TÉCNICO LISBOA

CASHED: Cloud-Assisted Adaptive and Scalable Video Streaming for Heterogeneous End-User Devices

Jérôme Mendes de Figueiredo

Thesis to obtain the Master of Science Degree in

Telecommunications and Informatics Engineering

Supervisors: Prof. Rui Antonio dos Santos Cruz
Prof. Mário Serafim dos Santos Nunes

Examination Committee

Chairperson: Prof. Paulo Jorge Pires Ferreira
Supervisor: Prof. Rui Antonio dos Santos Cruz
Members of the Committee: Prof. Artur Miguel do Amaral Arsénio

May 2014

Abstract

The Internet is a huge environment with a diversified number of connected terminals with different characteristics and specifications. In this reality, the heterogenous nature of the end-user devices introduces several problems to the distribution of multimedia content. Despite the huge growth and evolution of multimedia distribution services, many still do not offer adaptive and scalable mechanisms suitable for the networks and the diversity of end-user devices. This paper advances with a proposal for the development of a cloud assisted adaptive and scalable multimedia streaming solution for iOS or Android based end-user devices.

Keywords

Collaborative; scalable video coding; multimedia content; social environment; cloud computing

Resumo

A Internet é um ambiente enorme, com um diversificado número de terminais conectados apresentando diferentes características e especificações. Nesta realidade, a natureza heterogénea dos dispositivos moveis finais introduz vários problemas para a distribuição de conteúdo multimédia. Apesar do grande crescimento e evolução dos serviços de distribuição de multimédia, muitos ainda não oferecem mecanismos adaptativos e escaláveis adequados para as redes presentes e para a diversidade de dispositivos diferentes existentes. Este documento avança com uma proposta para o desenvolvimento de uma solução multimédia cloud-assisted de streaming adaptativo e escalável para dispositivos móveis suportando iOS ou Android.

Palavras Chave

Colaborativo; Codificação de Video Escalável; Conteúdo Multimédia; Ambiente Social; Computação em Nuvem

Contents

1 Introduction **1**

1.1 Motivation and Objectives . . . 3

1.2 Thesis layout . . . 4

2 Background and Related Work **5**

2.1 Traditional Streaming Technologies . . . 7

2.2 Web-based Streaming . . . 8

2.3 Video codecs . . . 11

2.4 MPEG Media File Formats . . . 16

2.5 Network Transmission Modes . . . 17

2.6 Mobile Operating Systems . . . 18

2.7 Media Technologies . . . 21

2.8 Content Distribution Networks . . . 22

2.9 Cloud Computing . . . 22

2.10 Related Work . . . 24

3 CASHED Design **25**

3.1 CASHED Architecture Design Requirements . . . 28

3.2 CASHED System Overview . . . 29

3.2.1 Media Presentation Description . . . 31

3.2.2 Streaming Operation . . . 31

3.3 Content Distribution System . . . 32

3.3.1 Video Transformation Subsystem . . . 33

3.3.2 Playlist File . . . 33

3.4 Client Application . . . 34

3.4.1 Adaptation Module . . . 34

3.4.2 Media Requester . . . 36

3.4.3 Media Description Parser . . . 36

3.4.4 Downloader . . . 36

3.4.5 Device Profiler . . . 37
3.4.6 Media Player and User Interface . . . 37

4 Implementation **39**
4.1 Development Process . . . 41
4.2 Development Environment . . . 42
4.3 CASHED Client Application . . . 42
4.3.1 User Interface . . . 42
4.3.2 Media Player/Decoder . . . 42
4.3.2.A Internal mini HTTP Server . . . 43
4.3.3 Media Description Parser . . . 44
4.3.4 Media Requester . . . 44
4.3.5 Downloader . . . 44
4.3.6 Adaptation System . . . 44
4.4 CASHED Distribution System . . . 46
4.4.1 SVC-to-AVC Rewriter . . . 46
4.4.2 Media Segmenter . . . 46
4.4.3 Log File . . . 46
4.4.4 HTTP Server . . . 47

5 Evaluation **49**
5.1 Evaluation Objectives and Experimental Scenarios . . . 51
5.2 Evaluation Results . . . 52
5.2.1 Startup Time . . . 53
5.2.2 Adaptation Performance . . . 56
5.2.3 Global Application Performance . . . 58

6 Conclusion and future work **61**
6.1 Conclusions . . . 63
6.2 System Limitations and Future Work . . . 63

List of Figures

1.1 CASHED ecosystem 4

2.1 H.264/AVC architecture 12
2.2 SVC Principle 13
2.3 Representation of possible SVC Scalability dimensions 13
2.4 MPEG2-Transport Stream 17
2.5 iOS layer architecture 19
2.6 Android architecture 20
2.7 Example of a Cloud Computing Architecture 23

3.1 CASHED Components Architecture 29
3.2 CASHED streaming message sequence 32
3.3 Distribution Side - Video Transcode and Partitioning 33
3.4 Adaptation Controller 35
3.5 User Interface 37

4.1 Complete User Interface 43
4.2 XML Parser Delegation 45

5.1 Test Environment 51
5.2 Startup Time - Simulator Wifi (Lower is better) 53
5.3 Startup Time Under Wifi - iPhone 4 vs iPhone 5s (Lower is better) 53
5.4 Startup Time - iPhone 5s LTE (Lower is better) 54
5.5 Startup Time Under 3G - Simulator vs iPhone 5s (Lower is better) 54
5.6 Startup Time - Simulator Edge (Lower is better) 55
5.7 Startup Time - Global View (Lower is better) 55
5.8 Adaptation System Behavior Test 56
5.9 Adaptation System Test 3 56

5.10 Adaptation System Behavior Test 57
5.11 Average Transition Time Between Different Networks 57
5.12 Global Performance 58
5.13 Gloabel Performance With Network Capture 59

List of Tables

2.1 Streaming Technologies Comparison 8
2.2 iOS Recommended Compression Standards 21
2.3 Android Operating System (OS) Recommended Compression Standards 22

4.1 MIME Type Server Configuration 44
4.2 SVC Profiles 45

5.1 Network Link Conditioner Profiles 51
5.2 Layers properties 52

List of Algorithms

3.1 Quality Level Estimation 36
3.2 Transfer Rate Estimation [in kbps] 36

Listings

2.1 Example of a .M3U8 file. 11
3.1 Example of a MPD file. 31
3.2 Example of a Playlist file for the play-out session. 34

Acronyms

AAC Advanced Audio Coding

API Application Program Interface

AV Audio-Visual

AVC Advanced Video Coding

BMFF Base Media File Format

CASHED Cloud-Assisted Adaptive and Scalable Video Streaming for Heterogenous End-User Devices

CC Cloud Computing

CDN Content Distribution Network

CPU Central Processing Unit

DASH Dynamic Adaptive Streaming over HTTP

DVD Digital Versatile Disk

ES Elementary Stream

ftyp File Type

GPRS General Packet Radio Service

H.264/AVC Advanced Video Coding

H.264/SVC Scalable Video Coding

HD High Definition

HLS HTTP Live Streaming

HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IETF	Internet Engineering Task Force
IGMP	Internet Group Management Protocol
IIS	Internet Information Services
IP	Internet Protocol
IPTV	IP Television
ISP	Internet Service Provider
IT	Information Technology
LAN	Local Area Network
LTE	Long Term Evolution
MDC	Multiple Description Coding
MIME	Multipurpose Internet Mail Extension
MPD	Media Presentation Description
MPEG	Moving Picture Expert Group
NAL	Network Abstraction Layer
NAT	Network Address Translation
OS	Operating System
OSMF	Open Source Media Framework
P2P	Peer-to-Peer
PaaS	Platform as a Service
PES	Packetized Elementary Streams
QoS	Quality Of Service
RTCP	RTP Control Protocol
RTMP	Real Time Messaging Protocol

RTP	Real-time Transport Protocol
RTSP	Real Time Streaming Protocol
SaaS	Software as a Service
SD	Standard Definition
SEI	Supplemental Enhancement Information
SMS	Short Message Service
SNR	Signal-to-Noise Ratio
SVC	Scalable Video Coding
TCP	Transport Control Protocol
TS	Transport Stream
TTL	Time-to-Live
UDP	User Datagram Protocol
UI	User Interface
UMTS	Universal Mobile Telecommunication System
URL	Uniform Resource Locator
VCL	Video Coding Layer
VoD	Video On Demand
WLAN	Wireless Local Area Network
WWAN	Wireless Wide Area Network
XML	Extensible Markup Language
DQId	Direct Quality ID
DTD	Document Type Definition
IST	Instituto Superior Tecnico
EDGE	Enhanced Data Rates for GSM Evolution
SNS	Social networking service

1

Introduction

Contents

1.1 Motivation and Objectives 3
1.2 Thesis layout 4

1.1 Motivation and Objectives

At the beginning of the multimedia content era, when it first started to be available across the Internet, solely high-end computers were capable of reproducing such contents. Due to the existing network connections and access types at the time, the main issues related to multimedia content streaming were the available bandwidth and network congestion. These issues were addressed with the evolution of the access technologies, and the growth and expansion of broadband networks. But this growth also brought an increase in popularity and consequently, a huge increase in the number of connected devices of all types, notably in handheld, following the introduction of Apple's iPhone and iPad. It is nowadays common to find a wide range of smartphones, tablet computers and game consoles equipped with multitouch screens supporting Standard Definition (SD) and High Definition (HD) resolutions and several network access technologies such as Wireless Local Area Networks (WLANs) and 3G/4G Wireless Wide Area Networks (WWANs). Allied with the evolution of broadband access technologies, video services also evolved, providing new ways to enjoy contents almost everywhere at any time. The evolution of video technologies also led to the adoption of new standards, in terms of image quality, driving the need to keep newer devices capable of supporting higher quality and higher resolution formats.

But the heterogeneity in capabilities and characteristics of handheld devices, such as screen size and resolution, Central Processing Unit (CPU) power and Operating System (OS) originates a huge fragmentation on the support of streaming media caused by the many different specifications and some lack of standards. This situation also presents huge challenges to the distribution of streaming media, hardly scalable for every device. The main efforts and solutions to address these situations have been made by companies such as Apple, Adobe and Microsoft [1–3], by supporting dynamic variations of the video streaming quality in an almost transparent fashion. These solutions provide services to numerous different devices, but to work they normally require the original content to be encoded in multiple qualities and bit-rates.

On the video encoding side, the Annex G of the H.264/Advanced Video Coding (AVC) standard, also know as Scalable Video Coding (SVC) [4, 5] offers new methods to encode and compress video, but in quality layers, combining spatial, temporal and fidelity dimensions, and consequently, providing streamable scalability for a single video content. SVC opens therefore new possibilities for multimedia content distribution since it allows the creation of scalable and adaptive media in a easy and simplified manner.

Additionally, Peer-to-Peer (P2P) networks and the emergence of Cloud Computing (CC) are changing the way Information Technologys (ITs) are used by allowing users to take advantage of resources available through the Internet. Cloud technologies provide ubiquitous, convenient and on-demand access to resources and computing capabilities, beyond human interaction needs, over standard network mechanisms.

Summarising, on one perspective there is the evolution of the video coding techniques, mobile end-user devices and network access technologies, and also the eagerness for multimedia content at any moment and any place. On another perspective, P2P communication methods and Content Distribution Network (CDN) networks, along with the elasticity offered by Cloud technologies can be appropriate for multimedia content distribution, namely for SVC adaptive and scalable video content. However, as far as known, these features and capabilities have yet to be combined in a single solution.

In an attempt to address the mentioned topics, this thesis describes the development of Cloud-Assisted Adaptive and Scalable Video Streaming for Heterogenous End-User Devices (CASHED), a Streaming Media environment supporting Adaptive and Scalable Video encoded media to "smart" mobile end-user devices. The idea behind CASHED, as illustrated in Figure 1.1, is for a Cloud-assisted streaming environment, where scalable and adaptive high-quality and high-definition multimedia contents can be consumed by common end-user devices (iOS or Android based) without requiring specialised decoding capabilities, but reaching the highest experienced streaming quality.

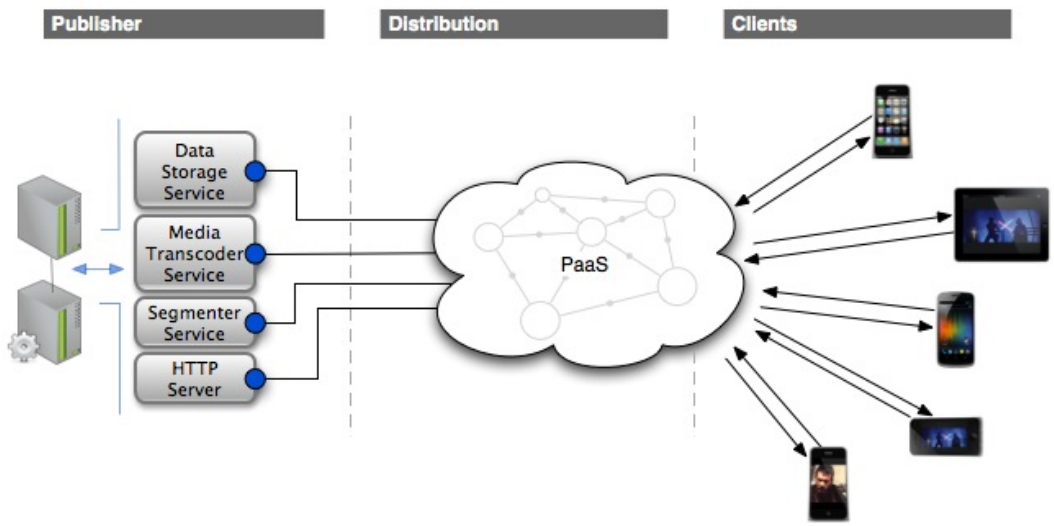


Figure 1.1: CASHED ecosystem

1.2 Thesis layout

This thesis is is organized as follows: Chapter 1 gives a general introduction to the problem, with the motivations and objectives for the work. Chapter 2 provides background information as well as related works. In Chapter 3 the concept of CASHED is introduced and the architecture of the solution described. Chapter 4 describes the prototype implementation process of CASHED, followed by its evaluation in Chapter 5. Chapter 6 summarizes the contributions, presents the architectural shortcomings and suggests areas for future work.

2

Background and Related Work

Contents

2.1 Traditional Streaming Technologies 7
2.2 Web-based Streaming 8
2.3 Video codecs 11
2.4 MPEG Media File Formats 16
2.5 Network Transmission Modes 17
2.6 Mobile Operating Systems 18
2.7 Media Technologies 21
2.8 Content Distribution Networks 22
2.9 Cloud Computing 22
2.10 Related Work 24

This chapter presents the concepts and the State-of-the-Art on streaming media, video coding techniques, network transmission modes, content distribution solutions such as Content Distribution Network (CDN) and the Cloud Computing (CC) model, as well as the key aspects of Apple´s iOS and Google's Android OS and their supported video technologies. A research on other works that address similar objectives as this thesis is also presented and discussed.

2.1 Traditional Streaming Technologies

While there are many protocols created with the purpose of sending data packets over the Internet, the focus in this section will be on the two most used traditional protocols for media streaming, i.e., the Real Time Streaming Protocol (RTSP) and Real-time Transport Protocol (RTP)/RTP Control Protocol (RTCP). When using either RTSP and RTP/RTCP, the content is fragmented and transported under the form of small packet payloads, each representing a temporal portion of the original segment. With both protocols, from the starting moment the client system makes a connection until its end, the serving system is aware of the client state as well as the state of the video buffer, therefore adjusting the rate of the transmission [6, 7].

Real-Time Streaming Protocol

RTSP, is an application-level protocol that establishes and provides control of continuous data streams for multimedia content with real-time properties. This protocol does not transmits or delivers the media content, but rather provides control over multiple sessions of media streams. This means that in RTSP the notion of connection does not exists, maintaining instead a session marked with an identifier. This allows RTSP to be untied of any transport-level protocols such as Transport Control Protocol (TCP) or User Datagram Protocol (UDP) and clients can open and close several transport connections relying on the the session identifier [6]. RTSP operations and purposes over the controlled stream are independent of the transport protocol, therefore RTSP may use an independent protocol such a RTP to stream the media and the data delivery will continue even if there are no RTSP requests to the media server. A important problem arises by the use of UDP for media streaming as downstream packets may be blocked when end-user devices are behind a firewall or a Network Address Translation (NAT) box. Additionally, the lack of congestion control mechanisms may affect the overall experience quality to the end-user due to the lack of congestion control mechanisms.

Real-time Transport Protocol (RTP)/RTP Control Protocol (RTCP)

RTP provides end-to-end delivery services for data with real-time properties such as audio and video. While RTP is responsible for the data transfer, this protocol is used in cooperation with RTCP to monitor data transmission statistics and Quality Of Service (QoS) as well as helping multiple

streams synchronization through periodical message interchange. RTP includes mechanisms for jitter compensation but also detection mechanisms for out-of-order sequence arrivals in data [7]. Each packet comprises a header as well as a payload and it was designed to support a wide variety of multimedia formats. With RTP the content is encoded and fragmented granting its type into the RTP packet payload and charging the header to includes all the necessary packet information such as the payload type, timestamp, sequence number and a marker.

2.2 Web-based Streaming

In a era where watching a video is as easy as taking out a smartphone from a pocket and searching for the intended video, one of the most popular ways to watch this type of content consist on websites like Youtube[1], Dailymotion[2] or Vimeo[3]. These sites generally use a streaming method called progressive download. This method cannot be considered a real streaming technique but a "play while downloading" mechanism. With the necessity to adapt to the always growing need for better performance, new web-based streaming methods have been developed, such as Adobe Dynamic Streaming [8], Microsoft Smooth Streaming [3] and Apple HTTP Live Streaming [1]. The main idea behind these methods of Hypertext Transfer Protocol (HTTP) Adaptive Streaming is the support for a dynamic adaptation of the video quality following a bitrate-quality ratio. A brief comparison of these systems is presented on Table 2.1.

Table 2.1: Streaming Technologies Comparison

	Dynamic Streaming	Smooth Streaming	HLS
Streaming Protocol	RTMP	HTTP	HTTP
Video Codec	H.264, VP6	H.264	H.264
Audio Codec	AAC, MP3	WMA, AAC	AAC, MP3
Container Format	MP4, FLV,	MP4	MPEG2-TS
iOS	NO	YES	YES
Android	NO	YES	YES

Progressive Download

Although progressive download is often categorised as a streaming solution, it is not really a streaming technique, but just a bulk HTTP file download of a video file. The confusion on categorising this solution as "streaming" comes with the fact that progressive download file playback behaves in a very similar way to streaming. The file starts playing a few seconds after it is selected (starting when there is enough content in the media player buffer) while the player continues downloading in the background the file content until the movie data is complete. Even when the user pauses the video playback, if the movie content is not yet complete the player continues the

[1] http://www.youtube.com
[2] http://www.dailymotion.com
[3] http://www.vimeo.com

download in the background. Since progressive download requires the file to be linearly downloaded, when it comes to fast forward and rewind, this system is constrained by the bandwidth of the connection.

Adobe Dynamic Streaming

Adobe Dynamic Streaming is a technology created and developed by Adobe for streaming multimedia content with support for adaptive bit-rate over Real Time Messaging Protocol (RTMP) (proprietary protocol developed by Macromedia for streaming media content and data over the Internet) or HTTP. This technology is highly integrated with the Flash Player which is another Adobe technology. Dynamic Streaming delivers the existing media in a fragmented format and requires a manifest (metadata) to describe the media. For adaptive bit-rate delivery, the video content is encoded in multiple segment streams with different bit-rates, all aligned to the same keyframe. This ensures a good synchronisation and allows a smooth transition between bit-rates since Flash can only make this transition at keyframes [8]. Adobe recommends the use of Open Source Media Framework (OSMF) to easily create video players incorporating the adaptation process. Adobe Dynamic Streaming requires a serving component for the bit-rate switching requested by the client where most of the streaming logic occurs [2].

Microsoft Smooth Streaming

Microsoft Smooth Streaming, as the name implies, was created by Microsoft as part of their Silverlight architecture and presents all the characteristics of a typical adaptive streaming solution. Smooth Streaming adapts the stream to the user's local conditions, allowing for a uninterrupted and seamless experience. The system detects bandwidth and CPU usage fluctuations in order to switch the content quality almost in real-time, i.e., end-users with higher bandwidth connections can enjoy contents with higher definitions than other users with lower bandwidth connections.

Smooth Streaming uses the Moving Picture Expert Group (MPEG)-4 Part 14 [9] file format for storage (disk) and transport (wire). This specification defines that the content is encoded in multiple MPEG-4 chunks of different bit-rates. These chunks are stored in contiguous MP4 container files, one for each bit-rate/quality. The Movie Fragment boxes in the MP4 file can then be served over HTTP by an Internet Information Services (IIS) Web server that extracts the chunks to be transferred upon requests from the client player [3].

Smooth Streaming also requires a manifest file encoded in Extensible Markup Language (XML) with metadata describing the content. The manifest file is transferred to the player at the beginning of the session to provide information about the encoders used, the the bit-rate alternatives, the resolutions as well as the Uniform Resource Locator (URL) to access he content chunks, allowing the correct initialisation of the decoder and the control of the playback session.

Apple HTTP Live Streaming

HTTP Live Streaming is a technology developed by Apple to grant content providers the possibility to distribute live or prerecorded audio and video content to iOS devices using a standard web server. The support for this technology was later included in their QuickTime for OS X. Apple HTTP Live Streaming was published as an Internet Engineering Task Force (IETF) Internet-Draft [10].

Conceptually, an HTTP Live Streaming solution consists of a server component, a distribution component, and a client software. The responsibility of the **server component** is to take the content input files or streams, encode them in several quality/bit-rate versions which are then encapsulated in a suitable transport format for delivery. Additionally, the server component has the responsibility to segment the encapsulated media for distribution.

When it comes to the **distribution component**, it simply consists of standard web servers responsible for the acceptance of client requests and consequently the delivering of prepared media and related resources (playlists) to the client. The **client software** is responsible for determining the appropriate quality/bit-rate versions and download the corresponding media segments.

The media player in iOS devices, which is a built-in client software, is automatically launched when the Safari browser finds tags like <OBJECT> or <VIDEO> with an URL whose Multipurpose Internet Mail Extension (MIME) type is supported by the media player. Additionally, the media player can be launched from custom iOS applications using the media player framework.

In a typical configuration, the source audio and video contents are encoded by an hardware encoder in H.264 video and Advanced Audio Coding (AAC) audio elementary streams and multiplexed into MPEG-2 Transport Stream (TS) container format. In the case of stored movies for on-demand consumption the MPEG-2 TS is then cut into multiple segments by a software segmenter and these files are then stored in a web server. The segmenter also creates and maintains an index file holding content metadata and the list (absolute or relative URLs) of the media segments. This index file is published on the web server allowing client software to read the index and request the listed segment files in the correct order to decode and display continuously without any gap [11].

The index file created by the stream/file segmenter contains not just the list of media segments but also content metadata. This index file is of the type .M3U8 playlist and provides the absolute or relative URLs of the resources to be accessed by clients allowing them to request in sequence those resources or alternate index files for different bandwidths (bit-rates/qualities).

Listing 2.1 shows a very simple example of an index file in the form of a .M3U8 playlist produced by a segmenter for a media stream, dividing it in three unencrypted 10-second duration media files.

Listing 2.1: Example of a .M3U8 file.

```
#EXTM3U
#EXT−X−TARGETDURATION:10
#EXT−X−MEDIA−SEQUENCE:1
#EXTINT:10, http://media.example.com/segment0.ts
#EXTINF:10, http://media.example.com/segment1.ts
#EXTINF:10, http://media.example.com/segment2.ts
#EXT−X−ENDLIST
```

If caching behaviour for downstream web caches is desired, it can be achieved with some Time-to-Live (TTL) values adjustments of the .M3U8 files, since, for live streaming, these files are frequently overwritten and the latest version should be downloaded for each new interval of requests. If no EXT-X-ENDLIST tag is found in the index file it means that the index file is part of an ongoing live streaming session leading the client to load a new version of the index file periodically.

Dynamic Adaptive Streaming over HTTP (DASH)

Apple, Microsoft and Cisco took active part in an initiative to harmonize the standards for Web-streaming, the so called MPEG DASH [12] aiming to converge the different industry solutions in this area (Apple HTTP Live Streaming (HLS), Microsoft Smooth-streaming, Adobe Dynamic Streaming, 3GPP Adaptive Streaming over HTTP and OpenIPTV Forum HTTP Streaming). DASH is an MPEG-Standard that defines the formats for multimedia delivery over HTTP and the description – the Media Presentation Description (MPD) – of the media to be downloaded.

2.3 Video codecs

The video coding world offers a wide variety of codecs each with its own specifications. It is always difficult to choose the most appropriate video codec but due to the success of web video distribution services, the H.264 standards [5,13] have been the main choice in general. Regardless of their success, H.264 standards have been the focus of some debates about their time-limited royalty-free status. As far as it goes, these standards are free to use up to 2016, but this period has been shorter in the past and then extended and may still change. Recently, the VP8 codec [14] and its successor VP9, developed by Google has gained popularity, specially due to its royalty-free nature.

MPEG and ISO/IEC H.264/AVC

This standard was created with the purpose to be deployed across a wide variety of devices and networks. An H.264 video encoder carries out prediction, transform and encoding processes to produce a compressed H.264 bitstream. H.264 video decoder carries out the complementary processes of fast decoding, inverse transform and reconstruction to produce a decoded video sequence. H.264/AVC architecture is represented in Figure 2.1. To answer the flexibility and customisation needs, H.264/AVC design comprehends a Video Coding Layer (VCL) and a Network

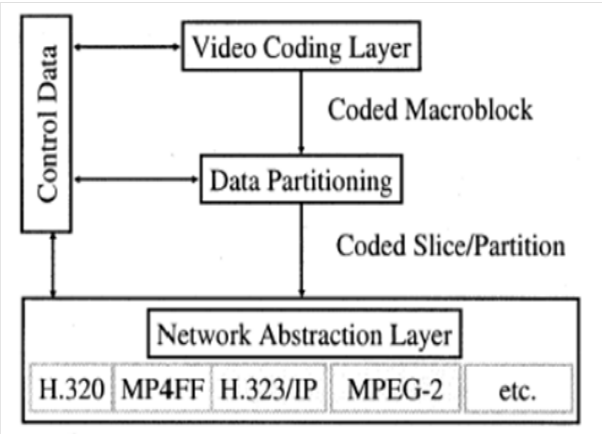


Figure 2.1: H.264/AVC architecture

Abstraction Layer (NAL). The VCL is macroblock-based, and designed to represent the video content (the signal processing functionality of the codec) while a NAL encapsulates the VCL slices (bit strings containing macroblock data) and supplies header information in a way that is appropriate for conveyance by a multitude of transport layers or storage medias. NAL units start with a one-byte long header that signals the type of NAL unit and if the VCL NAL part belongs to a reference or non-reference picture. NAL units can be classified as VCL or non VCL. VCL NAL units contain data partitions in the form of slices representing the coded content, while non-VCL NAL units contain additional information like Supplemental Enhancement Information (SEI) or parameter sets for the decoding process.

H.264/AVC was built with slice coding modes that include three types of slices: **I**, **P** and **B**, described as follows. The **I** (Intra) slices has all its blocks coded using intra prediction mode. In terms of bit-rate, these are the most costly slices since they can be rebuilt without any reference from any of the other frames. What this means is that, temporal prediction and temporal redundancy from any of the other slices is not used. The **P** (Predictive) slices use forward prediction from the previous **I** slice or **P** slices. This means that this type of slice cannot be reconstructed without the information of any other **I** or **P** slice from previous moments. The **B** (Bidirectional) slices use both forward and backward prediction, meaning that **B** slices can be reconstructed from previous and future **I** or **P** slices. While all the slices use Intra-picture coding which uses the temporal redundancy inside the same picture, **P** and **B** slices may also use Inter-picture coding which meaning that nearby pictures use temporal redundancy between them. The next step in H.264/AVC processing consist in the quantisation and compression of the residuals, followed by a context-based adaptive entropy coding, and finally the encapsulation of the VCL data into NAL units.

MPEG and ISO/IEC H.264/SVC

To support scalability, H.264/SVC was proposed and designed by the Fraunhofer Institute in Ger-

many as an extension of the H.264/AVC (Annex G) and has been since then an active research and standardisation area.

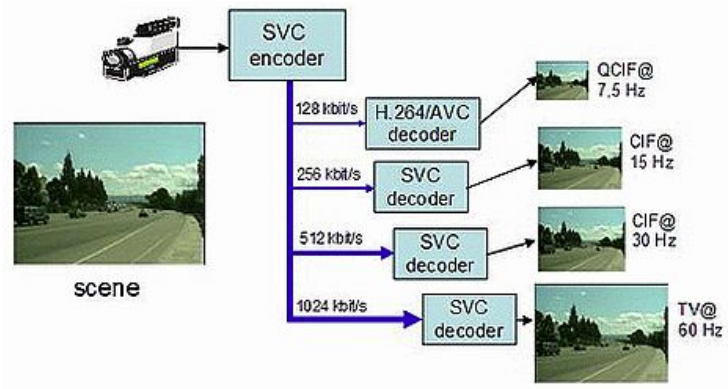


Figure 2.2: SVC Principle

With H.264/SVC it is possible to provide different spatio-temporal resolutions with variable fidelity, avoiding the need to encode the same media content multiple times with different bit-rates/qualities [15]. This solution was designed to be simple, not requiring high computation power to decode,

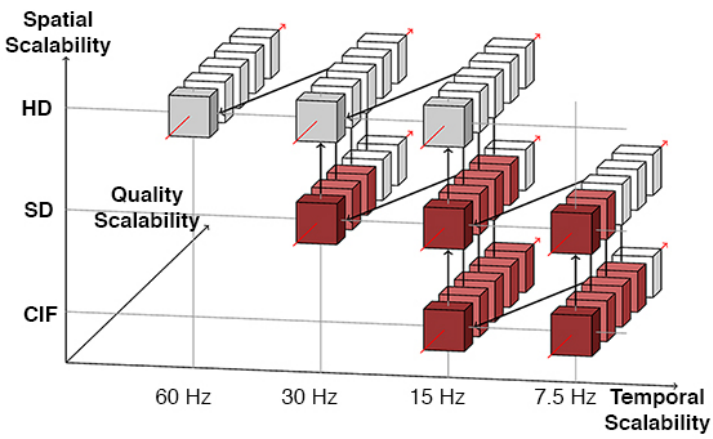


Figure 2.3: Representation of possible SVC Scalability dimensions

i.e., able to be performed in a mid-level network component or end-user device. In its design, the scalability in SVC allows partitioning the source bit-stream in multiple sub streams through cuts in parts of the data, creating several layers of quality with the partitions. It is important to mention that the base layer of any H.264/SVC encoded content is backward compatible with H.264/AVC, i.e., it can be decoded in legacy devices not requiring additional modifications to support layered coding. For H.264/SVC compliant devices, each of the enhancement layers represents an incremental quality level as illustrated in Figure 2.2. H.264/SVC considers three scalability dimensions which are, the **temporal scalability**, the **spatial scalability** and the **quality scalability**, also known

as Signal-to-Noise Ratio (SNR) or fidelity. The spatial scalability allows to encode a video with multiple spatial resolutions/dimensions, supporting therefore multiple display sizes (with arbitrary ratios), allowing therefore changes in the displayed frame size [16]. This implies that SVC content is not restrained to fixed scaling ratios. As an example, some lower layers can be encoded for SD with a 4:3 aspect ratio, while higher layers could be encoded for HD with a 16:9 aspect ratio. For seamless quality perception, spatial scalability should however be coherent in terms of picture aspect ratio, from SD to HD resolutions. Spatial Scalability uses motion compensated as well as intra-prediction techniques in each layer. Additionally, SVC includes an inter-layer prediction method which reduces the spatial redundancy between the enhancement layers by exploiting their statistical dependencies.

The temporal scalability dimension was already present in H.264/AVC (commonly implemented by B-Slices), and delineates the support for multiple frame rates. According to [15], both spatial and temporal scalability describe cases where the original content is represented with a reduced picture size (spatial resolution) or frame rate (temporal resolution) by subsets of the bit-stream.

Finally, quality scalability represents the levels of fidelity of the content displayed to the end-user. With this scalability dimension, the spatio-temporal resolution presented by the sub-stream is the same as the original content, but with an inferior fidelity [15]. Fidelity is often referred to as SNR. The three scalability dimension of SVC can be combined together to create many different representations inside a single scalable bit stream. H.264/SVC maintains the bit stream organisation introduced in H.264/AVC that uses NAL units, meaning that the scalability dimensions are distributed over the bit streams encapsulated by NAL units, and each scalability level is represented by a NAL unit [5]. Each NAL unit has fidelity identifiers, named **dependency_id(D)**, **temporal_id(T)** and **quality_id(Q)** represented by integer values indicating the corresponding fidelity component order in the hierarchy, from higher to lower fidelity. The NAL units store these values alongside other important ones in their extended header. These are important since they define the inter-dependency between NAL units, and the loss of a NAL unit results in a diminished quality in the layer, or in some severe cases it even forestalls the video decoding. Specifications for H.264/SVC dictates that layers are organised hierarchically from base layer up to the last enhancement layer, and that each layer is dependent of the previous one, having a different impact in the combined video quality [16].

The key advantages for using scalable coding over non-scalable coding techniques, can be summarised as follows:

- Using SVC, multiple heterogeneous clients with different capabilities can be served from one single media content as the scalable information can be transported within the same video stream. This avoids the switching between independent bit-streams and leads to a simplified

and better adaptation of the bit-rate;

- As scalable coding techniques use a layered structure to represent the encoded content, the lowest layer (base layer) contains sufficient information for the immediate video playback and each additional enhancement layer increases hierarchically the quality dimensions of the content.
- SVC provides good accommodation for heterogeneous network environments and devices, typical of mobile handheld devices which easily change their network conditions forming an non-homogeneous group.

Multiple Description Coding (MDC)

Multiple Description Coding (MDC) is conceptually similar to SVC but uses a different approach to achieve its goal. MDC is a coding technique which encodes a video in several sub-streams. Each one of these sub-streams bears a base quality level, is treated as a description and can be decoded independently. It is then possible to reconstruct the original content if all the descriptions are received and combined, or reconstruct to a quality close to the original if a lossy coding is used. This behaviour indicates that each description received adds to the quality improving the final result after the reconstruction of the content [17]. The main difference between MDC and SVC is found on the way the different parts of the video are prioritised. In MDC each description is similar and adds the same amount of quality to the final content while in SVC the different parts of the video have different priorities (from base layer up to the last enhancement layer), and each layer increases the quality amount of the previous layer hierarchically.

Google WebM and VP8/Vp9

As an alternative to H.264/AVC codec, On2 Technologies released in September 13, 2008 the VP8 codec. This codec proposed similar quality pictures and data rates as well as similar encoding speeds as H.264/AVC. On2 Technologies was later acquired by Google which announced WebM[4], an audio-video format to be used with the HTML5 video tag, designed to support the royalty-free VP8 codec [18]. VP8 shares many principles with H.264/AVC, but its greatest advantage may come from the licensing scheme, a BSD-like license, which allows it to stay royalty-free in the future. It is nonetheless important to indicate that in February 2010, H.264/AVC royalty-free period was extended to 2016 and it was acknowledged that end users would not be charged for any Internet broadcast visualisation throughout the entire life of the license [19].

In terms of architecture, VP8 uses two different types of frames. The first type are frames similar to H.264/AVC **I**-Frames, known as Key Frames, which are the ones where intra-prediction happens.

[4] http://www.webmproject.org

The second type of frames are inter frames which are similar to H.264/AVC **P**-Frames, hence prediction happens with reference to prior coded frames. VP8 does not uses bi-directional prediction frames making it different from MPEG codecs. A WebM file consists of VP8 video and Vorbis[5] audio streams, in a container based on a profile of Matroska[6].

2.4 MPEG Media File Formats

HTTP video streaming is a common topic in the multimedia communication domain, and since HTTP only defines the transport mechanisms, a media container format multiplexing the content streams (video, audio, timed-text) is usually necessary. The dominant solutions for this purpose are MPEG-2 TS [20] and ISO Base Media File Format (BMFF) [21], although other non-standard formats are sometimes used.

ISO BMFF

ISO BMFF comprises the timing, structure and media information for timed sequences of media data, such as Audio-Visual (AV) representations. ISO BMFF uses an object oriented structure where multimedia streams, as well as the respective metadata, are contained in structures referred to as "boxes". These boxes can be organised sequentially or hierarchically and each box comprehends a type and a size. Brands are used in the file format as identifiers of the steam specifications. Brands are signalled at the beginning of the container file in a File Type (ftyp)-box, and its presence denotes both a claim — indicating that the file complies with all the specifications of that brand — and a permission — for a reader, possibly implementing only that brand, to read and interpret the file. ISO BMFF supports network streaming and local file playback. To support streaming, "hint" tracks are commonly used to include the information about the data units in the stream such as the sequence order, timing and content, and may be used when one or more packet streams of data are recorded. Independent "hint" tracks, for different protocols, can be used, allowing the media to play when transported over the related protocols without the need to create supplementary copies of the media file.

MPEG-2 Transport Stream (TS)

MPEG-2 TS specifies a standard container format that encapsulates Elementary Streams (ESs) offering error correction mechanisms as well as stream synchronisation that preserves transmission integrity in the presence of a degraded signal. While Program Streams were designed for reliable media transfer from physical supports like Digital Versatile Disk (DVD) or BluRay discs, TSs where designed for media transfer over less reliable communication systems such as terres-

[5] http://www.vorbis.com
[6] http://www.matroska.org

trial, mobile or satellite broadcast. Transport packets in MPEG-2 TS have a fixed size of 188 bytes, including a 4 byte header [20]. All the data streams to be transported, whether video, audio or timed-text, including the sequence header and all the subparts of a sequence, form ESs. A Packetized Elementary Streams (PES) consists of a single packetised ES in which every packet starts with an additional packet header. PES packets are of variable length which do not correspond to

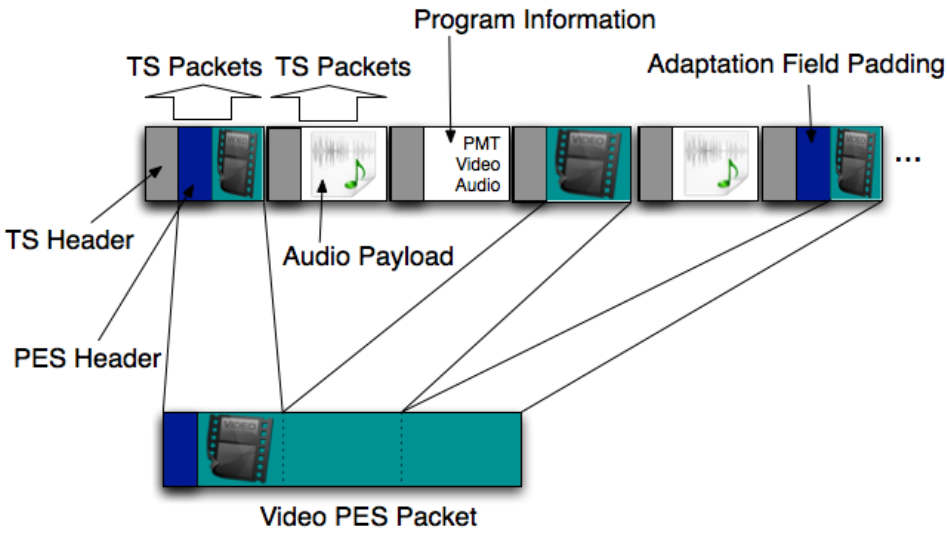


Figure 2.4: MPEG2-Transport Stream

the fixed transport packets size, and thus may be longer than a transport packet. When forming a transport packet from a PES, the header is always placed at the starting point of the transport packet payload, directly following the transport packet header. The remaining content of the PES packet fills consecutive transport packets payloads until all PES packets are used, leaving the left space at the final of the transport packet to be filled with stuffing 0 bytes. A 1 byte stream ID is included in each PES packet to identify the payload's source. Additionally, the PES header may also include timing references to help synchronise multiple streams such as audio and video for playback.

Even if MPEG-2 TS was not initially designed to make use of MPEG-4 encoded contents, there has been a wide adoption in IP Television (IPTV) distribution systems for the transport of MPEG-4 data. These mechanisms can be both TS over UDP/Internet Protocol (IP) or TS over RTP/UDP/IP [22].

2.5 Network Transmission Modes

In the IP world, there are two main modes of communication: Unicast and Multicast. Unicast is a two-way communication between two (and only two) networked devices. Multicast is a one-way com-

munication initiated by one device and acquired by zero or more devices. Numerous multicast-emitting devices still inject multicast packets on the network even if there are no joined listeners.

IP Unicast

IP Unicast consists in a one-to-one communication between two devices which typically are a server and a client. Stream wise, this means that a serving node dedicates a stream to each client leading each of them to take a portion of the total bandwidth. When unicast is used, the information is restrained by the receiver, meaning that if any packet is lost or gets corrupted, a request for retransmission can be made.

IP Multicast

IP Multicast broadcasts (pushes) a multimedia content over a network to a group of clients, in a one-to-many or many-to many communication between a source server and multiple clients. Due to its nature, in multicast there is no favorable circumstances to replace dropped or deformed packets, resulting in irreparable errors at the receiver side. This technology is bandwidth-conserving since it delivers a single data stream to multiple clients, with copies made only at the network routers where clients attach, reducing drastically the traffic in the network [23]. Internet Group Management Protocol (IGMP) is generally used in order to manage the delivery of the streams to a group of clients. This technology is highly complex and suffers from scalability issues since it requires routers that keep per-group state. Additionally, the deployment of IP Multicast is limited due to the fact that it needs routers and firewalls between networks to allow the transmission of the data packets with destination to multicast groups. These limitations severed the overall deployment and use of IP Multicast in the Global Internet, but it is nonetheless widely used in the managed distribution networks of IPTV service providers.

2.6 Mobile Operating Systems

A mobile operating system, or mobile OS, is an operating system specifically designed to operate on devices such as smartphones, tablets or devices provided with mobile communications and User Interaction interfaces. It is the operating system that allows running applications on end-user devices. Modern mobile OSs integrate and associate multiple features known and available in desktop computers OSs but modified to support hand-held touch screen interfaces.

Apple iOS

Apple iOS is an operating system developed by Apple Inc. for their iDevices, starting with the iPhone and the iPod touch platforms in 2007 and later expanded to the iPad and Apple TV families. iOS controls the hardware of the iDevices and supplies the necessary technologies for implemen-

tation of the native applications. iOS can be seen as the mobile (embedded) version of Apple's OS X operating system, a Berkeley Unix system, since iOS core was based on it and both share the Darwin foundation [24]. Apple adopted a layered architecture for iOS where four abstraction levels can be found: the Core OS layer, the Core Services Layer, the Media Layer and the Cocoa Touch layer, as illustrated in Figure 2.5. On this architecture, the lower layers present all the root services and technologies while higher layers hold more advanced services and capabilities. Applications exceptionally communicate with the elementary hardware meaning that the applications are highly protected from hardware changes since the communication between application and hardware components is made through a group of specific system/Application Program Interfaces (APIs), turning easier to design and create applications for iDevices with different hardware capabilities [25]. When iOS was first presented, it already supported the concept of direct manipu-

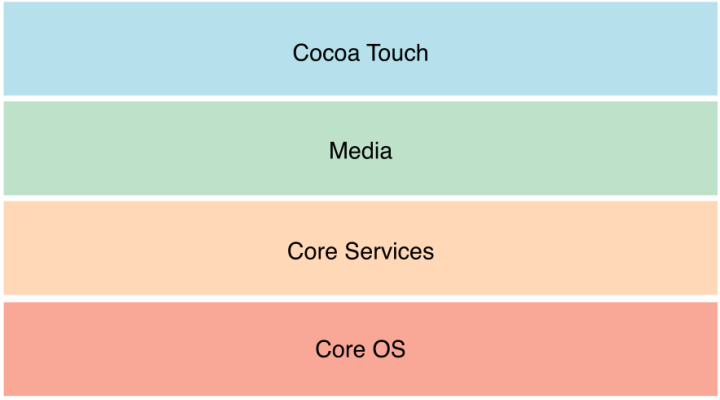


Figure 2.5: iOS layer architecture

lation for its human interface by multi-touch gestures such as pinching and reverse pinching, swipe, tap, among others, all defined on the iOS environment. Additionally, iDevices integrate a series of sensors such as gyroscopes, proximity, light, accelerometers, that can provide sophisticated experiences such as the rotation of the image according to the device's orientation or responses to shaking actions. The gestures on iOS are complemented by common soft interface elements like buttons, sliders, switches and many others.

Google Android OS

Android OS is Google's operating system designed with the main focus on mobile devices such as smartphones and tablet computers. It is an embedded Linux-based operating system and it was first developed by Android Inc. who was later purchased by Google. Android OS was first presented in 2007, shortly after iOS release, and was introduced along the first Android-powered smartphone in the market in 2008. Contrary to Apple's operating system, Google released Android OS code under the Apache License, making it open-source, therefore promoting a free access to

its source code. This licensing mode conceded the code to be modified and distributed freely by developers as well as manufacturers and other interested entities and resulted in a large developing community which build applications that extend the devices functionalities. Due to its huge

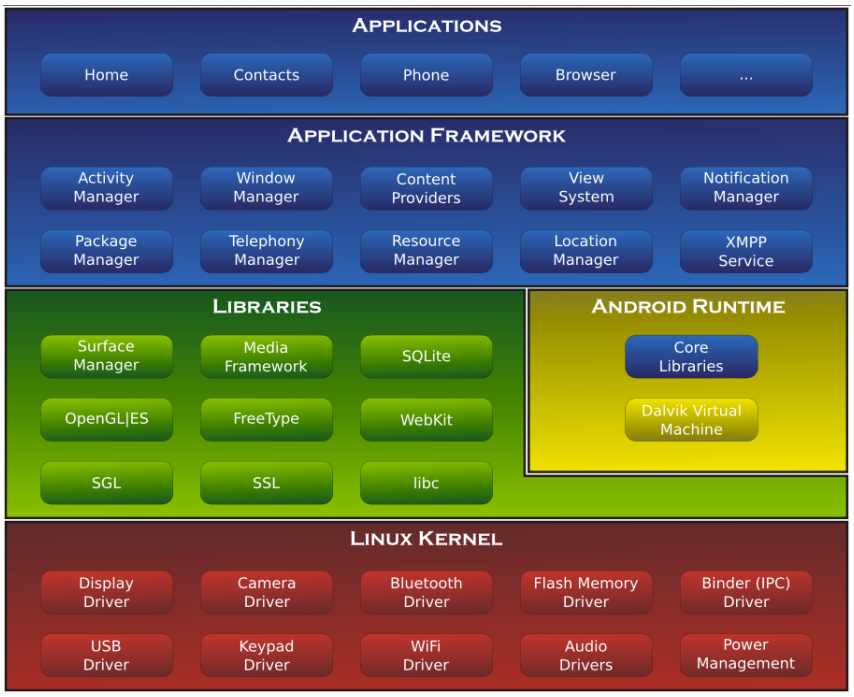


Figure 2.6: Android architecture

success and despite its main design for smartphones and tablet, Android has been adapted for a multitude of devices such as consoles, low consumption computers, televisions and many other devices [26]. Google also adopted the same concept of direct manipulation, as in Apple's iOS, which uses touch-screen inputs such as swiping, pinching, tapping and many others. Android OS also uses sensors from the devices it is installed to provide actions such as rotating and adjusting the image on screen to the device's orientation. The Android OS architecture consists of a stack of different layers, as illustrated in Figure 2.6. Each layer contains different programming components and consist mainly of the Linux Kernel layer, the Libraries layer, the Android Runtime layer, the Application Framework layer and the Applications layer.

In recent years Android OS equipped devices have been victims of several security issues. There has been an exponential growth in malware attacks, like Short Message Service (SMS) Trojans, backdoors and spyware, increasing the need for developers to create more secure applications to protect the end-users against the multiple threats.

2.7 Media Technologies

To develop a solution for operating systems such as iOS or Android, it is necessary to study and understand which media formats are supported on each of these systems. It is also important to understand the recommended specifications to better create an adequate solution. In this section, the media technologies supported on both iOS and Android OS will be analyzed for a better understanding of the respective requirements.

Multimedia support in Apple iOS

iOS was designed with the support of high-quality multimedia contents, providing an enhanced high-fidelity audio experience as well as the capability to decode, render and display HD video formats, whether from local files or by streaming the contents over a network either on-demand or in Live mode. iOS also provides multiple technologies to play and capture video contents. These video technologies in iOS support the playback of video files with .mov, .mp4, .m4v, and .3gp filename extensions. The compression standards recommended for audio and video are described in Table 2.2.

Table 2.2: iOS Recommended Compression Standards

Compression Standards					
Video Codec	H.264	H.264	H.264	MPEG-4	M-JPEG
Profiles	High@L4.0 Main@L4.0 Baseline@L3.0	Low-Complexity Baseline	Baseline $\leq$ @L1.3	Simple	
Video resolution	1920x1080 px	640x480 px	320x240 px	640x480 px	1280x720 px
Video frame rate	30 fps	30 fps	30 fps	30 fps	30 fps
Video bitrate	25 Mpbs	1.5 Mbps	768 kbps	2.5 Mbps	35 Mbps
Audio codec	AAC-LC	AAC-LC	AAC-LC	AAC-LC	ulaw
Audio channel	2 (stereo)	2 (stereo)	2 (stereo)	2 (stereo)	2 (stereo)
Audio bitrate	160 Kbps	160 Kbps	160 Kbps	160 Kbps	

Multimedia support in Android OS

Android OS supports multiple common media types into the applications and is able to play audio and video from media files stored in the device or from data streamed over a network connection. The list of media types follows Android Developers Center recommendations. This means that some devices may support additional formats but due to the huge fragmentation (i.e., diversity of hardware configurations) existing on the devices supporting Android OS the list follows the recommendations from the developers guide. The compression standards for audio and video are described in Table 2.3

Table 2.3: Android OS Recommended Compression Standards

Compression Standards			
Video codec	H.264 Baseline Profile	H.264 Baseline Profile	H.264 Baseline Profile
Video resolution	176x144 px	480x360 px	1280x720 px
Video frame rate	12 fps	30 fps	30 fps
Video bitrate	56 kbps	500kbps	2Mbps
Audio codec	AAC-LC	AAC-LC	AAC-LC
Audio channel	1 (mono)	2 (stereo)	2 (stereo)
Audio bitrate	24 Kbps	128 Kbps	192 Kbps

2.8 Content Distribution Networks

While P2P streaming solutions such as Skype[7], BBC iPlayer[8] or PPLive[9] have increased in popularity, offering inexpensive solutions for multimedia content distribution, there are other platforms for content distribution with high availability and performance, known as CDNs. A CDN is a system consisting on multiple interconnected computer nodes on the Internet with the purpose of distributing media content to a very large number of users. This distribution method is normally achieved by replicating the original content to various strategically located surrogate server nodes, typically housed in Internet Service Providers (ISPs) data centres, to serve end-users according to their proximity, offloading the source servers of the content providers.

The content redundancy provided by CDNs also offers a fail-safe protection level by disabling multiple servers during mass-scale attacks or in cases of deterioration of communications in the Internet. This will ensure that at least some content will remain available to end-users in unaffected parts of the network. Due to their nature, these networks offer a reinforced storage capacity as well as an enhanced data backup being profitable to enterprises as well as regular clients who depend on online storage and backup services.

2.9 Cloud Computing

With the success of services like Dropbox[10], Apple iCloud[11], Amazon Elastic Cloud[12], Microsoft Windows Azure[13] or Google Drive[14], Cloud Computing (CC) has become a trendy subject. According to [27], CC is a model that enables ubiquitous, convenient and on-demand network access to a group of non-homogeneous computing resources, such as servers, storage, applications and services, with the ability to release and supply these computing resources not just rapidly and with a minimum managerial

[7] http://www.skype.com
[8] http://www.bbc.co.uk/iplayer/radio
[9] http://www.pptv.com
[10] https://www.dropbox.com/home
[11] https://www.icloud.com
[12] http://aws.amazon.com/ec2/
[13] http://www.windowsazure.com/en-us/
[14] http://drive.google.com

effort, but also with a great independence from ISPs, as illustrated in Figure 2.7.

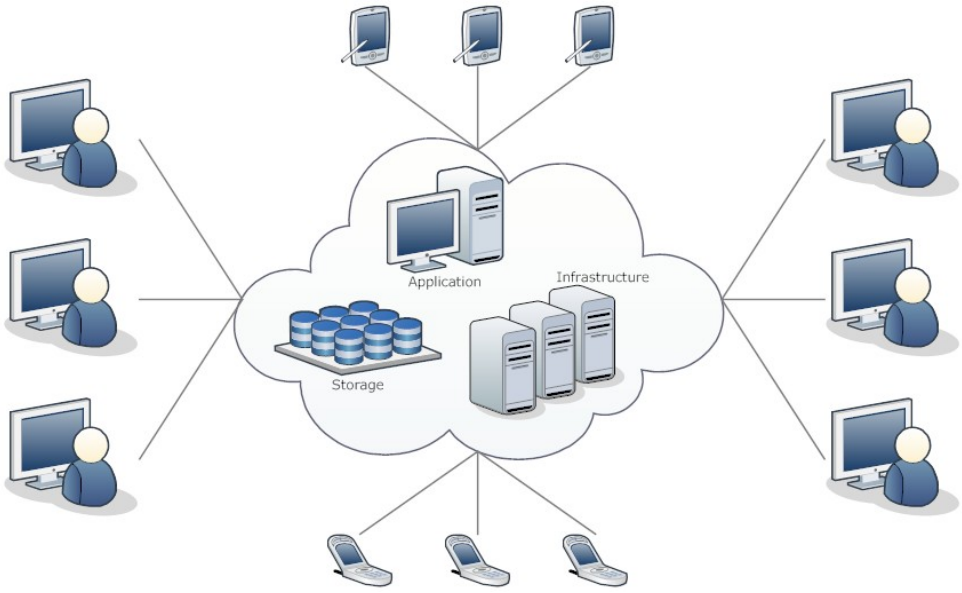


Figure 2.7: Example of a Cloud Computing Architecture

Although CC is a recent concept, it corresponds to an evolution of existing groups of technologies, integrating them in an orchestrated manner. CC has already changed the way ITs are used by allowing end-users or enterprises to take advantage of the its powerful resources, but used over the common Internet. An popular definition for CC is of a shared use of remote accessible IT resources over a communications network. This concept is commonly applied for services used over the Internet with users able to interact with these services through web interfaces or lightweight local applications. The main characteristics of CC is the on-demand self-service, providing computational capabilities without requiring human intervention, as well as access to large networks (typically worldwide), ready to be used from standard mechanisms over the Internet. It additionally counts with fast elasticity and a measured service, since the capabilities can be elastically provided and released, even automatically, controlled and optimised by weighting those capabilities against the requirements of the service. The most common service models of CC are the Software as a Service (SaaS), the Platform as a Service (PaaS), and the Infrastructure as a Service (IaaS):

- SaaS consist on a software delivery model in which the content data is stored on a cloud and clients access it using the provider's interfaces. Usually these interfaces are available on multiple platforms and devices in the form of desktop or browser applications, and does not includes capabilities for management and control of the infrastructure, which are exclusive tools for the provider of the application/software.

- PaaS offers a computing platform where users can distribute their created or earned software using

programming languages or tools from the provider. This solution normally let the users control the deployment methods.

- IaaS provides remote computing resources allowing users to run and control their own software in those resources. In this case, the provider is just responsible for the management and control of the infrastructure equipment.

The deployment models for CC comprise the **Private Cloud model**, the **Hybrid Cloud model**, the **Public Cloud model** and the **Community Cloud model**. The Private model corresponds to the use of a cloud service by a single organisation, with the cloud service either managed by the organisation itself or by a third party company. In the Hybrid model the service is a composition of multiple clouds (remaining as independent entities) connected through a proprietary or standardised technology. The Public model offers infrastructures to the general public, normally available for free and generally operated and controlled by the providers. The Community model proposes a cloud infrastructure to be used by a community whose mission is shared by its users, being the infrastructure managed and hosted either by the community or by third party entities [27–29].

2.10 Related Work

Research proposals or solutions combining the technologies described in this chapter for a Cloud-assisted mobile streaming environment supporting adaptive and scalable video are scarce. Some proposals combine video scalability and adaptability but relying on active control from the serving side, typically taking "hints" of transmission status provided by mobile clients to allocate suitable video streams. Other works base their cloud-assisted video delivery strategy on network coding for the distribution of scalable video streams in multiple routing paths. Other authors addressed these topics but focused their efforts in different aspects:

In [30], the work proposes a quality-oriented scalable video delivery using SVC, but it was only tested in simulated Long Term Evolution (LTE) networks.

In [31] the solution proposes a cloud-based SVC proxy to deliver high-quality media streaming but does not considers mobile networks and end-user devices.

The closest approach to the CASHED concept described in this thesis, is AMES-Cloud [32, 33], an adaptive mobile video streaming and sharing framework that stores the video on a cloud (VC) and uses the computing resources of the cloud to build private agents (subVC) to each mobile user. The AMES-Cloud solution is also based on SVC techniques to adapt the streams to the fluctuations of link quality. Additionally, AMES-Cloud tries to offer a "non-buffering" streaming video experience by background prefetching based on the the synergy of mobile users in their Social networking service (SNS).

3

CASHED Design

Contents

3.1 CASHED Architecture Design Requirements 28

3.2 CASHED System Overview 29

3.3 Content Distribution System 32

3.4 Client Application 34

Over the past few years, despite the efforts of mobile network operators to enhance the wireless link bandwidth, the soaring video traffic demand from mobile users is rapidly crushing the link capacity.

Multimedia streaming refers to multimedia content delivery from a “serving source”, that compresses and injects the contents into the network, as an ordered sequence of data to be consumed by “client” end-user devices through the Internet. The “clients” receive the ordered data to feed a suitable Media Player where the media content is decoded and rendered in real-time.

But for high-quality multimedia streaming, extensive bandwidth is usually required, posing serious constraints to mobile handheld devices due to fluctuations in link conditions caused by multi-path fading and user mobility.

Client-server streaming media architectures, where serving nodes need to to provide the service to each and every client, particularly in Video On Demand (VoD) scenarios, might be expensive if the distribution is aimed for a wide geographical range. But a system that modifies the client-server paradigm to allow streaming the contents but with cloud-computing elasticity and computing capabilities, promises to be a more scalable and economical solution than the traditional model.

It is then vital to improve the service quality of mobile video streaming while using the networking and computing resources efficiently. Two main aspects have been the focus of many studies on how to improve the service quality of mobile streaming services: **Scalability** and **Adaptability**. To address scalability, the Scalable Video Coding (H.264/SVC) compression standard combines a compatible Advanced Video Coding (H.264/AVC) base layer with multiple enhancing layers (for spatial, temporal and fidelity scalability). To address adaptability, the media bit-rate needs to be adjusted during the streaming process to the existing time-varying link bandwidth of each mobile user, minimizing packet losses and bandwidth waste.

Technologies or solutions combining the possibilities offered by cloud-computing and SVC, to offer a high quality adaptive and scalable media streaming solution to mobile end-user devices, are scarce.

For the above reasons, the work developed for this thesis was concentrated on the definition and development of a streaming media architecture, cloud-based, with simple network construction and maintenance mechanisms, able to deliver high-quality scalable contents, and on the development of the correspondent “smart” client application for mobile handheld networked devices, able to consume the streaming media contents. This chapter describes the design of the CASHED architecture, detailing its main components.

The CASHED concept is suitable for iOS and Android OS devices. However, the development efforts in this work, due to time constraints, were focused on iOS devices.

3.1 CASHED Architecture Design Requirements

The main goal for this work was to create a system, capable of supporting adaptive streaming mechanisms based on H.264/SVC media contents. The following set of functional requirements were considered for the solution:

Web-streaming: The client application should support streaming media using HTTP protocols.

Multi-source streaming: The client application should support multi-source streaming media, i.e., "simultaneous" streaming of media content components from a network, supported/complemented by CDN/CC services.

Support content Metadata Description: The client application should support content metadata description in a format similar or compliant with MPEG DASH [12].

Scalable and Adaptive Media Contents: The system should support on-demand streaming of scalable and adaptive contents based on SVC.

Heterogenous End-User Devices: The client application should be compatible with current and future generations of end-user devices form factors, irrespective of their performance, screen size and resolution.

Access Network independency: The solution should provide the expected service over different types of access networks supported by the end-user devices, such as Wireless Local Area Networks (LANs) (IEEE 802.11) or cellular data networks such as General Packet Radio Service (GPRS), Universal Mobile Telecommunication System (UMTS), LTE, etc.

Some vital non-functional requirements were also considered in order to achieve the aforementioned functional requirements:

- **Maximize the streaming quality to satisfy the end-user's expectations:** The application should be able to always present an adequate media content quality, adapted to the environment conditions of end-user device, with smooth and seamless quality variations, maximizing the end-user perceived quality.
- **Scalability and Reliability:** The solution should present a (desirably) high degree of functional reliability and stability, use the hardware and network conditions to determine the most appropriate configurations to deliver its functions, and should be scalable for large number of end-users.

To satisfy the aforementioned requirements, the architecture was designed using a modular approach for both the distribution and the client sides.

3.2 CASHED System Overview

The CASHED architecture considers a cloud-based Serving Platform and end-user device Client Application. The Serving Platform consist of a PaaS cloud, providing content transformation and distribution, i.e., capable of transcoding H.264/SVC video formats which are then segmented for distribution over HTTP.

The Client Application in end-user device nodes is capable of consuming the transformed media content from the Serving Platform.

For the distribution, the streaming process uses a segment transfer approach where the original media is cut in multiple smaller segments of a short duration. This approach allows for a distribution method appropriate to the Internet infrastructure.

The main components of the CASHED Serving Platform and Client Application are illustrated in Figure 3.1. For simplicity the underlying infrastructure for the PaaS and the iOS device, are not shown in the figure.

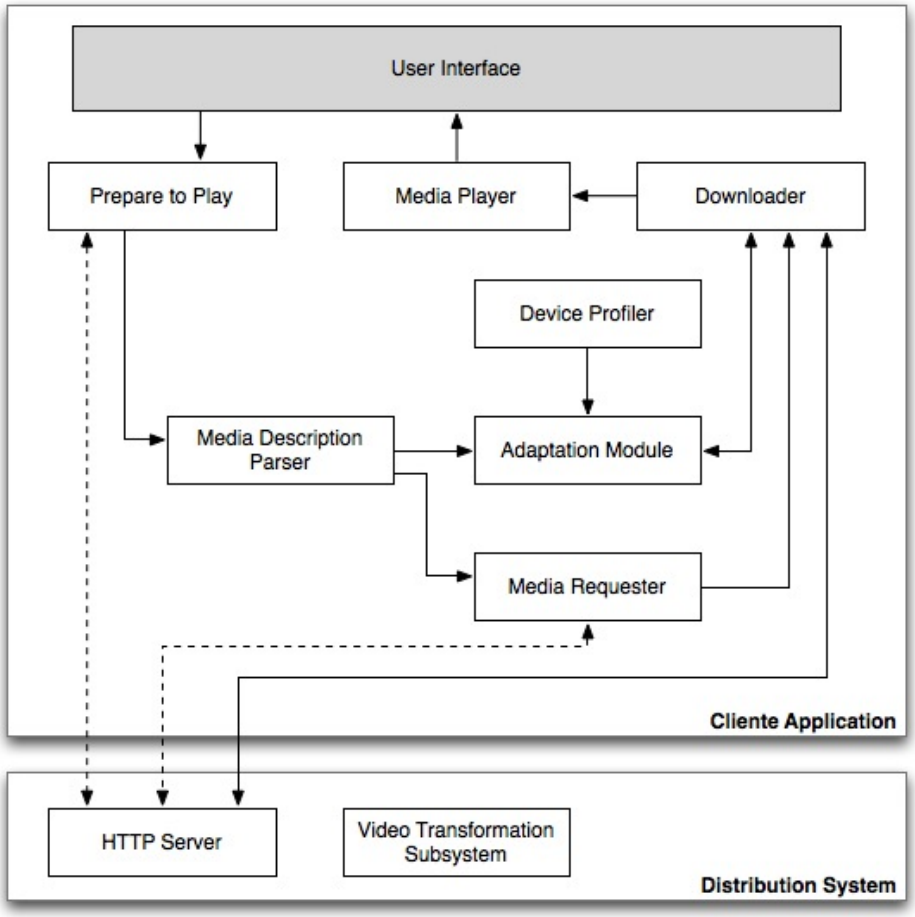


Figure 3.1: CASHED Components Architecture

In Figure 3.1 the PaaS Content Distribution System components are essentially the HTTP Server and the Video Transformation Subsystem:

HTTP Server: Responsible for distributing the media content over HTTP protocols.

Video Transformation Subsystem: Transforms the media for distribution, composed by:

- **Transcoder:** Responsible for transcoding the H.264/SVC file into H.264/AVC.
- **Segmenter:** Responsible for splitting the H.264/AVC media into transport segments, and create a m3u8 playlist.

Request interpreter: Responsible for resolving the requests from the Client Application.

In the same Figure 3.1 the main Client Application components are also illustrated, but essentially correspond to the Media Access components (Media Description Parser, Media Requester and Downloader), the Adaptation Module, the Device Profiler, the Media Player/Decoder and the User Interface (UI):

Media Access: Responsible for all actions related with access to the streaming media.

- **Media Description Parser:** Extracts the information from the MPD of the content requested over HTTP. The information extracted is used to build the downloading process of the media content.
- **Media Requester:** Provides the adequate methods for HTTP media components requests.
- **Downloader:** This component controls the download of the correct media segments and includes the methods to request the segments over HTTP.

Adaptation Module: The Adaptation Modules is in charge of deciding which segments need to be downloaded according to a set of heuristics.

Device Profiler: This module is in charge of analysing the device where the application is running. It extracts information such as the model of the device, its capabilities, the network being used and assists the Adaptation Module in the H.264/SVC quality level decision process.

Media Player: Decodes and plays the downloaded media.

User Interface: Responsible for the Human Interaction, bringing input and output for the application (supporting Trick Functions like Play, Pause, Timeline slider) but also displaying the decoded video.

3.2.1 Media Presentation Description

The MPD file includes information about the SVC media to be played representing the structure of the media, such as the number of segments, identification of the content, codecs, quality levels, video resolution, etc.

This file is a well formed XML document, meaning that it does not use a Document Type Definition (DTD), and has a format similar to MPEG DASH [12]. In its structure, it includes as Root Element a <StreamInfo> field enclosing all the metadata necessary to describe the content. This media information is grouped in <Clip> fields with <Representation> elements describing each component of the media and other metadata, as illustrated in Figure 3.1.

Listing 3.1: Example of a MPD file.

```
<?xml version="1.0" encoding="UTF-8"?>
<StreamInfo version="2.0">
    <Clip duration="PT01M0.00S">
        <BaseURL>videos/</BaseURL>
        <Description>svc_1</Description>
        <Representation mimeType="video/SVC" codecs="svc" frameRate="30.00" bandwidth="401.90"
            width="176" height="144" id="L0">
            <BaseURL>svc_1/</BaseURL>
            <SegmentInfo from="0" to="11" duration="PT5.00S">
                <BaseURL>svc_1-L0-</BaseURL>
            </SegmentInfo>
        </Representation>
        <Representation mimeType="video/SVC" codecs="svc" frameRate="30.00" bandwidth="1322.60"
            width="352" height="288" id="L1">
            <BaseURL>svc_1/</BaseURL>
            <SegmentInfo from="0" to="11" duration="PT5.00S">
                <BaseURL>svc_1-L1-</BaseURL>
            </SegmentInfo>
        </Representation>
    </Clip>
</StreamInfo>
```

3.2.2 Streaming Operation

The typical operation for watching a streaming program starts with the respective selection of the media on the CASHED Client Application UI. The selection triggers the retrieval of the content MPD into the end-user terminal after which a streaming session starts (Figure 3.2). Depending on the information contained in the MPD, the Requester module of the Client Application will send to the Content Distribution System a specific request with the necessary information to trigger the media transformation process for the requested content.

The Content Distribution System start the transformation process for a default initial sliding window of a small range of segments (typically, 10s duration, corresponding, for example, to 5 media segments of 2s duration each), and replies to the Client Application with the URL of the generated initial content Playlist (listing and describing the media segments URLs). This initial window may consist of base level quality segments, unless the Client Application already provides a quality level "hint" with the initial request, in which case the transformed first segments would already correspond to the "desired" quality.

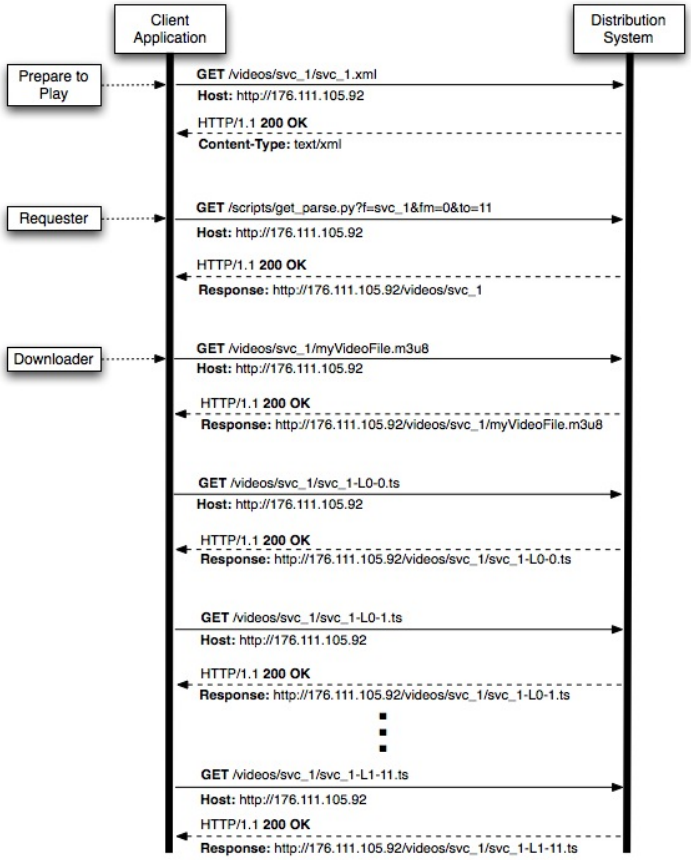


Figure 3.2: CASHED streaming message sequence

When the Client Application receives the response, the download process is triggered by loading the initial Playlist and the Adaptation Module will work with the Download module to tune the quality level of the subsequent media segments to request, as depicted in the message sequence chart of Figure 3.2.

The difference between the MPD and the Playlist file is easy to understand since the MPD file describes the media components structured in SVC and the Playlist file is a simple list of the most recent media segments generated, ready to be retrieved by the client.

3.3 Content Distribution System

The Content Distribution System prepares the media content to be partitioned, segmented and indexed for the distribution, interprets the requests from clients, and distributes the contents.

The transformation process, takes the original H.264/SVC encoded file or stream and transcodes it into H.264/AVC media which is then split into several sequences of segments (media transport files).

3.3.1 Video Transformation Subsystem

The Video Transformation Subsystem is composed by the Transcoder and the Segmenter modules, as illustrated in Figure 3.3.

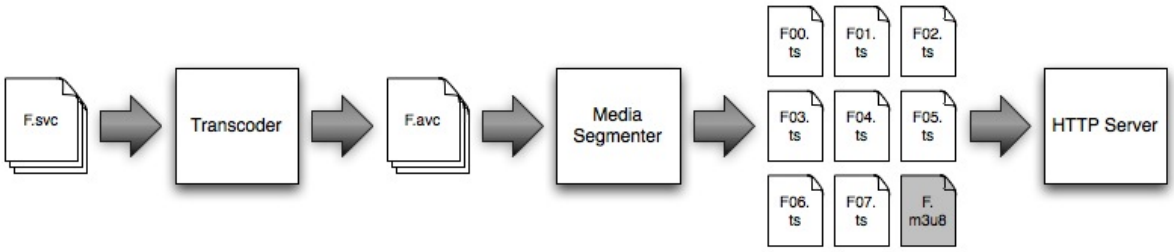


Figure 3.3: Distribution Side - Video Transcode and Partitioning

The task of the **Transcoder** is to make a bitstream reordering conversion of the H.264/SVC encoded media in order to feed a SVC-to-AVC JSVM-rewriter process to obtain a H.264/AVC format with the desired quality level. This works by converting all layer representations (increasing order of Direct Quality ID (DQId)). For this to work, the SVC video needs to be encoded with a special inter layer prediction flag with the value equal to one.

The resulting H.264/AVC converted (and multiplexed with audio) to a MPEG-2 transport stream feeds the **Segmenter** module that partitions it into file segments of relative equal duration for distribution over HTTP. The **Segmenter** also creates a Playlist (an index) file containing references to the individual media segments (relative URLs).

3.3.2 Playlist File

The Playlist file is saved as a .m3u8 playlist, which is a format extension of the .m3u format used for MP3 playlists. The creation of this file is important since it contains the information necessary for the play-out session. Each entry of the file encloses a specification that can be of the following types (example in Listing 3.2):

- Local pathname relative to the m3u file location;
- Absolute local pathname of the files to play;
- An URL of the file to play;

Listing 3.2: Example of a Playlist file for the play-out session.

```
#EXT−X−VERSION:3
#EXTM3U
#EXT−X−TARGETDURATION:10
#EXT−X−MEDIA−SEQUENCE:1
#EXTINF:10.0,
http://media.example.com/segment1.ts
#EXTINF:9.5,
http://media.example.com/segment2.ts
#EXT−X−ENDLIST
```

It is important to refer that the Playlist file generated in the **Segmenter** module of the Content Distribution System is similar to the Playlist of an HLS session, but with a single streaming set instead of an "adaptation set" of alternative streams.

This file is necessary for the Media Player to interpret which media segments it will decode and play. It is also important to refer that the Playlist for each segment group (depending on segment duration and corresponding buffer size) has a EXT-X-MEDIA-SEQUENCE element, informing that the Playlist will be updated at the serving side at the time the Playlist was loaded plus the duration of the Playlist. The Media Player will therefore know when it should re-load the refreshed Playlist during the streaming session.

3.4 Client Application

The Client Application is a very simple iOS App, in terms of UI, incorporating the "smart" adaptation mechanisms that decide on the level of quality of the media stream at any moment.

3.4.1 Adaptation Module

The **Adaptation Module** is where the decisions are made about the quality of the video segments to be downloaded. This module decides the video quality adjustments according to a set of heuristics related to the device model (capabilities) and the network conditions.

The device heuristics ensure that the device is capable of playing (decode and render) videos up to certain quality levels, i.e., up to the supported video resolution, bitrate and framerate.

The network heuristics ensure that there is always enough available bandwidth to download the required media segments. For each segment downloaded, the bandwidth used is checked, to predict that the Media Player does not stalls and the viewing experience is smooth. While the system downloads each segment, its size and the corresponding download completion time are stored and used to estimate the available bandwidth for the next segments.

Figure 3.4 illustrates the Adaptation Module workflow, with the decision heuristics described in Algorithm 3.1 and Algorithm 3.2.

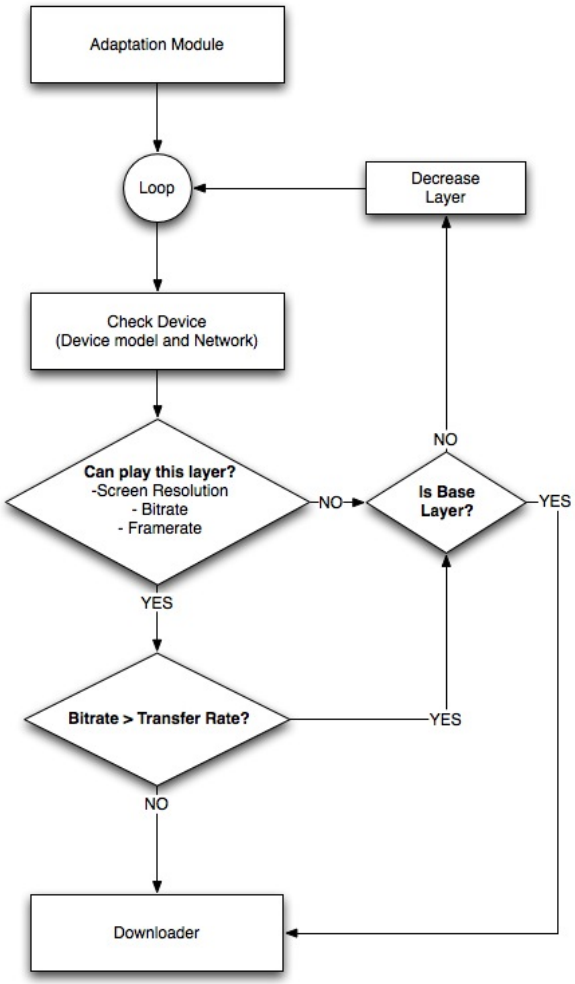


Figure 3.4: Adaptation Controller

The Adaptation Module starts by verifying the device profile and the type of network it is connected. With the information from the profiler, the module requests the MPD parser information on the content and, starting with the highest quality of the media, verifies if the current device is capable of reproducing the content with such properties. If it is not capable then the algorithm will repeat this step with a lower quality level. If the device is capable of supporting that quality level, then it will continue by verifying if there is enough bandwidth available for downloading, in time, the corresponding segments by comparing the current connection transfer rate with the media bitrate for that quality level. If the test fails the process starts again but with an even lower quality level. If both steps pass, then the decision is made for the quality level and the download is triggered. If tests are inconclusive, the algorithm decides for the base layer. The transfer rate is always estimated after each segment is successfully downloaded. As the first file of the media content to be downloaded is always the Playlist, that file is the one used to start the estimation process using Algorithm3.2.

Algorithm 3.1: Quality Level Estimation

```
if ([profiler checkDevice] isEqualTo "Device in SVC Profile") AND
([profiler checkForNetwork] isEqualToString "Network in SVC Profile") then
    while numLayer >= 0 do
        if [parser getWidth] > "width in SVCProfile" OR
        [parser getHeight] > "height in SVCProfile" OR
        [parser bitRate] > "bitrate in SVCProfile" OR
        [parser frameRate] > "framerate in SVCProfile" then
        LOG: This layer's properties are not supported on this device.
        numLayer - -
        else if media bitRate > transferRate then
            LOG: Not enough bandwidth
            numLayer - -
        else
            return numLayer

        if numLayer == 0 then
            return L0;
```

Algorithm 3.2: Transfer Rate Estimation [in kbps]

$rate = (file\ size)/(currentTime - transferStartTime)$
$transferRate = (rate * 8)/1024$

3.4.2 Media Requester

The **Media Requester** module is responsible for the requests to the CC platform of the MPD, for the interpretation of the extracted media description, and therefore, for triggering the streaming process.

3.4.3 Media Description Parser

The **Media Description Parser** is the component responsible for parsing and validating the XML file that contains the information to configure the media streaming process.

3.4.4 Downloader

The **Downloader** is in charge of requesting and downloading over HTTP the correct media segments according to the information given by the **Adaptation Module**.

The **Downloader** also informs the Media player when there are enough segments in buffer to start the play-out session. For each future segment, **Downloader** queries the **Adaptation Module** on the desired quality level for next subsequents segments to request to the CC platform.

3.4.5 Device Profiler

This **Device Profiler** is where the heuristics about the device are extracted. Information such as the model of the device and the type of network in use are obtained from this module and made available to the Adaptation Module. The Adaptation Module can then proceed with its calculations.

3.4.6 Media Player and User Interface

The **Media Player** is responsible for decoding the media segments received in H.264/AVC format and presenting render the media components to the output devices (display, speakers, etc.).

The **Media Player** also allows the interaction and control of simple Trick Functions, by the means of a "controller" that receives all the user inputs.

The **User Interface** is the application front-end. It is designed with two main components: a video renderer and video controls (Figure 3.5).

Figure 3.5: User Interface

- The renderer offers the interface where the player presents and renders the decoded video;
- The video controls offer simples Trick Functions, Time seeking and Video renderer resize.

4

Implementation

Contents

4.1 Development Process 41

4.2 Development Environment 42

4.3 CASHED Client Application 42

4.4 CASHED Distribution System 46

This chapter describes the implementation of the CASHED prototype. The description includes the process, a short review of the tools, programming languages and libraries used, as well as a description of the solutions for building the different components and the development environment.

4.1 Development Process

For the development of the CASHED prototype, several stages were followed:

- Technology Research and Related Works
- Requirements Gathering and Study
- Design of the Architecture
- Implementation Process
- Testing and Functional Validation

When developing the prototype it was important to maintain the maximum compatibility and portability with multiple devices (e.g. iPhone, iPad, iPod).

The iOS architecture turns easy to demonstrate and test the portability of the prototype in multiple devices since it allows scaling the presentation layer to the screen size of the device without the need to change the core of the application. The only requirement needed for the prototype is to maintain the same version of the OS on all the devices since some functionalities and libraries used may not exist in older versions.

For the implementation of CASHED Client Application the development efforts stayed with the iPhone form factor, in order to test the solution in devices with multiple screen sizes and processing capabilities,:

- iPhone 4/4s - Screen 3.5”
- iPhone 5/5c/5s - Screen 4”

On the Distribution System side the focus of development stayed on a functional transcoding and partitioning solution, allowing to obtain streamable content.

The software used for the H.264/SVC to H.264/AVC transcoding was the JSVM Reference Software [34], without any optimisations.

4.2 Development Environment

The prototype was developed on the most updated version of Apple OS X (version 10.9.2 - Mavericks). The programming language used was Objective-C since it is the official and supported language for iOS development.

On the cloud side, the system was built using shell scripts in Linux. The prototype Client Application was tested using Apple's iOS simulator and real iPhone Apple devices models 4/4s and 5/5S.

4.3 CASHED Client Application

The development using Apple's Objective-c and Xcode IDE, turned possible to use the advantages of object oriented programming and to build each component of the solution as a class.

4.3.1 User Interface

Figure 4.1(b) and Figure 4.1(a) illustrate the iOS graphical UI designed for the prototype. The UI is divided in two windows: the initial window, where the desired media is loaded and the player window where the media is played. The initial window is composed by a text box where the name of the video to load is entered, a play button to launch the play-out session, and a status bar displaying information about the loading status. The player window is divided in three sections, with the menu bar on top, the video renderer in the middle and the video controls on the bottom.

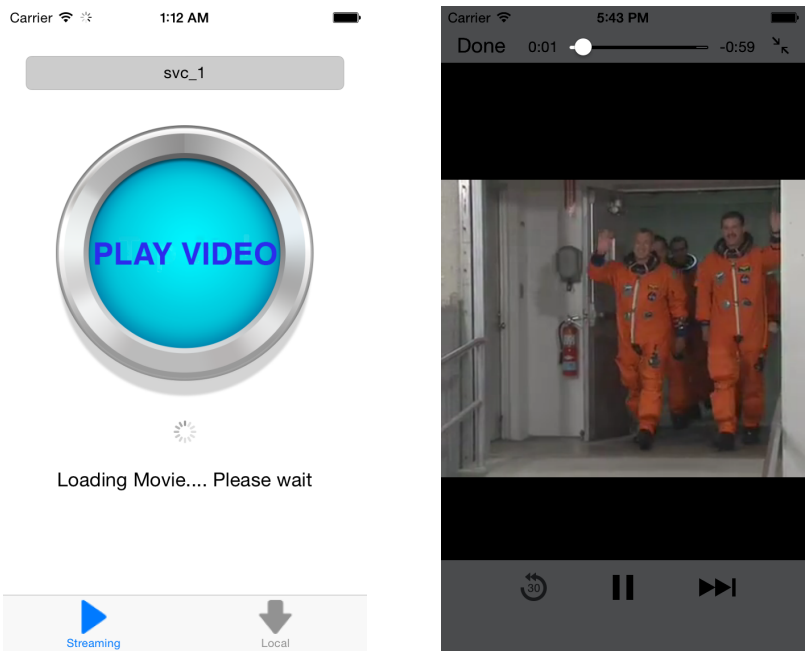
iOS Cocoa Touch Application UI: All application interfaces for iOS are designed with Apple Objective-C (native object-oriented programming language) using their API known as Cocoa Touch.

UI Controller: The UI controller in the CASHED prototype is an integral part of the AVFoundation's MPMoviePlayer class, which offers a set of multiple different controller layouts. The choice presented in Figure 4.1(b) shows the default layout. This choice was made because it filled the basic requirements for this prototype with simple and intuitive controls. The other options offered more complex layouts that would only increase the complexity of the prototype development.

4.3.2 Media Player/Decoder

Up to the latest version (version 7) of iOS, there is no implementation of decoder nor native support for H.264/SVC on iOS devices.

For the prototype solution the H.264/SVC content had to be transcoded into a supported format, the H.264/AVC. The Media Player in the application was build using subclass of Apple's Cocoa-Touch AVFoundation, called MPMoviePlayer which offers a set of options for the play controllers.



(a) Media Loading Window **(b)** Play-out Session UI

Figure 4.1: Complete User Interface

4.3.2.A Internal mini HTTP Server

Due to Apple's standardisation of the AVFoundation, some restrictions where found in the way the media content is loaded.

According to the specifications for their MPMoviePlayerController, there are essentially two methods to feed the Media Player:

- Initialize it with the URL of a movie file. This file can be located either in the App sandboxed folder or on a remote server.
- Initialize it with the URL of a Playlist file.

The problem with the first option is that it requires the movie to be a single file. Since the media is segmented in multiple segments, this would require the prototype application to merge the downloaded segments into a single file again. This is not viable since the application would need to download all the segments before merging them, leading to an undefined waiting time.

The second option also offers a problem, which consists in the way the Playlist file initialises the player. The MPMoviePlayer only reads Playlist files if served over HTTP. No special configuration is however required, apart from associating the MIME types of the files being served with their file extensions as shown in table 4.1.

The choice fall obviously on the second option, but requiring, in order to follow Apple's standards, a local mini HTTP server. This mini HTTP server is then responsible for feeding the Playlist file and the downloaded media content to the player. The Playlist file is the first to be served when there are enough segments downloaded in order to start the play-out session, while other segments are being downloaded.

Table 4.1: MIME Type Server Configuration

File Extension	MIME Type
.M3U8	application/x-mpegURL or vnd.apple.mpegURL
.ts	video/MP2T

4.3.3 Media Description Parser

The Media Description Parser is responsible for extracting information from a MPD file (an XML encoded document) served by the Distribution System. The module receives the URL of the XML file as the argument to download it and delegates methods to extract multiple different types of information, as illustrated in Figure 4.2.

4.3.4 Media Requester

The Media Requester is responsible for sending an HTTP request for the desired movie to the CC platform.

This HTTP request uses a GET Request method that includes in the resource information to engage the video transformation process.The request is responded with the URL of the Playlist file.

4.3.5 Downloader

The Downloader works with the Adaptation System to download the appropriate segments described in the Playlist. The Downloader starts by requesting the Playlist (from the respective URL sent by the Media Requester) and proceeds by requesting the listed segments. This HTTP request uses a GET Request method.

As soon as there are enough segments in buffer for a continuous play-out session, the Downloader notifies the Media Player to start decoding and rendering the media.

4.3.6 Adaptation System

This Adaptation System module is responsible for determining, at each moment, which quality level can be downloaded.

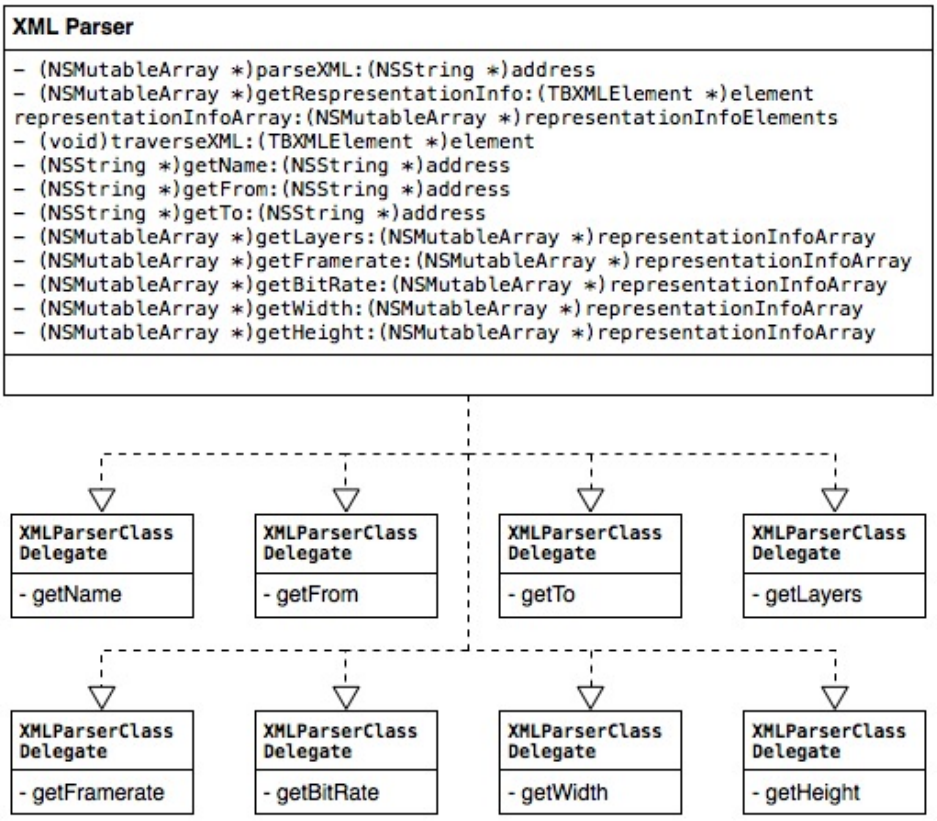


Figure 4.2: XML Parser Delegation

It was necessary to create profiles for the devices (Table 4.2) in order to help the Adaptation System to take the correct decisions.

Table 4.2: SVC Profiles

Network type	Device	Video Width (pixels)	Video Height (pixels)	Highest Bitrate (kbps)	Highest Framerate (kbps)
WiFi	iPhone 4/4s	720	576	12500	30
	iPhone 5/5c/5s	1920	1088	60000	30.1
	iPod Touch (5th gen.)	1920	1088	60000	30.1
LTE	iPhone 5/5c/5s	1280	720	20000	68.3
3G or lower	iPhone 4/4s	352	288	480	30
	iPhone 5/5c/5s	352	288	480	30

The Adaptation System checks with the Device Profiler which device is being used and the type of network the device is connected to. After obtaining this information, the Adaptation System works with the Media Description Parser to extract information on the media structure such as the picture size (Height, Width), Bitrate and Framerate, and uses this information to verify the quality levels that comply with the profile for the device.

4.4 CASHED Distribution System

The Distribution System for the prototype consists of a simple Linux Debian machine (version 6) running on a LunaCloud server with a public IP. To maximize the server response and communication, a bandwidth of of 2Gbps was allocated for this server.

4.4.1 SVC-to-AVC Rewriter

The SVC-to-AVC rewriter is responsible for transcoding the media content from H.264/SVC to H.264/AVC. This tool is part of the JSVM Reference Software and was used to build the media transformation script.

4.4.2 Media Segmenter

The Media Segmenter is responsible for segmenting the H.264/AVC media in MPEG-2 transport stream segment files, and to create a .m3u8 Playlist file containing the URLs addresses of the generated segments.

For the Video Transformation Subsystem, in order to obtain the segments and the Playlist file, the initial tool to be used was FFMpeg [35]. During the development process it was found that the generated segments using FFMpeg created issues (random jumps to previous frames, freezes, gaps) when chained for a play-out session. For this reason, and due to a tight time schedule for the development, the option fall on Apple's official segmenter.

4.4.3 Log File

The distribution system also implements a simple log mechanism, typically used for stored contents (VoD). This log mechanism serves to alert that the content as been already requested in the near past, being therefore still valid and in "cache", already transformed, fragmented and ready to be served, lowering considerably the startup time upon request from the Client Application.

For "Live" contents, it is assumed that the trigger is performed at the producer side, when the content is made available, therefore ready to be consumed from the latest generated "buffer window".

This mechanism was created due to some limitations in the H.264/SVC to H.264/AVC media transformation process related with real-time stream processing and the use of non-optimised JSVM code.

As such, the first request from a client triggers the transformation and the partitioning mechanisms for an initial sliding window of generated segments, the Playlist and the Log file, thus taking longer to send a response to the Client Application.

Further requests to that same media from the same or other Client are immediately served since the Log file indicates that the transformed media is already available and ready to be served.

In the original concept of the prototype, this mechanism was not necessary since each segment of the media would be transformed to the quality requested and served “on-the-fly”.

4.4.4 HTTP Server

In order to be able to response to CASHED Client Application requests, the Distribution System possesses an HTTP server configured using Apache 2.

Since all requests are GET Request methods with resources containing additional arguments, needed to trigger Shell Scripts in the Distribution System, a modification to the standard Apache was installed, called Mod_Python. This extension allows for Apache to take some arguments in an HTTP request and execute a set of scripts.

5

Evaluation

Contents

5.1 Evaluation Objectives and Experimental Scenarios 51

5.2 Evaluation Results 52

This chapter describes and presents the evaluation of the CASHED prototype system. A description of the evaluation objectives, metrics and experimental scenarios are also detailed.

5.1 Evaluation Objectives and Experimental Scenarios

In order to evaluate the behaviour and performance of the prototype, several tests where designed. These tests aimed to evaluate the behaviour under heterogenous access networks, the streaming capabilities and the adaptation mechanism.

To be able to correctly assess the solution, a set of experiments were prepared in a controlled environment.

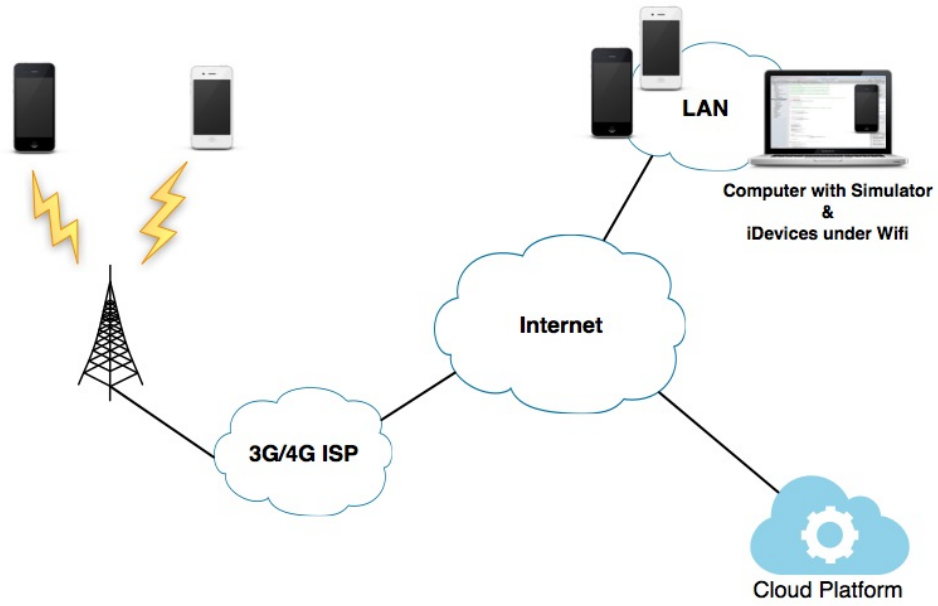


Figure 5.1: Test Environment

The test environment included a computer running an iOS simulator, an iPhone 4 and an iPhone 5s, providing a significant set of devices (oldest and latest iOS 7 supporting devices). In order to simulate multiple network condition, Apple's Network Link Conditioner was installed in the computer running the iOS simulator with the profiles described in Table 5.1 used in the tests. The Network Link Conditioner allows to force/simulate fluctuations in fixed network segments.

Table 5.1: Network Link Conditioner Profiles

Network Profile	Bandwidth	Packets Droped	Delay
Wifi	40 mbps	0%	1 ms
3G	780 kbps	0%	100 ms
Edge	240 kbps	0%	400 ms

The environment also considered a Cloud Platform with a standard web server, where the media encoded videos were transformed and made available to clients, along with the MPD and the generated Playlist file. The CC platform was is served by LunaCloud service with a public IP address.

The SVC video used was encoded with temporal scalability in two layers (for two quality levels). Table 5.2 describes the SVC layers used for each level after the segmentation.

Table 5.2: Layers properties

Layer	Bitrate	Framerate
0	401.90 kbps	30 fps
1	1322.60 kbps	30 fps

In order to proceed with the testing, some scenarios were considered using fixed network, mobile LTE network, mobile 3G network and a Wifi network. The set of tests was as follows:

- Network Streaming Performance
  - Analysis of the startup time, measuring the time the system takes to start streaming on multiple network types;
  - Analysis of the streaming performance with and without a Log file;
- Application Performance
  - Analysis of the adaptation system, measuring the quality adaptation for variations on the network load and type;
  - Analysis of the transition time between network types;
  - Overall analysis and benchmarks of the application performance;

Every test was executed at least ten times in order to obtain values with statistical significance.

5.2 Evaluation Results

The measurements taken with the Client Application simulator will serve as reference measurements since they where executed at the Instituto Superior Tecnico (IST) facilities, connected via Ethernet through a full Gigabit network. This condition makes available bandwidth that largely exceeds the necessary minimum for a sustained high-quality video streaming session.

5.2.1 Startup Time

The graphics in the following figures display a comparison of the startup time when using or not a Log file on the Distribution System. This Log file was created as a mechanism to alert that the video as been previously transformed and fragmented and was ready to be served.

Scenario with Terminals connected over a Wifi network: Figure 5.2, shows, as expected, the the shortest startup time of all the devices, both with and without the log file, for tests running on the simulator present.

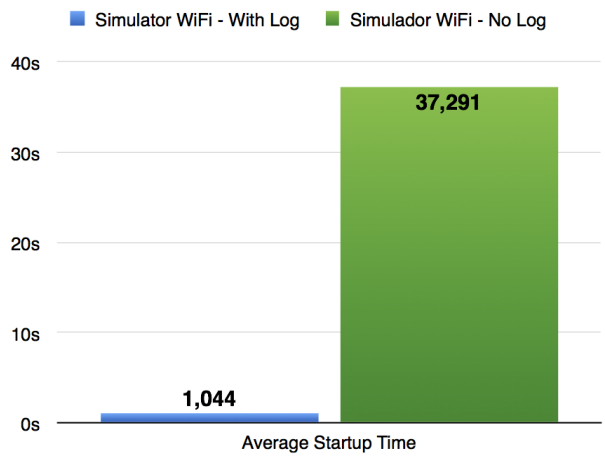
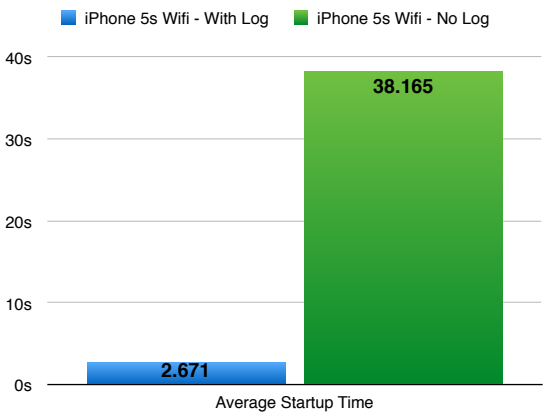
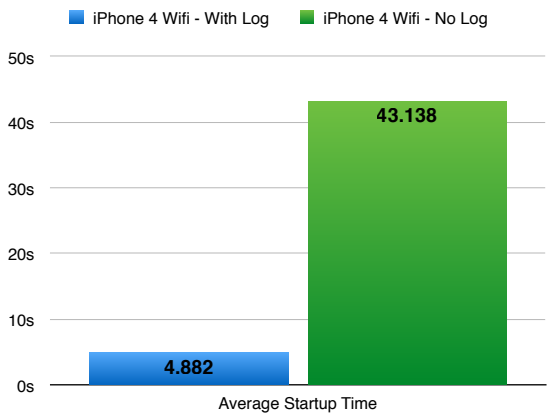


Figure 5.2: Startup Time - Simulator Wifi (Lower is better)



(a) Startup Time - iPhone 5s Wifi



(b) Startup Time - iPhone 4 Wifi

Figure 5.3: Startup Time Under Wifi - iPhone 4 vs iPhone 5s (Lower is better)

Figure 5.3 clearly shows that the time difference between the iPhone 5s and the reference time is very small. The iPhone 4 on the other hand, presents a slightly higher startup time due to older hardware with lower specifications and less processing power.

Scenario with Terminals connected over LTE: Figure 5.4 shows that the iPhone 5s under LTE has a very small startup time difference when compared with the reference time. This is possible due to goodt coverage at the test location and to the high-bandwidth provided by the 4G ISP.

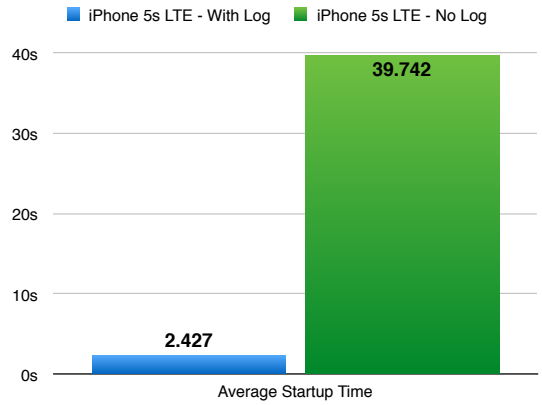
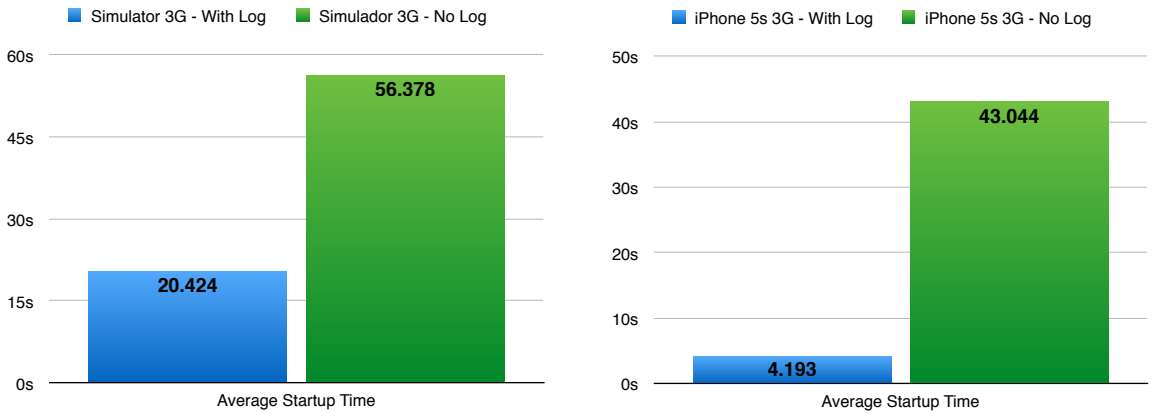


Figure 5.4: Startup Time - iPhone 5s LTE (Lower is better)

Scenario with Terminals connected over 3G and EDGE: Since the Network Link Conditioner forces the connection to emulate a 3G network with a 780kbps bandwidth, only the iPhone 5s is able to connects to a higher 3G bandwidth of several Mbps. Figure 5.5 shows a lower startup time on the iPhone 5s.



(a) Startup Time - Simulator 3G

(b) Startup Time - iPhone 5s 3G

Figure 5.5: Startup Time Under 3G - Simulator vs iPhone 5s (Lower is better)

Since Enhanced Data Rates for GSM Evolution (EDGE) is an older 2.5G network with a limited bandwidth, Figure 5.6 shows a really high startup time as expected. Even with a log file, the startup time

is over a minute which makes this type of connection not appropriate for media streaming.

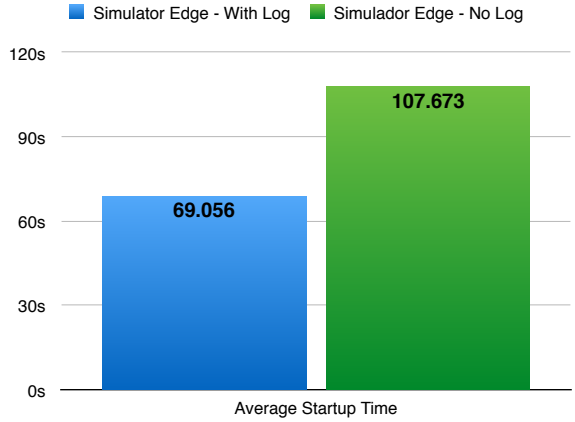
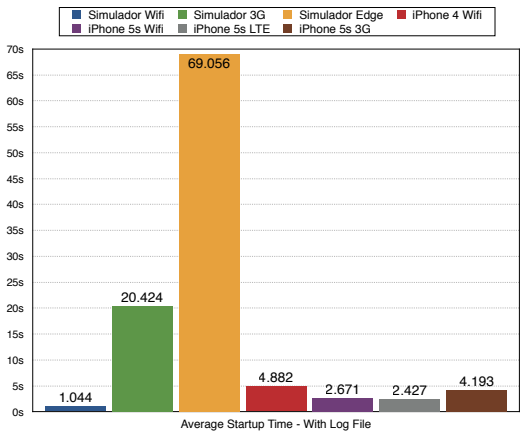
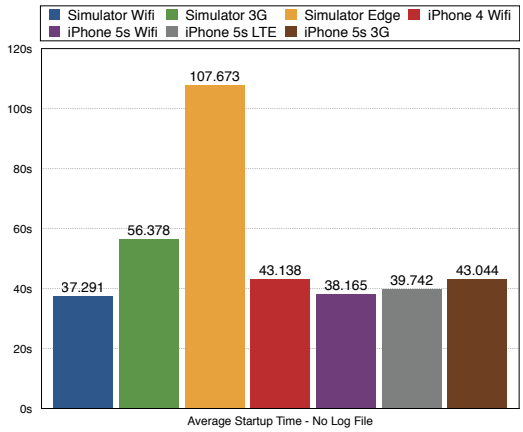


Figure 5.6: Startup Time - Simulator Edge (Lower is better)

Comparing all the network types and all the devices we can verify, as expected, that the startup time is significantly lower when the Log file exists and the video segments can be served right away. It is also easy to verify that the EDGE network is not appropriate to serve high quality media files.



(a) Global Startup Time - With Log



(b) Global Startup Time - No Log

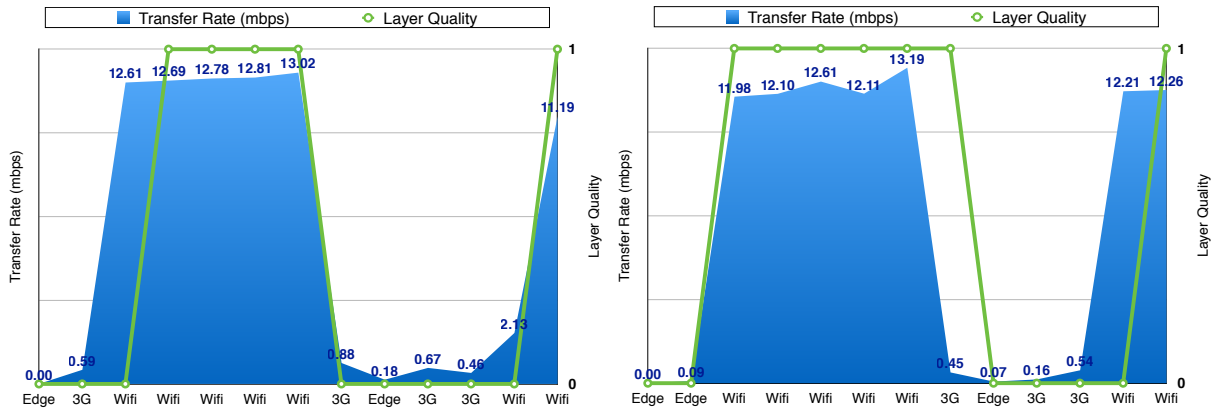
Figure 5.7: Startup Time - Global View (Lower is better)

Using the simulator with a restricted 3G network also shows that this technology can be slow to serve this type of media. The other networks present really similar values with acceptable startup times when the log file is not present. When the log file is present the startup times present fast startup times with an increased performance between 90% and 97%.

5.2.2 Adaptation Performance

This set of tests was designed to show the performance of the Adaptation System.

Figure 5.8(a) shows that the first transition to a quality 1 Layer is delayed. This situation happens because the the previous segment with Base layer quality was already requested when the Adaptation System receives the transmission rate and detects it has enough bandwidth for Layer 1.



(a) Adaptation System Test 1

(b) Adaptation System Test 2

Figure 5.8: Adaptation System Behavior Test

In a similar way, in Figure 5.8(b), the transition from a quality 1 Layer to base Layer when the network changes from Wifi, to 3G and then to EDGE is also delayed. This happens because, when the transition to 3G occurred, the segment with a Layer 1 Quality was already requested. When the system changes to EDGE, the systems verifies it still does not have the conditions for a Layer 1 quality and then transitions to a Base Layer Quality.

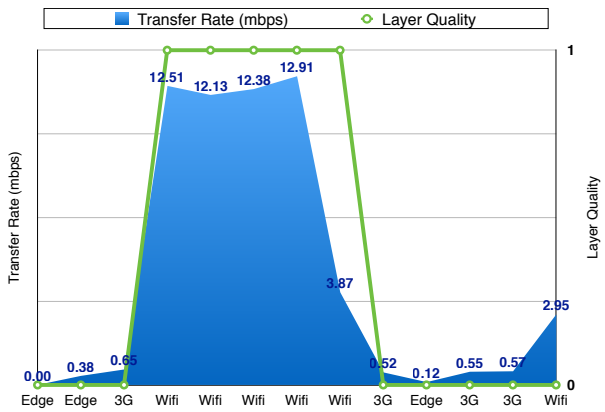


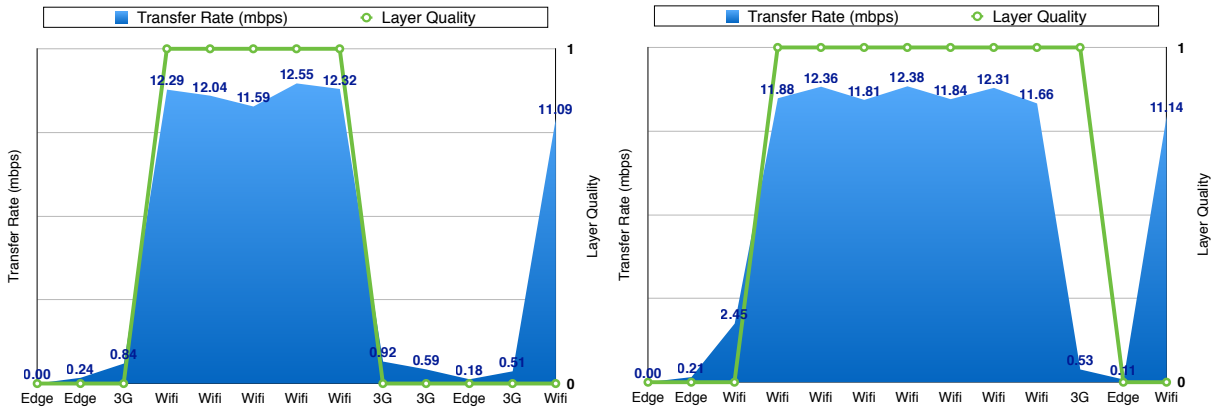
Figure 5.9: Adaptation System Test 3

Finally, in the last transition from 3G to Wifi, the systems present the same behaviour as can be

observed in Figure 5.8(a).

In Figure 5.9, the system only presents the same type of delay explained previously when a transition from Wifi to 3G occurs.

Both Figure 5.10(a) and Figure 5.10(b) present a “perfect” behaviour. The network transitions are well timed leading to a layer transition well timed and when expected.



(a) Adaptation System Test 4

(b) Adaptation System Test 5

Figure 5.10: Adaptation System Behavior Test

Looking at Figure 5.11 we can better understand why the delays occurred in some of the previously presented cases. In a set of measurements, the transition time from Wifi to 3G, in average presented a really high value leading the Adaptation System to continue using the Layer it was using previous to the network type switching. The remnant transitions between networks was fairly fast, not affecting the overall adaptation performance.

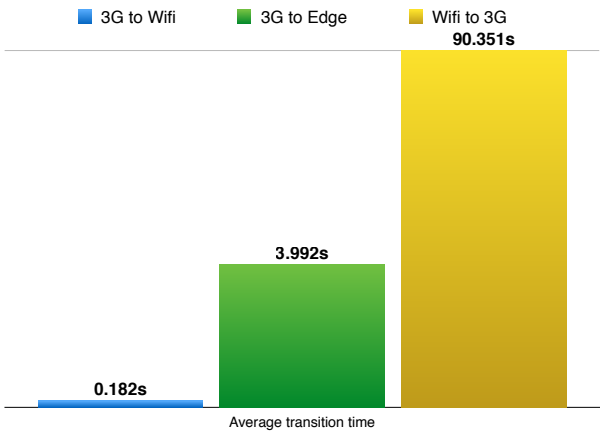


Figure 5.11: Average Transition Time Between Different Networks

5.2.3 Global Application Performance

The following screenshots were realized with Apple Instruments which is part of the the Xcode SDK for iOS developments. These screenshots display the overall performance and evaluations of CASHED running on an iPhone 5s.

Figure 5.12: Global Performance

Analyzing Figure 5.12 and Figure 5.13 it is possible to see that CASHED follows the expected behavior. The screenshots show synchronized peaks in the Connections, Network Activity and Activity Monitor modules. These peaks occurred every time the video was requested.

It also possible to check the wifi module activity in both screenshots. In Figure 5.12 the test always runs under WiFi and consequently the WiFi module is always active. In Figure 5.13 it is easy to observe a change in the WiFi module when it is turned off and the activity in the cellular module is also synchronized with the other modules.

Finally, it is possible to observe in those screenshots the overall consumption statistics of CASHED, when compared to other system applications such as the mail or SMS applications. These statistics show that the CASHED prototype uses less memory than most of the other system applications. It also shows a good performance in terms of CPU processing power and duration when playing the media contents.

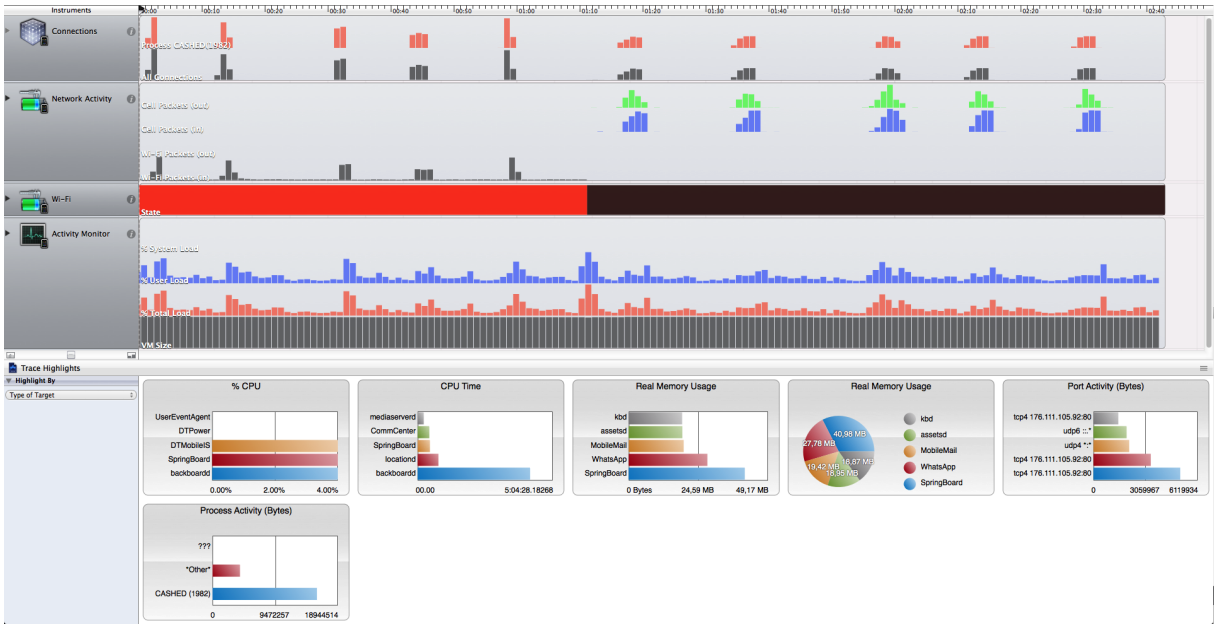


Figure 5.13: Gloabel Performance With Network Capture

6

Conclusion and future work

Contents

6.1 Conclusions 63
6.2 System Limitations and Future Work 63

6.1 Conclusions

This thesis presented the description of CASHED, a Cloud-Assisted Adaptive and Scalable Video Streaming for Heterogeneous End-User Devices.

For the design and development of CASHED, several technologies were considered, from video codecs, namely H.264/SVC, to mobile and cloud computing.

The architecture of CASHED respected functional and non-functional requirements, considers both client and distribution sides, with a particular focus on the main components of the client application and the mechanisms and techniques used to provide an optimal adaptation of the quality to the conditions of the device. In the distribution side, the SVC media transcoding and segmenting mechanisms were also described. The solution was designed and built in such a way that it is easy to adapt to other platforms such as Android OS.

The evaluation of the implemented prototype demonstrated a smooth and stable streaming solution that embodies mechanisms for quality adaptation and control, play-out session and device analysis.

In conclusion, the prototype developed proved to be stable, robust and simple to use with solid results. The nature and the modular approach of the design and development of the prototype created a solution easy to port to other platforms in future works.

6.2 System Limitations and Future Work

The prototype solution still has some limitations in the features that can be corrected and improved with further work, as some aspects considered not vital at this point of the development were not implemented.

The most critical limitations of the current prototype are as follows:

- The solution would benefit from an easy to use and functional SVC encoder to create media contents.
- A more appropriate overall UI needs to be designed.
- The Distribution System needs to be improved to be faster in the media treatment process.
- Due to the video rendering process in iOS, there are some limitations related with spatial scalability that makes noise appear in the image when a spatial change occurs. The video treatment processing can be optimized to correct this problem.
- Since the prototype was created with video only in mind, audio support needs to be implemented/tested.

- Additional adaptation mechanisms with better performance need to be developed. The heuristics and profiles created for the adaptation mechanisms can be refined to consider more machine conditions since the architecture of the prototype allows for the incorporation of additional heuristics.
- Even if the application runs on an iPad, a native UI for these devices needs to be designed and implemented.

Bibliography

[1] Apple, *HTTP Live Streaming Overview*, Apple Inc., 1 Infinite Loop, Cupertino, CA 95014, 408-996-1010 U.S., 2011. [Online]. Available: https://developer.apple.com/library/ios/documentation/networkinginternet/conceptual/streamingmediaguide/StreamingMediaGuide.pdf

[2] Adobe HTTP Dynamic Streaming. [Online]. Available: http://www.adobe.com/products/hds-dynamic-streaming.html

[3] Z. Alex. ISS Smooth Streaming Technical Overview. [Online]. Available: http://download.microsoft.com/download/4/2/4/4247C3AA-7105-4764-A8F9-321CB6C765EB/IIS_Smooth_Streaming_Technical_Overview.pdf

[4] Fraunhofer Heinrich-Hertz-Institute, "SVC: Scalable Extension of H.264/AVC," 2013. [Online]. Available: http://www.hhi.fraunhofer.de/de/kompetenzfelder/image-processing/research-groups/image-video-coding/scalable-video-coding/svc-scalable-extension-of-h264avc.html

[5] ISO/IEC, "Information technology – Coding of audio-visual objects – Part 10: Advanced Video Coding," International Organization for Standardization/International Electrotechnical Commission, International Standard ISO/IEC 14496-10:2012, Oct. 2012.

[6] H. Schulzrinne, A. Rao, and R. Lanphier, "Real Time Streaming Protocol (RTSP)," Internet Engineering Task Force, RFC 2326, Apr. 1998. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2326.txt

[7] S. Schulzrinne, H. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," RFC 3550 (Standard), Internet Engineering Task Force, July 2003.

[8] Adobe. HTTP Dynamic Streaming on the Adobe Flash Platform. [Online]. Available: https://bugbase.adobe.com/index.cfm?event=file.view&id=2943064&seqNum=6&name=httpdynamicstreaming_wp_ue.pdf

[9] ISO/IEC, “Information Technology – Coding of audio-visual objects – Part 14: MP4 file format,” International Organization for Standardization/International Electrotechnical Commission, International Standard ISO/IEC 14496-14:2003, November 2003.

[10] R. Pantos, Ed. and W. May, “HTTP Live Streaming draft-pantos-http-live-streaming-11,” Informational, Internet Engineering Task Force, Draft, October 2013. [Online]. Available: http://tools.ietf.org/html/draft-pantos-http-live-streaming-11

[11] Apple, *iOS Technology Review*, Apple Inc., 1 Infinite Loop, Cupertino, CA 95014, 408-996-1010 U.S., September 2012. [Online]. Available: http://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/iPhoneOSTechOverview.pdf

[12] ISO/IEC, “Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats,” International Organization for Standardization/International Electrotechnical Commission, International Standard ISO/IEC FCD 23009-1:2012, Apr. 2012.

[13] ITU-T, “Advanced video coding for generic audiovisual services,” International Telecommunication Union - Telecommunication Standardization Sector, Recommendation H.264 (03/2010), Mar. 2010.

[14] J. Bankoski, J. Salonen, P. Wilins, and Y. Xu, “VP8 Data Format and Decoding Guide,” RFC 6386, IETF, RFC 6386, November 2011. [Online]. Available: http://tools.ietf.org/html/rfc6386

[15] H. Schwarz, D. Marpe, and T. Wiegand, “Overview of the Scalable Video Coding Extension of the H.264/AVC Standard,” *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 17, no. 9, pp. 1103–1120, 2007.

[16] P. Moscoso, “Interactive Internet TV Architecture Based on Scalable Video Coding,” Master’s thesis, Instituto Superior Técnico, May 2011.

[17] Y.-H. Chiang, P. Huang, and H. Chen, “SVC or MDC? That’s the question,” in *Embedded Systems for Real-Time Multimedia (ESTIMedia), 2011 9th IEEE Symposium on*, 2011, pp. 76–82.

[18] C. Feller, J. Wuenschmann, T. Roll, and A. Rothermel, “The VP8 video codec - overview and comparison to H.264/AVC,” in *Consumer Electronics - Berlin (ICCE-Berlin), 2011 IEEE International Conference on*, 2011, pp. 57–61.

[19] Y. Sharrab and N. Sarhan, “Detailed Comparative Analysis of VP8 and H.264,” in *Multimedia (ISM), 2012 IEEE International Symposium on*, 2012, pp. 133–140.

[20] ITU-T, “Information technology – Generic Coding of moving pictures and associated audio information: Systems,” Internation Telecomunication Union - Telecomunication Standardization Sector, Recommendation H.222.0 (05/2006), May 2006.

[21] ISO/IEC, “Information Technology - Coding of audio-visual objects - Part 12: ISO base media file format,” International Organization for Standardization/International Electrotechnical Commission, International Standard ISO/IEC 14496-12:2012, July 2012.

[22] B. MacAulay, A. Felts and Y. Fisher, “IP Streaming of MPEG-4 Native RTP vs MPEG-2 Transport Stream,” WHITEPAPER, October 2005. [Online]. Available: http://www.envivio.com/files/white-papers/RTPvsTS-v4.pdf

[23] Cisco Systems, “IP Multicast Technical Overview,” Cisco Systems, Whitepaper, August 2007. [Online]. Available: http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6552/prod_white_paper0900aecd804d5fe6.pdf

[24] Apple Developer Center, “iOS,” 1 Infinite Loop, Cupertino, CA 95014, 408-996-1010 U.S., 2013. [Online]. Available: https://developer.apple.com/technologies/ios/

[25] Wikipedia, “iOS,” 2013. [Online]. Available: http://en.wikipedia.org/wiki/IOS

[26] Wikipedia, “Android (operating system),” 2013. [Online]. Available: https://en.wikipedia.org/wiki/Android_(operating_system)

[27] P. Mell and T. Grance, “The NIST Definition of Cloud Computing,” Special Publication 800-145, September 2011.

[28] G. Motta, N. Sfondrini, and D. Sacco, “Cloud Computing: An Architectural and Technological Overview,” in *Service Sciences (IJCSS), 2012 International Joint Conference on*, 2012, pp. 23–27.

[29] A. Hendryx. (2011, October). [Online]. Available: http://www.zdnet.com/cloudy-concepts-iaas-paas-saas-maas-caas-and-xaas-4010024679/

[30] P. McDonagh, C. Vallati, A. Pande, P. Mohapatra, P. Perry, and E. Mingozzi, “Investigation of scalable video delivery using h.264 svc on an lte network,” in *Wireless Personal Multimedia Communications (WPMC), 2011 14th International Symposium on*, Oct 2011, pp. 1–5.

[31] Z. Huang, C. Mei, L. Li, and T. Woo, “Cloudstream: Delivering high-quality streaming videos through a cloud-based svc proxy,” in *INFOCOM, 2011 Proceedings IEEE*, April 2011, pp. 201–205.

[32] X. Wang, M. Chen, T. Kwon, L. Yang, and V. Leung, “Ames-cloud: A framework of adaptive mobile video streaming and efficient social video sharing in the clouds,” *Multimedia, IEEE Transactions on*, vol. 15, no. 4, pp. 811–820, June 2013.

[33] X. Wang, T. Kwon, Y. Choi, H. Wang, and J. Liu, “Cloud-assisted adaptive video streaming and social-aware video prefetching for mobile users,” *Wireless Communications, IEEE*, vol. 20, no. 3, pp. 72–79, June 2013.

[34] Fraunhofer Heinrich-Hertz-Institute. (2014, May) JSVM Reference Software. Accessed 13-May-2014. [Online]. Available: http://www.hhi.fraunhofer.de/de/kompetenzfelder/image-processing/research-groups/image-video-coding/svc-extension-of-h264avc/jsvm-reference-software.html

[35] F. Bellard and M. Niedermayer. (2014, May) FFMpeg. Accessed 13-May-2014. [Online]. Available: http://ffmpeg.org/documentation.html