UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València

Development of a live video streaming system using Raspberry Pi and the DASH protocol

BACHELOR'S THESIS

Bachelor's Degree in Telecommunication Technologies and Services Engineering

Author: Eloy Sanchis López

Tutor: Juan Carlos Guerri Cebollada
Co-tutor: Pau Arce Vila

València, September 9, 2019

Resum

L'objectiu del TFG és el desenvolupament d'un sistema prototip per a la captura, transmissió i reproducció de manera selectiva de diferents fluxos de vídeo DASH capturats en directe des de dispositius Raspberry Pi. El TFG inclou treballar amb tecnologies i eines com ara FFMPEG, Raspberry Pi, càmeres, DASH i programació web (HTML, Javascript).

Paraules clau: DASH, streaming, codificació de vídeo, programació web

Resumen

El objetivo del TFG es el desarrollo de un sistema prototipo para la captura, transmisión y reproducción de forma selectiva de diferentes flujos de vídeo DASH capturados en directo desde dispositivos Raspberry Pi. El TFG incluye trabajar con tecnologías y herramientas como FFMPEG, Raspberry Pi, cámaras, DASH y programación web (HTML, Javascript).

Palabras clave: DASH, streaming, codificación de vídeo, programación web

Abstract

The goal of the Bachelor's thesis is the development of a prototype system to capture, transmit and play selectively different DASH video streams that are captured live from Raspberry Pi devices. The Bachelor's thesis includes working with technologies and tools such as FFMPEG, Raspberry Pi, cameras, DASH and web programming (HTML, Javascript).

Keywords: DASH, streaming, video encoding, web programming

Contents

List of Figures

List of Tables

CHAPTER 1
Introduction

The effects the information revolution has exerted on today's societies are hard to ignore. As technological advances become more affordable, the population incorporates them into their lives, and people's habits, attitudes and behaviours change.

Information and communication technologies are today more present than ever before in our lives, and one aspect where this is reflected is the usage of Internet. Internet traffic has rocketed up over the last years and its growth is expected to continue, with the annual global IP traffic predictedly reaching an estimate of 4.8 ZB in 2022 – roughly 3 times more than in 2017 [1]. Video in all of its forms is a very significant contributor to that trend, accounting for 57.69 % of global downstream traffic, as can be seen in figure 1.1 [2]. This situation creates an endless circle where higher bandwidths enable new Internet applications and new forms of communication, and the demand for these drives progress in bandwidth capacity.

One of those technological progresses is video streaming with an adaptive bitrate, that is, to adapt the bitrate of the video transmission – and thus its quality – to the available bandwidth. These systems rely on the idea that viewers would rather watch uninterrupted, low-quality video than a good video that played intermittently due to network conditions. That is an important factor in quality of experience (QoE) [3].

The availability of video streaming has fostered an enormous shift in video consumption habits and patterns. Over the last decade, traditional linear television, brought mainly over the air, cable or satellite, has given way to IPTV, over-the-top services such as Netflix or HBO, video on demand... The advent of social networks has also brought new narratives like transmedia storytelling or interactive video.

With this state of the industry, this Bachelor's thesis devises a web application that implements these ideas to provide interactive, adaptive video streaming generated by simple, low-power devices, with many possible applications ranging from low-scale broadcasting of events to IP video surveillance.

These devices are Raspberry Pi boards fitted with a simple camera that continuously generates video in two different qualities to form an adaptive video stream. From the client side, the interface to the video streams is a web application where every Raspberry Pi providing a video stream is listed and users can select which one they want to watch.

The system is functional thanks to different technologies working together like a set of clockwork gears. The technologies that make up this system will be introduced over the following pages.

Figure 1.1: Left: Global downstream and upstream traffic share by application type
Center: Downstream traffic share by application in Europe, Middle East and Asia
Right: Upstream traffic share by application in Europe, Middle East and Asia

1.1 Overview of streaming protocols and techniques

As was mentioned, the increase in available bandwidth paved the way for video streaming. New communication protocols were developed around the turn of the century in order to meet the demand for this service.

Some of the first protocols designed to enable media streaming are the pair RTP/RTCP, together with other related protocols such as RTSP and IGMP, which implement additional functions. They are still in use today, even though they are falling from favour as more modern techniques become commonplace, such as WebRTC, DASH or other streaming protocols over HTTP.

1.1.1. RTP, RTCP and other related protocols

RTP (Real-time Transport Protocol) is a protocol that implements the transmission of multimedia streams through the Internet. It was first published by the IETF (Internet Engineering Task Force) in 1996 as RFC 1889 [4], which was obsoleted by RFC 3550 [5] in 2003. Despite its name, its place in the protocol stack is not the transport layer, but the application layer, which means it is an end-to-end protocol that is agnostic to the inner working of transmission networks.

As in many other protocols, RFC 3550 defines a header structure for RTP (figure 1.2) divided in fields of a certain length which carry the necessary information for the protocol to work and sometimes convey a specific meaning depending on the value. The header is then followed by the payload.

The RTP header has the following format:

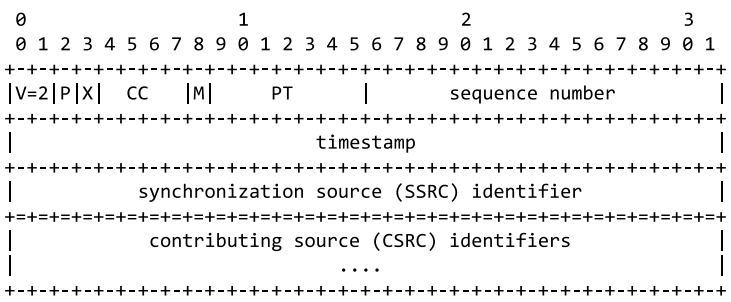

Figure 1.2: Definition of the header fields of RTP

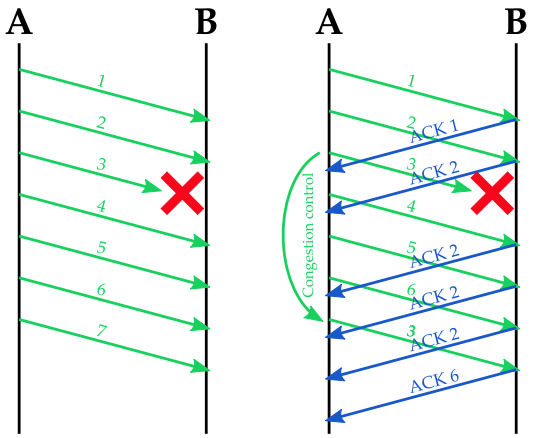

Figure 1.3: Transmission process of UDP (left) and TCP (right)

The purpose of RTP is barely the transmission of mutimedia streams, which includes stream identification, packet ordering by use of sequence numbers, inter-stream synchronization using timestamps, and not much more. RTP is designed to transport the multimedia streams without regard to the codec used to encode them or other video parameters. To allow this, a description of relevant options is interchanged at the beginning of an RTP session using the SDP protocol.

RTP is also indifferent to the particular protocol used in the transport layer, but it is mainly used together with UDP. UDP just sends data packets and does not control anything else. That makes it a very lightweight protocol that usually delivers packets with little delay at the cost of not guaranteeing packet delivery at all – it is in fact incapable of delivering packets in adverse conditions such as a congested network. But that is why it is often preferred over TCP for transport of media – audiovisual content allows for some packets to not be received, because they can go unnoticed, but it requires that those that arrive do so in time. Because playback is an ordered process, a packet arriving too late is completely useless [6]. A comparison of the transmission process in UDP and TCP is shown in figure 1.3.

Adaptation to network conditions is necessary in order to deliver a good quality of experiece (QoE) – at least, as good as possible. If a multimedia stream is sent with no adaptation to the available bandwidth, a scenario of congestion will cause a high packet loss, which will be translated into interrupted or intermittent playback in the client side. It is widely known after many QoE studies that viewers deem low-quality video to be

better (or less bad) than interrupted or intermittent video [3], so a mechanism must be put in place to regulate video quality according to network conditions.

Because RTP runs on UDP and UDP cannot handle congestion like TCP does, the necessary mechanisms to do it are implemented end to end in the application layer. RTP is capable of transmitting media with varying quality (and thus varying bitrate), but it is a unidirectional protocol. Because the client receiving the multimedia stream is aware of the state of the network better than the server, the RTP server needs some sort of feedback from the client. That is how RTCP was born.

RTCP (RTP Control Protocol) is defined as a companion to RTP in RFC 3550 [5], and its purpose is, as its name implies, to carry the necessary information to control RTP sessions. RTCP collects measurements on key parameters of the RTP session, such as packet loss, round-trip time, jitter... These measurements are taken into account by the media source in order to evaluate the state of the network and calculate, according to that, whether to increase, decrease or maintain the bitrate of the media stream in order to deliver the maximum quality while avoiding playback interruptions.

Besides that, RTCP also serves other purposes, such as the identification of the media sources or the establishment and teardown of an RTP session.

RTCP usually operates on the port number immediately next to RTP, but in certain scenarios both protocols may be multiplexed in the same port. This is not a problem, since RTCP only needs around 5% of RTP's bandwidth.

Besides RTCP, there are other protocols related to RTP, such as RTSP or IGMP. There are two different modalities of video streaming:

- Live streaming refers to a scenario where a video stream is being generated in a certain moment and transmitted on the fly. Users can only view the present moment of the stream.

- Video on demand (VoD) refers to a scenario where a video stream is stored in a server and transmission starts at users' request. Users have freedom to move up and down the video timeline at their will.

Live streaming scenarios usually require multicast transmissions. That is efficient because it avoids sending multiple copies of the same content and makes the system scalable. However, multicast transmissions require routers to know where they have to send the stream. IGMP (Internet Group Management Protocol) allows clients accessing a live stream to inform their local router about that. This router will inform another router, and this operation will take place recursively until a tree is formed with the media source at the root and branches to every client accessing the live stream. IGMP is, of course, also used to modify the tree dynamically or to remove branches. It is worth noting that IGMP is not an application layer protocol, but an Internet layer protocol.

In contrast to that, video on demand requires unicast transmissions because users can move freely up and down the video timeline. RTSP (Real Time Streaming Protocol) allows the client to control playback from the server in order to perform common operations on videos, such as play, pause, stop, fast-forward, and so on. It is transported on TCP and it is very similar to HTTP in its form, albeit with some differences, namely RTSP being a session protocol, RTSP servers generating requests as well or RTSP being just a signalling protocol (i.e. data is carried on RTP, not RTSP).

To sum up, video streaming requires an RTP connection for data transmission and an RTCP connection for quality of service (QoS) control. The signalling protocol is IGMP for live streams and RTSP for video on demand.

1.1.2. RTMP

RTMP (Real-Time Messaging Protocol) was developed by Macromedia (which was later purchased by Adobe) as a proprietary protocol with the aim of streaming video and audio to a Flash player.

RTMP works by packetizing multimedia content and fragmenting those packets, what in combination with the small header size, results in a high efficiency. Several channels can be transmitted for multimedia streams and control data alike. They are then multiplexed and they can be distinguished by a field in the header.

The usage of RTMP is non-existent due to its ties to Flash player, which was later deemed insecure, leading to the discouragement of its usage. The development of DASH also contributed to RTMP's fall.

1.1.3. WebRTC

WebRTC (Web Real-Time Communication) is a JavaScript framework that enables direct communication between browsers in real time [7]. It is highly oriented to interactive real-time media applications such as IP phone calls or video conferences.

The key point of WebRTC is that, in a scenario of communication between two clients (i.e. two web browsers), it removes the need of sending the data from one of the clients to a server and then from the server to the other client, even though a server is still needed for the first contact between the browsers. This reduces the delay and makes the system more scalable. In summary, it replaces the client-server architecture by a peer-to-peer scheme. The new aspect of WebRTC is that it enabled that scheme of communication among between web browsers. The fact that communication takes place among equal entities makes WebRTC more suitable for two-way communication.

The actual data content of the communications can be any type of file, but it is aimed at multimedia interactive communication, both because it features the possibility to specify codec options and because of the use of the UDP transport protocol, which is more suitable for multimedia streams, as explained before.

1.1.4. HTTP-based streaming technologies

While the development of protocols like RTP was good at making the first media streaming systems possible, they have been replaced by more modern technologies that use HTTP to the point that the vast majority of streaming systems today are HTTP-based. The most prominent of them is DASH.

All HTTP-based streaming techniques rely on similar procedures. Multimedia streams are stored in web servers in different qualities, bitrates and formats, and they are split in small fragments. A manifest file is created listing the available files. When a client desires to access the stream, it downloads the manifest in the first place, and then it starts downloading fragments successively by means of regular HTTP GET requests. Because clients know better than any other entity in the network the available bandwith and the network conditions in general, they evaluate the network metrics and, unlike in the RTCP mechanism, *they* decide which quality sould be transmitted. After all, quality selection only depends on the client because it is done by requesting the download of one media fragment or another one.

The reason of the popularity of HTTP-based streaming technologies lies precisely in the implications of its HTTP nature. No special servers are needed – standard web servers

are enough–, CDNs can be used, firewalls and NATs are easy to traverse because port 80 (corresponding to HTTP) is standard... To sum up, most or all existing HTTP machines and procedures can be reused for video streaming. Moreover, because it is the client who decides on quality adaptation, this adaptation can be different for each client, in contrast to live RTP where quality can only be the same for every client, even if network conditions differ greatly among them.

As explained before, UDP has always been considered more suitable for media delivery than TCP. HTTP runs on TCP, and so does HTTP-based streaming, but there seems to be no conflict with TCP congestion control or retransmission delays [8]. Other disadvantages are operational complexity and the need of more storage, but they are not significant enough to outweigh the advantages.

As stated before, there are several HTTP-based streaming technologies and they are similar in concept. The most important when this type of technologies began to be developed were Adobe HTTP Dynamic Streaming (HDS), which ran to serve Flash Player, Apple HTTP Live Streaming (HLS), which only supported MPEG2-TS, and Microsoft Smooth Streaming. However, the clearest reference in the matter of HTTP-based streaming is DASH, which has been fully standardized and is the default option when choosing a technique of this type. DASH is the chosen technology for this project, so it will be covered a little bit more in depth in the next section.

1.2 DASH

DASH (Dynamic Adaptive Streaming over HTTP), or MPEG-DASH, is a media streaming technique based on HTTP with an adaptive bitrate. It is standardized as ISO/IEC 23009-1 [9].

Before DASH was created around 2010, multiple proprietary solutions existed, as explained in the previous section. This resulted in a landscape where interoperability was not possible and multiple parallel solutions had to be adopted and implemented in the same system in order to support all types of HTTP-based streaming. To overcome the hurdles that competing solutions represented for market growth of HTTP-based streaming, several technology companies joined efforts in order to standardize DASH. DASH then replaced a complex situation of competing proprietary solutions with a single open standard. Those companies plus many others then formed DASH-IF (DASH Industry Forum) in order to help bring the standard to real-life business solutions [10].

DASH follows the same scheme as the other HTTP-based streaming techniques – in fact, it takes many elements from them. Media files are stored in a web server in different qualities, bitrates and formats, and divided in small fragments. A manifest file called MPD (Media Presentation Description) is also stored in the server and it lists the URL of each of the fragments. Clients start by downloading the MPD and then they start asking the server for fragments (figure 1.4 [11], [12]). They also run an algorithm that monitors some network metrics and decides what media quality is the best option given the network conditions and device characteristics (figure 1.5). Changes in the quality are performed just by requesting the corresponding fragments.

DASH is not a protocol. The goal of DASH is not to transport the media segment – that is treated just like normal HTTP (web) traffic. It is not an algorithm either; the algorithm of quality adaptation is managed by client implementations such as players and it is outside the scope of the DASH standard. Its goal is rather to define the MPD file, i.e. its structure, its possible values, how it relates to the content being requested and sent... That allows it to be agnostic to, for instance, the specific codec used to encode the

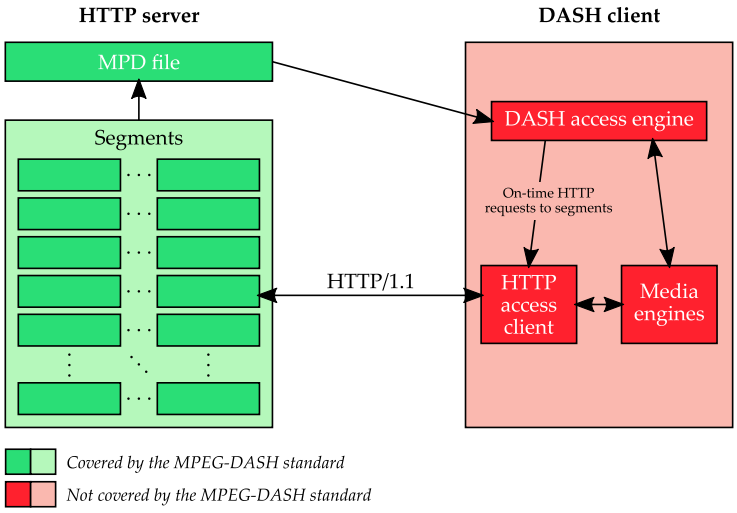

Figure 1.4: DASH architecture

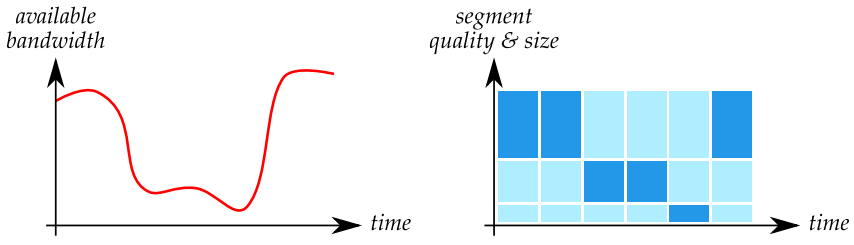

Figure 1.5: DASH quality adaptation according to network conditions

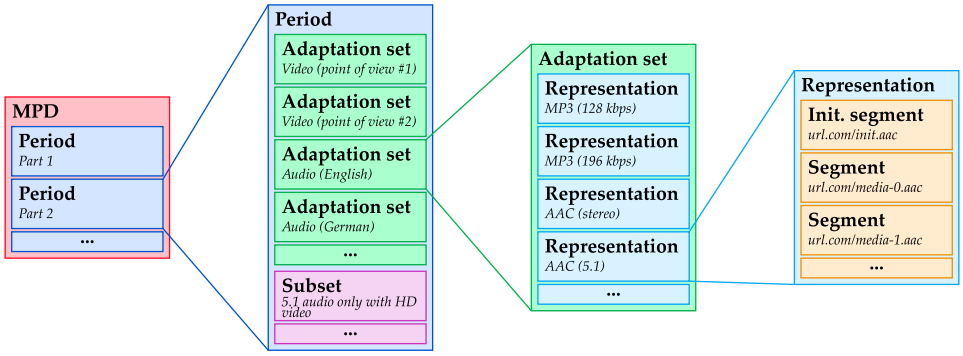

Figure 1.6: DASH MPD hierarchy

media and other aspects of the transmission. DASH also specifies some aspects on media segments.

MPDs are XML files that include the URLs referencing media fragments. They follow a hierarchical structure (figure 1.6 [11], [13]):

- Periods are the topmost level of the hierarchy. There are one or more periods in an MPD, and they represent a temporal part of the complete content. This could refer, for instance, to a chapter or a scene. They are also used to determine where advertisements can be inserted.
- Each period contains one or more adaptation sets. An adaptation set contains multimedia content that differs from that of other adaptation sets. For instance, audio and video belong to different adaptation sets; videos with different points of view belong to different adaptation sets too, as do audios in different languages.
- Each adaptation set is composed of one or more representations. All representations inside the same adaptation set are perceptually and/or logically equivalent, but they differ in aspects like bitrate, codec or other parameters alike.
- Each representation is divided into temporally successive segments – the lowest hierarchy level–, each of which is accessible through a URL or a byte range from a larger file. Segment duration is usually constant in time, and segment start and end is aligned across representations in order to make switching qualities as seamless as possible. Because that requires segments to be independent of each other, they have to start with key (or intra-coded) frames.

  When it comes to DASH segments, size does matter. The choice of short or long segments has implications. Long segments are efficient because they generate less header traffic in proportion to actual data traffic, and the number of files and thus the MPD size is smaller. However, short segments reduce playout delay and make it easier to switch when needed, so they are the recommended option for live stream cases [11].
- At the same hierarchical level as adaptation sets are the so-called subsets. If present, they set rules on what representations can or cannot be selected simultaneously. For example, a subset could force the use of surround audio instead of stereo audio if the selected video resolution is full HD or higher.

Besides that, there are several ways of creating URLs for the different segments in a systematic way (segment referencing schemes) [13]:

- Base URL is used when each representation contains only one file. In this case, segmentation is not performed by dividing the representation into various files, but by assigning to each segment a byte range of the single file. The only existent file just gets a URL.
- Segment list consists of the URL of an initialization segment and a list of the URLs of the other media segments in playback order.
- Segment template is just a single URL with the option to include special variables that will take different values for successive or different segments, such as time, representation, or segment number. This segment referencing scheme is useful for long files or streams with many representations in order to avoid a megabyte-sized MPD that could significantly delay playback.

The MPD contains much more information, such as content availability, resolution, bandwidth, DRM protection schemes [12]... Here is an example of an MPD file [14]:

```
<?xml version="1.0" encoding="UTF-8"?>
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011" profiles="urn:mpeg:dash:profile:isoff-live:2011
    " minBufferTime="PT12S" suggestedPresentationDelay="PT0S" type="static"
    mediaPresentationDuration="PT6S">
  <Period id="0" start="PT0S">
    <AdaptationSet segmentAlignment="true">
      <SegmentTemplate presentationTimeOffset="356038323" timescale="90000" media="http://
          ilab-dst1.philo.com:8000/video/$Bandwidth$/$Time$.ismv" initialization="http://
          ilab-dst1.philo.com:8000/video/$Bandwidth$/init.mp4">
        <SegmentTimeline>
          <S t="356038323" d="270270"/>
        </SegmentTimeline>
      </SegmentTemplate>
      <Representation id="0-video" mimeType="video/mp4" codecs="avc1.4d401f" width="1280"
          height="720" frameRate="30000/1001" sar="1:1" startWithSAP="1" bandwidth="
          2085966"/>
    </AdaptationSet>
  </Period>
</MPD>
```

As can be seen, DASH is a very flexible standard that has made media streaming easier and has helped boost its popularity.

1.3 Shaka Player

As mentioned before, DASH is a standard focused on the MPD and a little bit on the segments, but another element that is needed in the transmission of DASH media is a player capable of interpreting the MPD correctly and perform the appropriate operations to play the media.

There are many media players capable of playing DASH streams, such as dash.js (by DASH-IF), Shaka, Bitmovin, GPAC, Gstreamer, Libdash, VLC or the player of Microsoft Edge. They are all very diverse – some are DASH-specific, some are general, some are open-source, some support DASH features which others do not...

Shaka Player is an open-source JavaScript library developed by Google that enables browsers to play DASH streams, as well as other types of adaptive bitrate streams such as HLS without using Flash at all [15]. It was chosen for this project because of its open nature that would allow modifications if needed, because it provides several adaptation algorithms, and because it is actively maintained by a well-known company like Google.

Shaka Player has been tested and proved to work in all current major browsers on all platforms, namely Chrome, Firefox, Edge and Safari, a notable exception being iOS.

In order to include Shaka Player in a website, the source files must be downloaded to the web server and compiled with the appropriate options, or a pre-compiled version can be downloaded as well. Another option is to link in the `head` section of the web page to the pre-compiled Shaka Player located in Google's CDN, which is the chosen option for this project.

1.4 Raspberry Pi

Raspberry Pi is a series of low-cost, small-sized computers developed with the purpose of encouraging the development of basic computer and electronics projects in schools in order to develop those skills. However, the versatility they offer while having a low computing power has resulted in their being used for more professional projects and even in market solutions, especially in the Internet of Things field, where efficient, low-power devices are required.

A Raspberry Pi device consists of only one board where all components are mounted. A Rasbperry Pi does not provide, on its own, any means to interact with it, so all interaction means are external and they must be connected to the appropriate interfaces. The number and type of these interfaces depends on the specific model of Raspberry Pi, but all of them provide USB ports to connect a mouse, a keyboard and maybe other USB devices, as well as a port to connect a display. Some of them also include an Ethernet port and the newest ones have wireless connectivity through WiFi and Bluetooth. A MicroUSB port is provided to connect a power supply.

Quite interestingly, the hard drive storage is a removable MicroSD card which is not included, and all drive read-write operations are read-write operations on the MicroSD card. Therefore, the binary image of an operating system must be flashed onto the MicroSD card, which is then inserted into the corresponding slot in the Raspberry Pi. After that process, the device can boot up. Raspberry Pi devices lack a power switch, and they boot up automatically when power is received from the power supply.

Many different operating systems can run on Raspberry Pi, some of them having official dedicated versions. The "official" operating system provided by the Raspberry Pi Foundation is Raspbian, a Linux distribution based on Debian developed specially for this device. Third-party operating systems available for download from the Raspberry Pi website include Windows 10 IoT Core, Ubuntu Core, RISC OS or OSMC [16]. There are other operating systems that can run on Raspberry Pi but lack official endorsement, such as FreeBSD, Android Things, CentOS or Kali Linux.

Since the first Raspberry Pi was released in 2012, many models have been in the market. Raspberry Pi model generations are numbered sequentially, and there are usually several "types" of devices inside each generation. B models are the standard devices, whereas A models are more simple and lack some functions in comparison to B models. Zero models are minimal, both in capabilities and in physical size. The specific device used in this project is Raspberry Pi 3 Model B+ (figure 1.7 [17]), which was the latest model available at the moment of beginning the project. However, in June 2019 Raspberry Pi 4 Model B was released.

The most important specifications of Raspberry Pi 3 Model B+ can be seen in table 1.1 [18].

The functionality of Raspberry Pi devices can be extended by means of additional pieces of hardware called HAT (Hardware Atteched on Top) that can be connected to the

Figure 1.7: Raspberry Pi 3 Model B+

Processor	Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4 GHz
Memory	1 GB LPDDR2 SDRAM
Connectivity	2.4 GHz and 5 GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE Gigabit Ethernet over USB 2.0 4 USB 2.0 ports
Multimedia	H.264, MPEG-4 decode (1080p30); H.264 encode (1080p30); OpenGL ES 1.1, 2.0 graphics
Input power	5 V / 2.5 A DC

Table 1.1: Extract of the technical specifications of Raspberry Pi 3 Model B+

Raspberry Pi board. Currently existing HATs are the TV HAT, the Power over Ethernet HAT and the Sense HAT, as well as many other unofficial HATs. Other devices that can extend functionality are the Raspberry Pi Touch Display and, the Pi NoIR Camera V2 and the Camera Module V2 [19].

The Camera Module V2 (figure 1.8 [20]) will be used in this project in order to capture video streams. It can be connected to the Raspberry Pi by means of a CSI connector. The most important specifications of the Camera Module V2 can be seen in table 1.2 [21].

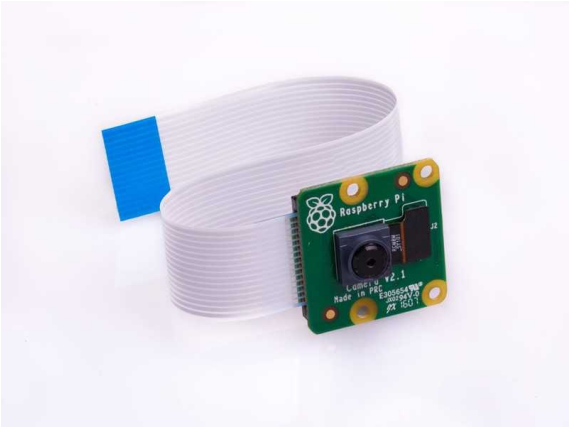

Figure 1.8: Camera Module V2

Weight	3 g
Sensor	Sony IMX219
Resolution	8 megapixels (3280 × 2464 pixels)
Video modes	1080p30
	720p60
	480p60 / 480p90
Video formats	Raw H.264

Table 1.2: Extract of the technical specifications of the Camera Module V2

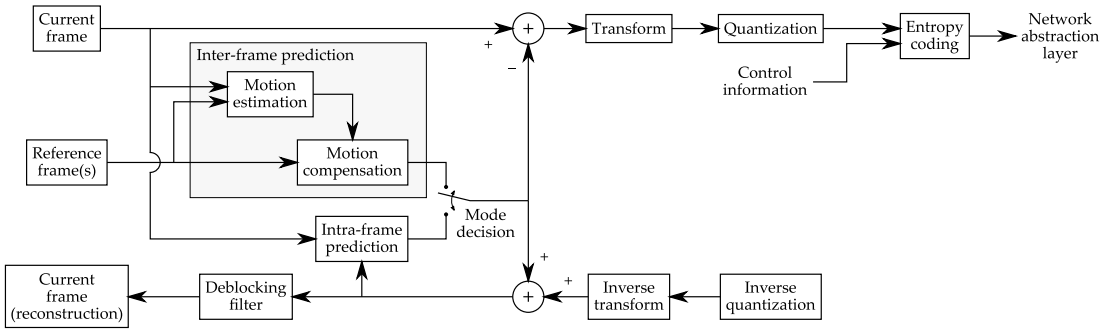

Figure 1.9: Scheme of H.264 encoding

1.5 Video codecs

Video transmission is a bandwidth monster: a standard definition TV video would need more than 200 Mbps if transmitted "as it is". Therefore, video content needs to be compressed in some way before it is transmitted and it needs to be restored after it is received in order to display it.

A video codec (from "encoder-decoder") is a piece of software or hardware that performs a series of operations on video in order to compress it so that it requires less bandwidth (encoding), and then reverts those operations and gets the original video (decoding) – or at least a similar one. In order to reduce the amount of data of a video, it is unavoidable to remove meaningful information if a decent result is desired, and that implies in a loss of quality because there is not enoguh information to reconstruct the original video at the decoding stage. The point is then to remove the appropriate information such that the quality loss is barely noticeable.

That depends on the specific mechanism used to encode the video, that is, on the codec. As years pass and research is done, codecs become more efficient, or what is the same, they can deliver a higher quality using the same bandwidth. This is possible thanks to improvments in hardware capabilities.

There are a lot of codecs, but the most popular ones are those developed by MPEG. They reduce the bandwith by eliminating both spatial redundancy – pixels close to each other with similar colours – with the usage of DCT, quantization and run-length encoding, and temporal redundancy – similar frames close in time – with the usage of motion detection and compensation (figure 1.9). The first MPEG codecs were MPEG-1, which is now obsolete, and MPEG-2. The most popular MPEG codec nowadays is MPEG-4 AVC, also known as H.264, and it is an evolution of MPEG-2. HEVC (H.265) is even more efficient than H.264 and it is now starting to take off. Taking all that into account, H.264 was chosen as the video codec for the project.

Codecs are implemented in libraries so that they can be used. One of the most popular libraries that implements H.264 is x264, which is open-source and developed by VideoLAN. Another option is the H.264 implementation by OpenMAX, which provides suport for hardware encoding. Hardware encoding uses less resources and is more efficient in terms of computational costs than software encoding, and because Raspberry Pi 3 Model B+ has the appropriate hardware, the OpenMAX H.264 video codec will be used in this project.

1.6 FFMPEG

FFMPEG is a software tool initially devised as a multimedia format converter. It includes an enourmous range of libraries and codecs, so it can work with virtually any existing multimedia format.

Its features have evolved a lot and now it can be used for format conversion, scaling, effects, media filtering, concatenation, mixing, and much more. FFMPEG has such a flexibility that if all its possibilities were to be described, the list would be longer than this document. It is truly the Swiss Army knife of multimedia files.

FFMPEG includes three tools [22]:

- `ffmpeg` is the main and most used tool of all. It is the versatile converter mentioned before.
- `ffplay` is a basic media player.
- `ffprobe` is a media analyzer that shows useful technical information.

FFMPEG is a key piece of this project, because it will be used to convert the video stream into the appropriate format (DASH) in real time.

1.7 NGINX

NGINX is a web server. There are many other web servers in the market, like Apache, Microsoft's IIS or Google Web Server. If we take a look at the evolution of market share (figure 1.10 [23]), we can see how the once hegemonic Apache has given way to other solutions and how NGINX has experienced a significant growth, trend which continues today. They have nowadays the same market share (29 %).

In comparison with Apache, its direct competitor, NGINX is less flexible and more difficult to install. However, NGINX can handle many more simultaneous requests, and its light weight makes it very power-efficient. That makes it ideal for small devices with tighter power requirements like Raspberry Pi, so it will be used in this project.

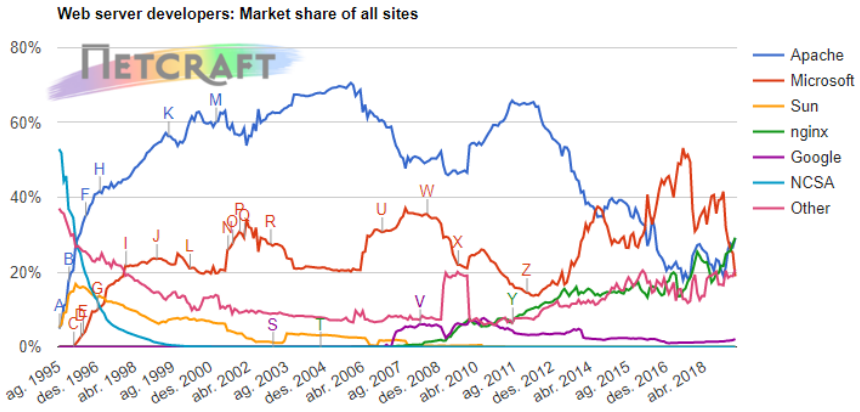

Figure 1.10: Evolution of web server market share

CHAPTER 2
Objectives

The main objective of this project is the development of a system capable of capturing, encoding and livestreaming video in an adaptive way, and providing the end user the option to interactively choose the visualization of one among several video streams.

Two different options can be considered regarding the structure of the system:

- In option A (figure 2.1), video capture devices would transmit all segments and manifest files to a web server that would act as a central node. Clients would then connect to this server to fetch both the appropriate DASH segments and manifests and the web page that would act as a user interface.
- In option B (figure 2.2), the web server would only provide the web page with the interface, and video capture devices would serve the DASH video segments and manifest files directly to the clients.

In any case, what becomes clear is that several web servers have to be installed in different places. All capture devices must have one web server that stores the video segments, since DASH is HTTP-based. Apart from that, another server must be installed preferably on a different machine. This one will store at least the web page, and the DASH segments as well if option A is chosen.

In the web page, users must be able to see a list of video capture devices and they must be able to select one in order to watch the corresponding stream in a video player.

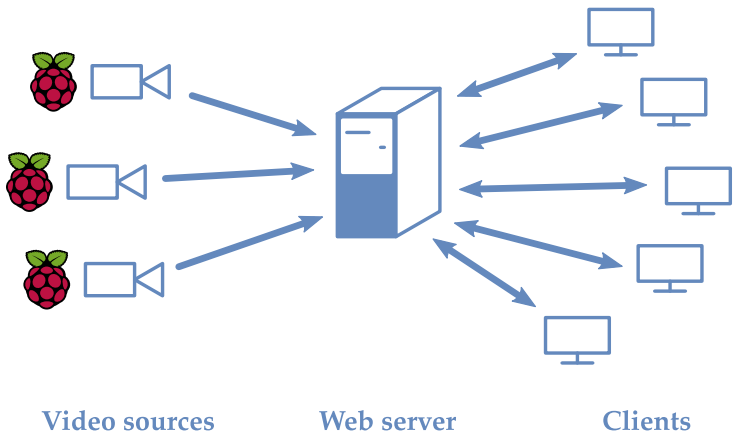

Figure 2.1: Diagram showing option A for the structure of the system

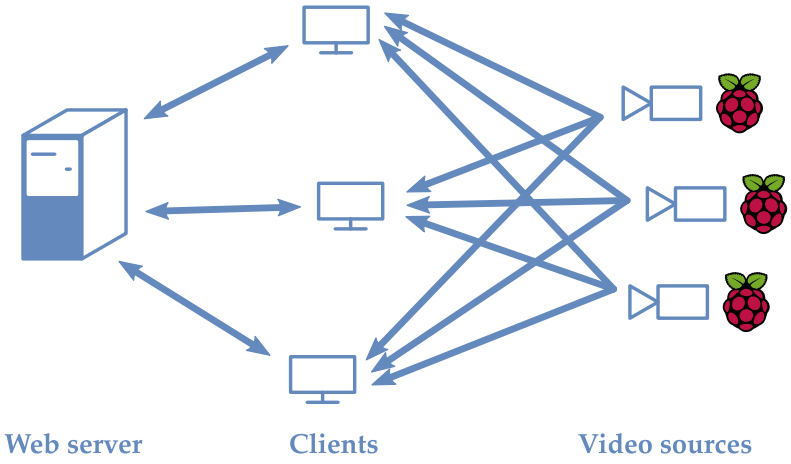

Figure 2.2: Diagram showing option B for the structure of the system

Users should be able to add new video capture devices or remove existing ones from the list, and they should be able to switch the viewed video stream flawlessly.

The video capture devices must be portable and power-efficient, what makes Raspberry Pi the most suitable kind of device. The appropriate software will have to be installed on the devices and configured so that they can provide the adaptive video stream as specified. The standard used for the adaptive video streaming will be DASH.

The quality of the video streams must vary according to available bandwidth, and the latency must be as low as possible. Interruptions in playback are not desired.

The web server will also have to be correctly set up and the web page will have to be programmed using HTML, CSS and JavaScript so that the specified service is provided to users.

CHAPTER 3
Methodology

3.1 Project management

The project was executed as a collaboration with the COMM research group at iTEAM. Progress was reported to the tutors when partial goals were reached and when problems arose. Emails alternated with meetings face to face for more detailed monitoring.

Before the actual work started, the objectives of the project were selected and a work plan was laid out that included the division of the project into separate tasks. They are shown in section 3.2.

3.2 Task distribution

This project was divided conceptually into four conceptual blocks, and each of them was further divided into several small tasks. This was done in order to tackle the development of the project in a more systematic way.

These were the four main blocks:

1. Acquisition of theoretical background
2. Multimedia content generation
3. Deployment of a web server
4. Development of a web application

They were thought of as consecutive blocks, but the actual development of the project did not follow that order.

Besides the tasks that strictly form the project, writing this report was another task that had to be done. It was written in parallel with the development of the project, and it was extensively reviewed afterwards.

The blocks and tasks are explained more in detail over the following subsections.

3.2.1. Acquisition of theoretical background

Some of the technologies used in this project were shallowly studied in different courses of the degree. However, a deeper knowledge was required, so some research had to be made on those technologies, most notably on web application programming. This

process was divided into the following tasks, which more or less correspond to sections of the introduction of this report:

1.a. Research on trends of Internet traffic

1.b. Research on streaming systems

1.c. Research on Shaka Player

1.d. Research on Raspberry Pi

1.e. Research on video codecs

1.f. Research on FFMPEG

1.g. Research on NGINX

1.h. Learning web application programming

In theory, these tasks should have been done at the beginning of the project, but they were actually carried out around June – i.e. when writing the introduction of this report – because a large part of that theoretical knowledge turned out to be unnecessary to develop the project. This can be seen in the time diagram (section 3.3).

The online course on web application programming (HTML, CSS and JavaScript) was started some weeks before the web app had to be created.

3.2.2. Multimedia content generation

This block consisted in generating a valid DASH stream locally inside the Raspberry Pi. It was divided into the following tasks:

2.a. Setup of Raspberry Pi and installation of Raspbian

2.b. Installation of FFMPEG and H.264 libraries and codecs

2.c. Installation of NGINX

2.d. Camera setup and configuration

2.e. DASH content generation with FFMPEG

2.f. Addition of audio support

2.g. Service creation

The different tasks and steps taken were partly based on [24], [25]. Both Bachelor's theses include systems of live video capture and streaming, so the multimedia generation tasks were expected to take just a small part of the time devoted to the project.

However, trying to replicate the steps of those Bachelor's thesis was unexpectedly unsuccessful, and the research for the causes until a solution was found delayed the project significantly in comparison to the schedule.

The addition of audio support was not initially planned, but it was carried out because it was judged to add much value to the project with a relatively small amount of work.

3.2.3. Deployment of a web server

This block included the setup and configuration of the web servers that will host the web aplication and the multimedia content. It consisted of the following tasks:

3.a. Installation of Apache on a PC

3.b. NGINX CORS configuration

It was initially expected that the server hosting the web application would be an actual dedicated web server from the university, but since the author's personal laptop proved to be sufficient for the development of the project, it was decided that the task of configurating a dedicated server completely was not necessary.

The CORS task was defined after unforeseen problems that arose when testing the DASH streams.

3.2.4. Development of a web application

This block consisted in the creation of a web page that would provide an interface to the end user in order to access the multimedia content in an easy and intuitive way. It was divided in the following tasks:

4.a. Development of a basic web page for testing

4.b. Upgrade to a functional web page

4.c. Addition of styles

Of those tasks, the first one had to be carried out while working on the content generation block, since a way of testing locally the generated DASH streams was needed – that is why it was just a simple player. The other tasks did belong to the actual development of the web page.

3.3 Time diagram

The distribution of the different tasks over time is shown on figure 3.1.

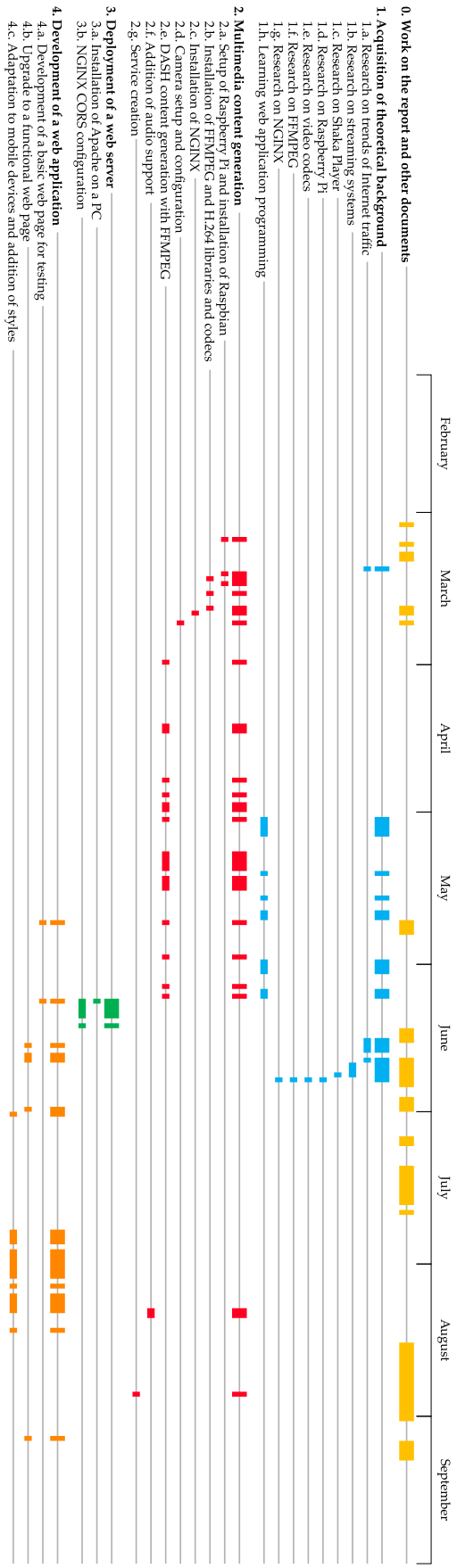

Figure 3.1: Time diagram of the project development

CHAPTER 4
Development and results

4.1 Multimedia content generation

The first step towards the deployment of the complete system was to achieve the generation of multimedia content – more specifically, a DASH video stream – locally inside a Raspberry Pi, since the aim was to use such devices as content generators in the form of DASH streams in the final system. The system was initially developed with only one Raspberry Pi as video source, and a second one was configured following the same steps when developing the web page, as can be seen in subsection 4.3.3

To achieve that, several components had to be installed, configured and put together so that a manifest file and video fragments of different qualities were generated. Among those components were the Raspberry Cam, libraries, encoders and a web server.

4.1.1. Setup of Raspberry Pi and installation of Raspbian

To begin, the Raspberry Pi was placed in a case and the Raspberry Cam was connected to its socket. An image of Raspbian that included a desktop environment and some software was then downloaded from the official site of Raspberry [26]. As required by the Raspberry Pi, the Raspbian image was flashed on a blank microSD card by using Etcher. Etcher is a program that makes it possible to format an SD card and flash an image on it with just one click. This is the simplest way to flash an image onto an SD card and it is recommended by Raspberry [27].

The microSD card was inserted into the Raspberry Pi, which was booted up, configured and updated. The VNC interface was enabled immediately in order not to depend on a screen a keyboard and a mouse to interact with the device, thus allowing mobility. Moreover, an account in RealVNC's systems was created in order to enable VNC connections outside the scope of the same local network through RealVNC's servers, which allowed to work on the project from a location different to that of the devices [28].

Some video settings in the file `config.txt` had to be modified as well specially for the first use in order to obtain a correct image on the display.

4.1.2. Installation of FFMPEG and H.264 libraries and codecs

As stated in the introduction, FFMPEG is the tool that performs the video processing, so it was installed. The chosen model of Raspberry Pi supports hardware encoding, which is of great importance because otherwise the processing power of the device is too low for encoding to be possible in real time applications like this one. The OpenMAX library,

Figure 4.1: Default web page of NGINX

which implements the H.264 codec using hardware acceleration, is already included in Raspbian, so its installation was not needed. Even though it was not necessary and the device did not have enough computing power for it to be used, the x264 library was installed because it could be useful for testing purposes.

The corresponding commands can be found in appendix A.

4.1.3. Installation of NGINX

The multimedia content generated by the Raspberry Pi had to be stored in a web server inside the same device so that it could be accessed from other machines. NGINX was chosen from among the different available web servers because of its suitability for low power devices, as explained in the introduction.

NGINX was installed, and the file `/etc/nginx/nginx.config` was edited to specify where in the local file system the resources would be stored and what the possible "main pages" were [29]. It was also set to start automatically every time the Raspberri Pi booted up. Additionally, the default permissions for the NGINX folder were modified so that no superuser permission was needed to create files in it.

In order to check that NGINX had been successfully installed and configured, the IP address of the Raspberry Pi was obtained and it was accessed from a web browser in a different machine. The result in figure 4.1 was obtained.

Details on the specific commands and configuration are shown in appendix A.

4.1.4. Camera setup and configuration

With all the necessary software in place, the hardware – the Raspberry Pi camera, in our case – had to be enabled and set up. The camera connector had already been inserted, so in this step it was enabled in the operating system, and a reboot was performed to make the change effective.

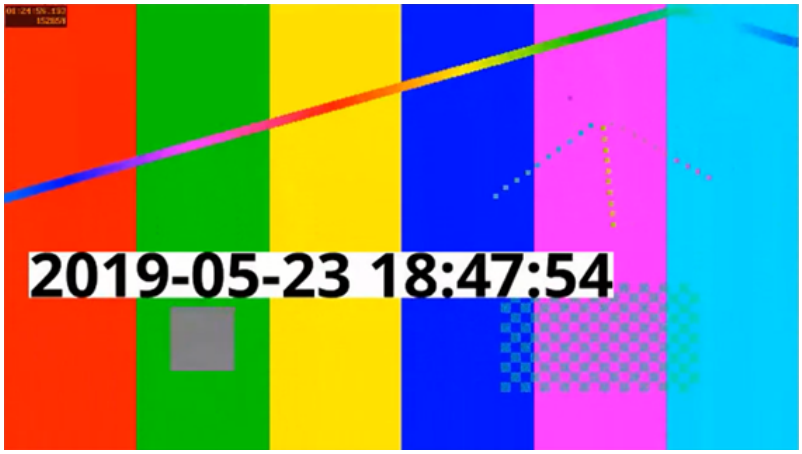

Figure 4.2: Frame of the generated test video

The camera was tested by taking some pictures, which were placed in `/var/www/html`, i.e. the web server folder. They were then accessed successfully from a PC in the same local network as the Raspberry Pi, thus checking that NGINX was working correctly.

Once it was clear that the configuration was correct, it was necessary to extract the camera stream. This is possible using the V4L2 module (Video for Linux 2) corresponding to the Raspberry Pi camera, which makes it appear inside the directory `/dev` as if it were a USB camera [30]. This makes it possible to extract the video stream. In fact, if we try to read the contents of `/dev/video0` (which is the camera node), what we get is a constantly changing screen of random characters which is presumably the video stream.

The V4L2 module was thus loaded and the file `/etc/modules` was modified to keep the module loaded even beyond the shutdown of the device.

More detail on the commands can be found in appendix A.

4.1.5. DASH content generation with FFMPEG

The only remaining step in order to generate multimedia content locally in the Raspberry Pi was the execution of the right command. In order to accomplish that, the documentation of FFMPEG [31] was read, especially the sections dealing with general file options, video parameters, `video4linux` devices, the `libx264` video encoder and the `dash` muxer.

In order to become familiar with the operation of FFMPEG, some simple commands were run, such as grabbing the video stream from the camera for 10 seconds and storing it using hardware-accelerated encoding into an MP4 file located inside the NGINX folder. That allowed to check that everything worked together correctly.

FFMPEG commands of all sorts were then tried in order to generate a DASH live stream with the video input, but they were all unsuccessful. A manifest file was always created, as well as the appropriate media files, but the actual media content was not inserted and thus it could not be played. This issue, which delayed the progress of the project considerably, was solved by specifying the extension `.m4s` for the media files instead of the much more common `.mp4`.

This solution was first tested with the live generation and transmission of a test screen (figure 4.2) that was viewed in a preliminary version of the web page (see section 4.3.2) from a PC in the same local network as the Raspberry Pi.

Figure 4.3: USB microphone MI-305

However, the command was soon modified to grab the content from the camera input and, most importantly, to generate two streams of different qualities by applying a scale filter on a copy of the high-quality stream.

Besides basic parameters such as the input video stream, the bitrate, the framerate or the resolution, there are options that need to be specified in order to obtain a successfully playableDASH stream. These include `media_seg_name` and `init_seg_name` to specify the file names, and `window_size`, `extra_window_size` and `adaptation_sets` as the most important ones.

The options `window_size` and `extra_window_size` deal with the number of media segments that are kept at the same time, or what is the same, how long the segments are kept before they are deleted. If this window size is too small, segments will be deleted before they are transmitted to the receiver, and if it is too large, media segments will take too much space in the server unnecessarily. The window size therefore depends on the segment duration as well and must be carefully set.

The option `adaptation_sets` was used in combination to the option `map` to specify which streams would be included in which DASH adaptation sets. That option was necessary in order to override the default behaviour of FFMPEG and include both the original and the downscaled stream in the same adaptation set, since they are variants of the same content. The command was inferred after the example in [32].

4.1.6. Addition of audio support

Once the generation of the video stream worked correctly, it was thought that adding audio to that stream was the natural step to take in order to complete the task. A small, cheap USB microphone such as the MI-305 (figure 4.3) [33] was used for that purpose.

In order to use the microphone, FFMPEG had to be run with ALSA (Advanced Linux Sound Architecture) support. Because ALSA was not in place, FFMPEG had to be uninstalled before installing the ALSA library and compiling and installing FFMPEG again. In that way, FFMPEG could detect and recognize the library.

Some ALSA commands were run in order to obtain the identifier of the USB microphone [34]. Once that was done, some simple commands were executed to check that FFMPEG worked correctly with the microphone input. The last step was to modify the video generation FFMPEG command in order to include audio in the DASH stream.

The installation commands and the media generation commands are explained in appendix A as if audio support had been included from the very beginning.

4.1.7. Service creation

The content generation was already functional at that point, but it could be made more convenient if, instead of having to execute the FFMPEG command every time the device was booted up, the content generation began automatically when the Raspberry Pi was started.

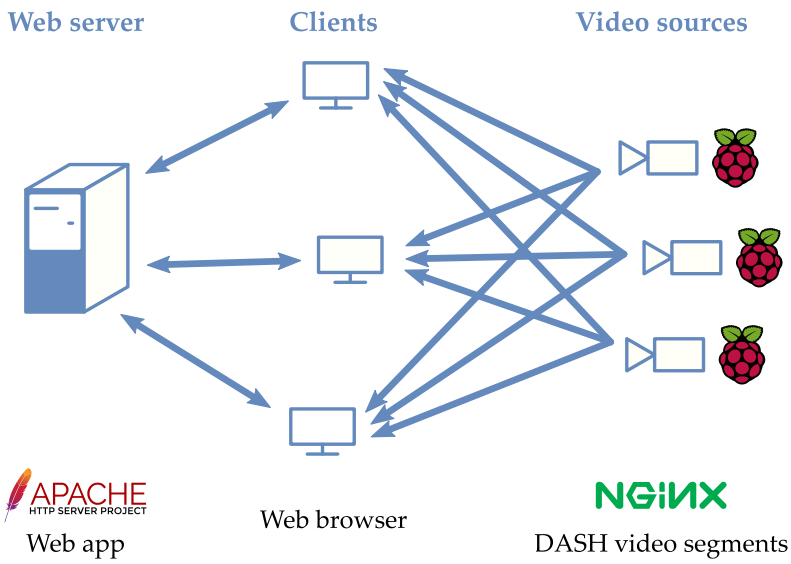

Figure 4.4: Diagram showing the chosen system structure

An easy way to configure that was the creation of a `systemd` service. A service is a process that runs silently in the background without user interaction. Most interestingly, services can be configured to run automatically after the system starts or depending on certain conditions, they can be set to restart if they fail, etc.

In our case, a new service was created that executed just the command that was determined previously in section 4.1.5. It was configured after the example in [35] to start after the network was ready and, if it failed, to keep trying forever to restart once every second. The specific service file is listed in appendix A.

Once the file was saved, the serviced was started and enabled for its execution at startup, and the Raspberry Pi was rebooted. The content generation proved to work correctly with no interaction.

4.2 Deployment of a web server

According to the scheme that was laid out in chapter 2, clients should access a web page that would play a stream when the user enters the IP of a Raspberry. From among the two proposed system structures, option B was chosen, where clients connected directly to the Raspberry Pi devices with no intermediate servers handling video content.

The reason for that was the greater simplicity of that system structure, given that option A would have implied the creation of mechanisms to control the server from the web page in order to receive and store the media segments. However, this decision came at the cost of increasing multimedia traffic and hindering the scalability of the system.

In order to deploy that structure (figure 4.4), there had to be a web server that stored the web page (Apache in our case), and another web server in each of the capture devices, which installation was described in subsection 4.1.3. Both had to be accessible by the clients.

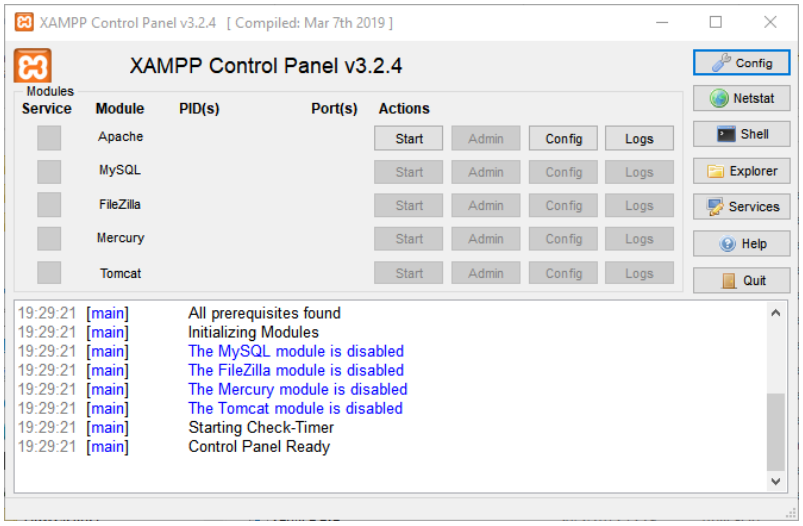

Figure 4.5: Control panel of XAMPP

4.2.1. Installation of Apache on a PC

The web server for the web page was installed in a personal computer in order to ease the development of the web page. Since that computer ran on Windows, the chosen web server was Apache because of the easy installation process. Besides that, a PC does not have the computing power limitations of a Raspberry Pi, so there was no need to use an efficient web server like NGINX. In summary, convenience was prioritized over efficiency.

Apache is distributed for Windows as part of the XAMPP package [36], which also includes other modules, namely: Tomcat, MySQL, FileZilla and Mercury. Because the installation of Apache was only carried out in order to develop the web page in a more convenient way, the other modules were not installed. The control panel of XAMPP is shown in figure 4.5.

4.2.2. NGINX CORS configuration

Once Apache had been installed on the PC, the web page displaying the video (see subsection 4.3.2) was moved from the Raspberry Pi web server to the PC web server. In theory, the only change to be made was the reference to the video stream manifest file, which was previously just a relative URL and was changed to include the Raspberry Pi IP address as well. However, CORS policies came across.

CORS (Cross-Origin Resource Sharing) is a mechanism that allows a server in a certain domain (or origin) to access resources that are located in a server in a different origin [37], [38]. This is achieved by means of extra HTTP headers: when requesting the content from another domain, the `Origin` header is added together with the domain of the server where the web page is hosted. In turn, the server of the requested resources adds an `Access-Control-Allow-Origin` header to the HTTP response together with either the domain specified in the request or a wildcard (meaning that servers in any domain can access the resources).

If the request is more complex (usually if it could potentially modify the server's resources), then a so-called pre-flight request is performed, where the requesting server checks if the operation will be allowed before performing it.

The reason why CORS exists is to soften the rather strict and inconvenient same-origin policy where only resources in the same domain can be accessed, while avoiding security breaches where a server of one domain could "steal" resources or data from a server in a different domain.

In our case, CORS policies were not thought of initially, and the client browser threw an error as a consequence when trying to fetch the media files from the Raspberry Pi. The solution was to configure CORS on the web server on the Raspberry Pi. Some research was done in order to find the specific procedure for that matter, and the solution was to add

```
add_header 'Access-Control-Allow-Origin' '*' always;
```

to the `http` block of the NGINX configuration file in `/etc/nginx/nginx.conf`. This was possible because, in the scope of our project, no pre-flight request is expected. Obviously, this CORS access control is actually no access control, because it allows servers in any domain to fetch the content. A proper access control would involve some logic that handled pre-flight requests, so the added line was a barrier-free entry that could be a security issue in a real-world environment.

The reason why the CORS access control was done that way was for convenience. The goal of this project is the development of a streaming service, and a security study is outside of its scope. The client browser simply required an `Access-Control-Allow-Origin` header and the instruction that was added provided it, so the multimedia content could reach the client browser.

After this configuration, the client browser was able to access both the web page server and the Raspberry Pi multimedia content server given that the machines could be reached. That means that communication was possible inside the same local network, but not if the machines were in different local networks (unless a port was open and port forwarding was in place, etc.).

4.3 Development of the web application

The generated video live streams had to be accessed in some way by the end user, and the chosen solution was to develop a web application because that would enable an easy access to the content. The web page would integrate Shaka Player (Google's DASH player) and would consist of a main area with a large video player and a sidebar with a list of available streams, in such a way that the selected stream would be played in the large player.

4.3.1. Online course on HTML, CSS and JavaScript

After defining the goals and methodology of this project, the insufficient level of knowledge on the necessary technologies for its development became apparent, so a MOOC (Massive Open Online Course) on HTML 5, CSS 3 and JavaScript 5 was taken [39].

The course covered, among other topics, the basics on:

- HTML blocks
- CSS structure and properties
- CSS inheritance

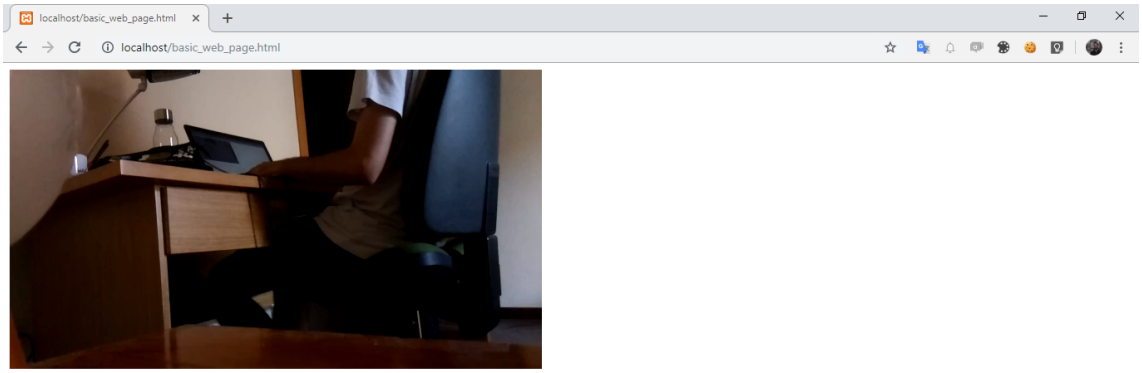

Figure 4.6: Screenshot of the basic web page

- Multimedia integration
- JavaScript expressions, types, operators, functions and objects
- HTML element handling with JavaScript
- jQuery

This was deemed sufficient for the development of the web page.

4.3.2. Development of a basic web page for testing

The web page started to be designed even before the final FFMPEG command was found. The reason for this was the need of visualizing the video stream that was generated as a test, since VLC proved incapable of playing those files, the demo of Shaka Player that is hosted online [40] does not support unsecure HTTP, and Shaka Player itself could not be compiled on the Raspberry Pi due to outdated, unupgradable components.

The adopted solution was to design a preliminary page after the guidelines found on [41] with the appropriate modifications. This preliminary page (figure 4.6), which was stored in the NGINX folder of the Raspberry Pi, consisted in just a video player pointing to the DASH manifest file, a link in the head section pointing to the compiled library of Shaka Player and a script to get the player to find and play the stream. The web page with the live stream were then accessible from any device on the same local network as the Raspberry Pi.

This preliminary web page was the base for all subsequent developments until reaching the fully functional site described at the beginning. The approach taken was to implement the functionalities in the first place without taking into account the appropriate graphic design, and then develop the aesthetical part in a second phase.

4.3.3. Upgrade to a functional web page

The simple web page served as a base to develop the required functionalities, i.e. a layout with a large video player and a list of selectable video sources that could also be added or removed.

An area was placed at the right side of the web page containing a list of the added video sources represented by a thumbnail – i.e. a small representation of the stream – and a description. A button was added to each thumbnail so that it could be removed from the list, and a couple of text boxes were placed to allow the addition of a video source to the list just by typing its IP address and, optionally, its description.

Each of the thumbnails is a Shaka video player by itself, for which the original player in the basic web page served as a template. The thumbnails are stored in an array of Shaka player objects, and their corresponding descriptions are stored in an array as well. Both arrays are updated when video sources are added or removed, and the corresponding HTML elements are added or removed as well.

The reason why only the IP address (or a URL) is necessary to add a video source is that the name and location of the video files in each of the sources (i.e. the Raspberries) is always the same, since they are under our control.

Clicking on one of the thumbnails displays it in full size in the main part of the web page, together with its description, and activates its associated audio stream. This was accomplished by using a `canvas` element in combination with a timer, so that the frames of the selected thumbnail player are copied to the main area repeatedly. This approach was chosen instead of creating an additional video player in order to reduce the video traffic and avoid the lack of synchronization between the main "player" and the corresponding thumbnail.

In order to test the functionality of the player, a second Raspberry Pi was set up following the same process as with the first one, and the FFMPEG command was executed in both of them. Their IP addresses were then noted down and used to add the corresponding thumbnail videos several times for each of the video sources. Removing and selecting sources was also tried.

The results were in almost all cases according to the expected behaviour – at least on the practical level. The playback of the video streams always started with low quality and switched to high quality after some seconds. After looking at the browser's developer console, it was discovered that the browser was sometimes making several attempts at getting the video segments before they were finally available for download, but because that did not seem to affect the video playback, it was considered a minor issue. Addition, removal and selection of thumbnail video players behaved as expected. With this, a fully functional web page had been achieved (figure 4.7).

4.3.4. Adaptation to mobile devices and addition of styles

Once the basic functionality of the web page was in place, it was time to make it look nice and make it work across devices. This was accomplished with a combination of CSS styles and JavaScript functions following the guidelines on [42].

The web page was made responsive so that it looked good regardless of screen size by using CSS media queries and setting different sizes and properties to some elements accordingly. In order to increase the responsiveness, plain old `div` and `span` elements were replaced by `flex` containers designed to adapt better to different window dimensions. For instance, the sidebar at the right part of the web page is moved below the main

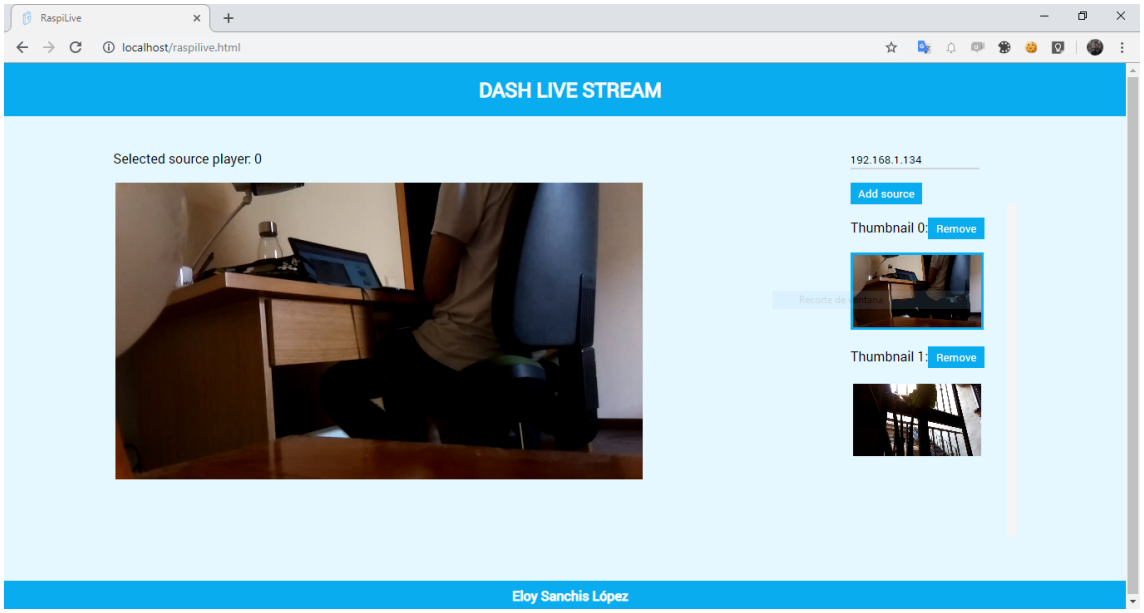

Figure 4.7: Screenshot of the functional web page

player when the window is narrow, such as in mobile phones, and the thumbnails are displayed horizontally instead of vertically in that case. Moreover, a JavaScript function was written that computes the size of some elements continuously when the window is resized.

As for the aesthetic part of the web page, the style is based on Google's well-known Material Design, which is present across the company's products. The reason why these design guides were chosen is their clarity despite their elegance. However, only some of their elements were taken.

The two main parts of the web page were visually structured as cards that appear to be raised in comparison to the light grey background, and the header and the footer were added some shadow too. The icons and the text boxes were taken from the Material Design library, together with their animations. All this was done by linking to MDL (Material Design Lite, a library of CSS and JavaScript files based on Google's Material design) [43] and adding the appropriate classes to the relevant elements.

Besides Material Design, the area with the text boxes was made collapsable with a JavaScript function so that it did not take too much space when many thumbnails were present. The video source removal button was programmed, in the desktop version, to appear only when the mouse is hovered over the corresponding thumbnail. In narrow screens such as mobile phones the button is always present.

Lastly, the project was given the name of "RaspiLive" and a logo was designed for it. A favicon was added.

The finished web page is shown in figure 4.8 and the code is listed in appendix B.

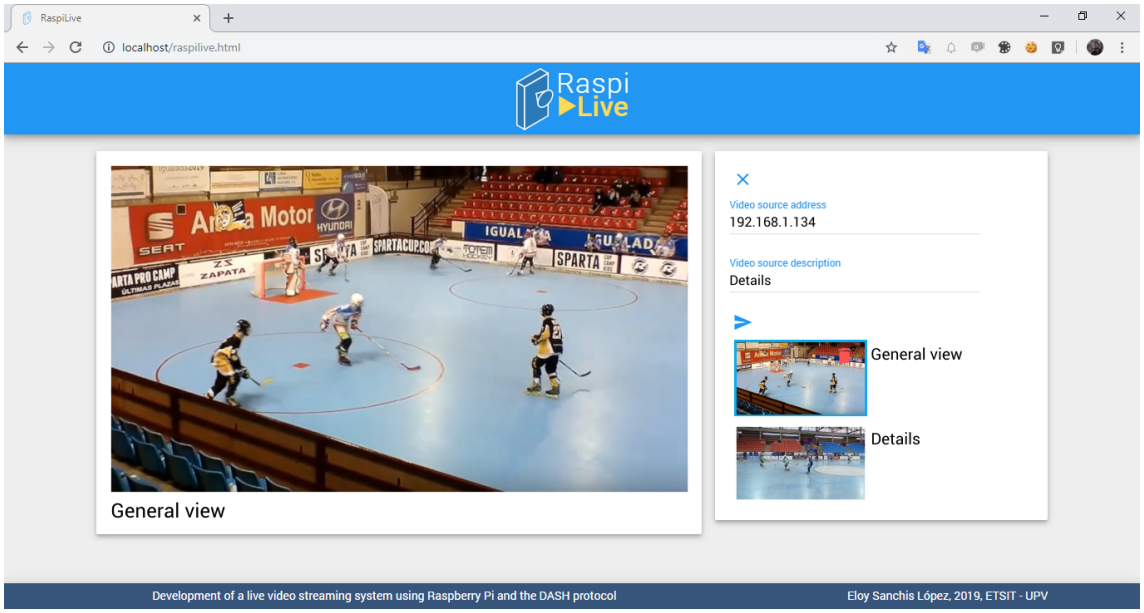

Figure 4.8: Screenshot of the final web page with simulated video sources

CHAPTER 5
Conclusions and proposals for further work

The main goal of the Bachelor's thesis was reached: the present project was developed successfully, and in view of the growing importance of video, it could become an interesting product with further development. A functional live video streaming system was completed and it was proven to work correctly, and the visible part of the project – i.e., the web app – works according to the specifications. The condition of making the system portable and power-efficient was fulfilled as well, since the capture devices are Raspberry Pi, which meet those criteria.

Regarding possible enhancements, some would be necessary in order to make the project suitable for a production phase and real-world scenarios. One of them would involve creating two different interfaces instead of one – namely one for the system administrator and another one that would be available for users. That would enable the administrator to add and remove video sources, whereas that possibility would not be available to users. This would be convenient in scenarios like broadcasting of events to the general public.

The most important enhancements, which would be key to user experience, would involve overcoming or reducing the current limitations of the project, namely latency, synchronization among different capture devices, audio-video synchronization, and scalability of the system. Scalability could be improved by implementing option A for the architecture, as described in chapter 2, while latency and synchronization would require more complex solutions. In particular, a reduction of latency would require the implementation of one of the several strategies of low latency delivery [44].

On a more personal level, the development of this project has allowed the author to face for the first time the construction of a full communications system integrating diverse technologies and has required to learn the usage of different tools which will be an asset in his professional skills as an engineer.

Bibliography

[1] Cisco, *Cisco Visual Networking Index: Forecast and Trends, 2017–2022*, February 27, 2019. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.pdf. [Accessed June 20, 2019].

[2] Sandvine, *The Global Internet Phenomena Report*, October 2018. [Online]. Available: https://www.sandvine.com/hubfs/downloads/phenomena/2018-phenomena-report.pdf. [Accessed June 20, 2019].

[3] H. Riiser, "Adaptive bitrate video streaming over HTTP in mobile wireless networks," June 16, 2013. [Online]. Available: https://www.duo.uio.no/bitstream/handle/10852/37404/dravhandling-riiser.pdf. [Accessed June 20, 2019].

[4] *RTP: A Transport Protocol for Real-Time Applications*, IETF RFC 1889, 1996.

[5] *RTP: A Transport Protocol for Real-Time Applications*, IETF RFC 3550, 2003.

[6] C. Perkins, *RTP: Audio and Video for the Internet*, 1st ed. Boston, MA: Pearson Education, 2003. [E-book]. Available: Google Books

[7] The WebRTC Project, "Frequent Questions." [Online]. Available: https://webrtc.org/faq/. [Accessed June 22, 2019].

[8] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP," February 23–25, 2011. [Online]. Available: https://www.cc.gatech.edu/~dovrolis/Papers/final-saamer-mmsys11.pdf. [Accessed July 7, 2019].

[9] "Part 1: media presentation description and segment formats," in *Dynamic Adaptive Streaming over HTTP (DASH)*, 3rd ed., ISO/IEC 23009-1, 2019.

[10] DASH Industry Forum, "About." [Online]. Available: https://dashif.org/about/. [Accessed June 25, 2019].

[11] T. Stockhammer, "MPEG's Dynamic Adaptive Streaming over HTTP (DASH) – Enabling formats for video streaming over the open Internet," from a webinar at EBU, November 22, 2011. [Online]. Available: https://tech.ebu.ch/docs/events/webinar043-mpeg-dash/presentations/ebu_mpeg-dash_webinar043.pdf. [Accessed June 24, 2019].

[12] "MPEG-DASH – An Overview," *encoding.com*. [Online]. Available: https://www.encoding.com/mpeg-dash. [Accessed June 25, 2019].

[13] C. Mueller, "MPEG-DASH (Dynamic Adaptive Streaming over HTTP, ISO/IEC 23009-1)," *bitmovin.com*, April 21, 2015. [Online]. Available: https://bitmovin.com/dynamic-adaptive-streaming-http-mpeg-dash/. [Accessed June 24, 2019].

[14] G. Kopley, "example.mpd," *gist.github.com*. [Online]. Available: https://gist.github.com/gkop/1bed9f8d888f2e4ceb966cf75a80b9d1. [Accessed July 14, 2019].

[15] Google, "shaka-player," *github.com*. [Online]. Available: https://github.com/google/shaka-player. [Accessed August 12, 2019].

[16] Raspberry Pi, "Downloads." [Online]. Available: https://www.raspberrypi.org/downloads/. [Accessed June 24, 2019].

[17] Raspberry Pi, "Raspberry Pi 3 Model B+." [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/. [Accessed June 24, 2019].

[18] Raspberry Pi, Raspberry Pi 3 Model B+ datasheet. [Online]. Available: https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf. [Accessed June 24, 2019].

[19] Raspberry Pi, "Products." [Online]. Available: https://www.raspberrypi.org/products/. [Accessed June 24, 2019].

[20] Raspberry Pi, "Camera Module V2." [Online]. Available: https://www.raspberrypi.org/products/camera-module-v2/. [Accessed June 24, 2019].

[21] Raspberry Pi, Camera Module V2 datasheet. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/camera/. [Accessed June 24, 2019].

[22] FFmpeg, "About FFmpeg." [Online]. Available: https://www.ffmpeg.org/about.html. [Accessed June 24, 2019].

[23] "May 2019 Web Server Survey," *news.netcraft.com*, May 10, 2019. [Online]. Available: https://news.netcraft.com/archives/2019/05/10/may-2019-web-server-survey.html. [Accessed June 24, 2019].

[24] L. Espinach Casacuberta, "Adaptive real time multimedia transmission," Bachelor's thesis, Universitat Politècnica de Catalunya, Barcelona, January 2017.

[25] S. García Jiménez, "Desarrollo de paneles de control para redes IoT basados en NodeRED," Bachelor's thesis, Universitat Politècnica de València, València, December 3, 2018.

[26] Raspberry Pi, "Raspbian." [Online]. Available: https://www.raspberrypi.org/downloads/raspbian/. [Accessed July 18, 2019].

[27] Raspberry Pi, "Installing Operating System Images." [Online]. Available: https://www.raspberrypi.org/documentation/installation/installing-images/README.md. [Accessed July 18, 2019].

[28] Raspberry Pi, "VNC (Virtual Network Computing)" [Online]. Available: https://www.raspberrypi.org/documentation/remote-access/vnc/README.md. [Accessed August 21, 2019].

[29] M. Fjordvald, "Nginx Configuration Primer," August 7, 2014. [Online]. Available: http://blog.martinfjordvald.com/2010/07/nginx-primer/. [Accessed July 21, 2019].

[30] "Stream Live Video from your Pi," *docs.dataplicity.com*. [Online]. Available: https://docs.dataplicity.com/docs/stream-live-video-from-your-pi. [Accessed July 22, 2019].

[31] FFmpeg, "FFmpeg Documentation," June 2019 [Online]. Available: http://ffmpeg.org/ffmpeg-all.html. [Accessed June 7, 2019].

[32] J. Kölker, "Raspberry Pi 3: How to Live Stream MPEG-DASH," *jungledisk.com*, July 3, 2017. [Online]. Available: https://www.jungledisk.com/blog/2017/07/03/live-streaming-mpeg-dash-with-raspberry-pi-3/. [Accessed June 7, 2019].

[33] Amazon, "Mini Micrófono USB Compatible Raspberry Pi MI-305." [Online]. Available: https://www.amazon.es/MINI-MICROFONO-COMPATIBLE-RASPBERRY-MI-305/dp/B072W1SS6B. [Accessed September 6, 2019].

[34] FFmpeg Wiki, "Capture/ALSA," February 25, 2016. [Online]. Available: https://trac.ffmpeg.org/wiki/Capture/ALSA. [Accessed September 6, 2019].

[35] B. Morel, "Creating a Linux Service with systemd," *medium.com*, September 5, 2017. [Online]. Available: https://medium.com/@benmorel/creating-a-linux-service-with-systemd-611b5c8b91d6. [Accessed August 27, 2019].

[36] Apache Friends, "XAMPP Installers and Downloads." [Online]. Available: https://www.apachefriends.org/index.html. [Accessed June 8, 2019].

[37] MDN Web Docs, "Cross-Origin Resource Sharing (CORS)," May 9, 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS. [Accessed June 15, 2019].

[38] Codecademy, "What Is CORS?." [Online]. Available: https://www.codecademy.com/articles/what-is-cors. [Accessed June 15, 2019].

[39] MiríadaX, "Desarrollo en HTML5, CSS y Javascript de Apps Web, Android, IOS... (9.ª ed.)," April 10, 2019. [Online]. Available: https://miriadax.net/web/html5mooc/inicio. [Accessed June 7, 2019].

[40] Shaka Player, "Shaka Player Demo." [Online]. Available: https://shaka-player-demo.appspot.com. [Accessed June 7, 2019].

[41] Shaka Player, "Basic Usage," June 13, 2018. [Online]. Available: https://shaka-player-demo.appspot.com/docs/api/tutorial-basic-usage.html. [Accessed June 25, 2019].

[42] J. Chan, "Build a Responsive Website Layout with Flexbox (Step-by-step Guide)," *coder-coder.com*, September 22, 2018. [Online]. Available: https://coder-coder.com/build-flexbox-website-layout. [Accessed July 30, 2019].

[43] Material Design Lite, "Components." [Online]. Available: https://getmdl.io/components. [Accessed August 7, 2019].

[44] P. Cluff, "The Low Latency Live Streaming Landscape in 2019," *mux.com*, January 23, 2019. [Online]. Available: https://mux.com/blog/the-low-latency-live-streaming-landscape-in-2019/. [Accessed September 7, 2019].

APPENDIX A
System installation and configuration

In this appendix a comprehensive relation is offered of all the commands and configurations used in order to develop the system. A Raspberry Pi with Raspbian Stretch installed is assumed, as explained in subsection 4.1.1.

A.1 H.264 library

The H.264 library was installed by executing the following commands in the terminal. VideoLAN's Git repository of the x264 library was cloned, the compilation was configured and it was carried out using the `-j4` option in order to use all four cores inside the CPU of our Raspberry Pi and reduce the compilation time.

```
cd /usr/src
sudo git clone https://code.videolan.org/videolan/x264.git
cd x264
sudo ./configure --enable-static
sudo make -j4
sudo make -j4 install
```

A.2 ALSA library

The ALSA library was installed by downloading the corresponding package and configuring and carrying out the compilation. The following commands were executed:

```
cd /usr/src
sudo wget ftp://ftp.alsa-project.org/pub/lib/alsa-lib-1.1.9.tar.bz2
sudo tar xjf alsa-lib-1.1.9.tar.bz2
cd alsa-lib-1.1.9
sudo ./configure --host=arm-unknown-linux-gnueabi
sudo make -j4
sudo make -j4 install
```

A.3 FFMPEG

To install FFMPEG, the corresponding Git repository was cloned, the compilation was configured with the H.264 and ALSA libraries and OpenMAX, and it was carried out using again the `-j4` option. These commands were executed in the terminal:

```
cd /usr/src
sudo git clone https://git.ffmpeg.org/ffmpeg.git ffmpeg
cd ffmpeg
sudo ./configure --target-os=linux --enable-gpl --enable-libx264 --enable-
    mmal --enable-omx-rpi --enable-nonfree
sudo make -j4
sudo make -j4 install
```

A check was performed to make sure that the installation was correct by executing `ffmpeg` and getting the help text from the tool. To check the hardware encoding, the following line was executed:

```
ffmpeg -encoders 2>/dev/null | grep h264_omx
```

and this was the output:

```
V..... h264_omx             OpenMAX IL H.264 video encoder (codec h264)
```

so hardware encoding and the H.264 codec were all in place.

A.4 NGINX

The following commands were executed in the terminal to install NGINX. Existing packages were updated just in case, and NGINX was installed using the `apt-get` package installer:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install nginx
```

This command was run to enable the execution of NGINX every time the Raspberry Pi booted up:

```
sudo systemctl enable nginx
```

These commands (not to be executed at this point) can be used respectively to start, stop or reload NGINX if needed:

```
sudo systemctl start nginx
sudo systemctl stop nginx
sudo nginx -s reload
```

To configure NGINX, some lines had to be added to the file `/etc/nginx/nginx.conf` in order to specify the server directory and to enable CORS. This is the resulting configuration:

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;
```

```
events {
  worker_connections 768;
  # multi_accept on;
}

http {

  ##
  # Basic Settings
  ##

  sendfile on;
  tcp_nopush on;
  tcp_nodelay on;
  keepalive_timeout 65;
  types_hash_max_size 2048;

  server {
    listen 80;
    server_name localhost;
    location / {
      root /var/www/html;
      index index.html index.htm;
    }
    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
      root /usr/share/nginx/html;
    }
  }

  include /etc/nginx/mime.types;
  default_type application/octet-stream;
  add_header 'Access-Control-Allow-Origin' '*' always;

  ##
  # SSL Settings
  ##

  ssl_protocols TLSv1 TLSv1.1 TLSv1.2; # Dropping SSLv3, ref: POODLE
  ssl_prefer_server_ciphers on;

  ##
  # Logging Settings
  ##

  access_log /var/log/nginx/access.log;
  error_log /var/log/nginx/error.log;

  ##
  # Gzip Settings
  ##

  gzip on;
  gzip_disable "msie6";

  # gzip_vary on;
  # gzip_proxied any;
  # gzip_comp_level 6;
  # gzip_buffers 16 8k;
  # gzip_http_version 1.1;
  # gzip_types text/plain text/css application/json application/javascript text/xml
      application/xml application/xml+rss text/javascript;

  ##
```

```
  # Virtual Host Configs
  ##

  include /etc/nginx/conf.d/*.conf;
  include /etc/nginx/sites-enabled/*;
}


#mail {
# # See sample authentication script at:
# # http://wiki.nginx.org/ImapAuthenticateWithApachePhpScript
#
# # auth_http localhost/auth.php;
# # pop3_capabilities "TOP" "USER";
# # imap_capabilities "IMAP4rev1" "UIDPLUS";
#
# server {
#   listen     localhost:110;
#   protocol   pop3;
#   proxy      on;
# }
#
# server {
#   listen     localhost:143;
#   protocol   imap;
#   proxy      on;
# }
#}
```

The last step of the installation of NGINX was to change permissions so that newly created files did not need superuser access. These commands were executed:

```
sudo chmod u=rwx,g=rwx,o=rwx /var/www
sudo chmod u=rwx,g=rwx,o=rwx /var/www/html
```

In the file `/home/pi/.bashrc`, the line `umask 000` was added, and in `/etc/login.defs`, `UMASK 022` was changed to `UMASK 000`.

A.5 Camera

First of all, existing packages were again updated to ensure that everything was up to date:

```
sudo apt-get update
sudo apt-get upgrade
```

Then, the camera interface was enabled in the start menu > Preferences > Raspberry Pi configuration > Interfaces, and the Raspberri Pi was rebooted for the changes to be effective. The Video for Linux 2 driver was then activated by executing the following command:

```
sudo modprobe bcm2835-v4l2
```

In order to make that activation permanent, the line `bcm2835-v4l2` was added to the file `/etc/modules`.

The configuration was then complete, and a picture was taken to test the camera:

```
raspistill -v -o test.jpg
```

A.6 Microphone

No additional configuration was needed for the microphone, but its identifier had to be noted down because it was needed for the FFMPEG command. In order to get it, the following command was executed:

```
arecord -l
```

A list of audio devices appeared, one of which was `USB PnP Sound Device`. The corresponding card number was noted down – in our case, card 1.

```
**** List of CAPTURE Hardware Devices ****
card 1: Device [USB PnP Sound Device], device 0: USB Audio [USB Audio]
Subdevices: 0/1
Subdevice #0: subdevice #0
```

A.7 DASH content generation with FFMPEG

This is the FFMPEG command used to generate the DASH video stream. It is explained in table A.1.

```
ffmpeg -f alsa -sample_rate 48000 -channels 1 -i hw:1 -c:a aac -re -f v4l2 -s:v 1280x720 -
    r:v 25 -i /dev/video0 -pix_fmt yuv420p -c:v h264_omx -map 1:v -b:v:0 1400k -profile:v
    high -bf 0 -refs 3 -bufsize 1400k -c:v h264_omx -map 1:v -b:v:1 300k -profile:v high -
    bf 0 -refs 3 -bufsize 300k -filter:1 "scale=160:90" -map 0:a -use_timeline 0 -
    media_seg_name 'chunk-stream-$RepresentationID$-$Number%05d$.m4s' -init_seg_name 'init
    -stream1-$RepresentationID$.m4s' -window_size 5  -extra_window_size 10 -remove_at_exit
     1 -adaptation_sets "id=0,streams=a id=1,streams=v" -f dash /var/www/html/stream.mpd
```

That command creates a manifest file like the following:

```
<?xml version="1.0" encoding="utf-8"?>
<MPD xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="urn:mpeg:dash:schema:mpd:2011"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xsi:schemaLocation="urn:mpeg:DASH:schema:MPD:2011 http://standards.iso.org/ittf/
      PubliclyAvailableStandards/MPEG-DASH_schema_files/DASH-MPD.xsd"
  profiles="urn:mpeg:dash:profile:isoff-live:2011"
  type="dynamic"
  minimumUpdatePeriod="PT500S"
  suggestedPresentationDelay="PT4S"
  availabilityStartTime="2019-09-07T21:27:41.500Z"
  publishTime="2019-09-07T21:28:01.888Z"
  timeShiftBufferDepth="PT24.0S"
  minBufferTime="PT9.6S">
  <ProgramInformation>
  </ProgramInformation>
  <Period id="0" start="PT0.0S">
    <AdaptationSet id="0" contentType="audio" segmentAlignment="true" bitstreamSwitching="
        true">
      <Representation id="2" mimeType="audio/mp4" codecs="mp4a.40.2" bandwidth="69000"
          audioSamplingRate="48000">
        <AudioChannelConfiguration schemeIdUri="
            urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="1" />
        <SegmentTemplate timescale="1000000" duration="5000000" initialization="init-
            stream1-$RepresentationID$.m4s" media="chunk-stream-$RepresentationID$-$Number
            %05d$.m4s" startNumber="1">
        </SegmentTemplate>
      </Representation>
    </AdaptationSet>
```

```
    <AdaptationSet id="1" contentType="video" segmentAlignment="true" bitstreamSwitching="
        true">
      <Representation id="0" mimeType="video/mp4" codecs="avc1.640028" bandwidth="1400000"
          width="1280" height="720" frameRate="25/1">
        <SegmentTemplate timescale="1000000" duration="5000000" initialization="init-
            stream1-$RepresentationID$.m4s" media="chunk-stream-$RepresentationID$-$Number
            %05d$.m4s" startNumber="1">
        </SegmentTemplate>
      </Representation>
      <Representation id="1" mimeType="video/mp4" codecs="avc1.640028" bandwidth="300000"
          width="160" height="90" frameRate="25/1">
        <SegmentTemplate timescale="1000000" duration="5000000" initialization="init-
            stream1-$RepresentationID$.m4s" media="chunk-stream-$RepresentationID$-$Number
            %05d$.m4s" startNumber="1">
        </SegmentTemplate>
      </Representation>
    </AdaptationSet>
  </Period>
</MPD>
```

A.8 Service creation

The configuration of the service consisted in creating the file `raspilive.service` inside the folder `/etc/systemd/system` with the following contents:

```
[Unit]
Description=RaspiLive DASH content generation
After=network.target
StartLimitIntervalSec=0

[Service]
Restart=always
RestartSec=1
ExecStart=/usr/local/bin/ffmpeg -f alsa -sample_rate 48000 -channels 1 -i hw:1 -c:a aac -
    re -f v4l2 -s:v 1280x720 -r:v 25 -i /dev/video0 -pix_fmt yuv420p -c:v h264_omx -map 1:
    v -b:v:0 1400k -profile:v high -bf 0 -refs 3 -bufsize 1400k -c:v h264_omx -map 1:v -b:
    v:1 300k -profile:v high -bf 0 -refs 3 -bufsize 300k -filter:1 "scale=160:90" -map 0:a
     -use_timeline 0 -media_seg_name 'chunk-stream-$RepresentationID$-$Number%05d$.m4s' -
    init_seg_name 'init-stream1-$RepresentationID$.m4s' -window_size 5  -extra_window_size
     10 -remove_at_exit 1 -adaptation_sets "id=0,streams=a id=1,streams=v" -f dash /var/
    www/html/stream.mpd

[Install]
WantedBy=multi-user.target
```

This command was run to enable the execution of the content-generating FFMPEG command every time the Raspberry Pi booted up:

```
systemctl enable raspilive
```

The device was then rebooted.

The following commands (not to be executed now) can be used to start or stop our service if needed:

```
systemctl start raspilive
systemctl stop raspilive
```

`sudo`	Used in Linux (hence in Raspbian) to execute the command as superuser
`ffmpeg`	Calls the FFMPEG executable file.
`-f alsa`	The input format (i.e. the microphone input) is ALSA.
`-sample_rate 48000`	Sets the microphone sample rate to 48 kHz.
`-channels 1`	Sets the number of audio channels to 1 (due to the nature of microphones).
`-i hw:1`	Selects the sound device corresponding to card number 1 as the input device.
`-c:a aac`	Selects AAC as the audio codec.
`-re`	Used in order to read from the input at the native frame rate.
`-f v4l2`	The input format (i.e. the camera input) is Video for Linux 2.
`-s:v 1280x720`	Sets the camera resolution to 1280×720 px.
`r:v 25`	Sets the camera framerate to 25 fps.
`-i /dev/video0`	Specifies the input file (the camera in this case).
`-pix_fmt yuv420p`	Specifies the source pixel format (YUV with 4:2:0 sampling).
`-c:v h264_omx`	Selects OpenMAX as the encoding library. This library encodes H.264 video using hardware acceleration.
`-map 1:v`	Maps the video streams of the second input to the only output stream.
`-b:v:0 1400k`	Forces an average bitrate of 1400 kbps for the first video stream.
`-profile:v high`	Selects the profile "high" for the output stream from among all the MPEG profiles.
`-bf 0`	Sets the number of consecutive B frames to 0 (i.e. no B frames).
`-refs 3`	Sets the maximum number of frames that can be referenced when encoding to 3.
`-bufsize 1400k`	Sets the size of the buffer for hardware encoding.
`-c:v h264_omx`	Selects OpenMAX as the encoding library for the second output stream. This library encodes H.264 video using hardware acceleration.
`-map 1:v`	Maps the video streams of the second input to the only output stream.
`-b:v:1 300k`	Forces an average bitrate of 300 kbps for the second video stream.
`-profile:v high`	Selects the profile "high" for the output stream from among all the MPEG profiles.
`-bf 0`	Sets the number of consecutive B frames to 0 (i.e. no B frames).
`-refs 3`	Sets the maximum number of frames that can be referenced when encoding to 3.
`-bufsize 300k`	Sets the size of the buffer for hardware encoding.
`-filter:1 "scale=160:90"`	This filter reduces the resolution up to 160×90 px.
`-map 0:a`	Maps the audio stream of the second input to the only output stream.
`-use_timeline 0`	Disables the use of `SegmentTimeline`.
`-media_seg_name 'chunk-stream-$RepresentationID$-$Number%05d$.m4s'`	Specifies the name of the media segments.
`-init_seg_name 'init-stream1-$RepresentationID$.m4s'`	Specifies the name of the inizialization segment.
`-window_size 5`	The manifest file will keep a maximum of 5 video segments.
`-extra_window_size 10`	Besides the 5 segments in the manifest, 10 more segments will be kept in the Raspberry Pi before they are deleted.
`-remove_at_exit 1`	Forces file removal when execution is stopped.
`-adaptation_sets "id=0, streams=a id=1,streams=v"`	Places the audio stream in one adaptation set and both video streams in another one.
`-f dash`	The output container format is DASH. This is not to be confused with the codec or the file format of the video stream.
`/var/www/html/stream.mpd`	Specifies the filename of the output.

Table A.1: Analysis of the final command generating a DASH stream

APPENDIX B
Web application

The web application is contained in three files: an HTML file, a CSS file with the styles and a JavaScript file with some functions, besides other auxiliary files accessed through the Internet.

raspilive.html:

```
<!DOCTYPE html>

<html>

  <head>

    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <link rel="shortcut icon" type="image/png" href="img\RaspiLive_favicon.png"/>

    <title>RaspiLive</title>

    <link href="https://fonts.googleapis.com/css?family=Roboto&display=swap" rel="
        stylesheet">
    <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
    <link href="https://code.getmdl.io/1.3.0/material.blue-red.min.css" rel="stylesheet">

    <link href="raspilive_style.css" rel="stylesheet">

    <script src="https://code.getmdl.io/1.3.0/material.min.js"></script>
    <!-- Shaka Player compiled library: -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/shaka-player/2.1.4/shaka-player.
        compiled.debug.js"></script>

    <script src="raspilive_script.js"></script>

  </head>

  <body>

    <main>

      <header class="header mdl-shadow--6dp">
        <img src="img\RaspiLive_header.svg" height="75">
      </header>

      <div class="flex-container wrapper">

        <section class="content mdl-card mdl-cell mdl-shadow--4dp">
          <canvas id="main_player" width="0" height="0">
          </canvas>
```

```
          <p id="selected_descr">[No camera is selected]</p>
        </section>

        <aside class="sidebar mdl-card mdl-cell mdl-shadow--4dp">
          <div id="fixed_subsidebar">
            <button id="hideable_form_button" onclick="toggleForm()" class="mdl-button mdl
                -js-button mdl-button--icon mdl-button--primary mdl-js-ripple-effect">
              <i class="material-icons">
                add
              </i>
            </button>
            <div class="hideable-form" id="hideable_form">
            <!-- <div class="hideable-form mdl-textfield mdl-js-textfield" id="
                hideable_form"> -->
              <div class="mdl-textfield mdl-js-textfield mdl-textfield--floating-label">
                <!-- <input type="text" id="ip_text_box" placeholder="Video source address
                    "> -->
                <input type="text" id="ip_text_box" class="mdl-textfield__input material-
                    input">
                <label for="ip_text_box" class="mdl-textfield__label">Video source address
                    </label>
              </div>
              <div class="mdl-textfield mdl-js-textfield mdl-textfield--floating-label">
                <!-- <input type="text" id="descr_text_box" placeholder="Video source
                    description"> -->
                <input type="text" id="descr_text_box" class="mdl-textfield__input
                    material-input">
                <label for="descr_text_box" class="mdl-textfield__label">Video source
                    description</label>
              </div><br>
              <button id="add_source_button" onclick="addSource()" class="mdl-button mdl-
                  button--icon mdl-button--primary mdl-js-button mdl-js-ripple-effect"
                  disabled>
                <i class="material-icons">
                  send
                </i>
              </button>
            </div>
          </div>
          <div id="scrollable_subsidebar" class="scrollbar">
          </div>
        </aside>

      </div>

      <footer class="footer mdl-shadow--6dp">
        <div class="footer-item-title">Development of a live video streaming system using
            Raspberry Pi and the DASH protocol</div>
        <div class="footer-item-other">Eloy Sanchis Lopez, 2019, ETSIT - UPV</div>
      </footer>

    </main>

  </body>

</html>
```

raspilive_style.css:

```
/*GENERAL APPEARANCE AND FOOTER AND HEADER POSITION*/

*, :after, :before {
  box-sizing: border-box;
}
```

```
html {
  background: #eeeeee;
}

body {
  margin: 0;
  padding: 0;
  font-family: 'Roboto', sans-serif;
  color: #333333;
}

main, header, section, aside, footer {
  display: block;
  margin: 0;
}

main {
  padding: 7em 1.25rem 3em 1.25rem;
  color: #333333;
}

.header {
  background: #2196f3;
  padding: 0.4em;
  color: #ffffff;
  text-align: center;
  font-weight: bold;
  width: 100%;
  position: fixed;
  left: 0;
  top: 0;
  z-index: 2;
}

.content {
  flex: 1;
  padding: 1.25rem;
  color: #333333;
  min-height: 0;
  width: auto;
}

.sidebar {
  flex: 0 0 400px;
  padding: 1.25rem;
  color: #333333;
  min-height: 0;
  width: auto;
}

.footer {
  background: #37547a;
  padding: 0.4em;
  color: #ffffff;
  text-align: center;
  width: 100%;
  position: fixed;
  left: 0;
  bottom: 0;
  z-index: 2;
  display: flex;
  justify-content: space-around;
  align-items: center;
}
```

```
.footer-item-title {
  flex: 2 1;
}

.footer-item-other {
  flex: 1 1;
}

p {
  margin: 10px 0 0 0;
  font-size: 120%;
}

#selected_descr {
  font-size: 150%;
  margin: 10px 0 0 0;
}

input {
  width: 100%;
}

.material-input {
  font-family: 'Roboto', sans-serif;
}

video {
  border-style: solid;
  border-width: 3px;
  border-color: rgba(8, 172, 239, 0.0);
  box-sizing: border-box;
  margin: 5px;
}

/*SCROLLING THINGS, APPEARING ICONS AND FORMS, ETC.*/

.wrapper {
  margin: auto;
  max-width: 1160px;
}

#hideable_form {
  display: none;
}

.thumb-image {
  position: relative;
}

.top-right {
  position: absolute;
  top: 8px;
  right: 16px;
}

@media screen and (min-width: 640px) {

  .flex-container {
    display: flex;
    align-items: flex-start;
  }

  #scrollable_subsidebar {
```

```
    overflow-y: scroll;
    max-height: calc(100vh - 227px);
  }

  .thumb-item {
    display: flex;
    flex-direction: row;
  }

  .scrollbar {
    overflow-y: scroll;
  }

  .scrollbar::-webkit-scrollbar-track {
    background-color: #FFFFFF;
  }

  .scrollbar::-webkit-scrollbar {
    width: 4px;
    background-color: #FFFFFF;
  }

  .scrollbar::-webkit-scrollbar-thumb {
    background-color: #777777;
  }

  .footer {
    flex-direction: row;
  }

}

@media screen and (max-width: 639px) {

  .wrapper {
    margin-bottom: 100px;
  }

  #scrollable_subsidebar {
    flex-grow: 0;
    display: flex;
    overflow-x: scroll;
  }

  .thumb-item {
    max-width: 170px;
  }

  .footer {
    flex-direction: column;
  }

  .footer-item-title {
    margin: 5px;
  }

  .footer-item-other {
    margin: 5px;
  }

}

@media (hover: hover) {
```

```
  .top-right {
    display: none;
  }

  .thumb-image:hover .top-right {
    display: block;
  }

}
```

raspilive_script.js:

```
// EVENTS, VARIABLES AND INITIALIZATION

var thumbnail_players = [];
var descriptions = [];
var selected_thumbnail;
var main_player_interval;

var ip_box;
var descr_box;
var form;
var show_form_button;
var add_source_button;
var scr_subsidebar;
var remove_div;
var canvas;
var context;
var selected_descr;

document.addEventListener('DOMContentLoaded', initApp);
window.addEventListener('resize', resizeMainPlayer, false);

function initApp() {
  // Get elements
  ip_box = document.getElementById('ip_text_box');
  descr_box = document.getElementById('descr_text_box');
  form = document.getElementById('hideable_form');
  show_form_button = document.getElementById('hideable_form_button');
  add_source_button = document.getElementById('add_source_button');
  scr_subsidebar = document.getElementById('scrollable_subsidebar');
  canvas = document.getElementById('main_player');
  context = canvas.getContext('2d');
  selected_descr = document.getElementById('selected_descr');
  // Events
  ip_box.addEventListener('keyup', formKeyPress);
  descr_box.addEventListener('keyup', formKeyPress);
  // Other
  initializeShaka ();
  setMainPlayerSize ();
}

function initializeShaka () {
  // Install built-in polyfills to patch browser incompatibilities.
  shaka.polyfill.installAll();
  // Check to see if the browser supports the basic APIs Shaka needs.
  if (shaka.Player.isBrowserSupported()) {
    // Everything looks good!
    console.log('Shaka browser support: OK');
  } else {
    // This browser does not have the minimum set of APIs we need.
    console.error('Browser not supported!');
  }
}
```

```
function onErrorEvent(event) {
  // Extract the shaka.util.Error object from the event.
  onError(event.detail);
}

function onError(error) {
  // Log the error.
  console.error('Error code', error.code, 'object', error);
}

// FORM

function toggleForm () {
  var media_query = window.matchMedia('(min-width: 640px)');
  if (form.style.display == 'none' | form.style.display == '') {
    form.style.display = 'block';
    show_form_button.firstElementChild.innerText = 'clear';
    if (media_query.matches) {
      scr_subsidebar.style.maxHeight = 'calc(100vh - 399px)';
    }
  } else {
    form.style.display = 'none';
    show_form_button.firstElementChild.innerText = 'add';
    if (media_query.matches) {
      scr_subsidebar.style.maxHeight = 'calc(100vh - 227px)';
    }
  }
}

function formKeyPress () {
  if (ip_box.value == '') {
    add_source_button.setAttribute('disabled');
  } else {
    add_source_button.removeAttribute('disabled');
  }
  if (event.keyCode === 13) {
    add_source_button.click();
  }
}

// THUMBNAILS

function addSource() {
  var new_ip = checkInputChars (ip_box.value);
  var new_descr = checkInputChars (descr_box.value);
  if (new_ip != '') {
    //Add HTML elements
    var next_index = thumbnail_players.length;
    addElement('scrollable_subsidebar', 'div', 'thumb_item_' + next_index, '');
    document.getElementById('thumb_item_'+next_index).setAttribute('class', 'thumb-item');
    addElement('thumb_item_' + next_index, 'div', 'thumb_image_' + next_index, '');
    document.getElementById('thumb_image_'+next_index).setAttribute('class', 'thumb-image'
        );
    addElement('thumb_image_' + next_index, 'video', 'thumbnail_' + next_index, '');
    document.getElementById('thumbnail_'+next_index).setAttribute('width', '160');
    document.getElementById('thumbnail_'+next_index).setAttribute('autoplay', '');
    document.getElementById('thumbnail_'+next_index).setAttribute('onclick', 'selectSource
        (this)');
    document.getElementById('thumbnail_'+next_index).muted = true;
    addElement('thumb_image_' + next_index, 'button', 'remove_button_' + next_index, '');
    document.getElementById('remove_button_'+next_index).setAttribute('class', 'top-right
        mdl-button mdl-button--icon mdl-button--accent mdl-js-button mdl-js-ripple-effect'
        );
```

```
    document.getElementById('remove_button_'+next_index).setAttribute('onclick', '
        removeSource(this)');
    addElement('remove_button_' + next_index, 'i', 'remove_button_icon_' + next_index, '
        delete');
    document.getElementById('remove_button_icon_'+next_index).setAttribute('class', '
        material-icons');
    addElement('thumb_item_' + next_index, 'p', 'thumb_descr_' + next_index, new_descr);
    descriptions[next_index] = new_descr;
    // Create a Player instance.
    thumbnail_players[next_index] = new shaka.Player(document.getElementById('thumbnail_'
        + next_index));
    // Listen for error events.
    thumbnail_players[next_index].addEventListener('error', onErrorEvent);
    // Try to load a manifest.
    thumbnail_players[next_index].load('http://'+new_ip+'/stream.mpd').then(function() {
      // This runs if the asynchronous load is successful.
      console.log('The video has now been loaded!');
    }).catch(onError);  // onError is executed if the asynchronous load fails.
  }
}

function removeSource (image) {
  var remove_index = parseInt(image.id.substring(14));
  console.log('Source ' + remove_index + ' removed');
  // Stop playback from manifest and remove player
  thumbnail_players[remove_index].unload();
  thumbnail_players[remove_index].destroy();
  thumbnail_players[remove_index] = undefined;
  // Remove HTML code
  var remove_div = document.getElementById('thumb_item_' + remove_index);
  remove_div.parentNode.removeChild(remove_div);
  // Stop playback in main player if selected source is removed
  if (remove_index == selected_thumbnail) {
    clearInterval (main_player_interval);
    context.clearRect(0, 0, canvas.width, canvas.height);
    selected_descr.innerHTML = '[No camera is selected]';
    selected_thumbnail = undefined;
  }
}

// MAIN PLAYER

function selectSource (thumbnail) {
  if (selected_thumbnail !== undefined && selected_thumbnail !== null) {
    document.getElementById('thumbnail_'+selected_thumbnail).style.borderColor = 'rgba(8,
        172, 239, 0.0)';
    document.getElementById('thumbnail_'+selected_thumbnail).muted = true;
  }
  if (main_player_interval !== undefined && main_player_interval !== null) {
    clearInterval (main_player_interval);
  }
  console.log('Source ' + thumbnail.id.substring(10) + ' selected');
  selected_thumbnail = parseInt(thumbnail.id.substring(10));
  document.getElementById('thumbnail_'+selected_thumbnail).style.borderColor = 'rgba(8,
      172, 239, 1.0)';
  document.getElementById('thumbnail_'+selected_thumbnail).muted = false;
  setMainPlayerSize ();
  main_player_interval = setInterval (updateMainPlayer, 40, document.getElementById('
      thumbnail_'+selected_thumbnail), context, canvas.width, canvas.height);
  if (descriptions [selected_thumbnail] == '') {
    selected_descr.innerHTML = '[No description]';
  } else {
    selected_descr.innerHTML = descriptions [selected_thumbnail];
  }
```

```
}

function updateMainPlayer (v, c, w, h) {
  if (v.paused || v.ended) {
    return false;
  }
  c.drawImage(v,0,0,w,h);
}

function setMainPlayerSize () {
  var main_player_width;
  var main_player_height;
  if (window.innerWidth > 1200) {
    main_player_width = 696;
  } else if (window.innerWidth < 640) {
    main_player_width =  window.innerWidth - 86;
  } else {
    main_player_width = window.innerWidth - 502;
  }
  main_player_height = Math.round(main_player_width * 9 / 16);
  canvas.setAttribute('width', main_player_width);
  canvas.setAttribute('height', main_player_height);
  console.log('Window: ' + window.innerWidth + ' x '+ window.innerHeight);
  console.log('Main player: ' + canvas.width + ' x ' + canvas.height);
}

function resizeMainPlayer () {
  setMainPlayerSize ();
  if (main_player_interval !== undefined && main_player_interval !== null) {
    clearInterval (main_player_interval);
  }
  if (selected_thumbnail !== undefined && selected_thumbnail !== null) {
  main_player_interval = setInterval (updateMainPlayer, 40, document.getElementById('
      thumbnail_'+selected_thumbnail), context, canvas.width, canvas.height);
  }
}

// AUXILIARY FUNCTIONS

function addElement (parentId, elementTag, elementId, html) {
  // Adds an element to the document
  var p = document.getElementById(parentId);
  var newElement = document.createElement(elementTag);
  if (elementId != '') {
    newElement.setAttribute('id', elementId);
  }
  newElement.innerHTML = html;
  p.appendChild(newElement);
}

function checkInputChars (input_string) {
  input_string = input_string.replace(/&/g, '&amp;');
  input_string = input_string.replace(/</g, '&lt;');
  input_string = input_string.replace(/>/g, '&gt;');
  return input_string;
}
```