Júlio Martins Santos

Grau de Mestre em Engenharia Eletrotécnica e de Computadores.

Portable Video Streaming Network

Dissertação para obtenção do Grau de Mestre em

Engenharia Eletrotécnica e de Computadores

Orientador: Luís Filipe Lourenço Bernardo, Professor associado,
NOVA School of Science and Technology

Júri

Presidente:	Nuno Filipe Silva Veríssimo Paulino
Arguente:	Pedro Miguel Figueiredo Amaral
Vogal:	Luís Filipe Lourenço Bernardo

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Junho, 2021

Acknowledgements

Throughout the writing of this dissertation, I have received a great deal of support and assistance.

I would like to thank my advisor, Professor Luís Bernardo, whose expertise was invaluable in formulating the research questions and methodology.

I am deeply grateful to André Palma for his help in the development of the thesis and his knowledge sharing on the subject.

I wish to express my sincere appreciation to my family for their support and love, my mother, Sofia, my brother, Guilherme, and my father, Luís. They kept me going on and moving forward even when I did not want to.

This work was partially funded by FCT/MCTES through national funds and when applicable co-funded by FEDER – PT2020 partnership agreement under the project UID/EEA/50008/2019.

Lastly, I would like to show my gratitude to all my friends for their advice, companionship, and for our good moments together.

Abstract

This dissertation addresses the challenge of developing a video call system capable of supporting both Android mobile devices and fixed computers. Additionally, it analyses the quality of video achieved and its variation in the presence of network bandwidth and packet loss constraints.

A prototype of a video call system was implemented using a web application and the Web Real-Time Communication (WebRTC) library. Clients use WebRTC to stream video over a Traversal Using Relays around NAT (TURN) relay server, allowing them to send video to any terminal connected to the Internet. Signalling was implemented using WebSockets and a Node.js server.

A quality testing prototype was also implemented, which supports sending pre-recorded videos and capturing and storing video recordings at the sender and receiver. The Video Multimethod Assessment Fusion (VMAF) metric was used as the main video quality metric, based on the comparison between the transmitted and received videos.

The quality of a video encoded using the open source video encoder VP8 was analysed in constrained network setups. The results measured the video quality degradation and percentage of received frames, showing that the system is resilient to some bandwidth strangulation and packet loss, although with a noticeable video quality degradation.

Keywords: Video call software, Video quality, Web application, WebRTC.

Resumo

Esta dissertação aborda o desafio de desenvolver um sistema de videochamada capaz de suportar dispositivos móveis Android e computadores fixos. Além disso, analisa a qualidade do vídeo obtida e sua variação na presença de restrições de largura de banda da rede e perda de pacotes.

Um protótipo de um sistema de videochamada foi implementado usando uma aplicação web e a biblioteca Web Real-Time Communication (WebRTC). Os clientes usam WebRTC para transmitir o vídeo através de um servidor de retransmissão Traversal Using Relays around NAT (TURN), permitindo que enviem vídeo a qualquer cliente ligado à Internet. A sinalização foi implementada usando WebSockets e um servidor Node.js.

Também foi implementado um protótipo de teste de qualidade, que suporta o envio de vídeos pré-gravados e a captura e armazenamento de gravações de vídeo no emissor e no recetor. A métrica Video Multimethod Assessment Fusion (VMAF) foi utilizada como a principal métrica de qualidade de vídeo, com base na comparação entre os vídeos transmitidos e recebidos.

A qualidade de um vídeo codificado usando VP8 foi analisada em configurações de rede com limitações. Os resultados mediram a degradação da qualidade do vídeo e a percentagem de tramas recebidas, mostrando que o sistema é resiliente a algum estrangulamento da largura de banda e perda de pacotes, embora com uma degradação percetível da qualidade do vídeo.

Palavras-chave: Software de vídeo chamada, Qualidade de vídeo, Aplicação Web, WebRTC

Contents

List of Figures

List of Tables

Acronyms

AOMedia Alliance for Open Media

CABAC Context-adaptive Binary Arithmetic Coding

CAVLC Context-adaptive VLC

CC Contributing Source Identifier Count

CODEC coder-decoder

CU Coding Unit

HAS HTTP Adaptive Streaming

HLS HTTP Live Streaming

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IDTX Identity Transform

IP Internet Protocol

MPD Media Presentation Description

MPEG-DASH Dynamic Adaptive Streaming over HTTP

NAL Network Abstraction Layer

OMBC Overlapped Block

PSNR Peak signal-to-noise ratio

PU Prediction Unit

QUIC Quick UDP Internet Connections

RTCP RTP control protocol

RTMP Real-Time Messaging Protocol

RTP Real-time transport protocol

SSIM Structural similarity index measure

SSL Secure Socket Layer

TCP Transmission Control Protocol

TM True Motion

TU Transform Unit

TURN Traversal Using Relays around NAT

UDP User Datagram Protocol

VCL Video Coding Layer

VMAF Video Multimethod Assessment Fusion

WebRTC Web Real-Time Communication

Chapter 1

Introduction

As the capabilities of the cell phones increase, so does the amount of things we can do with them, which includes the ability to process data in a fast manner. We can live stream with better and better quality, and with that added video quality comes the challenge to process and transport the data in a faster way, allowing us to watch video anywhere in the world smoothly and without delays. With that comes the necessity of improving the existing protocols or creating new ones.

The term streaming is commonly used to talk about the technique of carrying data over a computer network at a steady and continuous pace, permitting it to be played before all the data that has been received. An example of streaming is the platform Netflix which allows the user to watch movies-on-demand. Another example is Twitch which allows the user to watch live streams of video games.

1.1 Motivation

Due to the huge demands for bandwidth on the Internet, and improvements in video compression coding, video streaming has become one of the priorities for study. With the appearance of video streaming, it became possible to do live video calls using computers, mobile phones and tablets.

With the increase in the demands for simple and intuitive video call systems, the need to test these systems' quality appeared. The video call system is tested with different bandwidths and with different percentages of packets lost for loss of video quality and response.

1.2 Goals and Objectives

The objective of this thesis is to explore and identify practical ways to deploy a mobile live streaming facility using smartphones. This includes studying the software libraries available and the video streaming protocols and coder-decoder (CODEC)s supported and defining optimal configurations that improve the video transmission quality of experience.

The second objective is to deploy a quality of service system that can evaluate the mobile live streaming through video comparison and quality metrics.

1.3 Contributions

The main results and contributions from the dissertation are:

1. Development of the Video Call system for peer-to-peer video communication;
2. Development of the Video Testing system to test the quality of the Video Call system;
3. Analyzing the influence of bandwidth strangulation and packet loss on video quality.

1.4 Thesis Structure

The dissertation structure is as follows:

Chapter 2 provides a literature overview of the different protocols for streaming and the most important CODECs in use today. This chapter presents a short introduction to Quick UDP Internet Connections (QUIC), Dynamic Adaptive Streaming over HTTP (MPEG-DASH), HTTP Live Streaming (HLS) and to the CODECs H.264, H.265, VP8, AV1 as well as a comparision of the performance of all these CODECs in different enviroments.

Chapter 3 presents a description of the development done for each part of the Video Call application. This chapter begins with an explanation of the Signalling server, followed by the Users' webpage and its back end development, and lastly explains the relay server.

Chapter 4 displays the development process and materials used for the Quality Testing system and how it was used to test the Video Call application. This chapter explains the reference video used, how it was obtained, a description of

the Recording system and the Quality Comparing system, a brief overview of the quality metrics used, and the results of the testing done.

Chapter 5 provides the concluding remarks and potencial future work for improvements on the subject.

Chapter 2

State of the Art

2.1 Introduction

The focus of this chapter is the review done for the thesis about Portable Video Streaming. While there are more protocols and CODECs, the subjects of the research are the ones that are most used and that have a promising future.

By analyzing and comparing Video Developer Report 2017 [14] and Video Developer Report 2019[7], there were multiple facts found that are be explained in more detail while going through each area in this state-of-the-art.

While going through the graphics in Figure 2.2 of the 2019 report[7] and comparing it with the one in Figure 2.1 of the other report [14], one thing that can be taken from them is that HLS continues to be the most used streaming protocol although MPEG-DASH is catching up to it. The third most used streaming is still RTMP, but it has not grown much in percentage compared with the other ones which seem to be the most important for the future.

All these protocols are Streaming Protocols, although different, they share the same objective which is to allow users the ability to watch live streams or videos on demand, by delivering the necessary data over the Internet.

The other important graphics in Video Developer Report 2017 [14] and Video Developer Report 2019 [7] were the ones referring to the video CODECs since these are used for compressing and decompressing video to make easier transportation from one place to another. By analyzing the complementary texts of the articles, it was found that multiple video CODECs are in use, being H.264, the most used, in second comes a newer version of H.264 called H.265, and in third

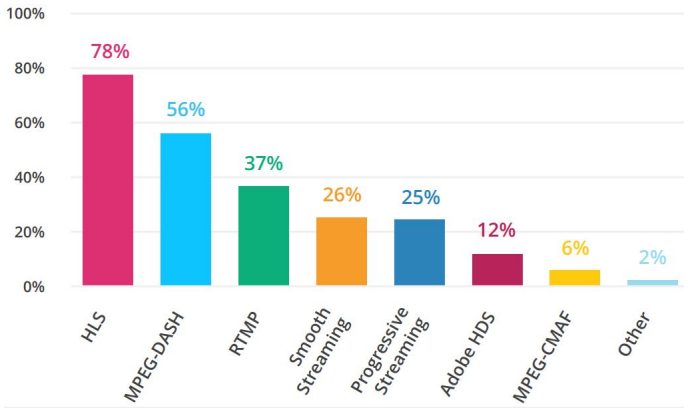


Figure 2.1: Streaming Protocols. Adapted from [14].

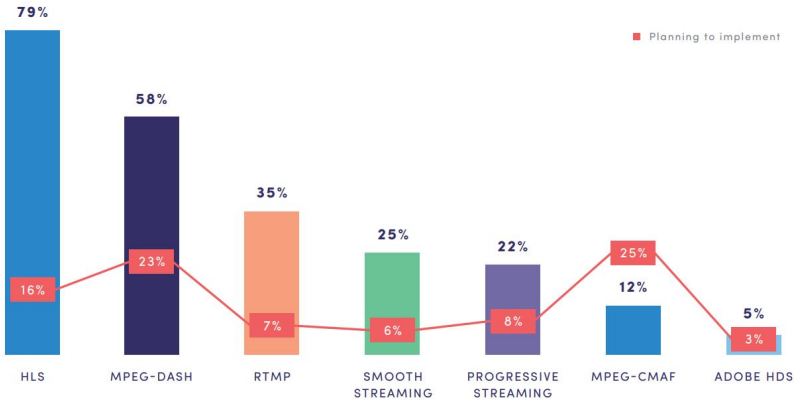


Figure 2.2: Streaming Protocols. Adapted from [7].

comes MPEG2. Although it only comes in fifth, AV1 is also considered since it is a recent royalty-free video CODEC with a lot of potentials and according to [7] , it will replace VP9 and is well-positioned to compete with H.265.

The research will start by describing the relevant Internet transport protocols QUIC, Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) since those are the base of data the transport of data on the Internet.

2.2 Internet Protocols

The review starts by presenting two essential transport protocols, TCP and UDP.

2.2.1 TCP

Transmission Control Protocol (TCP) is currently one of the main protocols in use in the Internet. It was created as an addition to the Internet Protocol (IP) and is commonly known as TCP/IP. TCP permits reliable ordered, and error-checked

delivery of a row of bytes between applications running on hosts communicating through an IP network.

TCP is a connection-oriented protocol, which means that the connection is established before data is sent from one user to another. And by that, it means that both parts must meet first and handshake, before the data can start go through.

2.2.2 UDP

User Datagram Protocol (UDP) is another of the main protocols in use on the Internet. This protocol is connectionless, not requiring the establishment of a connection between two endpoints, like in TCP. This allows for faster communication at the expense of the guarantees that the data reaches the right receiver, with the right order and without duplication of packets. UDP is a good protocol when error checking, correction, and flow control are not demanded by the application. Time-sensitive applications, like live streaming applications, usually use UDP since it does not introduce delay due to waiting for packets that are not confirmed and need to be retransmitted.

2.2.3 QUIC

Quick UDP Internet Connections (QUIC) is an internet transport and application protocol which is encrypted by default that was created with a few objectives in mind, like security, deployability, and reduction in handshake and head-of-line blocking delays [13]. The protocol merges the handshakes from cryptographic and transport layers to allow lower setup RTTs. It multiplexes the different requests/responses over a single connection by providing each with its own space so that no response can be blocked by another. QUIC encrypts and authenticates packets to avoid tampering by third parties. It improves loss recovery by using unique packet numbers to avoid retransmission problems and by using precise signaling in ACKs for accurate RTT dimensions. QUIC allows connections to migrate across an IP address instead of the IP/port 5-tuple. Finally, provides flow control to limit the amount of data buffered at a slow receiver and ensures that a single stream flow control limit.

2.2.3.1 Design

QUIC is split into multiple parts. A short explanation about the most important subjects is provided, starting with the establishment of connections [13].

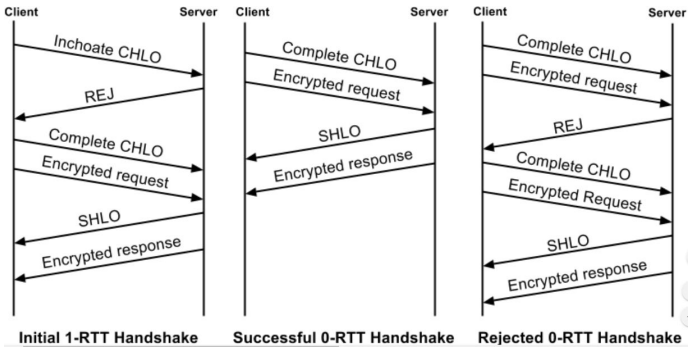


Figure 2.3: Connection Establishment. Adapted from [13].

At the start, the user does not have any information about the server, so it sends a greeting message (CHLO), to which the server responds with a rejection message (REJ) that contains the necessary information about it, like the server config and source-address token. With those, the user sends the complete CHLO and an encrypted request to which the server, if approved, will reply with an SHLO message and an encrypted response, like shown in Figure 2.3 [13].

Users and the servers do a version negotiation while making the connection establishment to avert delays. The user proposes a version to use for the connection in the first packet and encodes the handshake with that version. If the server does not work with that version, it sends a Version Negotiation packet to the user, that has all the supported versions of the server.

QUIC does stream multiplexing by joining multiple streams in the same connections, ensuring that a lost UDP packet only impacts the data in that packet, since other data continues to be delivered. These streams are lightweight abstractions that permit a meaningful bidirectional byte stream. Which means that it can be used a part from a few handshakes and reset packets. QUIC packets are fully authenticated and mostly encrypted. The parts of the packet that are not encrypted are needed either for routing or decrypting the packet.

The way QUIC deals with retransmission ambiguity, which is a big problem in TCP, is by putting different packet numbers in all packets including the ones corresponding to retransmission, so it always knows which packet was lost and to which packet that ACK corresponds to [13].

To have better flow control, QUIC limits the buffer that a single stream can absorb when the data absorption is slow, so it can avoid consuming the entire connection. QUIC uses two flow controls, one at the connection level to limit the joint buffer that a sender can absorb throughout all streams and another control in the stream level to limit the buffer that a sender absorbs in any stream.

To allow the identification of a QUIC connection it was created a 64-bit Connection ID. This ID allows changing the IP and port of an endpoint without the loss of connection caused by timeouts and rebinding, or even the change of networks [13].

2.3 Streaming Protocols

Streaming Protocols are standardized techniques for delivering video over the Internet. The protocols explain the techniques used for sending parts of the content from one place to another and explain how to merge the parts back together to reform the content at its destination.

There are two main types of Streaming Protocols, the traditional ones, and Hypertext Transfer Protocol (HTTP) based adaptive ones. The main differences between them are shown in Table 2.1.

Table 2.1: Comparing Traditional streaming protocols with HTTP based streaming protocols. Adapted from [4].

	Traditional streams	HTTP adaptive streams
Communication protocols	RTSP, RTP, UDP	HTTP, RTMP, FTP
Adaptation logic runs at	Server side	Client side
Transmission data units	Packets	Media segments
Multicast support	Yes	No
Caching support	Protocol specific	Web caches as used for HTTP

The traditional Real-time transport protocol (RTP) protocol is presented first, followed by the HTTP Adaptive Streaming (HAS) protocols.

Traditional Protocols

2.3.1 RTP

The RTP is a transport protocol that runs on the application layer, used for end-to-end purposes, like live streaming. RTP is a lightweight protocol, consisting only of the data message and having a so small number of features that it must run over another transport protocol like UDP. It works with the RTP control protocol (RTCP), and can be enhanced by control protocols like H.263 and SIP.

The focus of RTP is to verify, improve, and maintain, the quality of the transmitting data. The needs change depending on the application running but are not restrained only to good time deliveries of in-order, undamaged data. The effect of the Jitter on a video stream is the occurrence of distortions in the streams because

of gaps in the bursts of data and must be avoided by managing the delivery rate of data [2].

Figure 2.4 shows an RTP header with at least 12-bytes. The most important field is the timestamp, since it has the timestamp of the creation of the first byte of payload data. This stamp is used to resolve the jitter problems by being used for information.

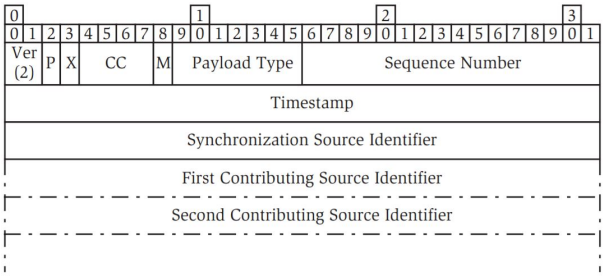


Figure 2.4: RTP Header. Adapted from [2].

The P bit is the bit that shows if the packet ends with one or more bytes of padding. The following bit is the X-flag and shows if the header has any private extensions. The sum of all the Contributing Source Identifiers attached to the header is called Contributing Source Identifier Count (CC). Then, there is the M-flag which boosts the management of the data by synchronizing events [2].

For control, multiple protocols can be used, but in this review, only RTCP is summarized. RTCP has three main functions: The first is to allow the registration of the presence and the leaving of participants in an RTP session. The second one is to observe data traffic and give information about the quality of the service delivered. Application-specific information can also be carried by RTCP [2].

2.3.2 RTMP

Real-Time Messaging Protocol (RTMP) is an application-level network protocol developed with the intention of streaming and delivering on-demand video, audio, and data through the Internet, created by Adobe. RTMP uses TCP has its base with the intent of maintaining resilient connections. To allow low-latency communications, it delivers streams steadily and pass as much data as possible. It splits the data into fragments, and the size of these fragments is negotiated as needed by the user and the server. The default size for video fragments is 128 bytes, and these can also be multiplexed and interleaved during a single connection. In longer pieces of fragments, the protocol diminishes the size of the header to reduce the overhead [26].

Figure 2.5 shows the system view and the sequence of events that happen between the person streaming and the one playing the stream. First, the sender and the receiver must include a receiver application, so they can be able to use RTMP. Second, the server being used for transmission must support RTMP [15].

The user starts the Handshake process, as can be observed in Figure 2.6, with the client sending C0 and C1 block and the server responding with S0 and S1 blocks. After the client receives all the S0 and S1 blocks, he starts sending C2 block and after receiving all the C0 and C1, the server sends S2. When all these blocks are fully received by both sides the Handshake process is completed and the Connection process starts. This part starts, as is shown in Figure 2.7, with the client sending the "connect" command message to the server and this responding with the "Window Acknowledgement Size" message and connecting with the application mentioned in the command message. The server also sends the "Set Peer Bandwidth" message to the client to define the output bandwidth. After processing the Bandwidth message, the client sends the "Window Acknowledgement Size" message. To complete this part of the process, the server sends the "Stream Begin" message and the "_results" to show the client the results of the Commands [26] [19].

With this, the client can create a stream by sending the "CreateStream" message and the server responding with the "_results" message to inform the client about the results of the message. As the stream gets established, the Play process starts with the client sending a "play" message and the server responding with "Set Chunk Size" to alert the client of the size of chunk that is being used. It also sends the "StreamBegin" user control message so the client can know that the stream is running, and the "NetStream.Play.Start"and "NetStream.Play.reset" command messages to inform the client that the "play" was successful. We can see the Play process in figure 2.8. After all these processes are completed, the server starts sending audio and video data to the client [15].

RTMP is compatible with H.264, VP8 and VP6.

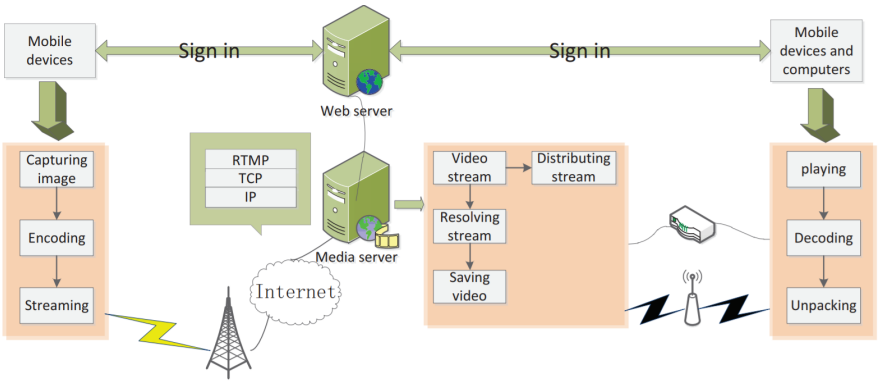


Figure 2.5: How the system works. Adapted from [26].

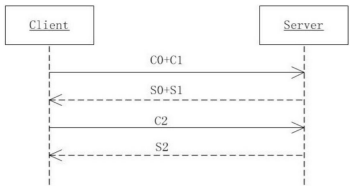


Figure 2.6: Handshake. Adapted from [15].

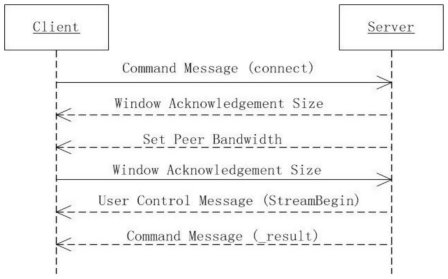


Figure 2.7: Connection. Adapted from [15].

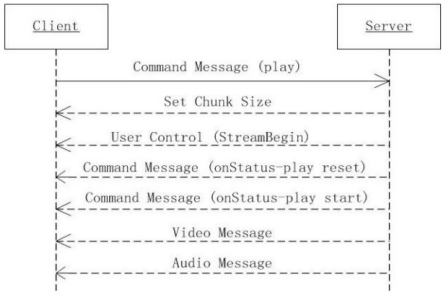


Figure 2.8: Play. Adapted from [15].

HTTP based Adaptive Protocols

2.3.3 HLS

HTTP Live Streaming (HLS) is a streaming protocol that is based on HTTP with an adaptive bitrate stream, that was created by Apple Inc. HLS is similar to MPEG-DASH since both work by dividing the overall stream into an array of small HTTP file downloads. Apple uses this streaming protocol as its main one.

HLS architecture is composed of three main elements: the server, the distribution, and the client [8].

- The server element has the responsibility of receiving media input streams and encoding and encapsulating them in a proper format for delivery and preparing this media for distribution.
- The client software determines the right media to request, downloading it and rebuild it so that it can be shown to the user.
- The distribution element contains standard web servers. These have the responsibility of receiving client requests and distributing the prepared media and needed resources to the client.

In the server element, it is needed a media encoder and a manner to break the encoded media into parts and preserve them as files like the media stream segmenter. This encoder receives a real-time signal from an audio-video device, encodes it and encapsulates it for distribution. This encoder can only work with certain video CODECs like H.264 and H.265.

The Stream segmenter is an operation that reads the Transport Stream from the local network and then partitions it into sets of small media files to the same extent. Although each part is a different file, these files, if combined, form a continuous stream in the sender, that can be reconstructed that the receiver. It also makes an index file that has references to each media file. Every time a media file is done at the segmenter, the index file is updated.

The distribution process is a web server that distributes the media files and index files to the clients over HTTP. There is no need for custom server modules and normally only a small configuration is needed on the web server.

The client starts by getting the index file, using the URL identifying the stream to find it. This index has the location of all accessible media files, decryption keys, and any alternative streams. The client downloads all the needed media files in a chain since the files have a consecutive part of the stream. After a certain extent of

media data has been downloaded, the client starts to observe the received stream. This process ends when the client finds the #EXT-X-ENDLIST tag in the index file [8].

HLS allows for both live and on-demand streams. In the live streams, new media files are formed, and the index file is always being updated to include these new files and the files that were already streamed get removed from the index file. The continuous updating index displays a moving window into a stream. For on-demand streams, the index file is static and has all the needed information already in it. The media files are already defined and available for download.

To create an on-demand video from a live broadcast, it is necessary that the index file adds new files and does not remove the old ones and put #EXT-X-ENDLIST tag at the end of the file [8].

For content protection, media files have stream segments that can be independently encrypted. If this encryption happens, references to the correct key files emerge in the index file. If a key file appears in the index file, this has a cipher key that must be used for decryption.

The media stream segmenter supports encryption and allows for three different modes. The first one permits you to define a path to a key file on the disk and the segmenter puts the URL of that key file in the index file. All media files will be encrypted using that key. The second one, tells the segmenter to create a random key file, guards it in a certain location and references it in the index file. Like in the first one all media is encrypted with this key. The third one, also tells the segmenter to create a random key file, and reference it, but then tells the segmenter to create a new key file for every n files. In this mode, the key differs according to the group the media file belongs to.

There is the possibility of Index files referencing alternative streams and these references can allow for the delivery of different streams with the same content, so the best one can be used for the client's bandwidth [8].

In case of failure in one stream, the client goes to the next highest bandwidth stream that the network can handle.

2.3.4 MPEG-DASH

Dynamic Adaptive Streaming over HTTP (MPEG-DASH) is an adaptive bitrate streaming technology that allows high-quality streaming of media content through the Internet by using normal HTTP web servers. The data is divided into multiples files, and each file is described by a media presentation description file that contains timing, URL, media CODEC, and certain characteristics [12].

Figure 2.9 serves as an example of how MPEG-DASH interacts with the HTTP servers. This figure shows how after being stored in the HTTP server, the content can be accessed and how it is divided inside the server, the Media Presentation Description (MPD) that details a manifest of the present content, the different alternatives, their URL. The other part of the content in the server is the segments which are the media bitstreams divided into groups [21] [4].

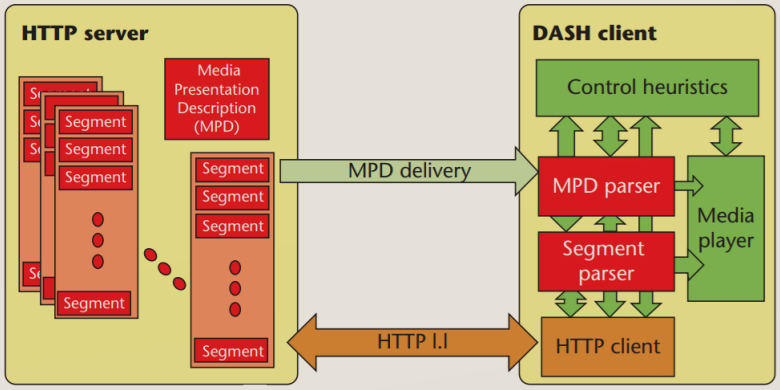


Figure 2.9: Overview of a interaction between a Server and a MPEG-DASH Client. Adapted from [21].

To play the content, the MPEG-DASH client needs to get the MPD, which is delivered using HTTP. By analyzing the MPD, the client learns about which content is available, of what type, the bandwidth interval and existing alternatives to that stream. With this information, the client can select the right stream of the content for him. There are only specifications for MPD and the format of the segments in MPEG-DASH. Everything else is out of the scope of MPEG-DASH.

For a dynamic HTTP streaming, multiple alternatives of the media content in the server are required. This content may consist of multiple media components with different characteristics [12].

MPD has one or more periods, where a period is an interval on the temporal axis. One period contains a starting time, duration and has one or several adaptation sets. These adaptation sets give information about the media components and their encoded alternatives and normally also have several representations. The adaptation sets are encoded alternatives of the same media component, just changing the bit rate, resolution, or another characteristic and each representation is one or more parts. Each part is a media stream block in the temporal sequence. It has a URL for easier localization [21].

The client begins by analyzing the MPD XML document and then chooses the right lot of representations based on the components in the MPD. After this, the

client constructs a timeline and requests the right media parts to start playing the media content [4].

The content can be collected as a lot of parts. A part is established as the body of the response to the client's HTTP GET. A media component is encoded and split into several parts. The first part can be the starting part that has the needed information to initialize the client's media decoder. The media stream is split into one or more consecutive media parts, each with a different URL, an index, start time and duration. Every media part has at least one stream access point, which is a random-access point in the media stream where only data is used, after that point. To start the download of several parts of the parts, the specification must define a method of signaling subparts utilizing the index box. The box characterizes subparts and stream access points in the part by communicating their byte offsets and extent.

MPEG-DASH delineates segment-container formats for ISO Base Media File Format and MPEG-2 Transport Streams. This protocol is video CODEC agnostic. This means that it can use any CODEC and allows for multiplexed and demultiplexed encoded content [21].

Every adaptive set may use a content protection descriptor to detail the DRM scheme. It can also use several content protection schemes for an adaptive set considering that the set recognizes at least one of them [21].

2.4 Video CODECs

Video CODECs are standardized processes of compressing and decompresses video. There are many different ones, but the most significant ones identified in the research were H.264, H.265 and AV1. VP8 is also going to be briefly explained since it was the CODEC used in the thesis because the main browsers used in the development have VP8 as their default CODEC. Before starting with the review of the different CODECs, there are a few definitions that must be explained.

An Intra frame or I frame is a frame that is coded without using other frames as a reference and in which the macroblocks are coded into using intra prediction.

A Predicted frame or P-frame is a frame that uses previous I or P frames for motion compensation and can be a reference frame for the following frames.

A Bidirectional predicted frame or B-frame can use the following and previous P and I frames for motion compensation and because of that can have the biggest compression rate.

The interaction between these three types of frames is shown in Figure 2.10.

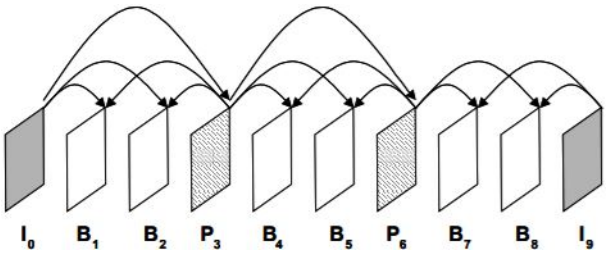


Figure 2.10: Interaction between the different. Adapted from [1].

2.4.1 H.264

H.264 is a video compression standard that is highly used and has gained dominance in his area, compared only to JPEG for image compression. The H.264 standard has also been extended to allow scalable video coding with a backward-compatible non-scalable base layer. This extension enables the implementation of advanced application scenarios with H.264, such as scalable streaming and universal multimedia access. Given the dominant application of H.264 as video compression, the necessity of practical security tools for H.264 is unquestionable.

In Figure 2.11, we can see the different elements composing the structure of H.264/AVC and the interactions existing between them. This section focuses mainly on the Video Coding Layer (VCL) and the Network Abstraction Layer (NAL) since those are the key points in this CODEC.

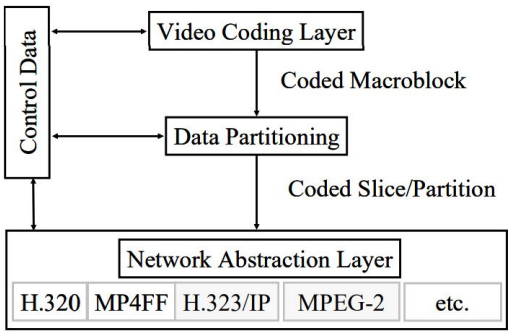


Figure 2.11: Overview of the sctruture of H.264. Adapted from [5].

NAL was made with the intent of having "network friendless" to allow adequate customization of the use of VCL for a vast array of systems. It also smooths the mapping of H.264/AVC VCL data since the video data is organized into NAL units, i.e. are packets containing an integer number of bytes. The first byte is the header and the remaining are the data of the type registered in the header. The data bytes are interleaved as needed with emulation prevention bytes, which are bytes that prevent certain patterns. The NAL units have a generic configuration so it can be used in packet-oriented and bitstream-oriented transport systems. Byte

stream NAL units are defined so that they are prefixed by a specific pattern of 3 bytes named start code prefix. With this, the limits can be found by looking for these prefixes. Packet-transport NAL units are different from this since they do not carry these prefixes and instead identify the boundaries themselves making these prefixes useless [5].

NAL units are classified as either VCL or non-VCL, the difference being that the VCL units have the data that embodies the values of the fragments in the video pictures, and the non-VCL have any further related information. For instance, the parameter sets which are supposed to have almost fixed information and allows the decoding of many VCL NAL units. These parameters allow the possibility of having a small extent of data referring to a broad extent of information without having to repeat that information in each VCL NAL units. This set can be sent before the VCL NAL units that they administer and can be repeated to reduce the data loss [5].

An access unit is a NAL unit with a specific role, decoding one of these ends in the formation of one decoded picture. All these access units include a set of VCL NAL units that combined from the primary coded picture. It can be prefixed by an access unit delimiter to help in find the start of this unit. The primary coded picture is a set of VCL NAL units that are themselves slices representing the fragments of the video picture, also it may have an inessential portrayal of areas of the same video picture. A sequential series of access units is a coded video sequence that only uses one sequence parameter set. They can also be coded independently of other sequences, given the right parameters set information. This sequence starts with an instantaneous decoding refresh access unit or IDR, that has an intra picture. An intra picture is a coded picture that can be decoded individually without the necessity of decoding any previous pictures in the NAL unit stream and the existence of this picture means that no following pictures in the stream will not need to mention previous ones [22].

The Video Coding Layer (VCL) has a block-based hybrid video coding approach. This means that each coded picture appears in block-shaped units of joined luma and chroma parts named macroblock. The simplest form of source-code algorithm is a combination of transform coding of the prediction residual to capitalize on spatial statistical dependencies and inter-picture prediction to capitalize on temporal statistical dependencies. VCL has no main code giving it a significant boost in compression efficiency but a multiplicity of small gains from different parts giving it a big increase [5].

In H.264/AVC, the coded video sequence is a sequence of coded pictures. These can be one of two things: an entire frame or a single field. A picture is

sectioned into fixed-sized macroblocks that have an area of 16x16 for the luma component and 8x8 for each component of the chroma (These are Cb and Cr, that correspond to the color deviation from gray to blue and red respectively, luma corresponds to the brightness). Macroblocks are the standard building block that the decoding process is specified for.

Like in previous CODECs H.264 uses I, P, and B frames but it adds two new frames the switching-I and switching-B frames, one is an I frame that allows a correct match of a macroblock in an SP frame for random access and error recovery, the other is a P frame that permits switching between different pre-coded pictures.

This CODEC takes advantage of the spatial redundancies by permitting intraframe prediction. Macroblocks are coded as intra macroblocks when there is no chance of temporal prediction or when it is inefficient. This prediction is made by doing a weighted average of the bordering pixels of the same frame to predict that pixels since these are most likely similar [10].

Interframe prediction is just like its predecessors but with added extensions like variable blocks sizes, multi-frame references, weighted prediction, and fractional pixel accuracy. These allow the code to perform better but have an added complexity. An I frame may utilize up to 16 distinct reference frames, but the actual number of reference frames is only restricted to the buzzer size. It predicts two predictive macroblocks from two predictions.

In H.264, there is entropy coding that is the encoding of parameters, coefficients and other numeric values with binary codes. This encoding tries to diminish the statistical redundancies in compressed video. The first part of the decoding is the recovery of numeric values from the binary codes. This coding uses universal variable-length coding for all components except the quantized coefficients. The coefficients can be coded using context-adaptive Context-adaptive VLC (CAVLC) or Context-adaptive Binary Arithmetic Coding (CABAC). This last one gives the most coding improvements, but it is computationally expensive [10] [22].

The deblocking filter increases the perceptual quality and the efficiency of the predictions in interframes. This filter is applied to macroblocks in the raster scan form and has the size defined by the transform code, either 4x4 or 8x8[22].

2.4.2 H.265

H.265 is a video compression standard, created to be the successor of H.264 with improved data compression, in comparison to AVC, of 25% to 50%, on videos with

the same quality level and improving considerably the video quality at the same bit rate. It supports big resolutions like 8K, and almost all supporting hardware has HEVC's higher fidelity Main10 integrated in them. HEVC is competing with the royalty-free AV1 CODEC in terms of future usage and quality [11].

Like in H.264, H.265 implements the block-based hybrid video coding with the added size of the macroblock up to 64x64. Three new novel block approaches are present in this CODEC, Coding Unit (CU), Transform Unit (TU) and Prediction Unit (PU). CU is like the macroblock of H.264 and can have multiple sizes while being confined in being square-shaped. TU is the unit related to transforming and quantization, it one can be bigger than PU, but not CU. PU is the unit for prediction, and the biggest PU can only be the same size as CU. The main structure of the coding is made of various sizes of CUs, PUs and TUs in a recursive manner as soon as the size of the Largest Coding Unit and hierarchical depth are delineated.

In H.265, the intra prediction combines two-directional intra prediction processes, the Angular and the Arbitrary Direction. This combination allows for a lower complexity where parallel processing is achievable and in which samples of already decoded neighbor PUs get used. The mode is signaled by the modes of neighboring Pus and syntax indicators [11].

H.265 does inter prediction by using the frames stored in a reference frame buffer, that permits multiple bi-direction frame reference. To choose the reference area, it is necessary to have a reference picture index and a motion vector displacement. Getting coding efficiency is done by having skip and direct modes identical to the ones in H.264, and a motion vector derivation that is executed on neighboring PUs. Motion compensation is done with a quarter-sized fragment of the motion vector precision. At the TU level, there is an integer spatial transform working and a rotational transform for block sizes larger than 8x8. In the CU level, there is an adaptive loop filter to minimize relative distortion with the original picture and its filter coefficients. In the prediction loop, there is a deblocking filter work that is identical to the one in H.264. After passing in the filters, the frame is stored in the CODEC buffer.

In terms of Entropy Coding, H.265 has two context-adaptive coding designs, one for lower complexity and another for higher complexity. The base for the lower complexity one is a variable-length code table selection for all the syntax members, although a context-based system does the selection of a code table. This is identical to the CAVLC in H.264 but implemented in less complex design thanks to the system's structure. The higher complexity one operates with a binarization and context adaption method identical to CABAC in H.264, with the

difference that this uses variable length to variable length codes rather than using an arithmetic coding engine. If we talk about performance this coding can be better parallelized and has higher throughput per processing cycle than CABAC [11].

2.4.3 VP8

Project "WebM" focuses on developing open-source media formats, and it was revealed by Google, with the first CODEC of the project being VP8. Since its release, VP8 has features that focus on achieving low computational complexity and high compression efficiency [3].

For the design of VP8, its developers had three main considerations:

- Low bandwidth requirement: VP8 was designed to work mainly on the lower side of quality of video, so smaller bandwidths can be used.
- Heterogeneous client hardware: Due to having clients with completely different hardware, from a phone to a computer with high processing power, VP8 runs implementations that are adeptly work on each one of them.
- Web video format: VP8 has a maximum of 16383x16383 pixels with 420 color sampling and a color depth of 8 bit per channel.

To improve video compression and reduce the complexity in the decoder, VP8 has some technical features that allow this [3]. The following bullet list has a brief explanation of each one:

- Flexible reference frames: three reference frames are used in inter prediction. Their scheme is different from other multiple reference motion compensation schemes and VP8 limits their buffer size.
- Efficient intra prediction and inter prediction: a new intra prediction mode called "TM_PRED" was created. For inter prediction, "SPLITYMV" mode was added to allow the coding of arbitrary block patterns inside a macroblock.
- Hybrid transform with adaptive quantization: like other video coding schemes, video frames are divided into macroblocks, each macroblock is one block of 16x16 for luma pixels and two blocks of 8x8 for the chroma pixels. VP8 divides these blocks even further into 4x4 during the transform and quantization process.

- High performance sub-pixel interpolation: for motion compensation, VP8 uses quarter-pixel accurate motion vector for luma pixels and for chroma pixel, it uses one-eight accurate motion vectors.
- Adaptive in-loop deblocking filtering: VP8 uses different types of filters depending on the prediction mode and computational complexity.
- Parallel processing friendly data partitioning: VP8's design allows for parallel processing with multiple cores while having almost zero impact in systems with a single core.

These make VP8 a serious contender for usage in the video call system.

VP8 was superseeded by VP9, developed by the same organization. More recently, VP9 is being superseeded by AV1, presented in the next section.

2.4.4 AV1

AV1 is designed to be an open, royalty-free video CODEC. It was developed by the Alliance for Open Media (AOMedia) as an improvement to the existing VP9, and with the objective of achieving considerable gains when it comes to compression while maintaining realistic levels of decoding complexity and hardware possibilities [6].

The coding block in AV1 is an improvement of the existing VP9 coding block by increasing to 128x128 from 64x64, expanding the partition tree to a 10-way structure and adding 4:1/1:4 rectangular partitions. AV1 has more flexibility than his predecessor because it can use partitions below 8x8, even being able to use 2x2 in some cases.

In the Intra Prediction subject, AV1 uses VP9 prediction modes (10 modes, 8 directional 45 to 207 degrees, and the DC and True Motion (TM)) upgrading them, allowing it to be an improvement from VP9, these upgrades divide into 6 main points [6].

1. AV1 takes advantage of a bigger range of spatial redundancies by extending the directional intra modes to angles with a more refined granularity. The 8 angles, also present in VP9, are named nominal angles and where the new angle variations are based on, in a 3 degrees step size. The 48 extension modes are composed of a combination of a directional predictor that connects the pixel to its corresponding sub-pixel location on the edge and interpolates the reference pixel with a bilinear filter. The number of directional intra modes is 56.

2. The way the AV1 works in intra-directional modes is by adding 3 smooth predictors that predict the block, making a quadratic interpolation in the vertical or horizontal directions. The PEATH predictor replaces MT, and in all pixels, we copy the left, the top and the left as references to make calculations to predict.

3. In AV1, there is an intra predictor dedicated to the chroma pixel model as a linear function of coincident reassembled Luma pixels. These are subsampled in chroma resolution and the DC part is removed to make the AC contribution. To bring the CA chroma closer to the CA contribution, AV1 calculates the criteria based on the original chroma pixels and signals them in the bitstream. This allows for less complexity and better predictions.

4. AV1's intra coder can call back to previous rebuilt blocks in the same frame. This can be great for screen content videos that have the same textures and patterns. There is a new prediction model called IntraBC that duplicates a rebuilt block in the present frame as the prediction. A displacement vector gives the position of the reference block.

5. Palette modes are offered to the intra coder, by AV1, as a generic coding tool. This predictor is indicated to each plane of the block by a color palette and color indices for all pixels in the block. These indices are entropy coded using the number of base colors calculated in the exchange between compactness and fidelity and the neighbourhood.

6. FILTER_INTRA modes are created for luma blocks, seeing them as 2-D non-separable Markov processes to allow the capture of decomposing spatial association with references on the edges. There are five filter intra modes and they are described by a set of eight 7-tap filters that echo association between seven adjacent neighbors and a pixel in a 4x2 patch.

Like in the intra prediction, AV1 is a big improvement from its predecessor VP9 since it had only 2 references and 3 candidate reference frames and because of that, the predictor could only function in either block-based translation motion compensation, while in AV1, we can increase the number of reference frames and motion vectors, outdoes old restraints from block-based translational prediction, plus improves compound prediction with the use of highly adaptable weighting algorithms.

AV1 increases the reference's count to 7 from VP9's 3. In terms of frames, AV1 uses the already existing ones (LAST, GOLDEN, ALTREF) and adds 4 new

frames, two future frames (BWDREF and ALTREF2) and two near past (LAST2 and LAST3). BWDREF is a coded designed with the intention of not using temporal filtering so it is better when applied as a backward reference in a relatively smaller length. ALTREF2 is a compromise refined future reference between ALTREF and GOLDEN. These references can be used as a pair to build a compound mode or by a single prediction.

Excellent motion vector coding is essential since it removes a big part of the rate cost for inter frames. AV1 integrates a refined MV reference determination system to get valuable references for a specific block seeking temporal and spatial contenders. This CODEC does a deeper spatial neighborhood search and uses a temporal motion field appraisal system. This field appraisal happens in three steps: vector buffering, trajectory creation and vector prediction [6].

AV1 uses two motion compensation, Overlapped Block (OMBC) and Warped. The first one uses predictions that were originated in bordering motion vectors by aggregating them, to lower prediction errors near block edges. A 2-sided overlying algorithm was created to allow OMBC an easy fit in the advanced partitioning framework. As it is working, it merges the secondary inter preditors with the block-based prediction by employing premade 1-D filters in both axis directions. OMBC is only used when using an individual reference frame and exclusively works with the primary predictor of a neighbor. Warped motion compensation is done by having two prediction modes, global and local. The local one has the objective of describing changing local motion applying the smallest overhead, by making the model parameters at the block level from 2-D displacements into the casual neighborhood. The global one is for taking care of camera movements and permits transmitting motion models particularly at the frame level for the motion amidst a current frame and its references [6].

In AV1, there is a group of advanced compound prediction tools, each one with certain characteristics like wedge prediction, frame distance-based compound prediction, inter-intra prediction, and difference-modulated masked prediction. These are compounds of previously talked predictions.

AV1 has two improvements when it comes to transforming coding one being the Transform Block Partition and the other the Extended Transform Kernels. In the first one, AV1 permits the split of the luma inter coding blocks into transform units of different sizes that have the possibility of being described by a recursive partition descending by up to two levels. And the second one is a better lot of transform kernels is made for both inter and intra blocks. All the 2-D kernel lot is a combination of different transform types. This CODEC adds flipADST which does ADST in the opposite order and adds the Identity Transform (IDTX) which

signifies jumping transform coding in a determined direction. It helps with the coding of difficult edges[6].

In terms of Entropy Coding, AV1 improves in two subjects, Multi-symbol Entropy Coding and Level Map Coefficient. In the first one, AV1 uses an adaptive multi-symbol arithmetic coder with a symbol to symbol focus. Every syntax component is a part of a alphabet of N components, and a context is a making of a collection of N probabilities combined with a sum to allow quicker adaptation. These probabilities are saved as 15-bit aggregated distribution functions. In the level map coefficient, AV1 uses an evolved version of VP9 and switches to a level map design for a considerable transform coefficient modeling and compression. It is built on the conclusion that the lower coefficient levels normally consider the bigger rate value[6].

Chapter 3

WebRTC and Thesis Development

This chapter describes the architecture of the video chat software developed in this thesis using the WebRTC platform. WebRTC was the solution found to implement a video chat application that may run in mobile phones and computers, is easy to use and hass lightweight implementation without huge requirements outside of a camera and Internet connection.

3.1 Video Call System

Working with WebRTC as a baseline for the video call, a distributed system was developed with the intent of having a system that can have peer-to-peer video calls while also having a control system that allows users to choose who to call and decline calls. This system is composed by a relay server (Traversal Using Relays around NAT (TURN)), a HTTPS server serving as signalling server (Signalling) and Users web page (User A and User B). When the users open the webpage, it automatically connects them to the signalling server.

Users webage communicate with the Signalling server via WebSocket messages in a process that allows them to know which users are online at any given time. The server updates the list each time a user logs into the system or drops out of it by sending a WebSocket message with the type roomusers.

For registration purposes every user has a randomly generated number associated which is the number used to identify him.

Users have two status, “OnCall” or “Available”, where one indicates if a user is already on call and the other indicates that a user can be called. When a user

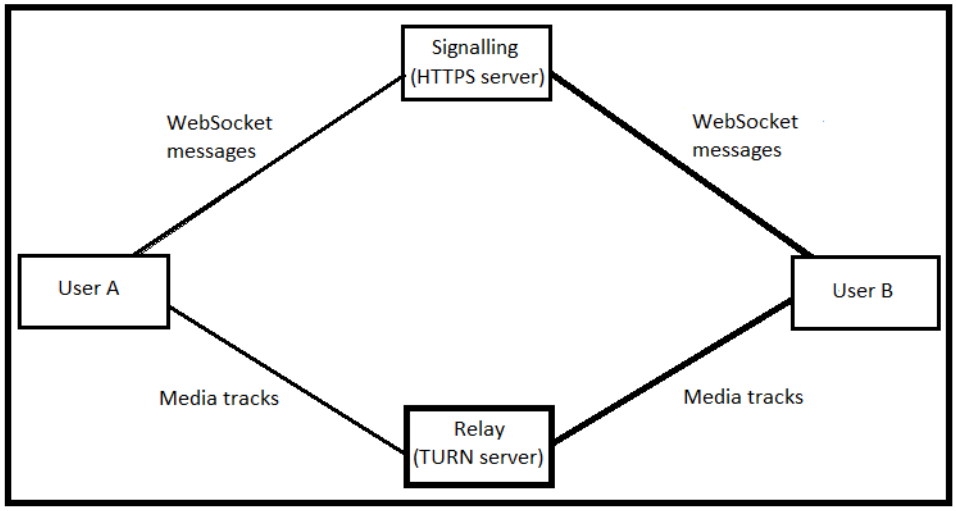


Figure 3.1: Video Call application Setup.

is "OnCall" any attempts from another user to call him will automatically be declined and the user that made said call will be informed of the refusion by the server with a WebSocket message.

To make or end video calls, a user must send multiple messages to another user so certain parameters are defined, and the other user must accept the call. All these messages pass through the signalling server that handles them and routes them to their destination.

The Signalling server runs a static folder which contains the user. This way, when the webpage is open, the browser becomes automatically a user, if it accepts the request to access camera and audio recorder.

When, the video call is established, all packets in the video stream pass through the relay server. This is a TURN server that allows users using IP private addresses behind a NAT router to bypass the NATs and users from different private networks to communicate with each other.

Transmitting the users' media to another user requires a connection between the two users. This connection is defined by the users through WebSocket messaging passing in the Signalling server. This connection is named RTCPeerConnection.

RTCPeerconnection object is the representation of a WebRTC connection from a user to another one, illustrated in figure 3.2. The focus is to allow for a reliable streaming of tracks between the two users, i.e. independent media steams. The interface allows for the implementations of methods to monitor the states of connections and tracks and to have handlers that handle the changes in connections and messages.

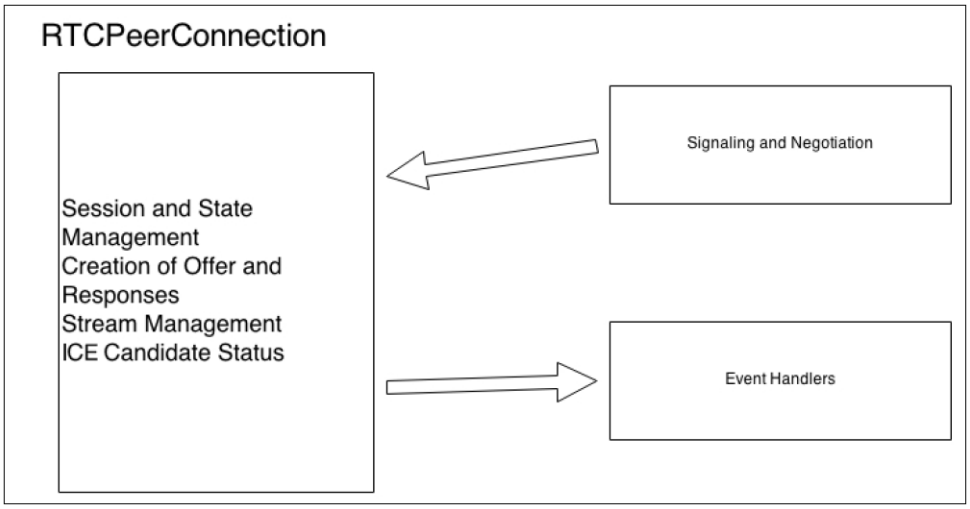


Figure 3.2: Overall of RTCPeerConnection.

Figure 3.2 shows the overall interaction between the Video Call System and the RTCPeerConnection. This connection is created through the exchanging of messages by the users. The connection defines the session by saying, which is the local and the remote user. After that it exchanges messages to define the ideal route for the media tracks to reach their destination and manages the tracks to and from the other user.

RTCPeerConnection has multiple states depending on the current state of the connection, like connected, when a connection just started, disconnected when the user drops out, and closed, when a call is finished. There is also failed for when a connection attempt fails.

The RTCPeerConnection events handled by the Video Call System are the receiving of tracks from the other user and assigning them to its viewing box. The connection state handler is for when the state of the connection changes so that the user webpage is updated to allow calls to other users.

3.1.1 WebSocket Messages

All the WebSocket messages are JSON messages sent by either the user or the signalling server. They have different types and purposes, which are described in the following texts and images. When they are sent, the messages are transformed to strings using the method “stringify” and when they are received, they pass through “parse” to reconstruct the JavaScript objects.

3.1.1.1 Roomusers

The Roomusers message contains the name of all users online and their current status. “roomUsers” is an array of users online and statusUsers is an array with all the users and their current status. Figure 3.3 represents the full content of the Roomusers message.

```
type          : "roomusers",
roomUsers     : roomUsers ,
statusUsers   : statusUsers
```

Figure 3.3: RoomUsers message.

3.1.1.2 Refuse

The Refuse message is created as a server message to the client informing it that his attempt to call another user was refused since that user is occupied and cannot accept his call. Figure 3.4 shows that it only contains the origin of the message.

```
type     : "refuse",
origin   : data.origin
```

Figure 3.4: Refuse message.

3.1.1.3 Offer

The "Offer"message type contains a users’ capabilities of video chat and shows them to its destination. This message is sent to a user to request a video call. It contains the origin and the destination of the message, as shown in figure 3.5.

```
type     : "offer",
offer    : data.offer,
destiny  : data.destiny,
origin   : data.origin
```

Figure 3.5: Offer message.

3.1.1.4 Answer

The "Answer"message type is the used to respond to an “Offer” message in which the user can either reply “Ok” to accept the call, or refuse “NOK” in the status of

the message. This message also contains the origin, the destination of the message and the capabilities of video chat of the destination and shows them to the call requester. The message fields are illustrated in figure 3.6.

```
type    : "answer",
answer  : data.answer,
status  : data.status,
destiny : data.destiny,
origin  : data.origin
```

Figure 3.6: Answer message.

3.1.1.5 Candidate

The "Candidate" message contains the information about which different possibilities of transportation and relaying are available for the video chat tracks. The fields are represented in figure 3.7 and include the origin, destination and the possible routes for the video call tracks.

```
type        : "candidate",
candidate   : data.candidate,
destiny     : data.destiny,
origin      : data.origin
```

Figure 3.7: Candidate message.

3.1.1.6 Hangup

The "HangUp" message is sent to terminate calls. The message fields are shown in figure 3.8 and include the destiny and user that ended the call.

```
type     : "hangup",
destiny  : data.destiny,
origin   : data.origin
```

Figure 3.8: HangUp message.

3.1.2 Temporal Description

The setup of the video call and definition of its settings require the exchange of a sequence of messages between the users and the server. To facilitate the understanding of the protocol, the sequence is illustrated in figure 3.9.

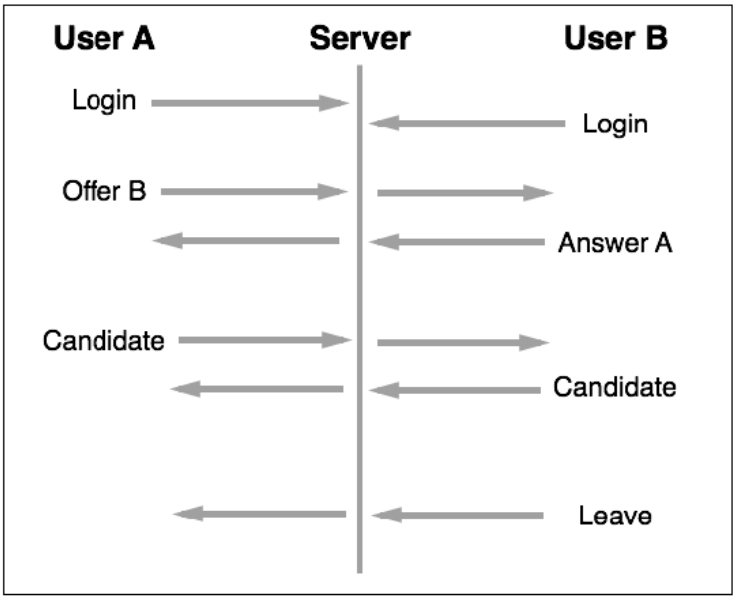


Figure 3.9: Users Interactions.

This sequence, in figure 3.9, starts with both users being online and one of them, in this case A, sending an offer to the server with B as its destination. B responds with an answer message, with the status "OK" , signaling that B accepts the call. After A receives the answer, it sends the "candidate" message so that the best route for the packets is found and used. B answer the "candidate" message with his own "candidate" message. With this preparations and setups concluded, the video chat starts, and the users can communicate. When a user wants to end a call, the "HangUp" message is sent to the server and then from the server to both sides to terminate the call.

In the case of a user, call refused by the other user the sequence of messages is the same until the Answer but the status of the Answer is "NOK", ending the sequence there. Another case is when a user that is already on call, is called. When this happens, the server detects this and automatically sends a "Refuse" message to the user, not letting the offer progress to the other user.

3.1.3 Signalling Server

The signalling server is implemented in Node.js using the Express server. The signalling server reads the headers of the messages to extract their type and to whom they are destined to, so it can forward them to the right user. It also checks for status of callers and callees, so that no calls are broke or cancelled by an incoming new call. Before starting a video call a user connects to the HTTPS

server. The SSL public key and certificate are provided during the setup of the server, as is shown in figure 3.10. Since this is an HTTPS server, the port that the server listens is the 443 port.

```
var options = {
    key: fs.readFileSync('./ssl/private.key'),
    cert: fs.readFileSync('./ssl/server.crt'),
} }
var app = express();
var server = https.createServer(options, app);
var wsApp = expressWs(app,server);
```

Figure 3.10: Establishment of the SSL certification and creation of the HTTPS server.

The Signalling server starts the static part of the project, which includes the HTML page with the client's Javacript page inside of it. Each time a user web page is open, it automatically becomes a client of the system and a random number is assgined to it. The list of users is showed in the web interface of all users as a dynamic list of buttons containing all users online.

The signalling server uses the group of objects listed in figure 3.11.

```
let users = new Map();          //user connection
let statuses = new Map();         //user status
let calls = new Map();     //user calls
```

Figure 3.11: Signalling Server Maps.

The first map, named users, is used to store the connection data of a user so that messages can be routed to it. This map existence helps the performance of a server since it allows the server to have all connections stored in one place and to get access to the connection by just using the user as the key.

The second map, called statuses, is used to store the current status of a user. It allows the server to make decisions about refusing calls and to update in all users the status of every user currently online in the system. The key for this map is also the user and the information is either "OnCall", when a user is on a cal, or "Available", when a user is available to make a call with someone.

The third map is calls. It stores all live video call users and their corresponding calling user. Each call is stored two times since both users need to be keys so that it is possible to close calls. The server uses this map to free the other user from the call, changing his status to Available and informing it that his call ended.

3.1.3.1 WebSocket Messages Handlers

The signalling server defines a Websocket listener, shown in figure 3.12, to communicate with the users. Each time a user webpage is opened, it connects to the signalling server via the WebSocket. After the detection of this new source of WebSocket messages, the server sets up the new user inside the server and inside the maps. It also informs all other users that a new user is present in the server by running the showRoomUsers() function, which sends to all users, the list of online users and their status.

```
app.ws('/ws', async function(ws, req)
```

Figure 3.12: WebSocket Listener.

Once a user is inserted into the signalling server, it can start sending WebSocket messages to the other users. These messages are handled by the server inside the handleWS() function, which receives the user's name and their WebSocket connection. The user is also added to the Users Map and to the Statuses Map with the "Available" status. On receiving WebSocket messages, they are processed by ws.on() function, shown in figure 3.14, for a message of the kind 'message'. The function analyses the type of message and processes its contents accordingly.

```
async function handleWS(username,ws)
```

Figure 3.13: WebSocket Handler.

```
ws.on('message', async function (message)
```

Figure 3.14: WebSocket on message.

When a message of the type "Offer"is received, the server gets the destination's connection from the Users Map and if the destination user is "Available" for a call. The server sends a "Refuse" message back to the user when it is not available, or the connection is not known. If it is "Available" the server forwards an "Offer" message with the same content and sends it to the destination. The server also relays the "Answer" message from the destination user and checks if the initiator user still exists, routing the message to it. The server also checks the status of the message and if it is "OK", it changes the status of both users inside the Statuses Map to "OnCall", adds them to the Calls Map and updates everyone else inside

the system about the occurred changes. For the "Candidate" message the server only checks if the destination exists and routes the message to it.

When a "HangUp" message is received, the server gets both of the users connection's and checks if they exists before sending a "HangUp" message to both sides, updating the Statuses of both users to "Available" and deleting them from the Calls Map since that call has ended. Finally, it broadcasts the changes to every user by running the showRoomUsers() function.

When a WebSocket connection is closed, the server knows which user closed it, so it proceeds to check if he was in a call with another. If he was, the server ends the call, changes the status of that user in the Statuses map to "Available", deletes both users from Call Maps and deletes the user that closed from the Users map and from the Statuses Map.

3.1.3.2 Show Room Users Handler

Each time a change happens in the server, every user gets a new list of online users and their current status in a "RoomUsers" message. The showroomUsers() function handles this message and starts by getting an array with all the users present in the system using the users map. Because a Map is not allowed to past through a WebSocket connection, an Array is built with the status in the statuses map. A "roomusers" message includes both arrays and is sent to every online user. The creation of the showRoomUsers() is observable in figure 3.15.

```
async function showRoomUsers()
```

Figure 3.15: Show room users function.

3.1.4 User Description

The following section describes the user's application and all necessary steps in its development, which corresponds to the User A and User B illustrated in figure 3.1.

The User software is divided into three main parts: the graphical part, the user setup and the Message Handlers. The graphical part contains the HTML file with the video boxes, the buttons and the list, while the user setup contains the coding part of the HTML, how to capture the camera, the RTCPeerConnection starter and the creation of the WebSocket connection. Finally, the Message Handlers has the processing of the different messages received by the user.

3.1.4.1 Graphical Part

When a human user opens the webpage, the first thing he sees are the 2 black boxes for the videos and after a small delay he can see himself on the left side box through his camera. If there are other users a list of buttons will appear below the camera with all those users and their current status. The dimensions of the video boxes are 640px width and 320px height. Figure 3.16 shows the webpage seen by a user.

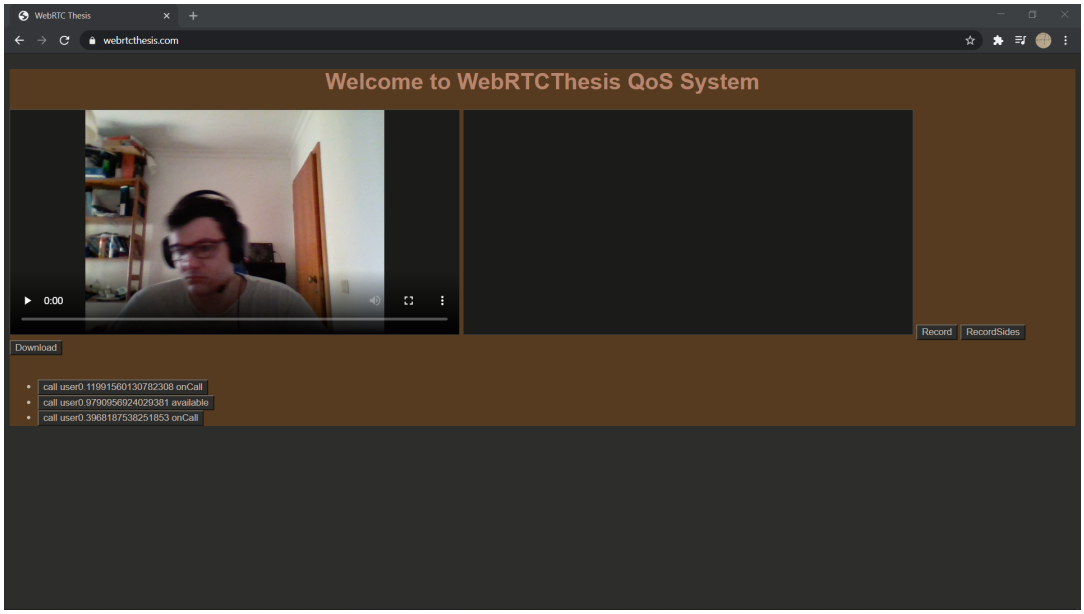


Figure 3.16: Screenshoot when a user opens the Webpage.

When a user receives an incoming call, a pop-up appears, as showed in figure 3.17, with the name of the user that is calling and a button "OK" to accept the call, or a "Cancel" to decline the call. If the user accepts, the buttons available to him change and he starts the video call.

On a call, the user can see the "hangUp" Button to finish the call with the other user. The users list is disactivated and only comes back once this call ends, to avoid changing calls without first ending the current one.

A user that tries to call a person already on a call gets a pop-up message saying Occupied and continues available for calls. This pop-up is a consequence of the user receiving the "Refuse" message from the server, as shown in Figure 3.19.

A user that makes a declined call attempt gets a pop-up message saying that the user declined his call. This user is free to try and call again or call other users after receiving this message, as shown in figure 3.20.

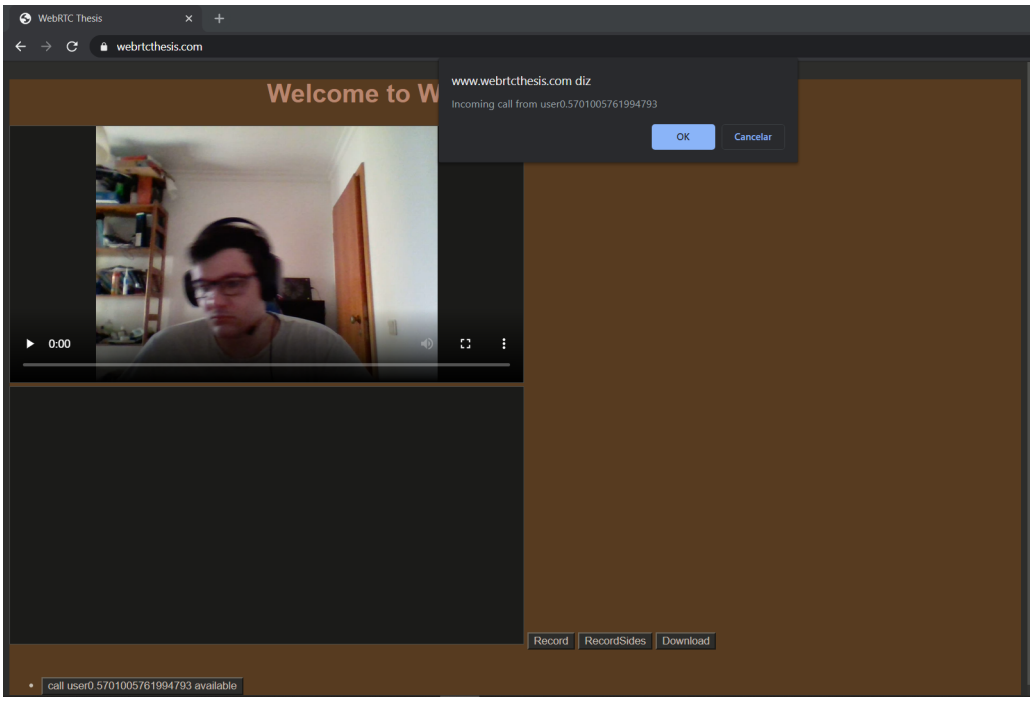


Figure 3.17: Screenshoot when a user receives a call.

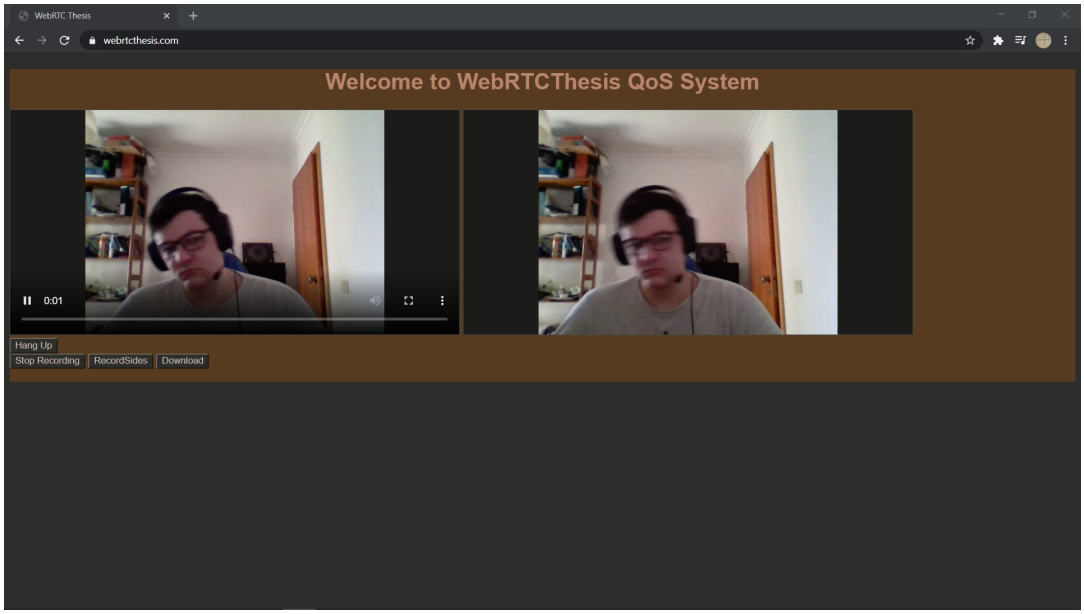


Figure 3.18: Screenshoot when a user is in a video call.

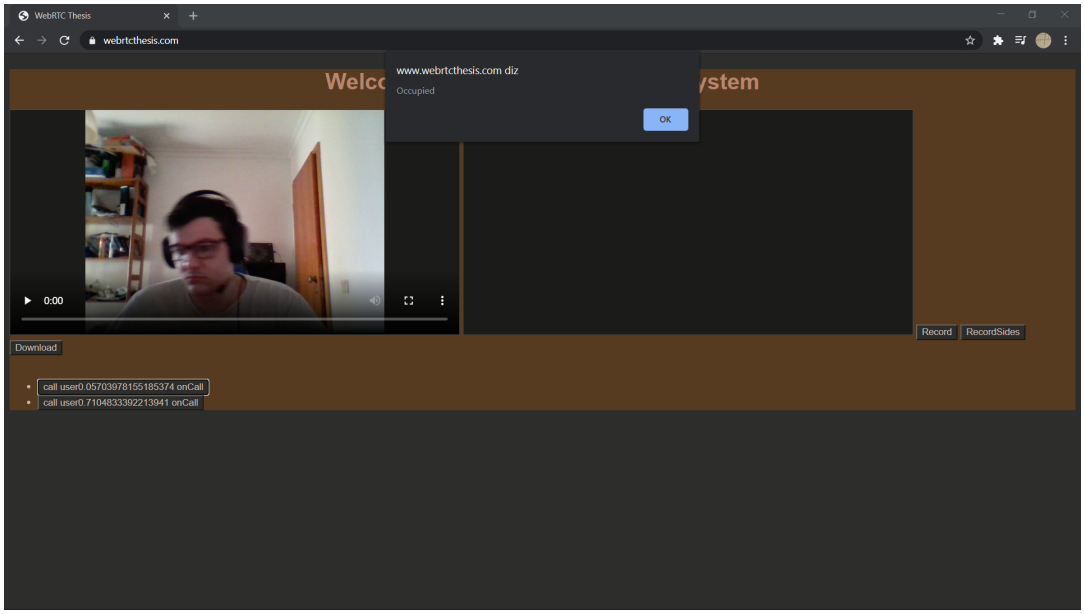


Figure 3.19: Screenshoot when a user tries to call a user already on call.

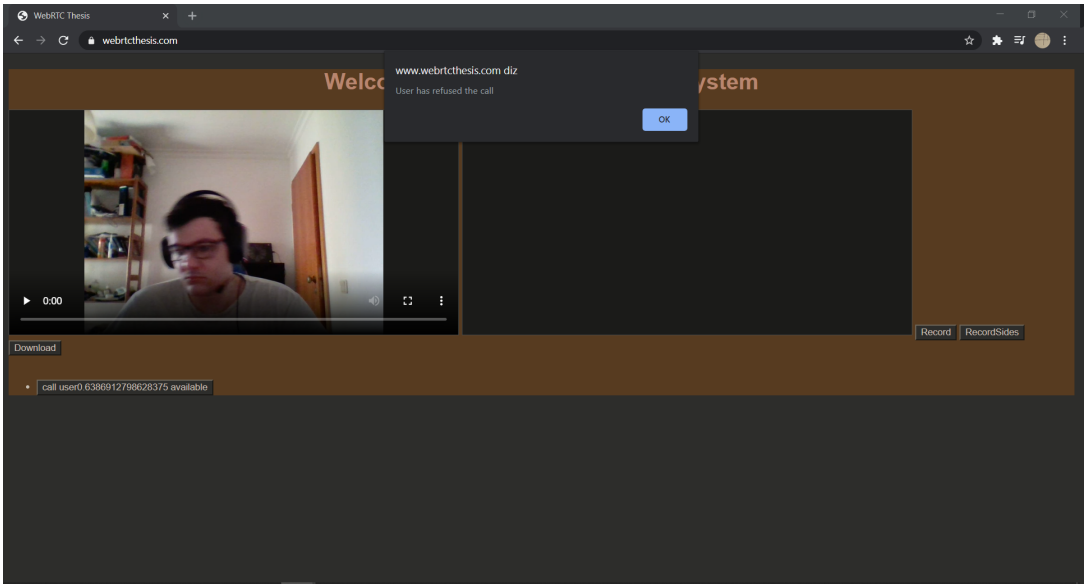


Figure 3.20: Screenshoot when a user gets his call refused.

3.1.5 User Setup

The HTML file calls the JavaScript File to run all the code for the user. This code includes the set-up of everything the user needs to run such as global variables, buttons,camera, audio and creation of the WebSocket connection to the server.

```
const myVideo = document.querySelector('#yours');
const theirVideo = document.querySelector('#theirs');
let completelist = document.getElementById("thelist");
let hangUpButton = document.getElementById("hang-up");
let rtcConnection;
let stream;
let ws;
let Uname;
```

Figure 3.21: Global variables.

The global variables used in the user are presented in figure 3.21:

- myVideo corresponds to the left side video box and it is the video of the users camera;
- theirVideo is the right side video box which is allocated to receive the video of the other user in a call;
- completeList is a dynamic list of buttons that contains all the online users and their status;
- HangUpButton is the button to end a video call;
- rtcConnection is a variable created to store a users' RTCPEERConnection during a video call;.
- stream is used to store a users' Media capture, both video and audio;
- ws contains the users' WebSocket connection to the signalling server;
- Uname is the users' username.

When a new user opens the webRTCThesis webpage, the application tests if its browser is capable of doing RTCPeerConnections, since without this a user can not make video calls. The system informs the user through the console that there is no RTCPeerConnection support.

The GetUserMedia() function was used to get the user's Media. This function asks the user for permission to access the camera and audio and produces a MediaStream divided in tracks consisting of the requested media, including both video

and audio. The usage of this method is presented in Figure 3.22. This functions asks for constraints as a parameter where video and audio is selected, where the configuration illustrated in figure 3.23 is used.

```
rstream = await navigator.mediaDevices.getUserMedia(constraints);
```

Figure 3.22: GetUserMedia() method in use.

```
{ video: true, audio: true }
```

Figure 3.23: Video and Audio configuration.

The user's application runs the getMedia() function, that calls the GetUserMedia() function, to get its media and afterwards defines it as the stream global variable, as shown in figure 3.25. This is followed by the assignment of the stream to the left video box, in the assignStream() function. The audio is set to true, except for testing.

```
stream = await getMedia({ video: true, audio: false })
```

Figure 3.24: Getting Media devices for stream.

```
await assignStream(myVideo, stream)
```

Figure 3.25: Assigning stream.

With the user's video now available, a Websocket setup is run in the setupWS() function where the ws variable is defined, as seen in figure 3.26. This setup consists of creating the WebSocket node and the creation of the message handler. After this operation, the user is ready to make and receive calls.

3.1.6 Call Setup

To start a call the user must press the button of another user on the list in the webpage. When this happens, the sendOffer() function is started. This function begins by creating a new RTCPeerConnection using the function setupRTCConnection() which also starts preparing tracks to be sent to the other user. Following this, the user creates an offer, sets the local description of the rtcConnection and sends it in the "Offer" message to the signalling server, which in turn sends it to the other user.

```
ws = new WebSocket(`wss://www.webrtcthesis.com/ws?user=${Uname}`);
```

Figure 3.26: Creation of the WebSocket.

```
rtcConnection = await setupRTCConnection(username);
```

Figure 3.27: RTC connection function.

Once the "Offer" message gets to the other user, it is handled by the handleOffer() function, shown in figure 3.28, that alerts the user of an incoming call. In the case the user accepts, the function creates a RTCPeerConnection with the calling user has header and sets the remote description of the rtcConnection as a new RTC session using the received offer.

The callee creates an answer, sets the local description and sends an "answer" message to signalling server with the status "OK", which routes it to the caller. If the callee refuses the call an "answer" message is created with the status "NOK" and an empty parameter.

```
async function handleOffer(payload)
```

Figure 3.28: Offer message handler.

```
await rtcConnection.setRemoteDescription(new RTCSessionDescription(payload.offer));
```

Figure 3.29: RTC connection setting the remote description.

The caller receives from the callee the "answer" message which is handled by the handleAnswer() function. This function verifies the status value and when it is "OK", the user sets the remote description as a new RTC session with the answer and starts sending tracks to the callee. When it receives tracks from the callee, it assigns them to the right video box in the function where the callee can be visually seen.

"candidate" messages are traded between members of a new call to define the route between them. These messages are exchanged between the users through the signaling server. Since the Video Call System includes a TURN server that functions as a relay server, the media tracks pass through this server to connect different private networks. There is a particular configuration that happens at the creation of the RTCPeerConnection that forces the media tracks to go through the relay server, as shown in figure 3.30.

For a user to be able to access the TURN server, it must have the right credentials and the right URL of the server. After the route is defined both sides start

sending tracks through the relay server, that routes them to the destination by checking the header.

```
var rtcConf = {
  iceTransportPolicy: 'relay',
  iceServers:[
    {
        urls: "turn:[redacted]:3478",
        credential: "thiskey",
        username: "thisuser"
    },
]};
```

Figure 3.30: RTC configuration.

Once the rtcConnection state changes to connect, the user receives a console message saying "connected", the statusOnCall() function is called, the list buttons is disabled and the hangUpButton is activated. Whenever this button is pressed, it sends a "hangup" message to the signalling server that in turn sends it to both users.

The handleHangUp() function is activated when the "hangup" message arrives and closes the rtcConnection and calls the setStatusAvailable() function to disable the hangUpButton and enable the user list.

In the case a "refuse" message is received after a call is attempted, the handleRefuse() function is called and it creates an alert saying "Occupied" and closes the rtcConnection.

For the "roomusers" message, the handleRoomUsers() goes through all the users present in the Array and places them and their status on the completelist variable as on click buttons.

3.1.7 Relay Server

The relay server is a TURN server written in Go language. The code was retrieved from an open source in GitHub called Pion [20]. Due to the way Pion is built and the examples in open source, only minor modifications to the source code, such as IP and authentication, were needed, so that the server can run.

The authentication allows the server to confirm which user they are receiving and routing tracks to. This way it guarantees the video gets to the right user and that no other user can receive it.

In the testbed, the relay server is implemented in a virtual machine so that it can simulate a real-life server and work as if it was off site, thanks to the network settings of the virtual machine.

Chapter 4

Quality Testing

Making quality testing possible requires the addition of new blocks to the system that measure the quality metrics values. Additional changes were made in the original system to support the recording of the videoconferences and the streaming of the previously recorded video. This chapter presents the new blocks made and the testing scenario used to measure the performance in the presence of bandwidth limitations and packet errors. It also presents and analyses the performance results.

4.1 Quality Evaluation System

An evaluation system was created to analyse the video quality in the Video Call system, where instead of a live video call, a file is used as the video source. The objective is to reduce the variance of the measurements due to the influence of changing environmental conditions in the received video quality. The quality changes with the lightning conditions and with the video frame sequence differences, which vary between each live videoconference. The other reason for choosing to send the video, instead of just capturing it from the camera, is the need to have a reference video to compare with the received video in the callee, so they can be compared and checked.

The Video Comparison system compares and processes the videos stored in the Video Recording system. Just as shown in figure 4.1, this system receives the video and returns the quality comparison values.

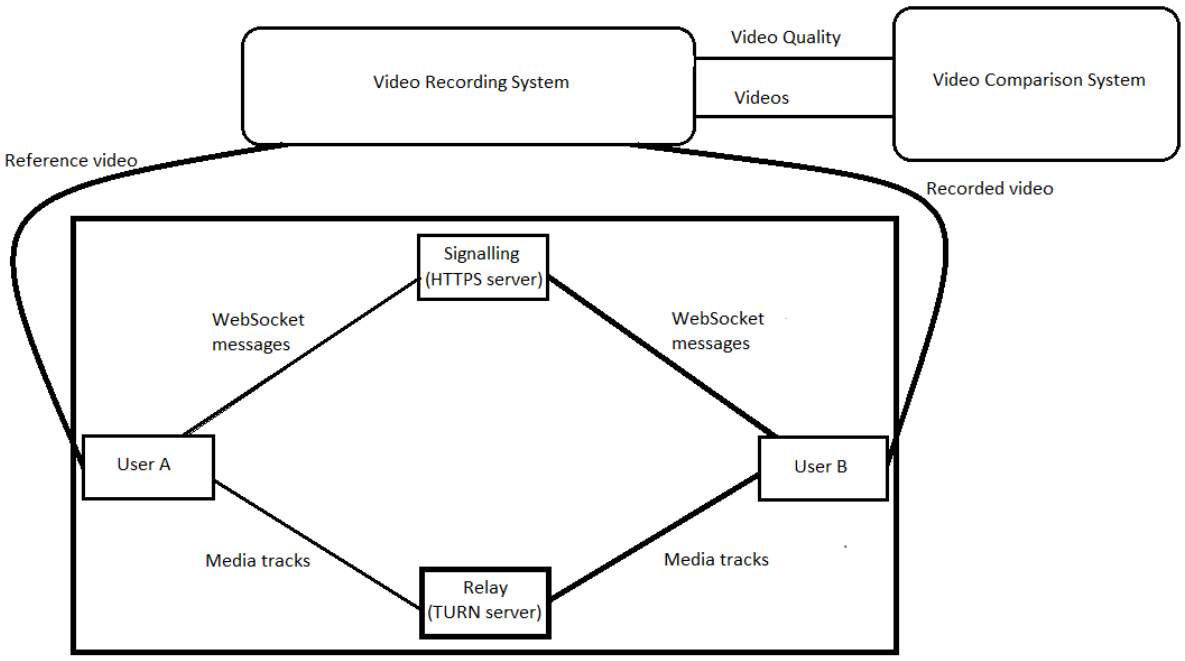


Figure 4.1: Overall view of the entire Quality Evaluation System.

4.1.1 Video Recording System

The modifications done to the original system to the support video quality measurements were done inside the user module, with the addition of new buttons and new functions, so that recording and download of video for the measurement of those qualities could be possible.

4.1.1.1 Video Reference

Creating the reference video required the video call system to run one time while recording the user's camera during a predefined time. A timeout was used to stop the recording at the end of the time. Afterward, the video is downloaded and inserted into a static folder.

Figure 4.2 illustrates the changes done to the HTML code that allow the visualization and control of the video. For the test duration, a command named loop was added to keep the video running sending tracks flowing through the video call system. While observing the HTML, it is possible to verify that the reference video was an MP4 file called senderRef.

A new media streaming object is used to send the video from a file. The code presented in section 3.1.5 was modified: the getMedia() function was modified and the call to assignMedia() function in the start() function was desactivated.

```
<video id="yours" playinline controls>
    <source src="senderRef.mp4" type="video/mp4"/>
</video>
```

Figure 4.2: HTML Code for the reference video.

4.1.1.2 New global variables

New global variables were defined to accommodate the new buttons and the handling of a recorded video, as shown in figure 4.3. Their names and functionality are:

- recordedVideo is the variable allocated to store the recorded video of a video call connection.
- recordButton is the variable created to handle the functions and actions of the record button.
- sendButton is the variable created to handle the functions of the button that send a request to the server to record the caller and callee, simultaneously.
- downloadButton is the variable created to handle the functions of the button that downloads the recorded video.
- mediaRecorder is used to store the media recorder of the quality of video system.
- recordedBlobs is used to store the recorded video.
- theirstream is to store the incoming stream of video.

```
let recordedVideo = document.querySelector('video#recorded');
let recordButton = document.getElementById('record');
let sendButton = document.getElementById('send');
let downloadButton = document.getElementById('download');
let mediaRecorder;
let recordedBlobs;
let theirstream;
```

Figure 4.3: All new global variables added.

4.1.1.3 Recording Functions

Two new functions were created to record the received video, one to start and another to stop recording.

In order to have comparable videos, the recording at the receiver side has to be synchronized with the start of the video. Initial tests where a message was sent to the two users showed that this does not work due to different transmission and processing delays. The solution implemented is to start the video playing only when the connection was confirmed and active, and start recording as soon as the first tracks from the caller get to the callee. This is only possible because RTCPeerConnection calls the ontrack method when a new track is received, allowing to start recording only when the first track arrives.

The startRecording() function, shown in figure 4.4, is a function that receives the stream of video to record as a parameter. It starts by checking which type of media recorder is supported. With that information, it creates the recorder and informs the console. This is followed by the disabling of the sendButton and downloadButton, and a checking to see if there are tracks to be recorded. It then starts recording.

```
function startRecording(recstream)
```

Figure 4.4: startRecording() function.

The stopRecording() function, shown in figure 4.5, is a one line function that just tell the media recorder to stop recording.

```
function stopRecording() {
```

Figure 4.5: startRecording() function.

4.1.1.4 Initial Testing system

Two testing systems were implemented for the thesis. An initial one, with the objective of collecting the reference video from a live videoconference. A the second one, to collect the received videos and measure the quality performance indicators.

This section describes the first Video Recording system used. Where the "record"message was the synchronizing factor of live video recording. But due to not being able to synchronize the recording, this system was disused but kept since it was used to record the reference video.

The "record"message was supposed to be sent by the server to both users at the request of one of the users so that the recording would synchronized but when

A new WebSocket message was added to control the remote recording, named “record”. The message contents are shown in figure 4.6.

```
type     : "record",
destiny  : conndestiny,
origin   : data.origin
```

Figure 4.6: "record"message contents.

The function handleRecordRequest() that appears in figure 4.7, handles the "record"message sent by the server to record. If that user is the caller, it records the outgoing stream, otherwise it records the incoming stream. This caller/callee role matching uses the origin field in in the “record” message. If the user matched with the origin of the request then it was the caller and the other user the callee but if it was not the origin then it was the callee. This function creates a thirty-second timeout that is used to stop the recording. The download of the video is possible after each recording.

```
async function handleRecordRequest(payload)
```

Figure 4.7: User’s "record"message handler.

In the Signalling server, the only change made was the addition of a handler function for the "record"mesage. This handler starts by confirming if the user is in a call, then uses the Calls Map to find who the user is on a call with. If both connections are still active it sends a "record"message to both users.

4.1.1.5 Buttons Handlers

Three handlers were created to deal with the three new buttons, one for each button. The sendButton.onclick() function, shown in figure 4.8 creates and sends a “record” message to the server defining its username as the origin.

```
sendButton.onclick = async function(){
```

Figure 4.8: Send button handler.

The downloadButton.addEventListener() function was created to support the download of the video, as shown in figure 4.9. This function starts by informing the console that it has begun a download, then it creates a new Blob with the

recordedBlobs and the type of video. A Blob is an immutable object that represents raw data. By creating a URL for the Blob, its content becomes obtainable and visible. The function creates a name for the Blob and allocates it. Finally, it starts the download. After finishing, the Blob is removed and erased.

```
downloadButton.addEventListener('click', async ()
```

Figure 4.9: Download button handler.

The recordButton.addEventListener() function is associated to the recordButton and at a click of the button, it calls the startRecording() function providing the user's stream as a parameter and changes the name of the button to "Stop Recording". When the button is clicked again the recording stops and a video becomes available to download.

```
recordButton.addEventListener('click', async ()
```

Figure 4.10: record Button handler.

4.2 Video Comparing System

Now that the system has the video of both sides of a call, it is possible to compare them to quantify the video in another part of this quality evaluation system. This part of the system, called the Video Comparison System, is implemented in a new virtual machine that runs the video quality and comparing algorithms to check the changes from the original video to the callee's received video.

The virtual machine runs a freshly installed Ubuntu 18.04. The only resource received is the folder with all the video recorded, allowing the comparison between them and the reference video. This comparison was done through FFMPEG with the video quality measurements VMAF, Peak signal-to-noise ratio (PSNR) and Structural similarity index measure (SSIM).

FFMPEG is an open-source command-line tool that is used to convert video and audio from one format to another but it can also be used to compare videos through its filters and libraries. It uses the library libvmaf to calculate VMAF. PSNR and SSIM values for pair of videos. FFMPEG is available at [9] and VMAF is available at [24].

The installation of FFMEG followed the instruction in [18], but it took a few attempts using new virtual machines to get the system to work properly, due to cross dependencies not described in the documentation of the tools. For instance,

the fact that VMAF had to be installed in a specific directory to be recognized by FFMPEG.

4.2.1 Quality metrics

Quality metrics are the metrics that quantify the quality of a video. Although there are a lot more quality metrics, only the following three were used in the thesis. They are VMAF, PSNR, and SSIM, which are very briefly described in the next paragraphs.

Video Multimethod Assessment Fusion (VMAF) is an open-source video quality metric that works by comparing two videos, frame by frame, and scoring each of those comparisons to achieve a number between zero and hundred, where in one hundred, the videos are an exact match of one another. Then it averages the whole video so that a final number is obtained.

VMAF was introduced by Netflix to evaluate the quality of their videos delivered over TCP. This metric is used to optimizate the scaling parameters and compression of encoding decisions that match a quality goal. An analysis to the quality metric theoretical concept is made in [17] and an overall view of VMAF is provided in [16], explaining how to interpret the VMAF scores.

Peak signal-to-noise ratio (PSNR) is the ratio between the maximum signal and the corrupting noise of a frame. PSNR can be used as a video quality measure if the video content and CODEC are the same in each testing attempt, according to [23].

Like VMAF, PSNR needs two videos to compare frame by frame and then gives an averaged result of the metric with a value range from zero to one hundred. At zero, the videos are different, and at a one hundred, they are an exact match.

Structural similarity index measure (SSIM) is the structural difference between the original and the distorted image in terms of luminance, contrast, and structure. In this metric, the callee's video is compared to the reference video, frame by frame and the result has a value between 0 and 1, where 0 means the videos are different and 1 means the videos are an exact match. Like in the other two metrics it returns the average of the entire video. Further details about the calculations and metrics used in SSIM are provided in [25].

4.2.2 Video Comparison

All videos recorded are stored by the Video Record System into a specific folder called Tests that is shared with the Video Comparison System through the sharing

settings of the virtual machine. The line command for the sharing is shown in figure 4.11.

```
ubuntu@ubuntu1804:/mnt/hgfs/Tests/repetição$ sudo vmhgfs-fuse .host:// /mnt/hgfs
/ -o allow_other -o uid=1000
```

Figure 4.11: Sharing folder command.

Now that the videos are available inside the Video Comparison System, the video comparison is done for each metric.

For VMAF calculation, the command line shown in figure 4.12 was used, producing the output shown in figure 4.13. In the output, it can be seen the CODEC used, the inputed videos, number of frames processed, the speed of the process, the video's size, length, and the VMAF score. For testing effects, the audio is disabled hence its size is zero.

The filter_complex option is used when there are multiple input videos. In the example of figure 4.12, the callee's video is receiver0.1packet3.mp4, and the reference video is the senderRef.mp4. The same example is used in all following figures on this section.

```
ubuntu@ubuntu1804:/mnt/hgfs/Tests/repetição$ ffmpeg -i receiver0.1packet3.mp4 -i
 senderRef.mp4 -filter_complex libvmaf -f null  -
```

Figure 4.12: VMAF calculating command.

```
ffmpeg version N-99557-g6bdfea8 Copyright (c) 2000-2020 the FFmpeg developers
  built with gcc 7 (Ubuntu 7.5.0-3ubuntu1~18.04)
  configuration: --prefix=/home/ubuntu/ffmpeg_build --pkg-config-flags=--static --extra-cflags=-I/home/ubuntu/ffmpeg_build/includ
lm' --bindir=/home/ubuntu/bin --enable-gpl --enable-gnutls --enable-libass --enable-libfdk-aac --enable-libfreetype --enable-libm
 --enable-libx265 --enable-libvmaf --enable-nonfree
  libavutil      56. 60.100 / 56. 60.100
  libavcodec     58.111.100 / 58.111.100
  libavformat    58. 62.100 / 58. 62.100
  libavdevice    58. 11.102 / 58. 11.102
  libavfilter     7. 87.100 /  7. 87.100
  libswscale      5.  8.100 /  5.  8.100
  libswresample   3.  8.100 /  3.  8.100
  libpostproc    55.  8.100 / 55.  8.100
Input #0, matroska,webm, from 'receiver0.1packet3.mp4':
  Metadata:
    encoder         : Chrome
  Duration: N/A, start: 0.000000, bitrate: N/A
    Stream #0:0(eng): Video: vp8, yuv420p(progressive), 640x480, SAR 1:1 DAR 4:3, 1k tbr, 1k tbn, 1k tbc (default)
    Metadata:
      alpha_mode      : 1
Input #1, matroska,webm, from 'senderRef.mp4':
  Metadata:
    encoder         : Chrome
  Duration: N/A, start: 0.000000, bitrate: N/A
    Stream #1:0(eng): Video: vp8, yuv420p(progressive), 640x480, SAR 1:1 DAR 4:3, 14.83 fps, 14.83 tbr, 1k tbn, 1k tbc (default)
    Metadata:
      alpha_mode      : 1
Stream mapping:
  Stream #0:0 (vp8) -> libvmaf:main
  Stream #1:0 (vp8) -> libvmaf:reference
  libvmaf -> Stream #0:0 (wrapped_avframe)
Press [q] to stop, [?] for help
Output #0, null, to 'pipe:':
  Metadata:
    encoder         : Lavf58.62.100
    Stream #0:0: Video: wrapped_avframe, yuv420p, 640x480 [SAR 1:1 DAR 4:3], q=2-31, 200 kb/s, 1k fps, 1k tbn, 1k tbc (default)
    Metadata:
      encoder         : Lavc58.111.100 wrapped_avframe
frame=  886 fps= 87 q=-0.0 Lsize=N/A time=00:00:29.89 bitrate=N/A speed=2.92x
video:464kB audio:0kB subtitle:0kB other streams:0kB global headers:0kB muxing overhead: unknown
[libvmaf @ 0x55db8de06700] VMAF score: 80.405124
```

Figure 4.13: VMAF output.

For PSNR calculations, the command line shown in figure 4.14 is used and outputs the results shown in figure 4.15. These results are a bit different from the ones with VMAF since they are given in YUV format and all the color components have their score, separate from the others. In the PSNR results, there is also the number of frames, size, and length of the video.

```
ubuntu@ubuntu1804:/mnt/hgfs/Tests/repetição$ ffmpeg -i receiver0.1packet3.mp4 -i
 senderRef.mp4 -filter_complex "psnr" -f null  -
```

Figure 4.14: PSNR calculating command.

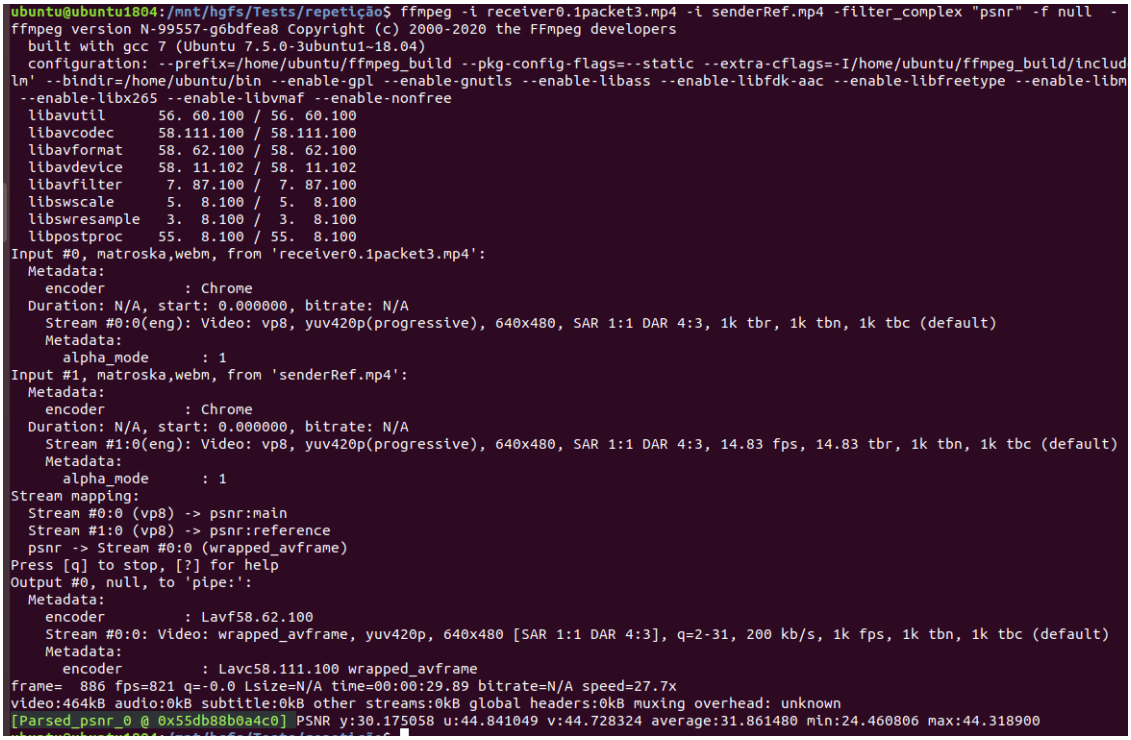


Figure 4.15: PSNR output.

The SSIM values are measured using the command shown in figure 4.16 , and output the results that appear in figure 4.17. As with PSNR, results are shown in the YUV format, but the score is from zero to one.

```
ubuntu@ubuntu1804:/mnt/hgfs/Tests/repetição$ ffmpeg -i receiver0.1packet3.mp4 -i
 senderRef.mp4 -filter_complex "ssim" -f null  -
```

Figure 4.16: SSIM calculating command.

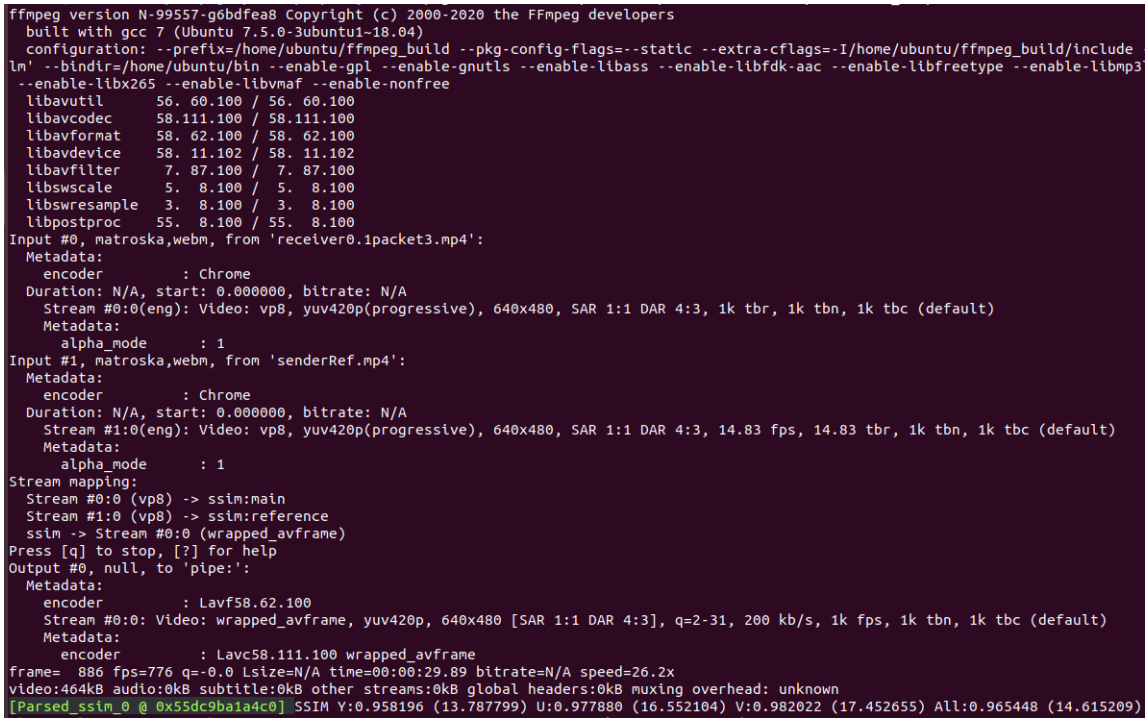


Figure 4.17: SSIM output.

4.3 Results

Using the Quality Evaluation System and the reference video, the performance of the Video Call System was measured and analysed considering different tests and restrictions. There were three types of tests done:

- Bandwidth strangulation, reducing the bandwidth until the video call can no longer be transmitted. Each test is then passed through the Video Comparison System to obtain the quality metrics and the number of frames. The bandwidth strangulation is made through the Network Adapter Advanced Settings of the virtual machine where the relay server is located, shown in figure 4.18. Those settings allow the simulation of different bandwidths, as if the Video Call System had bandwidth limitations, in its network.

- Packet Loss Test. The packet loss of the relay server was increased until the calls would not last the entire length of the reference video. This was possible because of the Network Adapter Advanced Settings of the virtual machine where the relay server is located, shown in figure 4.18.

- One-hour long test for Jitter variation according to bandwidth length. Jitter is the variation in latency over time in a end-to-end system, this implies that data going through the system is not constant and delays may happen.

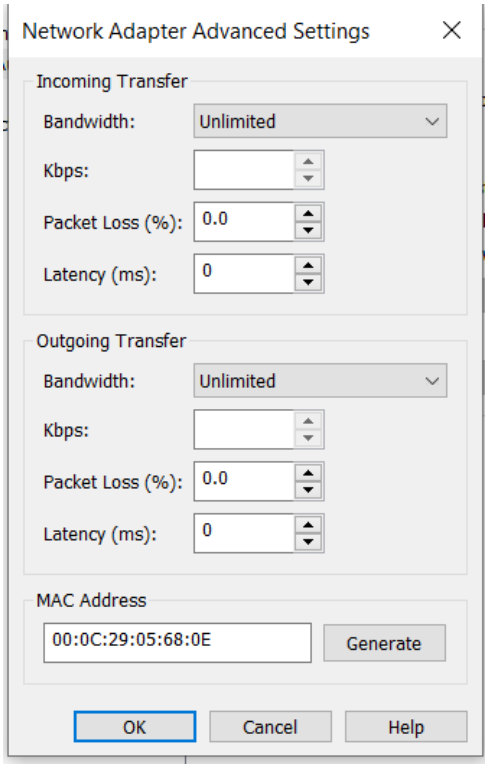


Figure 4.18: Network Adapter Avanced Settings.

4.3.1 Reference video

The reference video is a thirty second recording of the camera while doing continuous movement with the upper body and head. The recording happened on November 2, 2020 at 14:54, with natural light coming from behind through a window.

Comparing the reference video with itself before passing through the Video Call system was necessary to confirm that the comparison system was working properly. The results from this test are that VMAF scores works providing the output in figure 4.19. The expected result for a comparison was 100 but 99.83 was obtain which is close to the expected number. Also from figure 4.19, it can be seen that the CODEC used by the Video Call system is VP8, that the reference video has 14.83 frames per second and that the enconder was Chrome, meaning that the video was retrieved while using a Chrome web browser.

When the reference video passes through the relay system without having any network limitations, we measured the performance results presented in table 4.1. The results show that even in a "perfect" environment the VMAF score is not one hundred. This means that the CODECs used by the system are not perfect and that even in this type of environments the quality of the video decreases slightly.

```
ffmpeg version N-99557-g6bdfea8 Copyright (c) 2000-2020 the FFmpeg developers
  built with gcc 7 (Ubuntu 7.5.0-3ubuntu1~18.04)
  configuration: --prefix=/home/ubuntu/ffmpeg_build --pkg-config-flags=--static --extra-cflags=-I/home/ubuntu/ffmpeg_build/include --extr
lm' --bindir=/home/ubuntu/bin --enable-gpl --enable-gnutls --enable-libass --enable-libfdk-aac --enable-libfreetype --enable-libmp3lame -
 --enable-libx265 --enable-libvmaf --enable-nonfree
  libavutil      56. 60.100 / 56. 60.100
  libavcodec     58.111.100 / 58.111.100
  libavformat    58. 62.100 / 58. 62.100
  libavdevice    58. 11.102 / 58. 11.102
  libavfilter     7. 87.100 /  7. 87.100
  libswscale      5.  8.100 /  5.  8.100
  libswresample   3.  8.100 /  3.  8.100
  libpostproc    55.  8.100 / 55.  8.100
Input #0, matroska,webm, from 'senderRef.mp4':
  Metadata:
    encoder         : Chrome
  Duration: N/A, start: 0.000000, bitrate: N/A
    Stream #0:0(eng): Video: vp8, yuv420p(progressive), 640x480, SAR 1:1 DAR 4:3, 14.83 fps, 14.83 tbr, 1k tbn, 1k tbc (default)
    Metadata:
      alpha_mode      : 1
Input #1, matroska,webm, from 'senderRef.mp4':
  Metadata:
    encoder         : Chrome
  Duration: N/A, start: 0.000000, bitrate: N/A
    Stream #1:0(eng): Video: vp8, yuv420p(progressive), 640x480, SAR 1:1 DAR 4:3, 14.83 fps, 14.83 tbr, 1k tbn, 1k tbc (default)
    Metadata:
      alpha_mode      : 1
Stream mapping:
  Stream #0:0 (vp8) -> libvmaf:main
  Stream #1:0 (vp8) -> libvmaf:reference
  libvmaf -> Stream #0:0 (wrapped_avframe)
Press [q] to stop, [?] for help
Output #0, null, to 'pipe:':
  Metadata:
    encoder         : Lavf58.62.100
    Stream #0:0: Video: wrapped_avframe, yuv420p, 640x480 [SAR 1:1 DAR 4:3], q=2-31, 200 kb/s, 14.83 fps, 14.83 tbn, 14.83 tbc (default)
    Metadata:
      encoder         : Lavc58.111.100 wrapped_avframe
frame=  444 fps= 78 q=-0.0 Lsize=N/A time=00:00:29.93 bitrate=N/A speed=5.27x
video:232kB audio:0kB subtitle:0kB other streams:0kB global headers:0kB muxing overhead: unknown
[libvmaf @ 0x55bb4de26600] VMAF score: 99.837079
```

Figure 4.19: VMAF output for the reference video.

For the reference video, the PSNR and SSIM were calculated and show a similar variation, to VMAF.

Figure 4.20 shows the packets per second exchange between users while the reference video is being transmitted through the relay server. After the initial spike, the flow of packets stabilizes between 900 and 950 packets per second.

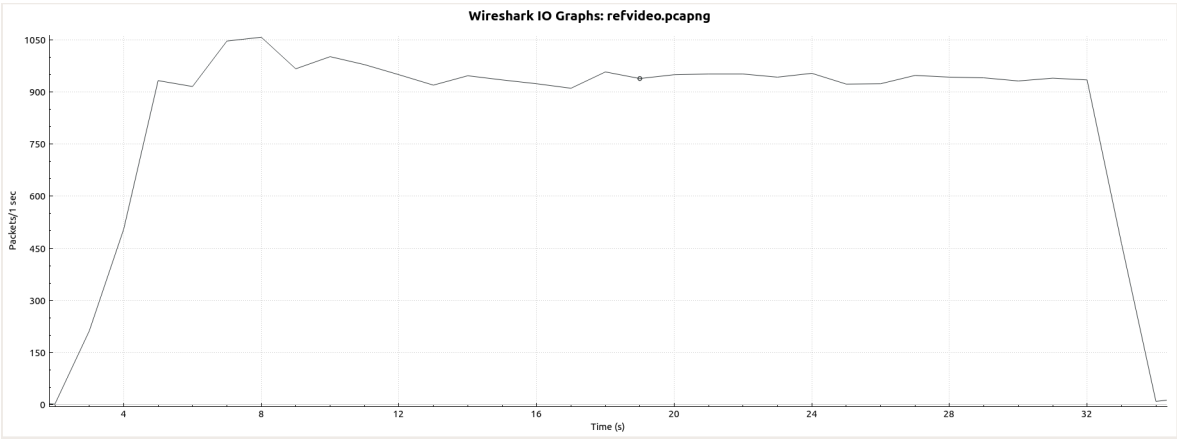


Figure 4.20: Packets per second for the reference video.

Video name	Bandwidth	PacketLoss(%)	VMAF	PSNR	PSNR2	PSNR3	PSNR4	SSIM	SSIM2	SSIM3	SSIM4	Jitter	Jitter2	Frames
videoRef	unlimited	0	87.2	35.6	34.01	44.71	44.91	0.974	0.97	0.979	0.98	0.017	0.0169	884

Table 4.1: Table with Video quality results of the reference video.

As more tests were done and evaluated, it was found that PSNR and SSIM were just confirming what VMAF was already showing, so the results presented in this chapter only show the VMAF metric values.

The jitter over an hour test run without network limitations in the Video Call System and the results are shown in figure 4.21. It can be seen that there are variations in the Jitter over time. These variations to the latency in the system can be because of the variation of the bitrate in the CODEC compression and decompression of video, the bandwidth variations inside the video itself due to movement of the person in the video and process scheduling limitations inside the computer where the tests were run. The computer has 12 logical processors and 16 GB of RAM, and was running one virtual machines in the host operating system, which was Windows.

Another point that can be checked is the maximum and minimum jitter considering intervals of tweenty seconds, that is shown in table 4.1. The jitter measured in ideal network conditions defines a baseline for the end-to-end jitter.

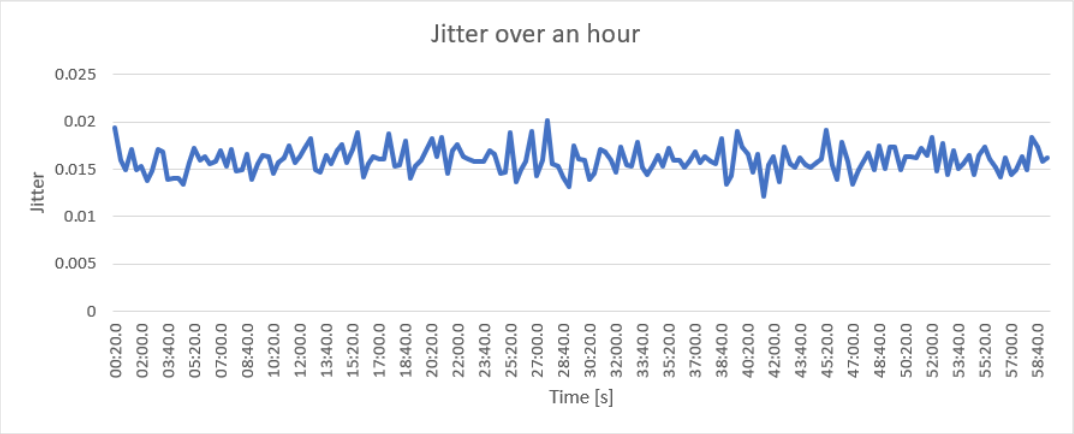


Figure 4.21: Jitter over an hour without bandwidth constraints.

4.3.2 Jitter over an hour

This section studies the influence of bandwidth limitation in the jitter at the receiver. Due to time constraints, the tests were limited to three bandwidth configurations: 180, 210 and 310 [Kb/s]. Table 4.2 presents the measured jitter averages and extreme values. It shows that the highest average was measured for a bandwidth of 180 [Kb/s] while the lowest for unlimited bandwidth. The average jitter and the jitter variation decreased each time the bandwidth increased.

Three jitter graphics are shown in figure 4.22, respectively for 310, 210 and 180 [Kb/s] bandwidth.

Bandwidth(kb/s)	unlimited	180	210	310
Average Jitter	0.015979	0.03068	0.02897	0.0266
Maximum Jitter	0.020122	0.0828	0.07773	0.04442
Minimum Jitter	0.012189	0.00887	0.00973	0.00892
Jitter Variation	0.007933	0.07393	0.068	0.0355

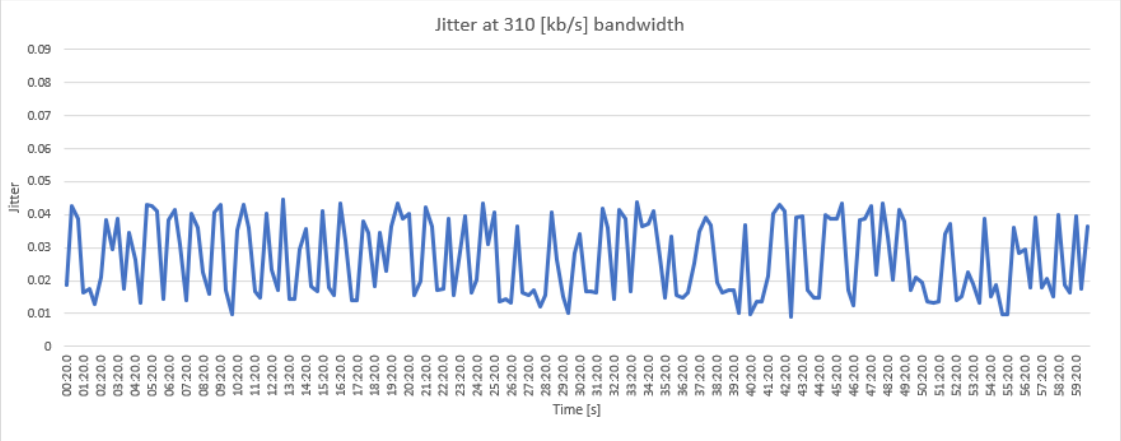
Table 4.2: Table with the different Jitter according to the bandwidth tested.

The Jitter variation in figure 4.22a is constant and has no major spikes, which translates into a smoother video call with no major desynchronization from the reference video but still with some visual delay in the video call.

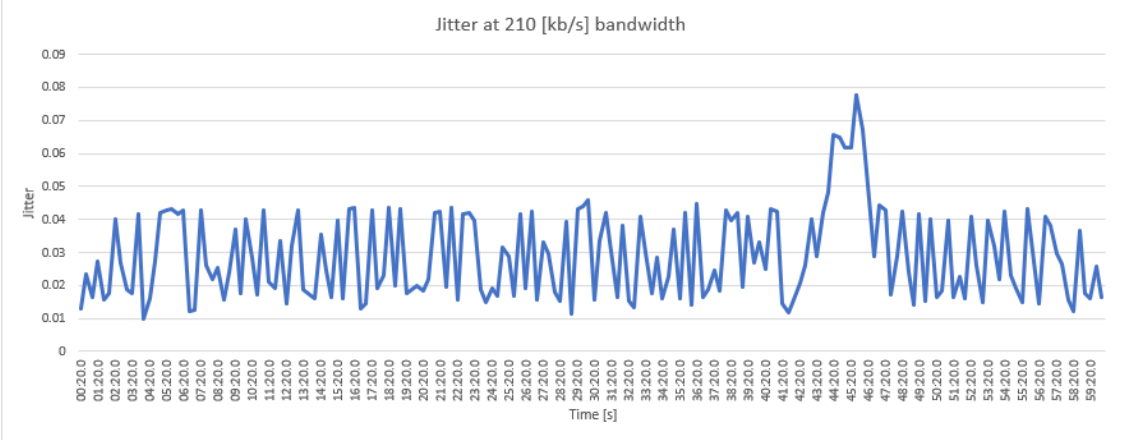
In figure 4.22b there is one major spike in the jitter that lasts from 43:00 to 47:00. This spike can be attributed to the increased transmission delay which results in a desynchronization of the video with noticeable degration of its quality.

In figure 4.22c, there are three major spikes from 15:20 to 18:20, at 35:20, and at 36:20, a medium spike at 54:00. All these spikes translated into a skip of frames with a relative fast recovery except the first one where it took more than a minute to recover, leading to frozen video seen by the callee for that time.

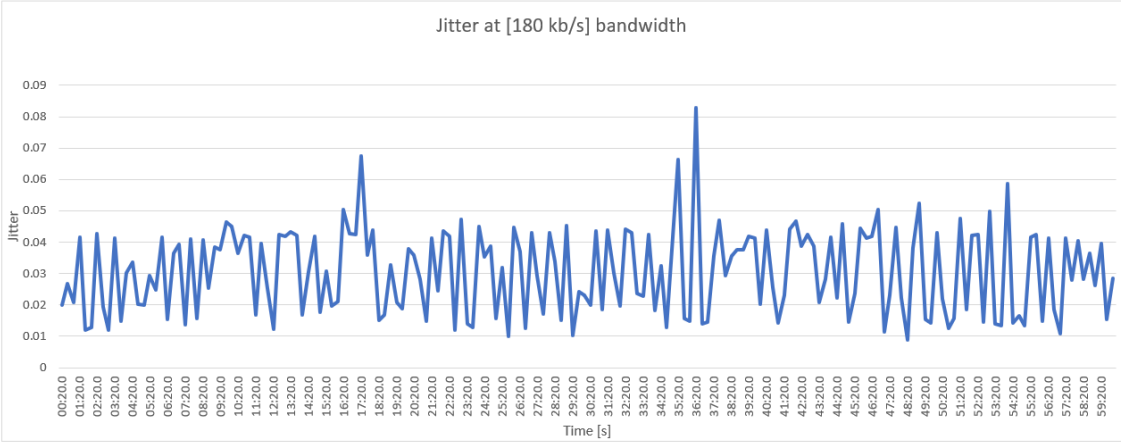
Comparing the three graphics and figure 4.21, it is clear that a smaller bandwidth increases the jitter, not only producing a wider vairation of values most of time but also the spikes happen more often, which reveals that more freezes and skipping of frames on the video will happen. It is important to notice that the spikes in jitter values may have other causes, such as sudden processing load due to other processes in the computer, such as Windows updates.



(a) 310 [Kb/s]



(b) 210 [Kb/s]



(c) 180 [Kb/s]

Figure 4.22: Jitter over an hour with Bandwidth strangulation.

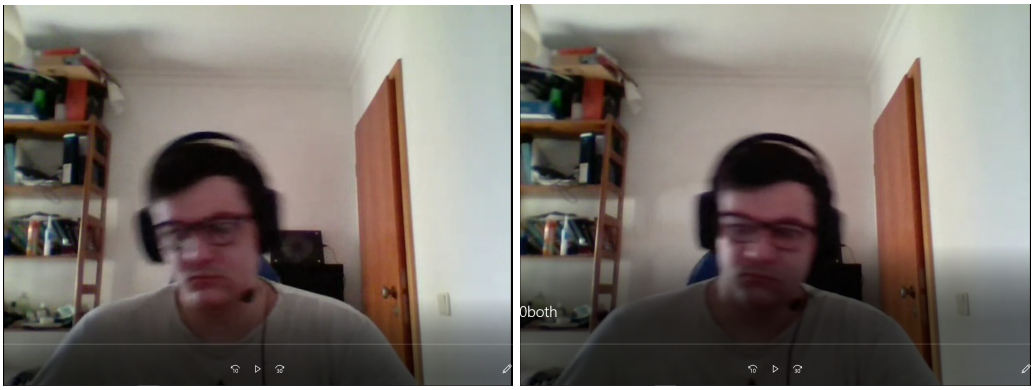
4.3.3 VMAF with Bandwidth strangulation

The main objective of this test was finding until which bandwidth could the video run through the Video Call System and what was the quality drop as the bandwidth kept getting smaller. A bandwidth limit between 165 [Kb/s] and 4000 [Kb/s] was defined and the measured VMAF scores and percentage of frames received are presented in table 4.3.

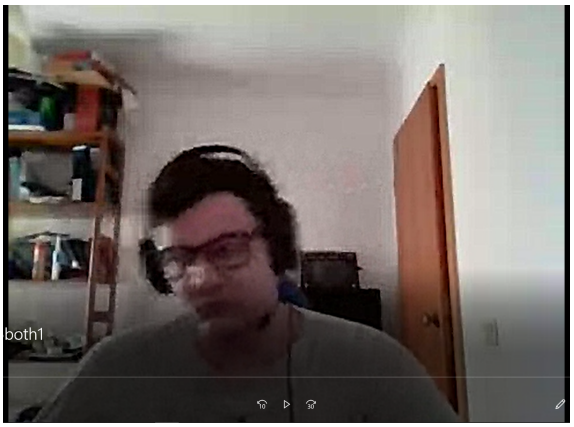
The minimum bandwidth that the system works with is 165 [Kb/s] with a VMAF score of 37.65. This was not the lowest score measured but it was measured for the lowest percentage of frames received at the callee. Although less frames got the callee, they had better quality than the ones received by the callee at 200 [Kb/s], which had a higher percentage of frames received. The worse quality in the bandwidth length resulted in the lowest VMAF score.

The video at 165 [Kb/s] is extremely pixelated and the changes of frames are visible, and movement of the person is blurred, as can be seen in figure 4.23c.

The highest score is 82.285 at 4000 [Kb/s] with 99% of frames received. This score is the closest to the score of the system without any limitation, but it still lost some frames. This video is smooth and with higher visual quality than the rest. Figure 4.23b shows a frame of the video and there are clear differences when compared with the 165 [Kb/s] video, present in figure 4.23c.



(a) 1000 [Kb/s] (b) 4000 [Kb/s]



(c) 165 [Kb/s]

Figure 4.23: Frame shots at different bandwidths on the nineteenth second.

In figure 4.23, the visual difference between 1000 [Kb/s] and 4000 [Kb/s] comes from the resolution of the video frames, and with watching the videos, a clear skip in frames is visible. This difference happens due to the compressing/decompressing limitations for the CODEC at this bandwidth. These limitations become more evident as the bandwidth gets smaller, as seen by comparing a 165[Kb/s] video with a 1000[Kb/s] one.

For 2000, 3000, and 3500[Kb/s], the percentage of frames received was above 100%. This may happen as a result of the initial video call connection instability that can send extra frames and send the same frame multiple times. This event seems to lower the VMAF score of the video call, but not the visual quality of the video at 3000 and 3500 bandwidths. Even at these bandwidths, small breaks and freezes can happen but very rarely and shortly.

Video name	Bandwidth(Kb/s)	VMAF	Frames	Frames (%)
sender165both	165	37.65	432	48.8687783
sender200both	200	35.6325	685	77.4886878
sender300both	300	42.177	692	78.280543
sender400both	400	45.43	852	96.3800905
sender500both	500	43.8585	874	98.8687783
sender700both	700	45.744	879	99.4343891
sender800both	800	45.4735	871	98.5294118
sender900both	900	46.02	879	99.4343891
sender1000both	1000	47.87	869	98.3031674
sender1500both	1500	47.52	882	99.7737557
sender2000both	2000	48.68	886	100.226244
sender2500both	2500	50.745	884	100
sender3000both	3000	55.01	885	100.113122
sender3500both	3500	60.019	885	100.113122
sender4000both	4000	82.285	878	99.321267

Table 4.3: Table with the different VMAF according to the bandwidth tested.

In figure 4.24, we can observe the evolution of the VMAF score as the bandwidth increases. At first, the VMAF score increases quickly from 200 to 400 [Kb/s] then it lowers slightly and continues to slowly grow until 1000 [Kb/s]. At 1000 [Kb/s] it stabilizes and has smalls variations until 2000 [Kb/s], where it begins to increase until it reaches 4000 [Kb/s]. This VMAF score value is the closest to the one in a Video Call system without limitations.

The points depicted in figure 4.25 resulted from an average of VMAF measured on 6 runs for each configuration. The instalibity observed betwenn 400 and 900 [Kb/s] probably resulted from statistical fluctuation of the measurements and it is not meaningful. The figure shows that the VMAF almost does not change between 1000 [Kb/s] and 2000 [Kb/s], only starting to grow noticeably when for higher bandwidths.

Figure 4.25 shows that as the bandwidth increases the percentage of received frames also rapidly increases. At 500 [Kb/s], the percentage is near 100% of the frames of the reference video and it stays near 100% after that.

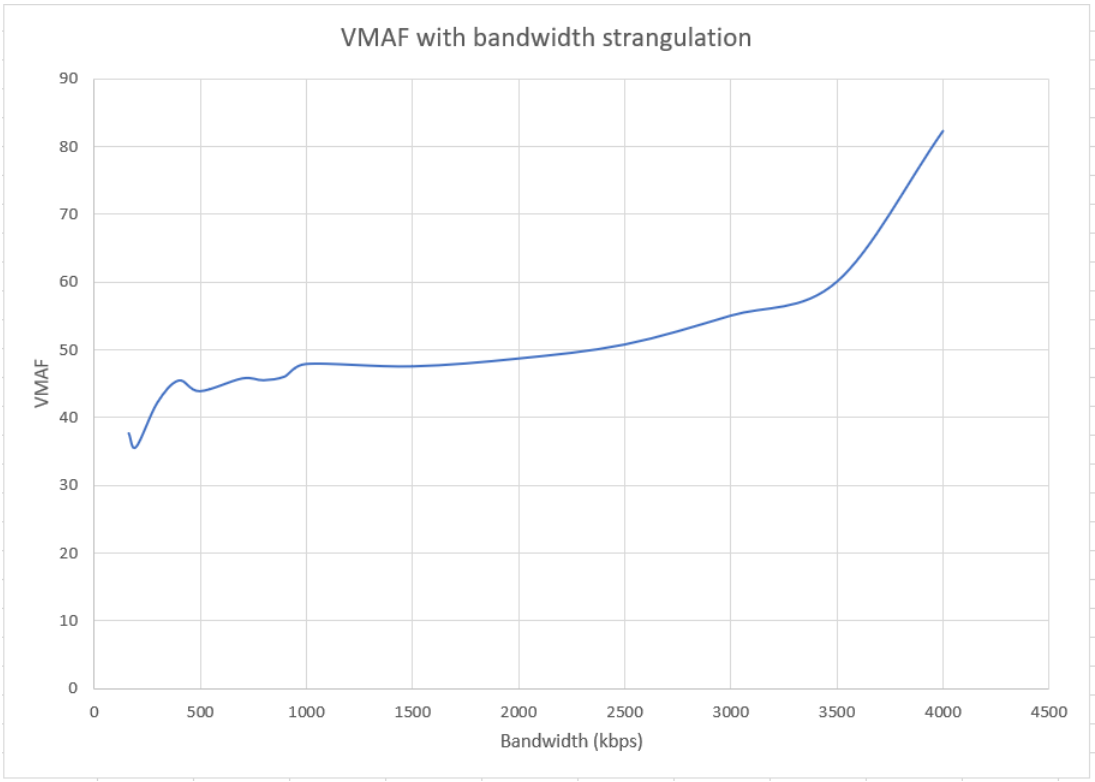


Figure 4.24: VMAF with Bandwidth strangulation.

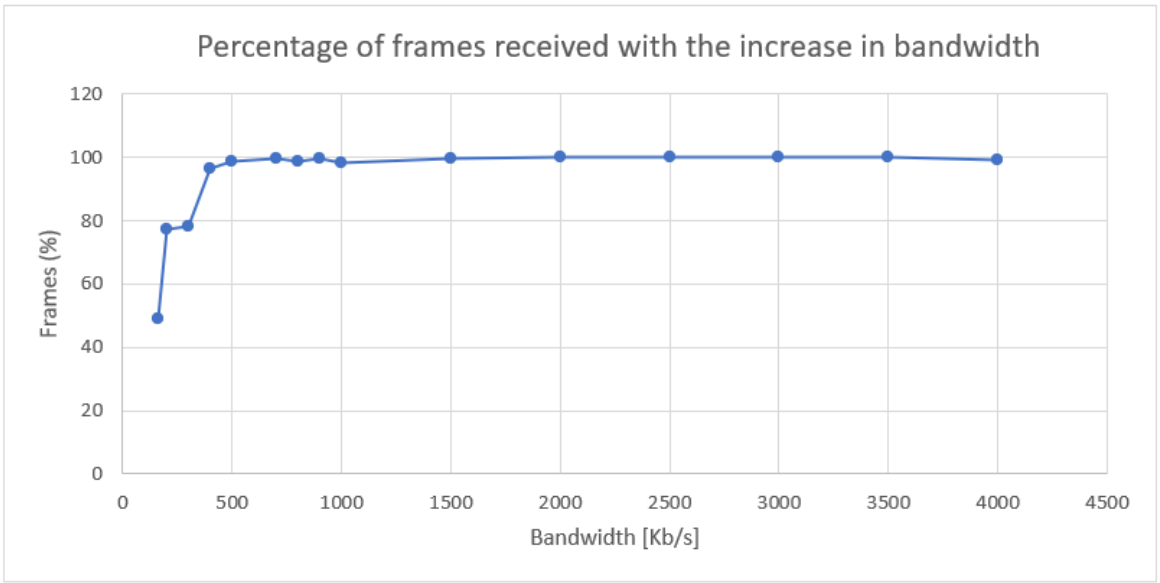


Figure 4.25: Percentage of frames received with the increase in bandwidth.

4.3.4 VMAF with Packet Loss

In the packet loss test, the objective was to find the response of the Video Call System to the increase of the percentage of packet loss. The changes of the quality of the received videos were measured and are presetned in table 4.4.

Just like in the bandwidth test, there are instances of the callee receiving more frames than in a no limitation system, possibly due to the same causes, such as, the initial desynchronization of the recording of the video call.

In terms of percentage of frames lost, the system seems not to loose any until the 10% PER mark, but the quality of each frames seems to be dropping according to the VMAF score in table 4.4. When the videos were watched, there were clear visual changes in the frames as the packet loss inscreased. The videos started skipping frames, losing quality, getting more and more pixelate and by 15%, there is freezing in the video and a slow down and a clear loss of quality.

No prints of frames of the rewatched videos were taken because packet loss is random and very rarely happens in the same frame of two different tries. Also packet loss affects mainly the frame that had a loss of a packet and the following ones, while the strangulation of bandwidth affects every single frame in the video.

Video name	Packet Loss(%)	VMAF	Jitter	Jitter2	Frames	Frames (%)
packet0	0	87.2	0.017	0.0169	884	100
packet0.1	0.1	81.5	0.017	0.0169	884	100
packet1	1	78.5	0.016	0.014	887	100.33937
packet2	2	74.61	0.015	0.03	888	100.45249
packet3	3	72.25	0.0129	0.0145	889	100.56561
packet5	5	71.13	0.0188	0.011	885	100.11312
packet7	7	68.1	0.019	0.015	886	100.22624
packet10	10	57.27	0.017	0.0169	741	83.823529
packet12	12	50.36			878	99.321267
packet14	14	48.36	0.016	0.024	573	64.819005
packet15	15	46.525	0.086	0.0276	579	65.497738
packet16	16	45.4645	0.0127	0.0383	586	66.289593
packet17	17	43.83	0.0214	0.0226	408	46.153846
packet18	18	42.31	0.0177	0.1434	269	30.429864

Table 4.4: Table with the different VMAF according to the packet loss tested.

The system can work at 18% with low quality and the call fails most of the time before the video ends. Due to the lack of frames that reach the callee, the call breaks and disconnects. So the number of tests that successfully passed through the Quality Evaluation System was small. It is seen in table 4.4, that at 18% only 30.43% of the frames arrived at the callee. This confirms that the call will be dropped most of the times since only one in every three frames got to the callee, compared with the system without limitation.

The 12% packet loss setup, in terms of VMAF score, is alined with the rest of the results, but in terms of frames, it is not alined and seems that it would need more tests to find if that is an unexpected occurrence or just a lack of testing at this packet loss percentage.

In figure 4.26, it is observed that the percentage of frames received decreases to around 64% at 14% packet loss and maintains it for 2% more around that value before it drops to 46.15% at 17% and then 30% at 18%.

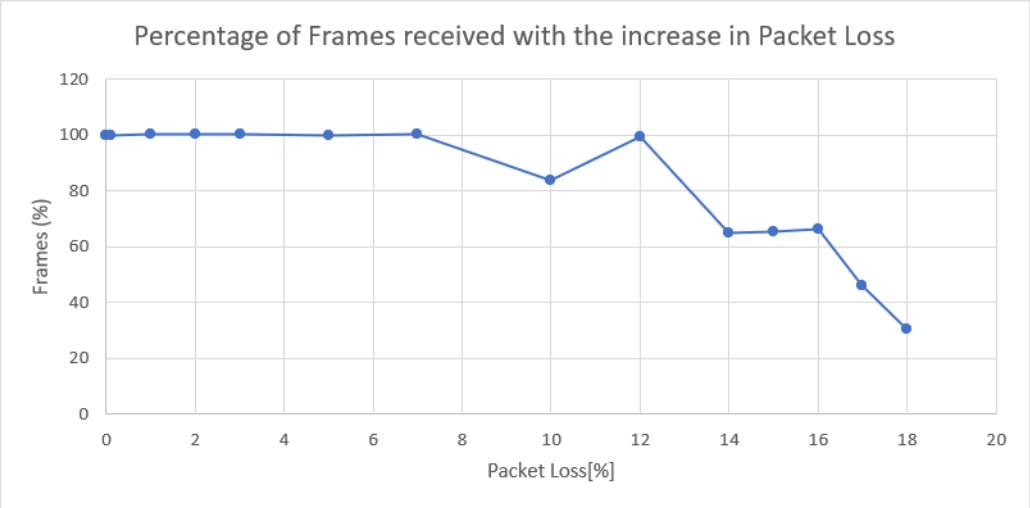


Figure 4.26: Percentage of frames received with Packet Loss.

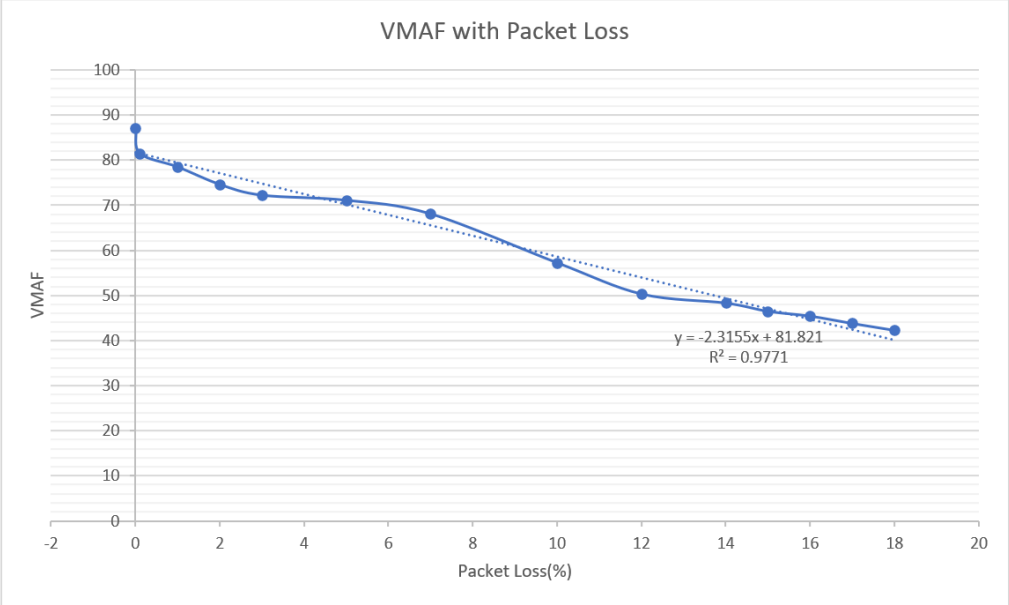


Figure 4.27: VMAF with Packet Loss.

In figure 4.27, there is a decrease in VMAF score as the Packet Loss increases. The VMAF score decreases 6 points as packet loss is increased from 0 to 0.1.

Then it slowly decreases by 2.34 on average for each 1% packet loss increase, as shown in the linear regression done in figure 4.27. The largest point of decrease is between 6% and 12% where it goes from about 70 to 50 VMAF score. The linear regression shows that the decrease in VMAF score value is almost linear and has small discrepancies or a PER between 0.1% and 18%.

Chapter 5

Conclusions and Future Work

An adaptive live streaming system was developed with a simple implementation using WebRTC. This software is capable of live video streaming on a peer-to-peer basis adding the benefit that the web application can be accessed from a mobile phone or a computer.

The Video Call system is a simple live video call web application. This application captures the media of a user and transmits it to another user through a relay server. The control and management of the users are allocated to the Signalling server. This server stores the information about every online user and its current status and routes messages from the users.

The Quality Testing system allowed the testing of the Video Call system through comparing the video of the caller and callee. This comparison provided results that were analyzed using the quality metrics. They provided an improved understanding of how the system reacts to bandwidth strangulation and the increase of the packet loss. The comparison was predominantly done with the VMAF quality metric and using the percentage of frames received.

To better understand the implications of the results, future studies should use different CODECs and compare the results obtained while using a similar system set up to check for changes and possible improvements.

Bibliography

[1] J. G. Apostolopoulos, W. T. Tan, and S. J. Wee. "Video streaming: Concepts, algorithms, and systems." In: *Handbook of Video Databases: Design and Applications* (2003), pp. 831–864.

[2] A. C. Approach. "The Internet and Its Protocols." In: *The Internet and Its Protocols* (2004). DOI: 10.1016/b978-1-55860-913-6.x5019-5.

[3] J. Bankoski, P. Wilkins, and Y. Xu. "Technical overview of VP8, an open source video codec for the web." In: *Proceedings - IEEE International Conference on Multimedia and Expo* (2011). ISSN: 19457871. DOI: 10.1109/ICME.2011.6012227.

[4] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, and R. Zimmermann. "A survey on bitrate adaptation schemes for streaming media over HTTP." In: *IEEE Communications Surveys and Tutorials* 21.1 (2019), pp. 562–585. ISSN: 1553877X. DOI: 10.1109/COMST.2018.2862938.

[5] B. Bross. "Overview of the HEVC video coding standard." In: *Next Generation Mobile Broadcasting* 262.July (2013), pp. 291–318. DOI: 10.1201/b14186-12.

[6] Y. Chen, D. Murherjee, J. Han, A. Grange, Y. Xu, Z. Liu, S. Parker, C. Chen, H. Su, U. Joshi, C. H. Chiang, Y. Wang, P. Wilkins, J. Bankoski, L. Trudeau, N. Egge, J. M. Valin, T. Davies, S. Midtskogen, A. Norkin, and P. De Rivaz. "An Overview of Core Coding Tools in the AV1 Video Codec." In: *2018 Picture Coding Symposium, PCS 2018 - Proceedings* (2018), pp. 41–45. DOI: 10.1109/PCS.2018.8456249.

[7] S. Einnahmen and W. D. Waren. "Report 2019." In: (2019).

[8] A. Fecheyr-Lippens. "A Review of HTTP Live Streaming Table of Contents." In: *Memory* January 2010 (2010). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.466.6043{\&}rep=rep1{\&}type=pdf.

[9] "FFMPEG." In: (). URL: https://ffmpeg.org/.

[10] H. Kalva. "Standards: The H.264 video coding standard." In: *IEEE Multimedia* 13.4 (2006), pp. 86–90. ISSN: 1070986X. DOI: 10.1109/MMUL.2006.93.

[11] H. Koumaras, M. A. Kourtis, and D. Martakos. "Benchmarking the encoding efficiency of H.265/HEVC and H.264/AVC." In: *2012 Future Network and Mobile Summit, FutureNetw 2012* (2012), pp. 1–7.

[12] J. Kua, G. Armitage, and P. Branch. "A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming over HTTP." In: *IEEE Communications Surveys and Tutorials* 19.3 (2017), pp. 1842–1866. ISSN: 1553877X. DOI: 10.1109/COMST.2017.2685630.

[13] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W. T. Chang, and Z. Shi. "The quic transport protocol: Design and internet-scale deployment." In: *SIGCOMM 2017 - Proceedings of the 2017 Conference of the ACM Special Interest Group on Data Communication* (2017), pp. 183–196. DOI: 10.1145/3098822.3098842.

[14] S. Lederer and S. Lederer. "Video Developer Report 2017." In: (2017).

[15] X. Lei, X. Jiang, and C. Wang. "Design and implementation of streaming media processing software based on RTMP." In: *2012 5th International Congress on Image and Signal Processing, CISP 2012* Cisp (2012), pp. 192–196. DOI: 10.1109/CISP.2012.6469981.

[16] Z. Li, C. Bampis, J. Novak, A. Aaron, and K. Swanson. "VMAF : The Journey Continues VMAF at Net ix Codec Comparisons." In: June 2016 (2019), pp. 1–13.

[17] Z. Luo, Y. Huang, X. Wang, R. Xie, and L. Song. "VMAF oriented perceptual optimization for video coding." In: *Proceedings - IEEE International Symposium on Circuits and Systems* 2019-May (2019). ISSN: 02714310. DOI: 10.1109/ISCAS.2019.8702513.

[18] *Ottverse installation guide*. URL: https://ottverse.com/vmaf-ffmpeg-ubuntu-compilation-installation-usage-guide/.

[19] H Parmar and E. M. Thornburgh. "Copyright Adobe Systems Incorporated." In: (2012), pp. 1–52.

[20] Pion. *Pion TURN Server*. URL: https://github.com/pion/turn (visited on 11/05/2020).

[21] I. Sodagar. "The MPEG-dash standard for multimedia streaming over the internet." In: *IEEE Multimedia* 18.4 (2011), pp. 62–67. ISSN: 1070986X. DOI: 10.1109/MMUL.2011.71.

[22] T. Stütz and A. Uhl. "A survey of H.264 AVC/SVC encryption." In: *IEEE Transactions on Circuits and Systems for Video Technology* 22.3 (2012), pp. 325–339. ISSN: 10518215. DOI: 10.1109/TCSVT.2011.2162290.

[23] E. G. Turitsyna and S Webb. "Simple design of FBG-based VSB filters for ultra-dense WDM transmission ELECTRONICS LETTERS 20th January 2005." In: *Electronics letters* 41.2 (2005), pp. 40–41. DOI: 10.1049/el.

[24] *Video Multi-Method Assessment Fusion- VMAF*. URL: https://github.com/Netflix/vmaf.

[25] S. Wang, A. Rehman, Z. Wang, S. Ma, and W. Gao. "Rate-SSIM optimization for video coding." In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings* (2011), pp. 833–836. ISSN: 15206149. DOI: 10.1109/ICASSP.2011.5946533.

[26] P. Zhao, J. Li, J. Xi, and X. Gou. "A mobile real-time video system using RTMP." In: *Proceedings - 4th International Conference on Computational Intelligence and Communication Networks, CICN 2012* (2012), pp. 61–64. DOI: 10.1109/CICN.2012.18.