Interactive Internet TV Architecture Based on Scalable Video Coding

Pedro Gomes Moscoso

Dissertação para obtenção do Grau de Mestre em Engenharia de Redes de Comunicações

Júri
Presidente: Prof. Doutor Paulo Jorge Pires Ferreira
Orientador: Prof. Doutor Mário Serafim dos Santos Nunes
Co-Orientador: Prof. Rui António dos Santos Cruz
Vogais: Prof. Doutor Paulo Luis Serras Lobato Correia

Maio de 2011
I would like to thank my parents for their friendship, encouragement and caring over all these years, for always being there for me through thick and thin and without whom this project would not have been possible. I would also like to thank my sisters, Ana, Teresa and Joana, grandparents, aunts, uncles and cousins for their support throughout all this time. I would also like to acknowledge my dissertation supervisors Professor Rui Cruz and Professor Mário Nunes for their insight, support and sharing of knowledge that has made this dissertation possible.

Last but not least, to all my friends and colleagues that helped me grow as a person and were always there for me during the good and bad times in my life. Thank you.

To each and every one of you - thank you.
Abstract

In a diversified environment as the Internet, with a wide diversity of terminals and connectivity, a proper video distribution is a complex problem. The multimedia distribution through several service providers has grown broadly in last years, constituting a large portion of the Internet traffic. However, many of these services do not support mechanisms to provide quality of user experience, causing stoppages in video playing, degradation of image or even failure. This is due to the lack of adequate and reliable adaptive mechanisms to fit the capabilities of the network and of terminals.

This thesis presents an analysis of the various techniques, architectures and concepts for video adaptation, and describes a novel web-based Adaptive Video Streaming solution prototype, supporting Scalable Video Coding (SVC) techniques. The proposed solution, focused on the client side, incorporates quality control mechanisms to maximize the end user Quality of Experience (QoE), is suitable for Interactive Internet TV Architectures and cooperative with Content Distribution Networks (CDNs) and Peer-to-Peer (P2P) web-streaming environments. The evaluation results show that the solution provides great robustness and the ability to adapt to extreme situations.

Keywords

Internet video streaming, adaptive streaming, H.264/AVC, scalable video coding, peer-to-peer
Resumo

Num meio tão diversificado como é a Internet, com uma grande heterogeneidade de terminais e ligações, a distribuição de vídeo de uma forma adequada apresenta-se como um problema de resolução complexa. A distribuição de conteúdos multimédia, através de diversos serviços, tem crescido amplamente nos últimos anos e representa já uma considerável fatia do tráfego na Internet. No entanto muitos destes serviços não tomam em consideração a qualidade de experiência do utilizador, originando paragens na visualização do vídeo, degradação da imagem ou mesmo incapacidade de reprodução. Tal facto deve-se à falta de mecanismos adaptativos que se ajustem as capacidades da rede e do terminal. Nesta tese apresenta-se uma análise sobre as técnicas, arquitecturas e conceitos para a adaptação de vídeo e apresenta-se uma solução inovadora para um protótipo de distribuição de vídeo adaptativo baseado nas tecnologias da Web, capaz de suportar vídeo escalável. A solução, focada na parte cliente, inclui mecanismos de controle de qualidade para poder maximizar a qualidade de experiência do utilizador, é apropriada para arquitecturas de TV interactiva pela Internet e compatível com redes de distribuição de conteúdos e de distribuição de vídeo Peer-to-Peer (P2P). Os resultados da avaliação do protótipo mostram que a solução apresentada oferece grande robustez e capacidade de adaptação a situações extremas.

Palavras Chave

Distribuição de vídeo, streaming adaptativo, H.264/AVC, vídeo escalável, peer-to-peer
Contents

1 Introduction ......................................................... 1
  1.1 Motivation and Objectives .................................... 2
  1.2 Contributions .................................................. 3
  1.3 Dissertation research topics ................................. 3
  1.4 Dissertation layout ............................................ 4

2 State of the Art and Related Work ................................ 5
  2.1 Introduction .................................................... 6
  2.2 Network Streaming Modes ...................................... 6
     2.2.1 IP Unicast .................................................. 6
     2.2.2 IP Multicast (Network-layer Multicast) .................. 7
     2.2.3 Overlay Multicast (Application-layer Multicast) ...... 7
  2.3 Classical Streaming Protocols .................................. 7
     2.3.1 Real-Time Streaming Protocol (RTSP) .................... 8
     2.3.2 Real-Time Transport Protocol (RTP)/RTP Control Protocol (RTCP) 8
  2.4 Web based Streaming Techniques ............................. 8
     2.4.1 Microsoft Smooth Streaming .............................. 9
     2.4.2 HTTP Live Streaming ..................................... 9
     2.4.3 Adobe Dynamic Streaming ............................... 10
  2.5 Video Codecs ................................................... 11
     2.5.1 H.264/AVC ............................................... 11
     2.5.2 H.264/SVC ............................................... 12
     2.5.3 MDC ..................................................... 14
  2.6 MPEG-2 transport stream ...................................... 15
  2.7 Adaptation system architectures .............................. 15

3 The Internet TV Architecture ....................................... 17
  3.1 Introduction .................................................... 18
  3.2 Architecture Design ............................................ 18
3.3 The Content Distribution side

3.3.1 Partition system
3.3.2 Index Manifest file

3.4 The Client side

3.4.1 Decoder
3.4.2 User Interface
3.4.3 User Interface (UI) Controller
3.4.4 Index/Manifest XML Download/Parser
3.4.5 Stream Controller
3.4.6 Adaptation System
3.4.7 Layer Requester
3.4.8 Reassembler
3.4.9 Real Time Live Video Streaming
3.4.10 External Interfaces
3.4.11 Media Stream Transport

4 Implementation

4.1 Introduction
4.2 Development Process and Environments
4.3 The Scalable Video Coding (SVC) Internet TV Player
4.3.1 Graphical User Interface
4.3.2 UI/Controller
4.3.3 Video Decoder
4.3.4 Index/Manifest XML Download/Parser
4.3.5 Adaptation System
4.3.6 Stream Controller
4.3.7 Layer Requester
4.3.8 Reassembler
4.3.9 Configuration
4.4 The Internet TV Distribution/Server components
4.5 P2P Gateway/P2P Engine

5 Evaluation

5.1 Introduction
5.2 Evaluation Objectives and Experimental Scenarios
5.3 Evaluation Results
5.3.1 100 Mbit/s Local Bandwidth
List of Figures

2.1 Data Structures of H.264/AVC video encoder. ................................................. 12
2.2 Spatial Scalability ............................................................................................ 13
2.3 Quality Scalability .......................................................................................... 13
2.4 Temporal Scalability [1] ................................................................................. 14
2.5 Multiplexing video, audio and program information. ..................................... 15

3.1 Client Side Component Architecture ............................................................. 20
3.2 SVC file NAL header ...................................................................................... 21
3.3 Server Side Component in SVC Streaming .................................................... 21
3.4 Identification of NAL order ............................................................................ 22
3.5 User Interface .................................................................................................. 25
3.6 Stream Controller .......................................................................................... 27
3.7 Layer Download .............................................................................................. 28
3.8 Transmission Control Protocol (TCP) Session Establishment ...................... 29
3.9 Multiple TCP Session with offset ................................................................... 30
3.10 Reconstruction Process .................................................................................. 31
3.11 Chunks available on the server ..................................................................... 33

4.1 Internet TV Player components ....................................................................... 37
4.2 Windows Forms UI Application ....................................................................... 38
4.3 Video Controls and Stream Information bar in the UI Application ............... 38
4.4 Internet Explorer running the Player plugin ...................................................... 39
4.5 Mac OS X Cocoa UI Application ...................................................................... 40
4.6 The Stream Information object with Stream Properties returned by the XML Parser ................................................................. 41
4.7 Message sequence to obtain the Index/Manifest (illustrated for Peer-to-Peer (P2P)) ............................................................. 42
4.8 Stream Controller Class .................................................................................. 42
4.9 Stream Controller Flow .................................................................................... 43
4.10 Configuration Form ................................................................. 45
5.1 Test environment ................................................................. 48
5.2 Network Load at 100 Mbit/s ...................................................... 51
5.3 Round-Trip Time (RTT) at 100 Mbit/s .......................................... 51
5.4 Cache at 100 Mbit/s ................................................................. 52
5.5 Downloaded Layers at 1 Mbit/s .................................................. 52
5.6 PSNR at 1 Mbit/s ...................................................................... 53
5.7 Network Load at 1 Mbit/s ......................................................... 53
5.8 RTT at 1 Mbit/s ......................................................................... 53
5.9 Cache at 1 Mbit/s ..................................................................... 54
5.10 Downloaded Layers at 256 kbit/s .............................................. 54
5.11 PSNR at 256 kbit/s ................................................................. 55
5.12 Network Load at 256 kbit/s ..................................................... 55
5.13 RTT at 256 kbit/s ................................................................. 56
5.14 Cache at 256 kbit/s ............................................................... 56
5.15 Downloaded Layers during a 100 Mbit/s to 256 kbit/s drop ............. 57
5.16 PSNR at a 100 Mbit/s to 256 kbit/s drop .................................... 57
5.17 Network Load at a 100 Mbit/s to 256 kbit/s drop ......................... 58
5.18 RTT at a 100 Mbit/s to 256 kbit/s drop ....................................... 58
5.19 Cache behavior at a 100 Mbit/s to 256 kbit/s drop ......................... 59
5.20 Downloaded Layers in 3G Mobile Network .................................. 59
5.21 PSNR in 3G Mobile Network .................................................... 60
5.22 Network Load in 3G Mobile Network ....................................... 60
5.23 RTT in 3G Mobile Network ..................................................... 60
5.24 Cache in 3G Mobile Network .................................................. 61
5.25 Variation in network load due to screen size change ....................... 61
A.1 PSNR between layers from video used in tests .................................. 72
A.2 Log Analyzer ........................................................................... 72
List of Tables

2.1 Adaptive Streaming technologies Comparison . . . . . . . . . . . . . . . . . . . . . . . . . 10
3.1 Chunk partitioned structure . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
5.1 Average size of each layer level file for the video used . . . . . . . . . . . . . . . . . . . . 49
5.2 Possible PSNR to Mean Opinion Score (MOS) conversion [3] . . . . . . . . . . . . . . . 50
List of Algorithms

3.1 Screen Size Adaptor .................................................. 29
3.2 Scalable Video Coding File Reassembler .......................... 31
Listings

3.1 Manifest file for H.264 SVC Streaming ........................................... 23
3.2 Manifest file for H.26 AVC Streaming .............................................. 24
3.3 Manifest file for H.26 SVC Live ...................................................... 32
4.1 canDownloadNextLayer function code ............................................. 44
A.1 Output log from encoder ............................................................... 73
# List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>3G</td>
<td>3rd Generation</td>
</tr>
<tr>
<td>ALM</td>
<td>Application Layer Multicast</td>
</tr>
<tr>
<td>AS</td>
<td>Autonomous System</td>
</tr>
<tr>
<td>AVC</td>
<td>Advanced Video Coding</td>
</tr>
<tr>
<td>CDN</td>
<td>Content Distribution Network</td>
</tr>
<tr>
<td>CIF</td>
<td>Common Intermediate Format</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DCIF</td>
<td>Double Common Intermediate Format</td>
</tr>
<tr>
<td>DVB</td>
<td>Digital Video Broadcast</td>
</tr>
<tr>
<td>FWA</td>
<td>Fixed Wireless Access</td>
</tr>
<tr>
<td>GOP</td>
<td>Group of Pictures</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HD</td>
<td>High Definition</td>
</tr>
<tr>
<td>HDTV</td>
<td>High-Definition Television</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IETR</td>
<td>Institut d'Electronique et des Telecommunications de Rennes</td>
</tr>
<tr>
<td>IGMP</td>
<td>Internet Group Management Protocol</td>
</tr>
<tr>
<td>IIS</td>
<td>Internet Information Services</td>
</tr>
<tr>
<td>INOV</td>
<td>INESC Inovação</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>INSA</td>
<td>Institut National des Sciences Appliquees de Rennes</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IPTV</td>
<td>Internet Protocol Television</td>
</tr>
<tr>
<td>ISP</td>
<td>Internet Service Provider</td>
</tr>
<tr>
<td>ITU-T</td>
<td>International Telecommunication Union - Telecommunications Standardization Sector</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>MB</td>
<td>Megabyte</td>
</tr>
<tr>
<td>MDC</td>
<td>Multiple Description Coding</td>
</tr>
<tr>
<td>MOS</td>
<td>Mean Opinion Score</td>
</tr>
<tr>
<td>MP4</td>
<td>MPEG-4 Part 14</td>
</tr>
<tr>
<td>MPEG</td>
<td>Moving Picture Experts Group</td>
</tr>
<tr>
<td>MVC</td>
<td>Multiview Video Coding</td>
</tr>
<tr>
<td>NAL</td>
<td>Network Abstraction Layer</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>PSNR</td>
<td>Peak Signal Noise Ratio</td>
</tr>
<tr>
<td>QoE</td>
<td>Quality Of Experience</td>
</tr>
<tr>
<td>RTCP</td>
<td>RTP Control Protocol</td>
</tr>
<tr>
<td>RTMP</td>
<td>Real Time Messaging Protocol</td>
</tr>
<tr>
<td>RTP</td>
<td>Real-Time Transport Protocol</td>
</tr>
<tr>
<td>RTSP</td>
<td>Real-Time Streaming Protocol</td>
</tr>
<tr>
<td>RTT</td>
<td>Round-Trip Time</td>
</tr>
<tr>
<td>SD</td>
<td>Standard Definition</td>
</tr>
<tr>
<td>SEI</td>
<td>Supplemental Enhancement Information</td>
</tr>
<tr>
<td>SNR</td>
<td>Signal-to-Noise Ratio</td>
</tr>
<tr>
<td>SVC</td>
<td>Scalable Video Coding</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>VCL</td>
<td>Video Coding Layer</td>
</tr>
<tr>
<td>VoD</td>
<td>Video On Demand</td>
</tr>
<tr>
<td>WAN</td>
<td>Wide Area Network</td>
</tr>
<tr>
<td>WLAN</td>
<td>Wireless Local Area Network</td>
</tr>
<tr>
<td>WWAN</td>
<td>Wireless Wide Area Network</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
Introduction

Contents

1.1 Motivation and Objectives ................................................. 2
1.2 Contributions .......................................................... 3
1.3 Dissertation research topics .......................................... 3
1.4 Dissertation layout ..................................................... 4
1.1 Motivation and Objectives

When the first multimedia contents started to be available in the Internet only high end personal computers were able to consume those contents with ease. But most of the problems related with multimedia content consumption were not essentially due to processing power but to bandwidth limitations and network congestion.

The growth and expansion of the broadband Internet and the increasing number of connected devices with multimedia content play-out capabilities, with recent handheld devices supporting high definition video [4], raises several problems to the distribution of multimedia contents.

From the end user device side, the wide range of networked devices with different characteristics and capabilities, such as screen size and resolution, Central Processing Unit (CPU) processing power, operating system and media player applications, poses big challenges to the distribution of contents, hardly scaled for all of those devices.

From the access network provider side, the heterogeneous nature of connectivity to the Internet, e.g., several wireless and fixed network technologies, either guarantee uniform and stable conditions of reception at any moment in time to the end user devices.

From the content producer and/or broadcaster side it is rather important not to waste resources and that multimedia contents reach the end users with the best quality possible. Producers and/or broadcasters are already offering the same multimedia content encoded in several formats and with different qualities/bitrates so that the end user can be better served.

These situations have been the main drivers for the techniques recently developed, aiming to reduce the mentioned problems in order to optimize multimedia distribution.

Recent developments by companies like Apple, Microsoft and Adobe are already providing mechanisms, more or less transparent to the end user, that are able to support dynamic variations in video streaming quality while ensuring support for a plethora of end user devices and network connections. For these techniques to work, the original content is re-encoded with different qualities and bitrates [5–7].

With the SVC extension to the H.264/AVC [8] standard for video coding, new approaches are now possible for video distribution and consumption as videos can now be encoded in layers corresponding to several hierarchical levels of quality.

These technologies created new opportunities for video content distribution systems, capable to provide scalable and adaptive multimedia contents in a more efficient and simpler way.

The work described in this thesis corresponds to the development of an Adaptive Video Streaming system prototype supporting scalable video coding techniques, suitable for Interactive Internet TV Architectures.
1.2 Contributions

The heterogeneity of the access networks to be supported and the diverse terminal capabilities characteristics were the main constraints taken into consideration for the design of the prototype solution.

From the analysis of these constraints new key ideas were developed for the processing of SVC media content distribution and for the adequate data requesting strategies, capable to be supported both on client-server or P2P transport mechanisms on the Internet, including Content Distribution Networks (CDNs).

New techniques were developed to allow the adequate media transport and re-assembling of adaptive and scalable video, eliminating the need to encode the video in different bitrates. The main contributions were the design and the implementation of the algorithms that allow the adequate video streaming over HTTP protocol and the partitioning of the source video in chunks (and then in quality layers for transport), that can be recombined to produce a playable video with variable quality (adaptive).

The Internet TV Player prototype solution described in this thesis was developed within the scope of the European project SARACEN [9].

SARACEN stands for “Socially Aware, collaboRative, scAlable Coding mEdia distributioN” and it is a research initiative funded partially by the Information Society and Media Directorate General of the European Commission under the Seventh Framework Programme.

The SARACEN initiative will research on distribution of multimedia streams supported through innovative techniques, both as regards media encoding and media distribution including P2P schemes, where the peer nodes will participate in an end to end media distribution system making use of social networking related information [9].

The implementation and evaluation of the Internet TV Player prototype has been described in a paper [10] accepted for, and to be presented at, the European Interactive TV Conference Workshop on Quality of Experience for Multimedia Content Sharing, Lisbon, Portugal in June 2011 [11].

1.3 Dissertation research topics

To reach the goals and objectives of the work described in this thesis, the following research topics were explored: Multimedia Streaming techniques, from classical Internet Protocol (IP) streaming modes and protocols to Web-based streaming. Media coding and decoding techniques, with special focus on Scalable Video Coding (SVC). Adaptation Techniques for the scheduling of media distribution based on client terminal capabilities and access network capabilities. Streaming Mode constraints: On-demand versus Live (real-time “broadcast” of TV shows) modes.
1.4 Dissertation layout

This dissertation is composed of 6 chapters that share a common structure. They are logically separated in two main sections: a general introduction and the body of the chapter where the subject matter is discussed.

Chapter 2 gives an introduction to the state-of-the-art and related work.
Chapter 3 presents the Architecture of the Interactive Internet TV, with focus on the SVC Player.
Chapter 4 describes the various stages followed in the development process for the implementation of the Internet TV SVC Player prototype.
Chapter 5 describes the experimental scenarios and evaluation process of the Internet TV SVC Player prototype.
Chapter 6 summarizes the contributions of the dissertation, and also suggests areas for future work.
State of the Art and Related Work

Contents

2.1 Introduction ................................................................. 6
2.2 Network Streaming Modes .................................................. 6
2.3 Classical Streaming Protocols .............................................. 7
2.4 Web based Streaming Techniques ........................................ 8
2.5 Video Codecs ................................................................. 11
2.6 MPEG-2 transport stream .................................................. 15
2.7 Adaptation system architectures ......................................... 15
2.1 Introduction

The most used streaming modes on IP-based networks are the traditional unicast and multicast modes. But multicast, essentially due to business models, is limited to the realm of a Service Provider Autonomous System (AS). However, new “modes” are rising, changing dramatically the way these services reach the consumers, as is the case of Application Layer Multicast (ALM) and other P2P streaming technologies.

There are three major delivery methods for video streaming over the Internet:

- the traditional packet-based streaming, that uses protocols like RTSP and RTP
- progressive download, that allows the play-out of the video while the file is being downloaded from an Hypertext Transfer Protocol (HTTP) Server.
- HTTP Adaptive Streaming, a method that successively downloads small segments of video (and audio) to play-out as they arrive, also able to smoothly switch to different quality/bitrate of video (available at the source) depending on variations of network/host conditions, ensuring the best possible user experience.

A fourth emerging method also uses Application Layer overlay technologies and for multi-source/multi-network multimedia streaming and adaptation on the Internet.

The formats used for multimedia delivery are also diverse as both video and audio can be encoded using different codecs. But recently the trend for coding audio-visual information (e.g., movies, video, music) is with the Moving Picture Experts Group (MPEG) family of standards, in particular the H.264 [12] for video and the MP3 (MPEG-1/2 Audio Layer 3) or AAC (MPEG-2/4 Advanced Audio Coding) for audio [13–15].

2.2 Network Streaming Modes

Most of the video streaming services today still use traditional unicast or multicast techniques, requiring powerful centralized servers and high bandwidth connections.

2.2.1 IP Unicast

IP unicast is a one-to-one connection between two Internet nodes, typically one server and one client. On the streaming context, each client receives a dedicated multimedia stream from the server and takes up a fraction of the available bandwidth.

An example of an IP Unicast streaming service is Youtube [16], where there is always a flow of video data for each client, even with the help of CDNs to distribute the load on the YouTube source servers by caching frequently requested objects closer to the client location.
2.2.2 IP Multicast (Network-layer Multicast)

IP multicast [17] is a one-to-many or many-to-many communication between server and several clients. In IP multicast there is typically no direct relationship between each client and the server. It is a bandwidth-conserving technology implemented at the network elements (e.g., routers) that reduces traffic by simultaneously delivering a single stream to many clients. All the work of packet replication is performed by the network elements.

The multicast topology is based on the concept of Group. Each host (client) that wants to receive a stream (in multicast mode), must join the respective group [18]. These groups are managed by a leave/join protocol, the Internet Group Management Protocol (IGMP) [19].

Today, the IP multicast deployment remains limited as not all Internet Service Providers (ISPs) offer support for it unless their business model includes some form of multimedia content distribution like Internet Protocol Television (IPTV). The reasons are complex and do not involve technical issues only but also economic and political concerns [20]. IP multicast requires routers that can maintain group state, and this condition introduces high complexity and scalability problems.

However, the IP multicast is commonly used by IPTV service providers, confined to their distribution networks.

2.2.3 Overlay Multicast (Application-layer Multicast)

The concept of Overlay Multicast network is of a mix of unicast and multicast networks [21] with the purpose of creating multicast routing at the application level via unicast connections between the networked nodes.

As opposed to network-layer multicast, the application layer multicast requires no network infrastructure support for the multicast and can be easily deployed in the Internet, and across ASs.

Using the network endpoints (hosts) to implement all the multicast features, such as group membership, routing and packet duplication, creates more latency, additional signaling traffic and some redundant traffic on physical links. But eliminates all the economic and political difficulties, as well as the slow process required on the implementation of traditional IP multicast.

2.3 Classical Streaming Protocols

Traditional video streaming protocols, i.e., RTSP and RTP/RTCP, are stateful protocols, meaning that the server keeps track of the client state from the instant the client connects up to the end of the session [22–24]. In this delivery method the server also keeps the state of the client video buffer and adjusts the transmission rate accordingly. If the client pauses the video, the server pauses the transmission.
These protocols are packet-level protocols, meaning that the media (e.g., audio or video) is segmented and transported as payload of small packets that match a temporal segment of video.

2.3.1 RTSP

RTSP is an application level protocol that establishes and controls streams of real-time continuous media such as audio or video [23]. The objective of this protocol is to control multiple sessions of media streams. The transmission of streaming data itself is not a task of the RTSP protocol. This protocol does not infer a notion of “connection”, but of “session” allowing it to be flexible with transport protocols such as TCP or User Datagram Protocol (UDP) and also allowing clients to open and close connections to the server just by maintaining a session identifier.

With RTSP the content is streamed by an independent transport protocol like RTP. All operations are performed by the exchange of messages (i.e., trick functions) like SETUP, PLAY, STOP or PAUSE.

One of the major problems in the use of RTSP is when the transport protocol is UDP and the clients are located behind firewalls, as these may block the downstream of packets. Additionally, as UDP does not provide congestion control mechanisms, the reaction to congestion may take place too late and affect the user experience (interruptions, frame losses, jumps, artifacts, etc.).

2.3.2 RTP/RTCP

RTP/RTCP are application level protocols for end-to-end multimedia content transmission with real-time support [24]. RTP provides mechanisms for jitter compensation and for detection of out of sequence arrivals in the data. The RTP packet is composed by a Header and a Payload. The header has all information about the packet like payload type, sequence number, marker and timestamp. Multimedia content is segmented and formatted according to its type into the RTP packet payload. RTCP [22] is used in conjunction with RTP to monitor data distribution (transmission statistics and quality of service information) by periodic transmission of control packets between client and server.

2.4 Web based Streaming Techniques

The most common way to stream videos in the World Wide Web (the technique used in websites like YouTube and Vimeo [16][25]) is the Progressive Download technique.

However, Progressive Download is not a real streaming technique but a regular HTTP file download, used with a peculiar behavior in the media player client. That peculiar behavior, termed “progressive”, means that while in the background the media player client is downloading the movie content (until the download is complete) the movie is already playing-out (starting as soon as enough video content is in
the buffer). Even if the user pauses the movie while the download is still not completed, the media player client still keeps downloading the movie.

Recently, new techniques for Web streaming have emerged, namely, HTTP Adaptive Streaming (essentially Apple’s [HTTP] Live Streaming [26] and Microsoft’ Smooth Streaming [27]) and Adobe’s Dynamic Streaming [7]. This techniques are summarized in Table 2.1.

The key idea behind HTTP Adaptive Streaming approach is the capability to support dynamic adaptation in video quality by bitrate/quality switching.

### 2.4.1 Microsoft Smooth Streaming

Smooth Streaming is a Web streaming technology that acts like packet streaming but based on HTTP download of a long sequence of small files corresponding to segments of the media. This concept was created to enable an uninterrupted streaming experience [6].

For Smooth Streaming to work, multimedia sources are encoded in various bitrates/qualities and cut into short synchronized segments called “chunks”. Typically, each chunk corresponds to 2-to-4 seconds of video, cut along Group of Pictures (GOP) boundaries, without dependencies with past or future chunks, and can be played independently.

Smooth Streaming uses MPEG-4 Part 14 (MP4) Media File Format specification [28], for both the storage (disk) and transport (wire) formats, architected with H.264 video codec support. With this format, all the chunks of each multimedia source are stored within a single MP4 file, being each chunk a MPEG-4 Movie Fragment of a file with a certain bitrate, hosted on a specialized HTTP Web server.

Smooth Streaming uses a Extensible Markup Language (XML) index file named Manifest, that includes information about the encoder used, the bitrates and resolutions available and a list of the corresponding chunks (with start times and durations) for the media. This Manifest file is downloaded by the client media player at the start of the session in order to initialize the correct decoder and build the playback pipeline.

For maximizing the Quality Of Experience (QoE), Smooth Streaming uses bandwidth information and local machine conditions for the decision algorithm in order to seamlessly switch the video quality that the user is receiving as function of the variations of those conditions.

Smooth Streaming requires some “intelligence” on the web server (Smooth Streaming requires Microsoft’s Internet Information Services (IIS) 7.0 with special Extensions) in order to correctly extract the time-indexed media chunks from the corresponding MP4 container file.

### 2.4.2 HTTP Live Streaming

Live Streaming is a Web streaming technology developed by Apple for iPhone/iPad/iPod iOS [26]. HTTP Live Streaming and Smooth Streaming share the same functional concept but from two radical view-
HTTP Live Streaming is proposed as an open protocol [5] while Smooth Streaming is proprietary. Additionally, HTTP Live Streaming does not require a special Web server. HTTP Live Streaming defaults to MPEG-2 transport streams or program streams containing H.264 video and AAC audio, but is capable of accommodating other formats.

HTTP Live Streaming also segments the multimedia source in chunks. These chunks typically have 10 seconds of video and start with a key-frame, do not have dependencies on past and future chunks and can be decoded independently. In this technology the encoder is responsible not only for the segmentation process but also for creating the index file (a MP3 Playlist) containing the list of bitrates available as well as the corresponding chunks with their individual Uniform Resource Locators (URLs). This index file is stored in the Web server that contains the video chunk files.

### Table 2.1: Adaptive Streaming technologies Comparison

<table>
<thead>
<tr>
<th>Smooth Streaming</th>
<th>Dynamic Streaming</th>
<th>HTTP Live Streaming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Streaming Protocol</td>
<td>HTTP</td>
<td>RTMP</td>
</tr>
<tr>
<td>Stateless Connection</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Supported Platforms</td>
<td>Silverlight, Xbox 360, iOS</td>
<td>Flash Player 10 AIR</td>
</tr>
<tr>
<td>Media Players</td>
<td>H.264 Baseline, Main, and High</td>
<td>H.264 Baseline, Main, and High, VP6</td>
</tr>
<tr>
<td>Video Codecs</td>
<td>WMA, AAC</td>
<td>AAC, MP3</td>
</tr>
<tr>
<td>Audio Codecs</td>
<td>No limit</td>
<td>No limit</td>
</tr>
<tr>
<td>Maximum Bitrate</td>
<td>2-4 seconds</td>
<td>n/a</td>
</tr>
<tr>
<td>Chunk default Length</td>
<td>1.5 sec/1 Chunk</td>
<td>6 sec</td>
</tr>
<tr>
<td>E2E Latency</td>
<td>MP4</td>
<td>MP4, FLV</td>
</tr>
<tr>
<td>Media Container Formats</td>
<td>Contiguous</td>
<td>Contiguous</td>
</tr>
<tr>
<td>Container File type</td>
<td>Contiguous</td>
<td>Contiguous</td>
</tr>
</tbody>
</table>

### 2.4.3 Adobe Dynamic Streaming

Dynamic Streaming [29], developed by Adobe, is a technology created for their Flash Player to support adaptive streaming over Real Time Messaging Protocol (RTMP) or HTTP [20]. RTMP is a proprietary protocol developed by Adobe for streaming audio and video over TCP.

This technology does not fragment the media into chunks. The media is encoded into different bitrate
files and all key-frames are marked with a time stamp to ensure synchronization. The transitions between different bitrates are only possible at key-frame boundaries. All adaptation decisions need to be implemented by developers in the media players but the process requires cooperation from server functions in order to switch to a video bitrate on the correct time [7]. The transition between different bitrates is not smooth as during the transition the video player pauses to fill the buffer.

In this technology, there are no adaptation decisions supplied as reference libraries and so, all of the adaptation decisions need to be implemented by the developer of the player.

2.5 Video Codecs

There are plenty of video codecs, but lately, the trend falls on standard H.264, essentially due to the huge expansion of web-video services, although not without controversy related to “openness”, as H.264 is royalty-supported. For the work developed on this thesis, and in the SARACEN Project, the option fall on the H.264 coding standards, and its extension, the H.264 SVC.

2.5.1 H.264/AVC

H.264/AVC is a video compression coding standard developed by International Telecommunication Union - Telecommunications Standardization Sector (ITU-T) Video Coding Experts Group (VCEG) together with the ISO/IEC Moving Picture Experts Group (MPEG) [12,30].

With the growing number of High-Definition Television (HDTV) offers by broadcasters or content providers and the popularity of broadband transmission media such as Cable Modem, xDSL, or UMTS/HSDPA (although offering much lower data rates than terrestrial radio broadcast channels, i.e., Digital Video Broadcast (DVB)), the demand for higher coding efficiency and reduced bandwidth occupancy led to an almost unanimous choice of H.264/AVC video compression standard, as it is able to provide bitrate savings in a factor of two when compared to MPEG-2 Video and MPEG-4 Visual [31].

One of the goals of this new standard is on its flexibility to be deployed for a large variety of applications and networks. The design includes a Video Coding Layer (VCL), responsible for the creation of a coded representation of the source content, and a Network Abstraction Layer (NAL) to format the VCL representation of the video and to provide appropriate header information for conveyance by transport layer protocols (such as RTSP) or storage media formats.

The H.264/AVC NAL units are packets that start with one-byte header and with an integer number of bytes. The last 4 bytes represent the type of content in the packet. The content can be a VCL or a non VCL payload. If the content represented is VCL, the NAL contains a complete or a partitioned coded slice while non VCL NALs only contains auxiliary data for the Supplemental Enhancement Information (SEI) decoding process.
In H.264/AVC there are three slice (frame) types: I, P and B.
The I-slice (Intra Frame) contains spatial intra-picture predictive coding. This slice does not use temporal predictions from another slice and there is no temporal redundancy, making these frames rather expensive in rate. The P-Slice (Predictive) may only use forward prediction from previous I and P slices. The B-Slices (Bidirectionally) may use both forward and backward prediction from previous and future I and P slices. These B and P slices use Inter-picture and Intra-picture predictive coding. Inter-picture uses the temporal redundancy between nearby pictures. Intra-picture uses the spatial redundancy inside the same picture. After the coding processes the residuals are quantized and compressed. A context-based adaptive entropy coding is then performed and the data VCL is encapsulated into NAL units.

2.5.2 H.264/SVC

H.264 SVC is an extension of the H.264/MPEG-4 AVC standard (Annex G) proposed and designed by the Fraunhofer Institute. SVC can provide support for several display resolutions with variable bitrate without the need to encode the media in several bitrates. This scalability in video is possible by the creation of several layers of quality supported by the partition of the source bit stream into several sub-streams by simply cutting parts of the data. Due to its simplicity it may be performed in intermediate network elements without requiring high computation work like what happens with video transcoding.
The layers represent levels of quality for the end user. One important aspect is the backward compatibility with H.264/AVC. The base layer of an encoded H.264/SVC stream is compatible with H.264/AVC and can be decoded without modifications.
The scalability in SVC is offered by three metric dimensions: space, time and quality, illustrated in Figures 2.2, 2.3 and 2.4. The spatial scalability dimension represents the number of encoded pixels per image. Spatial scalability provides support for several display resolutions with arbitrary ratios. In spatial scalability both motion-compensated and intra prediction techniques are employed within the same spatial layer. However, to reduce the spatial redundancy between these layers, SVC also includes an inter-layer prediction mechanism between spatial resolutions.

The spatial scalability feature of SVC is not restricted to fixed scaling ratios. For example two spatial layers can be encoded and transmitted, the first for Standard Definition (SD) television with a 4:3 aspect ratio, and a second layer for High Definition (HD) television with 16:9 aspect ratio.

The temporal scalability dimension allows the support of various frame rates. This is done by discarding some temporal layers. The main concepts of temporal scalability was included in H.264/AVC and is usually implemented by B-slices.

With quality scalability dimension, the sub-stream provides the same spatio–temporal resolution as the complete bit stream, but with a lower Signal-to-Noise Ratio (SNR).

All of the three scalability dimensions can be combined with each other, and are distributed over the bitstream by NAL units, each representing a level of spatial, temporal and quality levels [12]. Each NAL have three identifiers named Definition (DID), Quality (QID) and Temporal (TID). Each ID is an integer in a hierarchical order from lowest to highest quality fidelity. The ID can be replaced for a wildcard “*”. In this case all components of the ID are merged in the same NAL.

These values are stored in the NAL extended header that contains other parameters, important to deter-
mine the interdependency between NALs and to identify, combine or discard packets. Interdependency means that the loss of a NAL can cause a quality reduction within the same layer by a quality reduction of other layers or simply prevent the decoding of the video.

In SVC each layer is dependent of the previous layer, starting from the base layer up to the last enhancement layer, and do not have the same impact in video quality.

Currently the OpenSVC MPlayer [1,32] is the only open-source media player that decodes H.264 SVC. However, there is an effort from project P2PNext [33] to provide SVC support to the VLC Media Player [34].

2.5.3 MDC

Another concept, with similar purposes of SVC but using a completely different approach, is Multiple Description Coding (MDC) [35].

In MDC the encoded video is divided into several sub-streams named “descriptions”. Each description can be decoded independently in a low quality video stream. The video stream quality improves with the addition of other descriptions. The original video quality can be obtained if all descriptions are decoded together [35].

The main difference from MDC to SVC relies in the prioritization of different parts of the video. While in MDC the coded bit stream has the same importance and contributes with the same amount to the final quality, in SVC each hierarchical layer increases the quality dimension of the previous layer.

Data portioning of MDC demands a substantial amount of coding overhead and increases the complexity of the decoder [36].
2.6 MPEG-2 transport stream

The MPEG-2 transport stream \[37\] is a standard container format for transmission of multiple encapsulated audio, video and/or data, that implements mechanisms for error correction and stream synchronization in order to maintain transmission integrity when the signal is degraded.

MPEG-2-TS packets are composed of 188 byte TS Packets, each with a 4 byte header (Figure 2.5) \[2\]. A video ES (Video Elementary Stream), consists of the element that is to be carried, e.g., video or audio. In MPEG-2 TS a packetized elementary stream, or PES, consists of a single ES which has been made into packets, each packet starting with an added packet header.

PES packets have variable length, not corresponding to fixed packet length of transport packets, and may be much longer than a transport packet \[37\]. In this case the remaining PES packet content fills the payloads of successive transport packets. The final transport packet is padded to a fixed length with "zero" bytes.

![Figure 2.5: Multiplexing video, audio and program information.](image)

Each PES packet header includes an 8-bit stream ID identifying the source of the payload. PES packet headers, as well as Adaptation Fields, contain timing information necessary to synchronize multiple streams for playback (for example, synchronize video with audio) . 

RTP is commonly used for transporting MPEG-2 TS packets.

2.7 Adaptation system architectures

The adaptation system that controls the variation of video bitrates can be deployed at different locations. There are typically three scenarios:

**Client Side:** In this scenario the client software implements the mechanisms to determine the current host conditions and the variations in the network, i.e., CPU type and load, bandwidth, delays, etc. The client can then request to the server the bitrate that best fits the measured conditions. Usually this kind of solution does not require from the server any stateful information about the client.
**Server Side:** In this scenario the client periodically informs the server about the variations of conditions in the point of reception of the stream. The server can then determine the most adequate video bitrate for the client.

**Network:** In this scenario the adaptation system is located in the network (e.g., in proxy servers). The client can inform the proxy about the variations of conditions in the point of reception of the stream. The proxy can then determine the most adequate video bitrate for the client and so, is able to transcode the original content, or to modify the number of frames of a video, or to modify the resolution of a stream. With **SVC** this type of modification can be done also by discarding some of the layers.
The Internet TV Architecture

Contents

3.1 Introduction ........................................... 18
3.2 Architecture Design ................................... 18
3.3 The Content Distribution side ....................... 21
3.4 The Client side ....................................... 24
3.1 Introduction

The Internet TV distribution network architecture considers end user nodes and serving platforms. The end user nodes are distributed peers (with P2P capabilities) that can produce, consume and share contents, offering their resources (bandwidth, processing power, storage capacity) to other end user nodes.

The serving platforms are centralized service nodes providing control (tracker for P2P), content treatment and distribution (transcoders and media servers), as well as interaction tools and facilities. The architecture is a multi-source HTTP client and server solution providing an advanced form of Web-Streaming and WebSeeding (HTTP-based P2P Streaming Protocols) [38].

The process used for streaming distribution relies on a chunk transfer scheme whereby the original video data is chopped into small video chunks with a short duration (of typically two seconds). The chunk based streaming protocols allow the deployment of a distribution network compatible with the Internet infrastructure, such as Web caches and CDNs as well as P2P distribution.

The description of both the serving platforms of the Internet TV distribution network and of the P2P Streaming Protocols are not covered in this work – just a brief overview of the SVC partition system and of the local P2P engine – as its focus is on the client side components for a solution able to consume Video On Demand (VoD) and Live video services, supporting multiple device types and resolutions with adaptive streaming mechanisms based on the SVC extension of H.264/AVC.

3.2 Architecture Design

The main purpose of this work was the creation of a complete system able to support multiple device types and resolutions with adaptive streaming mechanisms based on H.264/AVC SVC extension, as well as the support of H.264/Advanced Video Coding (AVC) media content not requiring adaptive mechanisms.

The functional requirements for the solution were defined as follows:

- **Video Player over HTTP**: Support for video streaming and Live video Streaming over HTTP Protocol.


- **Support for H.264/AVC/SVC/MDC**: Support for video streaming of contents encoded in variants of H.264/AVC, i.e., SVC and MDC.

- **Adaptative Video**: Support for Adaptative Video in contents encoded in H.264/SVC.
• **Terminal independency**: The solution shall be able to support several device types regardless of the machine performance or screen resolution or size.

• **Access Network independency**: The solution shall be able to provide services to Clients connected over Local Area Networks (LANs) or Wireless Local Area Networks (WLANs) and over several Wide Area Network (WAN)/Wireless Wide Area Network (WWAN) access network technologies, such as Ethernet, xDSL, Cable, Fixed Wireless Access (FWA), 3rd Generation (3G) or WiMax.

To comply with these requirements the design of the architecture followed a modular approach with the following main components for both the client side and the distribution side:

**In the content distribution side:**

- Encoder, responsible for video encoding in H.264/AVC/SVC/MDC.
- Partitioner System for H.264/SVC, including:
  - Chunk Partitioner, responsible for splitting the video files into chunks.
  - Intra-Chunk Layer Partitioner, responsible for splitting the video chunk files in several quality levels.
  - Manifest Creator, responsible for the creation of the index/manifest file of the SVC content.
- Partitioner System for H.264/AVC/MDC, including:
  - Chunk Partitioner, responsible for splitting the video files into chunks.
  - Manifest Creator, which is responsible for the creation of the index/manifest file of the MDC content.

**In the client side:**

- User interface: The human interaction component, providing input and output for the application, and display of the decoded video. This module can take the form of a traditional application (media player) or of a Web browser plugin.
- UI Controller: This component is the bridge between the media decoder, the UI and the Stream Downloading process.
- Decoder: Decodes the reassembled video file.
- XML Downloader/Parser: Interprets the index/manifest file requested over HTTP and configures the system for video downloading. This module provides configuration options for the P2P external component.
Stream Controller: Controls the entire flow of download and reconstruction of the media segments and assists other components on performing their functions.

Adaptation System: This module is where all the adaptation decisions take place using several heuristics in order to assist/inform the Stream Downloader of the maximum possible quality supported at any moment.

The Layer Requester: Provides the methods for HTTP Layer requests.

The Reassembler System: Rebuild the SVC or MDC video to the possible level of quality.

External Interfaces: These interfaces use HTTP for communication sessions with Web servers or Web Proxies (such as the P2P Gateway/Engine).

The main components on the client side are illustrated in Figure 3.3.
3.3 The Content Distribution side

The Content Distribution side provides the functions to prepare the media (files or raw inputs from cameras) to be encoded and indexed for the distribution. In this section, only the H.264 SVC related processes will be described, as they are the focus of the work.

For that purpose, the original binary H.264 SVC file (produced by the video encoder) is divided into a sequence of NAL units. Each NAL unit is preceded by a four bytes start code (‘00h’ ‘00h’ ‘00h’ ‘01h’) followed by a descriptor byte, as illustrated in Figure 3.2 [39].

In this byte, the value of “Type” is important for the analysis of the content of the NAL unit, as, also according to [39], NAL unit types of 14 and 20 indicate the presence of three additional bytes in the NAL unit header, that will contain the identifiers Definition (DID), Quality (QID) and Temporal IDs (TID), fundamental for the analysis or extraction processes of the SVC layer structure.

3.3.1 Partition system

The Partition System is the component that splits the original SVC video file into independent video Chunks (that can be played independently) and then into several layers.

The Partition system is composed by three components, the Chunk Partitioner, the Layer Partitioner and the index generator, as illustrated in Figure 3.3.
**Chunk Partitioner**

The first task of the Partitioner is the partition of the SVC encoded video file in a set of independent video chunks that can be played independently, each one with 2 seconds of length. This feature is the basis for the support of live video streaming, as it turns independent of the media timeline the moment when a user joins the stream (can join at any moment of the timeline and immediately begin watching the stream). This is done by cutting the data over SEI.

The chunks are delimited by \texttt{SEI NAL} units. Each chunk starts when a \texttt{SEI NAL} unit appears in the encoded file and ends in the \texttt{NAL} unit that precedes the next \texttt{SEI NAL} unit.

**Intra-chunk Layer Partitioner**

The second task of the Partitioner is the partition of the SVC independent chunk files into several transmission layers.

This process begins with the demultiplexing of the NALs and with their identification with one ID. This procedure is very important for the reconstitution of the original bit stream at the client side. The identification process is done by the insertion of a Sequence Numbering field with 2 bytes between the start code (a sequence of 0001 bytes) and the beginning of the \texttt{NAL} unit, as illustrated in Figure 3.4.

![Figure 3.4: Identification of NAL order](image)

After this step, chunks can be partitioned into several transmission layers by using the three identifiers already described: Definition (DID), Quality (QID) and Temporal IDs (TID).

In a typical scenario, two resolutions (the original and the half of the original) and 10 transport Layers can be used, exploring the definition scalability of SVC. The first layer (base layer) includes all the temporal layers for the base resolution, guaranteeing the decryption at full frame-rate (25 or 30 fps), but with less quality. The higher level enhancement layers increase the PSNR of some frames. In this scenario, up to layer 4, the video has low resolution but a high PSNR. From the fifth layer onwards the process is repeated but for the maximum resolution. Table 3.1 exemplifies the NAL division into layers.

After the partition of the video chunk files in layer files the Index/Manifest file is created.

**3.3.2 Index Manifest file**

An important detail of the solution for the SVC or MDC encoding process, was the non-use of a container file to embed the partitioned media. The partitioned files only include the encoded video without any
Table 3.1: Chunk partitioned structure

<table>
<thead>
<tr>
<th>Order of Interdependency</th>
<th>Filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAL units with (D,T,Q) = (0,*0) video chunkNum</td>
<td>L0.264</td>
</tr>
<tr>
<td>NAL units with (D,T,Q) = (0,0,1) video chunkNum</td>
<td>L1.264</td>
</tr>
<tr>
<td>NAL units with (D,T,Q) = (0,1,1) video chunkNum</td>
<td>L2.264</td>
</tr>
<tr>
<td>NAL units with (D,T,Q) = (0,2,1) video chunkNum</td>
<td>L3.264</td>
</tr>
<tr>
<td>NAL units with (D,T,Q) = (0,3,1) video chunkNum</td>
<td>L4.264</td>
</tr>
<tr>
<td>NAL units with (D,T,Q) = (1,*0) video chunkNum</td>
<td>L5.264</td>
</tr>
<tr>
<td>NAL units with (D,T,Q) = (1,0,1) video chunkNum</td>
<td>L6.264</td>
</tr>
<tr>
<td>NAL units with (D,T,Q) = (1,1,1) video chunkNum</td>
<td>L7.264</td>
</tr>
<tr>
<td>NAL units with (D,T,Q) = (1,2,1) video chunkNum</td>
<td>L8.264</td>
</tr>
<tr>
<td>NAL units with (D,T,Q) = (1,3,1) video chunkNum</td>
<td>L9.264</td>
</tr>
</tbody>
</table>

additional information such as timestamps, subtitles, etc., as this information for play-out is included in the manifest file.
This option turns also possible the decoding contents that use chunks as transport units, even if embedded in a container file, as is the case of H.264/AVC chunks used in the HTTP Live Streaming.

Example of XML for SVC Video

Listing 3.1: Manifest file for H.264 SVC Streaming

```xml
<StreamInfo>
   <Clip>
      <Name>video_complete</Name>
      <Description>some description</Description>
      <Video Codec="SVC" FPS="25.0">
         <VideoQuality Width="432" Height="240" Level="0" />
         <VideoQuality Width="842" Height="480" Level="6" />
      </Video>
      <VideoChunkIndex>
         <Segment Chunks="629" Levels="10" />
      </VideoChunkIndex>
   </Clip>
</StreamInfo>
```

The Index/Manifest file holds information about the content, i.e., describes the structure of the media, namely, the codecs used, the chunks, the number of layers (in case of SVC) or number of descriptions.
(in case of MDC), the audio component (in case it exists), etc., and the corresponding mapping within a container file system.

### Listing 3.2: Manifest file for H.26 AVC Streaming

```xml
<?xml version="1.0" ?>
<StreamInfo>
  <Clip>
    <Name>video_avc</Name>
    <Description>some description</Description>
    <Video Codec="AVC" FPS="25.0">
      <VideoQuality Width="432" Height="240"/>
    </Video>
    <VideoChunkIndex>
      <Segment Chunks="629"/>
    </VideoChunkIndex>
  </Clip>
</StreamInfo>
```

The Index/Manifest file is a Well-Formed XML Document, i.e., does not use a Document Type Definition (DTD), and is encoded as double-byte Unicode (UTF-8 encoding attribute value in the XML declaration). The Index/Manifest is a composite schema (as described in Listings 3.1 and 3.2) that includes, as root Element, a StreamInfo field that encapsulates all the metadata required for the play-out of the media in groups of Clip fields with ChunkIndex elements corresponding to each component of the media (video, audio) streams, as well as other associated metadata (subtitles, text descriptions, etc.). The different Clip elements may include specific descriptor elements, like Video and Audio and respective attributes for each of the media types.

### 3.4 The Client side

The Client Side corresponds to the Internet TV Player, complemented with a P2P Gateway/Engine, in case the streaming transport method id done via P2P.

This section describes the components of the Internet TV Player, and briefly the Media Stream transport modes.

#### 3.4.1 Decoder

The Decoder is responsible for the decoding of the video chunks, either if they are of H.264 SVC, H.264 Multiview Video Coding (MVC) or H.264 AVC.
This component provides an interface that permits the interaction and the control of basic trick function operations such as the Pause, Stop or Play of the media.

The Decoder module for the H.264 [SVC] is transparent to the system in order to be replaceable in the future by newer improved decoder module.

### 3.4.2 User Interface

The User Interface Module is the front-end of the whole system and may take the form of a typical application window, or of a web browser plugin.

The User Interface module is composed by the Video Renderer and the Video Controls (Figure 3.5).

![User Interface Diagram](image)

**Figure 3.5: User Interface**

- **Video Renderer**: The Video Renderer module provides an interface to the Media Player in order to give access to it and render the decoded video frames.

- **Video Controls**: Opens the [URLs] and provides Trick Function controls such as Play, Pause and Stop, as well as Time seeking of the media being played-out.

### 3.4.3 UI Controller

The UI Controller component allows the control the entire flow of information between the UI the player and the Stream Downloader component.
This component is responsible for receiving all user inputs (Open URL, Video Play, Pause, Seek, etc.) from the [UI] and pass them for other lower level components.

It is also responsible for receiving information from the Stream Download module and pass it to the higher level components, as in the case of re-buffering, where there is the need to stop the player and display the information of that condition in the [UI].

This component must also implement mechanisms to inform the Adaptation System of any changes in machine conditions, such as changes in the size of the video window or the [CPU] load. This component, together with the [UI], are the ones that are platform-dependent and need to be implemented differently for each target platform.

This component will also be responsible by providing values from some of the metrics to be used in the Adaptation system, such as display resolution or window frame size.

### 3.4.4 Index/Manifest XML Download/Parser

The XML Download/Parser is the component that downloads over HTTP and parses the manifest XML file in order to get the operation configuration for processing the media.

If the downloaded XML contains [P2P] information (PeerID, Tracker URL), this component connects to the [P2P] Gateway external component via the configuration interface to announce the beginning of the stream download.

### 3.4.5 Stream Controller

The Stream Controller is the core component of the system, responsible for managing the entire video streaming flow.

The stream download process is initiated from the [UI] Controller that informs the Stream Controller about the URL of the media manifest/index file. This information must be handled by the XML Parser that returns all the information required for processing the stream.

After the initial setup, the download process starts with the premise that the video buffer must always be filled (Figure 3.6).

**H.264/SVC:** Since H.264/SVC can be demultiplexed in various quality levels, the system makes use of this property to increase the user experience.

Before starting the download this component asks the adaptation system about the maximum recommended number of layers.

After this step, the system begins to download the various layers. These downloads are made through the layer Layer Requester module. The download process is entirely controlled by the Adaptation System that takes decisions on the maximum video layers that can be supported at any moment, depending
upon host and network conditions (Figure 3.7).

**H.264/AVC and H.264/MDC:** For video contents not split into layers, as are the cases of MDC and AVC, this component just downloads and reassembles the video from the chunks received.

### 3.4.6 Adaptation System

The Adaptation System is responsible for the adjustments in video quality by determining the number of layers to request from a server, based on a set of heuristics related to network and host system conditions.

**Network Load Adaptation:** The purpose of the Network Load Adaptation heuristic is to ensure that the maximum possible number of layers can be safely downloaded for the available bandwidth. Network conditions such as bandwidth and RTT are continuously measured, with their averages used as smoothing factors to prevent abrupt changes in the quality of the video. This ensures that the variations between layers are smooth and causes an almost imperceptible impact on the user viewing experience. While the system is downloading the layers, the average size of each layer, the RTT, and the bandwidth are stored in memory, and used to estimate the download duration of the next layer, and to decide if more layer levels can be downloaded. If the time already taken for the download of lower layers exceeds the estimated time for the new layer, and is higher than the video chunk duration (typically 2 seconds), the system decides not to download the next layer of the current chunk and proceeds for the next chunk layers.
This guarantees that if there is enough bandwidth available, for at least the base layer, the video never stops or the play-out never stops for re-buffering due to starvation.

**Screen Size Adaptation:** The Screen Size Adaptation is another important heuristic for the downloading scheme as it helps to determine the adaptation to the spacial scalability of the SVC content (levels of layers) being played-out, and therefore, the corresponding adaptation of the network traffic conditions (reducing it by requesting less layer levels in the case of small screen sizes).

The information about the video resolution of a certain content is obtained from the manifest file. Whenever the screen size ratio reduces below any of the resolutions described in the manifest file, the number of enhancement layers is automatically adjusted by this heuristic (Algorithm 3.2).

### 3.4.7 Layer Requester

The Layer Requester component includes all the necessary code for downloading the layers via HTTP.

This component implements mechanisms to estimate the download time, the remaining time, bandwidth and RTT in the current session, as well as to cancel the download and to support multiple simultaneous downloads. The values measured by this component are used by the Adaptation System.

Due to limitations imposed by the prototype version of the P2P Gateway used during the development...
Algorithm 3.1 Screen Size Adaptor

\[
\text{availableResolutions} = \text{getAvailableResolutions}(); \\
\text{screenSize} = \text{getVideoScreenSize}(); \\
\text{for } i = 0 \text{ to } \text{availableResolutions.length} \text{ do} \\
\quad \text{if } \text{ScreenSize} \leq \text{availableResolutions}[i].\text{resolution} \text{ then} \\
\quad \quad \text{return } \text{Level} = \text{availableResolutions}[i].\text{level}; \\
\quad \text{end if} \\
\text{end for} \\
\text{return } \text{maxLayerLevel};
\]  

work of this thesis, not supporting yet HTTP persistent connections, the download of each layer had to be done on separate TCP sessions. This method increases the download time, especially when we accessing remote HTTP peer servers due to the RTT between the client and the server, as the delay between the start of a connection and starting to get layer information is approximately 2xRTT as shown in Figure 3.8. During this time the bandwidth, if no workaround was used, would be wasted!

![Figure 3.8: TCP Session Establishment](image)

However, in order to optimize the usage of the bandwidth and reduce the impact of the time used in TCP session establishment, a mechanism of parallel sessions with an offset was developed, as a workaround, until being possible to start using HTTP persistent connections in the P2P mode.

The parallel sessions with an offset mechanism, uses information from the Adaptation System to predict how long the download will take to finish while downloading a layer. When the time to finish equals the value of two RTT and the Adaptation System allows the download of the next layer (Figure 3.9), a new download is immediately started by spawning a new Layer Requester instance, i.e., not waiting for the previous layer arrival to start the new download.
Figure 3.9: Multiple TCP Session with offset
3.4.8 Reassembler

The Reassembler is the component responsible for creating an independent video chunk file from the received layer files.

![Figure 3.10: Reconstruction Process](image)

The received layer files contain several NALs that are identified by the same ID. This ID identifies the order of the NAL in the final video chunk. The Reassembler system, after having received all the layers from the Stream Controller reorganizes all the NALs by ID order and creates a video chunk file with the corresponding downloaded quality levels. This video chunk is then sent to the video player buffer.

The pseudo-code for the process used in the SVC File Reassembler is presented in Algorithm 3.2.

**Algorithm 3.2 Scalable Video Coding File Reassembler**

```plaintext
chunk = newMemoryStream();
nalList = newEmptyList();
layerData = byte[numlayers];
for i = 0 to numlayers do
    layerData[i] = readfile(fileStreams[i])
end for
for i = 0 to numlayers do
    nalList.insert(GetH264LayerNALUnits(layerData[i]));
end for
nalList.Sort(bySeqNum);
for element = 0 to nalList.Count do
    WriteToMemoryStream(nalList[element]);
end for
return chunk.toByteVector();
```

31
3.4.9 Real Time Live Video Streaming

The architecture includes the support for real-time Live streaming. In this case, the index/manifest file does not provide the full information of the content but just the current time-window (the latest \( N \) chunks encoded, with \( N = 5 \), typically).

```
Listing 3.3: Manifest file for H.26 SVC Live

1 <StreamInfo>
2   <LiveStream>TRUE</LiveStream>
3   <UpdateTime>16:31:55</UpdateTime>
4     <Clip>
5       <Name>live</Name>
6         <Description>some description</Description>
7         <Video Codec="SVC" FPS="25.0">
8           <VideoQuality Width="432" Height="240" Level="0"/>
9           <VideoQuality Width="842" Height="480" Level="6"/>
10     </Video>
11     <Duration>1258</Duration>
12     <VideoChunkIndex>
13       <Segment FromChunk="41" ToChunk="45" Levels="10"/>
14     </VideoChunkIndex>
15   </Clip>
16 </StreamInfo>
```

The index/manifest file is updated periodically, for every \( N \) new chunks encoded. The chunk IDs in the manifest are available in the server for a duration such that, for any new connected client, there is still enough time to consume any of the indexed chunks without them being removed from the server as shown in Figure 3.11.

On the client side the whole process is identical, except that the initial chunk to be downloaded is the one referenced int the field FromChunk. For Real Time Live Video Streaming the video seek function is disabled.

3.4.10 External Interfaces

The system architecture considers two external interfaces: the HTTP/P2P Interface and the P2P Configuration Interface. The HTTP/P2P Interface defines the communication between the Stream Controller module and a streamer, either an external WebStreaming server or the local P2P Gateway engine. The P2P Configuration Interface is used to configure the P2P engine, namely the maximum upload and download bandwidth allowed, the type of layers that are allowed, etc.
3.4.11 Media Stream Transport

The solution provides an advanced form of WebStreaming, i.e., a reliable HTTP-based media streaming transport, not just in client-server mode but also in P2P mode.

The client side components include a local HTTP process that listens at a local port to redirect HTTP GET or POST methods initiated from either the local web browser or from the application Graphical User Interface (GUI), to either the P2P engine or to the appropriate external Web server, basing its decision on information taken from the Index/Manifest of the content.

The P2P engine in each node provides WebSeeding capabilities to other peer nodes, behaving as both a client and a server in the P2P network, using novel HTTP-based P2P Streaming Protocols being developed in the scope of the SARACEN project [9,38].

The P2P engine serving side is composed by two entities: a P2P messaging core engine and a HTTP server for the data transport and signaling, acting as the underlying transport layer for video, audio streams and accompanying metadata in the P2P network.

Peers use the P2P Streaming Protocols for resource discovery (peers and contents), signaling (connection management), streaming control (content request management) and streaming transport (content data exchange among peers). The communication messages of the P2P Streaming Protocols are grouped in the following sets:

- Tracker-Peer Messages (i.e., Tracker Protocol): The Tracker Protocol messages provide communication between peers and Trackers, by which peers provide content availability, report streaming status and request candidate peer lists from Trackers.

- Peer-Peer Messages (i.e., Peer Protocol): The Peer Protocol messages enable each peer to ex-
change content availability with other peers and request other peers for content.

This P2P Streaming technique is a pull model, i.e., it is the client that is responsible for handling the buffering and playback processes during the download of the media chunked files. All the intelligence of this HTTP streaming design resides on the P2P serving and client core engine, rather than in the network transport. The peer, based on the information received in the index/manifest file of the content, constructs a series of Uniform Resource Identifiers (URIs) pointing to the media chunk files (layer files or descriptions) using the HTTP GET method with the corresponding URI and host (peer address). Upon receiving the HTTP request, the other peer HTTP server sends the media file corresponding to the URI in the request.
4.1 Introduction

This chapter describes the stages followed in the development and implementation of the Internet TV Player prototype. A brief overview of the programming language and tools/libraries that were used is also presented, as well as the development environments and the solutions used for building the various modules.

4.2 Development Process and Environments

The development process of the entire prototype was divided into the following stages:

- Requirements gathering and analysis,
- Research,
- Architecture design,
- Implementation,
- Functional Validation.

The prototype was developed keeping in mind portability to multiple operating systems and types of equipment.

For that purpose the choice fall on the Mono framework [40]. Mono can be used to develop applications targeting Android, iOS, Linux, BSD, Mac OS X, Solaris, Unix or Windows, operating systems as well as some game console operating systems such as the ones for the PlayStation 3, Wii, and Xbox 360.

In order to demonstrate the platform portability capabilities three prototypes for the Internet TV Player were built:

- A standalone player for Windows,
- A standalone Player for Mac OS X
- A browser Plugin Player for Windows Internet Explorer.

On the Distribution side, the focus was on the SVC encoding and partitioning processes, not aiming to create full functional platform components, but just the tools to create/generate the contents for streaming.

For the SVC encoder the reference encoder [41] was used. For the partitioning a set of MATLAB scripts were developed in order to generate the chunks and then the layer files.
4.3 The SVC Internet TV Player

Taking advantage of using an object-oriented language, the various Client Side Components of the Internet TV Player (represented in the diagram of Figure 4.1) were developed as classes.

![Diagram of Internet TV Player components](Image)

Figure 4.1: Internet TV Player components

4.3.1 Graphical User Interface

For the GUI, three different types of interfaces were developed:

- A Windows Forms UI application,
- A Windows Internet Explorer plugin,
- A Mac OS X Cocoa application.

Windows Forms UI application

Figure 4.2 shows a screenshot of the application in during play-out. The Menu bar at the top, in the first panel, allows the user to select in the File menu option the desired SVC content, via the corresponding index/manifest file, available at web server and pointed by an URL. In the View menu option the user can choose showing or hiding the Log Window and the Stream Information Window. The video renderer is where the video output from the Video Decoder is rendered. The Video Controls bar, Figure 4.3, offers trick-functions, i.e., the ability to pause, play and stop a video streaming, and the track bar may be used to jump (video seeking) to a different video time.
Figure 4.2: Windows Forms [UI] Application

Figure 4.3: Video Controls and Stream Information bar in the [UI] Application
For debugging purposes, an auto-scrolling log window was created to allow the user to follow application events in real-time.

**Windows Internet Explorer Plugin**

Figure 4.4 shows a screenshot of the Internet Explorer running the Player Plugin UI. The plugin only implements the Video Renderer, with all remaining controls done via Javascript or direct interaction with the UI controller process.

![Internet Explorer running the Player plugin](image)

Figure 4.4: Internet Explorer running the Player plugin

To maintain compatibility with the `<video>` tag available in HTML 5, the javascript code uses functions with the same name as defined in the forthcoming specification, with only a few specific functions but related to SVC (i.e., the number of layers being viewed).

**Mac OS X Cocoa application UI**

Figure 4.5 shows a screenshot of the Mac OS X Cocoa UI during play-out. Cocoa is Apple’s native object-oriented (Objective-C) application programming interfaces.

Except for the Cocoa User Interface, all the remaining code of the application (used for Windows), remained intact and was just compiled for Mac OS X.

### 4.3.2 UI Controller

The UI Controller component manages the information flow between the UI, the Stream Controller and the Decoder, being also responsible to inform the Adaptation System about changes in Video Window Resolution. This component triggers the start of the stream controller and the decoder.
4.3.3 Video Decoder

At present, and as far as known, there exists only one Open Source Media Player implementation, the OpenSVC MPlayer [1,32], that supports the H.264 SVC extension with reasonable performance.

This OpenSVC project consists on a library of decoding tools that was created at the Institut d’Electronique et des Telecommunications de Rennes (IETR)/Institut National des Sciences Appliquees de Rennes (INSA) of Rennes, aiming to support H.264 SVC in The media player MPlayer [32].

MPlayer is available for all major operating systems, including Linux and other Unix-like systems, Mac OS X and Microsoft Windows. There are also versions for Android and iOS. The MPlayer includes decoders for most of the known video formats, i.e., MPEG/VOB, AVI, Ogg/OGM, VIVO, ASF/WMA/WMV, QT/MOV/MP4, RealMedia, Matroska, NUT, NupelVideo, FLI, YUV4MPEG, FILM, RoQ and PVA files, and is supported by many native, XAnim, and Win32 DLL codecs. It is also capable of playing contents from VideoCD, SVCD, DVD, 3ivx, DivX 3/4/5, WMV and H.264 movies.

The media player module needs, as input, the handle (memory pointer) of a graphical window to inform the video decoder of the location to output the rendered video, one pipe to receive the encoded video implemented by means of a file and another pipe to perform trick operations operation over the media, such as Play, Pause or Stop.
4.3.4 Index XML Download/Parser

The XML Download/Parser is responsible for fetching the Index/Manifest File from an HTTP server and interpret/transform it so that the information there contained can be used by the various system components. The process is invoked by the stream controller which passes as argument the base URL of the XML file. The received file is transformed and the information converted into an object of Stream Information (Figure 4.6).

![Stream Information](image)

Figure 4.6: The Stream Information object with Stream Properties returned by the XML Parser

If the downloaded XML contains P2P information (PeerID, Tracker URL), the XML is sent to the (previously configured) external P2P Gateway over HTTP via a POST method. In this case, the information passed to the Stream Controller is changed in order to reflect the location of the external source of the media (the P2P Gateway instead of an HTTP web streaming server).

Figure 4.7 shows an example of the sequence of messages used to fetch the Index/manifest file from an HTTP web server.

4.3.5 Adaptation System

The Adaptation System is the component that determines at each moment the number of layers of a chunk to be requested.

The Adaptation System offers three functions, one to receive the video window size (called whenever that UI window is resized), another to receive information about download completion status, such as the duration of layer download, the size in bytes of the layer and the level of the layer, and a third function to receive the average value of RTT for the current session.

The values of layer size and expected download duration are stored in memory and are used to infer the available bandwidth and the average size of the layer (Download rate = File size/duration of download). Two additional functions are used to compute the expected time for downloading a specific layer/level, and to estimate the maximum number of layers to download. These functions are based on historic data collected.
4.3.6 Stream Controller

The Stream Controller class implements the core streaming functions of the system. The methods and attributes of the Stream Controller class are shown in Figure 4.8. The DownloadedChunks attribute indicates the ID of the latest downloaded chunk with the attribute CurrentLevel indicating the corresponding number of layers downloaded. The CacheSize corresponds to the number of chunks not yet played-out. The value of the CurrentPlayed attribute is defined by the UI Controller and is used to control the cache.

![Stream Controller Diagram]

Figure 4.8: Stream Controller Class

Figure 4.9 represents the Stream Controller class flow. The Stream controller is launched by the UI.
Controller with the information about the URL of the Index/Manifest file. This parameter is sent to the XML Download/Parser that returns the information about the corresponding media. One loop cares for chunk download sequence and another subsequent loop for the corresponding layers download (in the case of SVC).

![Stream Controller Flow Diagram](image)

**Figure 4.9: Stream Controller Flow**

The decision to proceed with the next layer uses the functions provided by the adaptation system to obtain the expected time for the download.

This heuristic prevents deciding for the download of a chunk if the time to download it would exceed its play-out duration.

This loop is only stopped at the end of the media or by a specific call to stop. If the user seeks the movie (any point in the timeline), the Stream Controller adjusts automatically the chunk number and clears the cache.

To increase the performance and not delaying the download process the chunk reconstruction was implemented in a different thread. While the reconstruction process is underway, in parallel, the next chunk is reassembled.
4.3.7 Layer Requester

The Layer Requester includes all the necessary code for downloading content chunks (chunk layers in the case of SVC) using HTTP.

For that purpose the Layer Requester computes the instantaneous RTT of each connection by measuring the time since the request was made up to the arrival of the first byte of the response (the code snippet in Listing 4.1). This value represents 2xRTT and is passed to the Adaptation System that returns the average of all RTTs, that is used to smooth the process avoiding abrupt changes.

```java
if ((TotalBytesToReceive / (BytesReceived / downloadTime)) <= (2*rtt)) {
    return true;
} else {
    return false;
}
```

For layer downloading a new thread is launched. That thread informs the main thread about the download progress. With the file size, RTT and the estimated download time, the main class can inform the Stream Controller of the moment when it can start the download of the next layer in order to optimize bandwidth usage.

4.3.8 Reassembler

The SVC Reassembler is used to assemble a variable number of layers files into the corresponding video chunk in order for this one to be rendered in the video window. The Sequence Number of the NAL field is used to restore the order of the data, being withdrawn from the received NALs afterwards.

The algorithm starts by reading from each layer file the respective content into a byte vector. Each of these vectors are parsed in order to isolate the NALs and Sequence Numbers. Sequence Number become the keys in a new array that point to the corresponding NAL.

The array is then sorted by Sequence Number and the corresponding content can then be written to the buffer by the correct sequence.

4.3.9 Configuration

A Configuration class was implemented to allow configuring the operating mode of the player and facilitate testing and debugging.

Figure 4.10 shows the main configuration UI window.
The available options are:

- **Duration of chunk**: Set the duration of each chunk. Allows, for example, to test the system with chunks of variable duration.

- **Adaptation system**: Enable/disable partially or completely the Adaptation System, Play-out mode: Allows to set the video decoder and the working mode for either video Play-out or Simulation (video is not rendered).

- **P2P**: Set the P2P Gateway External interface (URL).

- **Log**: Set log options, log file location, etc.

- **Cache**: Set the cache size and location. Allows to store the downloaded layers in the local disk.

![Configuration Form](image)

**Figure 4.10: Configuration Form**

### 4.4 The Internet TV Distribution/Server components

For the Distribution/Server side of the architecture, in order to create a working solution, some prototype components were also developed, namely the encoding process and the Partition component. For the encoder, the reference encoder code was used [41]. The JSVM (Joint Scalable Video Model) software is the reference software for the SVC project of the Joint Video Team (JVT) of the ISO/IEC Moving Pictures Experts Group (MPEG) and the ITU-T Video Coding Experts Group (VCEG). The JSVM
Software is still under development and has low performance in encoding, not being suitable for real time video encoding. All of the media files used to test the solution were encoded with two levels of spatial resolution-definition, two levels for quality and three temporal dimension levels, which is represented in Table 3.1 from Chapter 3.

The Partition system was developed as a set of MATLAB scripts in order to generate the chunks and then the layer files.

### 4.5 P2P Gateway/P2P Engine

The [P2P](#) Gateway/Engine provides WebSeeding capabilities to other peer nodes, behaving as both a client and a server in the [P2P](#) network. From the perspective of the Internet TV Player it behaves as a (local) [HTTP](#)-based web streaming server. This component is being developed in the scope of the SARACEN project [9,38] and its implementation will not be described in this thesis.
5 Evaluation

Contents

5.1 Introduction .............................................. 48
5.2 Evaluation Objectives and Experimental Scenarios ............. 48
5.3 Evaluation Results ........................................ 50
5.1 Introduction

This chapter describes the evaluation process of the SVC Player prototype. The evaluation objectives, experimental scenarios and metrics used are detailed, followed by an explanation of the evaluation tests. The test results are then described, and conclusions on the implementation are also drawn.

5.2 Evaluation Objectives and Experimental Scenarios

The goals of the evaluation process are to assess the prototype behavior in heterogeneous access networks, evaluate the streaming capabilities of the solution and the functionalities it provides. Several tests were planned in order to measure the streaming quality adaptation of the system and assess the impact on perceived image and quality.

To properly evaluate the system a controlled environment was created in order to test the scenarios in which the solution may be used.

The test environment considered an HTTP web streaming server and multiple instances of the Internet TV Player. The web streaming server, was a standard web server, where the SVC encoded videos were made available, either as stored contents or as real-time encoded media chunk streams (simulating a Live TV program), together with the corresponding index/manifest files. The web streaming server was located at the INESC Inovação (INOV) Network premises, with public address accessible from the Internet.

Figure 5.1: Test environment
The Internet TV Player was available in several PC-based platforms with operating systems Microsoft Windows XP/7, Apple Mac OS X and Linux (Ubuntu distribution), as either a stand-alone application (Windows and Mac) or as a web browser plugin (Windows and Linux).

The [SVC] videos were encoded with ten layers for two spatial scalability levels. The first five Layers with a Common Intermediate Format [CIF] resolution and the other five Layers with a Double Common Intermediate Format [DCIF] resolution. The original video size, with all layers had the approximate size of 22 Megabyte (MB). After partition, the average file sizes for the different layers of each level were as listed in Table 5.1.

In order to have the ability of simulating/forcing fluctuations in network conditions for fixed network, a gateway allowing variations in the available bandwidth was interposed between the local network of the client computers and the Internet access. The gateway consisted in a Linux system where specific scripts were used for packet shaping on the network interfaces. The test environment is represented in Figure 5.1.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Average Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>27.0 kB</td>
</tr>
<tr>
<td>2</td>
<td>31.0 kB</td>
</tr>
<tr>
<td>3</td>
<td>6.0 kB</td>
</tr>
<tr>
<td>4</td>
<td>5.5 kB</td>
</tr>
<tr>
<td>5</td>
<td>5.5 kB</td>
</tr>
<tr>
<td>6</td>
<td>50.0 kB</td>
</tr>
<tr>
<td>7</td>
<td>61.0 kB</td>
</tr>
<tr>
<td>8</td>
<td>8.4 kB</td>
</tr>
<tr>
<td>9</td>
<td>8.5 kB</td>
</tr>
<tr>
<td>10</td>
<td>10.0 kB</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>212.9 kB</strong></td>
</tr>
</tbody>
</table>

Three scenarios were considered: a fixed network scenario in a client-server mode, a mobile (3G Network) also in client-server mode, with client computers connecting via a 3G wireless modem and a [P2P] scenario in a fixed network.

For the [P2P] scenario the web streaming server also run a tracker service controlling the peer clients (running as local proxies in the client computers). For the fixed network environment three bandwidth levels were prepared (via the packet shaping scripts) - 100 Mbit/s, 1 Mbit/s and 256 kbit/s - allowing also the simulation of a sudden and severe network congestion situation by a drop from 100 Mbit/s to
256 kbit/s.
The metrics used in each test measured the Bandwidth/Network Load, the RTT, the Cache size, the PSNR. For each relevant test the layer variation during streaming was also collected.

For a score reference on the perceived quality of the received media after compression and/or transmission, during the analysis of the results the following relationship between PSNR and MOS will be used (Table 5.2).

<table>
<thead>
<tr>
<th>PSNR</th>
<th>MOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 37</td>
<td>5    (Excellent)</td>
</tr>
<tr>
<td>31-37</td>
<td>4    (Good)</td>
</tr>
<tr>
<td>25-31</td>
<td>3    (Fair)</td>
</tr>
<tr>
<td>20-25</td>
<td>2    (Poor)</td>
</tr>
<tr>
<td>&lt; 20</td>
<td>1    (Bad)</td>
</tr>
</tbody>
</table>

The evaluation suite was composed by the following tests:

- Network streaming behavior:
  - analysis of downloaded layers of the video, measuring the stream adaptation to network conditions;
  - network load adaptation, measuring the load as per available bandwidth;
  - network latency, determining the RTT for different network conditions;
  - cache control, measuring the cache fullness for different network conditions;

Player capabilities behavior:

- analysis of stream adaptation to screen resolution, measuring the network load for variations in video screen resolution;
- image quality, computing the PSNR for the received video;

5.3 Evaluation Results

5.3.1 100 Mbit/s Local Bandwidth

The measurements were performed at full local network bandwidth (100 Mbit/s). This is the reference test, since the bandwidth available outperforms the bandwidth necessary for normally streaming the video.
As expected, at this bandwidth the maximum number of layers was immediately reached, with even moments where no bandwidth was used, as may be observed in Figure 5.2.

![Network Load at 100 Mbit/s](image)

**Figure 5.2: Network Load at 100 Mbit/s**

Latency was also very reduced, with RTT low almost all the time (Figure 5.4) and the cache reacting immediately by downloading a new chunk as soon as the current decoded chunk finishes playing.

![RTT at 100 Mbit/s](image)

**Figure 5.3: RTT at 100 Mbit/s**

Since the received video stream was complete (all chunks with 10 layers), it was identical to the source video, and so, the PSNR was not computed.
5.3.2 1Mbit/s Local Bandwidth

A limited bandwidth network is emulated by the 1 Mbit/s packet shaping. At 1 Mbit/s the maximum download rate reaches 122 kB/s. With an average chunk size of 212.0 kB, as shown in Table 5.1 and a video chunk duration of 2 seconds, it would be necessary, for maximum quality, an average of 106.45 kB downloaded per second (not considering the bandwidth consumed by the various protocols active in the network).

As the available bandwidth is close to the minimum required (Figure 5.7) the video can still be received at the maximum quality almost all the time, with the network almost always at maximum occupation. Due to variation in chunk/layer sizes, at some moments the cache is completely full and the download is momentarily paused. However, at other moments, it is not possible to download all the layers of a chunk as it may be clearly observed in Figure 5.5. This reduction in the number of layers causes variations in video quality, as evidenced by the PSNR values in Figure 5.6.
Figure 5.6: PSNR at 1 Mbit/s

Figure 5.7: Network Load at 1 Mbit/s

Figure 5.8: RTT at 1 Mbit/s
For this limited bandwidth network the RTT remains low and only increases at those moments when the system is unable to get all the layers of a chunk, due to congestion caused by the download of the layers (and bandwidth exhaustion).

Figure 5.9: Cache at 1 Mbit/s

Comparing the PSNR values plotted in Figure 5.6 with the MOS scores in Table 5.2 it is clear that the global perceived quality of the video received is considered very good, almost the same as the original video.

A remarkable fact is that despite the congestion of the channel the system keeps the maximum level for spatial scalability, with only a few moments of small reduction in the video quality.

5.3.3 256 kbit/s Local Bandwidth

The purpose of this test is to evaluate the system behavior in a condition where the bandwidth is about four times lower than the bandwidth needed to play the original video at full quality.

Figure 5.10: Downloaded Layers at 256 kbit/s
At 256 kbit/s the maximum download rate reaches 32 kBytes/s. With the first three layers of the video chunks summing up to an average size of 64 kB (Table 5.1), practically coinciding with the available bandwidth. But, as more bandwidth is necessary for protocols used in the communication, it is almost certain that the received video will hardly have more than 5 layers (as the spatial resolution increases precisely at the 6th layer).

As expected the results confirm (Figure 5.10) that the movie was played-out mostly with 1 layer, eventually reaching 5 layers due to the lower size of layers 3, 4 and 5. Additionally, whenever the system is able to get layer 2, then it will almost be able to reach up to layer 5.

The PSNR measured (Figure 5.13) shows that even with 1/4 of the bandwidth required to play the original video the quality is quite acceptable. The average of PSNR is near 30 dB, a MOS of 3 (Fair). The cache fullness (Figure 5.14) varied along the movie with an average of three chunks. This was due to the low bandwidth available and the adaptation algorithm system that always tries to offer the best possible quality.
Figure 5.13: RTT at 256 kbit/s

Figure 5.14: Cache at 256 kbit/s
5.3.4 100 Mbit/s to 256 kbit/s Sudden Drop

The purpose of this test was to emulate a moment of serious congestion on the network. The bandwidth variation, from 100 Mbit/s to mere 256 kbit/s, allowed to test the adaptation capabilities of the solution to fluctuations in network capacity.

![Figure 5.15: Downloaded Layers during a 100 Mbit/s to 256 kbit/s drop](image1)

The test started with the maximum bandwidth until arriving to second $t = 120$ when the bandwidth was suddenly dropped to the minimum (Figure 5.17). The results show, for $t < 120$, that the system does not occupies a constant bandwidth during the video stream, but has a spiky nature due to the small size of the video chunks (of 2 seconds) that are downloaded faster than their play-out duration. The RTT value (Figure 5.18) was also fairly small during that first period (around 10 ms) and the video could be watched with maximum quality.

At the instant $t = 120$, the system detects the variation in networks conditions and automatically adapts the number of layers to be requested from the server to the available bandwidth. From that moment onwards, the system uses the full available bandwidth continuously, and still manages at a few moments to download up to layer level 5, due to the variable size of the layer files (above the base layer) of each chunk. The value of the RTT rises dramatically due to the congestion.

The image quality also suffers a noticeable degradation with the higher layers dropping. The effect is
Figure 5.17: Network Load at a 100 Mbit/s to 256 kbit/s drop

Figure 5.18: RTT at a 100 Mbit/s to 256 kbit/s drop
even more dramatic because of the change in spatial resolution of the video. However, after the moment of bandwidth drop the image remain with a stable quality level.

![Cache behavior at a 100 Mbit/s to 256 kbit/s drop](image.png)

Figure 5.19: Cache behavior at a 100 Mbit/s to 256 kbit/s drop

Another important result is that the system manages to support severe drops in bandwidth without creating pauses or re-buffering.

### 5.3.5 3G Mobile Network

This test evaluates the system behavior in a harsh environment, not controlled, typically with deep variations in network conditions.

![Downloaded Layers in 3G Mobile Network](image.png)

Figure 5.20: Downloaded Layers in 3G Mobile Network

Due to poor signal conditions and low bandwidth (and high latency typical from this type of network) the system was only able to download the base layer of the video. However the system was still able to maintain stable levels of cache and a constant video quality.
Figure 5.21: PSNR in 3G Mobile Network

Figure 5.22: Network Load in 3G Mobile Network

Figure 5.23: RTT in 3G Mobile Network
5.3.6 Screen Resolution

A simple test was performed in order to demonstrate the terminal capabilities adaptation advantages. Figure 5.25 shows the variations in network load in reaction to variations in screen size. In that figure, at $t = 75$ seconds the bandwidth usage is reduced to $1/2$ after a resizing of the video window, and at $t = 150$ seconds it increases again when the video window was restored to full size.

This adaptation occurs because the system knows the different resolutions the video contains and always tries to display the most appropriate resolution (i.e., have the adequate layers) for the current video window size. This is due to the spatial scalability feature of SVC.
Conclusion and future work

Contents

6.1  System limitations and future work ........................................ 64
6.2  Conclusion ........................................................................... 64
6.1 System limitations and future work

The developed solution presents some limitations in features that, with future work, may be corrected and even enhanced. Some of the features considered in the SARACEN project that would bring improvements in some aspects, were not considered fundamental at this stage, and were not implemented. Concerning the limitations of the current solution, the most critical are those related to the server side components and those related with the P2P transmission:

- The SVC encoder needs to be improved, as the reference encoder has very low performance making impossible to encode video in real time.

- The partition system needs to be developed and optimized in order to turn the whole process (encoding, chunk partition and Layer partition) automatic and tunable.

- Due to some limitations in the SVC decoder, the adequate temporal scalability was not supported yet for the prototype solution. The decoder is still under development and does not include compensation mechanisms for faulty layers.

- The prototype does not include audio support yet. However, this support would be easy to implement and only for lack of time it could not be done. The Manifest File was designed to support multiple media type, as video, audio, text, subtitles, associated metadata, etc.

- Additional adaptation mechanisms based in machine conditions, also need to be developed and incorporated. Only the screen size heuristic has been explored. The prototype system architecture is prepared to allow more heuristics to be incorporated.

- Due to the limitation of the P2P component, the prototype system would be more efficient in network usage.

6.2 Conclusion

This thesis describes an Internet TV architecture supporting the distribution of video over heterogeneous networks, providing a very stable streaming solution. The solution incorporates quality control mechanisms to allow video play-out without pauses, stuttering or image artifacts, by smoothly minimizing variations from network and system conditions.

In Chapter 2 the related technologies were presented. These include the current technologies supporting adaptive mechanisms and the mechanisms involved in their development. The SVC codec was also studied for its advantages in video adaptation.
Chapter 3 described the design and implementation of the Internet TV architecture, including a summary of the main functional and non-functional requirements. The architecture was decomposed of its content distribution and client sides and their interaction presented. In the content distribution side the process for the SVC video division into chunks and layers and the respective XML/Manifest creation was described. In the client side, the Video Player prototype was also described on its component as well as the mechanisms and techniques to ensure a good adaptation response for changing conditions. This chapter also described the support for both VoD and Live real time video streaming and the external interfaces that give the system the capability to support various HTTP-based streaming modes.

Chapter 4 described the prototype development, the languages and technologies used and the implementation of the various components. Three prototypes were presented, a Windows Player application, a Mac OS X Player application and an Internet Explorer Plugin Player. It has also bee described how the solution was engineered in order to meet the requirements and the extra features that were created to produce a functional prototype.

Finally, in Chapter 5 the solution was validated by means of a suite of tests, ran on distinct network conditions, both in laboratory and in field (for 3G). The main focus of the evaluation was on the Adaptation System and the quality of the received video.

The developed solution proved to be robust and stable, demonstrating very satisfactory results.
Bibliography


[10] P. Moscoso, R. S. Cruz, and M. S. Nunes, “Internet TV Architecture Based on Scalable Video Coding,” in Proceedings of the 9th European Interactive TV Conference Workshop on Quality of


Appendix A

Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1</td>
<td>PSNR of the encoded video layers</td>
<td>72</td>
</tr>
<tr>
<td>A.2</td>
<td>Log Analyzer</td>
<td>72</td>
</tr>
<tr>
<td>A.3</td>
<td>Video Encoding Log</td>
<td>73</td>
</tr>
</tbody>
</table>
A.1 **PSNR** of the encoded video layers

Figure A.1 presents the **PSNR** between the various layers of the original **SVC** encoded video used for the tests.

![Figure A.1: PSNR between layers from video used in tests.](image)

A.2 Log Analyzer

To assist in layer creation, an application was developed to analyze the log files from the encoder. Figure A.2 shows a screenshot of the log analyzer.

![Figure A.2: Log Analyzer](image)
A.3 Video Encoding Log

The video encoding log is presented in Listing A.1

Listing A.1: Output log from encoder

<table>
<thead>
<tr>
<th>AU</th>
<th>DQ</th>
<th>T</th>
<th>L</th>
<th>Q</th>
<th>QP</th>
<th>Y</th>
<th>U</th>
<th>V</th>
<th>bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:</td>
<td>0</td>
<td>I</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>38.8290</td>
<td>41.2536</td>
<td>42.9609</td>
<td>44056</td>
</tr>
<tr>
<td>0:</td>
<td>1</td>
<td>I</td>
<td>0</td>
<td>1</td>
<td>24</td>
<td>42.7758</td>
<td>44.6593</td>
<td>46.2843</td>
<td>43440</td>
</tr>
<tr>
<td>0:</td>
<td>0</td>
<td>I</td>
<td>1</td>
<td>0</td>
<td>34</td>
<td>38.7403</td>
<td>41.5565</td>
<td>43.4643</td>
<td>74000</td>
</tr>
<tr>
<td>0:</td>
<td>1</td>
<td>I</td>
<td>1</td>
<td>1</td>
<td>26</td>
<td>43.9843</td>
<td>45.7546</td>
<td>47.4632</td>
<td>90592</td>
</tr>
<tr>
<td>8:</td>
<td>0</td>
<td>P</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>34.9219</td>
<td>40.2750</td>
<td>40.2977</td>
<td>110376</td>
</tr>
<tr>
<td>8:</td>
<td>1</td>
<td>P</td>
<td>0</td>
<td>1</td>
<td>24</td>
<td>39.5840</td>
<td>43.4935</td>
<td>43.6228</td>
<td>98216</td>
</tr>
<tr>
<td>8:</td>
<td>0</td>
<td>P</td>
<td>1</td>
<td>0</td>
<td>34</td>
<td>32.9548</td>
<td>39.7952</td>
<td>40.2263</td>
<td>241896</td>
</tr>
<tr>
<td>8:</td>
<td>1</td>
<td>P</td>
<td>1</td>
<td>1</td>
<td>26</td>
<td>38.8611</td>
<td>44.4395</td>
<td>44.6925</td>
<td>315928</td>
</tr>
<tr>
<td>4:</td>
<td>0</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>33</td>
<td>33.5315</td>
<td>40.0628</td>
<td>40.0107</td>
<td>2856</td>
</tr>
<tr>
<td>4:</td>
<td>1</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>27</td>
<td>37.2636</td>
<td>43.2503</td>
<td>43.2942</td>
<td>10000</td>
</tr>
<tr>
<td>4:</td>
<td>0</td>
<td>B</td>
<td>1</td>
<td>0</td>
<td>37</td>
<td>31.6844</td>
<td>39.6740</td>
<td>40.2011</td>
<td>11816</td>
</tr>
<tr>
<td>4:</td>
<td>1</td>
<td>B</td>
<td>1</td>
<td>1</td>
<td>29</td>
<td>36.6733</td>
<td>44.1442</td>
<td>44.4490</td>
<td>47048</td>
</tr>
<tr>
<td>2:</td>
<td>0</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>35</td>
<td>33.0161</td>
<td>39.9894</td>
<td>39.8980</td>
<td>1680</td>
</tr>
<tr>
<td>2:</td>
<td>1</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>29</td>
<td>36.5210</td>
<td>43.1434</td>
<td>43.0521</td>
<td>4320</td>
</tr>
<tr>
<td>2:</td>
<td>0</td>
<td>B</td>
<td>1</td>
<td>0</td>
<td>39</td>
<td>31.2276</td>
<td>39.5556</td>
<td>40.1009</td>
<td>7144</td>
</tr>
<tr>
<td>2:</td>
<td>1</td>
<td>B</td>
<td>1</td>
<td>1</td>
<td>31</td>
<td>35.7784</td>
<td>43.8284</td>
<td>44.1510</td>
<td>21864</td>
</tr>
<tr>
<td>1:</td>
<td>0</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>36</td>
<td>32.5919</td>
<td>39.9575</td>
<td>39.8423</td>
<td>744</td>
</tr>
<tr>
<td>1:</td>
<td>1</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>30</td>
<td>36.0512</td>
<td>43.0443</td>
<td>42.9436</td>
<td>2736</td>
</tr>
<tr>
<td>1:</td>
<td>0</td>
<td>B</td>
<td>1</td>
<td>0</td>
<td>40</td>
<td>30.7822</td>
<td>39.4531</td>
<td>39.9685</td>
<td>3128</td>
</tr>
<tr>
<td>1:</td>
<td>1</td>
<td>B</td>
<td>1</td>
<td>1</td>
<td>32</td>
<td>34.9784</td>
<td>43.5744</td>
<td>43.8365</td>
<td>9632</td>
</tr>
<tr>
<td>3:</td>
<td>0</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>36</td>
<td>33.2217</td>
<td>39.9888</td>
<td>39.9415</td>
<td>240</td>
</tr>
<tr>
<td>3:</td>
<td>1</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>30</td>
<td>36.8192</td>
<td>43.1799</td>
<td>43.1773</td>
<td>624</td>
</tr>
<tr>
<td>3:</td>
<td>0</td>
<td>B</td>
<td>1</td>
<td>0</td>
<td>40</td>
<td>31.3010</td>
<td>39.5789</td>
<td>40.1394</td>
<td>1384</td>
</tr>
<tr>
<td>3:</td>
<td>1</td>
<td>B</td>
<td>1</td>
<td>1</td>
<td>32</td>
<td>35.7477</td>
<td>43.9125</td>
<td>44.2606</td>
<td>4736</td>
</tr>
<tr>
<td>6:</td>
<td>0</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>35</td>
<td>34.0780</td>
<td>40.1923</td>
<td>40.1860</td>
<td>232</td>
</tr>
<tr>
<td>6:</td>
<td>1</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>29</td>
<td>37.8743</td>
<td>43.3706</td>
<td>43.4964</td>
<td>776</td>
</tr>
<tr>
<td>AU</td>
<td>B</td>
<td>T</td>
<td>L</td>
<td>Q0</td>
<td>QP</td>
<td>Y</td>
<td>U</td>
<td>V</td>
<td>Bits</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>--------</td>
</tr>
<tr>
<td>6:</td>
<td>T2 L1</td>
<td>Q0</td>
<td>QP 39</td>
<td>32.0227</td>
<td>39.7806</td>
<td>40.2722</td>
<td>2664 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6:</td>
<td>T2 L1</td>
<td>Q1</td>
<td>QP 31</td>
<td>36.6949</td>
<td>44.4416</td>
<td>44.6487</td>
<td>10984 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5:</td>
<td>T3 L0</td>
<td>Q0</td>
<td>QP 36</td>
<td>33.8636</td>
<td>40.11137</td>
<td>40.1194</td>
<td>136 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5:</td>
<td>T3 L0</td>
<td>Q1</td>
<td>QP 30</td>
<td>37.5758</td>
<td>43.2699</td>
<td>43.3720</td>
<td>384 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5:</td>
<td>T3 L1</td>
<td>Q0</td>
<td>QP 40</td>
<td>31.8780</td>
<td>39.7369</td>
<td>40.2638</td>
<td>1152 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5:</td>
<td>T3 L1</td>
<td>Q1</td>
<td>QP 32</td>
<td>36.5579</td>
<td>44.3647</td>
<td>44.6073</td>
<td>5096 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7:</td>
<td>T3 L0</td>
<td>Q0</td>
<td>QP 36</td>
<td>34.3740</td>
<td>40.2397</td>
<td>40.2495</td>
<td>152 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7:</td>
<td>T3 L0</td>
<td>Q1</td>
<td>QP 30</td>
<td>38.2900</td>
<td>43.4330</td>
<td>43.5438</td>
<td>208 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7:</td>
<td>T3 L1</td>
<td>Q0</td>
<td>QP 40</td>
<td>32.2359</td>
<td>39.8043</td>
<td>40.3078</td>
<td>992 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7:</td>
<td>T3 L1</td>
<td>Q1</td>
<td>QP 32</td>
<td>36.9725</td>
<td>44.7700</td>
<td>44.7410</td>
<td>44960 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16:</td>
<td>P</td>
<td>T0 L0</td>
<td>Q0</td>
<td>30 Y</td>
<td>34.5761</td>
<td>40.5711</td>
<td>40.6245</td>
<td>19120 bit</td>
<td></td>
</tr>
<tr>
<td>16:</td>
<td>P</td>
<td>T0 L0</td>
<td>Q1</td>
<td>24 Y</td>
<td>39.4867</td>
<td>44.0382</td>
<td>44.1792</td>
<td>44960 bit</td>
<td></td>
</tr>
<tr>
<td>16:</td>
<td>P</td>
<td>T0 L1</td>
<td>Q0</td>
<td>34 Y</td>
<td>32.8056</td>
<td>40.5461</td>
<td>40.9011</td>
<td>64000 bit</td>
<td></td>
</tr>
<tr>
<td>16:</td>
<td>P</td>
<td>T0 L1</td>
<td>Q1</td>
<td>28 Y</td>
<td>37.4892</td>
<td>43.6309</td>
<td>43.6871</td>
<td>768 bit</td>
<td></td>
</tr>
<tr>
<td>12:</td>
<td>B T1 L0</td>
<td>Q0</td>
<td>QP 33</td>
<td>34.3679</td>
<td>40.4989</td>
<td>40.3987</td>
<td>1256 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12:</td>
<td>B T1 L0</td>
<td>Q1</td>
<td>QP 27</td>
<td>38.4746</td>
<td>43.8018</td>
<td>43.7434</td>
<td>3832 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12:</td>
<td>B T1 L1</td>
<td>Q0</td>
<td>QP 29</td>
<td>37.5295</td>
<td>44.9932</td>
<td>45.0977</td>
<td>72312 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10:</td>
<td>B T2 L0</td>
<td>Q0</td>
<td>QP 35</td>
<td>34.5082</td>
<td>40.3799</td>
<td>40.4228</td>
<td>344 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10:</td>
<td>B T2 L0</td>
<td>Q1</td>
<td>QP 29</td>
<td>38.4992</td>
<td>43.6309</td>
<td>43.6871</td>
<td>768 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10:</td>
<td>B T2 L1</td>
<td>Q0</td>
<td>QP 39</td>
<td>32.5339</td>
<td>40.1024</td>
<td>40.5762</td>
<td>3000 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10:</td>
<td>B T2 L1</td>
<td>Q1</td>
<td>QP 31</td>
<td>37.4014</td>
<td>43.8191</td>
<td>45.0174</td>
<td>11040 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9:</td>
<td>B T3 L0</td>
<td>Q0</td>
<td>QP 36</td>
<td>34.7245</td>
<td>40.3753</td>
<td>40.4143</td>
<td>160 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9:</td>
<td>B T3 L0</td>
<td>Q1</td>
<td>QP 30</td>
<td>38.9632</td>
<td>43.6317</td>
<td>43.7452</td>
<td>240 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9:</td>
<td>B T3 L1</td>
<td>Q0</td>
<td>QP 40</td>
<td>32.7628</td>
<td>40.0088</td>
<td>40.4506</td>
<td>904 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9:</td>
<td>B T3 L1</td>
<td>Q1</td>
<td>QP 32</td>
<td>38.0134</td>
<td>44.6696</td>
<td>44.8730</td>
<td>4248 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11:</td>
<td>B T3 L0</td>
<td>Q0</td>
<td>QP 36</td>
<td>34.3875</td>
<td>40.4398</td>
<td>40.4219</td>
<td>224 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11:</td>
<td>B T3 L0</td>
<td>Q1</td>
<td>QP 30</td>
<td>38.4016</td>
<td>43.6738</td>
<td>43.6824</td>
<td>448 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11:</td>
<td>B T3 L1</td>
<td>Q0</td>
<td>QP 40</td>
<td>32.3894</td>
<td>40.2153</td>
<td>40.6609</td>
<td>1032 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11:</td>
<td>B T3 L1</td>
<td>Q1</td>
<td>QP 32</td>
<td>37.2581</td>
<td>44.8783</td>
<td>45.0786</td>
<td>4792 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14:</td>
<td>B T2 L0</td>
<td>Q0</td>
<td>QP 35</td>
<td>34.3189</td>
<td>40.5214</td>
<td>40.4608</td>
<td>776 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14:</td>
<td>B T2 L0</td>
<td>Q1</td>
<td>QP 29</td>
<td>38.5189</td>
<td>43.8630</td>
<td>43.7235</td>
<td>1568 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14:</td>
<td>B T2 L1</td>
<td>Q0</td>
<td>QP 39</td>
<td>32.3927</td>
<td>40.5012</td>
<td>40.8390</td>
<td>3528 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14:</td>
<td>B T2 L1</td>
<td>Q1</td>
<td>QP 31</td>
<td>37.4309</td>
<td>45.1299</td>
<td>45.2980</td>
<td>12696 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13:</td>
<td>B T3 L0</td>
<td>Q0</td>
<td>QP 36</td>
<td>34.3038</td>
<td>40.5383</td>
<td>40.4329</td>
<td>408 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13:</td>
<td>B T3 L0</td>
<td>Q1</td>
<td>QP 30</td>
<td>38.3689</td>
<td>43.8537</td>
<td>43.6655</td>
<td>704 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13:</td>
<td>B T3 L1</td>
<td>Q0</td>
<td>QP 40</td>
<td>32.2977</td>
<td>40.4325</td>
<td>40.7601</td>
<td>1560 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13:</td>
<td>B T3 L1</td>
<td>Q1</td>
<td>QP 32</td>
<td>36.9952</td>
<td>45.0406</td>
<td>45.1362</td>
<td>6312 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15:</td>
<td>B T3 L0</td>
<td>Q0</td>
<td>QP 36</td>
<td>34.2715</td>
<td>40.5177</td>
<td>40.4723</td>
<td>440 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15:</td>
<td>B T3 L0</td>
<td>Q1</td>
<td>QP 30</td>
<td>38.4194</td>
<td>43.8479</td>
<td>43.7512</td>
<td>832 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15:</td>
<td>B T3 L1</td>
<td>Q0</td>
<td>QP 40</td>
<td>32.4184</td>
<td>40.4697</td>
<td>40.8022</td>
<td>1808 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15:</td>
<td>B T3 L1</td>
<td>Q1</td>
<td>QP 32</td>
<td>37.4694</td>
<td>45.1176</td>
<td>45.2447</td>
<td>6256 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24:</td>
<td>P T0 L0</td>
<td>Q0</td>
<td>QP 30</td>
<td>34.5120</td>
<td>40.8306</td>
<td>40.9420</td>
<td>19464 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24:</td>
<td>P T0 L1</td>
<td>Q0</td>
<td>QP 24</td>
<td>39.6111</td>
<td>44.3409</td>
<td>44.6175</td>
<td>48720 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24:</td>
<td>P T0 L1</td>
<td>Q1</td>
<td>QP 34</td>
<td>32.8985</td>
<td>41.1818</td>
<td>41.4625</td>
<td>45848 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame</td>
<td>MB Type</td>
<td>MB Size</td>
<td>QP</td>
<td>Luma Y</td>
<td>Chroma U</td>
<td>Chroma V</td>
<td>Bit Rate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
<td>----------</td>
<td>-----</td>
<td>--------</td>
<td>-----------</td>
<td>-----------</td>
<td>----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>24: P</td>
<td>T0 L1 Q1</td>
<td>26</td>
<td>39.2737</td>
<td>45.5582</td>
<td>45.8490</td>
<td>138032</td>
<td></td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>20: B</td>
<td>T1 L0 Q0</td>
<td>33</td>
<td>34.2211</td>
<td>40.7568</td>
<td>40.6944</td>
<td>1592</td>
<td></td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>20: B</td>
<td>T1 L0 Q1</td>
<td>27</td>
<td>38.3794</td>
<td>44.2140</td>
<td>44.1551</td>
<td>3544</td>
<td></td>
<td></td>
</tr>
<tr>
<td>87</td>
<td>20: B</td>
<td>T1 L1 Q0</td>
<td>37</td>
<td>32.4335</td>
<td>40.9577</td>
<td>41.3134</td>
<td>8752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>20: B</td>
<td>T1 L1 Q1</td>
<td>29</td>
<td>37.5923</td>
<td>45.4543</td>
<td>45.6186</td>
<td>29776</td>
<td></td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>18: B</td>
<td>T2 L0 Q0</td>
<td>35</td>
<td>34.2486</td>
<td>40.6639</td>
<td>40.6142</td>
<td>696</td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>18: B</td>
<td>T2 L0 Q1</td>
<td>29</td>
<td>38.3323</td>
<td>44.0673</td>
<td>43.9686</td>
<td>984</td>
<td></td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>18: B</td>
<td>T2 L1 Q0</td>
<td>39</td>
<td>32.4011</td>
<td>40.7510</td>
<td>41.1076</td>
<td>3576</td>
<td></td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>18: B</td>
<td>T2 L1 Q1</td>
<td>31</td>
<td>37.5914</td>
<td>45.3110</td>
<td>45.5566</td>
<td>13480</td>
<td></td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>18: B</td>
<td>T3 L0 Q0</td>
<td>36</td>
<td>34.1245</td>
<td>40.8033</td>
<td>40.6412</td>
<td>648</td>
<td></td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>18: B</td>
<td>T3 L0 Q1</td>
<td>30</td>
<td>38.1456</td>
<td>44.0664</td>
<td>43.9520</td>
<td>520</td>
<td></td>
<td></td>
</tr>
<tr>
<td>95</td>
<td>18: B</td>
<td>T3 L1 Q0</td>
<td>40</td>
<td>32.2696</td>
<td>40.8061</td>
<td>41.2008</td>
<td>1448</td>
<td></td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>18: B</td>
<td>T3 L1 Q1</td>
<td>32</td>
<td>37.0200</td>
<td>45.3336</td>
<td>45.4765</td>
<td>6520</td>
<td></td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>19: B</td>
<td>T3 L0 Q0</td>
<td>36</td>
<td>34.1649</td>
<td>40.6664</td>
<td>40.5877</td>
<td>208</td>
<td></td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>19: B</td>
<td>T3 L0 Q1</td>
<td>30</td>
<td>38.1456</td>
<td>44.0664</td>
<td>43.9520</td>
<td>520</td>
<td></td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>19: B</td>
<td>T3 L1 Q0</td>
<td>40</td>
<td>32.2696</td>
<td>40.8061</td>
<td>41.2008</td>
<td>1448</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>19: B</td>
<td>T3 L1 Q1</td>
<td>32</td>
<td>37.2288</td>
<td>45.5346</td>
<td>45.7546</td>
<td>13728</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>21: B</td>
<td>T3 L0 Q0</td>
<td>36</td>
<td>34.0630</td>
<td>40.7866</td>
<td>40.6722</td>
<td>248</td>
<td></td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>21: B</td>
<td>T3 L0 Q1</td>
<td>30</td>
<td>38.1260</td>
<td>44.1807</td>
<td>44.1177</td>
<td>744</td>
<td></td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>21: B</td>
<td>T3 L1 Q0</td>
<td>40</td>
<td>32.2636</td>
<td>40.9904</td>
<td>41.3358</td>
<td>1696</td>
<td></td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>21: B</td>
<td>T3 L1 Q1</td>
<td>32</td>
<td>37.2024</td>
<td>45.4566</td>
<td>45.6629</td>
<td>5680</td>
<td></td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>23: B</td>
<td>T3 L0 Q0</td>
<td>36</td>
<td>34.1748</td>
<td>40.7804</td>
<td>40.7008</td>
<td>624</td>
<td></td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>23: B</td>
<td>T3 L0 Q1</td>
<td>30</td>
<td>38.5005</td>
<td>44.1437</td>
<td>44.1825</td>
<td>880</td>
<td></td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>23: B</td>
<td>T3 L1 Q0</td>
<td>40</td>
<td>32.5148</td>
<td>41.1193</td>
<td>41.4062</td>
<td>2096</td>
<td></td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>23: B</td>
<td>T3 L1 Q1</td>
<td>32</td>
<td>37.7046</td>
<td>45.5973</td>
<td>45.7909</td>
<td>7032</td>
<td></td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>23: I</td>
<td>T0 L0 Q0</td>
<td>30</td>
<td>35.0024</td>
<td>40.1057</td>
<td>40.1827</td>
<td>106944</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>23: I</td>
<td>T0 L0 Q1</td>
<td>24</td>
<td>39.6125</td>
<td>43.5272</td>
<td>43.6922</td>
<td>95784</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>23: I</td>
<td>T0 L1 Q0</td>
<td>34</td>
<td>33.0868</td>
<td>40.0461</td>
<td>40.3365</td>
<td>224640</td>
<td></td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>23: I</td>
<td>T0 L1 Q1</td>
<td>26</td>
<td>38.7649</td>
<td>44.1513</td>
<td>44.3478</td>
<td>300208</td>
<td></td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>28: B</td>
<td>T1 L0 Q0</td>
<td>33</td>
<td>34.2959</td>
<td>40.7275</td>
<td>40.3785</td>
<td>2456</td>
<td></td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>28: B</td>
<td>T1 L0 Q1</td>
<td>27</td>
<td>37.8255</td>
<td>43.6971</td>
<td>43.6182</td>
<td>6480</td>
<td></td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>28: B</td>
<td>T1 L1 Q0</td>
<td>37</td>
<td>32.8237</td>
<td>40.9052</td>
<td>41.1593</td>
<td>9488</td>
<td></td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>28: B</td>
<td>T1 L1 Q1</td>
<td>29</td>
<td>36.8012</td>
<td>44.4742</td>
<td>44.7071</td>
<td>34480</td>
<td></td>
<td></td>
</tr>
<tr>
<td>117</td>
<td>28: B</td>
<td>T2 L0 Q0</td>
<td>35</td>
<td>34.0827</td>
<td>40.3105</td>
<td>40.0620</td>
<td>1344</td>
<td></td>
<td></td>
</tr>
<tr>
<td>118</td>
<td>28: B</td>
<td>T2 L0 Q1</td>
<td>29</td>
<td>37.4511</td>
<td>43.1297</td>
<td>42.9431</td>
<td>2368</td>
<td></td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>28: B</td>
<td>T2 L1 Q0</td>
<td>39</td>
<td>32.1168</td>
<td>40.3600</td>
<td>40.6812</td>
<td>7664</td>
<td></td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>28: B</td>
<td>T2 L1 Q1</td>
<td>31</td>
<td>35.6210</td>
<td>43.6518</td>
<td>43.9852</td>
<td>22680</td>
<td></td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>25: B</td>
<td>T3 L0 Q0</td>
<td>36</td>
<td>33.9049</td>
<td>40.0381</td>
<td>39.7831</td>
<td>920</td>
<td></td>
<td></td>
</tr>
<tr>
<td>122</td>
<td>25: B</td>
<td>T3 L0 Q1</td>
<td>30</td>
<td>37.3563</td>
<td>42.7541</td>
<td>42.7274</td>
<td>1536</td>
<td></td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>25: B</td>
<td>T3 L1 Q0</td>
<td>40</td>
<td>31.6061</td>
<td>40.0905</td>
<td>40.3882</td>
<td>2936</td>
<td></td>
<td></td>
</tr>
<tr>
<td>124</td>
<td>25: B</td>
<td>T3 L1 Q1</td>
<td>32</td>
<td>34.7673</td>
<td>43.2725</td>
<td>43.4907</td>
<td>9808</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>27: B</td>
<td>T3 L0 Q0</td>
<td>QP 36</td>
<td>Y 34.0495</td>
<td>U 40.4738</td>
<td>V 40.2209</td>
<td>576 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-------</td>
<td>---------</td>
<td>-------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>--------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27: B</td>
<td>T3 L0 Q1</td>
<td>QP 30</td>
<td>Y 37.3584</td>
<td>U 43.2914</td>
<td>V 43.2941</td>
<td>1240 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27: B</td>
<td>T3 L1 Q0</td>
<td>QP 40</td>
<td>Y 32.2764</td>
<td>U 40.5184</td>
<td>V 40.9641</td>
<td>2296 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27: B</td>
<td>T3 L1 Q1</td>
<td>QP 32</td>
<td>Y 35.7622</td>
<td>U 43.8632</td>
<td>V 44.2345</td>
<td>8576 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27: B</td>
<td>T3 L1 Q0</td>
<td>QP 39</td>
<td>Y 32.7081</td>
<td>U 40.7671</td>
<td>V 41.2074</td>
<td>6304 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30: B</td>
<td>T2 L0 Q0</td>
<td>QP 31</td>
<td>Y 36.7443</td>
<td>U 44.6520</td>
<td>V 44.9325</td>
<td>18576 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30: B</td>
<td>T2 L0 Q1</td>
<td>QP 30</td>
<td>Y 37.7385</td>
<td>U 43.7807</td>
<td>V 43.7885</td>
<td>1880 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30: B</td>
<td>T2 L1 Q0</td>
<td>QP 32</td>
<td>Y 36.4248</td>
<td>U 45.5250</td>
<td>V 44.8306</td>
<td>7704 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30: B</td>
<td>T2 L1 Q1</td>
<td>QP 31</td>
<td>Y 36.8367</td>
<td>U 44.6138</td>
<td>V 44.7575</td>
<td>9008 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31: B</td>
<td>T3 L0 Q0</td>
<td>QP 36</td>
<td>Y 34.5116</td>
<td>U 40.3420</td>
<td>V 40.2206</td>
<td>616 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31: B</td>
<td>T3 L0 Q1</td>
<td>QP 30</td>
<td>Y 38.2511</td>
<td>U 43.6565</td>
<td>V 43.5836</td>
<td>928 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31: B</td>
<td>T3 L1 Q0</td>
<td>QP 40</td>
<td>Y 32.6986</td>
<td>U 40.5476</td>
<td>V 40.8665</td>
<td>2360 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31: B</td>
<td>T3 L1 Q1</td>
<td>QP 32</td>
<td>Y 36.8367</td>
<td>U 44.6138</td>
<td>V 44.7575</td>
<td>9008 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31: B</td>
<td>T3 L1 Q0</td>
<td>QP 34</td>
<td>Y 32.7558</td>
<td>U 40.9385</td>
<td>V 41.2486</td>
<td>48856 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31: B</td>
<td>T3 L1 Q1</td>
<td>QP 26</td>
<td>Y 39.1012</td>
<td>U 44.8131</td>
<td>V 45.2179</td>
<td>152584 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32: B</td>
<td>T1 L0 Q0</td>
<td>QP 33</td>
<td>Y 33.8515</td>
<td>U 40.4682</td>
<td>V 40.5629</td>
<td>3624 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32: B</td>
<td>T1 L0 Q1</td>
<td>QP 27</td>
<td>Y 37.6857</td>
<td>U 43.9013</td>
<td>V 43.9532</td>
<td>8008 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32: B</td>
<td>T1 L1 Q0</td>
<td>QP 37</td>
<td>Y 32.4765</td>
<td>U 41.0338</td>
<td>V 41.1003</td>
<td>11720 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32: B</td>
<td>T1 L1 Q1</td>
<td>QP 29</td>
<td>Y 37.3978</td>
<td>U 45.1793</td>
<td>V 45.4185</td>
<td>13808 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32: B</td>
<td>T2 L0 Q0</td>
<td>QP 35</td>
<td>Y 33.9335</td>
<td>U 40.3399</td>
<td>V 40.1707</td>
<td>1440 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32: B</td>
<td>T2 L0 Q1</td>
<td>QP 24</td>
<td>Y 39.6080</td>
<td>U 44.4984</td>
<td>V 44.6464</td>
<td>48920 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32: B</td>
<td>T2 L1 Q0</td>
<td>QP 34</td>
<td>Y 32.3701</td>
<td>U 40.7241</td>
<td>V 40.7461</td>
<td>7024 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32: B</td>
<td>T2 L1 Q1</td>
<td>QP 26</td>
<td>Y 37.0006</td>
<td>U 44.7413</td>
<td>V 44.8892</td>
<td>20920 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33: B</td>
<td>T3 L0 Q0</td>
<td>QP 36</td>
<td>Y 34.2500</td>
<td>U 40.2149</td>
<td>V 40.0964</td>
<td>800 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33: B</td>
<td>T3 L0 Q1</td>
<td>QP 30</td>
<td>Y 38.1061</td>
<td>U 43.6198</td>
<td>V 43.4415</td>
<td>1328 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33: B</td>
<td>T3 L1 Q0</td>
<td>QP 40</td>
<td>Y 32.4743</td>
<td>U 40.4607</td>
<td>V 40.4815</td>
<td>2480 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33: B</td>
<td>T3 L1 Q1</td>
<td>QP 32</td>
<td>Y 37.1864</td>
<td>U 44.5862</td>
<td>V 44.6915</td>
<td>10248 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33: B</td>
<td>T3 L0 Q0</td>
<td>QP 36</td>
<td>Y 33.6989</td>
<td>U 40.3807</td>
<td>V 40.1704</td>
<td>712 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33: B</td>
<td>T3 L0 Q1</td>
<td>QP 30</td>
<td>Y 37.2536</td>
<td>U 43.6923</td>
<td>V 43.6427</td>
<td>1344 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33: B</td>
<td>T3 L1 Q0</td>
<td>QP 40</td>
<td>Y 32.0848</td>
<td>U 40.9059</td>
<td>V 40.9942</td>
<td>3352 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33: B</td>
<td>T3 L1 Q1</td>
<td>QP 32</td>
<td>Y 36.4835</td>
<td>U 44.9305</td>
<td>V 45.1360</td>
<td>11440 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35: B</td>
<td>T1 L0 Q0</td>
<td>QP 33</td>
<td>Y 33.9335</td>
<td>U 40.3399</td>
<td>V 40.1707</td>
<td>1440 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35: B</td>
<td>T1 L0 Q1</td>
<td>QP 29</td>
<td>Y 37.5981</td>
<td>U 43.6343</td>
<td>V 43.5845</td>
<td>2584 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35: B</td>
<td>T1 L1 Q0</td>
<td>QP 37</td>
<td>Y 32.4765</td>
<td>U 41.0338</td>
<td>V 41.1003</td>
<td>11720 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35: B</td>
<td>T1 L1 Q1</td>
<td>QP 29</td>
<td>Y 37.3978</td>
<td>U 45.1793</td>
<td>V 45.4185</td>
<td>13808 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35: B</td>
<td>T2 L0 Q0</td>
<td>QP 35</td>
<td>Y 33.9335</td>
<td>U 40.3399</td>
<td>V 40.1707</td>
<td>1440 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35: B</td>
<td>T2 L0 Q1</td>
<td>QP 24</td>
<td>Y 39.6080</td>
<td>U 44.4984</td>
<td>V 44.6464</td>
<td>48920 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35: B</td>
<td>T2 L1 Q0</td>
<td>QP 34</td>
<td>Y 32.3701</td>
<td>U 40.7241</td>
<td>V 40.7461</td>
<td>7024 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35: B</td>
<td>T2 L1 Q1</td>
<td>QP 26</td>
<td>Y 37.0006</td>
<td>U 44.7413</td>
<td>V 44.8892</td>
<td>20920 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37: B</td>
<td>T3 L0 Q0</td>
<td>QP 36</td>
<td>Y 33.5418</td>
<td>U 40.4927</td>
<td>V 40.3468</td>
<td>1064 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37: B</td>
<td>T3 L0 Q1</td>
<td>QP 30</td>
<td>Y 37.0767</td>
<td>U 43.8273</td>
<td>V 43.7184</td>
<td>1728 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37: B</td>
<td>T3 L1 Q0</td>
<td>QP 40</td>
<td>Y 32.1111</td>
<td>U 41.0836</td>
<td>V 41.2127</td>
<td>3792 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37: B</td>
<td>T3 L1 Q1</td>
<td>QP 32</td>
<td>Y 36.7249</td>
<td>U 45.0972</td>
<td>V 45.3836</td>
<td>11672 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39: B</td>
<td>T3 L0 Q0</td>
<td>QP 36</td>
<td>Y 33.8157</td>
<td>U 40.7219</td>
<td>V 40.6821</td>
<td>1152 bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>39: B</td>
<td>T3 L0 Q1</td>
<td>QP 30</td>
<td>Y 37.5768</td>
<td>U 44.2080</td>
<td>V 44.1112</td>
<td>1384 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>48: P</td>
<td>T0 L0 Q0</td>
<td>QP 30</td>
<td>Y 34.5693</td>
<td>U 40.9504</td>
<td>V 41.0352</td>
<td>23936 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>48: P</td>
<td>T0 L0 Q1</td>
<td>QP 34</td>
<td>Y 32.7469</td>
<td>U 41.5307</td>
<td>V 41.6355</td>
<td>54480 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>44: B</td>
<td>T1 L0 Q0</td>
<td>QP 33</td>
<td>Y 33.5995</td>
<td>U 40.5921</td>
<td>V 40.3955</td>
<td>4056 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>44: B</td>
<td>T1 L1 Q0</td>
<td>QP 37</td>
<td>Y 32.1077</td>
<td>U 41.5222</td>
<td>V 41.5245</td>
<td>7936 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>44: B</td>
<td>T1 L1 Q1</td>
<td>QP 39</td>
<td>Y 36.8427</td>
<td>U 45.3702</td>
<td>V 45.4231</td>
<td>20800 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>41: B</td>
<td>T3 L0 Q0</td>
<td>QP 36</td>
<td>Y 33.7354</td>
<td>U 40.6866</td>
<td>V 40.5863</td>
<td>1184 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>41: B</td>
<td>T3 L0 Q1</td>
<td>QP 30</td>
<td>Y 37.5486</td>
<td>U 43.9359</td>
<td>V 43.8479</td>
<td>8552 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>41: B</td>
<td>T3 L1 Q0</td>
<td>QP 34</td>
<td>Y 32.1592</td>
<td>U 41.5528</td>
<td>V 41.5245</td>
<td>5008 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>41: B</td>
<td>T3 L1 Q1</td>
<td>QP 37</td>
<td>Y 36.7913</td>
<td>U 45.2265</td>
<td>V 45.4740</td>
<td>12432 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>43: B</td>
<td>T3 L0 Q0</td>
<td>QP 36</td>
<td>Y 33.2394</td>
<td>U 40.5657</td>
<td>V 40.4300</td>
<td>1064 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>43: B</td>
<td>T3 L0 Q1</td>
<td>QP 30</td>
<td>Y 36.9679</td>
<td>U 43.8401</td>
<td>V 43.6359</td>
<td>2112 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>43: B</td>
<td>T3 L1 Q0</td>
<td>QP 40</td>
<td>Y 31.8572</td>
<td>U 41.5316</td>
<td>V 41.5297</td>
<td>4976 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>43: B</td>
<td>T3 L1 Q1</td>
<td>QP 32</td>
<td>Y 36.5143</td>
<td>U 45.2375</td>
<td>V 45.3301</td>
<td>13616 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>46: B</td>
<td>T2 L0 Q0</td>
<td>QP 35</td>
<td>Y 33.3776</td>
<td>U 40.6596</td>
<td>V 40.5648</td>
<td>1912 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>46: B</td>
<td>T2 L0 Q1</td>
<td>QP 29</td>
<td>Y 37.1716</td>
<td>U 43.9246</td>
<td>V 43.9264</td>
<td>3784 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>46: B</td>
<td>T2 L1 Q0</td>
<td>QP 39</td>
<td>Y 31.9763</td>
<td>U 41.5005</td>
<td>V 41.4654</td>
<td>7752 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>46: B</td>
<td>T2 L1 Q1</td>
<td>QP 31</td>
<td>Y 36.7957</td>
<td>U 45.1383</td>
<td>V 45.3378</td>
<td>21744 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>45: B</td>
<td>T3 L0 Q0</td>
<td>QP 36</td>
<td>Y 33.1837</td>
<td>U 40.7249</td>
<td>V 40.4931</td>
<td>984 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>45: B</td>
<td>T3 L0 Q1</td>
<td>QP 30</td>
<td>Y 36.8423</td>
<td>U 44.0711</td>
<td>V 43.8510</td>
<td>2088 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>45: B</td>
<td>T3 L1 Q0</td>
<td>QP 40</td>
<td>Y 31.8234</td>
<td>U 41.6313</td>
<td>V 41.7030</td>
<td>5192 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>45: B</td>
<td>T3 L1 Q1</td>
<td>QP 32</td>
<td>Y 36.5459</td>
<td>U 45.3171</td>
<td>V 45.3316</td>
<td>13424 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>47: B</td>
<td>T3 L0 Q0</td>
<td>QP 36</td>
<td>Y 33.7041</td>
<td>U 40.9139</td>
<td>V 40.7513</td>
<td>1104 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>47: B</td>
<td>T3 L0 Q1</td>
<td>QP 30</td>
<td>Y 37.6097</td>
<td>U 44.2811</td>
<td>V 44.1125</td>
<td>1520 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>47: B</td>
<td>T3 L1 Q0</td>
<td>QP 40</td>
<td>Y 31.8799</td>
<td>U 41.6084</td>
<td>V 41.7179</td>
<td>4240 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>47: B</td>
<td>T3 L1 Q1</td>
<td>QP 32</td>
<td>Y 36.7392</td>
<td>U 45.1752</td>
<td>V 45.3503</td>
<td>11776 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>49: P</td>
<td>T3 L0 Q0</td>
<td>QP 36</td>
<td>Y 33.0763</td>
<td>U 40.6965</td>
<td>V 40.8034</td>
<td>1960 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>49: P</td>
<td>T3 L0 Q1</td>
<td>QP 30</td>
<td>Y 36.3460</td>
<td>U 43.9396</td>
<td>V 44.1564</td>
<td>3600 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>49: P</td>
<td>T3 L1 Q0</td>
<td>QP 40</td>
<td>Y 31.6573</td>
<td>U 41.3783</td>
<td>V 41.5749</td>
<td>6512 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>49: P</td>
<td>T3 L1 Q1</td>
<td>QP 32</td>
<td>Y 36.4858</td>
<td>U 44.8198</td>
<td>V 45.2475</td>
<td>18432 bit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SUMMARY:**

<table>
<thead>
<tr>
<th>bitrate</th>
<th>Min-bitr</th>
<th>Y-PSNR</th>
<th>U-PSNR</th>
<th>V-PSNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>77</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>219</td>
<td>432x240 Ø 3.1250</td>
<td>346.3964</td>
<td>152.5643</td>
<td>40.0417</td>
</tr>
<tr>
<td>220</td>
<td>432x240 Ø 6.2500</td>
<td>400.0885</td>
<td>171.9154</td>
<td>39.0361</td>
</tr>
<tr>
<td>221</td>
<td>432x240 Ø 12.5000</td>
<td>436.8680</td>
<td>185.6960</td>
<td>38.3951</td>
</tr>
<tr>
<td>222</td>
<td>432x240 Ø 25.0000</td>
<td>460.4120</td>
<td>193.9840</td>
<td>38.0408</td>
</tr>
<tr>
<td>223</td>
<td>848x480 Ø 3.1250</td>
<td>1252.6393</td>
<td>481.3821</td>
<td>39.7563</td>
</tr>
<tr>
<td>224</td>
<td>848x480 Ø 6.2500</td>
<td>1507.2077</td>
<td>555.5923</td>
<td>38.5741</td>
</tr>
<tr>
<td>225</td>
<td>848x480 Ø 12.5000</td>
<td>1725.9240</td>
<td>618.3800</td>
<td>37.7320</td>
</tr>
<tr>
<td>226</td>
<td>848x480 Ø 25.0000</td>
<td>1895.0800</td>
<td>660.9280</td>
<td>37.2146</td>
</tr>
</tbody>
</table>