it.pt, port 2212;

4.1.4 TV LISTINGS

To extract the TV listings that will feed the EPG service on the client side, we need to extract the programming information from the DVB-T signal. As analysed in section 4.1.2, the programming information is transported in a PMT table (see image 4.7). Next, we present some tools that aim to extract or provide this information.

²<https://github.com/phusion/baseimage-docker>

TV_GRAB_DVB

TV_GRAB_DVB is an open-source Linux based tool that dumps EPG data into XMLTV format. This tool has the purpose to extract the data from DTV broadcasts. To run this software, is required to have a DVB card working on Linux and tuned to a multiplex. The Code 16 shows the program usage.

```
% tv_grab_dvb [-t timeout] > output_file.xml
```

Code 16: Example of the interaction with TV_GRAB_DVB.

As a downside, TV_GRAB_DVB is not maintained anymore, it doesn't decode channel names from broadcast (names are decoded from the tuning files used by tzap) and doesn't allow to specify the DVB card to use.[110]

EPG FOR IPTV

EPG for IPTV is a service that uses XMLTV format to provide to its users a personalised EPG. This software provides data for TV channels for all of Europe, China, Japan, North and South America and other countries. The data provided to the users is updated every day in order to guarantee content precision in terms of programming information.

This is a paid service and offers distinct payment plans for business or end users. Opposite to end-users, the business plan has no country restrictions.[111][112]

OVERVIEW

In this subsection we compare the TV listings tools/services previously analysed, to point out the benefits of using each one.

	EPG for IPTV	TV_GRAB_DVB
Open-source	No	Yes
Free	No	Yes
Data source	IP	DVB
Delivery format	XMLTV	XMLTV

Table 4.3: Comparison between TV listings data providers.

As Table 4.3 shows, the main benefit of using TV_GRAB_DVB is the fact of being a free, open-source tool. Yet, it needs a DVB linux compatible board to extract the broadcasted information. On the other hand, EPG for IPTV service provides channel data from a large set of countries, being useful when user needs Television listings from non public channels.

In order to successful run the TV_GRAB_DVB program we must have the board tuned in the frequency (with tzap or MuMuDVB). The *code 17* shows the usage of this tool. Optionally we can set a timeout value.

```
% tv_grab_dvb [-t timeout] > tvlistings.xml
```

Code 17: tv_grab_dvb tool usage.

This tool is called once every five minutes to update the output file when a change is made in the programming. To achieve this, we used the system's crontab. The problem with this method is that the file generated doesn't decode the channel ID to the channel name, and it also relies on a file transfer to clients in order to provide the programming information. For that reason, we have created a REST API that provides endpoints to get not only the channel's programming but also the available server channels and the channels logos. The API was built in Python using the Flask³ framework and to improve response times, it was activated a cache system on the requests.

The API endpoints are:

/ OR /INDEX

This endpoint renders the API description page that contains all the available methods;

/EPG/CURRENT/{CHANNEL}

This endpoint returns the current program for the specified channel, in JSON format.

```
{  
  "channel": "RTP1",  
  "desc": "As tardes estao cheias de surpresas, jogos, dicas, alegria e muito humor.  
Tania Ribas de Oliveira e Ze Pedro Vasconcelos vao estar consigo de segunda a  
sexta no \"Agora Nos\". A eles juntam-se Joana Teles e Tiago Goes Ferreira quer  
em estúdio, quer no exterior, em reportagem.",  
  "end": "20170518180000",  
  "start": "20170518155000",  
  "title": "Agora Nos"  
}
```

Code 18: Example of the JSON object obtained when requested the RTP1 program that was being broadcasted.

/EPG/NEXT/{CHANNEL}/{SIZE}

The goal of this endpoint is to get a set of programs that will be broadcasted for the specified channel. The size specifies the number of programs to return in the answer, in JSON format.

³<http://flask.pocoo.org/>

```

{
  "1": {
    "channel": "RTPMEM",
    "desc": "Com adaptacao e dialogos do jornalista Joao Alves da Costa e de Francisco
    Nicholson, com autoria de Nicolau Breyner e Thilo Krasmann, cuja historia foi alvo,
    contudo, de uma \"recriacao\" em termos televisivos, \"Vila Faia\" destaca-se porque
    marca o inicio da telenovela portuguesa, realizada e interpretada respetivamente por
    profissionais e atores portugueses",
    "end": "20170519045300",
    "start": "20170519042500",
    "title": "Vila Faia"
  },
  "2": {
    "channel": "RTPMEM",
    "desc": "Ines Lopes Goncalves modera um debate com um painel de luxo: Fernando Alvim,
    Nuno Markl, Julio Isidro e um convidado especial. Numa conversa desempoeirada,
    lanca-se a semana da RTP Memoria a mesa, com os melhores conteudos e surpresas
    em destaque. Reciclamos o Cartaz TV e trazemos para a frente um debate divertido,
    mas doutorado, sobre o imaginario da televisao.",
    "end": "20170519050300",
    "start": "20170519045300",
    "title": "Traz pra Frente: Best of"
  }
}

```

Code 19: Example of the JSON object obtained when requested the next two programs for the RTP Memória channel.

`/EPG/PAST/{CHANNEL}/{SIZE}`

This endpoint works the same way that the previous one but returns the already broadcasted programs instead of the programs that will be broadcasted in the future.

`/LOGO/{CHANNEL}`

This method returns the logo of the requested channel.

`/PLAYLIST`

The goal of this endpoint is to return a playlist containing the address of each channel stream. The file format is .m3u witch is recognised by a large set of media players like VLC.


```
#EXTM3U
http://iptv.atnog.org/hls/rtp1.m3u8
http://iptv.atnog.org/hls/rtp2.m3u8
http://iptv.atnog.org/hls/sic.m3u8
http://iptv.atnog.org/hls/tvi.m3u8
http://iptv.atnog.org/hls/rtp3.m3u8
http://iptv.atnog.org/hls/rtpmem.m3u8
http://iptv.atnog.org/hls/artv.m3u8
```

Figure 4.12: M3U playlist file with the available channels.

All the previous endpoints are HTTP GET methods. The list of channels is:

- rtp1;
- rtp2;
- rtp3;
- rtpmem (RTP Memória)
- sic;
- tvi;
- artv;

/CHANNELS

With this endpoint is possible to provide an aggregate of essential data related to the available channels. The response includes a JSON file with the channel's name, source and logo. The appendix C illustrates the response for the current implementation.

Besides the previous API we also provided the access to some static files through the endpoint `/static/{filename}`. Through this endpoint it is possible to get the file generated by the `tv_grab_epg` (`epg.xml`).

4.1.5 SERVE THE CONTENTS

This subsection is related to the preparation and delivery of the media. Our server will receive the RTMP streams from the transcoding step and prepare it in order to follow the required streaming specifications. Additionally, the server should proxy to the TV listings API (see subsection 2.8) and to the Rancher interface (see subsection 4.1.3).

To serve the contents, we have chosen the NGINX server (see subsection 2.6.2). With this software is possible to serve media and proxy requests to the desired location.

The "nginx_rtmp_module" was used to perform the RTMP streams preparation to delivery it OTT. The installation of this module was made on compilation time, by cloning its GitHub repository⁴ and pointing the source code to the NGINX's install configuration script.

⁴<https://github.com/arut/nginx-rtmp-module>

The module configuration was made on the main configuration file "nginx.com", the *code 20* shows the configurations made to serve the channels via HLS and DASH. The reason to use both protocols was explained in subsection 2.5.4.

As we can see, first is created an RTMP server entry. Inside this entry are defined the applications and the server definitions. The listen directive specifies the open port to the RTMP streams. The chunk_size defines the maximum size for stream multiplexing (bigger value lowers the CPU overhead).

The applications are entries that define different streaming profiles. It can vary in terms of transport specification, variants and other playlist specifics. The "hls on" and "dash on" (marked at a)) directives enable the HLS and DASH streaming, respectively. The hls_fragment and dash_fragment (marked at b)) specifies the fragment size, in seconds. The nested mode (marked at c)) specifies that there should be created a subdirectory per each stream. The continuous mode (marked at d)) assures that the old fragments are kept in the playlist and the fragments sequence number is started from where it stopped last time. The playlist_length (marked at e)) sets the size of the recorded segments in terms of time, in this case, seconds. This directive limits how many fragments are stored in the server. The fragments outside the playlist_lenght are removed from the storage and the playlist. At f) we inform that the stream is a live stream, to avoid the players to start playing the streaming channels from the beginning of the playlist (where the older fragments are specified). Finally, the variant option specifies the name and the recommended bandwidth of the different media representations.

```

rtmp {
  server {
    listen 1247;
    chunk_size 4096;
    notify_method get;

    application hls {
      allow play all;
      live on;
      hls on; # a)
      hls_path /tmp/hls;
      hls_fragment 5s; # b)
      hls_playlist_length 3600s; # e)
      hls_continuous on; # d)
      hls_nested on; # c)
      hls_fragment_slicing aligned;
      hls_type live; # f) live-> stream played from the current live position

      # Instruct clients to adjust resolution according to bandwidth
      hls_variant _low BANDWIDTH=288000; # Low bitrate
      hls_variant _mid BANDWIDTH=448000; # Medium bitrate
      hls_variant _high BANDWIDTH=2500000; # Source bitrate
    }

    application dash {
      live on;
      dash on; # a)
      dash_path /tmp/dash;
      dash_fragment 5s; # b)
      dash_playlist_length 3600s; # e)
      dash_continuous on; # d)
      dash_nested on; # c)
    }
  }
}

```

Code 20: "nginx_rtmp_module" server configuration

Even though this module provides a way to stream using HLS and DASH, there are some limitations. The first is that this module segments the media into ".ts" files only which means that it's not possible to segment the media into fMP4 files. Segmentation in fMP4 allows to keep in the storage only one media format, compatible with both DASH and HLS improving the interoperability between them. The other limitation is the multiple representation support on DASH. The `nginx_rtmp_module` doesn't support multiple variants on this standard and consecutively the client can't adapt to the network conditions properly.

In order to serve the contents to the clients, we need to create directives to access to the stream related files and to access the Television listings API. The available directives are:

/

This directive proxies to the port 8000 at the same IP address as the nginx server. At this port is running the TV listings API, so when we open the address "<http://iptv.atnog.org/>" on any browser,

the API specification page will be rendered.

/IOPLAYER

This directive will redirect users to our custom player location. In this case, we are deploying the client on <http://ioplayer.herokuapp.com/>, using Heroku's platform as a service.

/STATIC

This directive maps to a folder that contains some static files like the "epg.xml" file containing the TV listings file originally generated, or the channels logos. For example, to request the epg file we must perform a get request to <http://iptv.atnog.org/static/epg.xml>.

/STAT

This directive shows the default "nginx_rtmp_module" RTMP statistics in real time. Those statistics are only related to the RTMP streams.

/BASIC__STATUS

This directive provides information about the number of active connections, number of waiting clients, number of clients writing information on the servers and other basic information provided by the NGINX.

/DASH

The directive to request DASH stream data. For instance, to request the channel named "test" we should perform an HTTP request to <http://iptv.atnog.org/dash/test.mpd>.

/HLS

The directive to request HLS stream data. For example, to request the channel named "test" we should perform a HTTP request to <http://iptv.atnog.org/hls/test.m3u8>.

In order to be able to play the video streams from web clients from different domains, we need to enable the Cross-Origin Resource Sharing (CORS). In addition, and since our channels represent a live event, we can't cache the requests of the playlist file. The problem around that question is that, on NGINX, when disabling the cache on the "/hls" path location, we are disabling the cache for both .m3u8 and .ts files. The logical solution was to verify, inside the location, the requested file extension and decide either to cache it or not. By reading the NGINX website, we noticed that there are inconsistencies about using IF statements inside path locations.[113] Another solution would be mounting a ramdisk to speed up the I/O process. The problem about this solution is that the fragment storage can vary up to 21 GB of data and the server only accepts a maximum of 8 GB of RAM.

4.2 CLIENT SIDE IMPLEMENTATION

This section is dedicated to the implementation of the client that will be responsible to reproduce the channel's streams, displaying the EPG and to create control over the data to display. Each subsection presented refers to each module present on the proposed architecture (see figure 3.4 from the section 3.4), respectively.

4.2.1 PLAYER MODULE

The player module is a React Component that provides control over:

1. **source**, to specify the remote address to request the stream data;
2. **play/pause**;
3. **fullscreen** mode;
4. **seek**, in seconds, to navigate in the video reproduction;
5. **mute/unmute** mode;
6. **volume** control;
7. **select_quality** mode, can force the desired quality, the default is auto;
8. **autofullscreen** mode, to start the channel in fullscreen by default;

This module also provides information about the available playback qualities. At the component creation, if a source is specified, an internal video player is created and rendered. Posteriorly, if any of the previous controls are changed by any entity the component will update only the required to perform the required action. The figure 4.13 shows the dependency diagram for this component. As we can verify, the internal video player relies on the hls.js video framework.

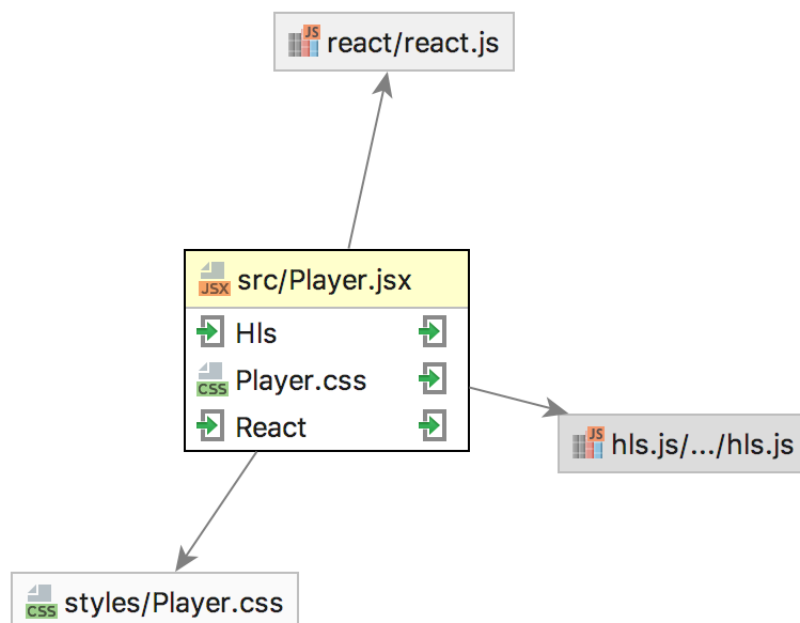


Figure 4.13: Player component diagram.

The Player.css is a CSS file that defines the video player styling. In our implementation we use it to fill the player to the page size, maintaining the default aspect ratio and orientation. Figure 4.14 shows the interface where the player is represented as background, by default.



Figure 4.14: Player running as background on the client application.

4.2.2 EPG MODULE

This module is responsible for requesting and displaying the TV listings for the requested player to the client. Was built as a React Component and enables users to select or restart some program that has already has been broadcasted previously. When the user interacts with that action, the module calculates the difference between the current time and the target time (start time of the requested program) and informs the parent that an action to seek the video was requested by the user. The figure 4.15 shows the dependencies of this component.

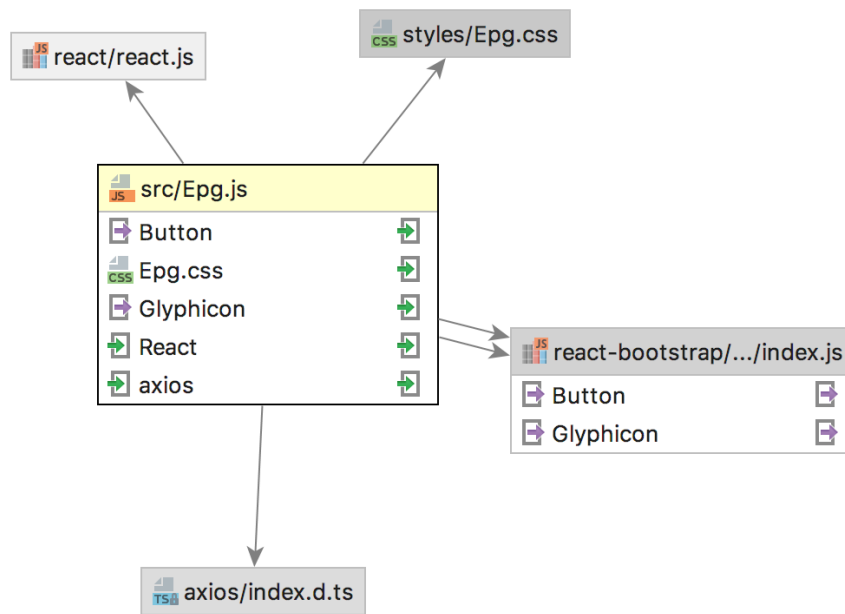


Figure 4.15: EPG component diagram.

The axios framework was used to request the JSON data to our TV listings API. It is possible to customise the aspect of this component’s interface, by tuning the Epg.css file. Figure 4.16 shows how it is possible to restart the program using the information provided by the implemented API.

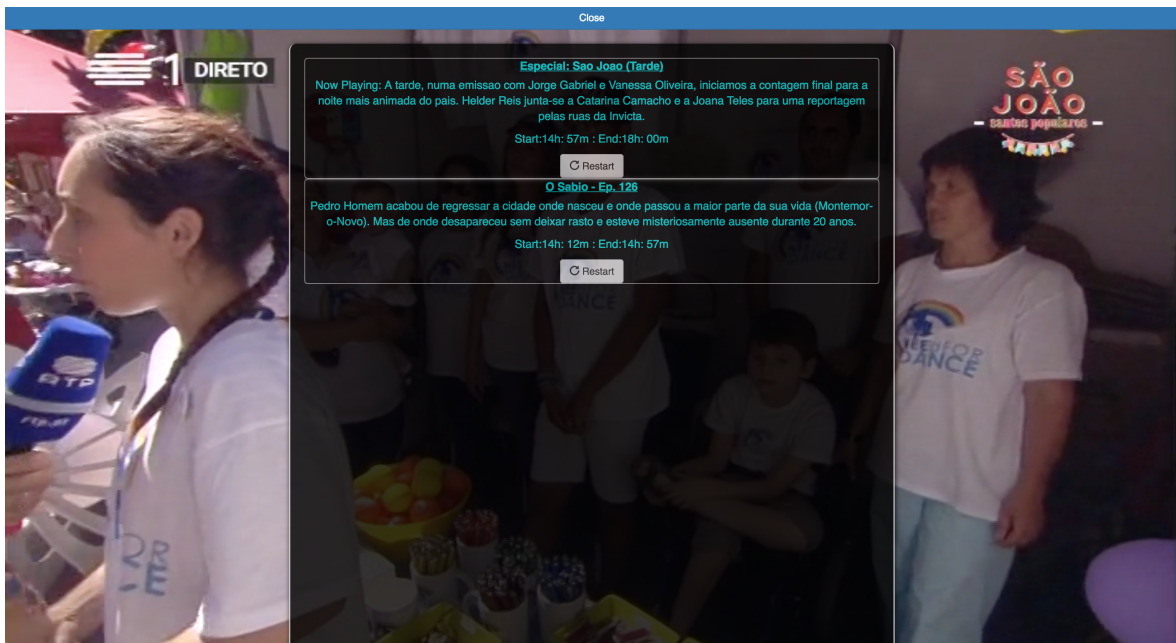


Figure 4.16: Restarting program accordingly to the EPG information.

4.2.3 HELP MODULE

The main purpose of this module is to inform users how to interact with the platform in terms of keyboard shortcuts and show other relevant information. This component's visual aspect can be also changed by tuning the Help.css file.

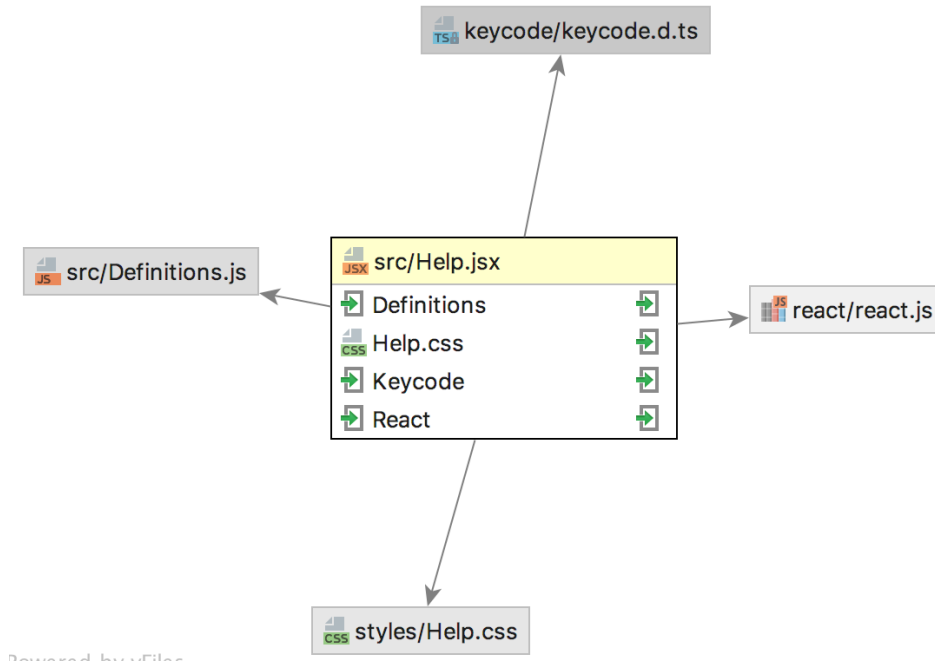


Figure 4.17: Help component diagram.

Figure 4.18 shows the default layout and information, of this module, presented to users. As we can see, the presented information is related to the keyboard shortcuts, that improve accessibility to users.

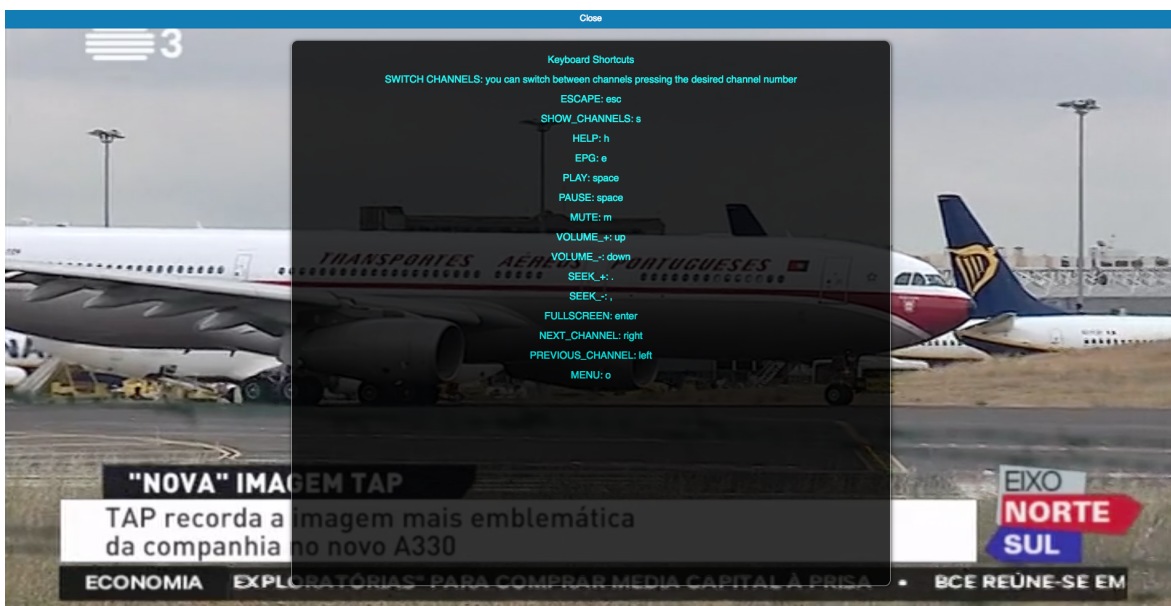


Figure 4.18: Help interface on client's platform.

4.2.4 DEFINITIONS MODULE

This module is a standard JavaScript file that contains the application's definitions and configurations. Any change in this file will trigger a change in the Components that depend on it. For instance, the Help module reads the Definitions module to display the information related to keyboard shortcuts therefore, changing the Definitions file will change this module's information and all the other modules that depend on it.

4.2.5 ANIMATIONS MODULE

This module is a JavaScript file, written with JSX syntax and its purpose is to present to users a visual feedback on their actions. It works by the following logic: In the module are defined all the recognised user actions. For each recognised action, an icon is associated. When an action is triggered, the module loads the correspondent module and checks if there is an ongoing animation. If an animation is being shown, the module updates the animation with the new one.

By default, the animation's component is visually hidden. When there is the need to show the animation, the component is shown and a timer is launched to hide the component after a defined time. Every time an update is required, the module checks if there is any timer launched. Any defined timer is canceled, and the component is shown with the new animation. After that, a new timer is created to hide the new animation.

Figure 4.19 shows, as an example, the animation presented when the increase volume key is pressed.



Figure 4.19: Increasing volume animation.

4.2.6 APPLICATION MODULE

This React Component is the central module of the platform. It is responsible for intermediating the communication between the components and to render the appropriate assets accordingly to the

user's iterations. This module is also responsible for reacting to the keyboard iterations, each iteration has a handle function associated that updates the required variables. The available handlers are:

- **setNewSource**, that receives the channel number and updates the video source;
- **handleHelp**, that shows or hides the Help module;
- **handleChannels**, that shows or hides the channel list view;
- **handleEpg**, that shows or hides the TV listings view;
- **handleSettings**, that shows or hides the settings view;
- **handleQualityLevels**, to display the levels that are available to the player;
- **changeVideoQuality**, to request to the player a quality change;
- **handleSeek**, to inform the player that it needs to seek to the requested time shift;

The figure 4.20 shows the dependency diagram for this module.

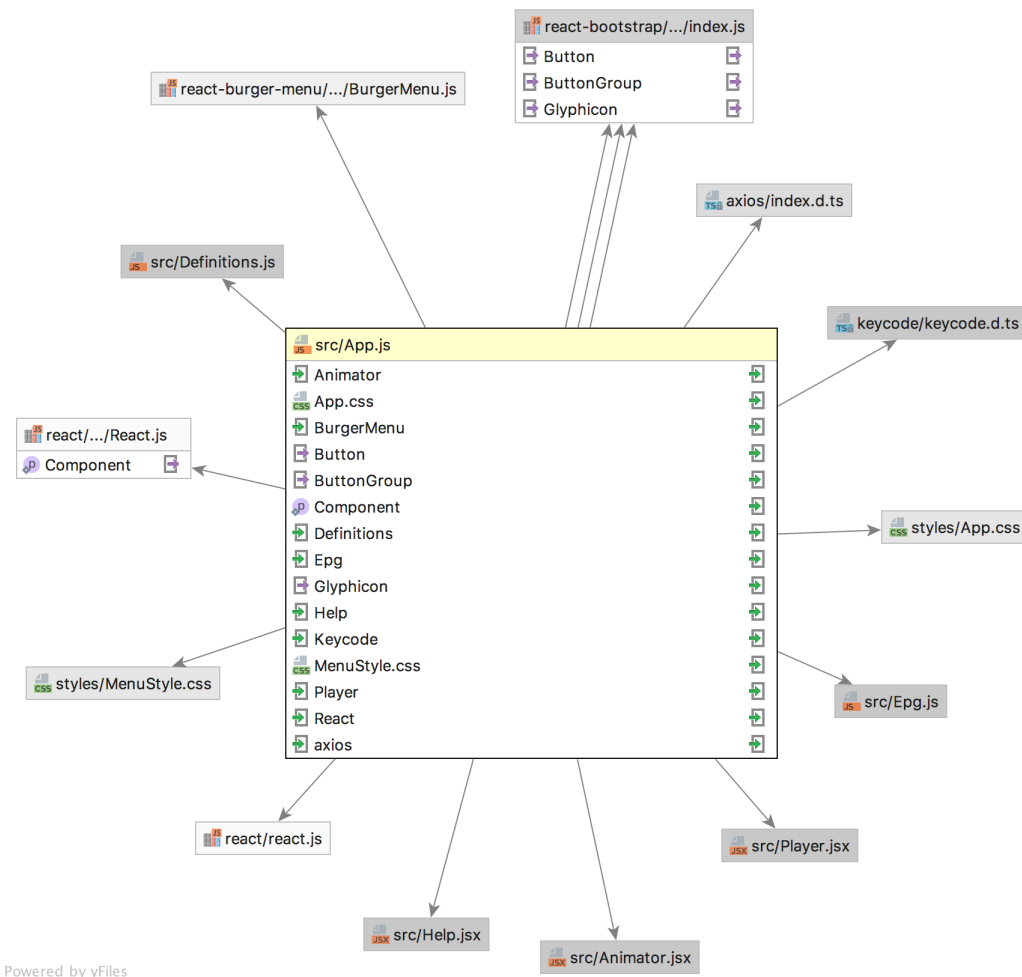


Figure 4.20: Application component diagram.

At figure 4.21 is represented the main menu interface with the available options to users. It is possible to choose the desired channel, view and interact with the programming guide, change the settings and check the help information.

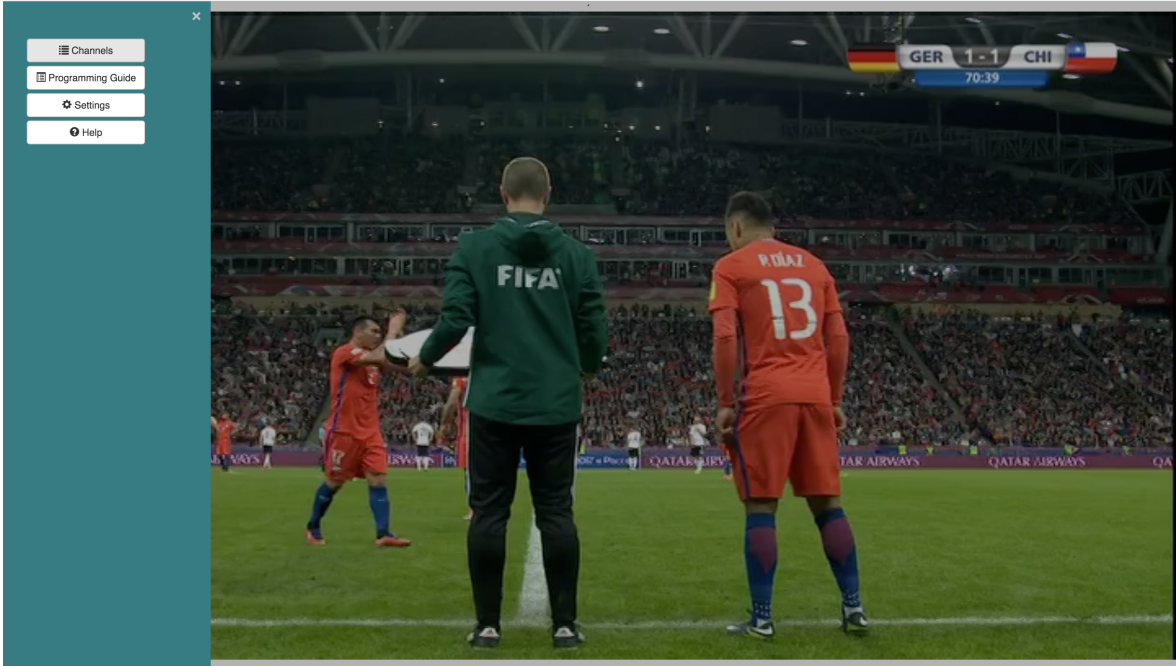


Figure 4.21: Main menu interface with the available options.

Figure 4.22 shows the menus for changing the application settings (at left) and to choose the desired channel (at right). As we can see, in the settings menu is possible to manually change the video quality and the menu side as well.

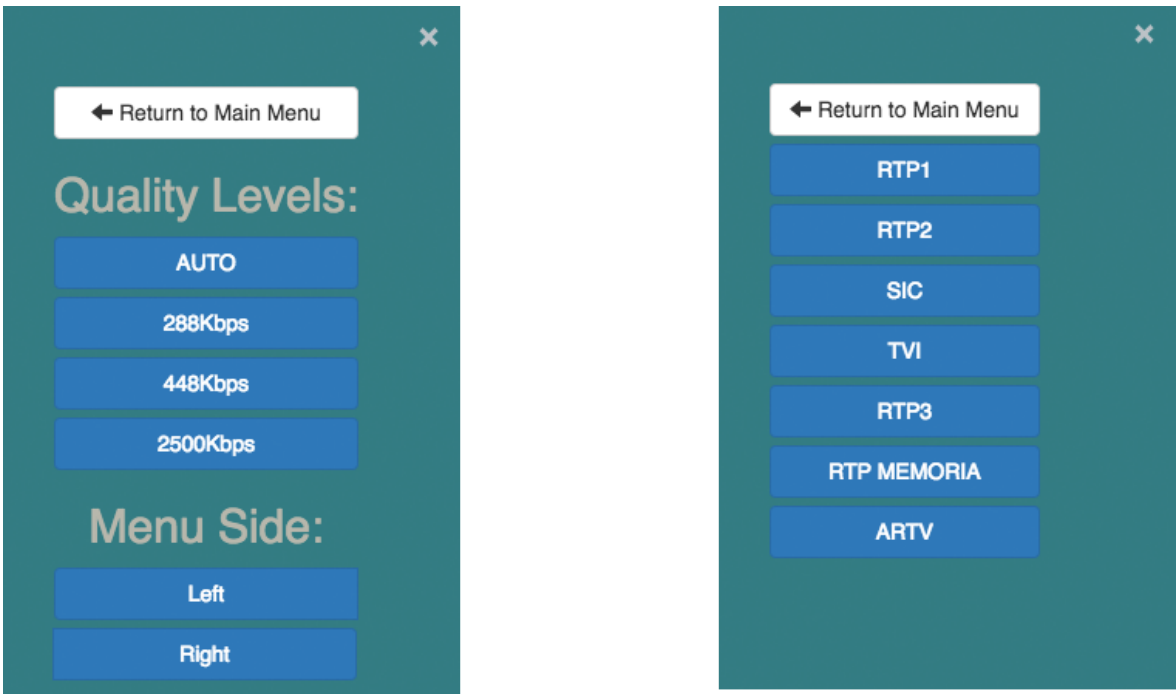


Figure 4.22: Layout of Settings menu (at left) and Channels menu (at right).

4.3 SETUP OVERVIEW

In figure 4.23 is possible to see the physical representation of the developed setup. At the right side of the image is located the notus server with the DVB-T dongle connected (marked at a)). In the center (marked at b)) is possible to see a screen showing the API description. Finally, at the left side (marked at c)) is present a computer running the client platform.

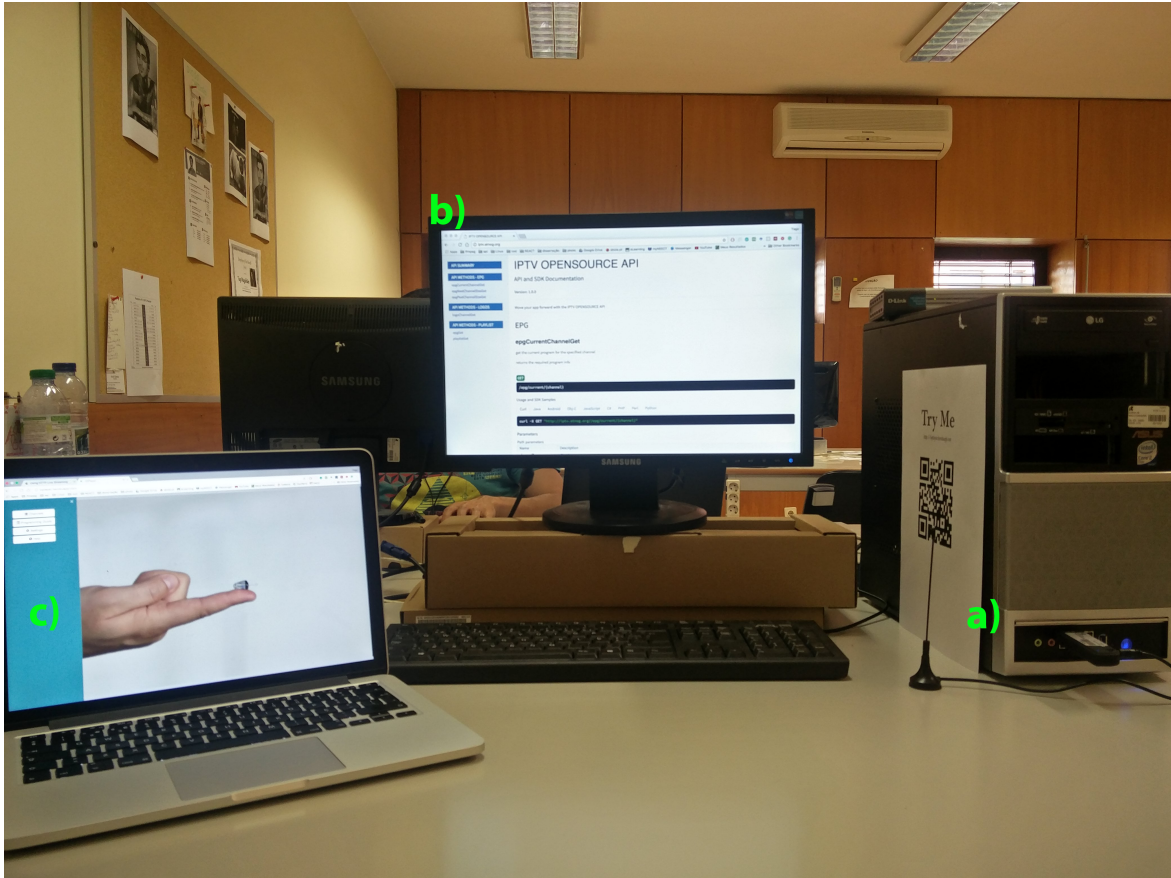


Figure 4.23: Setup with representation of the server, API specification and client player.

EVALUATION AND RESULTS

In a multimedia platform, features and performance are crucial to build quality software. Since part of this platform is focused on providing multimedia data to users, through the clients (multimedia players), is important to understand how many users this service can handle. On the other end, is also important to understand the behaviour of the client when "consuming" the provided multimedia, in different network conditions.

This chapter presents a description of the test challenges and performance of the implemented solution.

5.1 DVB

After implementing the architecture, we have tested the stability of the solution and detected that, after some time (it could be one hour or one week), the MuMuDVB showed some issues at the unscrambling process. The warning was "<channel name> is now partially unscrambled (<number>%" and after some time MuMuDVB was able to recover from the error. This problem caused instability in the transcoding process, resulting in a creation of a bad quality image with digital noise. Initially, and since the load on the transcoding machine was near to 100% Central Processing Unit (CPU) usage, we thought that the problem was in the transcoding process and decided to turn off some channels to test the effect (see section 5.2). After that, we discover that the problem was on the signal reception, therefore, to solve this issue, we connected the DVB-T dongle to the external antenna present on the *Instituto de Telecomunicações* building.

As we can see in figure 5.1, the mumudvb has a low impact on the CPU load with the medium value of 4,57%. It also has no impact on the memory. These values were extracted from the iptvs machine.

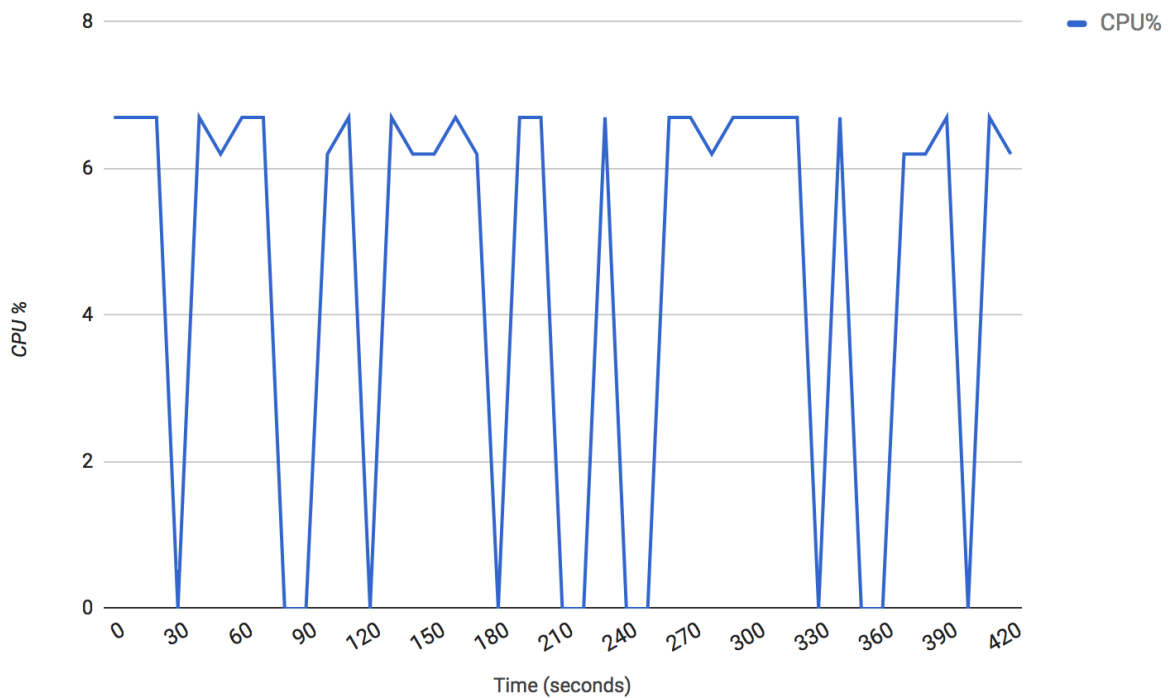


Figure 5.1: MuMuDVB impact on the CPU.

5.2 TRANSCODING

After implementing all the channels it is important to test the server impact of the implementation, in terms of processor loads and memory usage over the time. To test the impact, it was created a bash script that runs every 10 seconds and logs the CPU load and memory usage of the FFMPEG processes, to a Comma Separated Values (CSV) file.

In the implementation step, we only had one machine available to perform the transcoding process of the channels. While we were adding more channels we decided to monitor the CPU impact of these processes on the server. Figure 5.2 shows the obtained CPU load values when transcoding 4 or 5 channels. The average loads obtained for the 4 channel's test was 604% , whereas for the 5 channel's one, it was 795%, of CPU usage. Since the processor has 8 cores, the representation of the load goes from 0 to 800% (100% per core). By analysing the graphic it's possible to see that sometimes the load exceeds the upper limit of 800% which means that we are not able to transcode more than 4 channels concurrently, using our configurations. In order to be able to implement the transcoding of the 7 channels, it was added one more machine to our server architecture (iptv2). The first machine (iptvs) performs the transcoding of 3 channels and the second one transcodes the remaining 4.

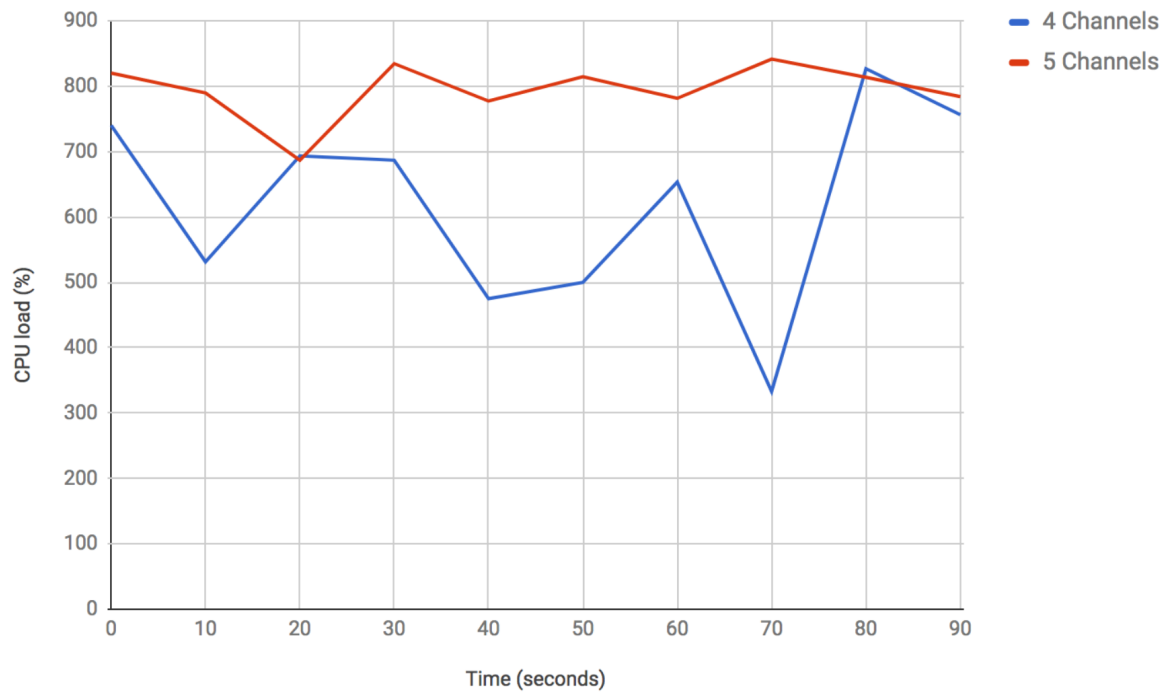


Figure 5.2: FFMPEG’s impact on iptvs server, in terms of CPU load, when transcoding 4 and 5 channels.

The FFMPEG’s impact, when transcoding 3 channels, is represented in the figure 5.3. The obtained load average was 504% and the memory usage obtained was constant with 17,7%.

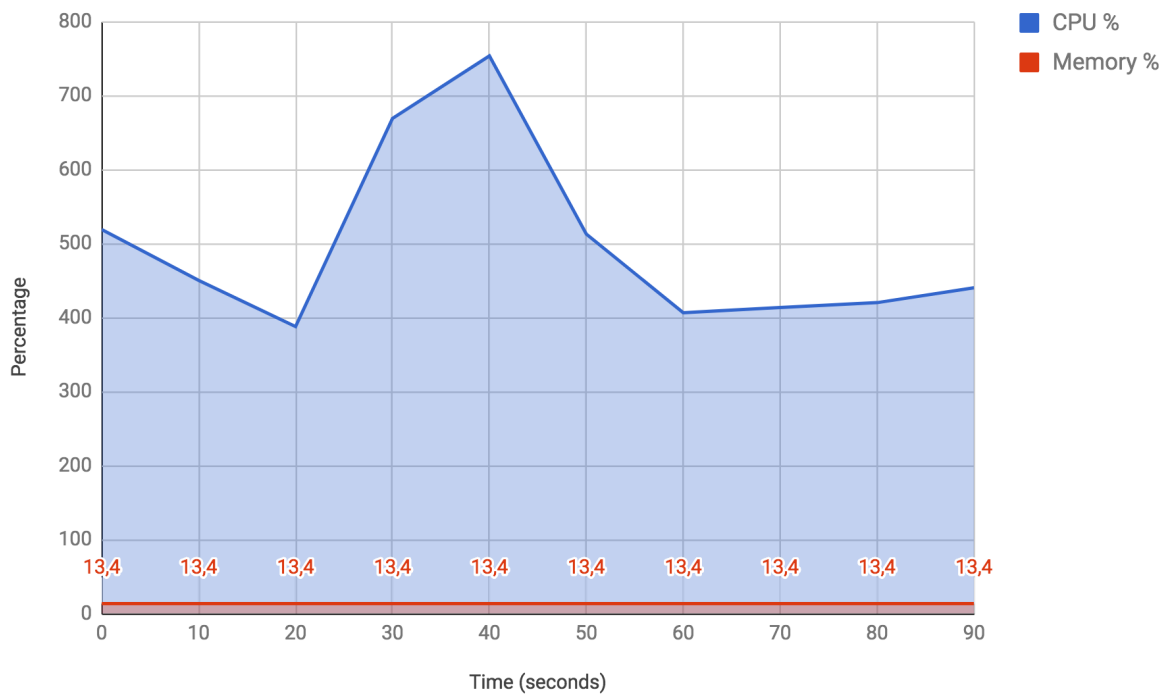


Figure 5.3: FFMPEG’s impact on CPU and memory, on the iptvs server.

The FFMPEG impact on the iptv2 machine, transcoding 4 channels, in terms of CPU and memory, is represented in the figure 5.4. The average load on the CPU was 529% and the memory usage was constant with 17,7% of usage.

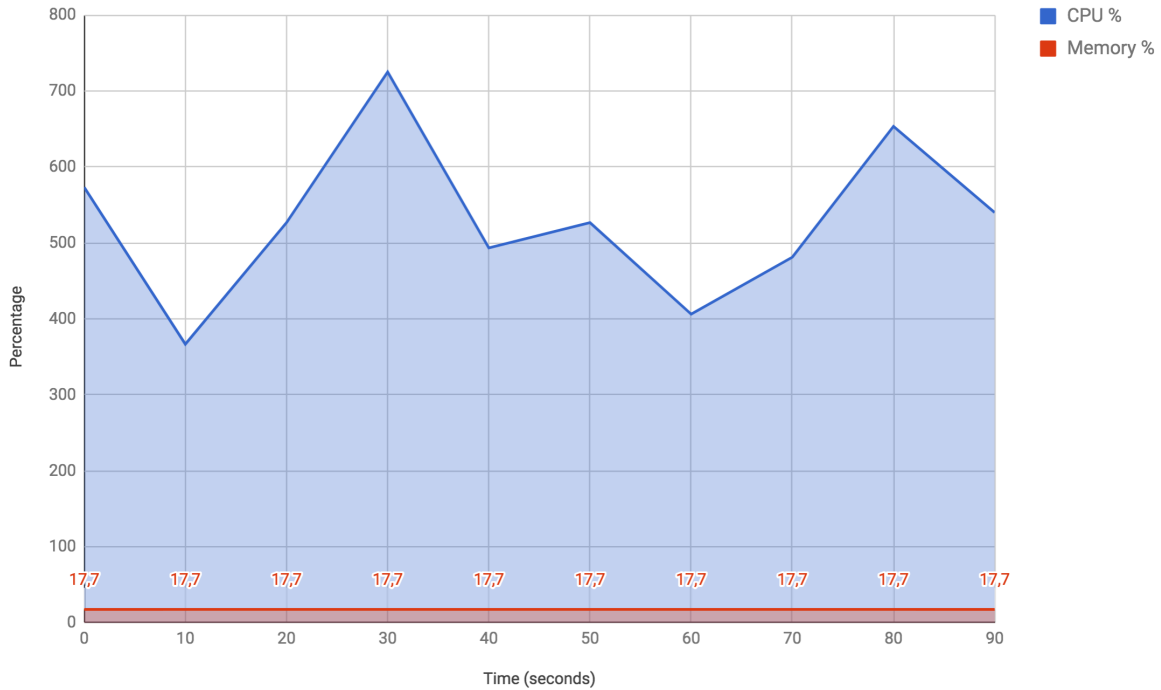


Figure 5.4: FFMPEG's impact on CPU and memory, on the iptv2 server.

To properly adjust the bitrate levels, in the metadata files that will instruct the players to adjust the proper encoding level accordingly to the available bandwidth, we need to test the bandwidth for the highest quality level, since it relies on the constant rate frame encoding that produces variable video bitrate. The other two quality levels have constant bitrates. The figure 5.5 shows the bitrate variation over the time. To obtain these values, we captured, using FFPROBE, the `packet_size` and `packet_duration_time` of each frame. Due to the fact that the highest quality level video has a constant framerate, the bitrate for each second was calculated by adding (for each of the captured parameters) the values of 25 consecutive frames. As we can state, there is, in this time window, a high variance in terms of bitrate, with a difference of 2596 Kbps between the minimum and maximum values. This variance causes the problem of know how to inform users what's the appropriate resolution. To solve this issue, we decided to test the bitrate without the constant frame rate parameter (see chapter 4.1.3).

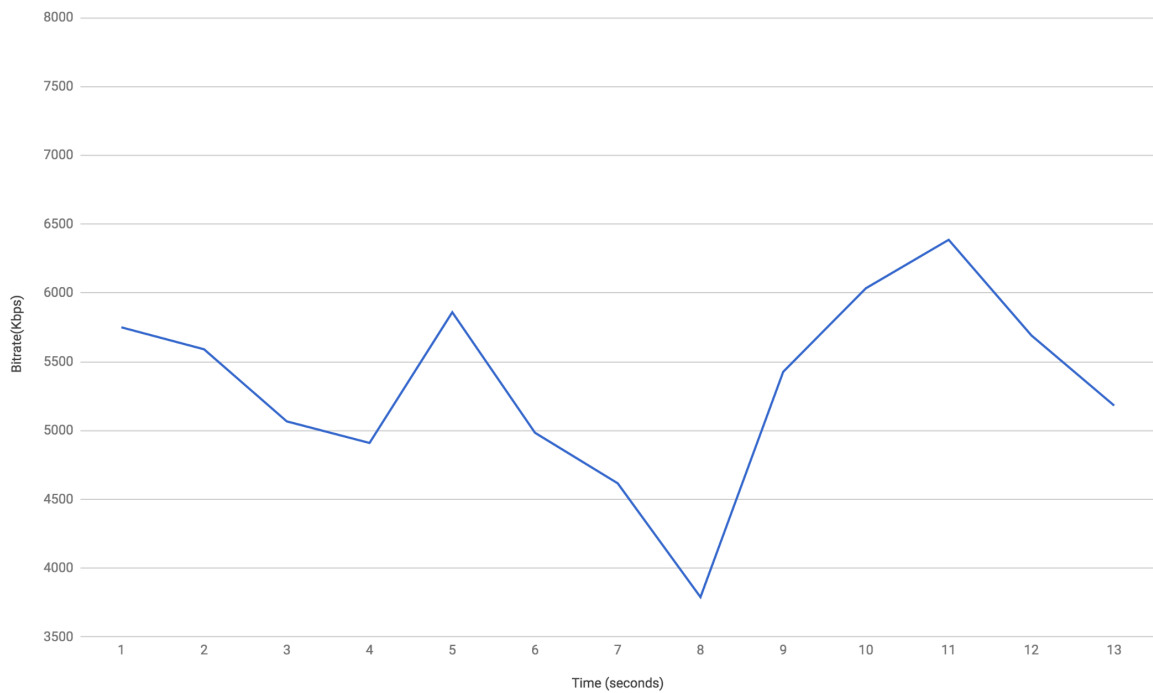


Figure 5.5: Bitrate variations over time with constant rate frame option.

The figure 5.6 shows the variance of the bitrate of the highest quality encoding level without the CRF flag. Comparing to the figure 5.5, this approach shows a minor bitrate variance, 1285 Kbps. It's also noticeable that the maximum and minimum bitrates values are between 2000 Kbps and 7000 Kbps. That means that we can transmit to users the same quality using less bandwidth.

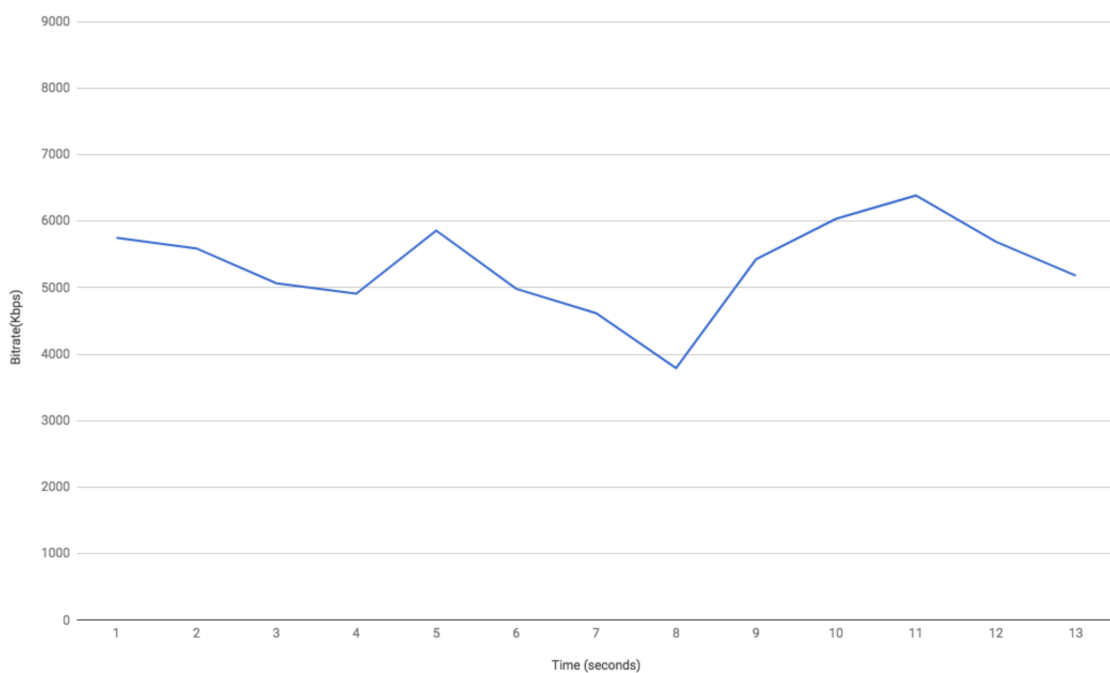


Figure 5.6: Bitrate variations over time without constant rate frame option.

For debugging the frame drops and duplicate packages, we ran each quality individually and at a final step, we monitored all the qualities at once, as the code 14 from the section 4.1.3 shows. For each case, we test 10 times the scenario, with a variable duration, and then calculate the average of the values gathered. To get these values from the FFMPEG we included the flag `FFREPORT=file=<file_name>:level=32` before the original command.

At the low quality, we are expecting high frame drops because we are reducing the streams frame rate from 25 to 12 FPS. The expected drop frame percentage is:

$$(12 - 25)/25 = 52\%$$

, where 12 is the target frame rate and 25 is the source frame rate. The figure 5.7 shows the percentage of dropped frames per each test. At the test number 9, it was obtained the lowest value with 48% but all the others were close to the expected value.

At high and medium qualities, since the frame rate is the same as the source FPS we do not have any frame drop or duplicated frame.

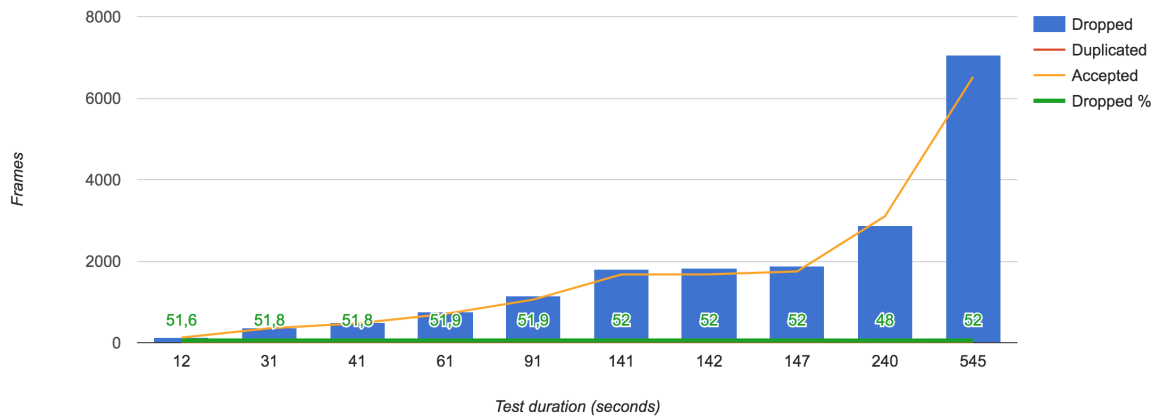


Figure 5.7: Low quality video statistics.

5.3 SERVER

The performance of the server depends on the machine performance. Our machine has two CPU cores and can handle up to 1024 connections. We also have to consider that, the server, even without clients requesting data, is constantly processing RTMP input streams.

In order to test the server performance the following parameters will be measured on the machine:

- CPU ;
- RAM usage;
- I/O ;
- Network load;

On the request results, it will also be considered:

- Number of concurrent users;

- Network throughput;
- Transaction rate;
- Response time;
- Latency;
- Error percentage;

In order to simulate clients viewing the streams without playback stops, we must guarantee that the response time is less than the duration of the downloaded media file. In our case, the response time should always be less than 10 seconds. There were followed the NGINX recommendations to meet the performance requirements.[114]

To test the server implementation, it was created a jmeter script that simulates the download of a HLS playlist. Then all the fragments related to one of the channels are downloaded. The purpose of this script is to perform load tests on the server. Since it's pretended to run a large set of concurrent users at once, we decided to use the flood platform ¹. The test server location is in London and is supplied by AWS.

To simulate a certain number of clients is difficult, firstly because it depends on how players buffer the data and, secondly, on the client's bandwidth condition and geolocation. In our case, we will test the worst case that still enables players to see the streams smoothly.

The script created, will perform requests on the highest quality video and try to download as much as possible from de oldest to the newest video fragment. By other words, this test simulates clients watching the better quality of video with no buffer limit.

The figure 5.8 shows the response times for 50 concurrent users. It is calculated every 15 seconds and presented the mean of the response time for each user.

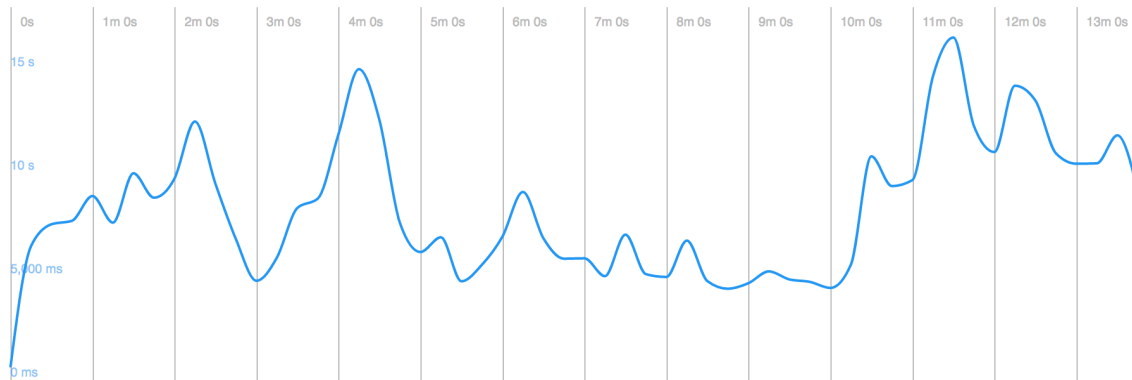


Figure 5.8: Response times for 50 concurrent users.

The highest response time was registered, at 11 minutes and 30 seconds, with 16 seconds. The average was 7,881 seconds. The figure 5.9 displays the latency variations over the test. The maximum latency registered was 4,024 seconds at 11 minutes and 30 seconds, the same moment that we got the highest response time. The average latency registered was 1,682 seconds.

¹www.flood.io

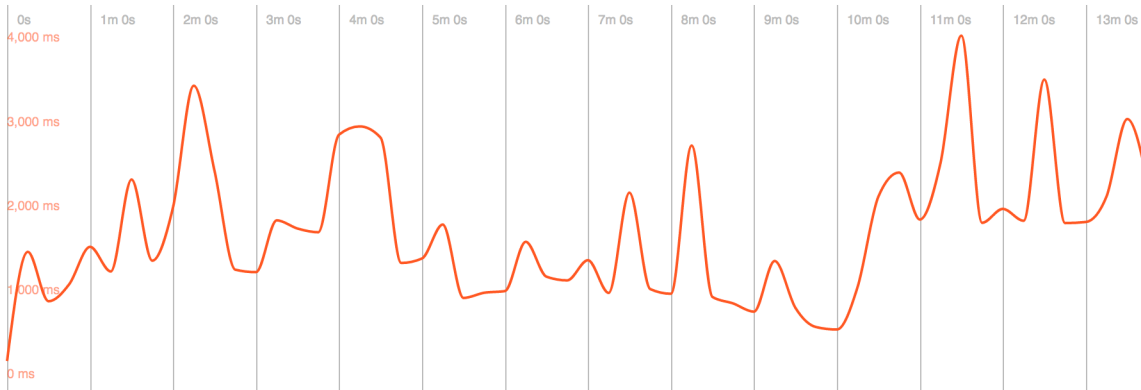


Figure 5.9: Request latency for 50 concurrent users.

In terms of network throughput, the variations go from 48 Mbps minimum to 130 Mbps maximum, as the figure 5.10 shows. The average was 98 Mbps.

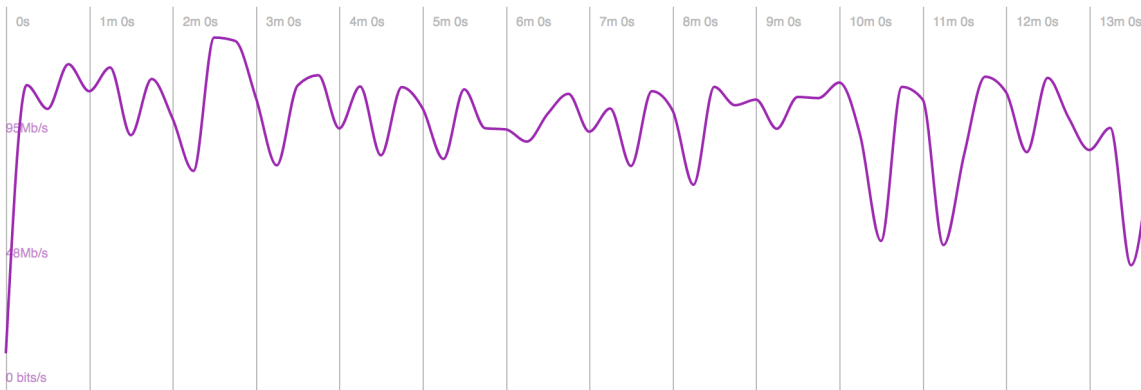


Figure 5.10: Network throughput for 50 concurrent users.

The error rate was of 0,03% with two transactions failed when trying to download video fragments. In terms of transactions, we obtained a mean of 437 requests per minute and the same value for the passed transactions.

When running the same test but for the medium quality stream, we got, for 100 concurrent users, a medium response time of 9,97 ms and a transaction rate of 654 requests per minute.

By analysing the server health during these tests, it was possible to notice that for handling the requests made by the clients, the CPU threads block waiting for the process of reading or writing to the disk. With a ramdisk and with a CPU with more threads we could handle responses quickly and therefore have better results. The figure 5.11 shows a snippet of the blocked processes during the test, with values captured every second.

```

Sun May 28 20:16:01 WEST 2017
tiago 3251 5.0 0.3 71592 14152 ? D May24 316:01 \_ nginx: worker process
tiago 3252 6.9 0.5 71844 23452 ? D May24 432:49 \_ nginx: worker process
Sun May 28 20:16:02 WEST 2017
tiago 3251 5.0 0.3 71592 14152 ? D May24 316:01 \_ nginx: worker process
tiago 3252 6.9 0.5 71844 23452 ? D May24 432:49 \_ nginx: worker process
Sun May 28 20:16:03 WEST 2017
tiago 3251 5.0 0.3 71592 14152 ? D May24 316:01 \_ nginx: worker process
tiago 3252 6.9 0.5 71844 23452 ? D May24 432:49 \_ nginx: worker process
Sun May 28 20:16:04 WEST 2017
tiago 3251 5.0 0.3 71592 14152 ? D May24 316:01 \_ nginx: worker process
tiago 3252 6.9 0.5 71844 23452 ? D May24 432:49 \_ nginx: worker process
Sun May 28 20:16:05 WEST 2017
tiago 3251 5.0 0.3 71592 14152 ? D May24 316:01 \_ nginx: worker process
tiago 3252 6.9 0.5 71844 23452 ? D May24 432:49 \_ nginx: worker process
Sun May 28 20:16:06 WEST 2017
tiago 3251 5.0 0.3 71592 14152 ? D May24 316:01 \_ nginx: worker process
tiago 3252 6.9 0.5 71844 23452 ? D May24 432:49 \_ nginx: worker process
Sun May 28 20:16:07 WEST 2017
tiago 3251 5.0 0.3 71592 14152 ? D May24 316:01 \_ nginx: worker process
tiago 3252 6.9 0.5 71844 23452 ? D May24 432:49 \_ nginx: worker process
Sun May 28 20:16:08 WEST 2017
tiago 3252 6.9 0.5 71844 23452 ? D May24 432:49 \_ nginx: worker process
Sun May 28 20:16:09 WEST 2017
Sun May 28 20:16:10 WEST 2017
tiago 3252 6.9 0.5 71844 23452 ? D May24 432:50 \_ nginx: worker process
Sun May 28 20:16:11 WEST 2017
tiago 3251 5.0 0.3 71592 14152 ? D May24 316:02 \_ nginx: worker process
Sun May 28 20:16:12 WEST 2017
tiago 3252 6.9 0.5 71844 23452 ? D May24 432:50 \_ nginx: worker process
Sun May 28 20:16:13 WEST 2017
tiago 3251 5.0 0.3 71592 14152 ? D May24 316:02 \_ nginx: worker process
tiago 3252 6.9 0.5 71844 23452 ? D May24 432:50 \_ nginx: worker process
Sun May 28 20:16:14 WEST 2017
tiago 3251 5.0 0.3 71592 14152 ? D May24 316:02 \_ nginx: worker process
tiago 3252 6.9 0.5 71844 23452 ? D May24 432:50 \_ nginx: worker process
Sun May 28 20:16:15 WEST 2017
tiago 3251 5.0 0.3 71592 14152 ? D May24 316:02 \_ nginx: worker process
tiago 3252 6.9 0.5 71844 23452 ? D May24 432:50 \_ nginx: worker process

```

Figure 5.11: A snippet of the threads in uninterruptible mode during the test.

In terms of CPU load and memory usage, the values are presented in figures 5.12 and 5.13, respectively. The CPU load reached 20% (out of 200, because of the two CPU threads) and the memory 1,2%. The low values on the CPU are possibly because of the threads block on read/write, as previously explained.

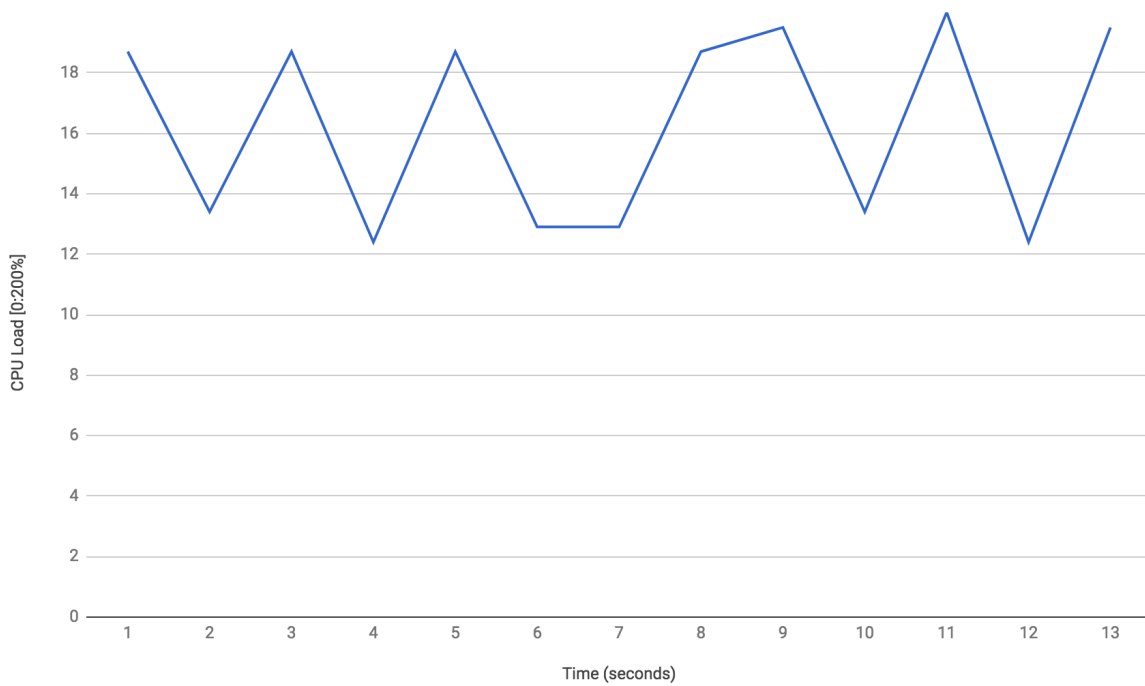


Figure 5.12: CPU load during the test made on the server.

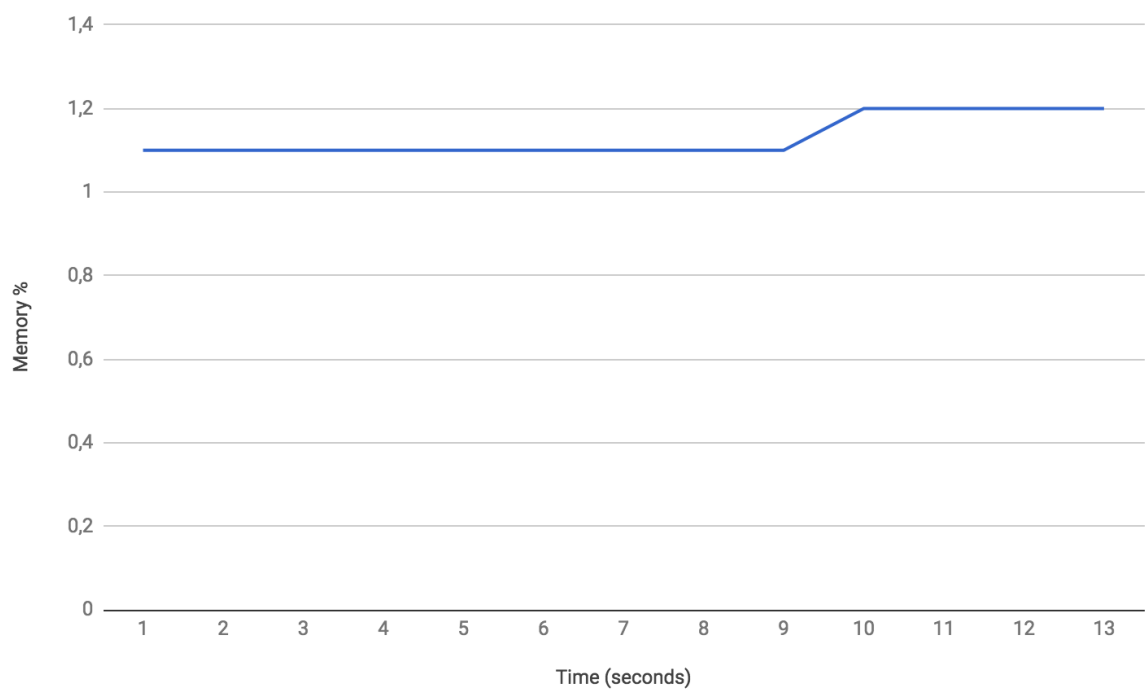


Figure 5.13: Memory usage during the test made on the server.

In terms of network load, the figure 5.14 shows the variations during the test, in megabits per second. As it's possible to see, at 9 minutes and 30 seconds (570 seconds) the test stopped and the network load reduced. The highest load value was 81,6 megabits per second, at the 460 seconds.

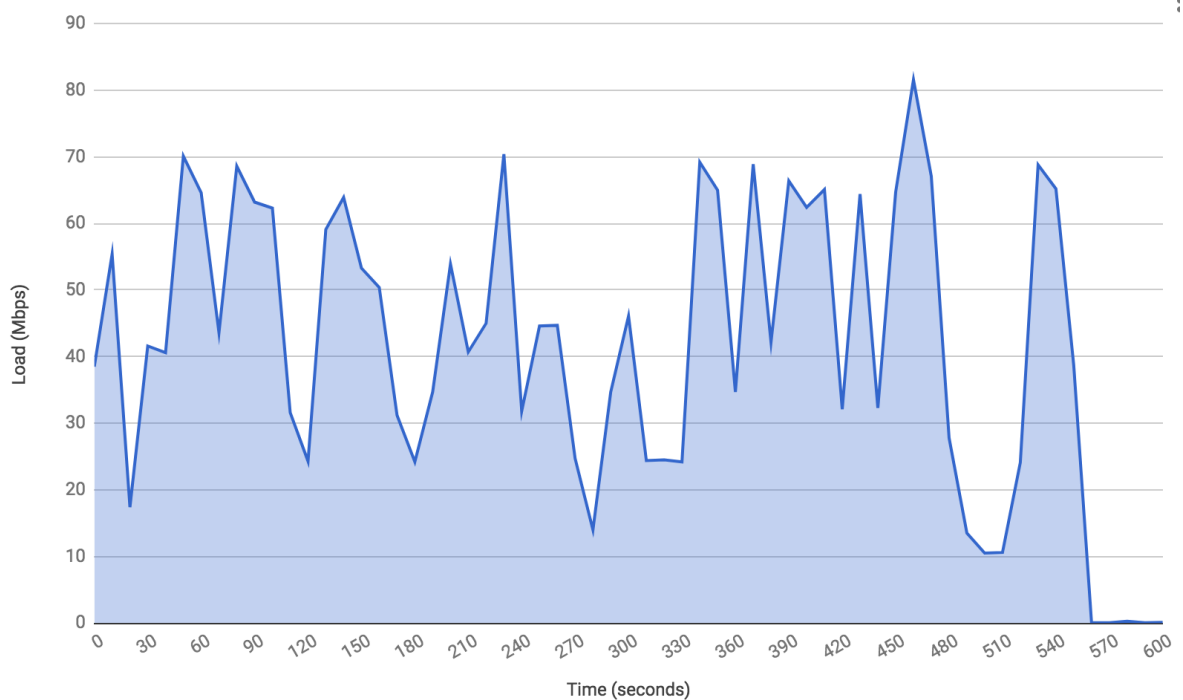


Figure 5.14: Network load variation during the test made on the server.

5.4 CLIENT

To test the performance of the client's adaptation to the network's bandwidth, we are using the Network Link Conditioner tool. With this tool is possible to strangle the available bandwidth, simulate frame drops and create profiles that simulate, for example, mobile networks.

The first test will simulate a mobile 3G network with 700 kbps and a 100 milliseconds (ms) delay. The player, with this configuration, adapts to the lowest quality level with no failure. We forced the player to manually set the medium bitrate but after 20 seconds the player stopped the reproduction. One interesting thing was that we were not able to force the player to try to download the highest quality, the reason behind that behaviour is the recommendation bandwidth being higher than the available bandwidth.

The other test on the network was the simulation of a very bad network. This network presents 1 Mbps, with 20% packet drops and a delay of 500ms on the requests. This network profile shows that we are still not able to request the highest profile bandwidth but the player tries to reduce to the lowest quality as possible to reproduce the streams. The reproduction was not fluid, with a lot of playback stops.

The other test made on the client's side was the delay of the reproduction when comparing to other solutions that provide the same multimedia to users. To test the delay we compared our stream of the RTP1 channel to the RTP Play's RTP1 channel. This analysis shows that in our implementation, it is possible to watch the contents in advance if we seek the video to the end of the video fragment once it downloads. This action is not possible in the RTP Play because we are unable to seek the video. When performing this action, it is possible to watch about 2/3 seconds before but, of course, it will

require a very good network connection to provide a low latency responses from the server. In that way, the buffer will never be empty. By default, if users just load the pages at the same time, and ignore the time of the advertisements on the RTP Play platform, it is possible to watch the contents about 3/4 seconds after on the compared platform. These variations depend on the fragment sizes, response delays, and on the load of the servers.

5.4.1 EPG

This subsection aims to analyse the performance of the TV listings API, in terms of response time. The figure 5.15 shows the response time variations during 10 consecutive requests to the address <http://iptv.atnog.org/current/rtp1>. As we can notice, the first request has the highest response time. Although the response time doesn't depend exclusively on the transfer time (it also depends on address lookup, connect time and others), the biggest difference was on it. The transfer time reduction occurred because of the cache mechanism applied on the server. After the first request, all the next ones are cached and so, there is no need to recalculate the response.

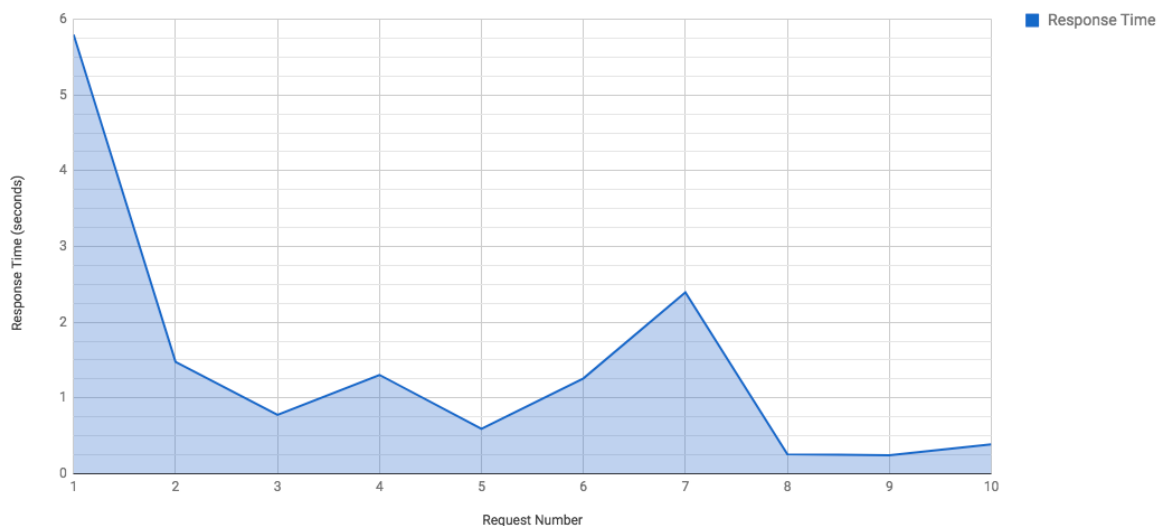


Figure 5.15: Response time variations when requesting the current program for the RTP1 channel.

Figure 5.16 shows the previous test applied to the address <http://iptv.atnog.org/next/tvi/10>. This location calculates the next 10 programs for the TVI channel. As we can see, the first request has, once again, the highest response time.

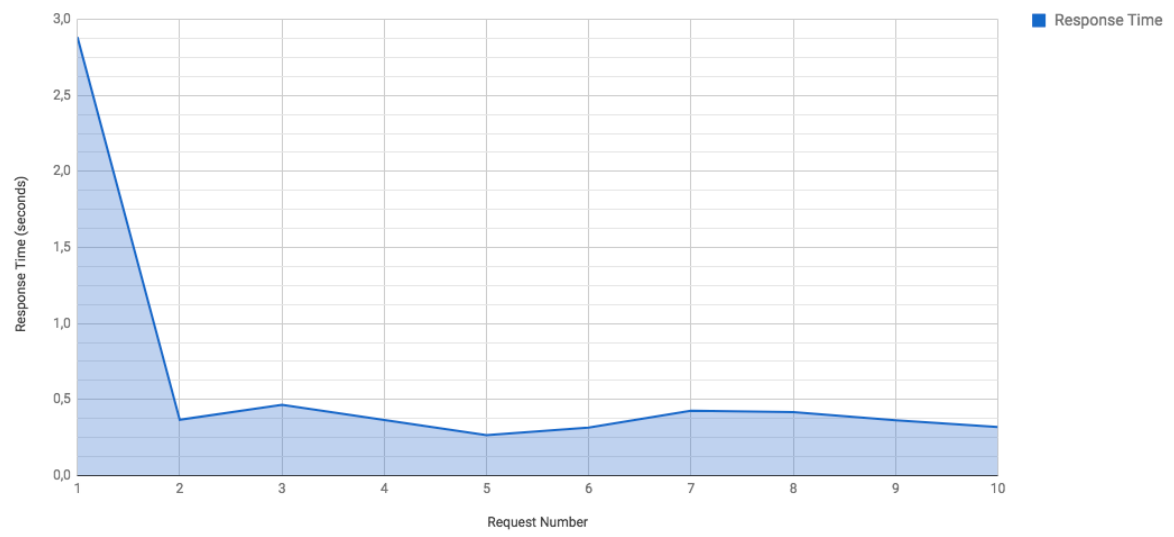


Figure 5.16: Response time variations when requesting the next 10 programs for the TVI channel.

CONCLUSIONS

The mobile devices market expansion created a large variety of vendors and models in the market. The smartness of the TVs and the popularity of the set-top-boxes also creates the possibility to create applications and serve streams compatible with them. This multitude of devices introduces a problem to developers, that is to target as many devices as possible in their applications.

The HTTP streaming is the state of the art in multimedia content distribution over the internet. These streams work over-the-top and consist in download and reproduce, small pieces of the content that user request, sequentially. The most known standard in this technique is the DASH protocol though some vendors have some limitations through which standard or specifications that can be followed. For that reason, it is also important to consider HLS and Microsoft Smooth streaming, from Apple and Microsoft respectively. Because of Apple's limitation to use their own specification and the number of targets of HLS being larger than DASH, it was decided to implement the Apple's solution on the server but providing an architecture also compatible with DASH or other specifications.

Our server structure was built to transcode and serve, in real time, an adaptive solution that could target as many devices as possible. Comparing to other platforms that use the same protocol to stream their multimedia, the HLS protocol, the results show that our implementation got close delay times. Moreover, our client architecture showed that is possible to seek back the reproduction according to the tv listings broadcasted in the digital television. The use of the ReactJS framework created the possibility to build a modular client structure with reusable and independent components.

6.1 FUTURE WORK

Even though the provided implementation is stable and fulfilled all proposed objectives, there is still a large room for improvements. These improvements would create a more interactive platform with more streaming solutions. The following list describes some of those possible improvements:

- Enable adaptive streaming using also DASH. The current solution is DASH capable but the module used doesn't recognise multiple bitrate representations;
- Reduce the I/O, either creating a cache system or keeping the media files in memory (requires large quantities of RAM);

- Implement a DASH player on the client side;
- Increase the actions and the quantity of information on the EPG module;

6.2 COLLABORATIONS

This work was done in collaboration with the *Departamento de Comunicação e Arte da Universidade de Aveiro (DECA)*, inserted in the project "+TV4E". This project consists, briefly, in an interactive platform for displaying relevant information (about public and social services), through the Television, for elderly people. Moreover, this platform should be running on Set-Top-Boxes connected to the Internet via mobile network (E.G. Long Term Evolution (LTE) connection). ¹

Since the first steps of development this contribution was important because it helped to understand the expected result of the implemented service, the strengths and weaknesses, stability of the solution over time and, mostly, the adaptive behaviour on different network conditions.

6.3 CONTRIBUTIONS

This section presents the contributions made during and related to the development of this dissertation. The first one, presented on section 6.3.1 refers to a Journal proposal and the second, presented on section 6.3.2, refers to a conference that will occur on July 18.

6.3.1 A CLOUD-BASED MOBILE VIDEO DELIVERY OVER HETEROGENEOUS NETWORKS

As part of this work, a proposal was made for the Journal of Computer Communications, with the work "A Cloud-based Mobile Video Delivery over Heterogeneous Networks". My contribution was to create an Android application and a headend component. The Android application detects and informs the network when a call is answered by the mobile user and the closest television identifier. As such, the device is constantly reading Bluetooth signals from the beacon devices attached to a Television, informing the network about the nearest one when a call is answered while viewing an online video stream. Figure 6.1 illustrates the application architecture.

When the application is running, a broadcast receiver is created to receive events: (i) from the Android system when a voice call is answered or ended (Figure 6.1-1); and (ii) from the Beacon Service (Figure 6.1-3). The Beacon Service is launched by the application main activity (named *NotifyOnCall*) (Figure 6.1-2) and it relies on the Android Beacon Library ² to read the Bluetooth signals that are sent by the beacons in range. The distance estimation between the mobile device and the TV is based on the received signal strength of each beacon device. When the broadcast receiver is signaled with an

¹<http://socialitv.web.ua.pt/index.php/projects/sponsored-projects/tv4e/>

²<https://altbeacon.github.io/android-beacon-library/>

actualisation, it updates his internal information and if the application is not running on background in that time, it updates the main activity (Figure 6.1-4).

When the broadcast receiver detects that the user answered an incoming call and if there is, at least, one TV in range, a trigger is sent to the network, identifying the nearest TV (Figure 6.1-5). When a mobile voice call is ended, the application triggers the network to retrieve the video back to the mobile device. Finally, while this application is running in the background, the user is visualising the online video stream in the VLC application.

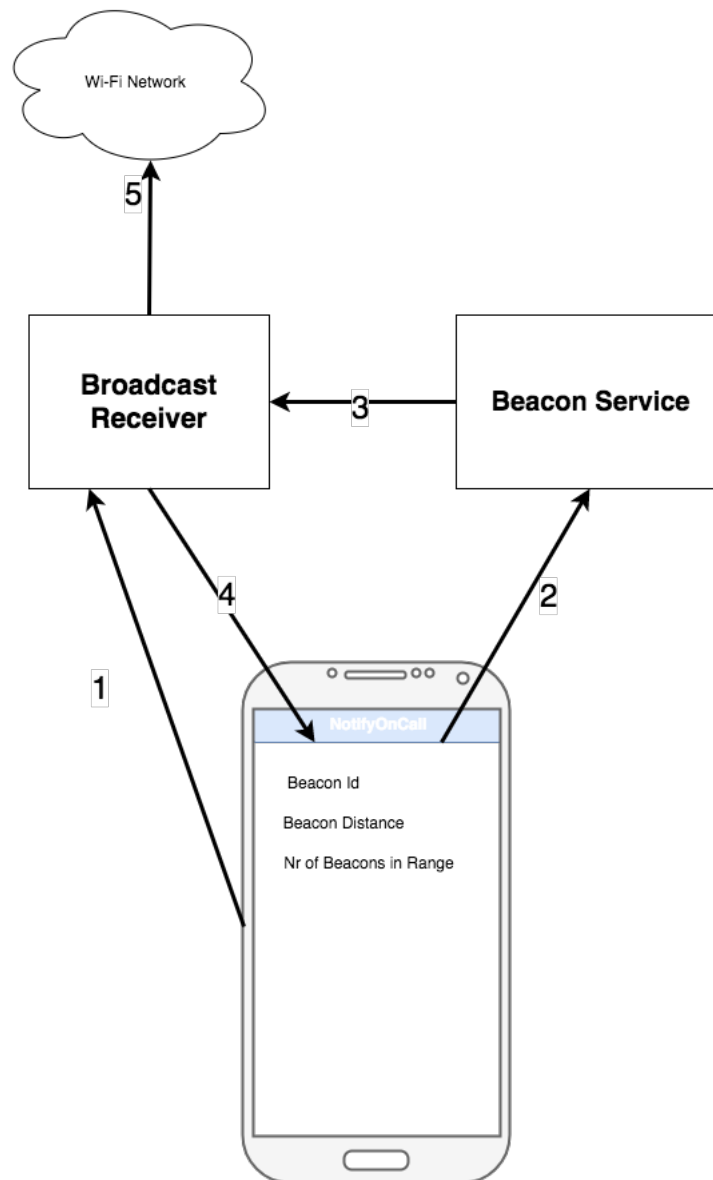


Figure 6.1: Mobile application architecture.

To deploy the headend, it was used the VLC application was used to stream and transcode a video, mimicking a video server, as shown in figure 6.2. As such, through the VLC capability of reading an input video and stream it to multiple destinations with different audio/video encodings, two video streams are available through the same IP address but on different ports. In this way, we can have, in

the same IP, the same video but in different resolutions, enabling to switch the video quality as we switch the destination device.

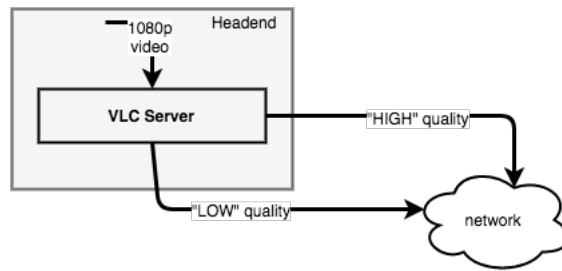


Figure 6.2: Headend architecture.

6.3.2 23RD SEMINAR OF THE MOBILE COMMUNICATIONS THEMATIC NETWORK

The 23rd seminar of the Mobile Communications Thematic Network (RTCM) will take place on the 18th of July at the Amphitheatre of the Instituto de Telecomunicações, in Aveiro. The work developed on this dissertation was selected to be presented, as part of multimedia services theme, containing a presentation of the work developed and a small demonstration of the platform and his functionalities.

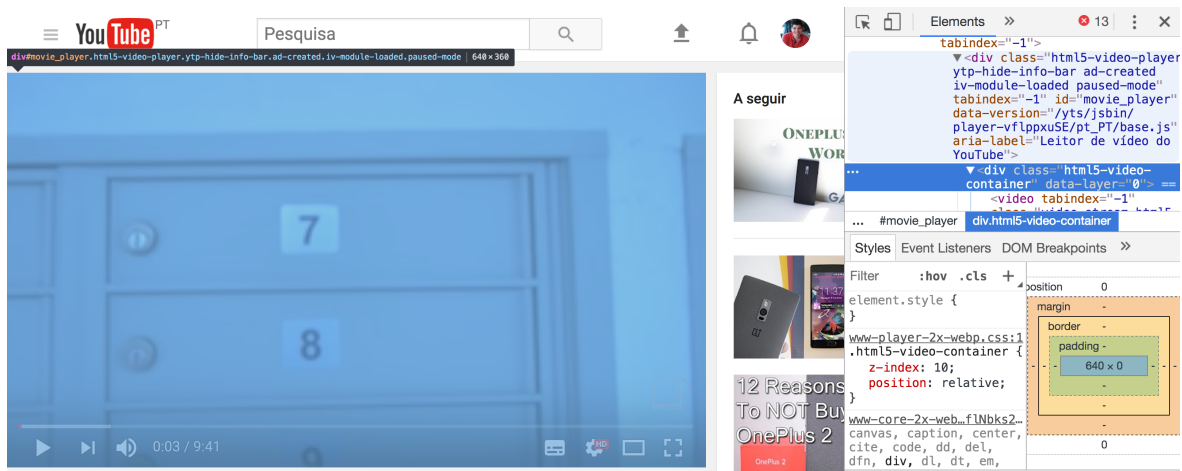
APPENDIX A

Youtube's page snippet showing the player elements information, on Safari for macOS:

The screenshot shows a YouTube video player on Safari for macOS. The video is titled "OnePlus 2 | 2 years later" and has 14,702 views. The player is currently at 0:03 / 9:41. The browser's developer tools are open, showing the HTML structure of the player. The selected element is a `<div class="html5-video-player ytp-hide-info-bar ad-created iv-module-loaded paused-mode" tabindex="-1" id="movie_player" data-version="yts/jsbin/player">`. The developer tools also show the `ytplayer` object and the `ytplayer.config` object.

```
<div id="player-unavailable" class="hid player-width player-height player-unavailable"></div>
<div id="player-api" class="player-width player-height off-screen-target player-api" tabindex="-1">
  <div class="html5-video-player ytp-hide-info-bar ad-created iv-module-loaded paused-mode" tabindex="-1" id="movie_player" data-version="yts/jsbin/player">
    <script>if (window.ytcsi) {window.ytcsi.tick("cfg", null, '');}</script>
    <script>
      var ytplayer = ytplayer || {};ytplayer.config = {"attrs":{"id":"movie_player"},"min_version":"8.0.0","url":"https://s.ytimg.com/yts/swfbin/player-vfSBFF6x8oA","fflags":"html5_background_quality_cap=360\u0026dynamic_ad_break_pause_threshold_sec=0\u0026html5_max_vss_watchtime_ratio=0.0\u0026signed_out_vf1Q8jxoV/cps.swf","sts":17277,"url_v9as2":"","assets":{"css":"\u0026cssbin/player-vf1AZ48hd/www-player-2x.css","js":"\u0026jsbin/player-vf1ppxuSEV"}
    </script>
    <div id="watch-queue-mole" class="video-mole mole-collapsed hid" style="display: none;"></div>
    <div id="player-playlist" class="content-alignment watch-player-playlist"></div>
  </div>
</div class="clear"></div>
```

Youtube's page snippet showing the player elements information, on Chrome for macOS:



APPENDIX B

```
tiago@aquilon:~$ sudo dvbsnoop -n 1 16
dvbsnoop V1.4.50 -- http://dvbsnoop.sourceforge.net/
```

```
-----
SECT-Packet: 00000001  PID: 16 (0x0010), Length: 114 (0x0072)
Time received: Mon 2017-05-15 21:00:37.227
-----
```

```
0000: 40 f0 6f 34 01 c9 00 00 f0 1a 40 0f 15 54 44 54 @.o4.....@..TDT
0010: 20 43 6f 6e 74 69 6e 65 6e 74 65 4a 07 04 4d 22  ContienteJ..M"
0020: c8 00 00 04 f0 48 04 4d 22 c8 f0 42 41 15 04 4d  ....H.M"..BA..M
0030: 16 04 4e 16 04 4f 16 04 50 16 04 53 16 04 54 16  ..N..O..P..S..T.
0040: 04 55 16 5a 0b 04 7e 83 40 1f 81 1a ff ff ff ff  .U.Z..~.@.....
0050: 83 1c 04 4d c0 01 04 4e c0 02 04 4f c0 03 04 50  ...M...N...O...P
0060: c0 04 04 53 c0 05 04 54 c0 06 04 55 c0 07 8a bf  ...S...T...U....
0070: d6 19 ..
```

```
PID: 16 (0x0010) [= assigned for: DVB Network Information Table (NIT), Stuffing Table (ST)]
```

```
Guess table from table id...
```

```
NIT-decoding...
```

```
Table_ID: 64 (0x40) [= Network Information Table (NIT) - actual network]
```

```
section_syntax_indicator: 1 (0x01)
```

```
reserved_1: 1 (0x01)
```

```
reserved_2: 3 (0x03)
```

```
Section_length: 111 (0x006f)
```

```
Network_ID: 13313 (0x3401) [= --> please lookup at http://www.dvb.org]
```

```
reserved_3: 3 (0x03)
```

```
Version_number: 4 (0x04)
```

```
current_next_indicator: 1 (0x01) [= valid now]
```

```
Section_number: 0 (0x00)
```

```
Last_Section_number: 0 (0x00)
```

```
reserved_4: 15 (0x0f)
```

```
Network_descriptor_length: 26 (0x001a)
```

```
    DVB-DescriptorTag: 64 (0x40) [= network_name_descriptor]
```

```
    descriptor_length: 15 (0x0f)
```

```
    Network_name: "TDT Contiente" -- Charset: reserved
```

```
    DVB-DescriptorTag: 74 (0x4a) [= linkage_descriptor]
```

```
    descriptor_length: 7 (0x07)
```

```
    transport_stream_ID: 1101 (0x044d)
```

```
    original_network_ID: 8904 (0x22c8) [= >>ERROR: not (yet) defined... Report!<<]
```

```
    service_ID: 0 (0x0000) [= --> refers to PMT program_number]
```

```
    linkage_type: 4 (0x04) [= TS containing complete Network/Bouquet SI]
```

```

reserved_5: 15 (0x0f)
Transport_stream_loop_length: 72 (0x0048)

Transport_stream_ID: 1101 (0x044d)
Original_network_ID: 8904 (0x22c8) [= >>ERROR: not (yet) defined... Report!<<]
reserved_1: 15 (0x0f)
Transport_descriptor_length: 66 (0x0042)

DVB-DescriptorTag: 65 (0x41) [= service_list_descriptor]
descriptor_length: 21 (0x15)
  service_ID: 1101 (0x044d)[ --> refers to PMT program_number]
  service_type: 22 (0x16) [= advanced codec SD digital television service]

  service_ID: 1102 (0x044e)[ --> refers to PMT program_number]
  service_type: 22 (0x16) [= advanced codec SD digital television service]

  service_ID: 1103 (0x044f)[ --> refers to PMT program_number]
  service_type: 22 (0x16) [= advanced codec SD digital television service]

  service_ID: 1104 (0x0450)[ --> refers to PMT program_number]
  service_type: 22 (0x16) [= advanced codec SD digital television service]

  service_ID: 1107 (0x0453)[ --> refers to PMT program_number]
  service_type: 22 (0x16) [= advanced codec SD digital television service]

  service_ID: 1108 (0x0454)[ --> refers to PMT program_number]
  service_type: 22 (0x16) [= advanced codec SD digital television service]

  service_ID: 1109 (0x0455)[ --> refers to PMT program_number]
  service_type: 22 (0x16) [= advanced codec SD digital television service]

DVB-DescriptorTag: 90 (0x5a) [= terrestrial_delivery_system_descriptor]
descriptor_length: 11 (0x0b)
Center frequency: 0x047e8340 (= 754000.000 kHz)
Bandwidth: 0 (0x00) [= 8 MHz]
priority: 1 (0x01) [= HP (high priority) or Non-hierarchy.]
Time_Slicing_indicator: 1 (0x01) [= Time Slicing is not used.]
MPE-FEC_indicator: 1 (0x01) [= MPE-FEC is not used.]
reserved_1: 3 (0x03)
Constellation: 2 (0x02) [= 64-QAM]
Hierarchy_information: 0 (0x00) [= non-hierarchical (native interleaver)]
Code_rate_HP_stream: 1 (0x01) [= 2/3]
Code_rate_LP_stream: 0 (0x00) [= 1/2]
Guard_interval: 3 (0x03) [= 1/4]
Transmission_mode: 1 (0x01) [= 8k mode]
Other_frequency_flag: 0 (0x00)
reserved_2: 4294967295 (0xffffffff)

DVB-DescriptorTag: 131 (0x83) [= User defined/ATSC reserved]
descriptor_length: 28 (0x1c)
Descriptor-data:
  0000: 04 4d c0 01 04 4e c0 02 04 4f c0 03 04 50 c0 04 .M...N...O...P..
  0010: 04 53 c0 05 04 54 c0 06 04 55 c0 07 .S...T...U..

CRC: 2327827993 (0x8abfd619)
=====

```

REFERENCES

- [1] Xiaojun Hei, Chao Liang, Jian Liang, Yong Liu, and K. Ross, “A Measurement Study of a Large-Scale P2P IPTV System”, *Ieee trans. multimed.*, vol. 9, no. 8, pp. 1672–1687, Dec. 2007, ISSN: 1520-9210. DOI: 10.1109/TMM.2007.907451. [Online]. Available: <http://ieeexplore.ieee.org/document/4378423/>.
- [2] Nielsen, *Total Audience Report: Q4 2016*. [Online]. Available: <http://www.nielsen.com/us/en/insights/reports/2017/the-nielsen-total-audience-report-q4-2016.html> (visited on 06/07/2017).
- [3] MarketsandMarkets, *Over The Top (OTT) Market by Content Type Platform - 2020*. [Online]. Available: <http://www.marketsandmarkets.com/Market-Reports/over-the-top-ott-market-41276741.html> (visited on 06/07/2017).
- [4] Reportlinker, *Over the Top - Global Market Outlook (2016-2022)*. [Online]. Available: <http://www.prnewswire.com/news-releases/over-the-top---global-market-outlook-2016-2022-300374156.html> (visited on 06/07/2017).
- [5] Telegraph, *YouTube and Netflix boost mobile data traffic by 65pc - Telegraph*. [Online]. Available: <http://www.telegraph.co.uk/technology/news/11999378/YouTube-and-Netflix-boosts-mobile-data-traffic-by-65pc.html> (visited on 06/07/2017).
- [6] Diffen, *Hulu vs Netflix Comparison - 6 Differences | Diffen*. [Online]. Available: http://www.diffen.com/difference/Hulu_vs_Netflix (visited on 06/22/2017).
- [7] Sandvine, *Global Internet Phenomena*. [Online]. Available: <https://www.sandvine.com/trends/global-internet-phenomena/> (visited on 06/07/2017).
- [8] Anacom, *Transição para a TDT*, 2014. [Online]. Available: <https://www.anacom.pt/render.jsp?categoryId=350823>.
- [9] G. Philpott and A. Kattukaran, *Evolution of TV: Reaching Audiences Across Screens*. [Online]. Available: <https://www.thinkwithgoogle.com/marketing-resources/evolution-of-tv-reaching-audiences-across-screens/> (visited on 06/03/2017).
- [10] *Digital Video Broadcasting - LinuxTVWiki*. [Online]. Available: https://www.linuxtv.org/wiki/index.php/Digital_Video_Broadcasting (visited on 05/03/2017).
- [11] “Digital video broadcasting”, 2001. [Online]. Available: <https://www.google.com/patents/US7224837>.
- [12] *DVB-T2 - LinuxTVWiki*. [Online]. Available: <https://www.linuxtv.org/wiki/index.php/DVB-T2> (visited on 05/03/2017).
- [13] L. Vangelista, N. Benvenuto, S. Tomasin, C. Nokes, J. Stott, A. Filippi, M. Vlot, V. Mignone, and A. Morello, “Key technologies for next-generation terrestrial digital television standard DVB-T2”, *Ieee commun. mag.*, vol. 47, no. 10, pp. 146–153, Oct. 2009, ISSN: 0163-6804.

- DOI: 10.1109/MCOM.2009.5273822. [Online]. Available: <http://ieeexplore.ieee.org/document/5273822/>.
- [14] *Standards - DVB*. [Online]. Available: <https://www.dvb.org/standards> (visited on 05/03/2017).
- [15] *ANACOM - Introdução em Portugal da Televisão Digital Terrestre (DVB-T)*. [Online]. Available: <https://www.anacom.pt/render.jsp?categoryId=342919> (visited on 05/03/2017).
- [16] *ANACOM - Esclarecimento sobre Televisão Digital Terrestre*. [Online]. Available: <https://www.anacom.pt/render.jsp?contentId=754058> (visited on 05/03/2017).
- [17] *DTV Status - Digital Terrestrial Television (DTT)*. [Online]. Available: <http://en.dtvstatus.net/> (visited on 05/03/2017).
- [18] 3CX, *What is RTP (Real-time Transport Protocol)?* [Online]. Available: <https://www.3cx.com/pbx/rtp/> (visited on 06/22/2017).
- [19] P. Zhao, J. Li, J. Xi, and X. Gou, "A Mobile Real-Time Video System Using RTMP", in *2012 fourth int. conf. comput. intell. commun. networks*, IEEE, Nov. 2012, pp. 61–64, ISBN: 978-0-7695-4850-0. DOI: 10.1109/CICN.2012.18. [Online]. Available: <http://ieeexplore.ieee.org/document/6375070/>.
- [20] 3CX, *What is RTCP (Real Time Control Transport Protocol)?* [Online]. Available: <https://www.3cx.com/pbx/rtcp/> (visited on 06/22/2017).
- [21] A. Durresi and R. Jain, "RTP, RTCP, and RTSP - Internet Protocols for Real-Time Multimedia Communication", in *Ind. inf. technol.* CRC Press, 2005, ch. 28. [Online]. Available: <http://www.cse.wustl.edu/~jain/books/ftp/rtp.pdf>.
- [22] Adobe, *Real-Time Messaging Protocol (RTMP) specification | Adobe Developer Connection*. [Online]. Available: <http://www.adobe.com/devnet/rtmp.html> (visited on 06/22/2017).
- [23] GlobalDots, *RTMP - Real Time Messaging Protocol Explained*. [Online]. Available: <http://www.globaldots.com/rtmp-real-time-messaging-protocol-explained-2/> (visited on 06/22/2017).
- [24] Cisco Networks, "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021", Tech. Rep., 2017, p. 35. [Online]. Available: <http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.pdf>.
- [25] G. Pallis and A. Vakali, "Insight and Perspectives for CONTENT DELIVERY NETWORKS", *Commun. acm*, vol. 49, no. 1, 2006. [Online]. Available: dl.acm.org/citation.cfm?id=1107462.
- [26] J. Kua, G. Armitage, and P. Branch, "A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming over HTTP", *Ieee commun. surv. tutorials*, pp. 1–1, 2017, ISSN: 1553-877X. DOI: 10.1109/COMST.2017.2685630. [Online]. Available: <http://ieeexplore.ieee.org/document/7884970/>.
- [27] S. D. Vishnubhotla Venkata Krishna, "Cost Aware Virtual Content Delivery Network for Streaming Multimedia : Cloud Based Design and Performance Analysis", 2015. [Online]. Available: <http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A861965&dswid=-2397>.
- [28] T. Stockhammer and Thomas, "Dynamic adaptive streaming over HTTP –", in *Proc. second annu. acm conf. multimed. syst. - mmsys '11*, New York, New York, USA: ACM Press, 2011, p. 133, ISBN: 9781450305181. DOI: 10.1145/1943552.1943572. [Online]. Available: <http://portal.acm.org/citation.cfm?doid=1943552.1943572>.

- [29] O. Oyman and S. Singh, “Quality of experience for HTTP adaptive streaming services”, *Ieee commun. mag.*, vol. 50, no. 4, pp. 20–27, Apr. 2012, ISSN: 0163-6804. DOI: 10.1109/MCOM.2012.6178830. [Online]. Available: <http://ieeexplore.ieee.org/document/6178830/>.
- [30] T. C. Thang, H. T. Le, A. T. Pham, and Y. M. Ro, “An Evaluation of Bitrate Adaptation Methods for HTTP Live Streaming”, *Ieee j. sel. areas commun.*, vol. 32, no. 4, pp. 693–705, Apr. 2014, ISSN: 0733-8716. DOI: 10.1109/JSAC.2014.140403. [Online]. Available: <http://ieeexplore.ieee.org/document/6774590/>.
- [31] S. Wei and V. Swaminathan, “Low Latency Live Video Streaming over HTTP 2.0”, *Proc. netw. oper. syst. support digit. audio video work.*, p. 37, 2014. DOI: 10.1145/2578260.2578277. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2578277>.
- [32] I. Sodagar, “The MPEG-DASH Standard for Multimedia Streaming Over the Internet”, *Ieee multimed.*, vol. 18, no. 4, pp. 62–67, Apr. 2011, ISSN: 1070-986X. DOI: 10.1109/MMUL.2011.71. [Online]. Available: <http://ieeexplore.ieee.org/document/6077864/>.
- [33] T. Lohmar, T. Einarsson, P. Frojdh, F. Gabin, and M. Kampmann, “Dynamic adaptive HTTP streaming of live content”, in *2011 ieee int. symp. a world wireless, mob. multimed. networks*, IEEE, Jun. 2011, pp. 1–8, ISBN: 978-1-4577-0352-2. DOI: 10.1109/WoWMoM.2011.5986186. [Online]. Available: <http://ieeexplore.ieee.org/document/5986186/>.
- [34] Apple Inc., *HTTP Live Streaming Overview*, 2016. [Online]. Available: <https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/StreamingMediaGuide/Introduction/Introduction.html> (visited on 05/04/2017).
- [35] A. Fecheyr-Lippens, “A review of http live streaming”, Tech. Rep., Jan. 2010. [Online]. Available: http://issuu.com/andruby/docs/http_live_streaming?viewMode=magazine&mode=embed.
- [36] R. Pantos and J. May, William, “draft-pantos-http-live-streaming-21 - HTTP Live Streaming”, 2017, [Online]. Available: <http://datatracker.ietf.org/drafts/current/>.
- [37] Apple Inc., *What’s New in HTTP Live Streaming - WWDC 2016 - Videos - Apple Developer*. [Online]. Available: <https://developer.apple.com/videos/play/wwdc2016/504/> (visited on 05/04/2017).
- [38] Mozilla, *Live streaming web audio and video - App Center / MDN*. [Online]. Available: https://developer.mozilla.org/en-US/Apps/Fundamentals/Audio_and_video_delivery/Live_streaming_web_audio_and_video (visited on 05/04/2017).
- [39] A. Zambelli, “IIS Smooth Streaming Technical Overview”, 2009. [Online]. Available: http://www.bogotobogo.com/VideoStreaming/Files/iis8/IIS_Smooth_Streaming_Technical_Overview.pdf.
- [40] A. Begen, T. Akgul, and M. Baugher, “Watching Video over the Web: Part 1: Streaming Protocols”, *Ieee internet comput.*, vol. 15, no. 2, pp. 54–63, Mar. 2011, ISSN: 1089-7801. DOI: 10.1109/MIC.2010.155. [Online]. Available: <http://ieeexplore.ieee.org/document/5677508/>.
- [41] Microsoft and IStreamPlanet, “Delivering Live and On-Demand Smooth Streaming A Step-by-Step Development”, [Online]. Available: http://download.microsoft.com/download/4/E/5/4E599FBB-6E34-4A74-B3C5-1391CB0FD55F/Delivering_Live_and_On-Demand_Smooth_Streaming.pdf.
- [42] M. Corporation, “IIS Media Services 3.0 Overview”, no. April, 2010. [Online]. Available: http://download.microsoft.com/download/3/3/8/33899CAB-98AD-4A25-A1FF-4172734BF6F4/IIS_Media_Services_30_Overview_FINAL.pdf.
- [43] S. Akhshabi, A. C. Begen, and C. Dovrolis, “An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP”, in *Proc. second annu. acm conf. multimed.*

- sysst. - mmsys '11*, New York, New York, USA: ACM Press, 2011, p. 157, ISBN: 9781450305181. DOI: 10.1145/1943552.1943574. [Online]. Available: <http://portal.acm.org/citation.cfm?doid=1943552.1943574>.
- [44] C. Müller, S. Lederer, and C. Timmerer, “An evaluation of dynamic adaptive streaming over HTTP in vehicular environments”, in *Proc. 4th work. mob. video - movid '12*, New York, New York, USA: ACM Press, 2012, p. 37, ISBN: 9781450311663. DOI: 10.1145/2151677.2151686. [Online]. Available: <http://dl.acm.org/citation.cfm?doid=2151677.2151686>.
- [45] *Adaptive Media Streaming: HLS vs MSS vs DASH | Digiflare*. [Online]. Available: <http://www.digiflare.com/adaptive-media-streaming-hls-vs-mss-vs-dash/> (visited on 05/04/2017).
- [46] O'Reilly, *Brief History of HTTP*, 2013. [Online]. Available: <https://hpbn.co/brief-history-of-http/>.
- [47] D. Mosberger and T. Jin, “httperf- A Tool for Measuring Web Server Performance”, [Online]. Available: <http://dl.acm.org/citation.cfm?id=306235>.
- [48] Clement Nedelcu, *Nginx HTTP Server Second Edition*, Second, P. Publishing, Ed. 2013, p. 318. [Online]. Available: https://books.google.pt/books?hl=pt-PT&lr=&id=9-R1VK5iX60C&oi=fnd&pg=PT17&dq=nginx+performance&ots=Uq0hq04uwU&sig=iMVZDTovtexztjJrujIhv0n5AvM&redir_esc=y#v=onepage&q=nginx%20performance&f=false.
- [49] Reese and Will, *Linux journal*. 173. Robert F. Young, 1994, vol. 2008, p. 2. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1412204>.
- [50] X. Chi, B. Liu, Q. Niu, and Q. Wu, “Web Load Balance and Cache Optimization Design Based Nginx under High-Concurrency Environment”, in *2012 third int. conf. digit. manuf. autom.*, IEEE, Jul. 2012, pp. 1029–1032, ISBN: 978-1-4673-2217-1. DOI: 10.1109/ICDMA.2012.241. [Online]. Available: <http://ieeexplore.ieee.org/document/6298691/>.
- [51] R. Arutyunyan, *Nginx-rtmp-module*. [Online]. Available: <https://github.com/arut/nginx-rtmp-module> (visited on 02/14/2017).
- [52] NGINX, *Nginx*. [Online]. Available: <https://nginx.org/en/> (visited on 04/05/2017).
- [53] Netcraft, *April 2017 Web Server Survey | Netcraft*. [Online]. Available: <https://news.netcraft.com/archives/2017/04/21/april-2017-web-server-survey.html> (visited on 05/05/2017).
- [54] NGINX, *Compare Features in NGINX Plus and NGINX Open Source*. [Online]. Available: <https://www.nginx.com/products/feature-matrix/> (visited on 05/05/2017).
- [55] C. Allen and J. Grden, “Introducing Red5”, *Essent. guid. to open source flash dev.*, pp. 309–335, 2008. DOI: 10.1007/978-1-4302-0994-2_12. [Online]. Available: http://link.springer.com/10.1007/978-1-4302-0994-2_12.
- [56] Sritrusta Sukaridhoto, Nobuo Funabiki, Toru Nakanishi, and D. Pramadihanto, “A comparative study of open source softwares for virtualization with streaming server applications”, in *2009 ieee 13th int. symp. consum. electron.*, IEEE, 2009, pp. 577–581, ISBN: 978-1-4244-2975-2. DOI: 10.1109/ISCE.2009.5156885. [Online]. Available: <http://ieeexplore.ieee.org/document/5156885/>.
- [57] Red5, *Red5-server*. [Online]. Available: <https://github.com/Red5/red5-server>.
- [58] —, *Red5 Pro*. [Online]. Available: <https://red5pro.com/> (visited on 04/05/2017).
- [59] —, *Red5 Media Server*. [Online]. Available: <http://red5.org/> (visited on 04/05/2017).

- [60] VideoLAN, *Transcode - VideoLAN Wiki*. [Online]. Available: <https://wiki.videolan.org/Transcode/> (visited on 04/18/2017).
- [61] Tomar and Suramya, “Converting video formats with FFmpeg”, *Linux j.*, vol. 2006, no. 146, p. 10, 2006, ISSN: 1075-3583. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1134792%7B%5C%7DCFDID=758080973%7B%5C%7DCFTOKEN=82886382>.
- [62] FFmpeg, *FFmpeg-Documentation*. [Online]. Available: <https://www.ffmpeg.org/documentation.html> (visited on 04/05/2017).
- [63] Blender, *Video Output — Blender Manual*. [Online]. Available: <https://docs.blender.org/manual/ko/dev/render/output/video.html> (visited on 06/22/2017).
- [64] Fabio Sonnati, *FFmpeg - the swiss army knife of Internet Streaming - part I*, 2012. [Online]. Available: <https://sonnati.wordpress.com/2011/07/11/ffmpeg-the-swiss-army-knife-of-internet-streaming-part-i/> (visited on 06/22/2017).
- [65] C. Müller and C. Timmerer, “A VLC media player plugin enabling dynamic adaptive streaming over HTTP”, in *Proc. 19th acm int. conf. multimed. - mm '11*, New York, New York, USA: ACM Press, 2011, p. 723, ISBN: 9781450306164. DOI: 10.1145/2072298.2072429. [Online]. Available: <http://dl.acm.org/citation.cfm?doid=2072298.2072429>.
- [66] *Official download of VLC media player, the best Open Source player - VideoLAN*. [Online]. Available: <http://www.videolan.org/vlc/> (visited on 05/05/2017).
- [67] GPAC, *GPAC | Multimedia Open Source Project*. [Online]. Available: <https://gpac.wp.imt.fr/> (visited on 05/05/2017).
- [68] —, *Support for Apple latest adaptive streaming format | GPAC*. [Online]. Available: <https://gpac.wp.imt.fr/2016/06/22/support-for-apple-latest-adaptive-streaming-format/> (visited on 05/05/2017).
- [69] Y. Kamen and L. Shirman, *Electronic programming guide*, US Patent 6,421,067, Jul. 2002. [Online]. Available: <https://www.google.com/patents/US6421067>.
- [70] *XMLTV*. [Online]. Available: <http://wiki.xmltv.org/index.php/XMLTVFormat> (visited on 05/05/2017).
- [71] IDC, *IDC: Smartphone OS Market Share 2016, 2015*. [Online]. Available: <http://www.idc.com/promo/smartphone-market-share/os> (visited on 04/21/2017).
- [72] A. K. Saha, “A Developer’s First Look at Android”, *Linux you*, pp. 48–50, Jan. 2008.
- [73] *Supported Media Formats | Android Developers*. [Online]. Available: <https://developer.android.com/guide/topics/media/media-formats.html#network> (visited on 04/24/2017).
- [74] *TV Input Framework | Android Open Source Project*. [Online]. Available: <https://source.android.com/devices/tv/> (visited on 04/05/2017).
- [75] Google, *ExoPlayer - Home*. [Online]. Available: <https://google.github.io/ExoPlayer/> (visited on 05/05/2017).
- [76] Apple Inc., *Media Layer*. [Online]. Available: <https://developer.apple.com/library/content/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/MediaLayer/MediaLayer.html> (visited on 04/24/2017).
- [77] Apple, *Using HTTP Live Streaming*, 2016. [Online]. Available: https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/StreamingMediaGuide/UsingHTTPLiveStreaming/UsingHTTPLiveStreaming.html#/apple_ref/doc/uid/TP40008332-CH102-SW1 (visited on 06/27/2017).

- [78] S. S. Brad Green, “AngularJS”, in *Angularjs*, S. St.Laurent and M. Blanchette, Eds., 1st, O’Reilly Media, Inc., 2013, ch. 1, ISBN: 1449344852 9781449344856. [Online]. Available: https://books.google.pt/books?hl=pt-PT&lr=&id=eNExy_X1YYcC&oi=fnd&pg=PR2&dq=angularjs&ots=wz8bdZOi_6&sig=_nkLsc8fU0k115h8S0eRCgQTxfA&redir_esc=y#v=onepage&q=angularjs&f=false.
- [79] *AngularJS: Tutorial: 6 - Two-way Data Binding*. [Online]. Available: https://docs.angularjs.org/tutorial/step_06 (visited on 04/25/2017).
- [80] *AngularJS — Superheroic JavaScript MVW Framework*. [Online]. Available: <https://angularjs.org/> (visited on 05/08/2017).
- [81] ACM, *React: Facebook’s Functional Turn on Writing JavaScript*, 2016. [Online]. Available: <http://queue.acm.org/detail.cfm?id=2994373>.
- [82] *Introducing JSX - React*. [Online]. Available: <https://facebook.github.io/react/docs/introducing-jsx.html> (visited on 04/25/2017).
- [83] *JSX - a faster, safer, easier JavaScript*. [Online]. Available: <https://jsx.github.io/> (visited on 04/25/2017).
- [84] *JSX Optimizer*. [Online]. Available: <https://www.slideshare.net/kazuho/jsx-optimizer> (visited on 04/25/2017).
- [85] *Flux | Application Architecture for Building User Interfaces*. [Online]. Available: <https://facebook.github.io/flux/docs/in-depth-overview.html#content> (visited on 04/25/2017).
- [86] Video-dev, *Video-dev/hls.js*. [Online]. Available: <https://github.com/video-dev/hls.js/tree/master> (visited on 04/06/2017).
- [87] Dash-Industry-Forum, *Dash-Industry-Forum/dash.js: A reference client implementation for the playback of MPEG DASH via Javascript and compliant browsers*. [Online]. Available: <https://github.com/Dash-Industry-Forum/dash.js/wiki> (visited on 04/06/2017).
- [88] Clappr, *Clappr: An extensible media player for the web*. [Online]. Available: <https://github.com/clappr/clappr> (visited on 04/06/2017).
- [89] Google, *google/shaka-player: JavaScript player library / DASH client / MSE-EME player*. [Online]. Available: <https://github.com/google/shaka-player> (visited on 04/07/2017).
- [90] *Statistics - YouTube*. [Online]. Available: <https://www.youtube.com/yt/press/statistics.html> (visited on 05/08/2017).
- [91] *YouTube for Developers - API Resources - YouTube*. [Online]. Available: <https://www.youtube.com/yt/dev/api-resources.html> (visited on 05/08/2017).
- [92] *Kodi | Open Source Home Theater Software*. [Online]. Available: <https://kodi.tv/> (visited on 05/08/2017).
- [93] *Kodi v17 (Krypton) changelog - Official Kodi Wiki*. [Online]. Available: [http://kodi.wiki/view/Kodi_v17_\(Krypton\)_changelog](http://kodi.wiki/view/Kodi_v17_(Krypton)_changelog) (visited on 05/08/2017).
- [94] *RTP Play - RTP*. [Online]. Available: <http://www.rtp.pt/play/> (visited on 05/08/2017).
- [95] *MEO Go - TV em qualquer lugar | MEO Go*. [Online]. Available: <https://meogo.meo.pt/> (visited on 05/08/2017).
- [96] Restreamer, *Requirements*. [Online]. Available: <https://datarhei.github.io/restreamer/wiki/requirements.html> (visited on 05/09/2017).

- [97] —, *Restreamer*. [Online]. Available: <https://datarhei.github.io/restreamer/> (visited on 05/09/2017).
- [98] *IPTV HLS SUPPORT - Tvheadend*. [Online]. Available: <https://tvheadend.org/boards/5/topics/14432> (visited on 05/09/2017).
- [99] *Visão geral - Tvheadend*. [Online]. Available: <https://tvheadend.org/> (visited on 05/09/2017).
- [100] *RealTek RTL2832U - LinuxTVWiki*. [Online]. Available: https://www.linuxtv.org/wiki/index.php/RealTek_RTL2832U (visited on 05/15/2017).
- [101] *dvbsnoop - DVB Stream Analyzer, MPEG Analyzer*. [Online]. Available: <http://dvbsnoop.sourceforge.net/> (visited on 05/16/2017).
- [102] *Program Association Table (PAT)[MPEG Syntax]*. [Online]. Available: <http://www.etherguidesystems.com/help/sdos/mpeg/syntax/tablesections/pat.aspx> (visited on 05/15/2017).
- [103] *What does the network information table (NIT) provide? | Tektronix*. [Online]. Available: <http://www.tek.com/support/faqs/what-does-network-information-table-nit-provide> (visited on 05/15/2017).
- [104] *Program Map Table (PMT) [MPEG Syntax]*. [Online]. Available: <http://www.etherguidesystems.com/help/sdos/mpeg/syntax/tablesections/pmts.aspx> (visited on 05/15/2017).
- [105] *mumudvb - LinuxTVWiki*. [Online]. Available: <https://www.linuxtv.org/wiki/index.php/Mumudvb> (visited on 05/16/2017).
- [106] *MuMuDVB*. [Online]. Available: <http://mumudvb.net/> (visited on 05/16/2017).
- [107] *CompilationGuide/Ubuntu - FFmpeg*. [Online]. Available: <https://trac.ffmpeg.org/wiki/CompilationGuide/Ubuntu> (visited on 05/16/2017).
- [108] *HWAccelIntro - FFmpeg*. [Online]. Available: <https://trac.ffmpeg.org/wiki/HWAccelIntro> (visited on 05/16/2017).
- [109] *Encode/H.264 - FFmpeg*. [Online]. Available: <https://trac.ffmpeg.org/wiki/Encode/H.264> (visited on 05/16/2017).
- [110] M. Bryars, *Tv_grab_dvb*, 2010. [Online]. Available: http://bryars.eu/projects/tv_grab_dvb/ (visited on 04/19/2017).
- [111] *EPG for IPTV*. [Online]. Available: <https://www.iptv-epg.com/> (visited on 05/05/2017).
- [112] *IPTV-EPG API documentation*. [Online]. Available: <http://iptv-epg.readthedocs.io/en/latest/> (visited on 05/05/2017).
- [113] NGINX, *If Is Evil | NGINX*. [Online]. Available: <https://www.nginx.com/resources/wiki/start/topics/depth/ifisevil/> (visited on 05/22/2017).
- [114] —, *Tuning NGINX for Performance*. [Online]. Available: <https://www.nginx.com/blog/tuning-nginx/> (visited on 05/22/2017).