FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Sistema de Video-On-Demand para IPTV

Nuno Mota

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Orientador: Maria Teresa de Andrade (Prof. Dr.)

Co-Orientador: André Mendes (M.Sc.)

Julho de 2011

U. PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MIEEC - MESTRADO INTEGRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES **2010/2011**

A Dissertação intitulada

"Sistema de Vídeo-On-Demand para IPTV"

foi aprovada em provas realizadas em 20-07-2011

o júri

[illegible]

Presidente Professor Doutor José António Ruela Simões Fernandes
Professor Associado do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

[illegible]

Professor Doutor José Manuel de Castro Torres
Professor Associado do Faculdade de Ciências e Tecnologia da Universidade Fernando Pessoa

[illegible]

Professora Doutora Maria Teresa Magalhães da Silva Pinto de Andrade
Professora Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

[illegible]

Mestre André Mendes
Entone

O autor declara que a presente dissertação (ou relatório de projeto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extractos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são corretamente citados.

[illegible]

Autor - Nuno Ricardo Mesquita Pereira da Mota

Faculdade de Engenharia da Universidade do Porto

Resumo

O Internet Protocol Television (IPTV) baseia-se na distribuição de serviços de televisão através de uma arquitectura de redes IP. O IPTV está a inovar o conceito de transmissão de conteúdo multimédia, disponibilizando um conjunto de serviços integrados num ambiente único. A revolução digital ajudou a criar uma solução multimédia de alta qualidade utilizando baixas taxas de transmissão. Um dos serviços de maior sobrecarga é o de Vídeo a Pedido. Este implica uma ligação dedicada a um só cliente, de modo a possibilitar comandos de utilização normais. Tome-se como exemplo as funções *parar*, *avançar rápido*.

As empresas comerciais oferecem produtos personalizados que por vezes não viabilizam todas as capacidades aos clientes. E, estas ofertas estão longe de serem soluções acessíveis a nível monetário. Apenas um sistema personalizado e construído somente com ferramentas gratuitas pode fornecer todos os requisitos de que necessitamos. O nosso objectivo baseia-se em criar uma plataforma de Vídeo a Pedido confiável e escalável, capaz de lidar com as tecnologias mais usadas actualmente.

Visto que este projecto era destinado somente a um ambiente web, uma exposição de Serviços Web mostrou-se fundamental para maximizar o seu potencial, fornecendo a outros programadores as ferramentas necessárias à construção e integração dos nossos serviços noutras aplicações. O Simple Object Access Protocol e o REpresentational State Transfer eram até ao momento os protocolos mais usados neste contexto, estando disponíveis em serviços como o Facebook e o Google. Mostrou-se conveniente a criação de uma interface de administração e de usuário para controlar todos os conteúdos e informação de vídeo, bem como, visualizá-los num ambiente apelativo, de modo a provar o conceito.

A conclusão da presente tese permitiu aferir que o sistema desenvolvido, longe de ser um produto profissional, apresentou-se capaz de competir facilmente com soluções existentes e tornar-se ainda melhor com algum trabalho futuro.

Abstract

Internet Protocol Television (IPTV) is based on the distribution of television services over IP networks. IPTV is redefining the concept of media broadcast providing a vast amount of services in one single environment. The digital revolution also helped to create and deliver a high quality media solutions, using lower bit rates. One of the most demanding services is the Video-On-Demand (VOD). This implicates a dedicated streaming channel for each user in order to provide normal media player commands (i.e. pause, fast forward).

Most multimedia companies offer personalized products that sometimes do not fulfil all the users needs, and are far from being a cheap solution. Only a personalized system built solely with open-source tools, may provide all the requirements we want. Our goal was to create a reliable and scalable VOD service, fully capable of dealing with the present state-of-the-art multimedia technologies.

As this project was mainly for an Internet environment a Web Services (WS) exposure was needed to maximize the potential of our service, providing other developers the right tools to build and integrate our services in different applications. The Simple Object Access Protocol (SOAP) and the REpresentational State Transfer (REST) were currently the most used Web Services protocols and each one was widely spread among services like Facebook and Google. Both a Administration and User's interface was needed to fully manage all the video metadata and contents, and properly view it in a rich and appealing application providing the ultimate proof of concept.

In the end of this thesis we were able to acknowledge that the developed system, far from being an enterprise solution, was in fact capable of levelling with existing solutions, and could become even better with further development.

Acknowledgements

This thesis will only be complete after I acknowledge all the support and contributions that everyone shared with me along this project's path.

First I would like to thank both of my supervisors, Prof. Dr. Maria Teresa Andrade and M.Sc. André Mendes for their insightful guidance and support when it was most needed.

I thank my friends Marcelo, for his help with the experiments and Marco, for his suggestions and feedback.

Finally, my family for their never-ending support and patience, and my girlfriend for her companionship and support during the most difficult times throughout the entire project.

Nuno Mota

*“We never fail when we try to do our duty,
we always fail when we neglect to do it.”*

Robert Baden-Powell

Contents

List of Figures

List of Tables

Abbreviations and Symbols

ANACOM	Autoridade Nacional de Comunicações
API	Application Programming Interface
AVC	Advanced Video Coding
BSP	Basic Security Profile
CCIR	Consultative Committee on International Radio
CDN	Content Delivery Network
CPU	Central Processing Unit
DCT	Discrete Cosine Transform
DTT	Digital Terrestrial Television
DVB	Digital Video
DVD	Digital Versatile Disc
GOP	Group of Pictures
GUI	Graphical User Interface
HDTV	High Definition Television
HTML	Hyper-Text Markup Language
HTTP	Hyper-Text Transfer Protocol
HTTPS	Hyper-Text Transfer Protocol Secure
HVS	Human Visual System
IETF	Internet Engineer Task Force
IP	Internet Protocol
IPTV	Internet Protocol Television
ISP	Internet Service Provider
Mb	Megabits
Mbps	Megabits per second
MB	MegaBytes
MPEG	Moving Pictures Experts Group
MV	Motion Vector
NAL	Network Abstraction Layer
OSS	Open Source Software
OTT	Over-The-Top
PHP	Hypertext Preprocessor
REST	Representational State Transfer
RFC	Request For Comments
RGB	Red, Green and Blue
RIA	Rich Internet Application
RM	ReliableMessaging
RR	Receiver Reports
RTCP	Real-time Transport Control Protocol

RTP	Real-time Transport Protocol
RTSP	Real Time Streaming Protocol
SAML	Security Assertion Markup Language
SHA-1	Secure Hash Algorithm 1
SD	Standard Definition
SDK	Software Development Kit
SDP	Session Description Protocol
SFTP	Secure File Transfer Protocol
SNMP	Simple Network Management Protocol
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SR	Sender Reports
SSL	Secure Sockets Layer
STB	Set-Top Box
TCP	Transmission Control Protocol
TDT	Terrestrial Digital Television
TLS	Transport Layer Security
TS	Transport Stream
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locater
VOD	Video-On-Demand
VBR	Variable Bit Rate
VLC	Variable-Length Code
VCL	Video Coding Layer
W3C	World Wide Web Consortium
WS	Web Services
WSDL	Web Services Description Language
WSS	Web Services Security
WWW	World Wide Web
XML	eXtensible Markup Language

Chapter 1

Introduction

Nowadays, millions of users have access to digital contents through many applications that did not exist a couple of years ago. Both on the internet as well as in digital television, the viewer transported himself from a passive usage to a place where he can access any type of contents and information. The revolution of IPTV brought thousands of new possibilities, from choosing the camera view in a football match, to selecting the favourite shows, series, or a selection of movies without the need to wait for the scheduled time.

Video-on-demand (VOD) has become very popular over the years, for example, Youtube has more than 3 billion views per day. In one year Youtube had a growth of 50% in its traffic [5]. One important aspect is that broadband access has seen a tremendous growth in speed and infrastructures. Coaxial cables are being switched to fiber optic cables, delivering over 1Gbps of speed to any domestic environment. This exponential growth shows us that delivering high quality streaming over IP networks is possible, as we see already in broadcast media. Nowadays, Content Delivery Networks (CDN) provide scalability, with many servers distributed across the Internet "closer" to costumers (see fig. 1.1). This broadband requirements, widespread deployment and popularity proves to be a costly service to provide.

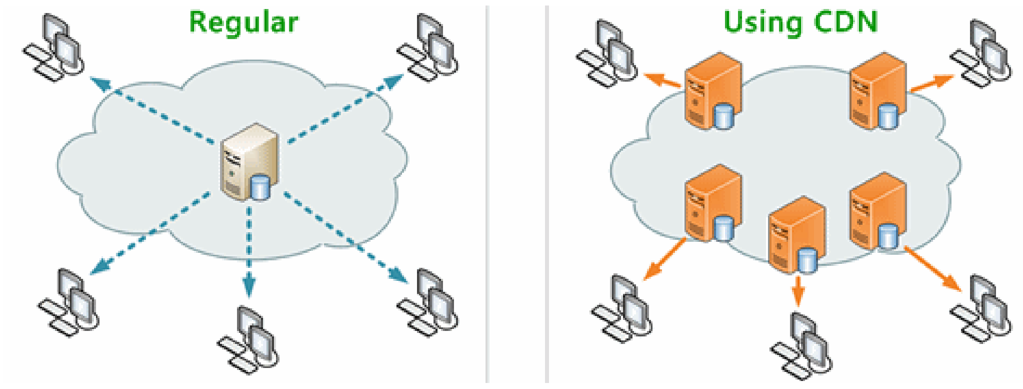


Figure 1.1: Content Delivery Network [1]

Within the television market, where Video-on-Demand has faced a stronger development, the choices are dictated by the Service Providers and each service has an associated cost. Normally the user needs to rent a Set-top Box (STB) to be able to get the most out of these services, which a basic set-up does not provide. Thus, UK TV networks came up with an idea of a free-to-air collection of services for the Digital Terrestrial Television (DTT). There is no need for a subscription or contract, and all that the user needs is a TV or apparatus capable of receiving DTT services. Their motivation is "Buy today, watch today, free forever". They called it the Freeview and already spread to other countries like New Zealand and Australia.

On the Internet these type of services are also starting to grow together with the expansion of broadband access. Still, most of these applications are paid, because digital contents also means copyright contents. In most of the cases the specific software for this type of applications is proprietary, which means you either buy a license to use an available software or pay to create your own software.

The most famous examples on the Internet today are Netflix [6] and Amazon Prime [7]. In both these services a user can have an account for a minimum of $7,99 a month or $79 the annual fee. TV episodes and movies are available, and some even deliver unlimited DVDs for just $2 a month. The movies and series availability is the only thing that can distinguish any service. People usually arrive home after a day's work, missing their favourite shows. The only thing they would want from such a service is for that specific show to be available after it is broadcast. The same thing happens with movies. After a few months in the theatres they are sold as DVDs, and should be available online as well. Netflix counts over 23 millions members today, being probably the best online solution available.

1.1 Motivation

The internet world is always evolving and Video-on-Demand is gaining more and more users everyday. However technology has not evolved as expected. It is difficult to find an application where the user can watch a movie or a series, like he was playing from its own DVD. Trick mode operations, such as seeking to the middle of the movie to start from the last viewed point, fast forward to skip undesired parts, pause for a quick break are functionalities that are not easily found. Video-on-Demand requires even more than what simple television broadcast requires, because each streaming session is a private session that the server needs to handle. At present, there is no software complete enough and available as open source to provide us with a sufficiently good solution.

By taking advantage of the Internet's potential it is possible to make things even more interesting. Companies like Facebook, Ebay, Google, Amazon, Flickr, Twitter and many others provide special interfaces (i.e. Web Services) for developers, other than their own, to make use of their services in other applications. This is a way to give the power to other people to create new applications using their own services, taking a tremendous load out of their hands. Probably every mobile application that accesses any of the mentioned services, use some of these tools.

1.2 Objectives

The goal of this dissertation is to develop a tool that can offer the above referred desired functionality, allowing users to access and consume videos and other contents in an IPTV environment. In this case this system is supposed to work both on Over-The-Top (OTT) layers and in private networks, however, tests will only be conducted in a private network. Users should be able to visualize the information of all available contents and access them inside the same application. Content should be retrieved from a server, which also exposes metadata associated to the audiovisual content, collected from the appropriate websites. Users should be able to browse metadata, obtaining descriptive information about the available A/V contents and only then, select and start receiving a specific movie. The application should provide the user major trick mode functionalities, offering the same experience a user would have with a normal household media player.

The server should have the capacity to serve several users and at the same time be able to be configured through a web-based administration interface. The supported video formats and network protocols should be the ones used in today's industry, to provide full support and interoperability. This application should also provide the specific tools for others to create applications using our services.

There are several open source software (OSS) tools capable of streaming and receiving contents on the internet but they are just a mere startup point for this project, because they don't fulfil the basic requisites of the application we are looking for. The solution is to use this existing tools and start building up our application.

1.3 Methodology

The main question that is posed before start developing the proposed system is the following:

Is it possible to develop an open source Video-on-Demand Application fully capable, reliable, scalable and ready for IPTV?

The main objective of this thesis is to build an application, meaning that development will be the first priority. However, a big research should be involved when looking for the right tools to help achieve every defined goal. There are a lot of options that can help leverage some of the functionalities we want, leaving us with the task of bringing them all together into one application. These tools must be stable and fully tested. The main concern must be to create a reliable application, providing enough evidence and references that everything will work as expected.

We will start by researching the protocols, specifications and applications used in the current multimedia environment. A Software Requirements Specification will also be produced with some of the basic guidelines, which aims at creating a system that works the way that was originally designed. We will analyse the users perspective to acquire all their basic needs, and identify the administration's necessities in order to create a tool able to provide total access and control.

The major foreseen problem concerns the time needed for each task. A complete Video-on-Demand system requests background knowledge in multiple domains, and sufficient time to develop each part of the system. During each task, the right tool needs to be chosen, otherwise precious time will be spent trying to figure out where things went wrong.

1.4 Thesis Structure

This thesis is divided into 6 chapters. Chapter 2 describes the state of the art in multimedia technology is described as well as the existing web services protocols and architectures.

In chapter 3 a requirements specification is presented with a complete description of the system's behaviour and requirements. This chapter provides the guidelines for the software development.

The development work will be described in chapter 4, which contains a description of every needed step to create our application, and some explanations why each implemented framework was chosen for this project.

Chapter 5 will provide some discussions concerning a few developed modules and the results of the work developed and the tests performed. Finally chapter 6 concludes this thesis.

Chapter 2

State of the Art

This chapter presents the state of the art on multimedia standards, streaming solutions and web services technologies. It first describes some of the most important video coding techniques available and the most relevant file formats for multimedia data.

Afterwords an overview of the main network streaming protocols is given and some necessary tools to implement a web services communication. Some software solutions, regarding the streaming service are also given, in particular solutions where the General Public Licenses [8] applies or other free/open-source software that can provide the best results for our application.

2.1 Video Codecs

Video is a sequence of pictures, in which each picture is described by an array of pixels. The red, green and blue signals (RGB) can be combined and expressed as luminance (Y) and chrominance (UV) components. The chrominance signals may be compressed in relation to the luminance without affecting the picture quality due to the characteristics of the Human Visual System (HVS). The CCIR recommendation 601 [9] defines how the YUV video signals can be described as pixels. Each of these pixels can have millions of colors associated to a number of bits. In each second, several of this pictures, called frames, can be reproduced and a minimum of 15 frames is necessary to obtain a "moving image" with an acceptable quality.

In the CCIR recommendation 601 a normal video would be 720pixels x 480 pixels x 30 frames and if each pixel had 16 bits of resolution, the video data rate would be around 165Mbit/s. This transmission rate is too high for the current broadband access and it would also impose a strong load on the user's CPU, which would take a considerable time to process the information, possibly longer than real-time. According to ANACOM [10], in Portugal during the last quarter of 2010, almost 2 million users had cable internet, where speeds could go up to 24Mbit/s. As each day goes by, fiber optics becomes more real, but the ISPs still have prohibitive prices, having a 5% share of the market.

To solve this problem several ways were created to compress video and audio, to reduce them to a size possible of being transmitted at lower speeds, without loosing too much quality. The Moving Pictures Experts Group [11], a committee created by the International Organization for Standardization [12], was established to create the standard of codification of digital content. MPEG has developed so far 3 standards for the compression of audiovisual information: MPEG-1, 2 and 4. Each standard is divided in several parts including systems components, which specifies container formats, video coding specifications, etc. The most relevant to multimedia streaming are: MPEG-1 Audio, MPEG-2 Systems and Video parts, MPEG-4 part 10 and 14. MPEG-4 part 10 is most widely known as H.264 or Advanced Video Coding (AVC) and resulted of a combination effort of MPEG and the Video Coding Experts Group (VCEG) of International Telecommunication Union (ITU [13]). MPEG standards require a license granting rights to manufacture and sell products under this standards or use such products to provide video content for profit. However , H.264 is royalty free for non-profit applications.

Besides MPEG there are also other types of video codecs and systems and open-source formats, that can be applied in the genre of applications this thesis aims at.

2.1.1 MPEG-2/H.262

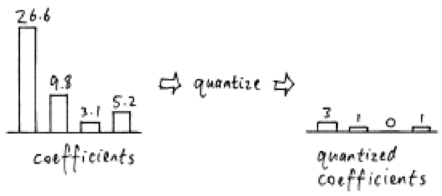
The video part of MPEG-2 is also known as H.262 given that its final specification was achieved as a joint work between MPEG and ITU. This standard is used nowadays in all kinds of digital applications, for example Digital Video Broadcast(DVB). It involves four parts and its primarily goal is coding of CCIR 601 or higher resolution videos to achieve lower data rates, without compromising the quality of these videos. The result is a video with a Variable Bit Rate (VBR). For now we will only discuss the part 2 of this standard that specifies the video coding.

It can achieve lower data rates from less than 2 Mbit/s up to 16Mbit/s, but for HDTV content and movie productions it goes up to 80Mbit/s. The principle is to remove redundant information prior to transmission. Two major techniques are employed: Discrete Cosine Transform (DCT) coding and motion-compensated inter-frame prediction. A main feature of this standard is the three types of compression it uses: an intra-frame (I-frame) encoding, a predictive frame (P-frames) and a bidirectional predicted technique (B-frames).

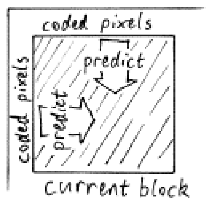
I-frames are the biggest frames in size and encoded using the DCT and quantization to reduce the required number of bits to be transmitted in an image block. The quantized DCT block is then scanned for low-frequency coefficients and occurrences of zero-value coefficients. The list of values produced are entropy coded using a variable-lenght code (VLC).

In other words, I-frame (see fig. 2.1) uses spatial reduction and takes advantage of the incapacity of the human eye, called the phsycovisual redundancy, to notice certain changes in a picture.

P-frames and B-frames are obtained by applying motion prediction prior to the DCT and quantization processes. P-frames can have a higher compression in relation to I-frames because they use motion-compensated inter-frame prediction, which means they are based on precedent frames, I-frames or P-frames. B-frames have the highest compression of the three frames because it uses the past and future frames (see fig. 2.2) as reference, but B-frames cannot be used as a reference.



(a) Quantization



(b) Intra prediction

Figure 2.1: Intra-Frame [2]

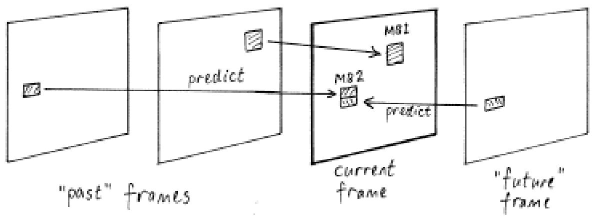


Figure 2.2: B-Frame prediction [2]

For each predicted image, Motion Vectors (MV) are calculated on a 16x16 pixels block basis (Macro Blocks, MBs), indicating the displacement in x,y coordinates that each MB has suffered in relation to a MB in the reference(s) image(s). Normally the MB being predicted is not exactly equal to the reference MB with the indicated displacement applied. There is normally a prediction error associated to each predicted MB in addition to the MV. To this information (MVs and prediction error) is then applied the DCT, followed by quantization and entropy encoding just like it happens in intra-frame coding.

These different types of frames (I, P and B) are arranged in Groups Of Pictures (GOP). Each GOP contains only one I frame and distinct types of GOPs can be obtained by combining differently I, P and B frames. Figure 2.3 presents some possible GOP structures. Due to the fact that B pictures can be predicted based on future references, frames need to be re-ordered prior to transmission so that the decoder receives the reference before the image that has been predicted based on that reference.

Given the complexity of the standard and that most applications do not need to support its full implementation, levels and profiles were created to satisfy distinct requirements of different applications. This way it is much easier to develop applications that are compliant to the standard without having to implement it at full extent. Profiles are related with the type of coding tools that are used and thus are normally associated with the complexity. Levels are related with the range of values of encoding parameters and are normally associated with the quality. For example, some profiles use only I and P frames, whilst others use also B frames, which is more complex to implement; some levels constrains the spatial and temporal resolution of the video or the maximum

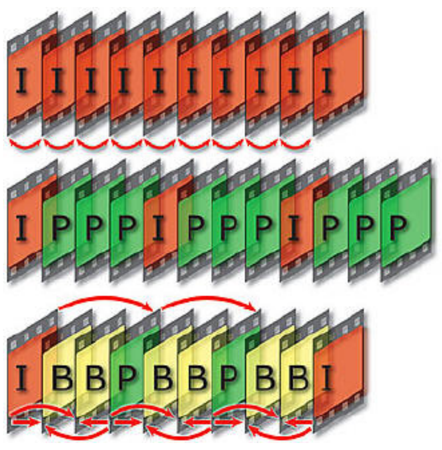


Figure 2.3: Group Of Pictures [3]

allowed bit rate.

Table 2.1: Levels

	Max. width (pixels)	Max. height (pixels)	Max. Frame (Rates)	Max bit rate (Mbit/s)	Application
Low	352	288	30	4	Set-top boxes
Main	720	576	30	15	DVD, SD-DVB
High-1440	1440	1152	60	60	HDTV
High	1920	1152	60	80	Movie productions

Table 2.2: Profiles

	Picture Coding	Chroma Format	Scalable Modes	Application
Simple Profile	I,P	4:2:0	none	Video-conference
Main Profile	I,P,B	4:2:0	none	STB, DVD, HDTV
SNR Profile	I,P,B	4:2:0	SNR scalable	TDT
Spatially Scalable Profile	I,P,B	4:2:0	spatial-scalable	HDTV
High Profile	I,P,B	4:2:0 and 4:2:2	SNR and Spatial	-

2.1.2 H.264/MPEG-4 AVC

This standard was created with the goal of substantially reducing the data rate transmissions of other standards, such as MPEG-2, without increasing too much its complexity and implementation costs. Studies show that, if well implemented, it can reduce up to 50% the data rate [14] when compared to MPEG-2 video. Another goal was to achieve usability within several types of applications, e.g. different kinds of networks, because an increasing number of services, for example HDTV, needed higher coding efficiency.

The standard specification is divided into two parts: the video coding layer (VCL) is responsible for coding the video, and the Network Abstraction Layer (NAL), the part that formats the coded video in a way that can be used in several transport layers(e.g. RTP packets 2.3.5) or storage media. This standard has similar specifications as other video codecs, because it uses inter-prediction with motion compensation, transform, quantization and entropy encoding processes to achieve a H264 bitstream.

A macroblock is used to make a prediction of the previous coded data in two ways, from the same frame (intra prediction) or from already coded and transmitted frames (inter-prediction). These methods are more adjustable than other standards. A reconstruction filter is applied to every macroblock in order to reduce blocking distortion. With this technique the quality of the images is improved and the prediction error is reduced.

This standard also includes detailed information on how to represent video data and other information. The raw H264 stream consists of a series of pieces called the NAL units. These can include two things: information to proper decode the stream called parameters, and the video frames itself called slices.

When it comes to profiles and levels, we now have 17 profiles with several improved features and 16 levels with maximum bit rates from 64 Kbit/s in the baseline profile to 960Mbit/s in the highest profile (High 4:4:4 Predictive Profile - Hi444PP). We can also have levels of resolution from 128x96@30.9 in the first level to 4096x2304@26.7 in the top level.

The blu-ray discs use this standard because the video has better quality at the same bit rate as others, providing more viewing hours. Most of the internet content providers also use this standard for video transmission. For example Youtube uses mostly H264, but with all the infrastructures costs, it’s now moving to a new standard called WebM that uses an open source video codec called VP8(see section 2.1.3).

2.1.3 VP8

VP8 was originally created by On2 Technologies, now a subsidiary of Google. This standard, like many video compression codecs, uses MBs further decomposed into 4x4 subblocks. It predicts subblocks using previously constructed blocks, and adjusts such predictions using a DCT transform. However, in one special case it uses a "Walsh-Hadamard" transform instead of a DCT.

Perhaps the most notorious difference between other video codes is the absence of the B-frame. Instead, it introduces a notion of alternate prediction frames, called golden frames and alternative reference frames (altref). Blocks in a P-frame can be predicted using blocks from a previous frame as well as using the most recent golden or altref frame. This P-frames may optionally replace the most recent golden or altref frame. Every I-frame is a golden and altref frame, and may be used to partially overcome the intolerance to dropped frames (e.g. a P-frame is dropped or corrupted and cannot be correctly decoded until a I-frame is correctly received).

Independent testers [15, 16] state that H264 has better quality with the same data rates and that the VP8 standard may have some patent issues because of its similarity to the H264 specifications. There's also an open source video codec called Theora, supported by Firefox and Opera but in compression-wise it's worst than VP8 and H264.

2.2 Containers

Most of the streaming software available supports all the standards described in this paper. Like video codecs, video containers have much importance in a video streaming application. Containers describe how the video file is organized as a file in the computer and later network protocols are responsible for streaming this data over the internet. So the most important thing is to find a container suitable for video files which are going to be streamed from a server.

2.2.1 Ogg

Ogg [17] is a free video container used for streaming or for data storage. It's supposed to be as simple as possible with only three major principles: framing, ordering and interleave. It has a simple seek design which leads to stream capture with only 128kB of data. Choosing any timeline of the Ogg file should then be very quick. Essentially every packet is packed into an unframed logical bitstream. This logical bitstream is grouped and framed into Ogg Pages, each with a unique serial number, to form a physical bitstream. Figure 2.4 shows an example of an elementary stream. Packets may be spread across two pages boundaries or even multiple pages. Several logical bitstreams can be multiplexed into a single stream.

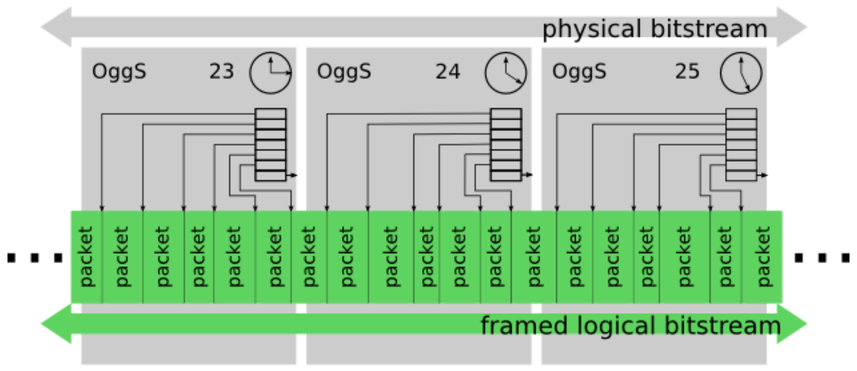


Figure 2.4: Ogg Elementary Stream [4]

2.2.2 Matroska

The Matroska Container is aimed for multiple applications. Some of its main goals are: fast seeking in the file, high error recovery, chapter entries, and streaming capabilities over HyperText Transfer Protocol (HTTP) and Real-time Transport Protocol (see section 2.3.3). The container is divided into 9 different sections. Matroska is not meant to be used in streaming applications where RTP (see section 2.3.3) is already being used, because RTP possesses timing and channel mechanism that would be wasted if repeated with Matroska. However, Matroska is perfectly suitable for streaming in HTTP transport.

2.2.3 WebM

WebM is a media file format, which defines the container structure, video and audio formats. WebM files use the VP8 video codec to compress the video frames and Vorbis audio codec to compress audio. The WebM file structure is based on the Matroska container.

2.2.4 Transport Stream - MPEG-2 part1

The Transport Stream (TS) is a format specified in part 1 of the MPEG-2 standard constituting one of the syntaxes of the MPEG-2 system level. Its main characteristics are the multiplexing of several streams, like Ogg, to have a synchronized output, and also the fact that it is a packet oriented structure. It is mostly used in DVB applications (e.g Set-top boxes) and offers error correction mechanisms, in channels where reliability is not an issue. The packets in the transport layer are 188 bytes in length, constituted by one synchronization byte (0x47), three one-bit flags a 13-bit packet identifier, followed by other options and payload data.

Eventually, the communication medium may add error correction bytes to the packet, depending on the transmitting signal. The packet identifier is responsible for identifying the different programs/channels present in the transport stream. There's also a feature called Program Clock Reference, that enables a decoder to synchronize audio and video. Figure 2.5 shows an example of a TS stream.

Broadcast industry nowadays uses this standard because it has advantages even with the trade-off of increasing the payload. We can have audio, video and other information like subtitles multiplexed into one single file adding the ability to seek the content more easily.

2.3 Network Protocol Stack

Different protocols are used nowadays to exchange information on the Internet and divided into several layers according to their specific function. In this section we mention the transport layer and the application layer protocols involved in the streaming process. In the transport layer we have the Transport Control Protocol (TCP [18]), the User Datagram protocol (UDP [19]) and the Real-time Transport Protocol (RTP [20]). Despite the claim that RTP is a transport layer, from the developer's perspective RTP belongs to the application layer. In the application layer we have

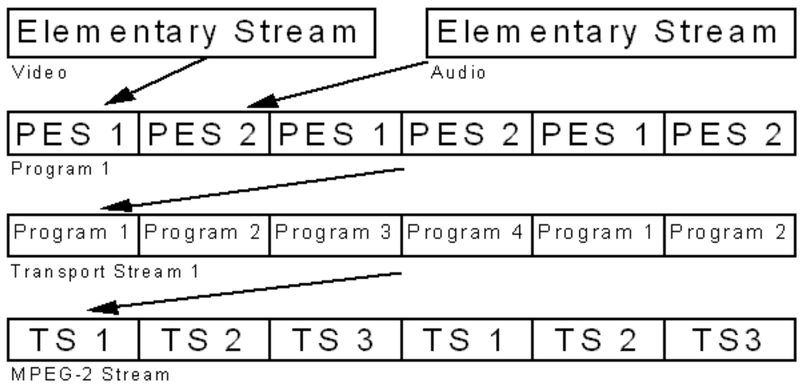


Figure 2.5: Transport Stream Video Stream

three important protocols: (i) a RTP Control Protocol (RTCP) associated with RTP specifications, (ii) a Real-time Streaming control Protocol (RTSP [21]), and (iii) a Session Description Protocol (SDP [22]).

2.3.1 Transport Control Protocol

The TCP is used on top of a basic Internet Protocol (IP [23]). It was created to be a reliable protocol where no exchanged packet should be lost between a host to host communication. It is a connection-oriented protocol and intended to recover from corrupted or lost data by properly signalling the occurrence. The receiver should send back an ACK signal to indicate that the packet was successfully received. By using sequence numbers it guarantees that the receiver can eliminate duplicated packets. If it does not signal back, that means something went wrong during the transmission. To provide multiplexing of different communications from different processes within the same host, the TCP provides a set of ports for each host. Together with the network and host address (e.g. IP Address) it forms a socket. This sockets are responsible to identify the connection between two hosts.

Despite its reliability, in live streaming applications this protocol carries to much overhead and unnecessary information exchange, and that is why most streaming solutions are based on a connection-less protocol also known as UDP.

2.3.2 User Datagram Protocol

The UDP protocol also uses the IP as the underlying protocol, using a socket to perform a connection between two hosts. The idea of connection-less comes from the fact that the information exchange can happen without the other host's prior knowledge. The packets are also transmitted without sequence numbers, and may arrive out of order while no data is retransmitted in case of error or failure. This protocol is used when waiting for dropped packets is undesired. This means that it is perfect in situations like live broadcast, where missing some packets is not an issue.

2.3.3 Real-time Transport Protocol

The most important streaming protocol is the Real-time Transport Protocol. RTP is a standard that delivers real-time data streams, carrying audio and video over unicast or multicast network services. It's typically used on top of UDP, adding a minimum of 12 bytes of overhead, but it runs in other network or transport protocols. RTP does not ensure that data is delivered sequentially, nor does it guarantee the error-free delivery of the packets, nevertheless it includes mechanisms and data that can be used at the application level to provide those guarantees. For example, the sequence numbers included in the RTP will tell the receiver how to properly reconstruct the sequence. Figure 2.6 shows an example of a RTP packet.

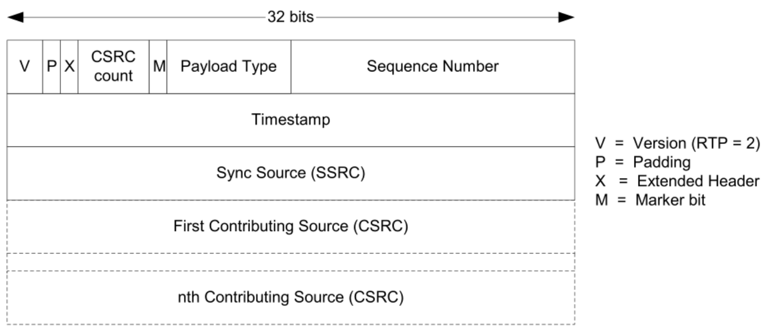


Figure 2.6: RTP Packet

This protocol supports different kinds of media types such as the ones described in this work: H.264 [24] and MPEG-2 Video [25]. For each type of media, RTP has different ways of dealing with the payload. RTP protocol uses two UDP connection ports, one for each video and audio streams and one for the respective RTCP information. The port numbers information is exchange between the client and the server during the Server's Setup. In cases where a TS container is used, RTP only needs two ports because both audio and video are multiplexed into a TS packet. For this reason and because of the unnecessary overhead due to redundant information (i.e timestamps), STBs that specifically use TS video containers, request raw-UDP connections.

2.3.4 RTCP

Real-time Transport Control Protocol is associated with RTP 2.3.5 and its main goal is to provide a control channel for each media session. It provides information of reception statistics and current activities. With this information it is able to properly configure any problem with the connection due to its unreliability. RTCP is carried over the same protocol as RTP. This may be an important protocol if we are interested in an adaptive situation where, for example, different bit-rates may be achieved, or just for monitoring purposes.

It works on a report basis with two functions available: sender reports (SRs) and receiver reports (RRs). The sender reports, detail the number of packets exchanged which provides a way to calculate the proper mean data rate for the all session or for every transmission interval.

The receiver reports includes statistics like: packet loss, highest sequence number received and a moving average of the inter-arrival jitter of the media packets, which gives an indirect view of the playout buffer used in the receiver.

2.3.5 Real Time Streaming Protocol

Real-time Streaming Protocol is designed to control the media streams, sending the directives from the client to the streaming server. This protocol works like an HTTP connection, the only difference is that RTSP is a stateful protocol. In other words, a session identifier is created to keep track of sessions; so there's no need for a permanent TCP connection. As described in the RTSP's RFC, the RTSP behaviour is detailed in figure 2.7.

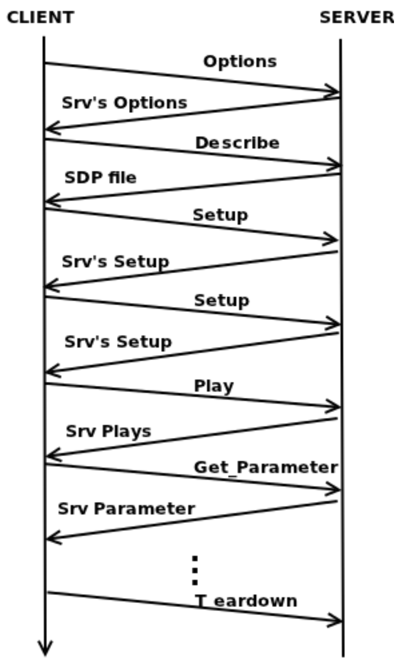


Figure 2.7: RTSP Sequence 1

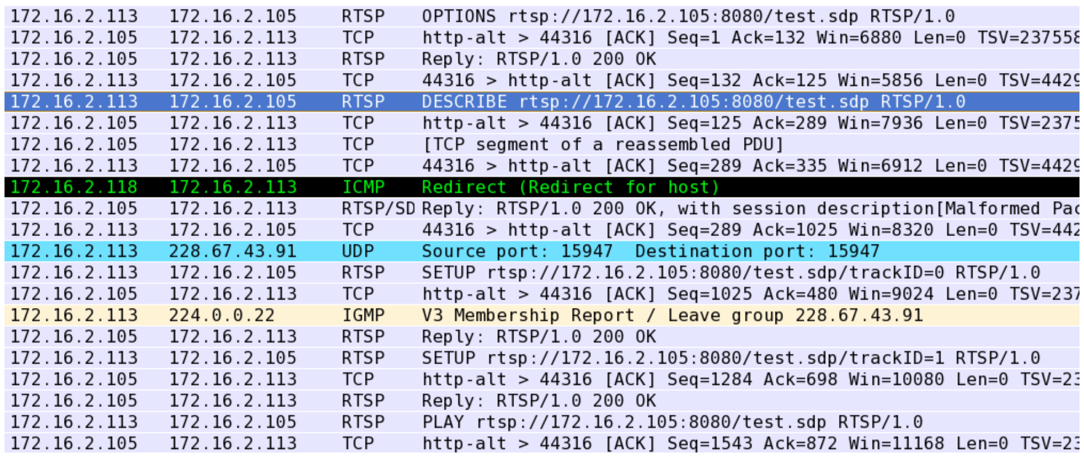
Initially the client asks the server for available control commands with the command *Option*. Then he asks for a description of available sessions or specific URL and this is where the Session Description Protocol comes in. The SDP another protocol used at the application level to describe the media communication sessions, with the intent of session announcement, session invitation and also parameter association. It does not deliver the media itself, it is only used to negotiate between end points all the media properties involved in the communication.

After receiving the description, the client needs to specify how the media is going to be transported, so he uses the *Setup* command, requesting either a raw UDP or RTP connection. Then he starts the streaming issuing the *Play* control, and begins to receive. The *GetParameter* is a control used to check the server's liveness. More commands could be performed, like *Pause*, *Record* and *SetParameter*.

In figure 2.8 we can see all the RTSP commands, exchanged between client and server.

The most important feature in our VOD application will be the support for trick play functionally. Meaning that besides normal Playback, the user should be able to fast-forward and fast-rewind the video, plus seeking desired time references. For all these to happen two concepts are present in the RTSP's RFC, *Range*, and *Scale*. Both information should be used in a Play command according to the desired output. Normally the range information is present in the Session description. The first Play command would look something like this:

```
PLAY rtsp://172.30.41.186:8554/Redbull_720.ts/ RTSP/1.0
```



```
172.16.2.113   172.16.2.105   RTSP    OPTIONS rtsp://172.16.2.105:8080/test.sdp RTSP/1.0
172.16.2.105   172.16.2.113   TCP     http-alt > 44316 [ACK] Seq=1 Ack=132 Win=6880 Len=0 TSV=23755
172.16.2.105   172.16.2.113   RTSP    Reply: RTSP/1.0 200 OK
172.16.2.113   172.16.2.105   TCP     44316 > http-alt [ACK] Seq=132 Ack=125 Win=5856 Len=0 TSV=442
172.16.2.113   172.16.2.105   RTSP    DESCRIBE rtsp://172.16.2.105:8080/test.sdp RTSP/1.0
172.16.2.105   172.16.2.113   TCP     http-alt > 44316 [ACK] Seq=125 Ack=289 Win=7936 Len=0 TSV=237
172.16.2.105   172.16.2.113   TCP     [TCP segment of a reassembled PDU]
172.16.2.113   172.16.2.105   TCP     44316 > http-alt [ACK] Seq=289 Ack=335 Win=6912 Len=0 TSV=442
172.16.2.118   172.16.2.113   ICMP    Redirect (Redirect for host)
172.16.2.105   172.16.2.113   RTSP/SD Reply: RTSP/1.0 200 OK, with session description[Malformed Pa
172.16.2.113   172.16.2.105   TCP     44316 > http-alt [ACK] Seq=289 Ack=1025 Win=8320 Len=0 TSV=44
172.16.2.113   228.67.43.91   UDP     Source port: 15947  Destination port: 15947
172.16.2.113   172.16.2.105   RTSP    SETUP rtsp://172.16.2.105:8080/test.sdp/trackID=0 RTSP/1.0
172.16.2.105   172.16.2.113   TCP     http-alt > 44316 [ACK] Seq=1025 Ack=480 Win=9024 Len=0 TSV=23
172.16.2.113   224.0.0.22     IGMP    V3 Membership Report / Leave group 228.67.43.91
172.16.2.105   172.16.2.113   RTSP    Reply: RTSP/1.0 200 OK
172.16.2.113   172.16.2.105   RTSP    SETUP rtsp://172.16.2.105:8080/test.sdp/trackID=1 RTSP/1.0
172.16.2.105   172.16.2.113   TCP     http-alt > 44316 [ACK] Seq=1284 Ack=698 Win=10080 Len=0 TSV=2
172.16.2.105   172.16.2.113   RTSP    Reply: RTSP/1.0 200 OK
172.16.2.113   172.16.2.105   RTSP    PLAY rtsp://172.16.2.105:8080/test.sdp RTSP/1.0
172.16.2.105   172.16.2.113   TCP     http-alt > 44316 [ACK] Seq=1543 Ack=872 Win=11168 Len=0 TSV=2
```

Figure 2.8: RTSP Sequence 2

```
CSeq: 5
User-Agent: LibVLC/1.1.9 (LIVE555 Streaming Media v2010.11.17)
Session: 2C559161
Range: npt=0.000-
```

This means we would want to play the video starting from position 0. Using this range option, we could also specify a range in the middle of the video timeline, providing us the seeking ability. The fast-forward and fast-rewind ability belongs to the scale option. Normally this value is 1, which means normal playback. If this value is 2, it means the video ratio should be twice the normal playback rate. The same applies to negative numbers.

The play command for fast-forward should look like this:

```
PLAY rtsp://172.30.41.186:8554/Redbull_720.ts/ RTSP/1.0
CSeq: 7
User-Agent: LibVLC/1.1.9 (LIVE555 Streaming Media v2010.11.17)
Session: 2C559161
Scale: 1.5
```

The RTSP protocol provides ways to use this kind of features, but the RFC only comes with suggestions on how to implement this type of situations in the a media server. How this request is processed in the server side remains a question only developers can answer.

2.4 Web Services

Web Services can be seen as a set of technologies based on a message-type design and most widely used on the web in enterprise solutions. It is maintained by the World Wide Web Consortium (W3C). Web services are a Service-oriented architecture directed for the web. SOA defines that applications are built based on services. Each service is nothing but a business functionality (e.g retrieving a picture, movie information). This has enormous potentials. Companies can provide these services allowing others to reuse their existing assets in different applications, leveraging investments.

Web services technology basically allows to implement a machine-to-machine interaction on the web. Traditionally this has been implemented using HTTP and eXtensible markup language (XML). Web services offer more powerful tools such as the Simple Object Access Protocol (SOAP 2.4.2) and Web Services Description Language (WSDL 2.4.1). In the Web 2.0 movement the big web enterprises started to develop open Application Programming Interfaces using this technologies, so developers could create new services. The most used protocols nowadays are: SOAP and Representational State Transfer (REST).

2.4.1 WSDL

As web services and communications protocols were being standardized, with so many new services being created everyday, it became important to describe this communications in a structured way. WSDL [26] accomplishes this goal using XML to describe network services as a collection of network endpoints, called ports, capable of exchanging messages. Ports can be described as operations supported by the service. It is mostly used together with the SOAP protocol. Each client needs to access this WSDL file in order to know which operations are available in the service, and which variables each each one works with (see figure 2.9). Technically, WSDL defines a binding mechanism to relate a specific protocol, data format or structure to an operation.

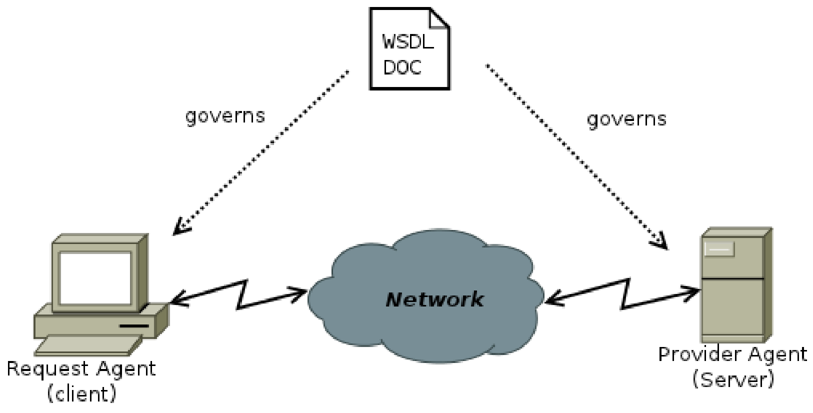


Figure 2.9: Web Services Environment

An example of a WSDL document follows:

```
<definitions>
        <types>
                definition of types........
        </types>
        <message>
                definition of a message....
        </message>
        <portType>
                definition of a port.......
        </portType>
        <binding>
                definition of a binding....
        </binding>
</definitions>
```

Types defines each type of object that the service will use. Normally each exchanged *message* uses a specific type. The *portType* section is where the all the service operations are defined. Each operation may have an input and/or output message with the specific type of object. In the end all the services need to have a description of how they will be implemented on the Internet.

2.4.2 SOAP

Simple Object Access Protocol [27] is a simple way to exchange structured and typed information between agents in a spread environment using XML grammar. It does not specify a programming model or implementation semantics but rather defines a way to encode data in packed modules. With this, SOAP can be used in a large variety of services.

The SOAP message consists of a mandatory Envelope with a SOAP Body and an optional SOAP Header. This envelope is the element that identifies the XML document as a SOAP message. The SOAP Body contains the call and response information. A specific element was defined in the SOAP Body to handle error and status information, the Fault element. The SOAP Header provides a way to add extensions without the prior knowledge of both agents, examples include authentication, payment etc.

```
<?xml version="1.0"?>
<soap:Envelope
        xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
                soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
        <soap:Header>
                ...
        </soap:Header>
        <soap:Body>
                ...
                <soap:Fault>
                        ...
                </soap:Fault>
        </soap:Body>
</soap:Envelope>
```

The SOAP messages are exchanged through HTTP requests like HTTP POST or HTTP GET.

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
<?xml version="1.0"?>
```

```
<soap:Envelope
        xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
        soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
        <soap:Body xmlns:m="http://www.example.org/stock">
                <m:GetStockPrice>
                        <m:StockName>IBM</m:StockName>
                </m:GetStockPrice>
        </soap:Body>
</soap:Envelope>
```

The server then processes the request and answers with an HTTP response.

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
<?xml version="1.0"?>
<soap:Envelope
        xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
        soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

        <soap:Body xmlns:m="http://www.example.org/stock">
                <m:GetStockPriceResponse>
                        <m:Price>34.5</m:Price>
                </m:GetStockPriceResponse>
        </soap:Body>

</soap:Envelope>
```

2.4.2.1 WS-Stack

Initially SOAP was designed by two big companies, IBM and Microsoft. This companies used SOAP protocol to provide other Enterprises the benefits of Web Services, with a limited set of requirements. Among other things developers were concerned with reliability and security aspects, also responding to the costumers and the companies needs. This meant that for each new aspect the protocol didn't provide support for, a new set of rules had to be created. Today SOAP has a long list of tools, and many open source products do not fully support every single aspect of the SOAP protocol capabilities.

The result was a complete set of documents containing specifications to achieve different types of needs. The documentation is currently managed by the OASIS, a group responsible for advancing open standards for the information society. Some of this specifications are: SOAP Message Security 1.1 [28] (WS-Security 2004), WS-SecurityPolicy 1.2 [29], WS-SecureConversation 1.4 [30], WS-Trust 1.3 [31], SAML Token Profile 1.1 [32] and finally WS-Addressing [33]. All these specifications appear in the Header of the SOAP envelope.

The SOAP Message Security objective is to provide three mechanisms: send security tokens as part of a message, message integrity with the use of signatures and timestamps, and message confidentiality. For example one can send a simple username token, or a username token with a signature for veracity purposes. The last part specifies that both cases can be encrypted with certain keys to provide message confidentiality. This mechanisms can be used with others to provide application-level security. WS-Security among with a UsernameToken profile and a KerberosToken Profile create the Basic Security Profile (WS-I BSP) 1.0 [34].

WS-SecureConversation is an extension of WS-Security. WS-Security is subject to several forms of attack as stated in the Security Considerations section of the WS-Security specification. According to those disadvantages WS-SecureConversation introduces a security context establishment and sharing, and session key derivation and respective passage.

WS-Trust is also an extension to WS-Security. WS-Security defines basic methods for secure messaging. WS-Trust defines additional rules to permit dissemination of credentials within trusted domains. It defines methods for issuing, renewing and validation security tokens, and to realize the presence of trusted or rogue relationships.

WS-SecurityPolicy is a set of specifications that provide web services the ability of stating what type of security policy they want within a SOAP message. They can state both constraints and requirements, and these policy assertions are specific to security features provided by SOAP Message Security, WS-SecureConversation and WS-Trust. Usually policy requirements can be included in the WSDL file regarding the policy for an individual *port* or for all the existing *ports*.

Security Assertion Markup Language is an open standard assertions for authentication and authorization exchange. The SAML Token Profile uses this with or without WS-Security procedures assertions as security tokens from the SOAP Message Security header blocked. The last one, WS-Addressing, specifies a convention to identify endpoints, giving information that is typically provided by the transport protocols.

Two other specifications are also important to mention. Normally they come in extra frameworks while the others are bundled all together in one single tool. These are WS-ReliableMessaging [35] and WS-Eventing [36]. WS-ReliableMessaging allows messages to be reliably delivered when any system, network or software failure occurs. Sometimes when people are interested in other events and occurrences in other services, they want to be notified. WS-Eventing provides a mechanism for registering interest, like a subscription, next to the desired service.

All these Profiles result in an extensive and sometimes confusing set of rules for the SOAP utilization. And these are all just Application-level Security Profiles. Besides these profiles we can also introduce a Transport Security Layer. In other words SOAP services can be used over HTTPS. In this case WS-Security is not needed at all, reducing complexity while gaining performance. However the presented WS-Stack covers issues of integrity and confidentiality that HTTPS doesn't.

2.4.3 REST

The Representational State Transfer [37] was introduced by Roy Fielding. This architecture has nothing to do with the other webservices described above, but it shares some similarities like the use of XML as a document format and the use of HTTP forms. The motivation for REST was to capture the characteristics of the Web which has highly desirable architectural properties: scalability, performance, security, reliability, and extensibility.

There are four basic commands that REST uses: HTTP GET, POST, PUT and DELETE. This brings interesting properties, for example, HTTP GET has no side effects as it is only an information retrieval and a very simple one. However the goal of this architecture is not only to gather information but to process, update and delete resources. Any information that can be named can be a resource: a document, an image, a stock information. This resources are identified by a Uniform Resource Identifier (URI).

Example of an information retrieval:

http://stockquoteserver.example/query?symbol=MSFT

WSDL can also be used in REST services using an HTTP binding and all methods are supported:

```
<binding name="HttpBinding"
              interface="m:GetTemperature"
              type="http://www.w3.org/2005/08/wsdl/http">

      <operation ref="m:location" whttp:method="GET"
                      whttp:location="{country}/{city}"/>
</binding>
```

This allows for an HTTP GET on http://weather.example/Sweden/V%C3%A4xj%C3%B6 with the respective response:

```
HTTP/1.1 200 Here's␣the␣temperature␣for␣you
Content-Type:␣application/xml
...
<weather␣xmlns="...">
␣␣␣␣␣␣␣␣<temperature>23</temperature>
</weather>
```

2.5 Software Streaming Solutions

2.5.1 Darwin Streaming Server

Darwin Streaming Server [41] is the open source version of Apple's Quicktime Streaming Server. It uses RTP and RTSP protocol to deliver media streams to clients across the Internet and with its webadmin interface it provides a highly configurable environment. Various platforms are supported and it is intended to stream Quicktime and MPEG-4 media. Features like Authorization, Spam Defense and RTSP redirection are included.

2.5.2 VLC

Almost everything that VLC plays it can be also streamed. VLC [42] is the most famous open source media player and was created in 2001. It can play almost any media file available, stream most of what it plays and can be used in every platform Windows, Mac, Linux, Unix, etc.

For encoding and decoding most of it video files it uses a library called libavcodec from the FFmpeg project, but it also includes its own muxer and demuxers. For its serving capabilities VLC uses the LiveMedia library from LIVE555 to support RTSP, RTP and SDP. The current stable version of VLC - libVLC 1.1.9 - supports most of the trick play functionalities. However some tests demonstrates a few bugs when streaming the desired content like incorrect timeline. The current version of VLC doesn't support Fast-Rewind.

2.5.3 LIVE555 Media Server

LIVE555 Media Server is a complete RTSP server based on the LIVE555 Streaming Media [43] library. This includes source-code set of C++ libraries for multimedia standards RTP/RTCP/RTSP suitable for embedded streaming applications. It can stream TS and H264 elementary files, among others. There's also the possibility to stream to set-top boxes that require raw UDP streaming, rather than standard RTP streaming. It can also stream its RTP(and RTCP) packets over TCP for firewall purposes.

The server supports RTSP *trick play* functionality for some media types. Seeking , Fast forward and Reverse play is possible for TS files for example. The trick play functionality in the Live555 Streaming Media library follows the RTSP specification. For the Live555 trick play functionality to be available, an index file is required in order to map a video file position with a playing time. Each index represents a chunk of video data that appears within one TS packet. The tool provided to create this index file, analyses the TS packet to find I-frames and indexes it to be able to start always with a clean picture, whenever a trick play is requested (e.g Seeking a position in the middle of the video timeline). When we want to fast forward, the server constructs a new Transport Stream that it's made up from I-frames taken from the original stream.

The latest release included a tool to wrap H264 elementary streams to TS containers to take advantage of the trick play functionality. Other encoding and decoding tools are also available as test programs, which is very useful.

2.6 Programming Tools

2.6.1 HTML5

HyperText Markup Language (HTML) has been in use in the world wide web since 1990. Twenty one years have passed and a lot has changed. This standard was first introduced by Tim Berners-Lee and published in 1995 as HTML 2.0 by the Internet Engineer Task Force (IETF) in the RFC 1966 and has had several improvements over the years. The later HTML 4.0 has some features that now are obsolete and the need to improve some characteristics was increasing.

In 2009 the group that was developing the HTML joined with W3C to create the next generation of HTML the HTML5 [38]. This standard is not yet official because it's considered a work in progress, however, most modern browsers have some HTML5 support. The goal of this new standard was to handle today's internet use and so it needed to follow a few rules to prevent some mistakes of the past. For example, it was established that there was a need to reduce external plugins (like Flash), that the standard should be device independent to be able to adapt to any application and it should have better error handling.

New interesting features are now included in HTML5: the video and audio elements are incorporated for media playback, new input type attributes were created, new content specific elements were introduced like the article, footer, video, audio, progress and many others. New local data objects were created to handle large amounts of data because the previous feature, cookies, was not suitable. In the event section several new events were created to deal with window events, media events, keyboard and mouse events.

2.6.2 CSS3

Cascading Style Sheets [39] defines a way to display HTML elements. HTML was intended to contain the content of a document and never to contain tags for formatting. Adding color tags and other formatting tags brought a lot of difficulties to web developers. These styles were added by W3C to HTML 4.0 to overcome those problems and to save lot of work in design implementation. The external style sheets are stored in CSS files.

CSS3 is an improvement of past releases and is divided in several modules: Selectors, Box Model, Backgrounds and Borders, it's able to deal with Text Effects, Animations and much, much more.

2.6.3 PHP

The Hypertext Preprocessor [40] is an open-source scripting language that can be embedded into HTML to provide a dynamic web page creation. Instead of writing lots of commands to output HTML we include instructions that do "something". Using this language is very easy and brings a lot of new features to HTML design.

One advantage in using PHP is that many webservices can be easily developed because it supports several protocols. It can be used in all major operating systems, including Linux, and it has support for most of the web servers today.

2.6.4 Qt

Qt [44] is a programming framework that brings several libraries to support multiple features and facilitate their integration in our applications. It has been in the market for 15 years and since then it has seen a major development. It's indicated for advanced and highly innovative applications and devices. For example VLC 2.5.2, one of the most used video players, uses Qt for its visual engine both in Windows and Linux.

The success behind this framework is that it brings all the tools needed to develop advanced GUI applications with embedded multimedia characteristics. The use of native APIs of each supported platform provides full advantage of the system resources with a native look and appeal. The

Phonon Multimedia framework library makes it easy to include audio and video in Qt applications and besides that Qt also brings a native XML support library.

In terms of licensing Qt as three strands:

- Commercial license where we can create proprietary applications without the obligation to share the source code and modifications;
- LGPL license where proprietary applications are possible but under the LGPL license and all the source code must be provided;
- GPL license does not give the possibility of proprietary applications and all source code must be provided.

2.6.5 JavaFX

JavaFX [45] is a way to create expressive, multi-rich content which brings capability and performance to our applications. It uses a set of essential technologies, tools and other resources required to develop and create powerful content. It has also lots of flexibility because of its intuitive Java platform. The main goal of this new tool was to compete face to face with Flash technology. Flash technology also aims to deliver the best web experience across different platforms.

Figure 2.10 shows an example of how the JavaFX architecture is structured. A new concept of scene graph is introduced, meaning the components are organized in a hierarchical tree of nodes. We can see the similarities it shares with Flash. First we have our window called *Stage*, in our *Stage*, there's a *Scene*. This *Scene* can have whatever contents we desired, much like the Swing API existent in Java. In JavaFX is also possible to have KeyFrames and Timelines to apply any desired effect.

But the JavaFX current stable version - 1.3.1 - can only be used within the JavaFX script. JavaFX script was a new language created to support the JavaFX API. It is possible to reuse Java code but this only possible with the use of a JavaFX transport API's. Since it's creation users have tried and achieve different ways to interconnect JavaFX language and Java, to have the best out of two worlds. However the JavaFX script will be discontinued, because the idea of having this technology in multiple screens wasn't possible to achieve. This means that from now on the new JavaFX API 2.0 will be integrated with the current Java Development Kit. Another improvement in the future release is the hability of javaFX to use the new Prism hardware pipeline if the system supports it. This will provide even better graphics acceleration.

The Java language derives much from C and C++ but it has simpler object model and fewer low-level facilities. This language was created by Sun in 1995 and it is known for its capability to run in any platform.

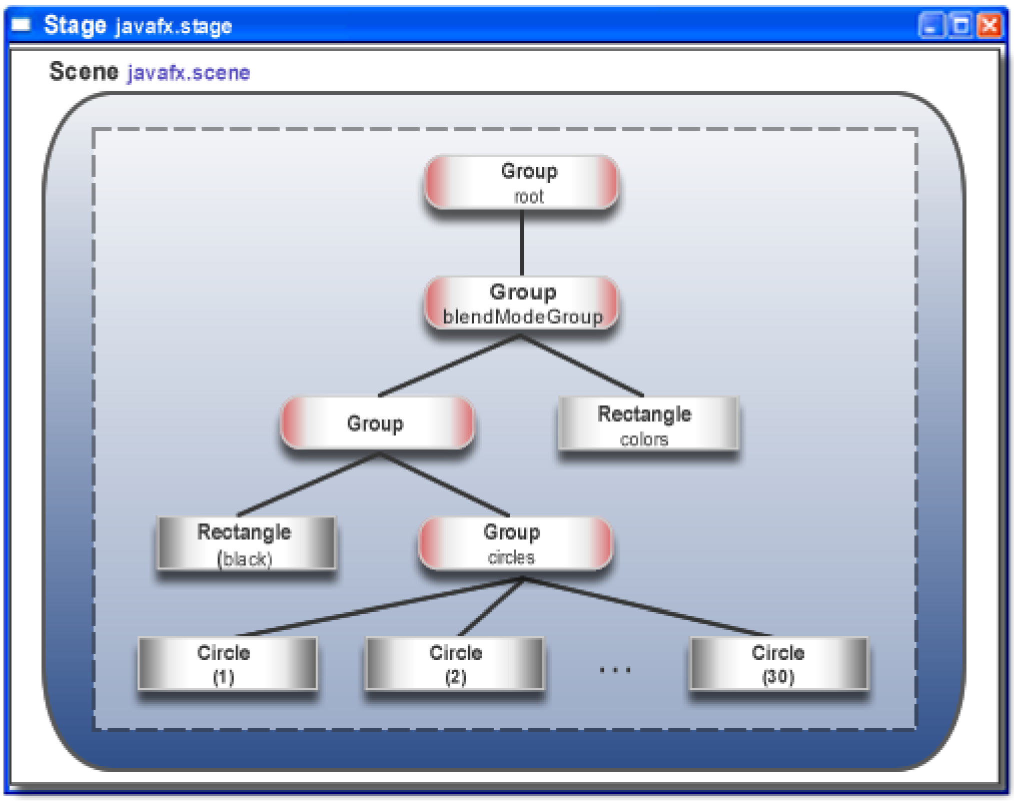


Figure 2.10: JavaFX Scene Graph

2.7 Conclusion

All the researched tools described in this section will provide the best options in order to make our work easier and to create a robust system. Only the open-source tools will be used during the development, aiming for a commercially competitive and viable product. The selected Video Codecs and Containers will be used in the streamed video files. The described Streaming Software is capable of handling all the required Network Protocols, especially RTP and RTSP, with trick play functions for the specified video formats. Finally all the functionalities regarding information retrieval, like metadata, will be accessible through the specified protocols.

Chapter 3

Software Requirements Specification

This chapter describes how the system has been implemented. It includes an overview of the system architecture in section 3.1, describing its components and their roles. The system is to be composed of three modules: the *Server*, the *WebAdmin Application*, to control the Server and the *User Application*. All these applications must run in a Linux environment and use open-source software for development.

All the system requirements are defined in section 3.2 and then the captured use cases in section 3.3. The database present in the server's main interface is described in section 3.4. One important topic of this service is the Web Services Application Programming Interface (API) defined in section 3.5, created to provide a set of rules to use the services and resources that this system offers. This API has three relevant groups, the *Administration* and the *Client* interface and the generic *User*. A WSDL file has been created to describe this set of rules of our web service.

3.1 System Architecture

3.1.1 Server

The Server is responsible to address all the user agents and to manage the streaming server. To make all this possible, a Main Interface is responsible to interconnect all the parts of the system as shown in figure 3.1. Among this parts we can include: a web services interface to communicate with both actors, administrator and user; a database to store all the user's information and video metadata; a streaming server responsible to stream the video content to the clients interface through the appropriate channels. Every actor will first connect with the main interface to retrieve/manage the information.

Today, the most used approach in this kind of environment is the content delivery network (CDN). CDNs have several servers spread around the network close to the costumers to provide a better streaming solution. If the usage of one server starts to increase, another server and network connection can be setup to provide sufficient bandwidth for a reliable service.

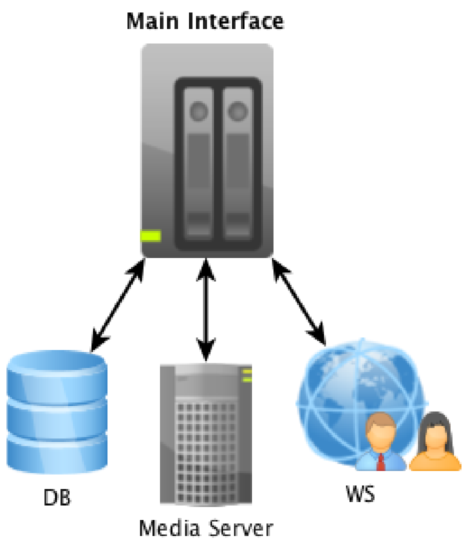


Figure 3.1: Server's Diagram

3.1.2 WebAdmin

To customize and configure the servers behaviour and working requisites, a web interface shall be created. This environment consists of a web page designed with HTML5, CSS3 and a PHP framework to implement all the needed tools. The interface will communicate with the server via a web services API and it's important to highlight that this type of dimensioning will give us a decentralized solution. With the admin interface, video files and metadata which can be retrieved from the appropriate websites, will be uploaded to the media servers local storage and main interface's database respectively. The video's information must be retrieved from official websites like IMDB, this information is required to be serialized by the admin's interface and then uploaded to the servers DataBase. An overview of the WebAdmin interface can be seen in figure 3.2.

3.1.3 User

The User's major concern is to properly play the video stream. However this application should use the state-of-the-art tools to create the best multimedia experience and usability. This is called Rich Internet Application (RIA). One of the programming tools ideal for this part of the project is the JavaFX platform because of its capability to bring a feature-rich application. Every communication will start in the client. After a successful login to the services, he will be able to search for the content available in the server. When it chooses a movie/series, all the respective information will be shown, and if it's eligible he can play the content. A user's diagram can be seen in figure 3.3.

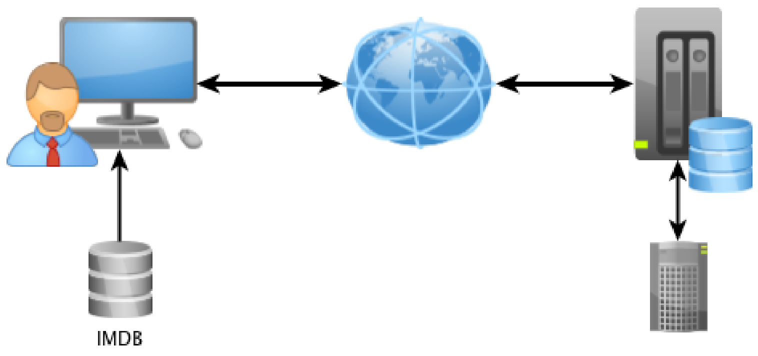


Figure 3.2: WebAdmin's Diagram

3.2 System Requirements

The functional requirements define the basic functions of the system and how it behaves under specific circumstances. This section lists the requirements identified for the system thus enabling to identify the functionality that the system will need to implement in order to support them. These requirements are then captured in a set of use cases assigned to the identified actors of the system or to modules of the system.

3.2.1 Server

1. The main interface will communicate with both actors through a web services interface.
2. The main interface shall have a database to store all the clients and the video contents information.
3. Every communication starts in the client or the admin.
4. The main interface is responsible to control and monitor the streaming server.
5. The streaming server must be a stand-alone application and provide feedback to the main interface; this information shall only be accessible to the administrator.
6. The streaming server must support RTSP along with trick play functionality.
7. Every multimedia stream must use Transport Stream container.
8. Every multimedia file must be codified with H264 codec.
9. All the videos must be uploaded and stored in the media server's computer; the appropriate format should be created by the main interface in case the video file is not in according to the video file specification.
10. Every video file must have a unique identifier.

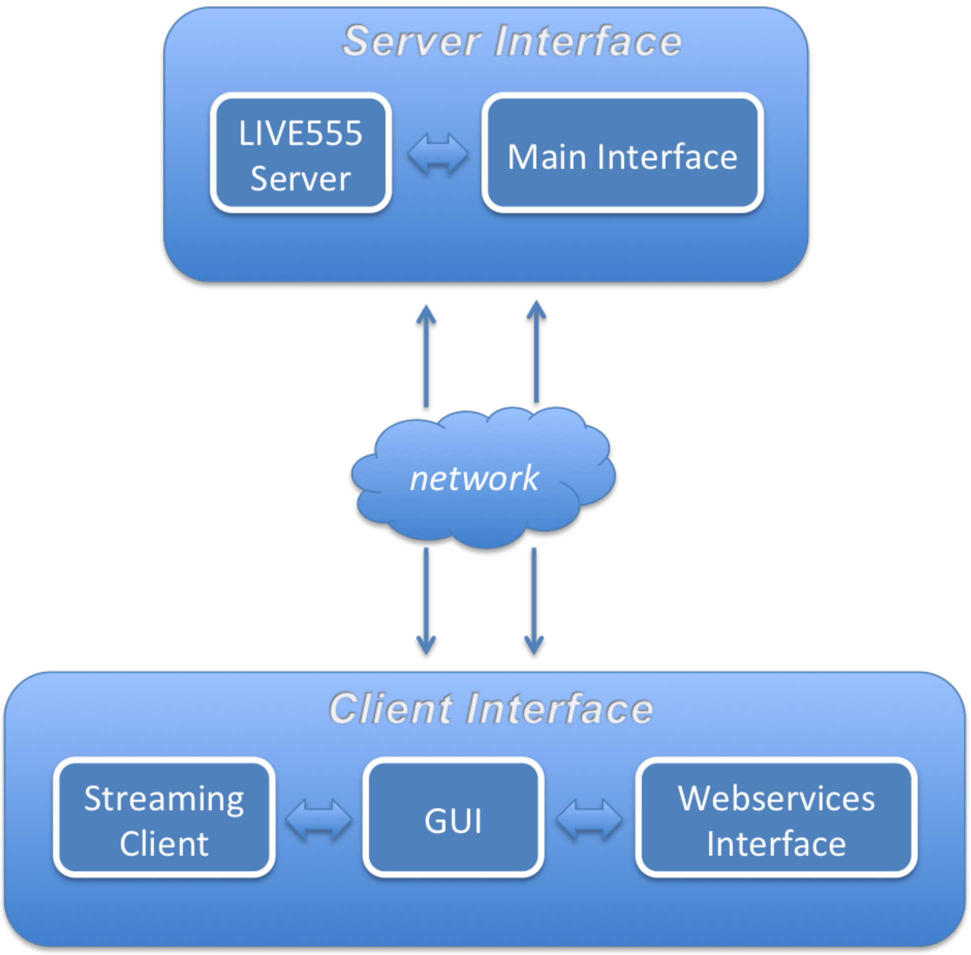


Figure 3.3: User Application Diagram

11. A TCP/IP connection will be used to the main interface and the streaming server's communication.
12. To describe the web services a WSDL file must be used.
13. To provide the web services a SOAP web server must be implemented.

3.2.2 WebAdmin

1. This application shall use a web based interface.
2. All video files must be uploaded through this interface.
3. An authentication process must be implemented.
4. The interface shall provide appropriate accessibility to all the content information and streaming server's feedback .
5. The administrator must be able to collect and serialize all video information properly, and upload it to the server's interface.
6. The video information must be collected from IMDB.
7. To connect to the server the web service's API will be used.

3.2.3 User

1. An authentication process must be implemented.
2. All video content available must be displayed in the client interface.
3. When requested by the user the video information must be visualized, including the associated image.
4. The user must be able to play and pause the video stream including the ability to seek, fast-forward and reverse-play the stream.
5. To access to the server's information and stream URL the web service's API will be used.
6. To play the multimedia stream a RTP/RTSP connection must be used in the client interface.

3.3 Use Cases

This section refers to all the use cases the users will be able to engage in their respective interface. Use-case diagrams are provided for each available interface. Tables are provided, containing a more detailed description of each use case. The complete set of defined use cases captures all the functional requirements listed in the previous section.

3.3.1 User Application

As in every application across the Internet, security is an important issue. In order to protect the service from unwanted access, the server needs tools to assure that only authorized users have access to the application, or to determine which contents the user is authorized to view. This means the application should have an authentication process, depending on the web service's platform the service is built in.

The main purpose of the user interface is to provide an entry point to the user into a "wall of movies" presenting minimum descriptive information about each of the available movies and also more detailed data as the user selects one given movie title. A picture of each movie should be initially presented, together with basic information such as name and the year of the movie. Usually there can be movies with the same name but made in different years. After selecting the desired movie, more detailed information should be presented to the user. For example, actors that performed in the movie, who was the director, the rating that other user's gave to the movie, a synopsis, etc. Only then the play option should be available. After requesting to play, a movie player should pop up, giving the user the opportunity to view the desired video with every trick play option available. Figure 3.4 shows the user's use cases.

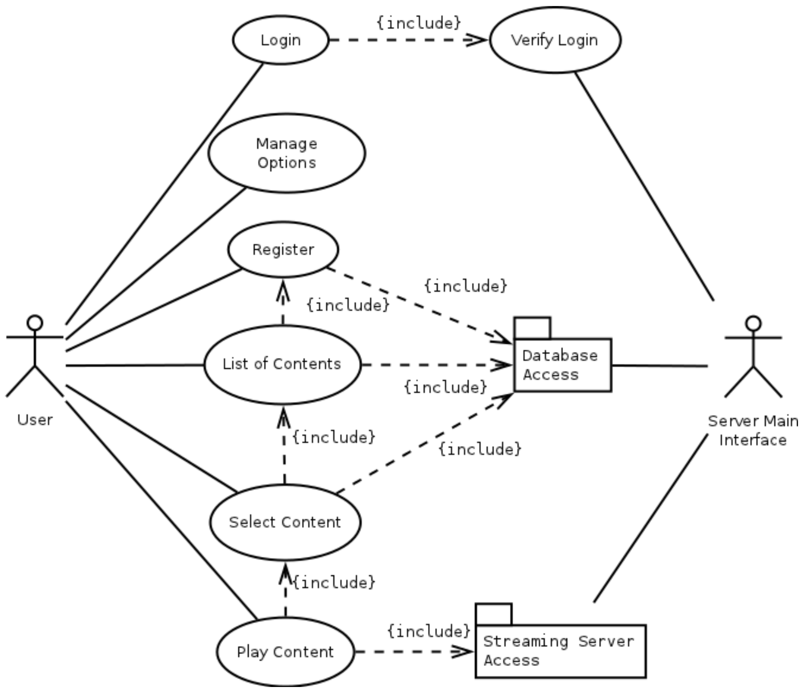


Figure 3.4: User Application

Table 3.1: User's use cases

Register	The client needs to register before being able to access all the content.
Manage Options	The client can setup configuration options like viewing parameters.
Login	The client needs to authenticate to the server.
Verify Login	The main interface needs to verify the client's login.
Play Content	The client requests to play a video stream.
Select Content	The client selects content from the content list provided, to view more detailed information and to play the video stream.
List Available Contents	The client can list all available contents provided by the VOD service.

3.3.2 Admin Application

The Administration application is the most important aspect of the hole system. It needs to control both the main interface information, as well having access to each and every media server spread across any part of the world. With this type of scalability, security is even more important. The admin should not connect directly to a media server but through the Main Interface only.

The interface will be divided into two parts: the Content Management and the Session Management. In Content Management, the administrator can upload a video into the media server and add it to the Main interface database. Then list all the available videos in the main interface and edit each video information. The Content's list comes from the advantage of the administrator being a common user. While editing, the interface will gather the desired movie content from the IMDB webpage. In Session Management, the administrator can load or terminate each video session, choosing for that matter any media server available. This interface should provide a way for the admin to load each media server's information into the Main Interface database, and retrieve it at any time.

3.3.3 Server Application

3.3.3.1 Database Access

The service database is located in the server's Main Interface. The only way to load any information to the database is to connect with the Main Interface.

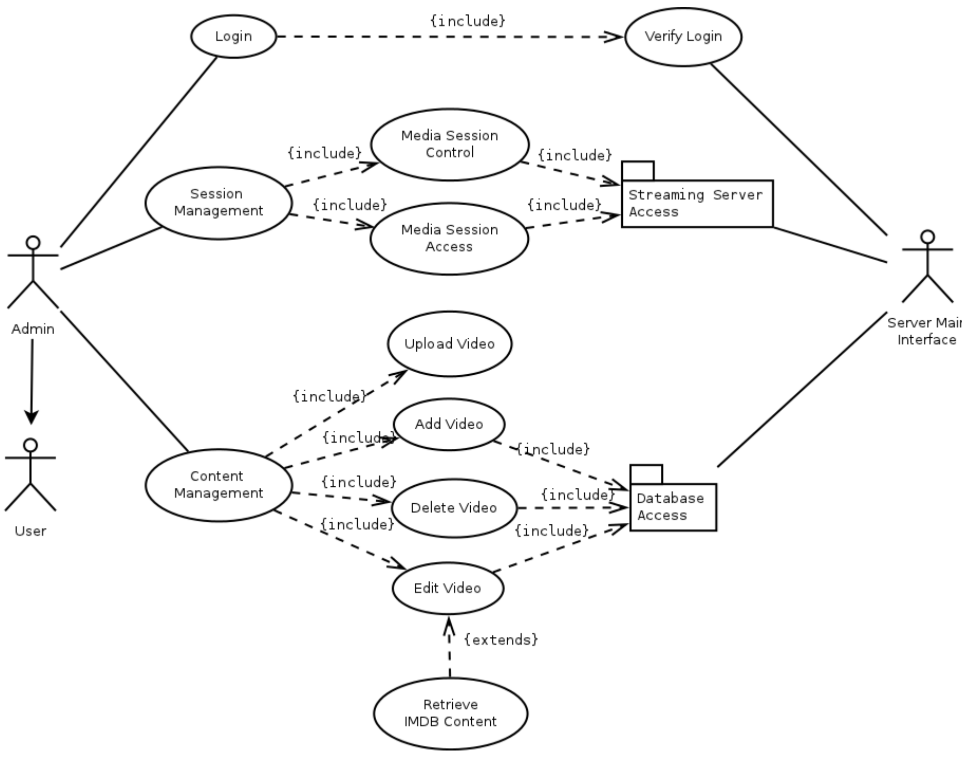


Figure 3.5: Admin Application

Table 3.2: Admin's use cases

Login	The admin needs to authenticate to the server.
Verify Login	The main interface needs to verify the admin's login.
Content Management	The admin has access to a variety of administration tools in order to manage the service's content.
Upload Video	The admin can upload a file to the server.
Add Video	The admin adds a video to the Main Interface database.
Edit Video	The admin retrieves the movie information from the appropriate website and loads the information into the Main Interface database.
Delete Video	The admin deletes a movie, and any related information that is no longer needed.
Retrieve IMDB content	The admin can retrieve information from the Internet Movie Database Site.
Session Management	The admin can manage each Media Sessions
Media Session Control	The admin can load a video file to the media server, terminate one and retrieve a media session Url.
Media Session Access	The admin can give each user access to any media server available.

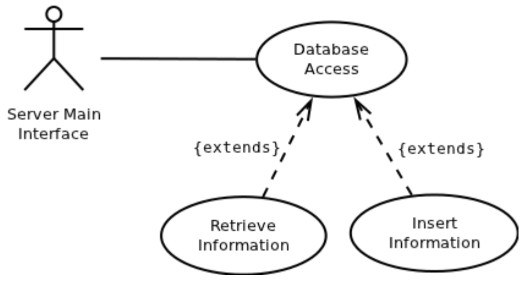


Figure 3.6: Database Access

Table 3.3: Database's access use cases

Database Access	The main interface accesses the database to retrieve or insert information.
Retrieve information	The main interface accesses the database to query information.
Insert information	The main interface accesses the database to insert information.

3.3.3.2 Streaming Server Access

The access to the Media Server's information and controls is only possible through the Server's Main Interface. Each Media Server has a Media Session. In each Media Session there's a Sub Media Session, which corresponds to each video loaded into the Media Session. There are two options, one is to load and the other is to terminate the Video Session. A detailed figure can be observed in 3.7. If one Video session does not exist, it won't be possible to retrieve the respective URL information for viewing purposes.

Table 3.4: Streaming Server's Access use cases

Media Session	The media server's main session, where all videos and user information is loaded.
Load Video Session	The server's main interface can load a video stream into the server's Media Session.
Get Media Url	Retrieve the Media URL in order to play the desired content.
Add access	Give a user access to a Media Server's Session.
Remove access	Remove a user's access to a Media Server's Session.
Terminate Video Session	The server's main interface can terminate a video stream in the server's Media Session.
Media Server information	The server's main interface can collect several information from the Media Server.
System Statistics	Access the Media Server's system information and statistics.

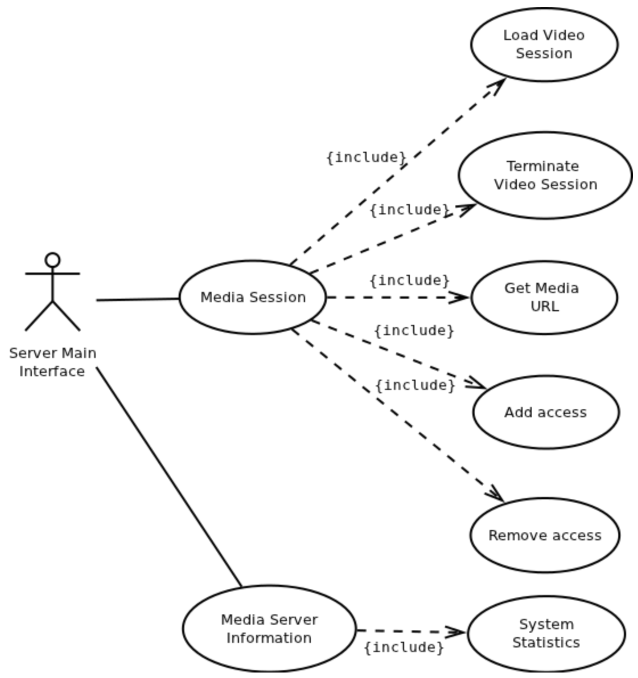


Figure 3.7: Streaming Server Access

3.4 Database Modelling

As referred in the previous section, the Main Interface will provide the access to the media repository. In an IPTV environment, such repository can contain movies, series or live broadcasts. Given that this project is focussed on the Video On Demand service, the implemented repository contains only movies.

To easily access the stored movies and provide descriptions of the movies to the users so that they may be well informed when selecting a movie for playout, additional, descriptive data (metadata) should be stored alongside the media data.

Movies can be characterized by means of a variety of information. The most basic information about the movie is its title, but one very important characteristics is the genre which it belongs to. Also, names of persons who participate in the movie can constitute an important information to help users in selecting one movie among several. Different roles can be assumed by those persons, namely, actor, director, producer, screenwriter, etc. The attributes or metadata that are used to characterize each movie is illustrated in the database diagram of figure 3.8. Most of these features are obtained from the Internet Movie DataBase (IMDB) Web site, which is queried by the administrator using the name of the movie, each time he inserts a new title into the repository.

The media server or repository is thus composed of a storage area for the movie data itself and a database for the movies' attributes. The database also includes information about the organization of the media server for management purposes. It allows the system to distribute the load between different servers, directing clients to the server located nearer to him.

Users of the database can be regular users, i.e., clients who wish to select and play-out a movie, or an administrator of the system. In this way, different access permissions are defined for the access to the database contents.

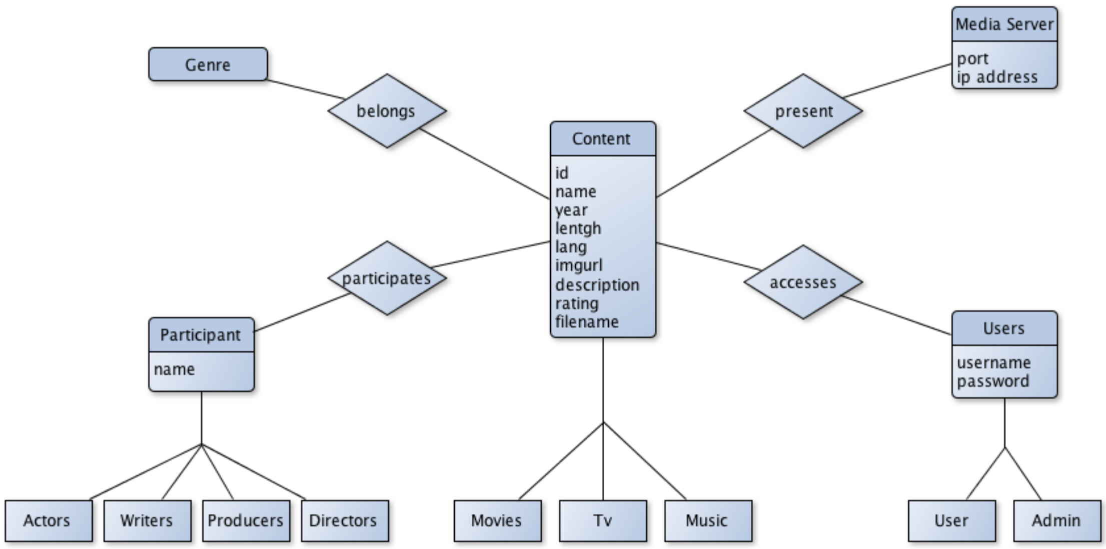


Figure 3.8: Main Interface's Database

- Content (id, filename, name, description, year, length, lang, rating, imgurl)
- Participant(name)
- Genre(genre)
- Media Server(ipaddress,port)
- User (username, password)

  Associations:

- participates (Participant, Content)
- accesses (User, Content)
- belongs (Genre, Content)
- ispresent (Content, Media Server)

3.5 Interface Specification

3.5.1 Web Services API

Web services have shifted the way enterprises conduct their business nowadays. Internet it is not just a collection of pages but a collection of services that interoperate through the Internet.

This part discusses the VOD API as an interface for external services to communicate and use these web services.

The service will be based on two main agents, the *User* and the *Administrator*. The User application will only need a set of tools to reach the content information. Two operations are provided, one to retrieve the movie's list, other to search for each movie information. In the end, to view the selected content's video, the stream url is required.

The administration services are a bit more complex because it has more tools to work with. From providing a user access to any media server, to loading and checking the Media Servers status. Tools to store and retrieve all the media server's list are available too.

User Methods:

- user.getContentList – retrieve the list of available contents from the server.
- user.getContent – retrieve the specified content information.
- user.getMediaURL – retrieve the media URL to play the video stream.
- user.editPassword – change the user's password.

Client Methods:

- client.createClient – register a new client.

Admin Methods:

- admin.addContent – add content information to the server's database.
- admin.editContent – edit existing content information.
- admin.deleteContent – delete existing content information from the server's database.
- admin.loadMediaSession – load a video stream into the server's Media Session.
- admin.checkMediaSession – check if a video stream is loaded into the server's Media Session.
- admin.terminateMediaSession – terminate a video stream from the server's Media Session.
- admin.getStatistics – get statistical information from the Media Server.
- admin.getUsers – get Users information form the Server.
- admin.addMediaSessionAccess – provide each User access to any Media Server.
- admin.addServer – add a Media Server to the database.
- admin.removeServer – remove a Media Server's reference from the database.
- admin.getServers – retrieve a list of all the Media Server's available.

Chapter 4

Development

This sections discusses the development work, describing step by step each solution found to help us create this application and how we overcame certain problems and issues found at the time. The development was based on the Software Requirements Specifications document present in section 2. Figure 4.1 provides a high level architectural view of the system, after the development was over.

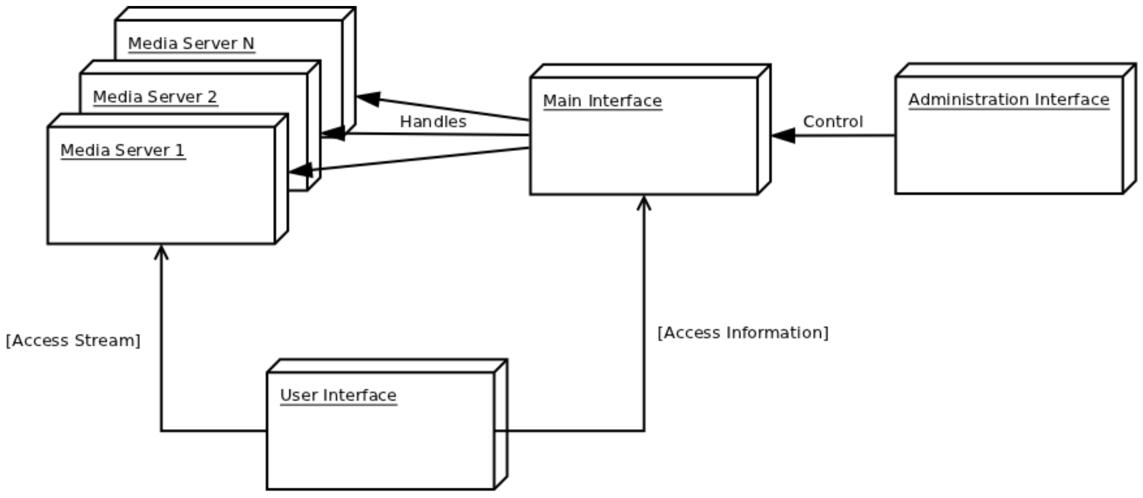


Figure 4.1: System Overview

4.1 Main Interface

As explained before in section 3.2.1 the main interface is responsible for interconnecting all the system's parts, including the database where all the information will be stored. For familiar and interoperability reasons, concerning also the Media Server's native language, the programming language chosen to develop all the Main Interface was C/C++.

The first thing needed for the system to work was a database able to store our information, and retrieve it easily. A static library was a simple yet powerful way to provide all the needed structured query language (SQL). SQLite3 [46] was the chosen library because it had an intuitive C library meaning we could easily build a structured database interface for our system. The Database development description can be found in section 4.1.1.

Secondly, for the system to connect to any Media Server in a secure way it was necessary to create a dynamic and scalable way to achieve that. A basic communication type is the connection-oriented socket, which provides reliable data transmission using for that matter the TCP protocol. Yet, the need for security forces us to use Secure Sockets Layer (SSL) and for that we have OpenSSL [47], an open source toolkit able of delivering robust and commercial-grade applications.

We cannot forget that all the above tools should be exposed as web services so that anyone could reach our application, in any kind of environment. There are many available frameworks that can do the job for us after, of course, some implementation. Searching through many options like gSOAP, Axis/C, Axis/C++, the choice fell on a tool, WSF/CPP, from the WSO2 group, that was basically an Axis/C wrapper. A more detailed explanation can be found in section 4.1.3.

To conclude this introduction an overview of the Main Interface components can be observed in figure 4.2. The next sections will provide more detail regarding each used component.

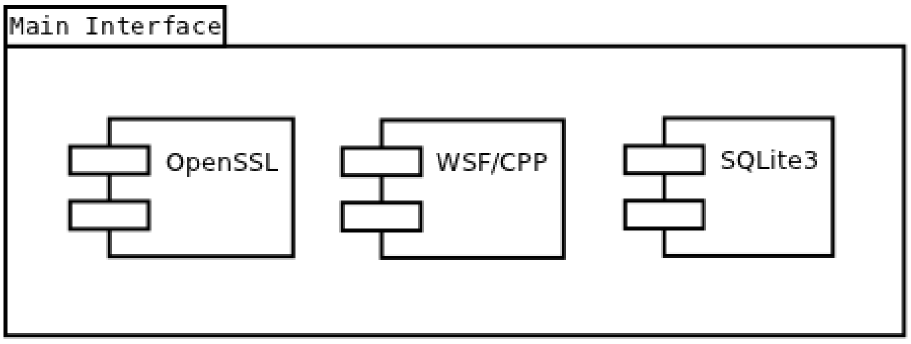


Figure 4.2: Main Interface Components

4.1.1 Database

The main concerns in the database are: availability for users, and full access for the administrator. The information should be available to all the users and at the same time the administrator should both access and edit that information.

When it comes to SQL options MySQL, SQLite3 and PostgreSQL are the most popular ones nowadays. PostgreSQL and MySQL are probably the most advanced databases available and usually they run in a single process, meaning each one is a server itself. They come with features like database replication, allowing several servers to properly achieve load balancing. Other feature is that if a server fails, other can quickly take over in order to provide full availability.

In order to keep things practical in the beginning of this thesis and for demonstration purposes only, a simple solution seemed more convenient. A database with all the SQL features and minimally scalable was enough. This is were SQLite plays its part. SQLite is a serverless database meaning we don't need any extra process besides the ones that already exist, allowing multiple applications to access the database at the same time. Another feature is that this database is self-contained and we can easily duplicate it for backup reasons, without the need for installation procedures. Also, in multi-thread environments SQLite uses Mutexes for thread synchronization and a Lock configuration to prevent race conditions.

When several users are accessing the database SQLite provides five locking states to prevent database corruption. If the database is in a SHARED lock for example, it means that anyone is able to safely read data from the database but no other thread or process is allowed to write. Regarding operations where the administrator needs to edit any information several lock flags need to be acquired. The EXCLUSIVE lock means someone is about to write and no other type of lock is allowed. SQLite tries to minimize the amount of time that EXCLUSIVE locks have.

To dynamically use these locks a transaction method is used. Before doing any operation for updates or insertions a BEGIN TRANSACTIONTYPE is queried. Depending on the transaction type different behaviours can occur. By default SQLITE works in BEGIN DEFERRED where it acquires locks immediately after a read or write operation. In write operations it acquires a RESERVED lock indicating that is planning to write on the database soon. Only when all the queries finish the transactions are committed to the database. This provides a scalable way to write information without compromising other clients that try to access the database.

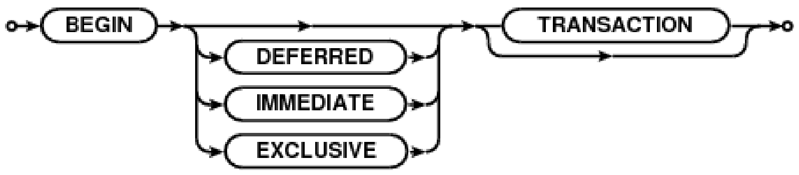


Figure 4.3: SQLite Transactions

These reasons and others lead the SQLite developers to believe that SQLite is the most deployed database in the world [48], with an estimate of 500 million deployments to 100 million of other SQL engines.

As our service will request mostly read-only operations assuming that the administrator wont be editing the database all the time, we can foresee that the information will always be accessible and concurrency will not be an issue, due to the SQLite3 architecture. However, if we expect multiple processes to write to the database at the same time, it is suggested to consider other databases, more fit to client/server applications.

To be possible to use SQLite3 in our application a Database Class was created with some of the basic operations like information insertion and Content retrieval. This Database included an sqlite3 header file in order to use all the SQLite3 library functions. More information regarding this implementation can be found in appendix A.4.

The current database implementation is designed in a way to provide a movie search with any parameter: movie name , actor, director, genre. However the current application only provides a genre search to prove the concept. The SQL ability of foreign keys helps in this kind of advanced search. The method used to activate foreign keys was to implement triggers. With this triggers if a movie is deleted, then all its references are deleted too.

Furthermore, the database could be used to provide one last feature. For example, in case the service provides viewing packages, where only certain movies can be accessed, the database would be the place were this information should be stored. The associations between users and movies could be used to grant users access only to specific contents or sets of contents. In the future this functionality could prove to be very useful. Separating Music distribution from Video-on-Demand content is one example.

4.1.2 SSL Client

Secure Socket Layer was the chosen method to provide reliable and secure communications between the a Main Interface and each Media Server. The Main Interface is the one responsible for starting the connection and terminating it, by signalling a connection terminate command. Being a TCP connection no information exchanged is lost. The API chosen to deal with information exchange was simple yet very effective.

The possible commands follow:

- LOAD - Load media Session (e.g. "LOAD test.ts")
- CHCK - Check Media Session status (e.g. "CHCK test.ts")
- TERM - Terminate Media Session (e.g. "TERM test.ts")
- GURL - Get Media Session URL (e.g. "GURL test.ts")
- TERC - Terminate Connection
- STAT - Current statistics
- ACCE - Provide a specific user access to that Media Server (e.g. "ACCE username password")

Every response from the server included a string stating the response status and if needed other requested information. For example the possible responses include:

- SMSSUCCESS - Request was successful (e.g "SMSSUCCESS rtsp://localhost/test.ts")
- SMSERROR - Error occurred.
- SMSEXISTS - Media Session exists
- TERM - Connection terminated

The most widespread tool in terms of SSL/TLS connection is OpenSSL which comes with an easy-to-use C API to implement this encrypted communications. Basically SSL connections use a public key derived from a private key, needing both of them to encrypt a peer-to-peer connection. In our application the Main interface has the Media Server's public key. It uses that key to encrypt a message and sends it to the Media Server. The Media Server then decrypts the message with its private key.

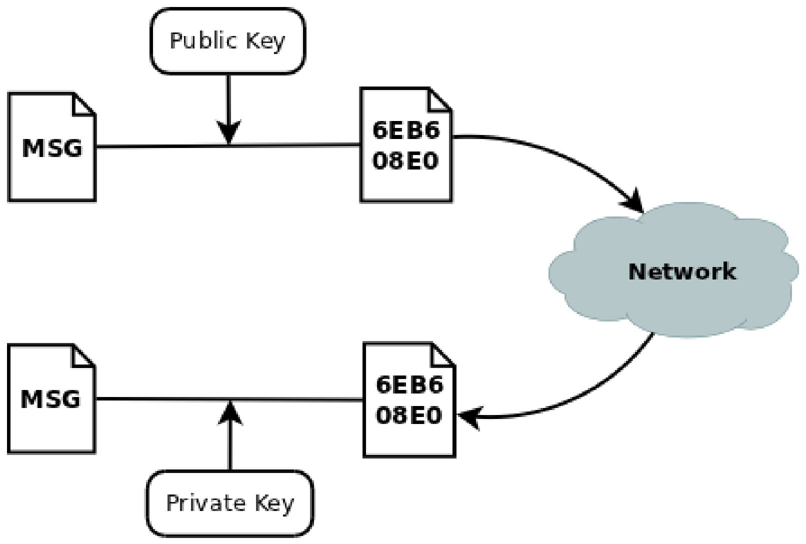


Figure 4.4: Encryption

The public key is normally given to anyone that accesses the service (e.g. a user visiting a HTML web page using HTTPS). In the internet domain the private key is used to generate a certificate (public key) and afterwords its sent to a certificate authority (i.e. VeriSign) for approval. However, for this application only a self-signed certificate was created for academic purposes.

These keys were created with OpenSSL tools. Usually a certificate as an expiration date, and after that it turns obsolete. As these communications are done only between the Main Interface and a Media Server, one can easily control this keys and make them both for private use only, and even create several for each existing Media Server. If any side is compromised in a security breach, the administrator can easily switch one or all the keys provided.

4.1.3 Web Services

Before giving any details about the framework used in this WS exposure, other tools must be mentioned first. The tool used in this application, WSF/CPP, is built on top of other WS C frameworks. Those are the Axis2/C, Rampart/C and others. The first two are the most important to acknowledge because they help understand how the WS were implemented.

Axis2 is a Web Services framework developed by the Apache Foundation, also responsible for the Apache HTTP web server. This application provides a wide range of tools to deal with

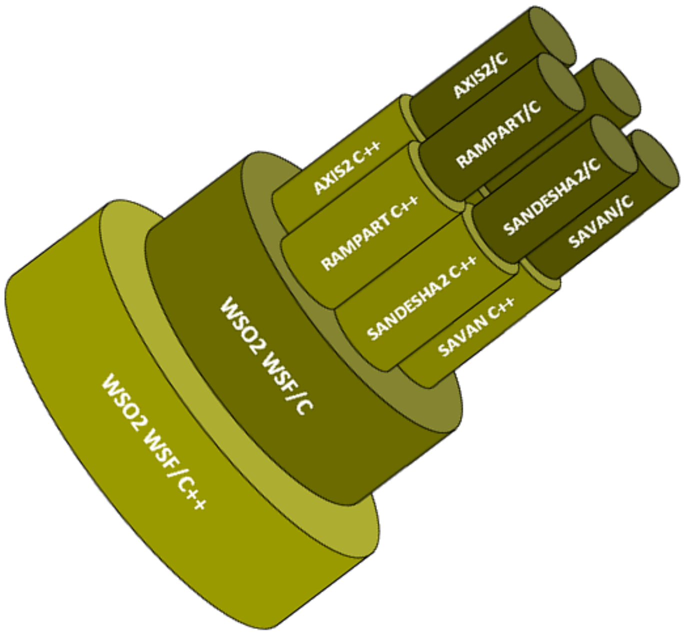


Figure 4.5: WSF/C++

the SOAP protocol, both for server and client applications. It provides integrated support for WS-Addressing and WS-SecurityPolicy. Axis2 uses other existing tools like AXIOM, a framework capable of parsing XML and a SOAPBuilder API. It benefits from many features being the most important one the integration with the Apache HTTP web server. This WS frameworks come in different languages like C and Java. A C++ version exists but since 2009 it has not seen any recent developments.

The Axis2 tool comes with a WSDL to C script in order to create stubs and file skeletons for a fast service deployment. A developer using a WSDL file from any of our services will only need to implement the business logic and nothing else. As our service was going to be developed from scratch the development was addressed first, and only after that we created the WSDL file with the service's description. This file as explained before in section 2.4.1 turned out to be very useful when deploying the administration's interface.

In a server application the Axis tool uses a simple XML file to state the possible operations a service has, and to direct them to the right implemented class. Attached to this file can also be the Security Policies assertions, concerning each operation or the hole service. Our Client's service example follows:

```
<service name="client">
        <parameter name="ServiceClass" locked="xsd:false">client</parameter>
        <description>
                Client API including all client methods to access to the web services.
        </description>
        <operation name="createClient">
                <messageReceiver class="wsf_cpp_msg_recv"/>
        </operation>
```

```
</service>
```

The framework used, WSF/CPP, is built on top of Axis meaning that it can provide all the features mentioned above. For example the Axis logging tools were used to leverage some time in the WS development. And the most useful one is the ability of Axis to expose each service by simply adding two lines in the services.xml file. Through this file the SOAP implementation can be managed as REST services as well, without any modifications at all. The following example shows our User's service getContentList operation, exposed as a REST service.

```
<service name="user">
  <parameter name="ServiceClass" locked="xsd:false">user</parameter>
  <description>
          This is the user Methods available in the Web Services (REST Mode)
  </description>

  <operation name="getContentList">
           <messageReceiver class="wsf_cpp_msg_recv"/>
    <parameter name="RESTMethod">GET</parameter>
           <parameter name="RESTLocation">getContentList/{search}</parameter>
  </operation>
  .
  .
  .
</service>
```

To deal with WS-Security, WS-SecureConversation, WS-Trust and SAML protocols, the Rampart framework is provided. Figure 4.6 shows the Rampart architecture. This tool is responsible for the SOAP Security implementation where there can be more than 24 examples of security options (i.e. according to Rampart samples). From a simple timestamp to a signed-encrypted SOAP message, and even SAML implementation.

Rampart/C provides callback modules to implement other features, being those defined in the policy assertions. An Authentication Provider Module (i.e. similar to the callback module) was used to deal with the user's authentication process in a more application manner.

```
     <rampc:RampartConfig xmlns:rampc="http://ws.apache.org/rampart/c/policy">
       <rampc:PasswordType>Digest</rampc:PasswordType>
       <rampc:AuthnModuleName>
          /etc/wso2/wsfcpp/services/admin/libauthn.so
          </rampc:AuthnModuleName>
     </rampc:RampartConfig>
```

For testing purposes, a simple UsernameToken with signature was used in every SOAP implementation to demonstrate the security capabilities. However, it is not possible to use the UsernameToken policy without signature, because Rampart/C implementation requires by default a signature to enhance security. During development some unsupported SOAP specifications were also found. The SOAP layout policy is always strict meaning that the SOAP header sequence has

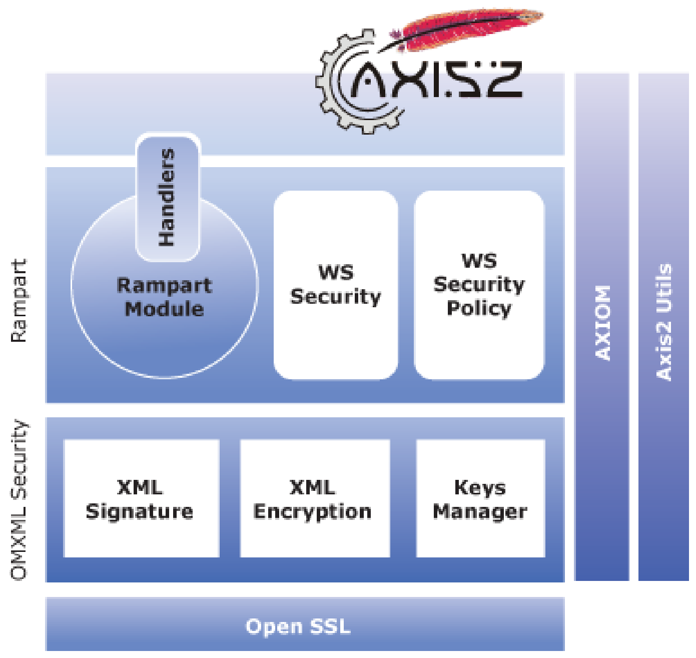


Figure 4.6: Rampart Architecture

to follow the rampart implementation order. This lead to a series of problems connecting both Rampart/C and Rampart/Java in the Client Interface 4.4.

The UsernameToken is formed as described below:

```
    digest = Base64_encode(SHA−1(nonce+timecreated+password))
```

This is just a SHA-1 digest using the password, a set of random bytes(nonce) and a *timecreated* value depending on the creation time. The client sends this digest with the *timecreated* and the *nonce* in order for the server to do the same encoded digest and compare both results. This is done in the Authentication Provider callback module and returns the result with a boolean.

The signature provides integrity to any message or token, to prove that the message was not altered during exchange and it belongs to its producer. The signature in this case comes associated with a UsernameToken, but other possible options may be used. For example, a full SOAP message can be signed first and then encrypted. This order can be changed. Each user will know what to do according to the server's security policy, present in the respective WSDL file.

The example below states that an AsymmetricBinding policy is used with asymmetric keys (public/private key combinations). This encryption or signing is similar to the ones used in SSL/TLS. In this case the service specifies which input/output message will have a signature; the certificate it will use for that matter, X.509V3; and which parts of the message will be signed. In this case it states that the UsernameToken will be the signed part.

```
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
  <wsp:ExactlyOne>
    <wsp:All>
```

```
      <sp:AsymmetricBinding  xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
          <sp:InitiatorToken>
            <wsp:Policy>
                <sp:X509Token  sp:IncludeToken="http://.../IncludeToken/AlwaysToRecipient">
                <wsp:Policy>
                  <sp:WssX509V3Token10/>
                </wsp:Policy>
                </sp:X509Token>
            </wsp:Policy>
          </sp:InitiatorToken>
          <sp:RecipientToken>
            <wsp:Policy>
                <sp:X509Token  sp:IncludeToken="http://.../IncludeToken/Never">
                <wsp:Policy>
                  <sp:WssX509V3Token10/>
                </wsp:Policy>
              </sp:X509Token>
            </wsp:Policy>
          </sp:RecipientToken>
          <sp:Layout>
            <wsp:Policy>
                <sp:Strict/>
            </wsp:Policy>
          </sp:Layout>
          <sp:AlgorithmSuite>
            <wsp:Policy>
                <sp:Basic256Rsa15/>
            </wsp:Policy>
          </sp:AlgorithmSuite>
          </wsp:Policy>
      </sp:AsymmetricBinding>
      <sp:Wss10  xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
          <sp:MustSupportRefKeyIdentifier/>
          <sp:MustSupportRefEmbeddedToken/>
          <sp:MustSupportRefIssuerSerial/>
          </wsp:Policy>
      </sp:Wss10>
      <sp:SignedSupportingTokens  xmlns:sp="http://.../ws/2005/07/securitypolicy">
        <wsp:Policy>
            <sp:UsernameToken  sp:IncludeToken="http://.../IncludeToken/Always"/>
        </wsp:Policy>
      </sp:SignedSupportingTokens>
      <rampc:RampartConfig  xmlns:rampc="http://ws.apache.org/rampart/c/policy">
          <rampc:PasswordType>Digest</rampc:PasswordType>
        <rampc:AuthnModuleName>
          /etc/wso2/wsfcpp/services/admin/libauthn.so
        </rampc:AuthnModuleName>
      </rampc:RampartConfig>
  </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

SOAP has a vast amount of options in its specifications, and the examples we present here only show a very small percentage. For more information we suggest a review of the profiles involved in this situation [28, 29] briefly mentioned in chapter 2 in section 2.4.2.1.

The last two frameworks that support WSF/CPP structure are Sandesha/C and Savan/C. These tools both provide capabilities to apply respectively WS-RM and WS-Eventing. All in all, it would be possible to provide all the 8 presented SOAP specifications in our application.

Although, when using the REST mode operation the SOAP's WS-Security cannot obviously be used, because both are implemented in different ways. In this case Security Policies are obsolete, but the Apache HTTP web server can provide a solution using a transport security layer. By using the Apache web server we can activate the SSL module to provide encrypted communications. As well as our Main Interface and Media Server communication, Apache also uses OpenSSL framework to accomplish HTTPS capability.

Besides that, enterprises nowadays use an API key method to make sure the user accessing the service is a trusted user. Facebook uses a OAuth authorization model which is starting to gain supporters among different services. Implementing this kind of feature is easy. For example a developer signs for an OAuth API Key and Secret. When the developer wants to interact with the service he first signs in with the respective information, receiving in return a user-authorized token to use while accessing the application data. This method can be used together with SSL to improve security in our application. For reasons concerning the development progress schedule we were not able to properly implement this situation.

According to a Netcraft survey [49] Apache server is at the time of this writing the most used web server when it comes to WWW deployment, having almost 70% of market share among the busiest sites. This means the Apache Web Server is very successful. Some examples show the benefits this OSS can deliver to high demand enterprise services, competing and sometimes displacing other existing commercial options. In 2002 the Ireland's National Research and Education Network chose to switch their mirror service to the Apache Engine. During 2005 3.5 million downloads were made each day from their service, and 90% of those represented HTTP traffic. In a study [50] they provide a in depth look on how to properly tune Apache Web Server in ways to achieve high scalability when having several arriving requests at the time. Obviously access and scalability issues will depend solely on the number of arrivals (web service requests) our server will have. One can tune the Apache server to process more concurrent processes/threads or to process determined number of processes with finite number of threads first, and then address the arrival queue.

In our case, the web services have a stateful sequence were one request might depend of another one. For example, before retrieving a movie content or playing, first we will need to retrieve the full content's list. However, after choosing to play we don't need that open socket any more, leaving us an idle process/thread. There were at least 3 requests involved in this situation. For that matter, one might use options like KeepAlive, MaxKeepAliveRequests and KeepAliveTimeout to better deal with this kinds of situations, and leverage the queue line. Besides these options, Apache comprises many others to deal with other aspects.

Though we cannot forget that each response time will also depend on the database and Media Server's time response. As explained before, the database wont pose as a performance threat because it is expected to work smoothly, in a read-only fashion. Yet when the requests depend on the Media Server's response, only its CPU load, and internet speed connection can dictate the possible output.

After all the services were developed, and after some tests using implemented C clients, a WSDL file was created for each service to properly describe each one. Using this kind of implementation, authentication was much easier to implement, being the Admin services only available to the administrator, and the user services available to any registered user, including the admin. We could as well apply different security policies for each service like encryption in the administrator services or SAML tokens. An example of our User WSDL file is present in the appendix A.9.1.

4.2 Media Server

In this section we present the chosen tool for the streaming server, and some of the implementations that were needed in order to meet our requisites.

Only a few tools exist that are fully developed in the most used video codecs and streaming solutions available nowadays. Darwin Streaming Server is a complex system with over than 14Mb of occupied space, meaning it brings to much functionalities that are not needed. Among its features are the embedded web interface. The reasons for not choosing DSS were simple, this tool is not updated since 2008 and our administration interface needed other kind of implementation, especially a more scalable one.

Live555 was the selected tool to use in the streaming server for several reasons. First this was a very small framework, easy to expand, and the best for someone who's starting from the beginning. VLC implements Live555 in the RTSP client interface using its Media Library to demux the incoming RTP/TS packets. Besides, stable versions are released with some regularity and the development list has feedback everyday. The last update dates back to 16/06 of the current year.

As explained before in section 4.1.2, in order to provide full scalability in a wide spread and unreliable environment, a SSL Server was implemented with the same OpenSSL tool. For statistical feedback one more tool was needed to retrieve the media server's system information. Libstatgrab was the chosen tool to provided cross-platform access to system statistics.

An overview of the Media Server interface can be observed in figure 4.7

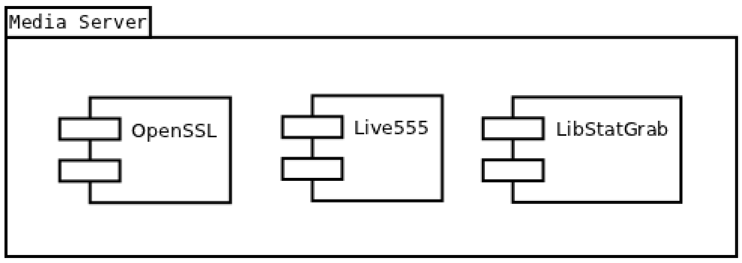


Figure 4.7: Media Server Components

4.2.1 Live555

Live555 provides all the needed tools to implement a capable RTSP server. A RTSPServer class exists to create a proper on-demand server with any file format possible. A Media Server solution already exists in the official source code, a DynamicRTSPServer, extending the original class but it does not meet our basic needs. So, in order to provide a more customized server the RTSPServer class was used. The basic operation of the Live555 is simple, the program runs a one-threaded process in an environment loop, using event-driven methods to operate. When a client connects to the media server a unicast connection is initiated. The processing sequence of a movie, in our case a TS file, that will be transmitted is the following:

```
'source.ts' -> 'RTPsink'
```

The program extracts the information from a socket (file descriptor), in this case the source.ts, and transmits it to a RTPsink. Note that this is only happening inside the application. Afterwords, the RTPsink as a side effect transmits the RTP packets.

In terms of the RTSP server the application had a Media Session with several sub Media Sessions as explained in section 3.3.3.2. RTSPServer provides functions like add, remove and lookup media sessions. In order to add a video file to the Media Session, a file name must be provided by the admin. If the server finds a video file with that specific name it proceeds to the next step.

One important aspect of the media server when loading a media session was to check if the video file was accompanied with an index file. This functionality was a basic requisite specified in section 3.2.1. Because indexing a file takes a big chunk of the computers load capacity, it is advised to use a fast computer when doing it. The current implementation used a second thread in order to set aside a second environment loop to index the file. A simple POSIX thread was used with a Join command, to wait for the indexation to finish and return the result.

After checking for an index file and doing the necessary work the media session is loaded with the video stream capable of trick play functionality. Only then we are ready to view the streaming in any RTSP capable device by accessing its URL:

```
rtsp://mediaservername.com/filename.ts
```

Examining the official website it states that Live555 is capable of streaming to normal video players and also to Amino set-top boxes for example, for playing TS streams only. This means our application is fully capable of streaming both to STBs and to Desktop clients through the internet. We used an Entone Kamai set-top box to test the capabilities of the media server, and the tests were successful. We were able to properly stream the video, play, pause and fast forward the stream, demonstrating that Live555 has a lot of potential. This tests can be done using a provided test tool called openRTSP, where we can demonstrate all the trick play functionalities. This program saves the received content to a file confirming the expected results.

Set-top boxes do not need RTP packets because TS packets already provide the necessary tools to demultiplex the stream, and usually request a raw-UDP transmission. That was exactly what happened in this case. The reason why this happens is because most STBs are designed to work only with TS stream files, and for obvious reasons don't need the extra RTP overhead.

In IPTV environments there is more than video-on-demand in the options toolkit. Normally IPTV provides music, live TV broadcast and other kinds of contents like games. Live555 may one day be developed for gaming experiences but is fully capable of delivering other formats. The official website provides also tools to create music multicast sessions, and there are product examples where Live555 is used for broadcasting live feeds.

The last used feature that comes embedded in the Live555 library is an authentication process, which might be used to increase streaming security. Live555 comes with an authentication database to store all the clients necessary information, username and password, for authentication purposes. This feature was fully implemented and can be easily controlled through the Main Interface, where the admin can provide users access to any specific Media Server. On the other hand, this system can be used to provided for example, a special security feature where the access URL has a special authentication procedure that changes every few minutes. When a registered client accesses the movie's URL it would include already this special access information.

However, there's a glitch in this process. Providing authentication access to a Media Server, any registered user will be able to access any Media Sessions available, after guessing their URL stream. This should not happen in cases where this user only has access to certain contents. In this case, Live555 would need improvements like providing access only to the respective content, which seems pointless. Other method would be to use the Main Interface's database to acknowledge Users/Movies association in order to provide access. Additionally, we could change the way that the Media Server creates the specific video URL, by changing the stream name in the Media Server to something else like:

$$rtsp://mediaservername.com/newdiffstreamname$$

For this reasons Live555 proved to be the best possible choice when it came to implementation and development. The Video-on-Demand experience was the only one used in this application, with all trick play functions available. Yet, others could easily coexist like music and live TV broadcast.

4.2.2 SSL Server

As explained before, SSL communications were used to provide secure communications, however, this type of communication also proved to be valuable in scalable situations. By providing an integrated SSL server any Media Server could be reached from anywhere in the world with constant monitoring and control through the Main interface. With this method, the Main Interface could be responsible to accommodate routing capabilities, redirecting users to any server according to load or bandwidth considerations. With this kind of implementation many content delivery issues could be easily covered.

For development reasons the current Live555 source was not modified at all in order to handle the administration's requests. By doing this we could assure if needed, an easy update to the current library in case a new version was released.

Live555 source code comes with event triggers that could be applied in this case, where an external agent could stop the loop environment to issue commands in the current Media Session environment. This was not the chosen method however. The used method implied that both the RTSP and the SSL server would run in their own thread, in a single process. With this technique any information could be easily exchange between both parties. Aside from that, several main interfaces could coexist. Nevertheless, in case different interfaces access the server, race conditions should be prevented, which is not the case, as the dimensioning only covered one Administration agent. A simple Mutex situation would suffice as POSIX threads were used to initialize the respective threads.

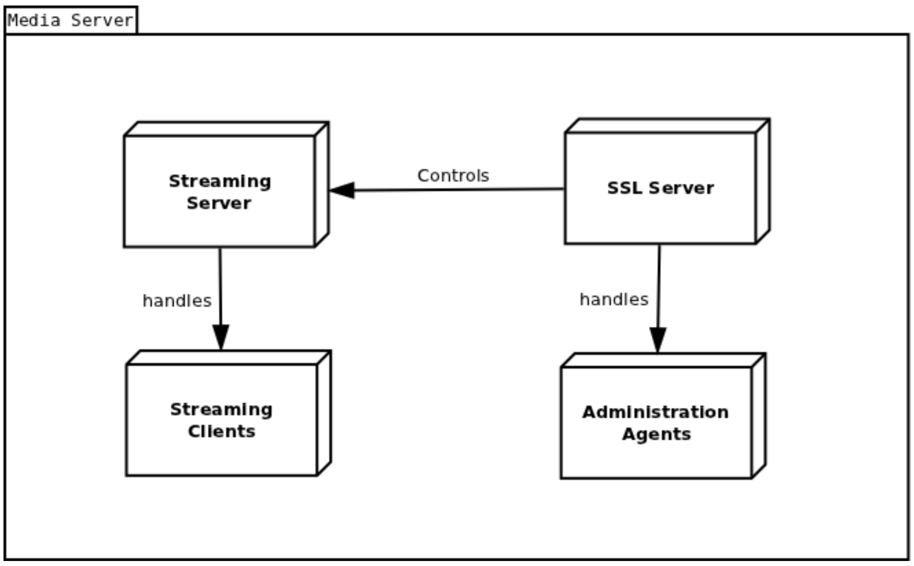


Figure 4.8: Media Server Architecture

The only missing aspect of this Media Server implementation was the system's statistical feedback. Usually this is done by using a specific protocol, like Simple Network Management Protocol (SNMP), to control and receive information from the Media Server. Though, this protocol would require both agents to implement the SNMP protocol. A third server in the Media Interface and a client in the Main interface would be needed just to monitor the Media Server's progress. Be that as it may, there was a more elegant solution that could be applied in this case. LibStatGrab is a framework that can retrieve any statistical information we would need. CPU information, Memory and Bandwidth usage and much more. Retrieving this information and sending it back to the Main Interface when requested, was a practical option compared to other possible choices.

4.3 Web Administration

The web administration interface was one of the basic requisites of this thesis. The main objective was to provide a portable and cross-platform interface, and achieve full control over the entire VOD system.

Using a HTML web interface we would provide an interface able of being used in any browser-able OS. Although the choice was to use a web interface with the web services exposure, this interface could be built in any language like C/C++, Java and others and be ported to another computer rather than the main interface's station.

In light of this choices two things were needed to start development. One was the PHP framework to work with the SOAP protocol and WS-Security for signed UsernameToken support. The only complete framework able to deal with these requests was a tool also provided by the WSO2 group, the WSF/PHP. This tool was the only current PHP framework capable of dealing with WS-Security, WS-SecurePolicy and WS-SecureConversation. The others were HTML tools able to provide an appealing web interface. In this chapter HTML5 and CSS3 were the latest available options to design our Administration web page.

To retrieve all the movie's information another PHP framework was used to scrap any IMDB information about any desired movie. Only after retrieving and serializing the information we wanted we were able to send it to the Main Interface's database. The chosen tool was IMDBPHP, always up to date with IMDB official website and very easy to deal with, including a test engine to assure that all the required IMDB functions are working properly.

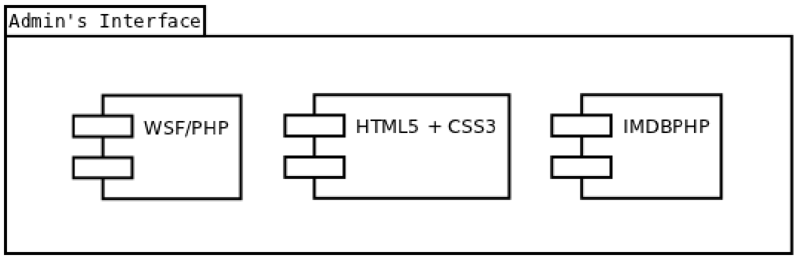


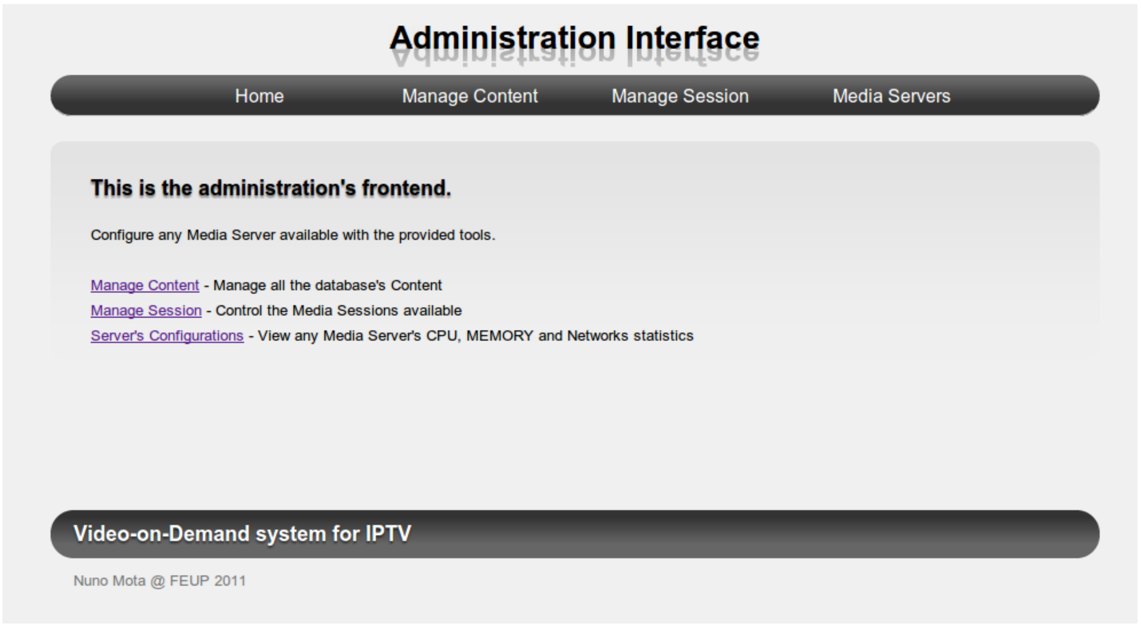
Figure 4.9: Administration Interface Components

4.3.1 Administration Design

Nowadays, powerful enterprise web pages are designed with the help of several HTML tools. For example, HTML5, CSS3, Javascript, PHP, JQuery. All these tools provide different kinds of options when building dynamic web interfaces.

Nonetheless, many people argue that HTML5 is nothing more that HTML language allied with other set of tools, like CSS3 in order to provide all these amazing graphic options. To keep things simple, those were the only tools used to design this web interface. In the end, PHP was needed to process all the required web services requests, in order to retrieve any needed information and to access the Main Interface's controls.

As explained in section 3.3.2 a simple division was chosen to create this interface. A section to manage the Media Server's list was created. When initializing this interface and before any management procedure, the administrator should retrieve from the Main Interface the existing server's list from the Main Interface. Only then he could proceed with management options. In figure 4.10 we can see the Administration's Interface. An upper menu bar was created with dropdown menus to access the desired option.



This is the administration's frontend.

Configure any Media Server available with the provided tools.

Manage Content - Manage all the database's Content
Manage Session - Control the Media Sessions available
Server's Configurations - View any Media Server's CPU, MEMORY and Networks statistics

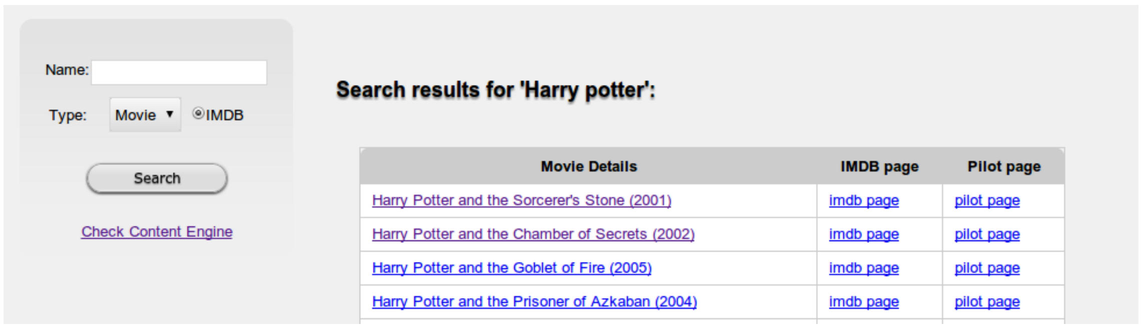
Video-on-Demand system for IPTV

Nuno Mota @ FEUP 2011

Figure 4.10: Administration Interface

Another tool able to retrieve the contents information and add it to our database was a framework to reach IMDB content. IMDBPHP framework was very useful because it brought search capabilities, and a content scraping API. Officially IMDB website does not provide any open APIs for others to retrieve its content into their own applications. Over the time, IMDB changes their implemented functions, and as a consequence the IMDBPHP needs to be changed. A testing engine was also available to check if every IMDB option was accessible. After integrating this tool in our interface, and properly serializing the necessary content we were able to populate our Main Interface database with the proper content. The IMDB framework turned out to be the most easy-to-use tool found during this thesis. In figure 4.11 we can see an example of the search results.

To access any Media Server and upload the video file an open source FTP Java applet was integrated in this interface. Using an applet the administrator is able to access any Media Server and use the FTP protocol or the SFTP protocol to upload any content to the Media Server's disk. Obviously, each Media Server needs to have a FTP server up and running for this to be possible. Though this was made to meet the requisites specification, the media storage has several implications, mainly economical costs. Nonetheless, this was not a pressing issue during this thesis and so the development in details of media storage were not further extended.


Name:
Type: Movie ▼ ◉IMDB
Search
Check Content Engine

Search results for 'Harry potter':

Movie Details	IMDB page	Pilot page
Harry Potter and the Sorcerer's Stone (2001)	imdb page	pilot page
Harry Potter and the Chamber of Secrets (2002)	imdb page	pilot page
Harry Potter and the Goblet of Fire (2005)	imdb page	pilot page
Harry Potter and the Prisoner of Azkaban (2004)	imdb page	pilot page

Figure 4.11: Search Results

4.3.2 Web Services

The web services was the most important aspect in developing the Administration interface. Without the right tools to access the Main Interface through the SOAP protocol it would not be possible to create this application. The chosen tool to help us in this implementation was WSF/PHP. This tool was developed by the same group that created the WSF/CPP, and for obvious reasons was also built on top of the previous tools, Axis2/C and Rampart/C. By making use of the same frameworks full compatibility could be achieve between applications. In this case between our Main Interface and the Administration Interface.

But WSF/PHP is much more advanced than WSF/CPP. The tool was used only as a PHP SOAP Client but REST mode and Server modes are also possible to achieve. A similar WSDL to PHP script exists to provide fast service deployment. As mentioned before in section 4.1.3, as we had now WSDL files available, we made the most out of them. By using the WSDL files we were able to rapidly create the necessary objects to use our services (see fig. 4.12). This tool created all the variables and objects class mappings necessary to use the Main Interface's web services.

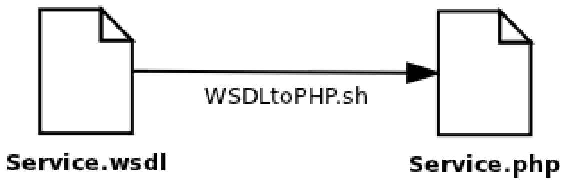


Figure 4.12: WSDL to PHP

However, the best part is yet to come. WSF/PHP has one more feature that no other PHP framework, neither the WSF/CPP, could provide. This feature is called the WSDL mode. Besides creating all the necessary structure to accommodate the services procedures the WSDL mode was available to make use of the WSDL service file, in order to connect all the variables and objects with the right messages and respective operations. The WSDL mode was available as an option in the WSF/PHP client using the following:

```
$client = new WSClient(array("wsdl" => "php/AdminService.wsdl",
                    "classmap" => $class_map,
                    "useWSA" => TRUE,
                    "policy" => $policy,
                    "securityToken" => $security_token));
```

To fully understand this option the reader needs to understand that the PHP source code did not have any information regarding how to connect all the service input variables with the respective messages and operations, neither how to deal with the service's response. In order to achieve this the framework would take all the needed information and procedures from the WSDL file. Using the WSDL file the PHP interface would know to which operations corresponded received input and output objects.

The basic WSDL Mode operation is explained in figure 4.13. With this capabilities there was no need for any more redundant information besides the one already contained in the WSDL file. Input SOAP objects received from the Main Interface were also easily handled by the WSDL mode. The PHP tool would receive the respective SOAP message, according to an operation, and serialize all the information in order to return the correct object with the received content.

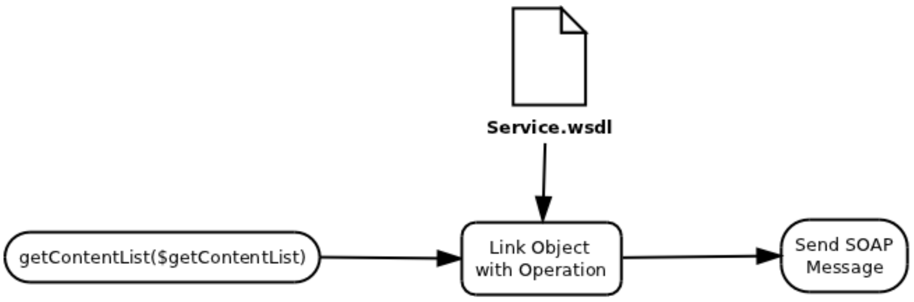


Figure 4.13: WSDL Mode procedure

Because WSF/PHP was built on top of Rampart/C, security proceedings were not an issue. This framework API provided the necessary tools to implement our desired Security Policy, in this case the signed UsernameToken. As shown before in the WSDL mode source code 4.3.2, two more options are assured in the WS Client operation. One is the policy chosen in the service, and the other is the required information to construct the UsernameToken.

```
$security_options = array("useUsernameToken" => TRUE );

$policy = new WSPolicy(array("security" => $security_options));

$security_token = new WSSecurityToken(array("user" => "admin",
                                   "password" => "blablabla",
                                   "passwordType" => "Digest",
                                   "privateKey" => $my_key,
                                   "certificate" => $my_cert));
```

4.4 User Interface

After implementing a Main Interface, connecting a Media Server and manage everything through a web interface, only one thing was left to do. Without an integrated solution to show how a user-end application would work or how it should look like, everything done so far would seem rather useless.

This chapter was divided into two parts. First, our goal was to create a RIA application. Rich Internet Application is a term used nowadays for rich internet multimedia experiences. And second, integrate all the features our VOD system provided. We needed to collect all the information of the available movies, and watch any movie using any of the trick play options.

Java introduced in 2008 a new set of visual APIs to compete with Flash. The choice fell on this tool because GUI results seemed very promising. However, at the time of writing this thesis the current stable version was 1.3.1, built on top of JavaFX script. This meant that in the future, in terms of support, everything would have to be ported to Java language. Yet the API is the same, which means much of that work would be simplified.

However, JavaFX media library lacked compatibility with the current video format that we used in our Media Server. As we wanted the hole experience to be in a single application, JavaFX was not an option. The solution was to implement the video player using Java multimedia APIs, capable of interacting with the JavaFX implementation. Both GStreamer and VLC Java bindings were tested in this situation. These tools used the official C source code which was well documented and perfectly stable. The reader needs to bear in mind that the trick play functionality (play/pause, fast forward, etc) was mandatory, and the mentioned libraries were the only possible way to achieve such goal.

The overview of the User Interface is described in figure 4.14

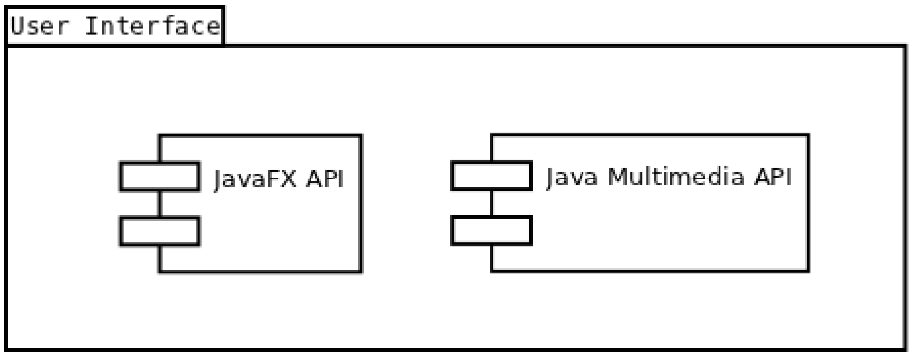


Figure 4.14: User Interface Components

4.4.1 Web Services

Before we could start designing the User GUI we should first figure out a way to retrieve all the needed information from the service. This meant that we needed to access the Main Interface's web services, in order to collect the Content's list and so on. As SOAP was the chosen protocol to access the application web services, a SOAP implementation was needed.

In order to access the WS interface the User's interface had to support somehow the WS-Security implementation. JavaFX lacked the proper tools to fully implement the SOAP protocol. Nevertheless, Axis platform also provided a Java interface to use the SOAP protocol with WS-Security, supported by the Rampart/Java platform. Afterwords we would need to connect the Java WS SOAP access to the JavaFX interface.

We started by testing a simple client's implementation in order to reach our Main Interface web services. A simple SOAP implementation without using Rampart platform was possible to achieve. But some problems occurred when testing the security platform that Rampart/Java provided. When trying to connect using a signed UsernameToken situation, our Main Interface for some reason stopped working. When this happened the client could not retrieve any information at all, waiting for the Main Interface transaction to complete. Tracing back the Main Interface logs the information was inconclusive. Our service was able to properly authenticate the signature and the user's authentication, but stopped after doing so.

Using a network logging tool (i.e. Wireshark) we were able to understand what the problem was. Although Rampart/C and Rampart/Java share similarities because both belong to the Apache foundation, they are developed in different ways. The reason for this to happen is that the teams responsible for each platform differ. This resulted in SOAP security sequence verification and creation differences in both applications, culminating in incompatibilities when joining them together with the necessary security policies. The Layout SOAP specification could help in this situation, but changing the Layout mode to Lax (relaxed) did not solve anything. It turned out that the Strict mode was the only one available in Rampart/C.

After understanding the problem we had two possible solutions to overcome this issue. One was to change the Rampart/C source code to accommodate the Java security sequence, and the other was to change the used WS protocol. By looking into the Rampart source code we could easily understand that the necessary changes would be no easy trick. To fully understand the source code structure we would need to spend some time we could not afford to loose. Just as a reminder, the JavaFX GUI design and the media player were at this time not developed yet.

As for the second option, the solution was already available in the Apache Axis/C framework and easily supported by the JavaFX API. Over the last years, REST as been evolving rapidly because of its architecture and simplicity. As explained in section 4.1.3, Axis/C could easily support REST operation. By using this functionality we could now use JavaFX to implement the WS interface as well.

As REST uses HTTP architecture one could use the JavaFX HTTP API to access our web services. The class used was HttpRequest which provided a simple way to use the HTTP Get in any specified URL. The operation was simple, whenever there was the need to retrieve any content from the Main Interface a search option would be issued. In this search function when the input arrived a parser would be issued in order to parse correctly each content. By using an abstract search class we could then provide the proper XML parsing in another class, according to each operation. An overview of each class can be viewed in figure 4.15.

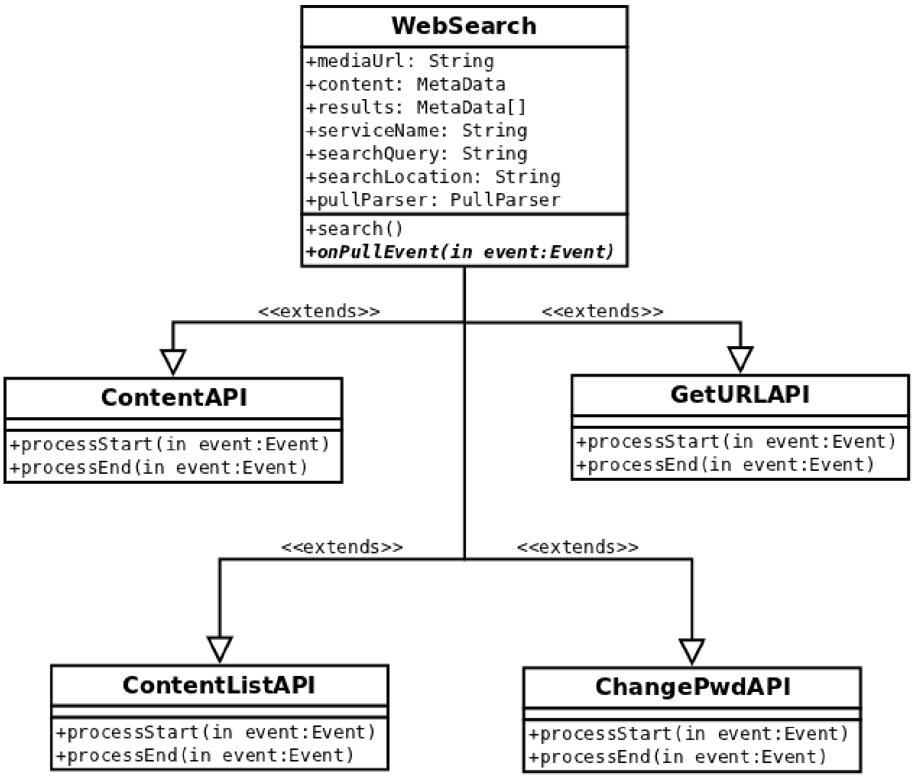

WebSearch
+mediaUrl: String
+content: MetaData
+results: MetaData[]
+serviceName: String
+searchQuery: String
+searchLocation: String
+pullParser: PullParser
+search()
+onPullEvent(in event:Event)
<<extends>>
<<extends>>
ContentAPI
+processStart(in event:Event)
+processEnd(in event:Event)
GetURLAPI
+processStart(in event:Event)
+processEnd(in event:Event)
<<extends>>
<<extends>>
ContentListAPI
+processStart(in event:Event)
+processEnd(in event:Event)
ChangePwdAPI
+processStart(in event:Event)
+processEnd(in event:Event)

Figure 4.15: Web Search architecture

Now that the web services were implemented using for that matter JavaFX tools, we could easily integrate the contents inside a GUI.

4.4.2 JavaFX

JavaFX was a very intuitive tool when designing the user interface. With just a few lines we could easily provide an appealing visual application. Besides, many interactions could be achieved using pre-defined JavaFX effects and timeline options, designing smooth transitions between elements. Also the new JavaFX binding properties were very helpful providing fast visual changes in the design layout.

The design was divided into three sections: the home menu, the Content's list wall, and the Content view. All this sections were presented in one single scene, displaying each one according to the user's inputs. In the main menu there were several options like watching movies, television or choose to listen to music. Obviously, only the Movie option worked in this case.

After choosing the Movie option, we retrieved from the web service all the Content's list, and display all the movies in a wall using their thumbnails. By choosing a thumbnail the movie picture would zoom in for a proper view. By clicking a second time, the movie content would appear in another scene. In this scene all the movie's information would be shown, like rating, actors and so on. Only then a play option would appear.

To connect both JavaFX and Java sources, a few extra tools were used. In order to provide the use of Java source code among JavaFX implementations one must use some JavaFX tools. In the future however this tools are deprecated as JavaFX will be integrated in Java. Essentially two things were needed, a special JavaFX class to initiate a background task, in this case our media player. Normally, JavaFX ran in single threaded application using this technique for computing algorithms in a background thread. This way the interface would not be unresponsive. Using the JavaTaskBase we could create a runnable Java environment.

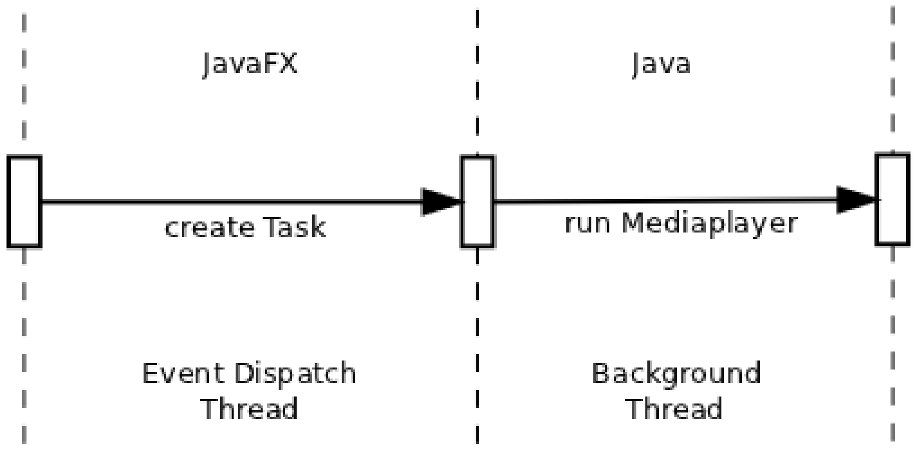


Figure 4.16: JavaFX <-> Java

4.4.3 GStreamer-java

GStreamer was one of the possible choices to develop our video player. Using any Media Player developed with GStreamer libraries (e.g. Totem Player) we were fully capable of testing our Media Server's video stream. However when it comes to trick play functionalities only two were possible to implement with the current stable version of GStreamer.

Playing the movie was possible, pausing provided no effort whatsoever with the Totem Player, but problems started to happen when the seek option was used. The image would simply freeze and return only a few minutes later, if it returned at all. Sometimes, media players may be compiled using old libraries or use unsupported API functions, already deprecated. Just to be sure this was not the case we developed a video player in Java just for testing purposes. In the end, our Java player demonstrated the same results as the Totem Player did, freezing whenever the seek operation was issued.

After some backtraces we discovered that all the seek commands were being issued properly, and consequently RTP packets were being received, apparently with no problem. With a debug option activated, some information could be retrieve to understand what was the problem with Gstreamer library. Even without the debug log one logic idea came to mind, that the GStreamer was not able to synchronize correctly the received information. The debug logs confirmed this theory.

The big problem with GStreamer is that the framework is composed of several parts proving to be very difficult for someone without any knowledge of the source code structure, to properly backtrace any error and fix the problem. Without the minimal trick play functionality GStreamer was not an option to continue the development of our application, because the basic usage was not enough to complete this application.

4.4.4 VLCJ

As explained before in section 2.5.2 VLC uses Live555 Media Library to implement its RTSP client, which means VLC was the best option to make a compatible video player. The stable version of LibVLC used for development was 1.1.9.

The experience with the official VLC player was very successful, providing us all the capabilities the VLC player had available. Play, Pause, Seek the content, and Fast forward. This meant that it was possible to build a VLC Java based Media Player, fully compliant with our Media Server. For demonstration purposes the Video Player was kept very simple with all the basic controls that a normal DVD player provides.

Still, one control was missing. In the original VLC player no fast-rewind option is available. When fast-forwarding the stream the VLC player demultiplexes the incoming RTP packets, and the fast-forward packaging operation is done in the server's end. This means that a similar operation is also expected in the case of fast-rewind. As explained before, these two options are achieved using a Scale variable, that goes from negative values (i.e. rewind) to positive values (i.e. forward).

By using the openRTSP client provided by Live555 Media Library, we could examine that fast-rewind was possible to achieve with our Media Server. The result was the following:

```
Setup "video/MP2T" subsession (client ports 60656-60657)
Created output file: "video-MP2T-1"
Sending request: PLAY rtsp://192.168.1.201:8554/test.ts/ RTSP/1.0
CSeq: 5
User-Agent: ./openRTSP (LIVE555 Streaming Media v2011.03.14)
Session: 3F1EC0B2
Scale: -2.000000
Range: npt=30.000-0.000

Received 212 new bytes of response data.
Received a complete PLAY response:
RTSP/1.0 200 OK
CSeq: 5
Date: Thu, May 26 2011 15:29:34 GMT
Scale: -2.000000
Range: npt=30.000-0.000
Session: 3F1EC0B2
RTP-Info: url=rtsp://192.168.1.201:8554/test.ts/track1;seq=62219;
rtptime=1756668699
```

Being the only missing option, some programming had to be done to provide support for this functionality. The first thing to do was to compile the VLC source code and redirect the Java VLC native library to our compiled source. Afterwords, we needed to find and edit the source code lines to provide such operation. As the result was similar to a fast-forward operation we should first locate that operation. By a quick inspection we uncovered the following.

```
int libvlc_media_player_set_rate( libvlc_media_player_t *p_mi, float rate )
{
    if (rate < 0.)
    {
        libvlc_printerr ("Playing␣backward␣not␣supported");
        return -1;
    }

  ...
}
```

There is a simple explanation for not supporting fast-rewind. When playing any video file from the local disk the fast-rewind is not supported because in theory no video format and codec supports rewind at a reasonable computational cost. Yet, as this computational effort is only done in the Media Server, it brings no effort to the client's end when there is only a need for demuxing. With this in mind, a solution had to be found in order to help us achieve our goal. VLC source code had a variable stating if the stream was able to rewind, but for some reason was not in use in this part of the source code. This variable is named *can-rewind* and according to some changelogs it was introduced in 2008. By retrieving this variable we could use it to provide rewind capabilities to VLC.

```
int libvlc_media_player_set_rate( libvlc_media_player_t *p_mi, float rate )
{

    input_thread_t *p_input_thread = libvlc_get_input_thread ( p_mi );
    bool b_rewindable;

    if( !p_input_thread )
        return 0;
    b_rewindable = var_GetBool( p_input_thread, "can-rewind" );

    if (rate < 0. && !b_rewindable)
    {
        libvlc_printerr ("Playing␣backward␣not␣supported");
        return -1;
    }
  ...
}
```

After recompiling the source code the results were successful. By accepting a negative rate our media player could properly receive the desired content. With a simple cpu usage inspection we could observe that the fast options were actually the ones that required less computer processing necessities.

```
PLAY rtsp://172.16.2.196:8554/Redbull_720.ts/ RTSP/1.0
CSeq: 11
User-Agent: LibVLC/1.1.9 (LIVE555 Streaming Media v2010.11.17)
Session: 68C9705
Scale: -2.000000

RTSP/1.0 200 OK
CSeq: 11
Date: Fri, May 27 2011 14:41:49 GMT
Scale: -2.000000
Session: 68C97056
RTP-Info: url=rtsp://172.16.2.196:8554/Redbull_720.ts/track1;seq=13622;
rtptime=2101514686
```

With this simple solution we were able to properly achieve all the desired requisites in our application, providing full trick play functionality that a VOD application demands. Although, a few problems persisted because the original VLC source code did not properly update the playback time when working in a fast state.

Chapter 5

Experimental Results and Discussion

This section will provide some of the results and consequent conclusions achieved with some experiments using the developed modules. The web services development raised some discussion regarding each used protocol, so in this chapter we will address some opinions in why these protocols are useful and highlight some of the advantages and disadvantages in using each one.

Afterwords, some experiments were conducted with the Live555 framework in order to acknowledge if it was a good tool for future expansion, and if it was able to evolve according to the users needs and content nature.

5.1 Web Services

Web Services deployment has seen an enormous evolution in the past decade. SOAP and REST have shared the internet domain, both with their own arguments, raising many discussions along the way. As explained before, they are both used with the same purpose, but built differently.

WS is definitely the best way to scale our services to others, while downsizing the investment and eliminating the need for a big development team. With the used WS platform we were able to provide both services to our application. In the future, this would please many developers with specific taste for each one. However, the argument over the past few years was that REST would be the replacement of SOAP protocol.

No one can argue that REST is more simple and effective to use than SOAP. With a quick HTTP request we can reach whatever services we want without the need for complicated requests. The XML or JSON response parsing support is widely spread among many development tools. Besides that, REST is totally integrated inside the HTTP architecture, therefore using that to its advantage.

SOAP on the other hand is an enormous beast and sometimes very difficult to tackle. The WS-Stack comprises so many and sometimes unnecessary tools that developers tend to put it aside just by the mention of it. To include all its features there will always be the need for an extra framework, while with REST that does not happen. By our experience this means several other frameworks working together as one. And not every existing framework is actually capable of handling all the SOAP specifications. The major providers that at the time were developing its specifications and profiles, by starting to compete against each other, created a protocol that along the way lost its supporters with its immense structure.

Nonetheless, both have their advantages depending on specific situations. While SOAP may sometimes be confusing with its stack, it provides security means that no other tool provides in the world wide web. SOAP has tools for some enterprise scenarios that no other protocol can deal with, and the advantages are not just in security issues but also in failure situations. For example, WS-RM provides a built-in way to deal with cases where errors occur, while in REST services when a failure happens, the client is supposed to retry its request. When describing each service, the job is facilitated with the use of WSDL which is more developed for the SOAP protocol, although in recent years WSDL as seen some evolution to support the REST specification.

REST provides an easier, yet different alternative to the SOAP implementation. Its simplicity makes it ideal for situations where information retrieval is the only goal, like in our User situation. By using the HTTP protocol, REST can be integrated with no effort in any application, only by using a HTTP API. As the years go by, we see an increase of popularity (see fig. 5.1) and deployment [51] of REST services.

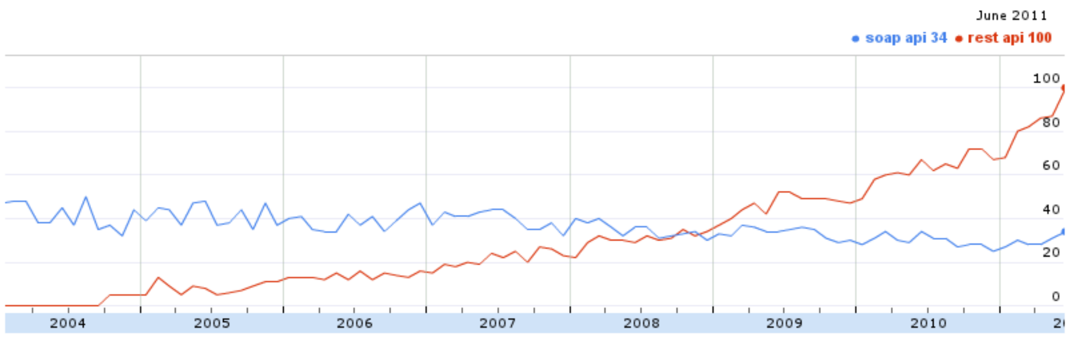


Figure 5.1: SOAP vs REST

Altogether, being able to provide SOAP and REST services in the same application we could answer to any possible need our application might face in the future. This start-up development, which showed us how easy it was to create any application (e.g. Administration/User interface), would definitely provide a big return of investment in years to come.

5.2 Media Server

Because Live555 was the only undocumented tool in terms of real situation deployment, we had to test its full capacity. During this experiments many design approaches and challenges emerged, providing a unique view in dimensioning a real enterprise VOD server.

5.2.1 Service Deployment

VOD requires a single unicast channel for each user to enable a DVD-like experience. This brings many scalability issues, because of the traffic it generates, and the costs of keeping large high definition contents in the local media storage. These are the main concerns of the service providers when achieving large scale CDN deployment in the most efficient way, to provide their clients the best viewing experience. However, we face other series of problems in this case.

As there was no possibility of testing this application in a real live environment, taking into consideration, possible router bottlenecks when routing different networks traffic, we used a private closed network. This eliminated the need of reaching other overlayer networks to provide the streaming service.

Several tests were conducted using a set of computers to imitate the clients stream. The server station in question was a Desktop computer with a Q9300 Core 2 Quad, 4GB of RAM memory and a 500GB SATA II hard disk. To provide an enterprise environment, this computer was connected to a Switch Gigabit interface. Initially, a single media server process was used. First, we started with something small and afterwords raised the bar to achieve the maximum server capacity. The video file used in this experiment was a high quality TS file of 1.3GB in size, with a MPEG2 video, an average bit rate of 10467 kb/s and a total duration of 17 minutes and 43 seconds. If the bit rate occurred at a constant rate during the hole video stream, it meant that a single client would use up to 10,5Mbit/s of bandwidth, making almost a total of 95 possible clients in a gigabit interface. However, one has to remember that the video is codified in a VBR rate, which means higher throughputs can be achieved, in parts where the video has more codified information.

Because many statistical information was retrieved from this experiments, only the ones considered more relevant are shown in this section. Yet, more information is accessible in the appendix C, regarding all the experiences presented in this section. A single experiment was conducted to acknowledge exactly what a single user required during a unicast transmission. Other experiments with more users were done to measure how much the system could actually withstand, and how much network bandwidth was needed to support those users. In some tests, several issues occurred leading to I/O bottlenecks both in the server CPU performance and in the network throughput.

A single user could help us understand what a unicast connection would demand from the server station perspective, and how the throughput would spike during more demanding parts of the movie. In figure 5.2 we can see exactly how the bandwidth varies according to the video file. This values show us that an average of 10.5Mb/s is very misleading because the transmitted bytes can spike up to 3MB/s. However, these are just sporadic spikes and transmission values are always below that line, reaching mostly 1.5MB/s.

The bottom line in the graphic represents the received bytes. This concerns the RTCP and the RTSP protocol meaning that the transmission of control information is insignificant when compared with the video information.

With only a single user accessing our VOD service the CPU did not show any signs of effort

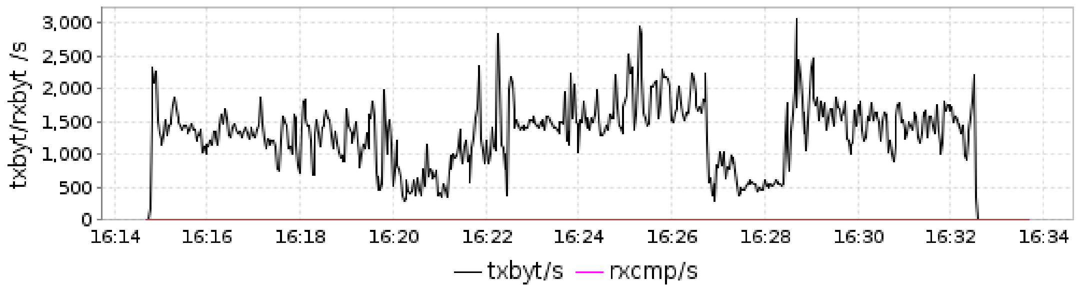


Figure 5.2: Single Unicast Throughput

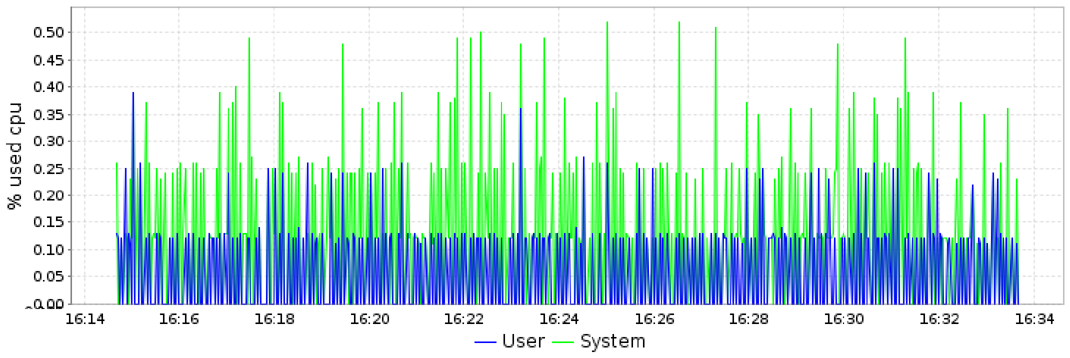


Figure 5.3: CPU Usage - 1 client

with values below 0.5% (see fig. 5.3). The higher values indicate the System usage while the lower ones are referent to the User usage. This represents an average of all the CPU cores, meaning that our thread had to spend less than 2% in processing. A few tests were then conducted, to find out how many clients the system could withhold, and a few things were uncovered. To provide several clients 6 clients per computer were set up with a 100Mbit interface. A script guaranteed that each client connected 5 seconds apart from the previous one. If running one script at the time in each computer it meant that the first client could have around 35 seconds of difference from the last user connected. One problem found was that reaching more than 60 clients the CPU load would reach values of 1, and when that happened the video stream would demonstrate some jitter. Assuming that each core would require less than 1.5% of usage per client, less than 70 users could actually use the system in this situation, occupying 100% of CPU. This meant that the Live555 process being a single threaded application could not scale through other cores. With the process requiring the full load capacity of one single core some of its arriving requests would have to wait in the queue, meaning that most of the information could not reach its client in due time.

Table 5.1 shows the streaming losses of an overloaded situation with 68 clients. In this case we used *tux44* as a server station. We started connecting a couple of clients and after a few minutes we connected the rest. For each user, openRTSP application was used. After all the clients terminated a single video file, representing each used computer, was saved in the system and compared with the original file. As we can see there are some significant losses in a few cases, with 5% of lost content. In a viewing situation this means that the viewer during a certain period of time would experience too much jitter in order to enjoy his video stream properly. According to the time, we can acknowledge that the first few clients lost some video time when compared to the rest of the clients. We can also see that an I/O bottleneck occurred in the network with clients connected through other switches, while the computers in the same switch were the ones that did not lose anything.

Table 5.1: Losses - 68 Clients/One thread

Computer	Received (MB)	Lost (%)	Bit rate (Kb/s)	Time
tux22	1 282	3.4	10343	17:20.2
tux23	1 288	2.9	10375	17:21.6
tux32	1 296	2.3	10423	17:23.3
tux33	1 292	2.6	10375	17:21.2
tux12	1 264	4.7	9973	17:21.2
tux13	1 297	2.3	10235	17:43.5
tux62	1 251	5.7	9874	17:43.5
tux63	1 254	5.5	9893	17:43.5
tux42	1 325	0.2	10455	17:43.5
tux43	1 326	0.1	10463	17:43.4
tux34 (8 users)	1 274	4.0	10051	17:43.5
Average	1 286	3.0	10223	-

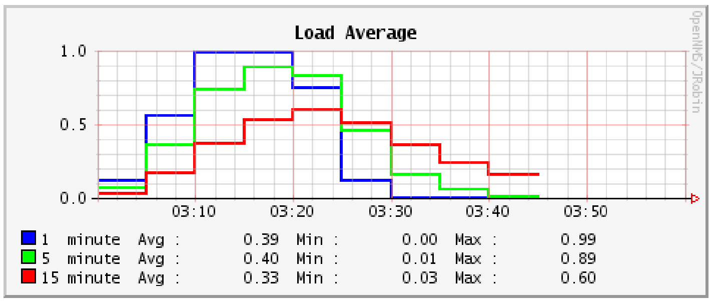


Figure 5.4: Load Average - 68 clients

Figure 5.4 shows the computer load when all those clients connected to one single Live555 process. Obviously this was an undesired situation. As Live555 used one single thread this meant that the CPU usage concerned only one single core. When a CPU load reaches 1.00 that's because something needs to be fixed. Even with 0.70 load the problem should be looked into and resolved.

With some research and source code inspection we actually found a way to decrease this CPU load, however, that would require a long time wasted in development. Live555 is built using *fread()* function to read its content from a file, and *sendto()* to send its packets to the respective socket. This implicates that the video frames need to (i) reach the user space, (ii) be encapsulated into RTP packets, (iii) sent to the kernel again and afterwords through the UDP connection. In a user perspective 4 copies were made during this procedure and as many context switches. First a direct copy is made from the disk space to the kernel buffer, afterwords this information is copied to the user space for processing. When its done, the formed packets are copied to the kernel space into a socket buffer, and then sent through the respective protocol engine. Figure 5.5 exemplifies this situation.

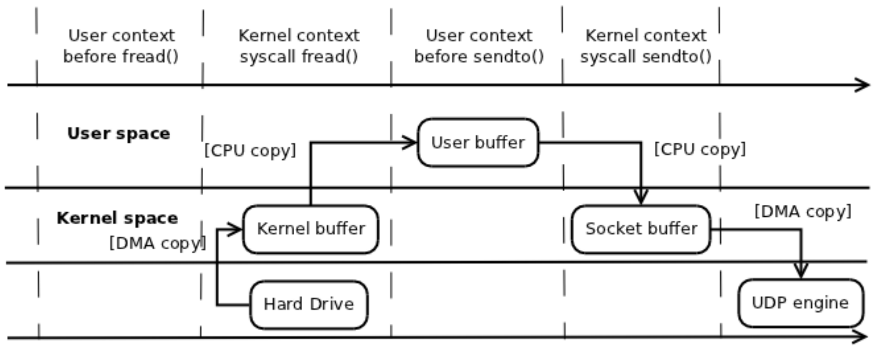


Figure 5.5: Read + Sendto()

To overcome the excess of copies a zero-copy method was introduced using a system call *sendfile()*. Instead of copying the information to the user space, the kernel buffer memory is appended to the socket buffer eliminating the need of copying memory unnecessarily. This however, would require a pre-processing of the packets into a file in order to skip the user space access. A recent study [52] demonstrates that implementing a zero-copy solution using Live555 tool and a simple MP3 sample could lead to an impressive 15% of CPU reduction, when compared to the original implementation.

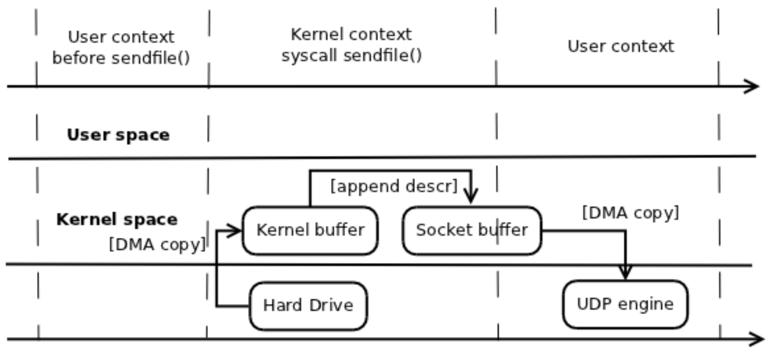


Figure 5.6: sendfile()

Yet, there was still another simple solution that could help us overcome this problem. At least another Live555 process should be running in the server station, so the clients could then be channelled through both Live555 threads. With a second Live555 process we conducted an experiment using 60 clients divided by both processes and this time the results were very satisfactory.

From table 5.2 we can conclude that the clients did not suffer any loss at all. Values rounding 0.4 are practically insignificant. This means that the video stream was flawless, and provided

Table 5.2: Losses - 60 Clients/Two threads

Computer	Received (MB)	Lost (%)	Bit rate (Kb/s)	Time
tux12	1 322	0.4	10428	17:43.5
tux13	1 322	0.4	10426	17:43.5
tux22	1 322	0.4	10425	17:43.5
tux23	1 321	0.5	10415	17:43.5
tux32	1 321	0.5	10421	17:43.5
tux62	1 322	0.4	10425	17:43.5
tux24	1 320	0.5	10410	17:43.5
tux44	1 327	0.0	10465	17:43.5
tux54	1 321	0.5	10417	17:43.5
tux64	1 322	0.4	10427	17:43.5
Average	1 322	0.4	10426	-

the best viewing experience for each user. By interpreting figure 5.7 we could now see a huge difference when compared with the one threaded case. Both the System and User CPU reached a maximum of 20%, meaning a 30/40% of utilization in two cores. Using this method we could perfectly scale all the users in order to balance the CPU load. As we were now using two cores the limit value could go up to 1.40 of CPU load. In figure 5.8 we can see that the load is way below those threshold values with a single peak around 0.8. Although the results were excellent we were yet to find out how many users the system could actually handle in terms of network bandwidth.

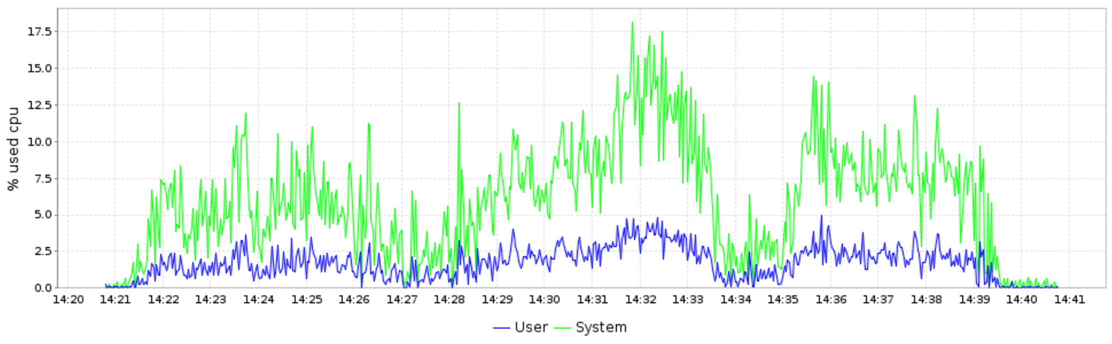


Figure 5.7: CPU Usage - 60 clients

This experiment was obviously a worst case scenario with all the clients requesting the same video file, almost at the same time. According to the traffic transmitted from the Gigabit interface (see fig. 5.9), we discovered that during a few seconds along the video stream all the 60 clients would essentially require the full Gigabit capacity. In a normal situation not all the clients would connect at the same time nor request the same video file.

In the next experiments in order to avoid this occurrence we separated each client with a 10 seconds delay and ran each script a few seconds apart. We were able to successfully reach 78 clients demonstrating that perhaps in a live case scenario a few more could be connected to

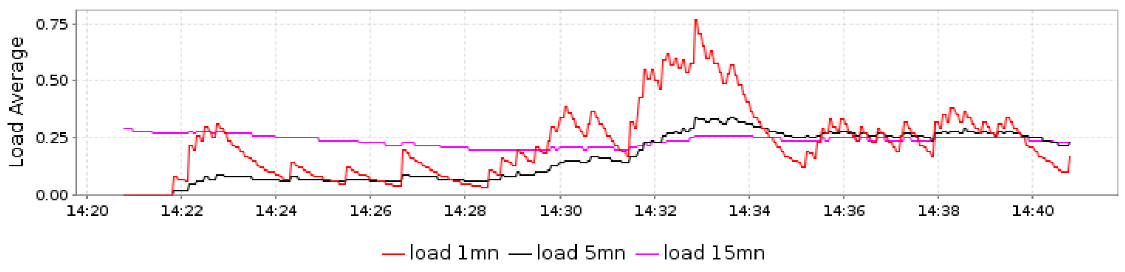


Figure 5.8: Load Average - 60 clients

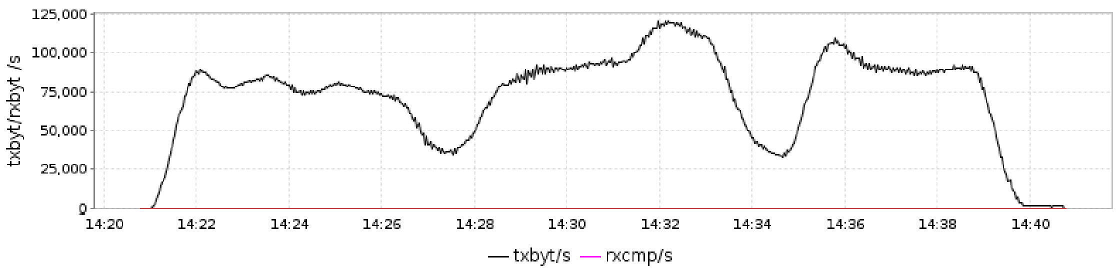


Figure 5.9: Transmitted bytes - 60 clients

this server station. In table 5.3 we can see that there were no significant losses during the test, demonstrating that no bottlenecks occurred during the video streaming.

The CPU usage was a little bit higher when compared with the previous case, but nothing to be concerned about (see fig. 5.10). The Load also achieved a higher value, but again, with two threads running at the same time, the limit values were around 1.40. Both of these results indicate that in terms of performance only a 1/4 of the total capacity was being used indicating that in the future it could scale even more if needed. Only the graphic concerning the total transferred bytes (see fig. 5.12) may suggest that in a normal usage situation a few more clients could connect to the service. For a HD scenario with 10Mbps of bit rate, reaching 80% of the theoretical capacity is a very good result.

The last conducted experiment concerned a Standard Definition (SD) scenario with a 4Mbps bit rate. Using FFMpeg tool we downsized a 10Mbps video to around 4Mbps average bit rate, a format more commonly used in television broadcast resolution. The result was a video file with an average of 4525kbps and a total size of 532MBs with a duration of 16:25.28. This time we could introduce 12 clients per computer maximizing the number of clients. During initial tests using only two Live555 processes and reaching 60 clients per each thread the computer would reach prohibitive loads, just like the situation with 68 clients. To overcome this problem two more processes were initiated to provide a better load balancing. With 4 running threads we reached a total of 168 connected clients, channelling 42 clients per thread. Each client had again a 10 seconds delay between the next one and all the scripts were initiated separately, with a few seconds of delay between each other.

Table 5.3: Losses - 78 Clients/Two threads

Computer	Received (MB)	Lost (%)	Bit rate (Kb/s)	Time
tux12	1 320	0.5	10409	17:43.5
tux13	1 321	0.5	10419	17:43.5
tux14	1 325	0.2	10450	17:43.5
tux23	1 321	0.5	10422	17:43.5
tux24	1 322	0.4	10425	17:43.5
tux32	1 320	0.5	10408	17:43.5
tux33	1 311	1.2	10340	17:43.5
tux34	1 323	0.3	10432	17:43.5
tux42	1 322	0.4	10426	17:43.5
tux43	1 320	0.5	10413	17:43.4
tux44	1 323	0.3	10437	17:43.5
tux62	1 321	0.5	10417	17:43.5
tux63	1 322	0.4	10425	17:43.5
Average	1 321	0.5	10417	-

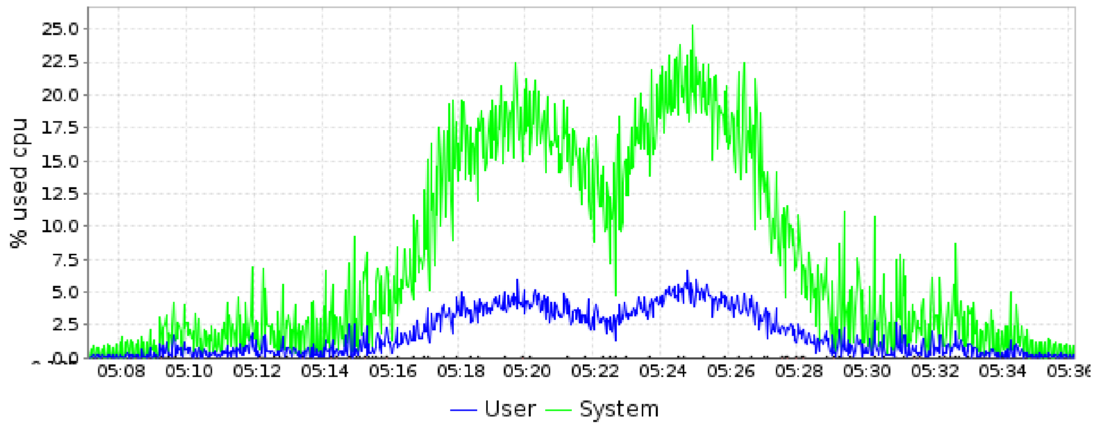


Figure 5.10: CPU Usage - 78 clients

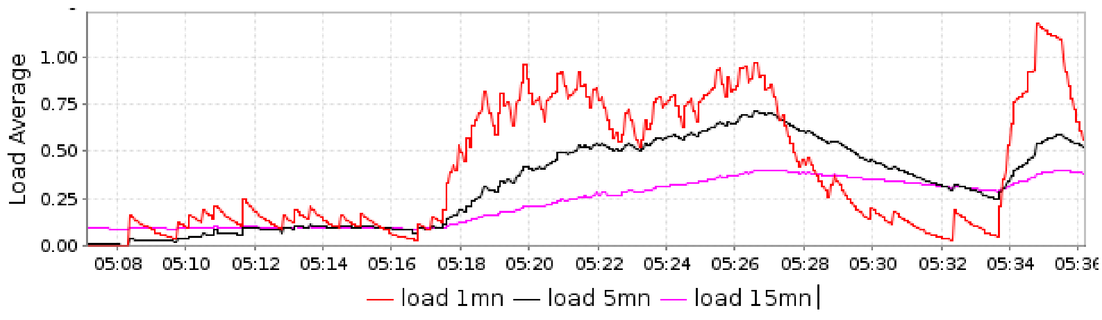


Figure 5.11: Load Average - 78 clients

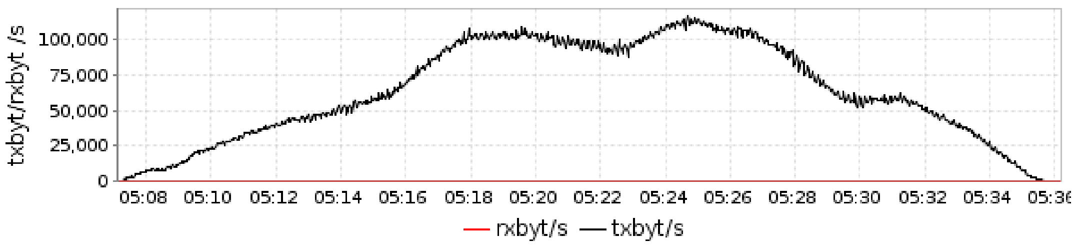


Figure 5.12: Transmitted bytes - 78 clients

Table 5.4 shows that this was yet another successful test. This time the losses were barely noticeable and demonstrated that with 168 clients at 4.5Mbps we could reach over 80% of the maximum network capacity. Figures 5.13 and 5.14 show a normal usage and indicate that the server station could support even more users, as the maximum load would now be 2.80. Figure 5.15 registers a more heterogeneous traffic compared with the previous examples, which attests that with more clients the experience could better simulate a normal usage.

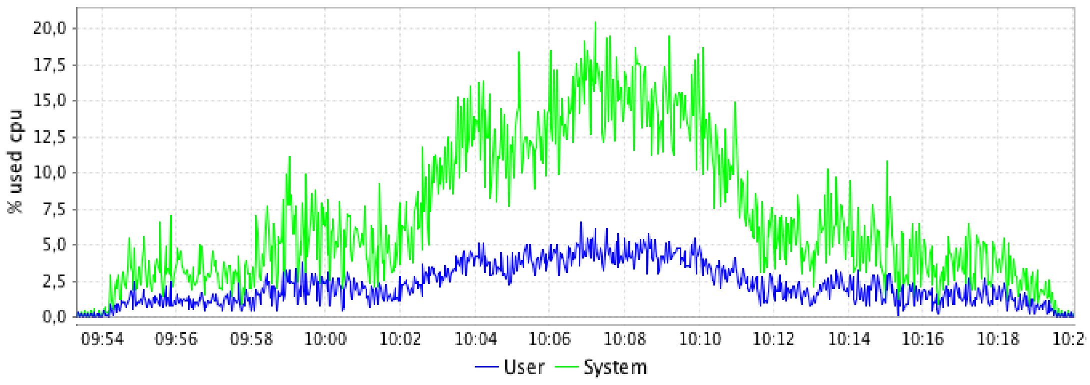


Figure 5.13: CPU Usage - 168 clients

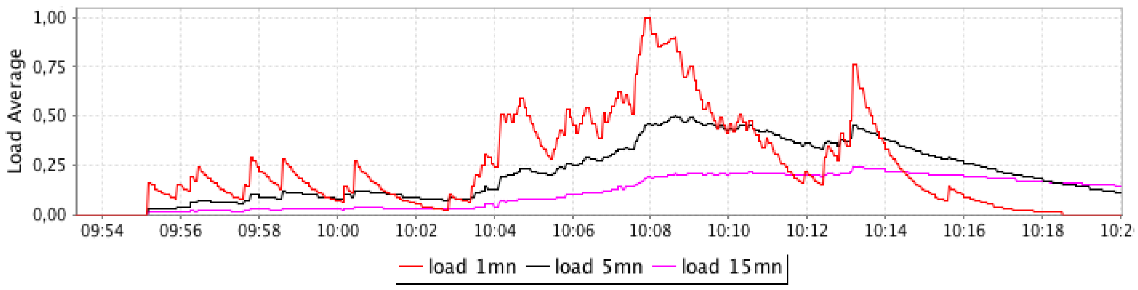


Figure 5.14: Load Average - 168 clients

Table 5.4: Losses - 168 Clients/Four threads

Computer	Received (MB)	Lost (%)	Bit rate (Kb/s)	Time
tux12	531.0	0.11	4520	16:25.3
tux13	531.1	0.09	4521	16:25.3
tux14	531.4	0.03	4523	16:25.3
tux32	531.3	0.04	4523	16:25.3
tux33	530.6	0.18	4517	16:25.3
tux34	531.4	0.03	4524	16:25.3
tux42	531.4	0.03	4523	16:25.3
tux43	531.3	0.05	4522	16:25.3
tux44	531.4	0.03	4523	16:25.3
tux52	530.8	0.15	4518	16:25.3
tux53	530.8	0.15	4512	16:25.3
tux62	531.3	0.05	4523	16:25.3
tux63	531.4	0.02	4524	16:25.3
tux64	531.5	0.02	4524	16:25.3
Average	531.2	0.07	4521	-

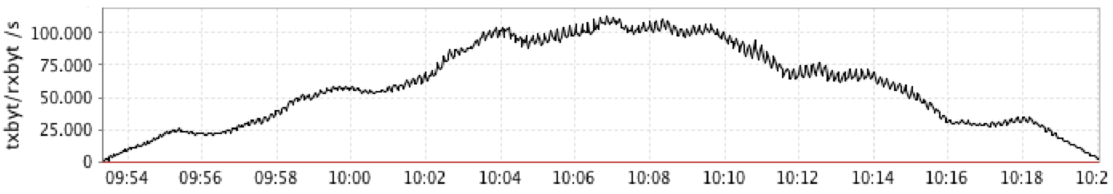


Figure 5.15: Transmitted bytes - 168 clients

5.2.2 Discussion

The results were conclusive, our Desktop deployment proved to be capable of dealing with the service demand both in SD and HD circumstances. Besides, we could actually achieve even more if it was not for the network bandwidth. This implicated a single Gigabit interface that it is normally available in dedicated servers, but some may actually provide a Dual Gigabit interface. In that case our service could easily reach twice as much users as our experiences. In both our cases we nearly reached the maximum full capacity. Values around 80% are in our opinion very good considering that it is difficult to mimic a real case scenario.

In the end the computer load proved to be the least of our problems. We would be perfectly capable of delivering the same service with a less capable server. Nevertheless, VOD services require much more than perfect computers and perfect network bandwidths. The development of streaming applications must be based on other major aspects: content allocation and request routing. In our example we showed that 78 clients were enough to reach almost the full capacity of our network. This helps to demonstrate the difficulties in balancing the network and computer's capacity. In a real situation this wont happen because each client may request a different video stream, and be routed through different media server's to properly scale the network. When routing each client through the many media servers we might have, the main interface should be aware of all these aspects. First it must check the media server's load, and if the current number of clients are not exceeding the throughput capacity. Routing a client to a possible bottleneck situation in a near future cannot happen because that would mean the service would degrade. The media server could perhaps downgrade the video resolution to accommodate all the clients, but with a trade-off of delivering a worst media experience. This is called the adaptive bit rate streaming, and already happens in some enterprise applications. Another downsize is the fact that several types of video files, encoded in different bit rates, are needed to support this kind of feature.

With the current media server development we can actually control each running thread independently, and properly route each client. In our case we needed to change the SSL server port in each process to control all the running threads. The statistical feedback would also need a few tweaks to report the current process and not just the current system statistics, which is easily provided with the LibStatGrab tool.

Other concern is the cost of keeping digital video content stored in every location. Normally content allocation implicates replicating the content through other servers, reducing the bandwidth in each media server increasing the storage cost. This is highly impractical because not every content is accessed, due to low popularity. Placing media contents in the servers according to popularity levels may be the best solution. This approach has shown in other examples [53] that storing popular content can accommodate up to 80% of the service requests.

The main interface can be used to keep track of video content popularity (i.e. the times each video is accessed) and to locate which media content is in each media server. This interface was also designed to easily apply routing algorithms, placing each user in a media server that it's close to him or to any server according to its load and capacity.

Content delivery is and will always be the most costly service when relating to video-on-demand. A direct comparison with existing CDNs is very hard to accomplish because we can not compare our application to a world wide network of private owned/rented computers, providing thousands of video streams per day, with fully developed routing and content allocation algorithms. Neither their software is available for companies and professionals to acquire, only their pre-defined VOD services that use their CDN network. Akamai [54], one of the CDNs used by Netflix for example, is responsible for 20% of the world's network traffic, reaching more than 1 million on demand streams in peak hours.

Still, several existing companies manufacture software and hardware easily comparable with our application. Normally the hardware specifies a rack unit with determined capacity to support VOD services, including a server with many features mentioned in this discussion, like specific content distribution. Among those companies is Anevia [55], that sells the Toucan VOD server, specific for corporations and hospitals. The Toucan 100, the middle package, refers to a rack unit capable of providing 60 streams (any format or bitrate) with 4TB of disk space, while the Toucan 500 refers to a native software capable of delivering unlimited streams depending on the hardware. This two examples demonstrate how our application can actually rival capacity-wise with enterprise solutions. Other example is the NetUP VOD server [56] commercialized by NetUP, a rack unit that according to the datasheet is capable of delivering over 100 concurrent streams at 4Mbps. This to be true implies that our application running in our Desktop Station is almost twice as better as NetUP solution.

Live555 application was definitely the right choice to start building this VOD application. It was capable of delivering a perfect streaming solution to 78 and 168 clients, and could easily support more than that. One cannot forget that the used video file was encoded with MPEG2 video codec, a format normally used with STBs, but if encoded with H264, it would mean that we would deliver the same definition with a lower bit rate both in SD and in HD content. This would lead to a slight increase of available bandwidth, providing extra capacity for more users.

The method of the zero-copy would actually help improving the server's capacity to greater numbers. And this is just related with VOD because as we know, Live555 could easily be configured for Music and other types of content, that require less bandwidth, much less. From a simple company deployment to a large scale VOD application our system would definitely be ready of delivering a reliable service, with the ability to scale and evolve according to the future needs.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The purpose of this thesis was to create a feasible structure for a VOD system capable of dealing with IPTV services in a reliable manner. We required a system capable of streaming video content to a client, and be administered through a web interface. By researching the available technologies, we could have a better perception of all the pieces that were needed to build this system.

Perhaps the biggest challenge was to properly expose our service to others, providing the most common tools. The Main Interface was where all this happened. Fully supported by a reliable and stable software, the Apache Web Server, our services could then be integrated in a variety of different applications, even in capable cellphones with a network connection. Security was also a major concern. Using the SOAP tools we could deliver a complete set of security options, that no other tool could provide. However, because of software incompatibilities and time issues, the REST implementation did not achieve the basic requisites included in the software specifications. Despite that with more development we would easily offer a complete REST solution.

On the other hand, the client application gave a good perception of what the future of Java might be in graphical terms. During the development, only the JavaFX script could allow the JavaFX tools, but in the future this will not be the case. The future Java development kit will have this tools available to develop with any Java application. Those tools demonstrated a new set of capabilities, indicating perhaps that Java might be the new Flash competitor for RIA applications.

The Live555 choice was very helpful and complete enough to deal with the needed streaming services. Only a proper channel of communications was needed to manage and retrieve all the session and statistic's information. With such feature, we were able to fully control each available media server through a web-based interface. The SSL channels also provided the best security option in this communication. Yet, the best part was perhaps the acknowledgement of the full potential that Live555 has.

VOD solutions do not come cheap. They have network costs, besides hardware and software. By using open-source tools and joining them into one big application, we proved to be possible to create a free VOD structure that could easily level with any available product, even by simulating a worst case scenario.

6.2 Future Work

The system that we created was not complete however. The VOD solutions nowadays have a lot more to offer. Our results demonstrated that features like request routing and content allocation, which are usually offered in commercial solutions, are mandatory in this kind of applications. Managing all communications as we did with our Main Interface could help us enable all those features.

More robust and sophisticated solutions were missing in the current implementation. As mentioned before, the database could be arranged to provide some sort of special accounts where the users would have access only to certain video contents. Scaling those accounts through different privileges would also be an interesting option with several available packages (e.g. Basic package, Premium package). A billing solution could provide those choices to the user.

In terms of administration, only the Main Interface was capable of providing a log with different types of levels. The Apache Server and the Axis2 logging tools both provide those options, with log outputs being written into a separate file. Live555 provides a debugging log but it required a little bit more work to offer a complete log in a separate file. Ultimately, this log files could converge into a single system log server or be displayed in the administration interface. Other features should also be available, like controlling the number of clients that connect to each media server, logging those events and provide content access statistics.

Preparing the system for other IPTV services should be the number one priority. Music, and Live broadcast could be easily integrated to provide the full IPTV package as the database and Live555 were already designed to support such kind of contents. The more diverse content we provided to the costumers, the more satisfied they would be.

Appendix A

Main Interface

A.1 user_db Struct Reference

User's information structure.

Public Attributes

- std::string **username**
- std::string **password**

A.2 content_db Struct Reference

Content information structure. All the information must be retrieved from the appropriate websites.

Public Attributes

- int **id**
- int **length**
- int **year**
- float **rating**
- std::string **moviename**
- std::string **description**
- std::string **filename**
- std::string **lang**
- std::string **imgurl**

- std::vector< std::string > **genres**
- std::vector< std::string > **part_actor**
- std::vector< std::string > **part_writer**
- std::vector< std::string > **part_producer**
- std::vector< std::string > **part_director**

A.3 associations_db Struct Reference

Movies associations's information structure. Movies associations's information structure. All the information must is retrieved from the apropriate websites.

Public Attributes

- std::string **name**
- int **id**

A.4 Database Class Reference

Main interface database API. This Database Class contains all the necessary values and methods to properly manipulate the server's database. The database framework used in the development was SQLITE3, a compact and very reliable SQL library with a C interface.

Public Attributes

- **Database** (char ∗filename)
- bool **database_open** (char ∗filename)

  Open the database for content management.
- std::vector< std::vector< std::string > > **database_query** (char ∗query)

  First it compiles the statement with the function sqlite3_prepare_v2(), if the query is properly set this returns SQLITE_OK. After the preparation we check what type of statement we are dealing with with sqlite_step(). If the SQL statement being executed returns any data, then SQLITE_ROW is returned each time a new row of data is ready for processing by the caller. sqlite3_step() is called again to retrieve the next row of data if present.
- int **database_close** ()

  Close the database.

- int **insert_server** (std::string mediaServer)

  Insert a media Server hostname into the server's database.

- int **remove_server** (std::string mediaServer)

  Remove a media Server hostname from the server's database.

- std::vector< std::string > **get_servers** ()

  Retrieve all mediaServer's hostname from the server's database.

- int **insert_movie** (**content_db** movie_info)

  Insert a movie's information into the server's database.

- int **insert_user** (**user_db** user_info)

  Insert a user's information into the server's database.

- int **insert_associations** (int table, **associations_db** participates)

  Insert the participants/users and content relationship.

- int **insert_genres** (int id, std::vector< std::string > genres)

  Insert the movie genres.

- void **database_create** ()

  Create VoD database.

- int **database_status** (char ∗query)

  Check database status.

- void **print_query** (std::vector< std::vector< std::string > > result)

  Print query. (Debugging purposes)

- std::vector< std::string > **get_ContentList** (const std::string username, const std::string search)

  Retrieve all the contents accessible for a specific user and a possible search string. For the current version the access feature is off.

- **content_db get_Content** (int id)

  Retrieve all the information regarding a specific content

- std::vector< std::string > **get_Users** ()

  Retrieve all the users registered in the service.

- std::string **get_Password** (std::string username)

  Retrieve a user's password.

- int **edit_Password** (**user_db** user_info)

  Change a user's password.

- int **edit_Content** (**content_db** movie_info)

  Edits the existing content

- int **delete_Content** (int id)

  Delete existing content.

A.5 SSLClient Class Reference

SSL Client for the main interface implementation. The SSL Client class includes all the necessary methods to communicate with a encrypted and secure connection with all the streaming server's available

Public Attributes

- **SSLClient** (const char ∗hostandport)

- BIO ∗ **connect_encrypted** (char ∗host_and_port, char ∗store_path, char store_type, SSL_-CTX ∗ ∗ctx, SSL ∗ ∗ ssl)

  Open an encrypted connection. Returns I/O handler for SSL connections.

- BIO ∗ **connect_unencrypted** (char ∗host_and_port)

  Open an unencrypted connection. Normal socket.

- int **write_to_stream** (BIO ∗bio, char ∗buffer, ssize_t length)

  Write data to an SSL connection. Returns number of characters written, 0 if there's any error.

- ssize_t **read_from_stream** (BIO ∗bio, char ∗buffer, ssize_t length)

  Read data from an SSL connection. Returns number of characters read, 0 if there's any error.

- int **close_connection** ()

  Close an SSL connection. Return >0 if successful, 0 otherwise.

- int **addMediaSessionAccess** (char const ∗username, char const ∗password)

  Add Media Session access to a specific user. Return SUCCESS if successful, EXISTS if it exists.

- int **loadMediaSession** (char const ∗streamName)

  Loads Media Session. Return SUCCESS if successful, EXISTS if it exists.

- int **checkMediaSession** (char const ∗streamName)

  Check Media Session. Return >0 if successful, 0 otherwise.

- int **termMediaSession** (char const ∗streamName)

  Indicates the session is to be finalized. Return SUCCESS if successful, ERROR otherwise.

- char ∗ **getURL** (char const ∗streamName)

  Get Media Session URL. Returns Media Session URL

- char ∗ **getStats** ()

  Get Media Session system statistics. Returns Media Session statistics in a C string.

- int **checkResponse** (char const ∗buffer)

  Check server response. Returns type of response.

- int **termConnection** ()

  Terminate connection.

- void **print_ssl_error** (char ∗message, char ∗content, FILE ∗out)

  Print the SSL error.

- int **open_connection** ()

  Open an SSL connection. Return >0 if successful, 0 otherwise.

A.6 ClientService Class Reference

Client web service's API. This class contains all the necessary methos to implement the available client's web services.

Public Attributes

- OMElement ∗WSF_CALL **invoke** (OMElement ∗message, MessageContext ∗msgCtx)

  This method is called for handling the service's invocation. Services implement this method in order to process the SOAP message's content and acknowledge which operation was issued, using for that matter the WS-Addressing information retrieved from the SOAP header. Returns the response OMElement.

- OMElement ∗WSF_CALL **onFault** (OMElement ∗message)

- void WSF_CALL **init** ()

- OMElement ∗ **createClient** (OMElement ∗ele)

  This functions is called when someone wants to create a new client to access the service. Returns the response OMElement constructed.

A.7 AdminService Class Reference

Admin web service's API. This class contains all the necessary methos to implement the available admin's web services.

Public Attributes

- OMElement ∗WSF_CALL **invoke** (OMElement ∗message, MessageContext ∗msgCtx)
- OMElement ∗WSF_CALL **onFault** (OMElement ∗message)
- OMElement ∗WSF_CALL **onExists** (OMElement ∗ele)
- OMElement ∗WSF_CALL **onSuccess** (OMElement ∗ele)
- OMElement ∗WSF_CALL **onMSFault** (OMElement ∗ele)
- OMElement ∗WSF_CALL **ondbFault** (OMElement ∗ele)
- OMElement ∗WSF_CALL **onCastingFault** (OMElement ∗ele)
- void WSF_CALL **init** ()
- OMElement ∗ **addContent** (OMElement ∗ele)
- OMElement ∗ **addMediaSessionAccess** (OMElement ∗ele)
- OMElement ∗ **editContent** (OMElement ∗ele)
- OMElement ∗ **deleteContent** (OMElement ∗ele)
- OMElement ∗ **addServer** (OMElement ∗ele)
- OMElement ∗ **removeServer** (OMElement ∗ele)
- OMElement ∗ **getServers** (OMElement ∗ele)
- OMElement ∗ **loadMediaSession** (OMElement ∗ele)
- OMElement ∗ **checkMediaSession** (OMElement ∗ele)
- OMElement ∗ **terminateMediaSession** (OMElement ∗ele)
- OMElement ∗ **getStatistics** (OMElement ∗ele)
- OMElement ∗ **getUsers** (OMElement ∗ele)

A.8 UserService Class Reference

User web service's API. This class contains all the necessary methos to implement the available user's web services.

Public Attributes

- OMElement ∗WSF_CALL **invoke** (OMElement ∗message, MessageContext ∗msgCtx)
- OMElement ∗WSF_CALL **onFault** (OMElement ∗message)
- OMElement ∗WSF_CALL **onMSFault** (OMElement ∗ele)
- OMElement ∗WSF_CALL **ondbFault** (OMElement ∗ele)
- void WSF_CALL **init** ()
- OMElement ∗ **getMediaUrl** (OMElement ∗ele)
- OMElement ∗ **getContentList** (OMElement ∗ele)
- OMElement ∗ **getContent** (OMElement ∗ele)
- OMElement ∗ **editPassword** (OMElement ∗ele)

A.9 Web Services

A.9.1 User WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://feupvod/services"
xmlns:xsd1="http://feupvod/schema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:http="http://www.w3.org/2003/05/soap/bindings/HTTP/"
xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
targetNamespace="http://feupvod/services">

<!-- TYPES

The types element describes all the data types used between the client and server. WSDL
is not tied exclusively to a specific typing system, but it uses the W3C XML Schema
specification as its default choice. If the service uses only XML Schema built-in simple
types, such as strings and integers, the types element is not required.

-->
<types>
  <xsd:schema elementFormDefault="qualified" targetNamespace="http://feupvod/schema">
      <xsd:element name="ContentListinfo">
   <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="movieinfo" maxOccurs ="unbounded">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="movieid" type="xsd:int"/>
```

```
              <xsd:element name="moviename" type="xsd:string"/>
              <xsd:element name="year" type="xsd:int"/>
              <xsd:element name="imgurl" type="xsd:string"/>
           </xsd:sequence>
       </xsd:complexType>
     </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
     </xsd:element>

    <xsd:element  name="Contentinfo">
    <xsd:complexType>
       <xsd:sequence>
          <xsd:element name="id" type="xsd:int"/>
          <xsd:element name="moviename" type="xsd:string"/>
          <xsd:element name="year" type="xsd:int"/>
          <xsd:element name="rating" type="xsd:decimal"/>
          <xsd:element name="imgurl" type="xsd:string"/>
          <xsd:element name="lang" type="xsd:string"/>
          <xsd:element name="length" type="xsd:int"/>
          <xsd:element name="description" type="xsd:string"/>
          <xsd:element name="filename" type="xsd:string"/>
          <xsd:element name="Genres">
       <xsd:complexType>
           <xsd:sequence>
              <xsd:element name="genre" type="xsd:string" maxOccurs ="unbounded"/>
           </xsd:sequence>
       </xsd:complexType>
     </xsd:element>
     <xsd:element name="Actors">
       <xsd:complexType>
           <xsd:sequence>
              <xsd:element name="actorname" type="xsd:string" maxOccurs ="unbounded"/>
           </xsd:sequence>
       </xsd:complexType>
     </xsd:element>
     <xsd:element name="Directors">
       <xsd:complexType>
           <xsd:sequence>
              <xsd:element name="directorname" type="xsd:string" maxOccurs ="unbounded"/>
           </xsd:sequence>
       </xsd:complexType>
     </xsd:element>
     <xsd:element name="Producers">
       <xsd:complexType>
           <xsd:sequence>
              <xsd:element name="producername" type="xsd:string" maxOccurs ="unbounded"/>
           </xsd:sequence>
       </xsd:complexType>
     </xsd:element>
     <xsd:element name="Writers">
       <xsd:complexType>
           <xsd:sequence>
              <xsd:element name="writername" type="xsd:string" maxOccurs ="unbounded"/>
           </xsd:sequence>
       </xsd:complexType>
     </xsd:element>
```

```
        </xsd:sequence>
     </xsd:complexType>
  </xsd:element>

  <xsd:element name="editPasswordinfo">
    <xsd:complexType>
        <xsd:sequence>
           <xsd:element name="username" type="xsd:string"/>
           <xsd:element name="oldpassword" type="xsd:string"/>
           <xsd:element name="newpassword" type="xsd:string"/>
        </xsd:sequence>
     </xsd:complexType>
        </xsd:element>

  <xsd:element name="getContentList">
    <xsd:complexType>
        <xsd:sequence>
           <xsd:element name="search" type="xsd:string"/>
        </xsd:sequence>
     </xsd:complexType>
        </xsd:element>

  <xsd:element name="getMediaUrl">
    <xsd:complexType>
        <xsd:sequence>
           <xsd:element name="streamName" type="xsd:string"/>
           <xsd:element name="ipaddress" type="xsd:string"/>
        </xsd:sequence>
     </xsd:complexType>
        </xsd:element>

  <xsd:element name="getContent">
    <xsd:complexType>
        <xsd:sequence>
           <xsd:element name="movie_id" type="xsd:int"/>
        </xsd:sequence>
     </xsd:complexType>
        </xsd:element>

  <xsd:element name="ServiceResponse">
    <xsd:complexType>
        <xsd:sequence>
           <xsd:element name="Response" type="xsd:string"/>
        </xsd:sequence>
     </xsd:complexType>
        </xsd:element>

   </xsd:schema>
</types>

<!--MESSAGES

  The message element describes a one-way message, whether it is a single message request
or a single message response. It defines the name of the message and contains zero or more
message part elements, which can refer to message parameters or message return values.

-->
```

```
<message name="getMediaUrlRequest">
   <part name="medianame" element="xsd1:getMediaUrl"/>
</message>

<message name="getContentListRequest">
   <part name="SearchParam" element="xsd1:getContentList"/>
</message>

<message name="getContentListResponse">
   <part name="contentListinfo" element="xsd1:ContentListinfo"/>
</message>

<message name="getContentRequest">
   <part name="movieId" element="xsd1:getContent"/>
</message>

<message name="getContentResponse">
   <part name="contentinfo" element="xsd1:Contentinfo"/>
</message>

<message name="editPasswordRequest">
   <part name="editPasswordinfo" element="xsd1:editPasswordinfo"/>
</message>

<message name="UserServiceResponse">
   <part name="Response" element="xsd1:ServiceResponse"/>
</message>

<!--PORTYPES

   The portType element combines multiple message elements to form a complete one-way or
round-trip operation. For example, a portType can combine one request and one response
message into a single request/response operation, most commonly used in SOAP services.
Note that a portType can (and frequently does) define multiple operations.

-->
<portType name="UserService_PortType">
   <operation name="getMediaUrl">
      <input message="tns:getMediaUrlRequest"/>
      <output message="tns:UserServiceResponse"/>
   </operation>

   <operation name="getContentList">
      <input message="tns:getContentListRequest"/>
      <output message="tns:getContentListResponse"/>
   </operation>

    <operation name="getContent">
      <input message="tns:getContentRequest"/>
      <output message="tns:getContentResponse"/>
   </operation>

   <operation name="editPassword">
      <input message="tns:editPasswordRequest"/>
      <output message="tns:UserServiceResponse"/>
   </operation>
```

```
</portType>

 <!--BINDING

  The binding element describes the concrete specifics of how the service will be
implemented on the wire. WSDL includes built-in extensions for defining SOAP services,
and SOAP-specific information therefore goes here.

-->
<binding name="UserService_Binding" type="tns:UserService_PortType">
  <soap:binding  xmlns="http://schemas.xmlsoap.org/wsdl/soap/"
              transport="http://schemas.xmlsoap.org/soap/http" style="document"/>

   <operation xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/" name="getMediaUrl">
    <soap:operation xmlns="http://schemas.xmlsoap.org/wsdl/soap/"
                 soapAction="getMediaUrl" style="document"/>
      <input xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/">
         <soap:body use="literal"/>
      </input>
      <output xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/">
         <soap:body use="literal"/>
      </output>
   </operation>

   <operation xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/" name="getContentList">
    <soap:operation xmlns="http://schemas.xmlsoap.org/wsdl/soap/"
                 soapAction="getContentList" style="document"/>
      <input xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/">
         <soap:body use="literal"/>
      </input>
      <output xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/">
         <soap:body use="literal"/>
      </output>
   </operation>

   <operation xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/" name="getContent">
    <soap:operation xmlns="http://schemas.xmlsoap.org/wsdl/soap/"
                 soapAction="getContent" style="document"/>
      <input xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/">
         <soap:body use="literal"/>
      </input>
      <output xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/">
         <soap:body use="literal"/>
      </output>
   </operation>

   <operation xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/" name="editPassword">
    <soap:operation xmlns="http://schemas.xmlsoap.org/wsdl/soap/"
                 soapAction="editPassword" style="document"/>
      <input xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/">
         <soap:body use="literal"/>
      </input>
      <output xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/">
         <soap:body use="literal"/>
      </output>
   </operation>
```

```
</binding>


<!--SERVICE

   The service element defines the address for invoking the specified service.
Most commonly, this includes a URL for invoking the SOAP service.

-->
   <service name="User_Service">
      <port xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/"
             name="UserService_Port"
               binding="tns:UserService_Binding">
         <soap:address  xmlns="http://schemas.xmlsoap.org/wsdl/soap/"
            location="http://localhost:9090/axis2/services/user"/>
      </port>
   </service>
</definitions>
```

A.9.2 Apache Integration

```
LoadModule axis2_module /usr/lib/apache2/modules/mod_axis2.so
Axis2RepoPath /etc/wso2/wsfcpp
Axis2LogFile /tmp/axis2.log
Axis2LogLevel trace
Axis2ServiceURLPrefix http://192.168.1.201/axis2/services
Axis2MaxLogFileSize 10

<Location /axis2>
  SetHandler axis2_module
</Location>
```

A.9.3 SOAP message exchange

```
Hypertext Transfer Protocol
    POST /axis2/services/user HTTP/1.1\r\n
    User-Agent: Axis2C/1.7.0\r\n
    SOAPAction: "getContentList"\r\n
    Content-Length: 3691\r\n
    Content-Type: text/xml;charset=UTF-8\r\n
    Host: localhost:9090\r\n
    \r\n
eXtensible Markup Language
 <soapenv:Envelope
     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
     <soapenv:Header
         xmlns:wsa="http://www.w3.org/2005/08/addressing">
         <wsa:To>
             http://localhost:9090/axis2/services/user
             </wsa:To>
         <wsa:Action>
             getContentList
             </wsa:Action>
         <wsa:MessageID>
             urn:uuid:2b9ddaa6-9679-1e01-2734-000000000000
             </wsa:MessageID>
```

```
<wsse:Security
    xmlns:wsse="http://.../oasis-200401-wss-wssecurity-secext-1.0.xsd"
    soapenv:mustUnderstand="1">
    <wsse:BinarySecurityToken
        EncodingType="http://.../...soap-message-security-1.0#Base64Binary"
        xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
        wsu:Id="CertID-2ba3d2ee-9679-1e01-2735"
        ValueType="http://.../oasis-200401-wss-x509-token-profile-1.0#X509v3">
        [truncated] ...T0FTSVMxIDAeBgNVBAsMF09BU0lTIEludGVyb3A
        </wsse:BinarySecurityToken>
    <wsse:UsernameToken
        xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
        wsu:Id="SigID-2ba3dfc8-9679-1e01-2736">
        <wsse:Username>
            admin
            </wsse:Username>
        <wsse:Password
            Type="http://.../...username-token-profile-1.0#PasswordDigest">
            yUg7nVx136HR0ZYDjzJNZkqw194=
            </wsse:Password>
        <wsse:Nonce>
            DVCa0bRJ756bdUBDvvcz2Ok1+boxtxx3
            </wsse:Nonce>
        <wsu:Created>
            2011-06-14T11:26:38.848Z
            </wsu:Created>
        </wsse:UsernameToken>
    <ds:Signature
        xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
        Id="SigID-2ba3e22a-9679-1e01-2737">
        <ds:SignedInfo>
            <ds:CanonicalizationMethod
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                </ds:CanonicalizationMethod>
            <ds:SignatureMethod
                Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1">
                </ds:SignatureMethod>
            <ds:Reference
                URI="#SigID-2ba3dfc8-9679-1e01-2736">
                <ds:Transforms>
                    <ds:Transform
                        Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                        </ds:Transform>
                    </ds:Transforms>
                <ds:DigestMethod
                    Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
                    </ds:DigestMethod>
                <ds:DigestValue>
                    fL8/oWpfkdKvxZ736BOymg6ulgI=
                    </ds:DigestValue>
                </ds:Reference>
            </ds:SignedInfo>
        <ds:SignatureValue>
            ...hvpl2L8=
            </ds:SignatureValue>
        <ds:KeyInfo>
            <wsse:SecurityTokenReference>
```

```
                        <wsse:Reference
                            URI="#CertID-2ba3d2ee-9679-1e01-2735"
                            ValueType="http://.../...x509-token-profile-1.0#X509v3">
                            </wsse:Reference>
                        </wsse:SecurityTokenReference>
                    </ds:KeyInfo>
                </ds:Signature>
            </wsse:Security>
        </soapenv:Header>
    <soapenv:Body>
        <ns1:getContentList
            xmlns:ns1="http://feupvod/schema">
            <ns1:search>
                ALL
                </ns1:search>
            </ns1:getContentList>
        </soapenv:Body>
 </soapenv:Envelope>


Hypertext Transfer Protocol
    HTTP/1.1 200 OK\r\n
    Date: Tue Jun 14 12:26:38 2011 GMT\r\n
    Server: Axis2C/1.6.0 (Simple Axis2 HTTP Server)\r\n
    Content-Type: text/xml;charset=UTF-8\r\n
    Content-Length: 3752\r\n
    \r\n
eXtensible Markup Language
 <soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Header
        xmlns:wsa="http://www.w3.org/2005/08/addressing">
        <wsa:Action>
            getContentList
            </wsa:Action>
        <wsa:From>
            <wsa:Address>
                http://localhost:9090/axis2/services/user
                </wsa:Address>
            </wsa:From>
        <wsa:MessageID>
            urn:uuid:2bbf15e0-9679-1e01-2f00-00236c7d7b79
            </wsa:MessageID>
        <wsa:RelatesTo
            wsa:RelationshipType="http://www.w3.org/2005/08/addressing/reply">
            urn:uuid:2b9ddaa6-9679-1e01-2734-000000000000
            </wsa:RelatesTo>
        <wsse:Security
            xmlns:wsse="http://.../oasis-200401-wss-wssecurity-secext-1.0.xsd"
            soapenv:mustUnderstand="1">
            </wsse:Security>
        </soapenv:Header>
    <soapenv:Body>
        <ns1:getContentListResponse
            xmlns:ns1="http://wso2/wsf/cpp/namespace1">
            <movieinfo>
                <movieid>
```

```
                    1
                    </movieid>
                <moviename>
                    Finding Nemo
                    </moviename>
                <year>
                    2003
                    </year>
                <imgurl>
                    http://ia.media-imdb.com/images/M/..._V1._SY317_CR0,0,214,317_.jpg
                    </imgurl>
                </movieinfo>
            <movieinfo>
                <movieid>
                    2
                    </movieid>
                <moviename>
                    Bruce Almighty
                    </moviename>
                <year>
                    2003
                    </year>
                <imgurl>
                    http://ia.media-imdb.com/images/M/..._V1._SY317_.jpg
                    </imgurl>
                </movieinfo>
             ...
            </ns1:getContentListResponse>
        </soapenv:Body>
  </soapenv:Envelope>
Hypertext Transfer Protocol
```

A.9.4 WSF/C++ log

```
  Starting Axis2 HTTP server....
  Apache Axis2/C version in use : 1.6.0
  Server port : 9090
  Repo location : ../
  Read Timeout : 60000 ms
 ...
 Module addressing found in axis2.xml
 ..//lib/libaxis2_http_sender.so shared lib loaded successfully
 ..//lib/libaxis2_http_receiver.so shared lib loaded successfully
 ...
 ..//modules/addressing/libaxis2_mod_addr.so shared lib loaded successfully
 ...
 ..//modules/logging/libaxis2_mod_log.so shared lib loaded successfully
 ...
 ..//modules/rahas/libmod_rahas.so shared lib loaded successfully
 ...
 ..//modules/rampart/libmod_rampart.so shared lib loaded successfully
 mep_url:http://www.w3.org/2004/08/wsdl/in-only
 ...
 ..//modules/sandesha2/libsandesha2.so shared lib loaded successfully
 ..//lib/libsavan_msgreceivers.so shared lib loaded successfully
 ...
 ..//modules/savan/libmod_savan.so shared lib loaded successfully
```

```
..//lib/libwsf_cpp_msg_recv.so shared lib loaded successfully
Module rampart will be engaged to admin
Service name is : admin
Add handler RampartInHandler to phase Security
Add handler RampartOutHandler to phase Security
Add handler RampartOutHandler to phase MessageOut
Add handler RampartInHandler to phase Security
Add handler RampartOutHandler to phase Security
Add handler RampartOutHandler to phase MessageOut
DLL path is : ..//services/client/libclient.so
..//lib/libwsf_cpp_msg_recv.so shared lib loaded successfully
DLL path is : ..//services/user/libuser.so
Module rampart will be engaged to user
Service name is : user
Add handler RampartInHandler to phase Security
Add handler RampartOutHandler to phase Security
Add handler RampartOutHandler to phase MessageOut
DLL path is : ..//services/user_rest/libuser.so
..//lib/libwsf_cpp_msg_recv.so shared lib loaded successfully
Add handler AddressingInHandler to phase Transport
svc name is:admin
Service name is : admin
Operation name is : addServer
Add handler AddressingOutHandler to phase MessageOut
phase_resolver.c(1113) Operation name is : editContent
Add handler AddressingOutHandler to phase MessageOut
...
svc name is:user_rest
Service name is : user_rest
...
svc name is:client
Service name is : client
...
svc name is:user
Service name is : user
...
[rampart][rampart_mod] rampart_mod initialized
[rahas]Rahas module initialized
Starting HTTP server thread
Client HTTP version HTTP/1.1
Identified soap version is soap11
Invoke the handler request_uri_based_dispatcher within the phase Transport
Checking for service using target endpoint address : http://127.0.0.1:9090/axis2/services/user
Service found using target endpoint address
Invoke the handler AddressingInHandler within the phase Transport
 Starting addressing in handler
Invoke the handler addressing_based_dispatcher within the phase Transport
Checking for operation using WSA Action : getContent
Operation found using WSA Action
...
[rampart]Trying to load module /etc/wso2/wsfcpp/services/user/libauthn.so
/etc/wso2/wsfcpp/services/user/libauthn.so shared lib loaded successfully
[rampart]Successfully loaded module /etc/wso2/wsfcpp/services/user/libauthn.so
[rampart]Processing security header in Strict layout
[rampart]Processing security header element BinarySecurityToken
[rampart]Processing security header element UsernameToken
 [rampart]Validating UsernameToken
 ...
 [rampart]Password authentication using AUTH MODULE
 [rampart]User authenticated
 [rampart]Set SPR_UT_Checked in Security Processed Results of message context
 [rampart]Validating UsernameToken SUCCESS
 [rampart]Processing security header element Signature
 [oxs][xml_sig] Verifying signature part #SigID-607fc1a2-967a-1e01-2736
 [rampart][c14n-OutPut] is

<wsse:UsernameToken ...>
<wsse:Username>admin</wsse:Username>...
<wsse:Nonce>nQFBE5Rkd3zFaImX9Kwwz4OLPGlFRaf6</wsse:Nonce>
<wsu:Created>2011-06-14T11:35:17.027Z</wsu:Created>
</wsse:UsernameToken>

 [oxs][xml_sig] Digest verification success for node Id= #SigID-607fc1a2-967a-1e01-2736
 [oxs][xml_sig] Digests verification SUCCESS
 [rampart] C14N (verif1)= ...
 [oxs][xml_sig] C14N (verif2)=...

 [oxs][sign_ctx] Public key is not available directly. Extracting the certificate
 [openssl][sig] Signature verification SUCCESS
 [oxs][sig] Signature verification SUCCESS
 [rampart][rampart_context] Nothing to sign outside Secyrity header.
 ...
 [rampart]Set SPR_Sig_Val in Security Processed Results of message context
 Security header processing done
 [rampart][shp] Replay detection is not specified. Nothing to do
 [rampart][rampart_context] Nothing to encrypt outside Secyrity header.
 ..//services/user/libuser.so shared lib loaded successfully
 user.cpp(434) [User Services (SOAP MODE)] Operation getContent
 user.cpp(134) [getContent] Initialized getContent
 user.cpp(138) [getContent] Running from path /etc/wso2/wsfcpp/bin
 user.cpp(165) [getContent] Opening database
 user.cpp(175) [getContent] Searching Movie
 user.cpp(179) [getContent] Sending Movie
 Invoke the handler AddressingOutHandler within the phase MessageOut
 Starting addressing out handler
 Invoke the handler RampartOutHandler within the phase Security
 [rampart][shb] Asymmetric Binding.
 [rampart][rampart_context] Nothing to sign outside Secyrity header.
 [rampart]Checking node Header for Timestamp
 [rampart]Checking node Action for Timestamp
...
 [rampart][rampart_context] Nothing to encrypt outside Secyrity header.
 [rampart][rampart_encryption] No parts specified or specified parts can't be found for encryprion.
 [rampart]Checking node Security for Signature
 [rampart]Checking node Security for EncryptedKey
 Request processed in 0.016 seconds
 Request served successfully
 Received signal SIGINT. Server shutting down
 Terminating HTTP server thread
 Terminating HTTP server thread.
 Successfully terminated  HTTP server thread
 Shutdown complete ...
 Service name :admin
 Service name :client
Service name :user
Service name :user_rest
[rampart][rampart_mod] rampart_mod shutdown
[rahas] Rahas module shutdown
```

Appendix B

VLC

B.1 Changelog

```
commit 57dda7ffd68f3d4052ea403d9536eb8f817ac30b
Author: Jean-Paul Saman <jean-paul.saman@m2x.nl>
Date:   Thu Nov 20 14:33:37 2008 +0100

    Remove FORWARD_S and BACKWARD_S from input state.

    The input core does separate playing states for forward or backward
    direction from the playing state PLAYING_S. If one wants to know in
    what direction VLC is playing, then he needs to look at the sign value
    of the "rate" value. Backward playing direction has a negative "rate" value.
    Forward playing direction has a positive one.

commit 75adef44b044b9b81e593e74db857d6b2a7e4eaa
Author: Jean-Paul Saman <jean-paul.saman@m2x.nl>
Date:   Thu Nov 20 14:19:37 2008 +0100

    Signal can_rewind for use by user interfaces.

commit f76cff41026a8c812644a57db33c4d2d846cf55a
Author: Jean-Paul Saman <jean-paul.saman@m2x.nl>
Date:   Wed Nov 19 16:07:34 2008 +0100

    Allowing for rate < 0 enables rewind playback for eg: RTSP streams.

    Don’t allow rate < 0 when p_input->p->input.b_rescale_ts is true.
    Setting b_rescale_ts to true says to vlc, rescale the timestamp.
    Setting it to false is like saying, I can completely handle the rate,
    just acknowledge the fact that the rate is not the default one.

commit 48fd6e792fbb1552d6a7f72a795b998ccebb871a
Author: Jean-Paul Saman <jean-paul.saman@m2x.nl>
Date:   Wed Nov 19 15:45:19 2008 +0100

    (live555) RTSP fastforward works from Qt4 interface.
```

B.2 VLC Revision

```
diff --git a/vlc-1.1.9_2/src/control/media_player.c b/vlc-1.1.9/src/control/media_player.c
index c8193e4..647e5ae 100644
--- a/vlc-1.1.9_2/src/control/media_player.c
+++ b/vlc-1.1.9/src/control/media_player.c
@@ -1148,19 +1148,24 @@ int libvlc_media_player_will_play( libvlc_media_player_t *p_mi )

 int libvlc_media_player_set_rate( libvlc_media_player_t *p_mi, float rate )
 {
-    if (rate < 0.)
+
+    input_thread_t *p_input_thread = libvlc_get_input_thread ( p_mi );
+    bool b_rewindable;
+
+    if( !p_input_thread )
+        return 0;
+    b_rewindable = var_GetBool( p_input_thread, "can-rewind" );
+
+    if (rate < 0. && !b_rewindable)
     {
         libvlc_printerr ("Playing backward not supported");
         return -1;
     }

     var_SetFloat (p_mi, "rate", rate);
-
-    input_thread_t *p_input_thread = libvlc_get_input_thread ( p_mi );
-    if( !p_input_thread )
-        return 0;
     var_SetFloat( p_input_thread, "rate", rate );
     vlc_object_release( p_input_thread );
+
     return 0;
 }
```

Appendix C

Experimental Results

C.1 Media Server Results

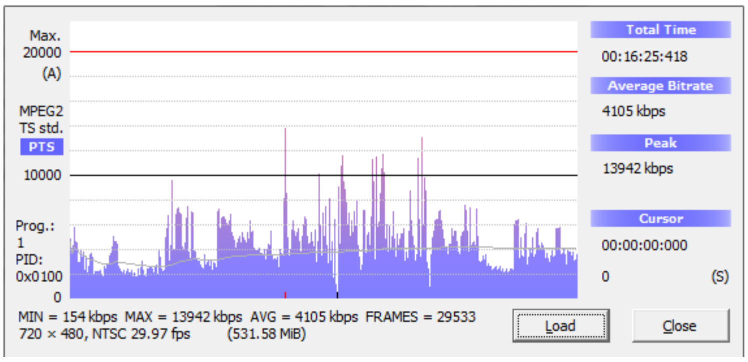
C.1.1 Media Files

Table C.1: Used Software

Application	Software
Computer Statistics	sar
Video Statistics	FFMpeg
Bitrate Chart	Bitrate Viewer

Table C.2: Media Files composition

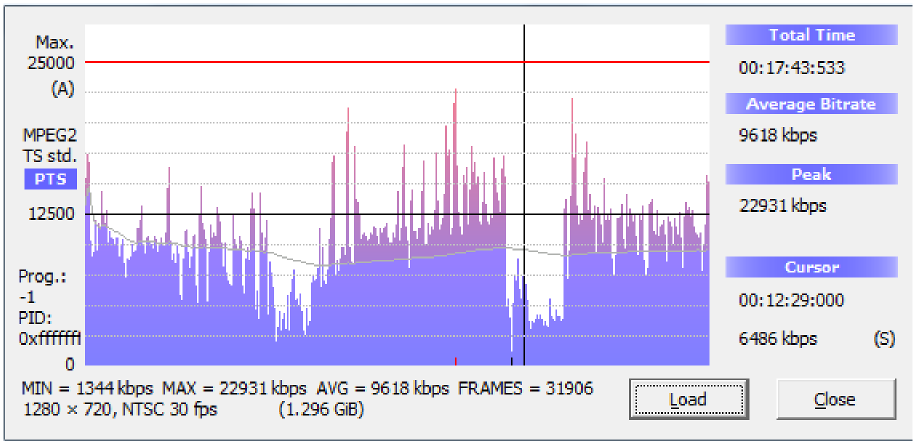
Definition	Size (MB)	Bit rate (kbps)	Time
SD	532	4525	16:43.5
HD	1 327	10467	17:43.5



Total Time: 00:16:25:418
Average Bitrate: 4105 kbps
Peak: 13942 kbps
Cursor: 00:00:00:000
0 (S)

```
MIN = 154 kbps  MAX = 13942 kbps  AVG = 4105 kbps  FRAMES = 29533
720 × 480, NTSC 29.97 fps        (531.58 MiB)
```

Figure C.1: SD video file bitrate average



```
Total Time
00:17:43:533
Average Bitrate
9618 kbps
Peak
22931 kbps
Cursor
00:12:29:000
6486 kbps          (S)
MIN = 1344 kbps  MAX = 22931 kbps  AVG = 9618 kbps  FRAMES = 31906
1280 × 720, NTSC 30 fps            (1.296 GiB)
```

Figure C.2: HD video file bitrate average

C.1.2 Single client - One Thread

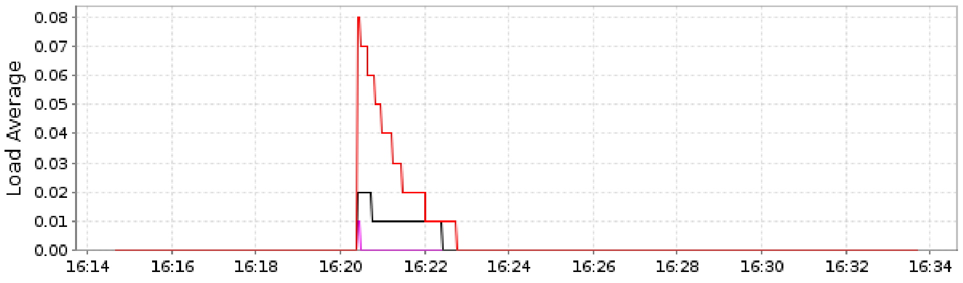


Figure C.3: Load - single client

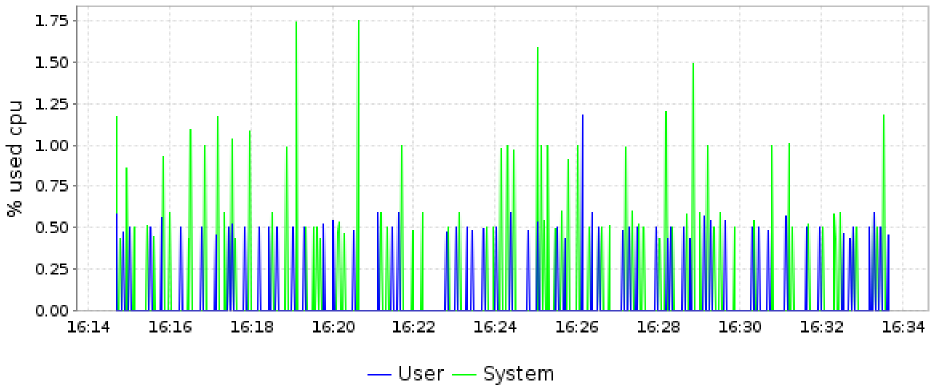


Figure C.4: CPU 0 Usage - single client

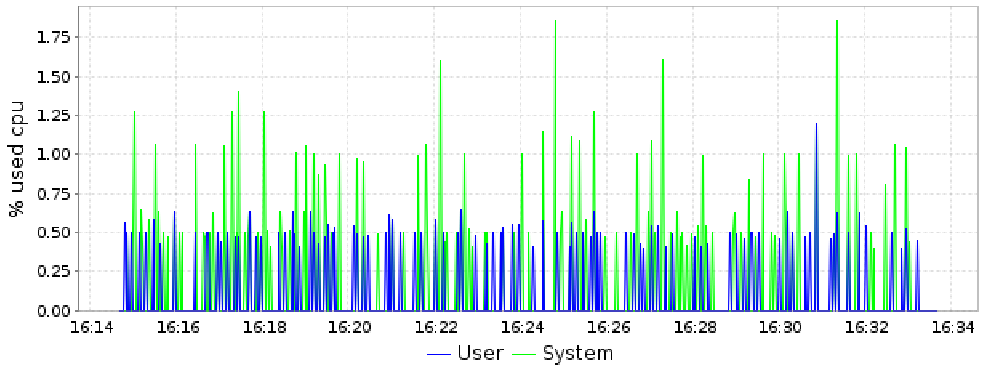


Figure C.5: CPU 1 Usage - single client

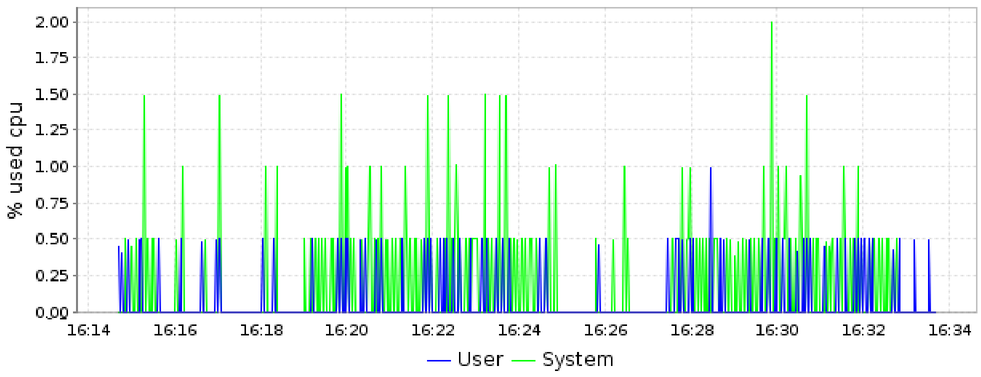


Figure C.6: CPU 2 Usage - single client

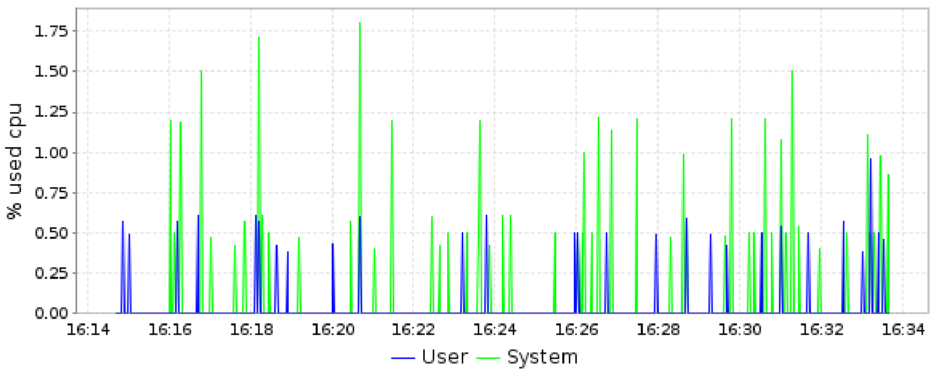


Figure C.7: CPU 3 Usage - single client

C.1.3 68 clients - One thread

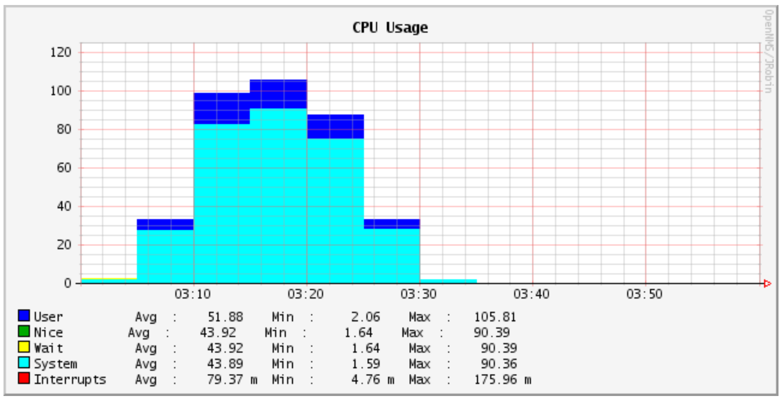


Figure C.8: CPU Usage - 68 Clients

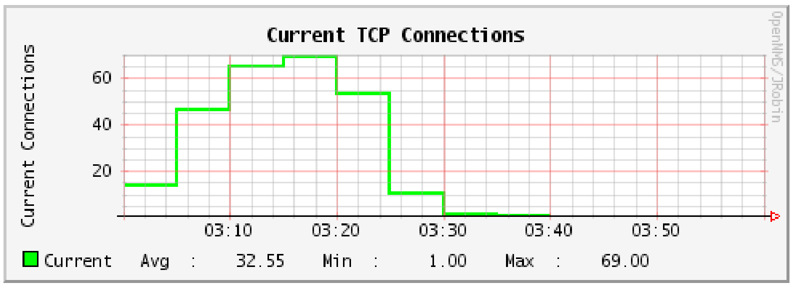


Figure C.9: TCP Connections - 68 Clients

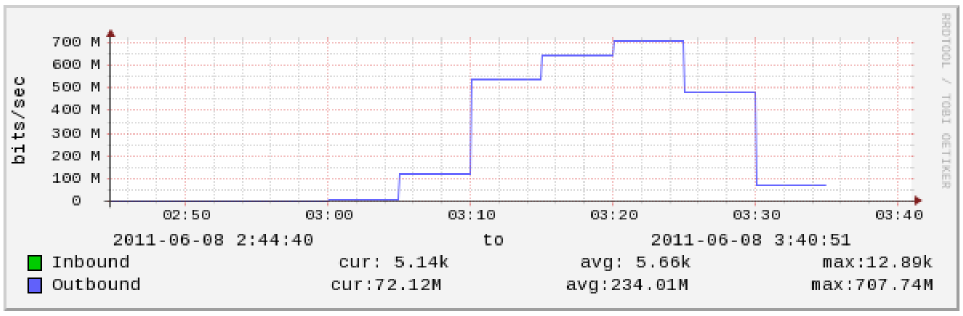


Figure C.10: Throughput - 68 Clients

C.1.4 60 clients - Two threads

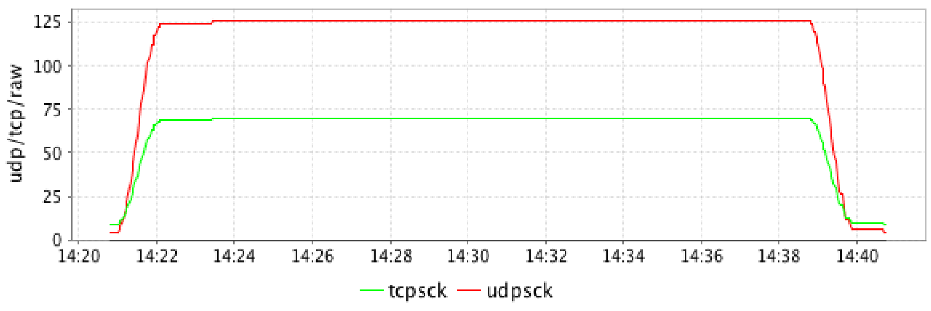


Figure C.11: Open Connections - 60 clients

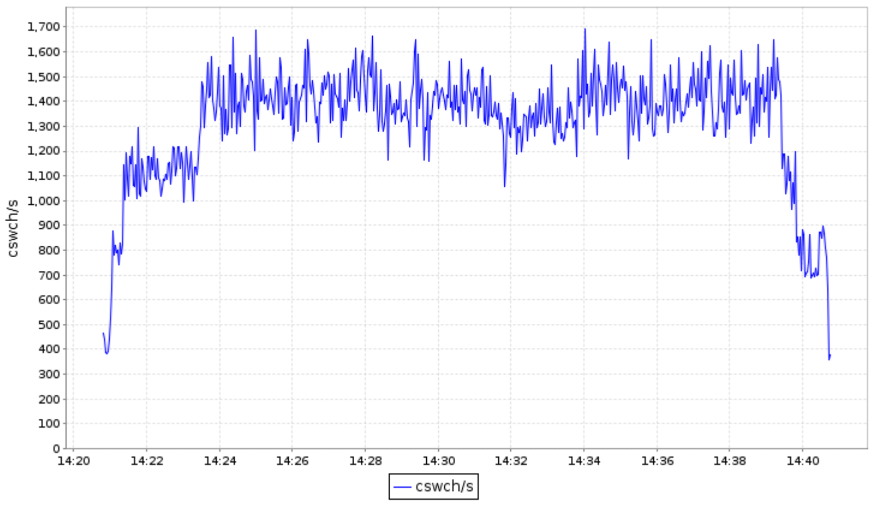


Figure C.12: Contexts - 60 clients

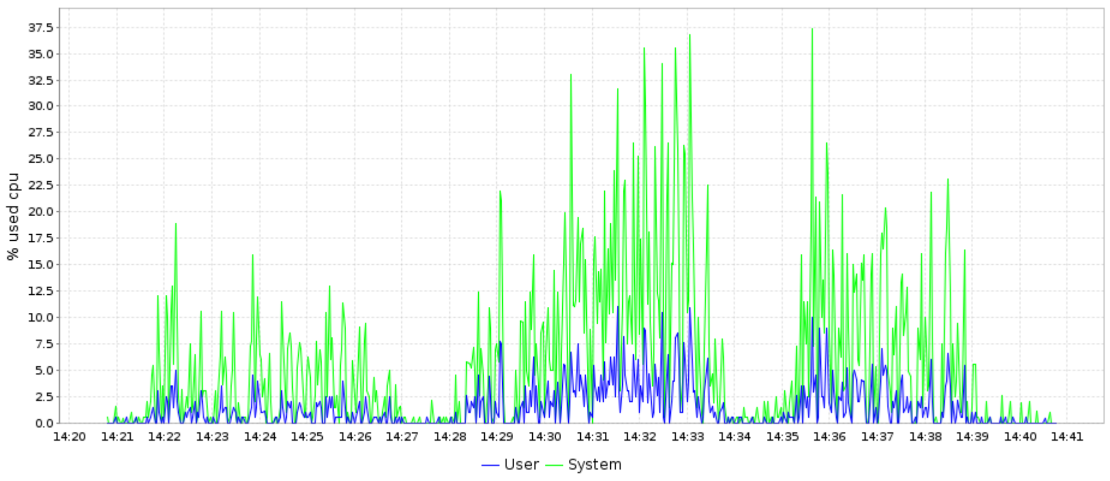


Figure C.13: CPU 0 Usage - 60 clients

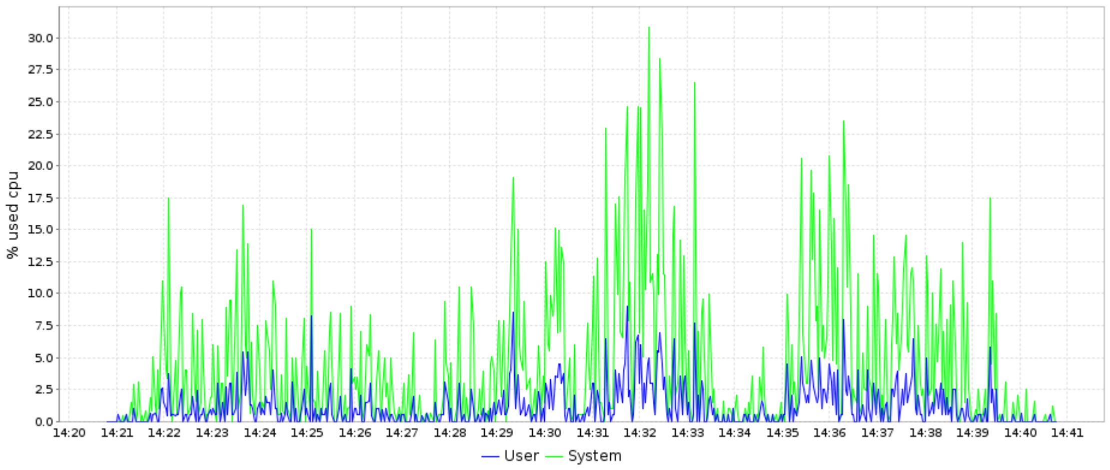


Figure C.14: CPU 1 Usage - 60 clients

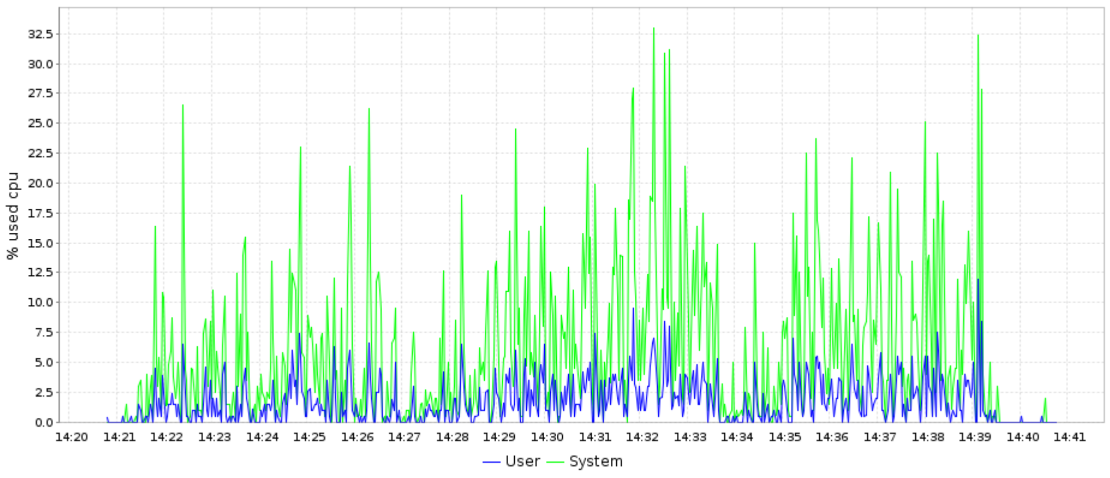


Figure C.15: CPU 2 Usage - 60 clients

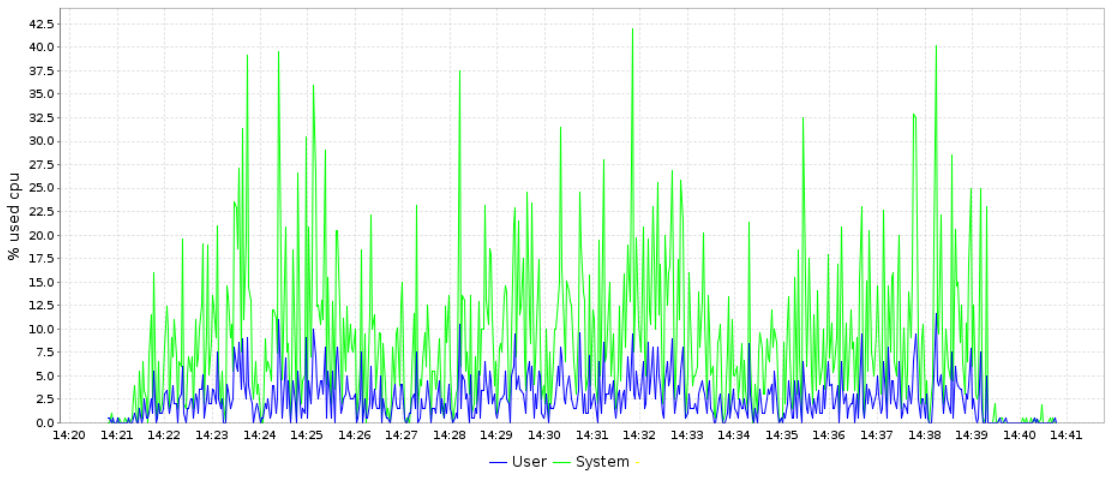


Figure C.16: CPU 3 Usage - 60 clients

C.1.5 78 clients - Two Threads

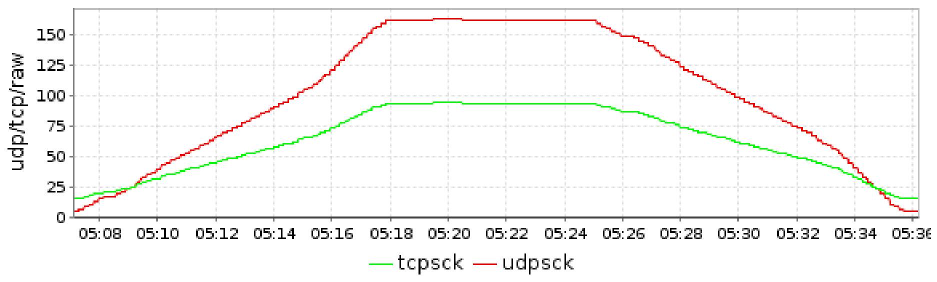


Figure C.17: Open Connections - 78 clients

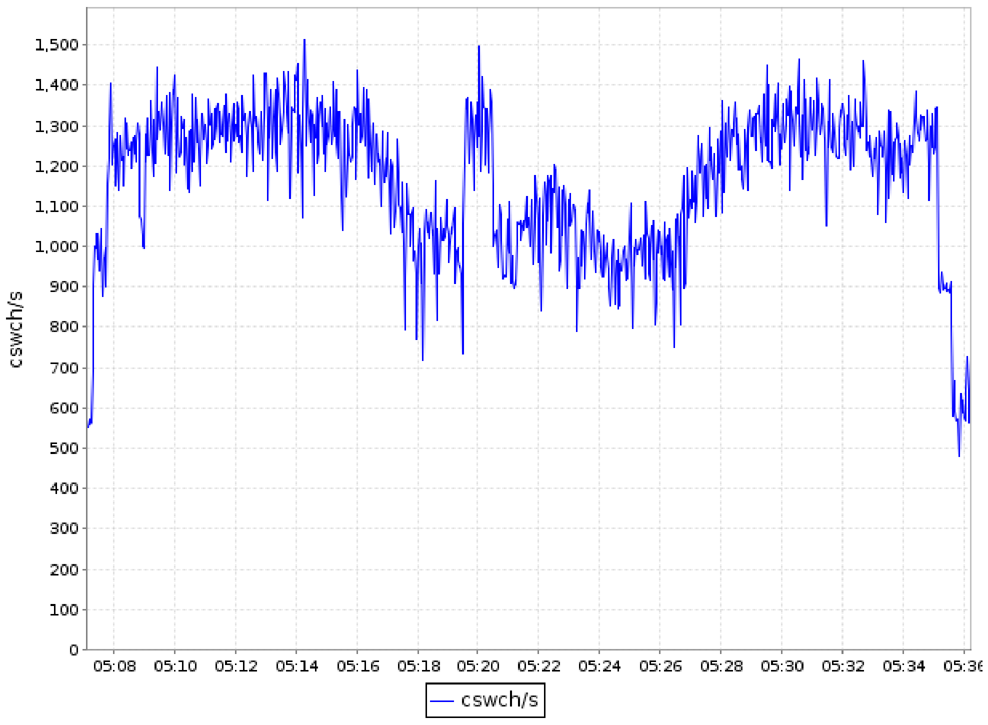


Figure C.18: Contexts - 78 clients

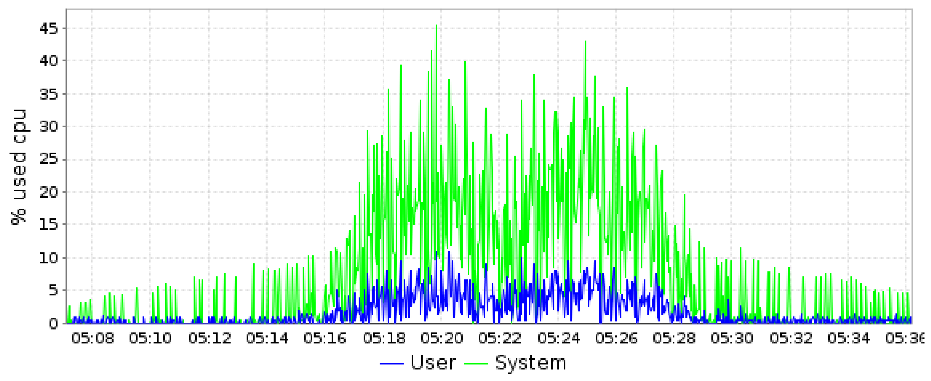


Figure C.19: CPU 0 Usage - 78 clients

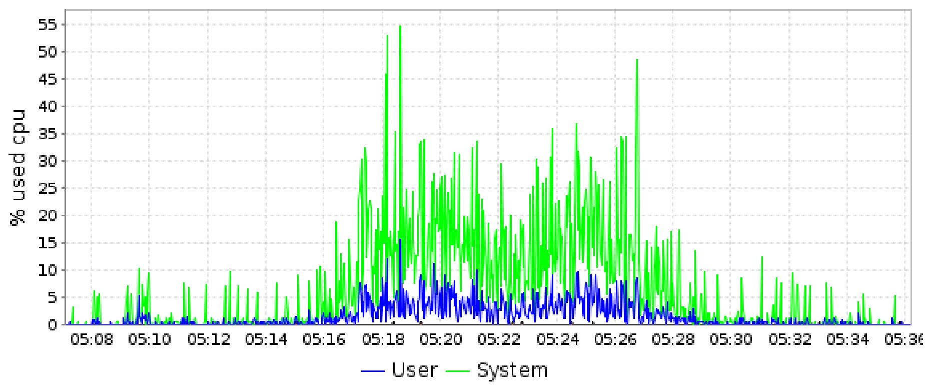


Figure C.20: CPU 1 Usage - 78 clients

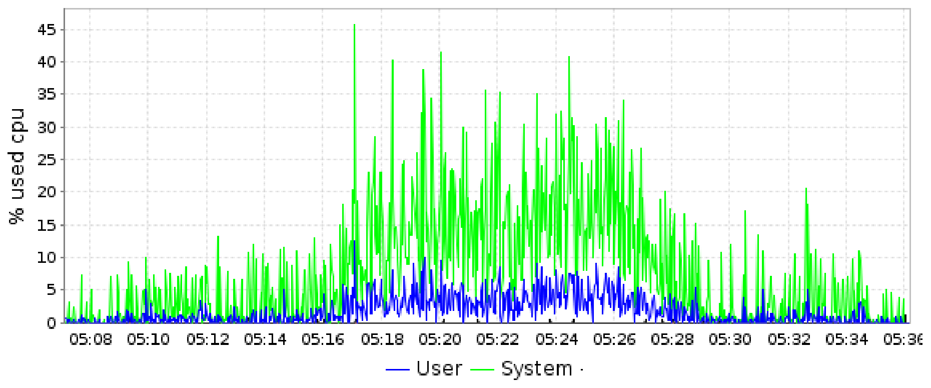


Figure C.21: CPU 2 Usage - 78 clients

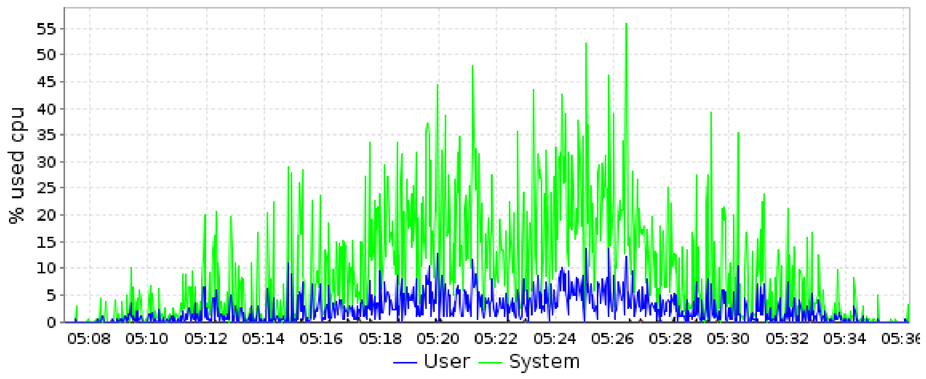


Figure C.22: CPU 3 Usage - 78 clients

C.1.6 168 clients - Four Threads

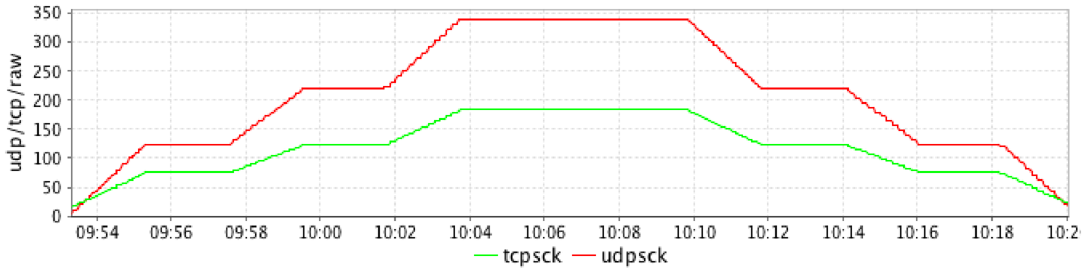


Figure C.23: Open Connections - 168 clients

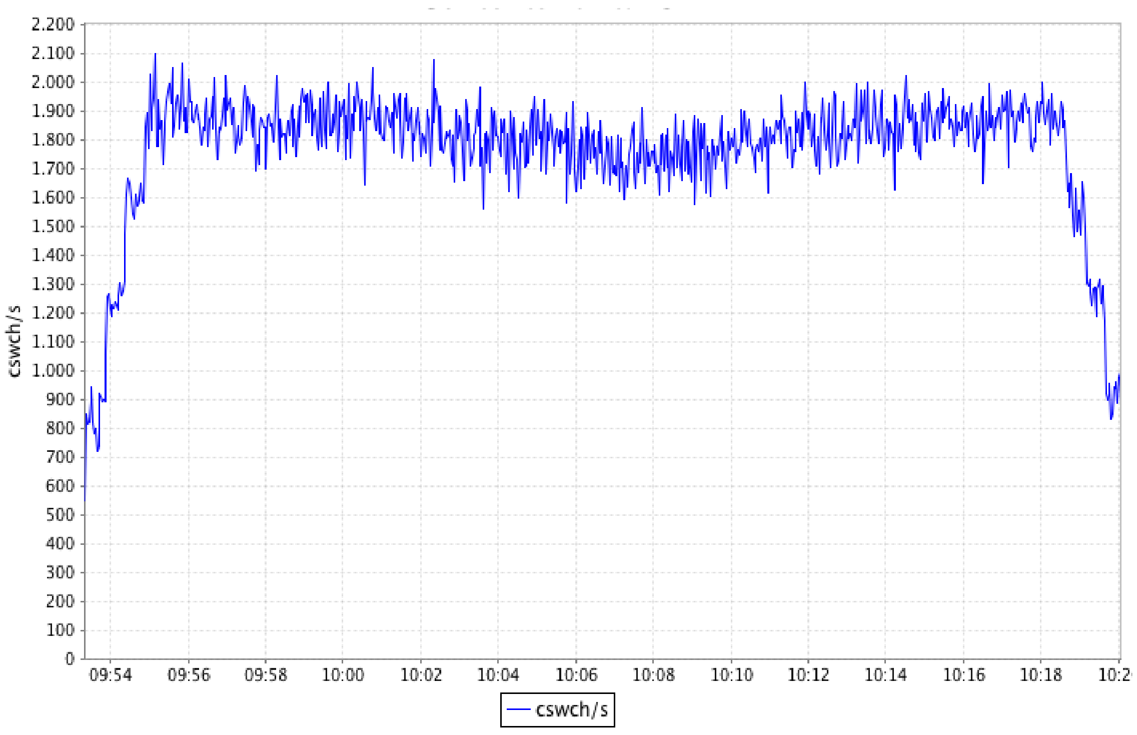


Figure C.24: Contexts - 168 clients

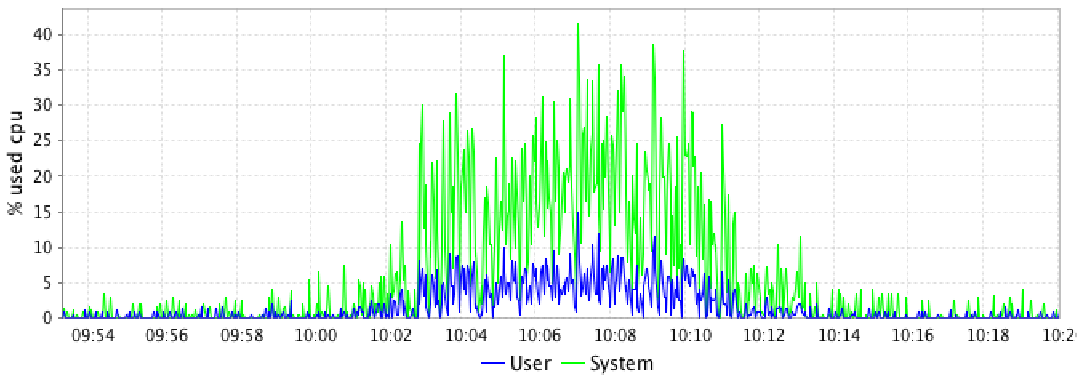


Figure C.25: CPU 0 Usage - 168 clients

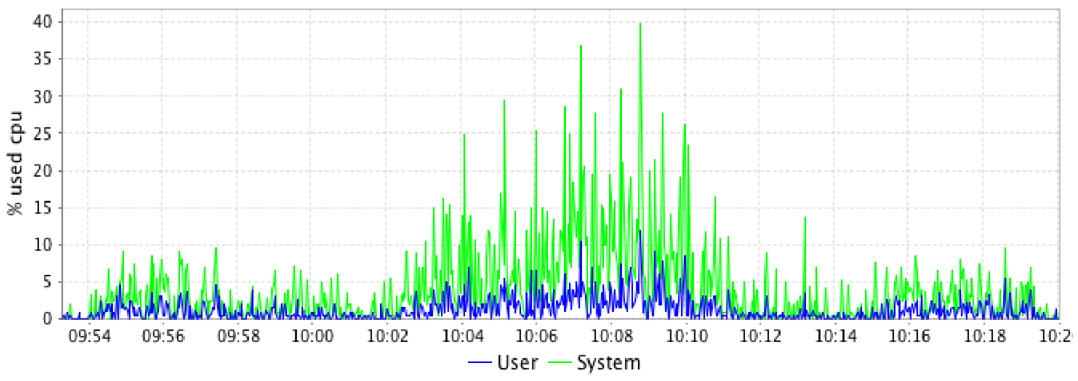


Figure C.26: CPU 1 Usage - 168 clients

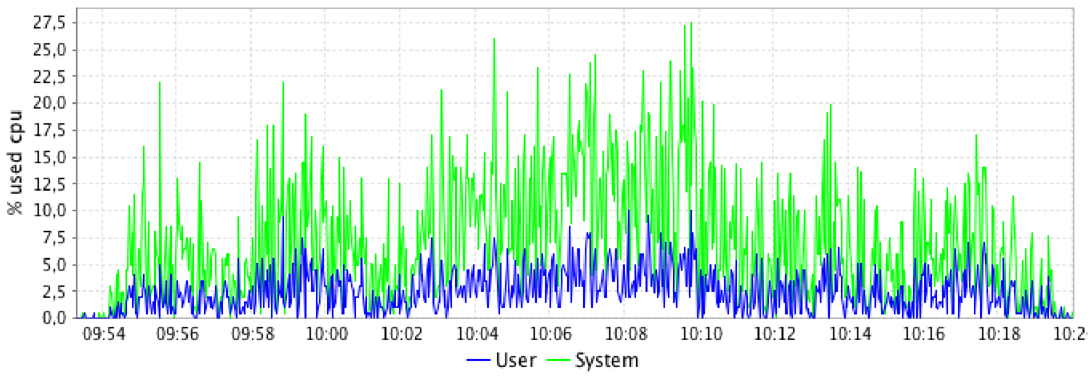


Figure C.27: CPU 2 Usage - 168 clients

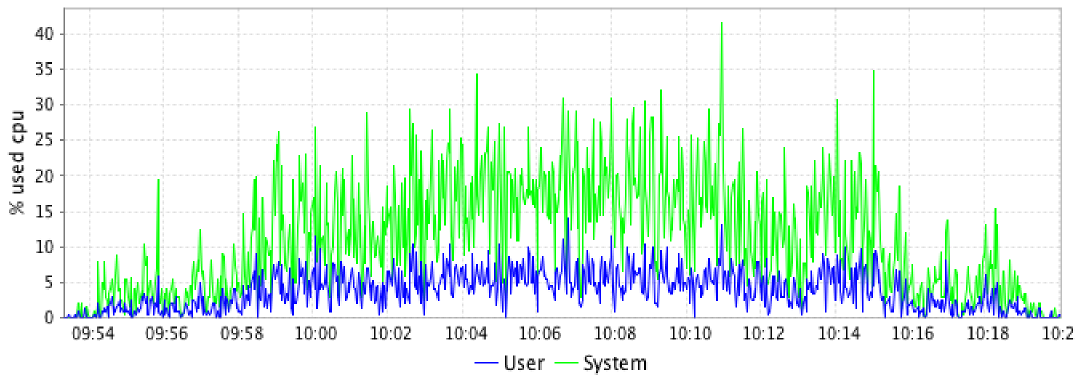


Figure C.28: CPU 3 Usage - 168 clients

C.1.7 Private Network

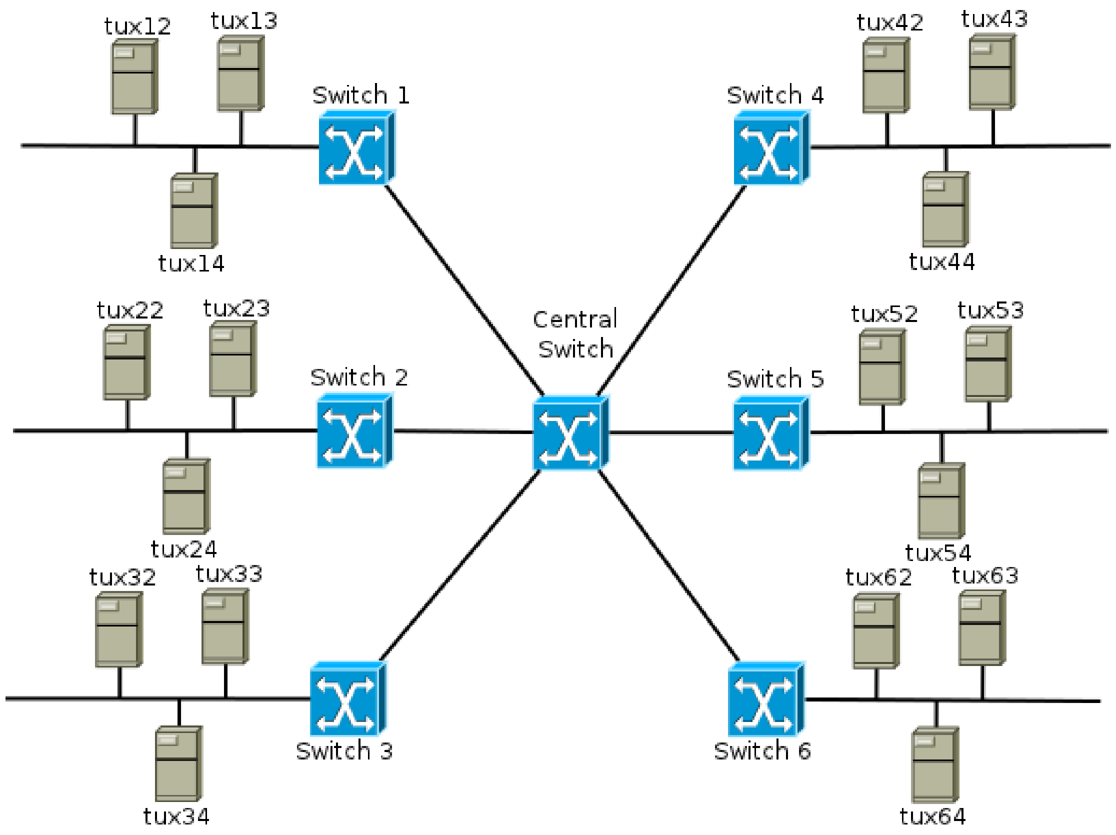


Figure C.29: Private Network

C.1.8 Bash script Example

```
sleep 240
openRTSP -V rtsp://172.16.1.54/Chuck_480.ts &
...
sleep 10
openRTSP -V rtsp://172.16.1.54:8554/Chuck_480.ts &
...
sleep 10
openRTSP -V rtsp://172.16.1.54:9554/Chuck_480.ts &
...
sleep 10
openRTSP -V rtsp://172.16.1.54:7554/Chuck_480.ts
```

References

[1] CDN Comparison. What is a cdn? March 2011. Available from http://cdn-comparison.com/tag/content-delivery-network/.

[2] Iain Richardson. Overview of h.264 / avc, 2008. Available from http://www.vcodex.com/h264overview.html.

[3] Seth Thomas Miller. Group of pictures, October 2009. Available from http://www.spiritalchemy.com/blog/feedback-and-levels-of-description.

[4] Xiph.org. Ogg bitstream overview. Available from http://www.xiph.org/ogg/doc/oggstream.html.

[5] Youtube Team. Thanks, youtube community, for two big gifts on our sixth birthday!, May 2011. Available from http://youtube-global.blogspot.com/2011/05/thanks-youtube-community-for-two-big.html.

[6] Inc Netflix. Unlimited movies & tv episodes, 2011. Available from https://www.netflix.com/.

[7] Amazon. Amazon prime, 2011. Available from http://www.amazon.com/gp/help/customer/display.html?ie=UTF8&nodeId=13819211.

[8] Free Foundation Software. Gnu general public license, June 2007. Available from http://www.gnu.org/licenses/gpl.html.

[9] XVIth Plenary Assembly Dubrovnik. Encoding parameters of digital television for studios, 1986.

[10] ANACOM. Serviço de acesso à internet - 4º trimestre de 2010, March 2011. Available from http://www.anacom.pt/render.jsp?contentId=1071970.

[11] MPEG. Moving picture experts group, 2011. Available from http://mpeg.chiariglione.org/.

[12] International Organization for Standardization ISO. International standards for business, government and society, 2011. Available from http://www.iso.org.

[13] United Nations. International telecommunication union. Available from http://www.itu.int.

[14] A. Luthra, G.J. Sullivan, and T. Wiegand. Introduction to the special issue on the h. 264/avc video coding standard. *Circuits and Systems for Video Technology, IEEE Transactions on*, 13(7):557–559, 2003.

[15] Jason Garret-Glase. Diary of an x264 developer, June 2010. Available from http://x264dev.multimedia.cx/archives/377.

[16] Dr. Dmitriy Kulikov and Alexander Parshin. Mpeg-4 avc/h.264 video codecs comparison 2010 - appendixes. *MSU Graphics & Media Lab (Video Group)*, 2010. Available from http://www.compression.ru/video/codec_comparison/h264_2010/appendixes.html#Appendix_8.

[17] I. Gonçalves, S. Pfeiffer, and C. Montgomery. Ogg media types. *Internet RFC 5334*, September 2008.

[18] J. Postel. Transmission control protocol. *Internet RFC 793*, 1981.

[19] R.J. Postel. User datagram protocol. *Internet RFC 768*, 1980.

[20] H. Schulzrinne, S Casner, R. Frederick, and V. Jacobson. Rtp: A transport protocol for real-time applications. *Internet RFC 3550*, July 2003.

[21] H. Schulzrinne, A. Rao, and R. Lanphier. Real time streaming protocol (rtsp). *Internet RFC 2326*, April 1998.

[22] M. Handley and V. Jacobson. Sdp: Session description protocol. *Internet RFC 2327*, 1998.

[23] J. Postel et al. Internet protocol. Technical report, Internet RFC 791, September 1981.

[24] S. Wenger, MM Hannuksela, T. Stockhammer, M. Westerlund, and D. Singer. Rtp payload format for h. 264 video. *Internet RFC 3984*, 2005.

[25] D. Hoffman, G. Fernando, V. Goyal, and M. Civanlar. Rtp payload format for mpeg1/mpeg2 video. *Internet RFC 2250*, 1998.

[26] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (wsdl) 1.1, March 2001. W3C 2011.

[27] M. Mitra and Y. Lafon. Soap version 1.2 part 0: Primer (second edition), April 2007. W3C 2011.

[28] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker. Web services security: Soap message security 1.1 (ws-security 2004). *OASIS Standard Specification*, 1, 2006.

[29] A. Nadalin, M. Goodner, M. Gudgin, A. Barbir, and H. Granqvist. Ws-securitypolicy 1.2. *OASIS Standard*, 2007.

[30] A. Nadalin, M. Goodner, M. Gudgin, A. Barbir, and H. Granqvist. Ws-secureconversation 1.3. *OASIS Standard*, 1, 2007.

[31] A. Nadalin, M. Goodner, M. Gudgin, A. Barbir, and H. Granqvist. Ws-trust 1.3. *OASIS Standard*, 19:2007, 2007.

[32] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker. Web services security: Saml token profile 1.1. *Technical Specification, OASIS Open, 2006c.[Online]. Available WWW: http://www. oasis-open. org/committees/download. php/16768/wss-v1*, 2007.

[33] D. Box, F. Curbera, et al. Web services addressing (ws-addressing), 2004.

[34] K. Ballinger, D. Ehnebuske, C. Ferris, M. Gudgin, C.K. Liu, M. Nottingham, and P. Yendluri. Basic profile version 1.1. *WS-I Specification*, 8:1–1, 2004.

[35] R. Bilorusets, D. Box, L.F. Cabrera, D. Davis, D. Ferguson, C. Ferris, T. Freund, M.A. Hondo, J. Ibbotson, L. Jin, et al. Web services reliable messaging protocol (ws-reliablemessaging). *EA, Microsoft, IBM and TIBCO Software, h ttp://msdn. microsoft. com/library/enus/dnglobspec/html/ws-reliablemessaging. asp*, 2004.

[36] D. Box, L.F. Cabrera, C. Critchley, F. Curbera, D. Ferguson, A. Geller, S. Graham, D. Hull, G. Kakivaya, A. Lewis, et al. Web services eventing (ws-eventing). *W3C member submission*, 15, 2006.

[37] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[38] I. Hickson. Html5 a vocabulary and associated apis for html and xhtml, May 2011. Available from http://www.w3.org/TR/html5/.

[39] E.J. Etemad. *Cascading Style Sheets (CSS) Snapshot 2010*. W3C 2011, May 2011. Available from http://www.w3.org/TR/css-2010/#css3.

[40] The PHP Group. Php: Hypertext preprocessor, 2011. Available from www.php.net/license.

[41] Apple Inc. Darwin streaming server, May 2008. Available from http://dss.macforge.org.

[42] and VideoLan Organization. Vlc media player, 2011. Available from http://www.videolan.org/vlc.

[43] R. Finlayson. Live555 streaming media, 2009. Available from www.live555.com.

[44] Nokia Corporation. Qt - a cross-platform application and ui framework, 2011. Available from http://qt.nokia.com.

[45] ORACLE. Javafx | rich internet applications development, 2011. Available from http://javafx.com.

[46] D. Richard Hipp. Sqlite: Small. fast. reliable. choose any three. June 2011. Available from http://www.sqlite.org/.

[47] E.A. Young and TJ Hudson. Openssl: The open source toolkit for ssl/tls. June 2011. Available from http://www.openssl.org/.

[48] SQLite. Most widely deployed sql database. Available from http://www.sqlite.org/mostdeployed.html.

[49] Netcraft News. May 2011 web server survey, May 2011. Available from http://news.netcraft.com/archives/2011/05/02/may-2011-web-server-survey.html.

[50] C. MacCárthaigh. Scaling apache 2. x beyond 20,000 concurrent downloads. *ApacheCon EU*, 2005.

[51] ProgrammableWeb.com. New job requirement: Experience building restful apis, June 2010. Available from http://blog.programmableweb.com/2010/06/09/new-job-requirement-experience-building-restful-apis/.

[52] V.K. Singh, S.H. Ankadi, and D. Das. Study of combined effects of zero copy and pre-processing on input/output performance of multimedia on demand streaming server. In *Communication Systems and Networks (COMSNETS), 2011 Third International Conference on*, pages 1–4. IEEE, February 2011.

[53] F. Thouin and M. Coates. Video-on-demand networks: design approaches and future challenges. *Network, IEEE*, 21(2):42–48, 2007.

[54] Akamai Technologies. Akamai : Providing a better internet, 2011. Available from http://www.akamai.com/html/technology/dataviz3.html.

[55] Anevia. Vod server - toucan, 2011. Available from http://www.anevia.com/EN/Products/IPTV-VOD-for-Hospitality/VOD-Server-Toucan.html.

[56] NetUP IPTV Solutions. Video on demand & virtual cinema, 2011. Available from http://www.netup.tv/en-EN/vod-nvod-server.php.