2014

Instituto Politécnico de Coimbra

INSTITUTO SUPERIOR DE ENGENHARIA DE COIMBRA

Network Distributed 3D Video Quality Monitoring System

MESTRADO EM AUTOMAÇÃO E COMUNICAÇÕES EM SISTEMAS DE ENERGIA

AUTOR | Nuno Alexandre Bettencourt Martins

ORIENTADOR | Prof. Doutor Fernando José Pimentel Lopes

Coimbra, setembro 2014

Departamento
de Engenharia Electrotécnica

Network Distributed 3D Video Quality Monitoring System

Trabalho de Projeto apresentado para a obtenção do grau de Mestre em Automação e Comunicações em Sistemas de Energia

Autor
Nuno Alexandre Bettencourt Martins

Orientador
Doutor Fernando José Pimentel Lopes
Instituto Superior de Engenharia de Coimbra

Coimbra, setembro, 2014

Acknowlegments

I would like to declare my sincere acknowledgements to everyone that helped me during this research work.

I would like to begin by expressing my gratitude to my supervisor, Professor Fernando Lopes whose guidance and availability were essential for this project completion.

I would like to thank Professor Luís Cruz and the *3DVQM - 3D Video Quality Monitor Project* (IT/LA/P01131/2011) the opportunity to work as a researcher in this project and for the guidance, support, time to discuss technical aspects and add new ideas during the work development. I also would like to thank Instituto de Telecomunicações for the laboratory facilities and conditions to accomplish this work. Thanks to my research lab colleagues Guilherme, Thaísa and specially Pedro for the help on completing the project demonstration.

Finally, a special thanks to my parents, Rosa and Armindo, my sister Cátia, my girlfriend Guida and my great friend Alex. To them I owe all that I have become. Thank you for being there.

Abstract

This project description presents a research and development work whose primary goal was the design and implementation of an Internet Protocol (IP) network distributed video quality assessment tool. Even though the system was designed to monitor H.264 three-dimensional (3D) stereo video quality it is also applicable to different formats of 3D video (such as texture plus depth) and can use different video quality assessment models making it easily customizable and adaptable to varying conditions and transmission scenarios.

The system uses packet level data collection done by a set of network probes located at convenient network points, that carry out packet monitoring, inspection and analysis to obtain information about 3D video packets passing through the probe's locations. The information gathered is sent to a central server for further processing including 3D video quality estimation based on packet level information.

Firstly an overview of current 3D video standards, their evolution and features is presented, strongly focused on H.264/AVC and HEVC. Then follows a description of video quality assessment metrics, describing in more detail the quality estimator used in the work. Video transport methods over the Internet Protocol are also explained in detail as thorough knowledge of video packetization schemes is important to understand the information retrieval and parsing performed at the front stage of the system, the probes.

After those introductory themes are addressed, a general system architecture is shown, explaining all its components and how they interact with each other. The development steps of each of the components are then thoroughly described.

In addition to the main project, a 3D video streamer was created to be used in the implementation tests of the system. This streamer was purposely built for the present work as currently available free-domain streamers do not support 3D video streaming.

The overall result is a system that can be deployed in any IP network and is flexible enough to help in future video quality assessment research, since it can be used as a testing platform to validate any proposed new quality metrics, serve as a network monitoring tool for video transmission or help to understand the impact that some network characteristics may have on video quality.

Resumo

Este relatório de projeto apresenta o trabalho de pesquisa e desenvolvimento no qual o principal objetivo foi o desenvolvimento e implementação de um sistema de determinação da qualidade de vídeo em redes IP distribuídas. Apesar de ser destinado à monitorização da qualidade de vídeo 3D codificado usando a especificação H.264 para vídeo estereoscópico, é também possível o seu uso para testar diversos outros formatos de vídeo 3D (tais como textura mais profundidade) assim como diferentes modelos de determinação de qualidade de vídeo, tornado-o numa solução facilmente personalizável e adaptada a diversas condições e cenários de transmissão.

O sistema utiliza informações recolhidas por um conjunto de sondas, convenientemente distribuídas por diversos pontos da rede, responsáveis por monitorizar, inspecionar e analisar pacotes de rede de maneira a obter informações acerca de vídeo 3D que esteja a passar pelas localizações das sondas. As informações recolhidas são enviadas para um servidor central para serem processadas de modo a estimar a qualidade de vídeo 3D.

Em primeiro lugar é apresentado um resumo da evolução e características dos standards de vídeo 3D, descrevendo em mais detalhe o H.264/AVC e o HEVC. Segue-se depois a descrição de métricas para a determinação da qualidade de vídeo, explicando a métrica utilizada para validar o trabalho desenvolvido. Os métodos de transporte de vídeo sobre redes IP são também descritos, para que se possa perceber com o é efetuada a análise dos pacotes de rede capturados e como deles são determinadas as informações necessárias ao nível das sondas do sistema.

Após a apresentação dos temas que definem o enquadramento do trabalho, segue-se uma descrição da arquitetura geral do sistema, detalhando conceptualmente todos os seus componentes e mostrando a forma como estes interagem para formar um todo funcional. Apresenta-se seguidamente uma descrição pormenorizada dos vários passos que foram implementados no desenvolvimento de cada um dos componentes do sistema global.

Foi criado um emissor de vídeo 3D como complemento ao projeto principal, para que fosse possível testar as implementações realizadas uma vez que não existe até ao momento software livre capaz emitir vídeo por uma rede IP.

O resultado é um sistema capaz de ser instalado em qualquer rede IP, flexível o suficiente para ajudar em futuros avanços no desenvolvimento de métricas de determinação da qualidade de vídeo, uma vez que pode ser usado como plataforma de testes para

validar novas propostas de métricas de qualidade de vídeo. Pode também servir para monitorizar uma rede ajudando a perceber qual o impacto que algumas características da rede possam ter sobre a qualidade de vídeo a ser transmitida.

Contents

List of Figures

List of Tables

List of Code Examples

List of Abbreviations

2D two-dimensional. 2, 11, 31, 34

3D three-dimensional. v, 1–3, 11, 16, 22, 24, 31, 33

AU Access Unit. 8

AVS Audio Video Standard. 5

CABAC Context-adaptive binary arithmetic coding. 7

CAVLC Context-adaptive variable-length coding. 7

CB Coding Block. 9

CSV Classical Stereo Video. 11

CTB Coding Tree Block. xi, 9, 10

CTU Coding Tree Units. 9

CU Coding Unit. 9, 10

DCT Discrete Cosine Transform. 8

DPCM Differential Pulse Code Modulation. 5

ES Elementary Streams. 16, 22

GOP Group of Pictures. 24, 29

GUI Graphical User Interface. 3, 29

HD High Definition. 5

HEVC High Efficiency Video Coding. 3, 9

IP Internet Protocol. v, 2, 3, 15, 20, 29–31, 33

IPTV television broadcast over the internet. 1, 6

JCT-VC Joint Collaborative Team on Video Coding. 9

JVT Joint Video Team. 6

LDV Layered Depth Video. 11

MPEG Moving Pictures Experts Group. 6, 9

MPEG2-TS MPEG 2 Transport Stream. 16, 20, 25, 31, 34

MRS Mixed Resolution Stero. 11

MSE Mean Square Error. 24

MVC Multi-View video Coding. 11, 12, 22

MVD Multi-View Video plus Depth. 11, 56

MVV Multi-View Video. 11

NAL Network Abstraction Layer. 8

NALU NAL Unit. 8, 11, 12, 16

P2P Peer-to-Peer. 1

PB Prediction Block. 10

PES Packetized Elementary Streams. 16, 18–20, 22, 35

PID Packet Identifier. 19, 20, 22, 35

PPS Picture Parameter Sets. 8

PSNR Peak signal-to-noise ratio. 6, 24

PU Prediction Unit. 10

QoE Quality of Experience. 1

QP Quantitization Parameter. 8, 24

RBSP raw byte sequence payload. 8

RDO Rate distortion Optimization. 10

RTP Real-time Transport Protocol. 15, 16, 20

SD Standard Definition. 5

SPS Sequence Parameter Sets. 8

SSIM Structural Similarity. 24, 25

TS Transport Stream. 18–20, 22, 33, 34, 38

TU Transform Unit. 10

UDP User Datagram Protocol. 15, 20, 33

UHD Ultra Hight Definition. 9

V+D Video plus Depth. 11, 56

VCEG Video Coding Experts Group. 6, 9

VCL Video Coding Layer. 7, 8

VoD Video-on-Demand. 1

Chapter 1

Introduction

This chapter presents an introduction to the developed work, contextualizing the reader and presenting the work objectives and the project structure.

1.1 Context and Motivation

With the increasing number of available services such as Video-on-Demand (VoD) and television broadcast over the internet (IPTV), comes the exponential growth of data traffic and bandwidth needs in the communication infrastructure. By 2018 video traffic will represent about 80% of all traffic, not including traffic generated by Peer-to-Peer (P2P) video sharing. Also, every second almost a million minutes of video content will cross the network [7].

In some cases all these services are delivered over unreliable communication channels, causing data losses and ultimately interfere with the user Quality of Experience (QoE). For this reason, the effectiveness of each video service must be monitored and measured for verifying its compliance with the system performance requirements, for benchmarking competing service providers, for service monitoring and automatic parameter setting, netw f price
polici

3I ents of

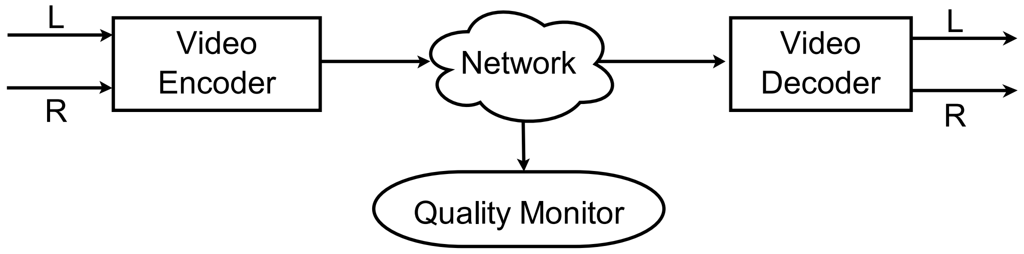


Figure 1.1: 3D Video Quality Monitoring scenario

immersive video technologies making 3D display devices more advanced and affordable for the end user. Nowadays most of the 3D video solutions use multiplexed left and right views, that need special glasses (passive anaglyphic, polarized, active shutter) to channel each view to the correspondent eye, allowing depth perception. Autostereoscopic displays are an emerging display technology that dispenses the need for the use of glasses, opening the way for a more comfortable viewing experience.

Since 3D video is only now starting to become more ubiquitous the interest in 3D video quality assessment is also growing. Due to their differing characteristics, video quality assessment of two-dimensional (2D) is not applicable for 3D video content so new video quality metrics are being developed and tested. Testing these new quality measures is usually done on a controlled environment using simulated errors, but for the cases where the video quality metric being tested is aimed for quality assessment in an IP network, the use of real scenario data or real-time tests is yet to be fully investigated.

For these cases, there is a need for a software tool capable of gathering general video data passing through the transmission network (Figure 1.1), to be used by any video quality metric (2D or 3D) in real-time or use the stored real case scenario data to understand the effect that network conditions may have on the transmitted video. This Project Report explains the work involved in developing and deploying such a system, capable of helping future investigations and developments in the video quality assessment field.

1.2 Objectives

The overall objective of this work is to develop a 3D capable Video Quality Monitor Tool. To achieve this, the following tasks were carried out:

- Study video quality assessment metrics to understand what parameters are usually needed to implement a video quality metric;
- Learn about video transmission schemes. There are several standards for video transmission over IP and one of them had to be chosen taking into consideration several characteristics, such as error resilience, ease of implementation and what parameters can be fed to a video quality algorithm;
- Create a system architecture capable of monitoring a video transmission network. It must be flexible, modular and conceptually simple to understand and use, since the goal is for the system to be used by several users;
- Develop network packet monitoring software capable of detecting and inspecting video streams and generate usable packet level statistical data. This software must

be reliable and capable of retrieve data to be further used on several video quality assessment models mainly for 3D video;

- Specify and implement a database for storage of all gathered data for future analysis. It has to be simple but at the same time avoid any data redundancy;
- Devise a way to test and implement several video quality assessment models. This work is a platform to test new quality models or old ones under several testing conditions, so there has to be a way anyone can implement a quality model;
- Design a Graphical User Interface (GUI) to configure and visualise the system status and the quality being transmitted.

1.3 Outline

Chapter 2 presents a general description of 3D video, as well as a description of video standards evolution, explaining in more detail the H.264 standard and the more recent High Efficiency Video Coding (HEVC).

Chapter 3 describes video streaming protocols over IP networks and video packetization techniques. Understanding how video is transmitted is of great importance because the system is based on packet retrieval and information extraction from network packets.

Video quality metrics are presented in Chapter 4. In this chapter, a video quality metric to use for testing purposes and validate the system implementation is also chosen.

In Chapter 5 the global system architecture is presented and described. It sums up the system with a simplified description of the several components and describes how it is supposed to operate. The constraints of system deployment and what steps must be taken during actual operation are also explained. This Chapter also presents a detailed description, from a development and implementation point-of-view, of each one of the system components that comprise the video monitoring system.

Testing procedures and results of the finalized system are presented and discussed in Chapter 6. Also the development of a 3D video streamer for testing purposes is also described given its importance for the project.

Chapter 7 concludes this project description and proposes future developments to improve or add features to the 3D capable video monitoring tool.

Chapter 2

Video Coding

2.1 Video Standards

Standards help to maximize compatibility, interoperability, safety and quality. Typically for a video coding standard, only the bitstream and syntax are standardized, so the codec developers have freedom to optimize the encoder/decoder as they wish, as long as the encoded result is compatible with the standard.

Fist video systems evolved from oscilloscopes and used CRTs to show a transmitted signal modulated on a radio frequency carrier [9]. An early form of compression was interlacing in which odd and even lines were transmitted alternatively, making possible to save halve the bandwidth or double the vertical resolution.

In 1984, H.120 was standardized making it the first digital video coding technology standard. It used Differential Pulse Code Modulation (DPCM), scalar quantization and variable length code techniques for PAL and NTSC transmission.

In 1990 H.261 was created. It used a hybrid video coding scheme that is still the basis of many coding standards. Motion between frames is estimated using data from previously encoded frames, then the residual difference of that prediction is encoded after being transformed to the frequency domain [10].

Motion JPEG used JPEG still image compression as basis [11]. Standardized in 1992 it basically encoded video as a series of individually encoded images, thus not taking advantage of the temporal redundancy resulting on poor compression ratio. On the other hand it enables easy frame-by-frame video edition.

MPEG-1 (1992) was designed to achieve acceptable video quality at 1,5 Mbit/s at 352 x 288 pixel resolution [12]. The lack of support for interlaced video formats lead to the development of MPEG-2/H.262 release on 1993, supporting Standard Definition (SD) (720 x 480 pixels) and High Definition (HD) (1920 x 1080 pixels) [13].

In 1995 H.263 was released with mobile phone video conferencing in mind enabling video calls at low bitrates for mobile wireless communications [14][15].

In an effort to avoid paying royalties to international patent holding companies, the People's Republic of China standardized Audio Video Standard (AVS) in 2005. Its coding efficiency is comparable to H.264, but has lower computational complexity.

2.1.1 H.264/AVC

H.264/MPEG-4 part 10/AVC [16] is currently the leading standard in use and was developed by the ITU-T Video Coding Experts Group (VCEG) together with the ISO/IEC JTC1 Moving Pictures Experts Group (MPEG), forming a group known as the Joint Video Team (JVT). H.264/AVC is present in applications such as terrestrial, cable and satellite HDTV broadcasting, Blu-ray disks and internet streaming (YouTube, IPTV).

It is a method and format for video compression. An encoder converts video into a compressed format and a decoder converts compressed video back into uncompressed format. It introduces several improvements when compared to previous standards and was created with the increase of high definition popularity and the need for higher coding efficiency.

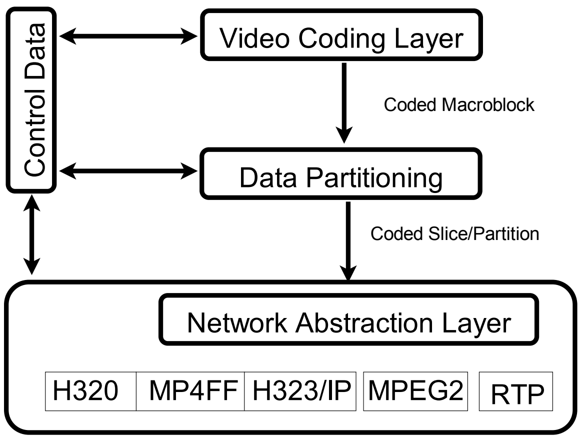


Figure 2.1: H.264 Layer Structure (adapted from [1])

It supports 21 Profiles for several applications making it a versatile codec.

Its main features are:

- **Multiple Reference Frames**: allow the video encoder to choose more than one previously decoded frame to base each macroblock in the next frame. The use of up to 16 previous frames as reference is allowed, considerably improving the compression efficiency as well as the perceived quality;
- **Flexible Motion Prediction Modes** : H.264 supports motion compensated prediction different block sizes. Each macroblock can be divided into smaller partitions with a block size of 16x16, 16x8, 8x16 and 8x8;
- **Search Range** : the increased number of reference frames and the wide search range leads to higher access frequency, and minimal impact on Peak signal-to-noise ratio (PSNR) and bitrate performances;

- **Deblocking Filter** : such filter is applied to blocks in a decoded frame to improve the visual quality by smoothing the sharp edges between blocks that may have been produced at coding time;
- **Context-adaptive variable-length coding (CAVLC) and Context-adaptive binary arithmetic coding (CABAC)** : these two context adaptive coding schemes are employed to improve coding efficiency by adjusting the code tables according to the neighbouring information.

Video Coding Layer

The Video Coding Layer (VCL) defines the techniques for prediction, transforming and encoding, in order to generate a compressed H.264/AVC bitstream.

The video color space used by H.264/AVC takes advantage of the human visual system, that perceives scene content with more sensitivity to brightness and separating it from color information. Color representation is therefore separated into three components called *luminance*(Y) that represents brightness and *chrominance*(Cb and Cr), that represent the color deviation from gray toward blue and red. The human visual system is also more sensitive to luminance than to chrominance so H.264/AVC uses a sampling structure called 4:2:0 where the chrominance component has one fourth of the luminance samples.

Video is captured as a sequence of images (frames), typically between 25 and 30 frames-per-second (fps) to give the viewer the perception of continuous movement. H.264 explores the temporal redundancies present between contiguous frames by computing the displacement between then, thus taking advantage of a reference picture to predict others following using motion information. Depending on the prediction, there are three basic frame coding types:

- **I**-frames, use intra picture prediction and are coded without using any reference from other frames;
- **P**-frames, that support intra picture coding and inter picture predictive coding;
- **B**-frames, which support intra picture coding, inter picture predictive coding and inter picture bi-predictive coding (combination of two predictions).

Frames are arranged by type into a Group of Pictures (GOP), including a I-frame and all subsequent frame combinations until the next I-frame is present (Figure 2.2). I-frames are the initial reference point to decode a video stream and are of great importance for error propagation recovery.

Frames are divided into macroblocks, each with a luminance sample size of 16x16 and 8x8 samples of each one of the chrominance types. Macroblocks are then grouped into Slices that divide a picture into portions that are individually encoded. This division

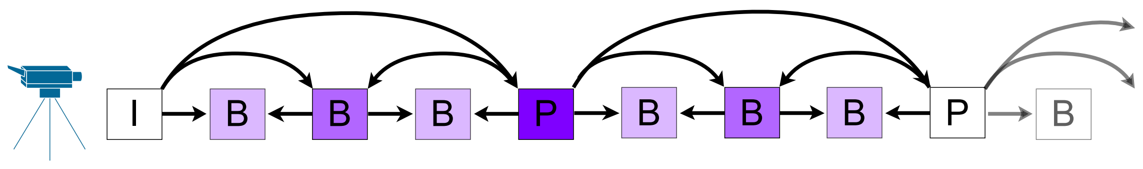


Figure 2.2: GOP frame reference relationship

plays an important role in error resiliency as it confines error propagation to a small area of a picture.

The encoder forms a prediction of the current macroblock based on previously coded data from the current frame using **intra** prediction, or from other frames using **inter** prediction. The resulting prediction is then subtracted from the current macroblock forming a **residual**.

Intra prediction uses 16x16 and 4x4 block sizes to predict the macroblock from previously coded pixels in the same frame [2]. This is done by extrapolating the values of previously coded neighbouring pixels to form a prediction, then the prediction is subtracted forming a residual block. Each block size has a number of possible prediction modes: four modes (vertical, horizontal, DC and Plane) for a 16x16 block size; nine modes for 4x4 (Figure 2.3).

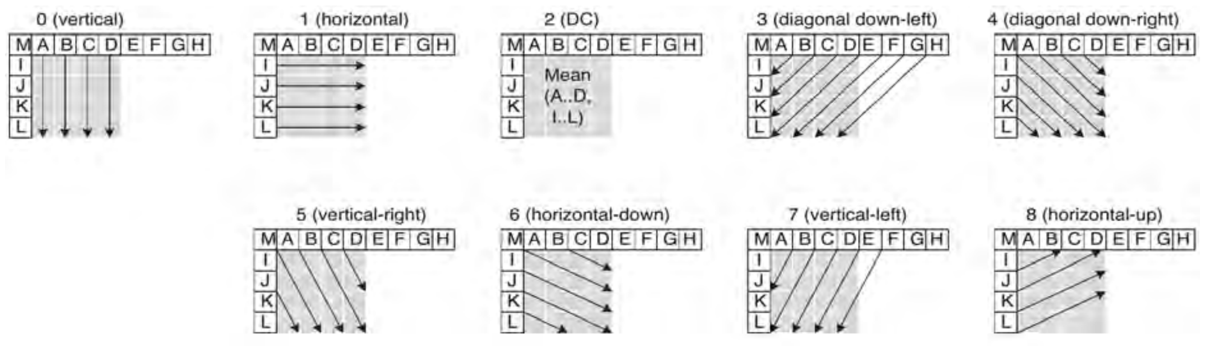


Figure 2.3: 4x4 prediction modes [2]

The choice of intra prediction block size is a trade-off between prediction efficiency and cost of signalling the prediction mode: a small 4x4 block gives a more accurate prediction (good match to the actual data in the block), meaning a smaller residual thus, fewer bits are necessary to represent that particular block. However, for each 4x4 block the encoder has to signal its presence to the decoder meaning that more bits are required to code this choice; larger blocks gives a less accurate prediction, but fewer bits are required to code the choice.

Inter prediction uses a range of block sizes (16x16 to 4x4) to predict pixels in the current macroblock using data from similar regions in previously coded frames. These frames may occur before or after the current frame in display order. The offset between the position of the current partition and the prediction region in the chosen reference picture is a motion vector that is differentially coded from other motion vectors in neighbouring blocks.

The resulting residual samples are coded using a 4x4 or 8x8 **integer transform**, that is an approximation to the Discrete Cosine Transform (DCT). The transform output is a block of transform coefficients that is then quantized (divide each value by an integer, keeping the integer part of the result). The Quantitization Parameter (QP) defines the precision of the transform coefficients, in a way that a high QP value results in high compression but produces poor decoded video quality.

Network Abstraction Layer

The Network Abstraction Layer (NAL) was designed to provide ”network friendliness” enabling simple and effective customization of the use of the VCL for a broad variety of systems [1] . It consists of a series of NAL Units (NALUs) that encapsulate the data provided by the VCL (VCL NALU) and other types (Non-VCL NALU) such as Sequence Parameter Sets (SPS) and Picture Parameter Sets (PPS) that contain control information. Each NALU consist of a 1-byte NALU header followed by a raw byte sequence payload (RBSP) containing control information or coded video data. The NALU header (Figure 2.4) is comprised of the following fields:

- **forbidden_bit** (1 bit): must be 0. If 1 the H.264 specification declares it as a syntax violation;
- **nal_ref_idc** (2 bits): 00 indicates that the content of the NALU is not used as reference picture for inter prediction, meaning that the NALU can be discarded without risking the integrity of the decoded video. Values above 00 indicate that the use of the NALU is required to maintain integrity;
- **nal_unit_type** (5 bits): this field specifies the NALU payload type according to Table A.1

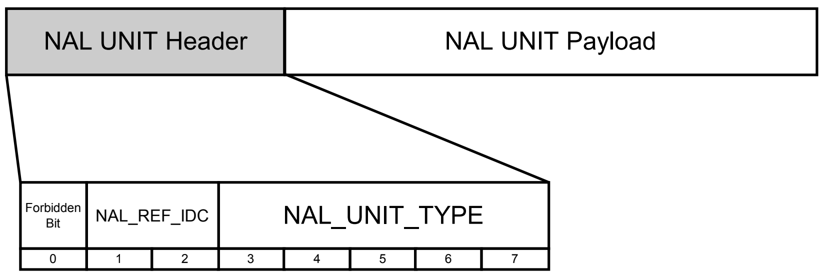


Figure 2.4: NALU header structure

An **Access Unit (AU)** is a set of NALUs that once decoded result in a decoded picture made up of one or more slices. Each slice consists of a Slice Header and Slice

Data, that in turn is a series of coded macroblocks containing the information about its type (Intra, Inter), prediction information, QP and residual data.

The NALUs can be mapped by several transport layer schemes or encapsulated in a media file. This subject will be addressed in detail in Chapter 3.

2.1.2 HEVC

HEVC [4] coding standard was released on January 2013 by the ITU-T VCEG and the ISO/IEC MPEG, together forming the Joint Collaborative Team on Video Coding (JCT-VC). The main goal of this standard is to improve compression performance relative to other existing standards, promising a bitrate reduction by half for equal perceptual video quality, at the expense of increased computational complexity. HEVC is targeted to handle two new Ultra Hight Definition (UHD) standards: UHDTV1 "4K" (3840 x 2160 pixels) and UHDTV2 "8K" (7680 x 4320 pixels) with frame rates ranging from 23,976 to 120 Hz as a progressive scan, and use parallel processing architectures. This means that in the future service providers can deliver contents with double perceptual quality using the same bandwidth of today's HDTV, or add services on the saved bandwidth due to HEVC use.

An HEVC compliant bitstream is produced by splitting each picture into blocks, this partitioning is reported to the decoder. The first picture of a sequence is coded using intrapicture prediction, the block of the remainnig pictures are typically coded using interpicture temporally predictive modes. The resulting residual signal from those two prediction schemes is transformed by a linear spatial transform, generating coefficients that are then quantized, entropy coded, and transmitted together with the prediction information.

So far this conceptual description is very similar to the one made about H.264, but although both standards employ the same hybrid approach (intra/ interpicture prediction and transform coding), HEVC has some interesting features that make it a standard for the future.

One of these features is its Quadtree Coding Structure (Figure 2.5). A quadtree is a tree data structure in which each node has exactly four children and are used to recursively partition a two-dimensional space by subdividing it into quadrants. So macroblocks in H.264 were replaced by Coding Tree Units (CTU) that in turn consists of a Coding Tree Block (CTB) containing luma, croma and syntax elements. The size of a CTB is chosen by the encoder and can be 64x64, 32x32 or 16x16 samples, generally larger sizes enable better compression. CTBs support partitioning into smaller blocks following the quadtree structure, generating Coding Units (CUs) and Coding Blocks (CBs). This means that a CTB can have one CU or be split into several CUs. A CU is in turn the root for the Prediction Units (PUs) and Transform Units (TUs). A Prediction Block (PB) can have the size of a CU or be split furthermore being the supported sizes 64x64, 32x32, 16x16, 8x8 and 4x4.

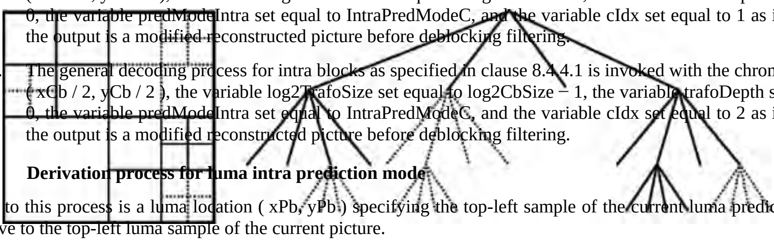
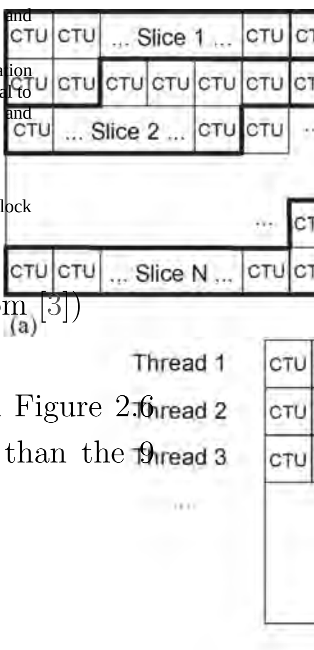


Fig. 4. Subdivision of a CTB into CBs [and transform block (TBs)]. Solid lines indicate CB boundaries and dotted lines indicate TB boundaries. (a) CTB with its partitioning. (b) Corresponding quadtree.



CTU CTU ... Slice 1 ... CTU
CTU CTU CTU CTU CTU CTU
CTU ... Slice 2 ... CTU CTU
CTU CTU ... Slice N ... CTU
(a)
Thread 1
Thread 2
Thread 3
CTU

Fig. 5. Subdivision of a picture
of wavefront parallel processing.

Figure 2.5: CTB partitioning and corresponding quadtree (adapted from [3])

further splitting is possible, as signaled by a maximum depth of the residual quadtree indicated in the SPS, each quadrant is assigned a flag that indicates whether it is split into four quadrants. The leaf node blocks resulting from the residual quadtree are the transform blocks that are further processed by transform coding. The encoder indicates the maximum and minimum luma TB sizes that it will use. Splitting is implicit when the CB size is larger than the maximum TB size. Not splitting is implicit when splitting would result in a luma TB size smaller than the indicated minimum. The chroma TB size is half the luma TB size in each dimension, except when the luma TB size is 4×4, in which case a single 4×4 chroma TB is used for the region covered by four 4×4 luma TBs. In the case of intrapicture-predicted CUs, the decoded samples of the nearest-neighboring TBs (within or outside the CB) are used as reference data for intrapicture prediction.

In contrast to previous standards, the HEVC design allows a TB to span across multiple PBs for interpicture-predicted CUs to maximize the potential coding efficiency benefits of the quadtree-structured TB partitioning.

F. Slices and Tiles

Slices are a sequence of CTUs that are processed in the order of a raster scan. A picture may be split into one or several slices as shown in Fig. 5(a) so that a picture is a collection of one or more slices. Slices are self-contained in the sense that, given the availability of the active sequence and picture parameter sets, their syntax elements can be parsed from the bitstream and the values of the samples in the area of the picture that the slice represents can be correctly decoded (except with regard to the effects of in-loop filtering near the edges of the slice) without the use of any data from other slices in the same picture. This means that prediction within the picture (e.g., intrapicture spatial signal prediction or prediction of motion vectors) is not performed across slice boundaries. Some information from other slices may, however, be needed to apply the in-loop filtering across slice boundaries. Each slice can be coded using different coding types as follows.

1) *I slice:* A slice in which all CUs of the slice are coded using only intrapicture prediction.
2) *P slice:* In addition to the coding types of an I slice, some CUs of a P slice can also be coded using interpicture prediction with at most one motion-compensated

using interpicture pr
compensated predict
tion). B slices use bo

The main purpose of sl
losses. Furthermore, slices
mum number of bits, e.g.,
fore, slices may often co
CTUs per slice in a mann
video scene. In addition t
which are self-contained a
gular regions of the pictur
enable the use of parallel p
and decoding. Multiple tile
being contained in the sam
may contain multiple slice
arranged group of CTUs (
all of them containing abo
shown in Fig. 5(b).

To assist with the gran
pendent slices are addition
slice is divided into rows
row can be begun as soo
for prediction and adaptati
made in the preceding row
of rows of CTUs by us
the encoder or decoder (c
Fig. 5(c). For design sim
used in combination with t
in principle, work properly

G. Intrapicture Prediction

Intrapicture prediction o

2. The general decoding process for intra blocks as specified in clause 8.4.4.1 is invoked with the chroma location (xCb / 2, yCb / 2), the variable log2TrafoSize set equal to log2CbSize − 1, the variable trafoDepth set equal to 0, the variable predModeIntra set equal to IntraPredModeC, and the variable cIdx set equal to 1 as inputs, and the output is a modified reconstructed picture before deblocking filtering.
3. The general decoding process for intra blocks as specified in clause 8.4.4.1 is invoked with the chroma location (xCb / 2, yCb / 2), the variable log2TrafoSize set equal to log2CbSize − 1, the variable trafoDepth set equal to 0, the variable predModeIntra set equal to IntraPredModeC, and the variable cIdx set equal to 2 as inputs, and the output is a modified reconstructed picture before deblocking filtering.

8.4.2 Derivation process for luma intra prediction mode

Input to this process is a luma location (xPb, yPb) specifying the top-left sample of the current luma prediction block relative to the top-left luma sample of the current picture.

In this process, the luma intra prediction mode IntraPredModeY[xPb][yPb] is derived.

Table 8-1 specifies the value for the intra prediction mode and the associated names.

Table 8-1 – Specification of intra prediction mode and associated names

Intra prediction mode	Associated name
0	INTRA_PLANAR
1	INTRA_DC
2..34	INTRA_ANGULAR2..INTRA_ANGULAR34

IntraPredModeY[xPb][yPb] labelled 0..34 represents directions of predictions as illustrated in Figure 8-1.

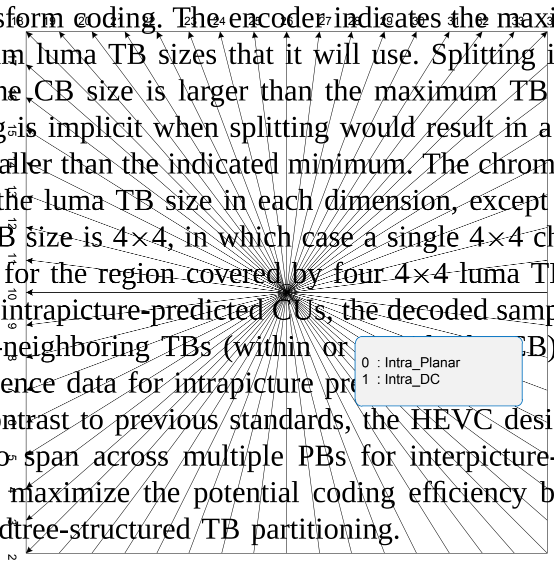


Figure 8-1 – Intra prediction mode directions (informative)

Figure 2.6: HEVC prediction modes [4]

IntraPredModeY[xPb][yPb] is derived by the following ordered steps:

1. The neighbouring locations (xNbA, yNbA) and (xNbB, yNbB) are set equal to (xPb − 1, yPb) and (xPb, yPb − 1), respectively.

Intra prediction specifies the 35 different prediction modes presented in Figure 2.6 (33 angular modes, a DC mode and an interpolation mode), much more than the 9 supported by H.264.

For inter prediction non-square PB modes are allowed although a PB inter frame cannot have the 4x4 size.

To generate a prediction for the current block the encoder must determine the best PB size to use by testing all prediction modes at all PB sizes available and then choose the best combination, based on Rate distortion Optimization (RDO). For example, to process intra prediction mode decision, the encoder needs to measure the values of distortion for each one of the 35 prediction modes and for every level of the PU subtree. Checking all these possibilities is a very demanding computational task and proves to be impractical for real-time encoding, so there are several schemes to enable smart mode decision without running the totality of the encoding process described in [17] for fast Intra Prediction mode decision, and [18] for Coding Tree Depth complexity reduction.

This standard is receiving great attention from research groups around the world, and although it is outside of the scope of this work, in the near future a similar approach could be taken, to develop a system similar to the one being presented, to monitor HEVC video quality

2.2 3D Video Coding

3D video is commonly known as a type of visual media that gives the viewer the perception of depth. To accomplish this, a whole system, from acquisition, to storage and display has to be in place while keeping interoperability and compatibility with existing 2D video infrastructures.

For professional 3D video capture, typically setups with 2 or 3 cameras are used. The problem associated with multi-camera systems are temporal synchronization, geometrical calibration [19] and colour balance between all cameras. There are also camera setups equipped with depth sensors capable of depth enhanced 3D video. This means that 3D video formats can be divided into two main classes [20]: video-only formats and depth-enhanced formats. Video-only formats can be further divided into Classical Stereo Video (CSV) with two views, Mixed Resolution Stero (MRS) video with one of the views spatially sub-sampled and Multi-View Video (MVV) with more than two views. In turn, depth-enhanced formats can also be sub divided into Video plus Depth (V+D), Multi-View Video plus Depth (MVD) and Layered Depth Video (LDV). The advantage to the inclusion of depth or disparity information into these video formats is that 3D video reproduction can be adapted to any display, since all views are synthesized at the decoding stage.

For the scope of this work only MVV coding schemes will be addressed. Apart from support for 3D video MVV also enables free-viewpoint video, were the video direction can be interactively changed and presented on a 2D display. All these features represent great amount of data that needs to be efficiently compressed, while keeping backwards compatibility.

2.2.1 H264/AVC Multi-View video Coding

The straightforward approach to encode MVV is to encode each one of the views separately treating them as independent videos ensuring backwards compatibility. Although possible, such approach proves to be very inefficient when comparing it to H264/AVC extension for Multi-View video Coding (MVC). Annex H of [16] describes all key features and syntax elements of this standard.

Inter-view prediction is the concept introduced in MVC, that is responsible for exploiting spatial and temporal redundancy for compression purposes. Since cameras on a multiview scenario capture the same scene from nearby viewpoints there is great inter-view redundancy (Figure 2.7), and temporal redundancy is also present [5].

The encoder is applied to the view sequences simultaneously enabling inter-view predictive coding, resulting in dependant bitstreams to which information about camera parameters may be included.

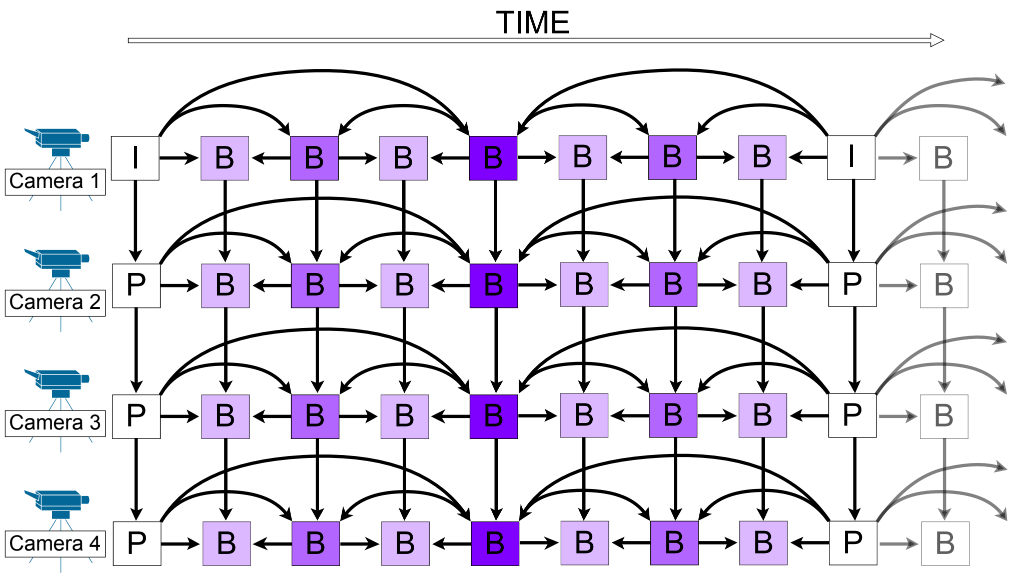


Figure 2.7: Picture prediction relation in MVC (adapted from [5])

To make backwards compatibility possible the MVC design states that the compressed multiview stream must include a baseview bitstream, coded independently from the other views. The remaining specifies a NALU header extension for NALUs of type 14 and 20 (Table A.1, Annex A). Since these NALU types were reserved for this purpose, decoders conforming to one or more of the profiles specified before MVC, thus not supporting it, will be capable of reproducing 2D video corresponding to the base view. Any unknown information is discarded, keeping backwards compatibility.

NAL units created using the MVC profile have additional fields to the ones in Section 2.1.1 described in [21]. They are illustrated in Figure 2.8 and described bellow:

- (R) **reserved_zero_bit** (1 bit): reserved for future extension. Must be 0 and should be ignored at reception;
- (I) **idr_flag** (1 bit): specifies whether the view component is a view instantaneous decoding refresh picture;
- (PRID) **priority_id** (6 bits): a lower value indicates higher priority in relation to other NALs;
- (VID) **view_id** (10 bits): this component corresponds to the identifier of the view the NALU belongs to;

- (TID)**temporal_id** (3 bits): temporal layer identification. A video bitstream can have several temporal layers where all NALUs of one layer form a valid bitstream with a given frame rate. The higher the temporal layer is, higher the frame rate;
- (A) **anchor_pic_flag** (1 bit): this field signals whether the view component is an anchor picture;
- (V) **inter_view_flag** (1 bit): specifies if the view component is used for inter-view prediction;
- (O)**reserved_one_bit** (1 bit): reserved for future extension. Must be 1 and should be ignored at reception.

The contents and structure of the NALU fields is of great importance for the project, since they will be used for network bitstream parsing purposes and data retrieval. NALUs contain actual video data so their header information is used to categorize the type of content they are carrying, and for this case, the information to retain is **nal_ref_idc** and **nal_unit_type** (defined in Section 2.1.1). The **nal_ref_idc** field specifies the importance of the content, and **nal_unit_type** specifies if the payload is part of the VCL (Type 1,5 and 20) and if the NALU contain an Instant Decode Refresh IDR picture (Type 5), meaning that this NALU contains a payload that contains reference information and will force decoder refresh. These combination of the content of these two fields will help determine if the content of a network packet is from an I, B or P slice. From MVC specific fields the ones used are the **idr_flag** and **view_id**, but for confirmation purposes only etwork
is fro

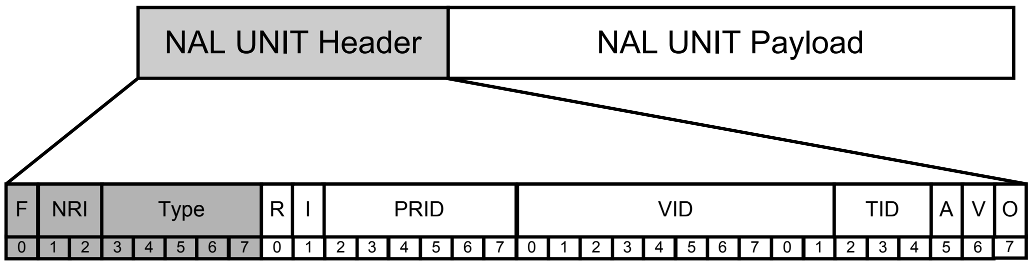


Figure 2.8: MVC NALU header structure

Chapter 3

Video Transport

This chapter describes standardized methods to broadcast/transmit video over IP networks and their characteristics. Also video encapsulation methods are explained, giving more emphasis to the encapsulation structure used in this project.

3.1 Video Packetization

3.1.1 Internet Protocol

The Internet Protocol [22] is datagram transport protocol used to relay data across networks, and is the base of the internet. IP packets consist of a header containing addressing and control information for packet routing through the network, and a payload, that consists of the data to be transmitted. The payload can carry several protocols for data streaming such as RTP and UDP.

3.1.2 User Datagram Protocol

The User Datagram Protocol (UDP) protocol [23] enables computer applications to send messages to other hosts on an IP network without prior communications to set up a transmission channel. It provides checksums for data integrity and port numbers for addressing different functions at the source and destination of the datagram. UDP is used in situations where error checking and correction is not a necessity or is done by the application, avoiding the overhead at the network interface level. Time-sensitive data transmission such as video use this protocol because dropping packets is preferable over waiting for delayed ones.

3.1.3 Real-time Transport Protocol

The Real-time Transport Protocol (RTP) protocol [24] provides end-to-end delivery services for data with real-time characteristics (video and audio). It provides payload type identification, sequence numbering, time-stamping and deliver monitoring. Normally it runs on top of UDP making use of its characteristics, and can be used with other suitable underlying network or transport protocols.

Since RTP doesn't provide any mechanism to guarantee quality-of-service, timely or out-of-order data delivery, but the included sequence numbers allow the receiver to reconstruct the packet sequence and detect packet losses.

H.264 video carriage over RTP is defined in [25], were NALUs are mapped into the payload accordingly.

3.1.4 MPEG 2 Transport Stream

This packetization method is described in more detail than the previous ones because of its importance for project development.

MPEG 2 Transport Stream (MPEG2-TS) [26, 27] was first specified in MPEG-2 Part 1. Due to its structure and flexibility is still in use and provides a way to transmit several types of data (audio and subtitles in several different languages, 3D video, ...) in the same stream while being compatible with all devices that may not support all its features.

Before transmitting the different sources of data must be packetized and MPEG2-TS does this by multiplexing data. Packet multiplexing is simply interleaving packets of data from several elementary streams one after another to form a single MPEG2-TS stream.

Synchronization is very important in multimedia and MPEG2-TS has several means to maintain it between the elementary streams that are being decoded. This is achieved by using **Time Stamps** and **Clock References**, that are added to the MPEG2-TS stream at the time of multiplexation. There are two types of Time Stamps (Decoded Time Stamp and Presentation Time Stamp) and they refer to a particular moment when a video picture or sound frame should be decoded and presented on the output device. They are represented by a 33 bit data field containing a time reference indicated by the **System Time Clock**. The Clock References (42 bit data field) are used by the decoder to update their own clock reference.

Packetized Elementary Streams

Elementary Streams (ES) are compressed data from a single source and all the attached auxiliary data needed for synchronization and identification of the source (NALUs in H.264). These sources are packetized into Packetized Elementary Streams (PES) that can be of fixed or variable length and it is comprised of a header and the stream data (payload).

The PES packet structure can be seen in Figure 3.1. Its fields are described bellow:

- **start_code_prefix** : 24 bits all 0 except for the last one;
- **stream_id** : 8 bits ranging from 0xBD to 0xFE and defines type of stream or an ID for different streams of same type;
- **PES_packet_lenght** : 16 bits stating the number of bytes that follow. For video packets, this field can have a value of 0 indicating an unspecified length;
- **padding_bytes** : Bytes equal to 0xFF that are discarded at the decoder;
- **marker_bits** : 2 bits with fixed valued bits usually all 1s ;
- **PES_scrambling_control** : 2 bits indicating scrambling mode;
- **PES_priority** : 1 bit indicating that a PES packet has higher priority;
- **data_alignment_indicator** : 1 bit indicating if the payload starts with a video Start-Code or audio sync-word ;
- **copyright** : 1 bit indicating if the PES has copyrights;
- **original_or_copy** : 1 bit indicating if stream is original;
- **PTS_DTS_flags** : 2 bits. 0 (decimal) indicates that there isn't any PTS or DTS. 3 indicates presence of both and 2 means that only PTS is present;
- **ESCR_flag** : 1 bit to indicate presence or a clock reference in PES header ;
- **ES_rate_flag** : 1 bit to indicate if bitrate value is present in PES header;
- **DMS_trick_mode_flag** : 1 bit indicates if there is an 8-bit trick mode field;
- **additional_copy_into_flag** : 1 bit indicates additional copyright information ;
- **PES_CRC_flag** : 1 bit to indicate presence of a CRC is present;
- **PES_extention_flag** : 1 bit indicating if PES header extension fields are present;
- **PES_header_data_lenght** : 8 bits containing the length of the optional PES header extension fields. ;
- **PTS** : Presentation Time Stamp defines when the output device should present the decoded information;

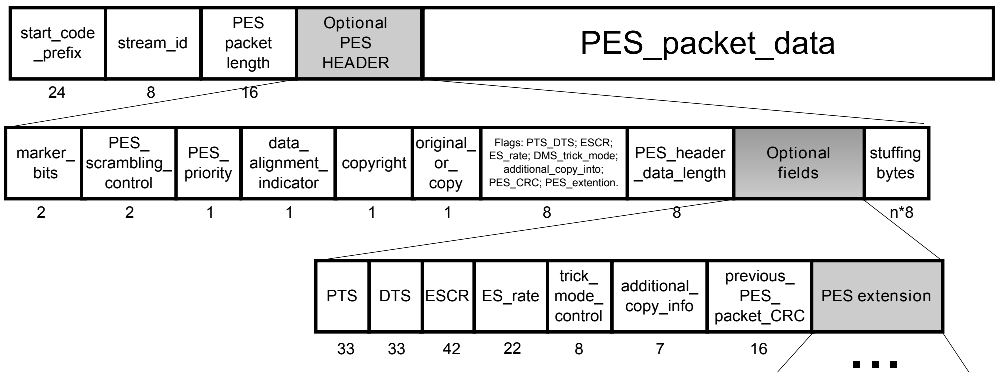


Figure 3.1: PES packet structure

- **DTS** : Decoding Time Stamp defines when a stream should be decoded (may have a value of 0);
- **ESCR** : Elementary Stream Clock Reference;
- **ES_rate** : Bitrate of a PES Stream;
- **trick_mode_control** : Indicates the trick mode applied to the video (fast forward, pause, rewind, ...) ;
- **additional_copy_info** : contains copyright information;
- **previous_PES_packet_CRC** : 16 bits containing information about previous PES packet (excluding header fields);
- **extension_data** : can include buffer sizes, sequence counters, bitrates...;
- **stuffing_data** : no more than 32 stuffing bytes are allowed in one PES header;
- **PES_packet_data** : Bytes of data from the elementary stream;

All fields from this packetization technique had to be described due to their content or structural importance in data parsing procedures. The **start_code_prefix** helps to identify if a PES packet header is present, followed by the **stream_id** that identifies type of data the PES packet is carrying (audio, subtitles, video ...) and for the case of video content the stream_id is equal to 0xe0. **PES_packet_lenght** is one of the essential fields to be retrieved for analysis, and if is 0, action needs to be taken to determine this parameter in an other way. PTS and DTS are also used to determine the elapsed video time passing through the network. Please note that two consecutive packets containing a PTS must not be more than 700 ms apart. **PES_header_data_lenght** is used to determine where the payload (NALU) data starts.

Program Streams

Program Streams (PS) are comprised of packs of multiplexed data. These packs contain a header followed by a variable number of PES packets from several elementary stream sources each one with an unique stream_id. Additionally to PES packets, PS Packets also contain descriptive information called PS Program Specific Information (PS-PSI), that defines the program and its parts storing, for example, where I-pictures are stored enabling random access to several points of the stream. Program Streams are indicated for media storage or transmission over networks where reliability possible video degradation is not a problem.

Transport Streams

This stream type is suitable for transmission networks that suffer from occasional transmission errors. It is also comprised of multiplexed data from PES packets and some more descriptive data.

Generally variable length PES packets are further packetized into fixed 188 bytes long Transport Stream (TS) packets. This division into constant length packets makes error detection and recover easier.

A TS packet contains a TS Header, optionally secondary data called Adaptation Field and some or all data from a PES packet. The Header contains synchronization information, a Packet Identifier (PID) and information on timing and error detection, all this described in detail below:

- **sync_byte**: 8 bits. Fixed value of 0x47;
- **transport_error_indicator**: 1 bit indicating the presence of an uncorrectable bit error in the current TS packet ;
- **payload_unit_start_indicator** 1 bit signalling the presence of a new PES packet or a new TS-PSI Section ;
- **transport_priority**: 1 bit stating higher priority over other packets ;
- **PID**: 13 bits for Packet Identification;
- **transport_scrambling_control**: 2 bits to indicate presense of the scrambling mode of the packet payload;
- **adaptation_field_control**: 2 bits to signal the presence of an adaptation field or payload ;
- **continuity_counter**: 4 bits. One for each PID incrementing every TS packet ;
- **pointer_field**: 8 bits indicates the number of bytes until a new TS-PSI Section ;

Additionally to the above fields exist some more optional ones which presence is signalled by the **adaptation_field_control**. They are called Adaptation Field contains System Time Clock timing information called *Program Clock Reference* (PCR). This information is important for decoder synchronization and is transmitted with a periodicity of at least 100 ms. The fields are described bellow:

- **adaptation_field_lenght**: 8 bits indicating the number of bytes following;
- **discontinuity_indicator**: 1 bit signals discontinuity in clock reference or continuity counter ;
- **random_access_indicator**: 1 bit if the next PES packet starts a video or audio sequence;
- **elementary_stream_priority_indicator**: 1 bit stating higher priority;
- **PCR_flag**: 1 bit. PCR is present;
- **OPCR_flag**: 1 bit. Original PCR is present;
- **splicing_point_flag**: 1 bit indicating that a splice countdown field is present;
- **transport_private_data_flag**: 1 bit. Adaptation field has private data;
- **adaptation_field_extension_flag**: 1 bit flag signaling presence of extension fields;
- **program_clock_reference (PCR)**: 33 bits, value based on a 90kHz reference clock + 6 padding bits + 9 bits extension based on a 27 MHz clock;
- **original_program_clock_reference (OPCR)**: same size as PCR, used to extract a single program from a multiprogram TS;
- **splice_countdown**: 8 bits with the number of remaining TS packets (same PID) until the end of an audio frame or video picture;
- **transport_private_data_legth**: 8 bits stating the number private_data bytes following;
- **private_data_bytes**: private data;
- **adaptation_field_extension_lenght**: 8 bits. Number of bytes of the extended adaptation field;
- **stuffing_bytes**: variable number of 8-bit values of 0xFF, to discarded by the encoder.

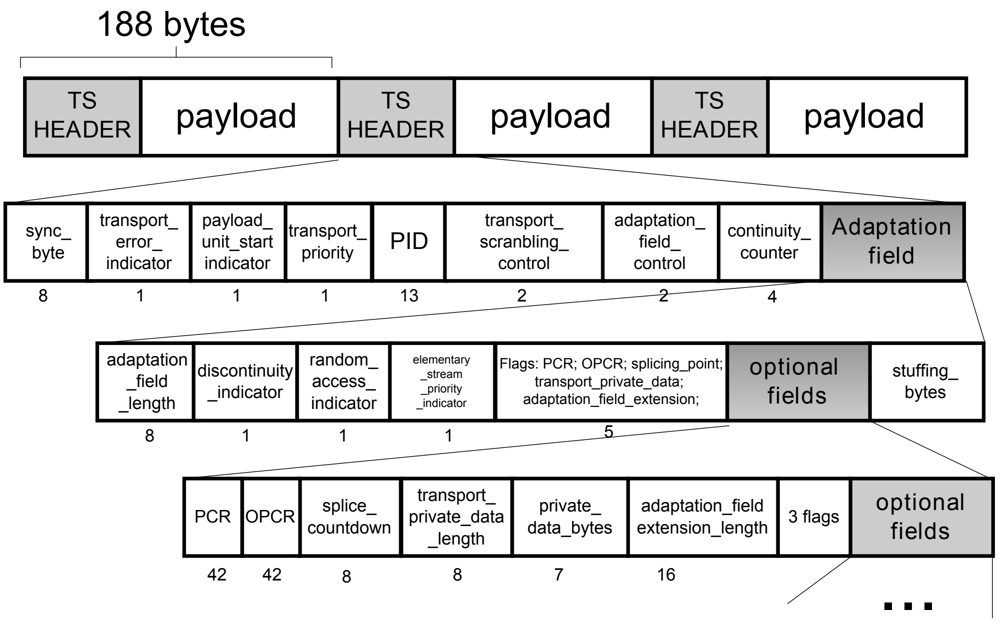

188 bytes
TS HEADER
payload
TS HEADER
payload
TS HEADER
payload
sync_ byte
transport_ error_ indicator
payload_ unit_start indicator
transport_ priority
PID
transport_ scranbling_ control
adaptation_ field_ control
continuity_ counter
Adaptation field
8
1
1
1
13
2
2
4
adaptation _field _length
discontinuity _indicator
random_ access_ indicator
elementary _stream _priority _indicator
Flags: PCR; OPCR; splicing_point; transport_private_data; adaptation_field_extension;
optional fields
stuffing_ bytes
8
1
1
1
5
PCR
OPCR
splice_ countdown
transport_ private_data _length
private_ data_bytes
adaptation_field extension_length
3 flags
optional fields
42
42
8
8
7
16
...

Figure 3.2: TS packet structure

From these fields **sync_byte** helps confirm that a TS packet is present, **payload_unit_start_indi** signals that a PES packet header is present, that is usually also followed by a NALU header. **PID** identifies a packet that contains a certain type of stream, for 3D each one of the views has a different PID. The **continuity_counter** is used to identify packet loss events and determine the number of packet losses per view. **Adaptation_field_lenght** and **adaptation_field_extension_lenght** are used to locate where in the bitstream is the start of the payload without doing a byte-by-byte sweep.

There are three special TS streams (TS - Program Specific Information)that contain additional information such as program descriptions and assignments of PES and PIDs. The first type of stream is the *Program Association Table* (TS-PAT), and always has a PID equal to 0, and is the first stream to be transmitted. For each program the TS-PAT defines the association between the program number and a PID. Another special stream is the *Network Information Table*, that carries data describing and characterizing the network carrying the TS. Since this data is dependent of network implementation it is considered Private data. The third type of stream are *Program MAP Tables* that carry information about each one of the programs in the transmission stream such as the assignments of the unique PID values for each of the PES streams. Although these types of TS streams could be useful to this work they were not used or implemented, since MVC streams can be identified without them, so implementing identification and parsing of these TS streams would not add any value to the work, hence the lack of description.

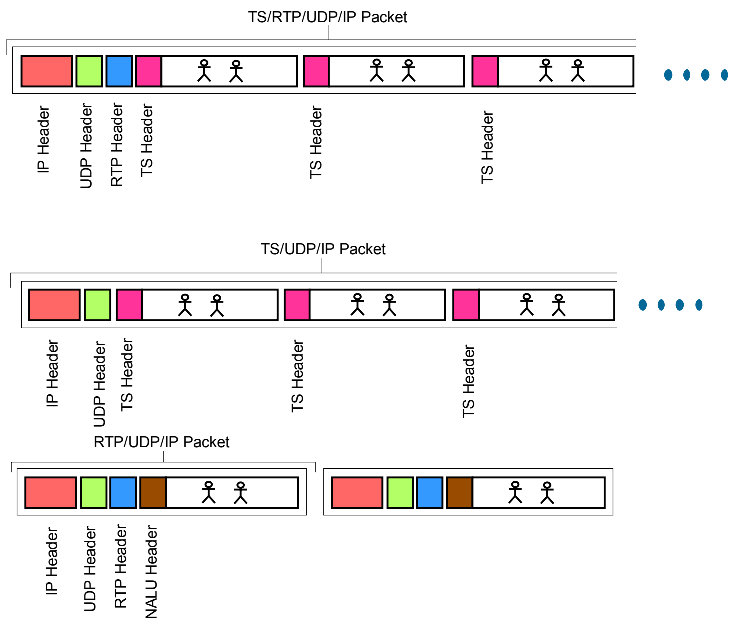


Figure 3.3: Comparison between transport methods

3.1.5 MPEG2-TS transport methods

One of the two methods used to transport MPEG2-TS over IP networks is to carry these packets over RTP (specified in [28] and [29]).

The other method, and the one being used in the scope of this work, selects a number of TS packets and adds them to the payload of the UDP datagram. For Ethernet networks the Maximum Transmission Unit (MTU) has 1500 bytes, so 7 TS packets may be transmitted on the same UDP packet ($1500/188 \approx 7$). Comparison of the structure of the several transport methods can be seen in Figure 3.3.

3.2 3D MPEG2-TS encapsulation

Prior to transmission MVC data has to be encapsulated into MPEG2-TS (Figure 3.4). This is done by splitting the MVC encoded bitstream into ES from the same view, and then encapsulate each one into PES packets. The resulting PES packets are in turn split into pieces with TS packet payload size and encapsulated. Since the size of a PES packet is variable, the number of TS packets needed to contain it also varies, and if the remainder of a PES packet is not enough to fill one TS, the rest of the available space is filled with 0xFF. This happens because a PES must start immediately after the TS header, thus not allowing remainder free TS payload space to be filled with a new PES packet header. For each one of the views present (two in 3D) TS packets have a different

PID to distinguish all the views, treating them as separate video streams.

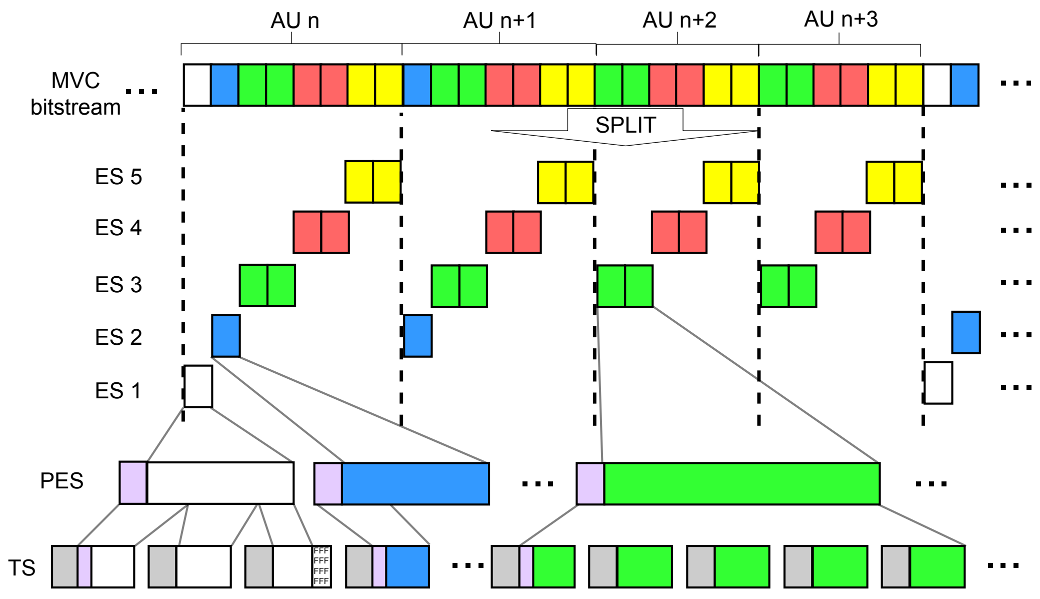


Figure 3.4: Illustration of MVC bitstream encapsulation into TS

To identify a 3D video stream at network level one must know the meaning of each field of the bistream. Since a TS packet can carry very different data types (several video views, audio channels and subtitles of different languages) by multiplexing their sources into one single stream, data parse can be a very complex task since it involves several variables from several packetization hierarchies.

Chapter 4

3D Video Qualiy Models

The importance of video quality measurement over IP networks is increasing due to the exponential growth of available streaming services to the consumer. Understanding the impact that packet loss events have in perceived video quality is therefore an important matter. In [30] a brief analysis of the H.264 MVC performance over error prone IP networks using RTP packetization is conducted, and in [31] the effects on video quality of corrupted H.264 bitstreams is evaluated. The main methods to evaluate video quality are subjective and objective techniques, being the former based on human observers that evaluate the video quality of samples that are presented to them. Results from the subjective methods are then used to compute Mean Opinion Score (MOS) and other statistics. However, these results are subject to great variations due to environmental conditions (lighting, viewing distance, shown video samples) and observer conditions [32, 33]. Also, they are expensive and take a fair amount of time to produce results; An objective video quality evaluation method enables automatic estimation of user perceived video quality without the need of any human intervention thus making it a faster, reliable and cheaper solution.

In objective techniques a set of quality related parameters of a video are gathered and processed to generate an objective quality metric capable of predicting the MOS. Depending on the information available from the original video, the objective methods can be divided into three categories [34]:

- **Full-Reference** (FR): the original video is available to be used for comparison with the distorted one. This proves to be impractical in cases were video quality needs to be monitored in remote locations, were the reference is not available;
- **Reduced-Reference** (RR): only some features of the original video are needed to determine the video quality. Although the needed data is much less than in the FR methods, it has the same problem and is not an option when trying to measure the quality at any point of the transmission channel;
- **No-Reference** (NR): these methods do not use any reference to compute video quality. This represents a great advantage over the other two categories since

it allows video quality assessment at any point of a transmission network. This category can be further divided into two [35]:

– **No-reference pixel-based**: the video quality is estimated by using the decoded video;

– **No-reference bit stream-only**: the video quality estimation is done without completely decoding the video to obtain decoded pixels. The needed information is extracted from packet headers at the network-layer, as in the scope of this work.

In [34] the most used quality metrics are detailed: Mean Square Error (MSE), PSNR and Structural Similarity (SSIM) [36] whose mathematical expressions are Equations 4.1 , 4.2 and 4.3 respectively.

$$MSE = \frac{1}{N}\sum_{N}^{n=1}(X_n - Y_n)^2 \tag{4.1}$$

$$PSNR = 10\log_{10}\frac{L^2}{MSE} \tag{4.2}$$

$$SSIM(x,y) = \frac{(2u_x u_y + c_1)(2\sigma_{xy} + c_2)}{(u_x^2 + u_y^2 + c_1)(\sigma_x^2+\sigma_y^2 + c_2)} \tag{4.3}$$

Where for the MSE X_n and Y_n are the pixel values of the original and distorted frames, and N is the total number of pixels in a frame. For the PSNR L is the maximum pixel value in a frame. Although these metrics are widely used, they have no correlation with the human visual system meaning that the results produced by them and video quality perception from a human are far from each other, thus they cannot be used in 3D video quality assessments.

The SSIM uses structural information from the video, approximating a video quality assessment metric to the perception of the human visual system. x and y are the original and the distorted frame information, u_x and u_y are the mean of x and y. σ_x , σ_y and σ_{xy} are the variance of x, y and the covariance between x and y. Finally c_1 and c_2 are constants to avoid the denominator becoming very close to 0.

4.1 NR Quality Models

There are several quality models reported in the scientific literature for image or video perceived quality prediction.

In [37] the quality of JPEG stereoscopic coded images is assessed by analysing image segments for disparities between views, comparing local features such as edge, flat and texture areas. Also [38] proposes a NR model that uses decoded 3D video to evaluate the video quality, by evaluating spatial and temporal inconsistencies.

A FR video quality metric was used in [39] in order to formulate a NR video quality model for 2D video, using packet loss rate and the distance between two consecutive Instant Decode Refresh (IDR) frames as inputs for video quality assessment. A similar approach was taken in [40] for 3D videos generated by Depth-Image-Based Rendering (DIBR), using packet layer parameters from impaired video sequences as input parameters to train a neuronal network with several layers, in order to determine the effect that packet losses have on this type of video, thus defining a NR video quality assessment model.

4.1.1 Proposed NR model

The NR model implemented in this work is explained in [6] and is the result of previous research and development done in the 3DVQM project at Instituto de Telecomunicações. The need to compare the development results lead to choose this model, since complementary information, in addition to scientific papers and reported results, is available for testing the validity of the network monitoring system.

To develop this NR model several 3D video datasets with GOP size equal to 21 frames, IBPBP GOP structure and QP ranging from 26 to 32, were encoded using H.264/AVC Stereo High Profile (Figure 4.1 - 1), then encapsulated into MPEG2-TS (Figure 4.1 - 2). After this, network loss conditions were simulated by subjecting the encapsulated files to the loss of packets (Figure 4.1 - 3) in a controlled manner, in order to model different types of impairments. These files were then decoded (Figure 4.1 - 4) and the results compared with the original coding sequence to compute the SSIM drop (Figure 4.1 - 5) ***Delta_SSIM*** in equation 4.4, that corresponds to the difference between the SSIM of the error-free stereo frame, EF_SSIM_x (SSIM=1) and that of the resulting frame after a loss event D_SSIM_x.

$$Delta\_SSIM_x = EF\_SSIM_x - D\_SSIM_x \tag{4.4}$$

$$dSSIM_n = p_n L_{fs}^n + ... + p_2 L_{fs}^2 + p_1 L_{fs} + p_0 \tag{4.5}$$

This experience was repeated for several frame-size losses resulting in various point clouds as exemplified in Figure 4.2, for each of the slice types P or B. These point clouds were then fitted into polynomial equations of the format shown in Equation 4.5 were

GOP structure. Three different datasets were created by encoding this sequence with QP ranging from 26 to 32, achieving different PSNR and bitrates. These datasets are described in Table 3.1. The resulting video stream was encapsulated into a TS stream using the reference software FFMPEG (Figure 3.6 stage 2).

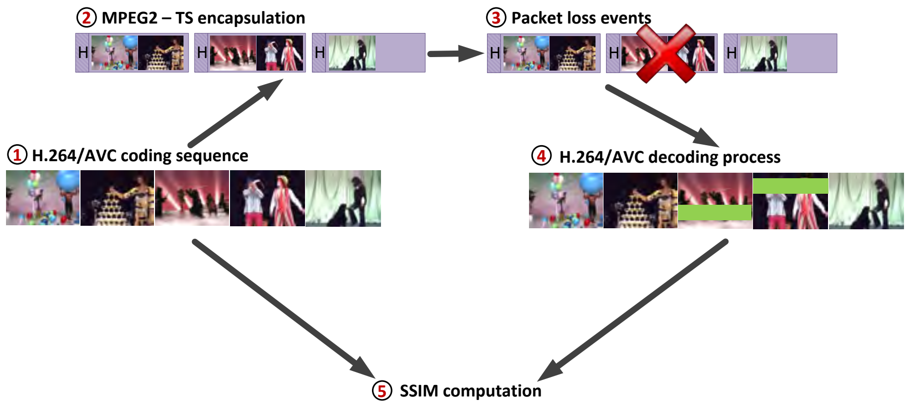

② MPEG2 – TS encapsulation
③ Packet loss events
H
① H.264/AVC coding sequence
④ H.264/AVC decoding process
⑤ SSIM computation

Figure 3.6: Experimental procedure.

Figure 4.1: SSIM computation [6]

To simulate network loss conditions, packetized streams were then subject to packet loss events (Figure 3.6 stage 3). These have to be created according to the error pattern produced in a controlled manner, to better modeling individual types of impairments. Aiming this, at present a single frame is lost in each loss event, but other impairment models can be used [53].

$dSSIM_n$ is the *Delta-SSIM*, n is the polynomial degree, L_{fs} is the lost frame size in bytes and p_n are the polynomial coefficients (Annex C).

4.1.2 Quality model overview

Table 3.1: QP's, PSNR and bitrate.

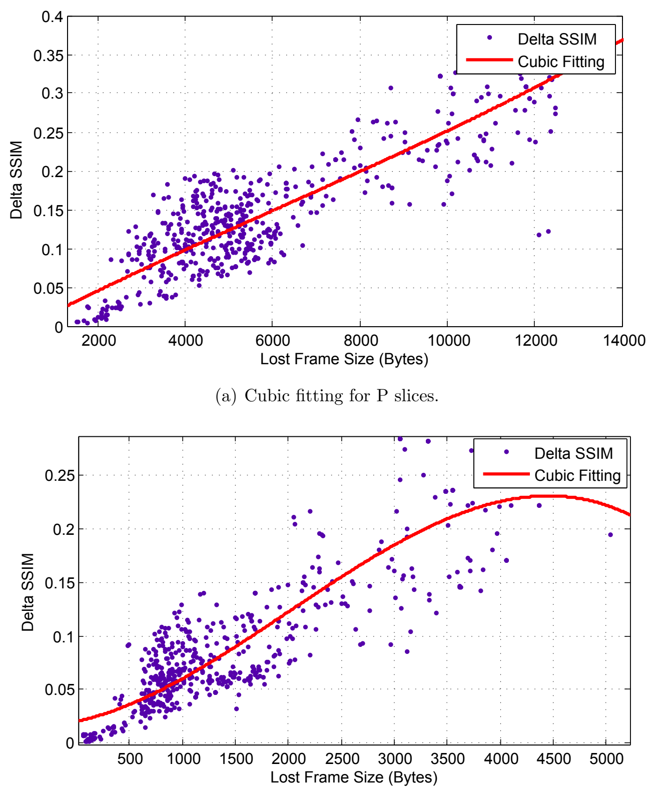
	I-QP	P-QP	B-QP	PSNR (dB)	Bitrate (Kb/s)
Dataset-1	26	28	28	42	2847
Dataset-2	28	30	30	41	2225
Dataset-3	30	32	32	40	1727

The objective of the proposed model is to estimate the quality degradation in isolated stereo frames due to errors in TS packet transmission. Degradation is measured by the difference between the SSIM of the error-free stereo frame (SSIM = 1) and that of the displayed frame, assuming that frame-copy is used as concealment method. This method does not take into account coding distortion, thus SSIM=1, for any correctly received frame, meaning that errors due to lossy encoding are not taken into account in this method.

Then, the corrupted TS stream was decoded using frame-copy for concealment of lost frames, as depicted in stage 4 of Figure 3.6. The decoded frames were used to

The used NR model uses an estimate of each lost frame size and type. The lost frame size is estimated from the average of the last frame sizes from the same view and type. The GOP size and structure are an user input parameter, since this type of information is in general static. In summary, quality estimation for this particular model needs the following parameters from packet headers and user inputs as illustrated by Figure 4.3:

- **T1: Continuity Counter** - detection of TS packet losses;
- **P1: PES packet length** - provides frame size;
- **A1: NAL_unit_type** - frame type information;
- **U1: GOP size**;
- **U2: GOP structure**;



(a) Cubic fitting for P slices.

(b) Cubic fitting for B slices.

Figure 3.9: Delta SSIM vs frame size for Dataset-3.

Figure 4.2: Delta SSIM vs frame size loss for one particular test Dataset[6]

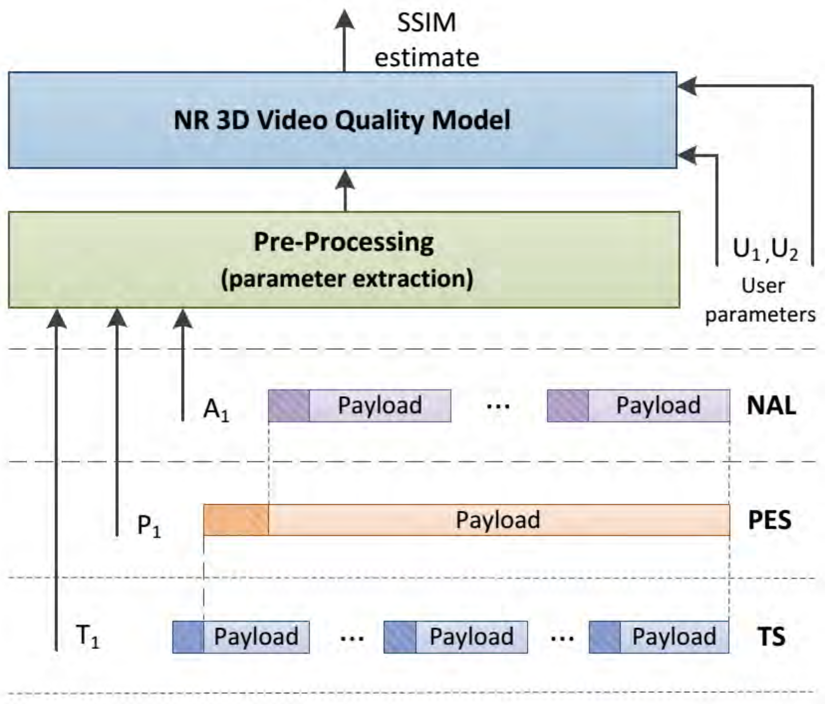


Figure 4.3: Packet layer model structure [6]

Chapter 5

Distributed Architecture

As mentioned in Section 1.1 the objective of this work is,in short, to build a system that is able to estimate the video quality of a service passing through an IP network. This chapter overviews the architecture of the implemented system and its components, focusing on a conceptual description, followed by a thorough description of every single component from a development point-of-view.

5.1 Proposed Architecture

In order to estimate where the video transmission is being degraded, a way to measure video quality in various network nodes between the source and receiver is needed. This is achieved by installing network traffic monitoring **Probes** on various network nodes. These Probes are capable of identifying, filter and interpret network packets carrying video information and to temporally retain relevant data of the last instants of video passing through the one particular network node where each one is installed. This data is then relayed to a central **Server** where data from all Probes is concentrated, processed and stored for future analysis.

The Server itself consists of a **Probe Report Handler**, a **Graphical User Interface** GUI and a **Database**:

- The Probe Report Handler is responsible for concentrating all incoming information from the Probes, treat it and store it into the database. It is also where any video quality metric can be easily implemented using the data arriving from the Probes;

- The GUI presents status feedback from the Probes and video quality metric results. It is also possible to configure Probe and video parameters.

- The Database stores all data gathered from the Probes as well as other relevant information concerning the video transmission (GOP size and structure, bitrate, ...) and the Probes (IP, identification, geographical location, ...);

Such implementation enables the identification of the network nodes where video quality is being degraded. Also, information stored in the database can be used for further analysis. Figure 5.1 illustrates the architecture described above.

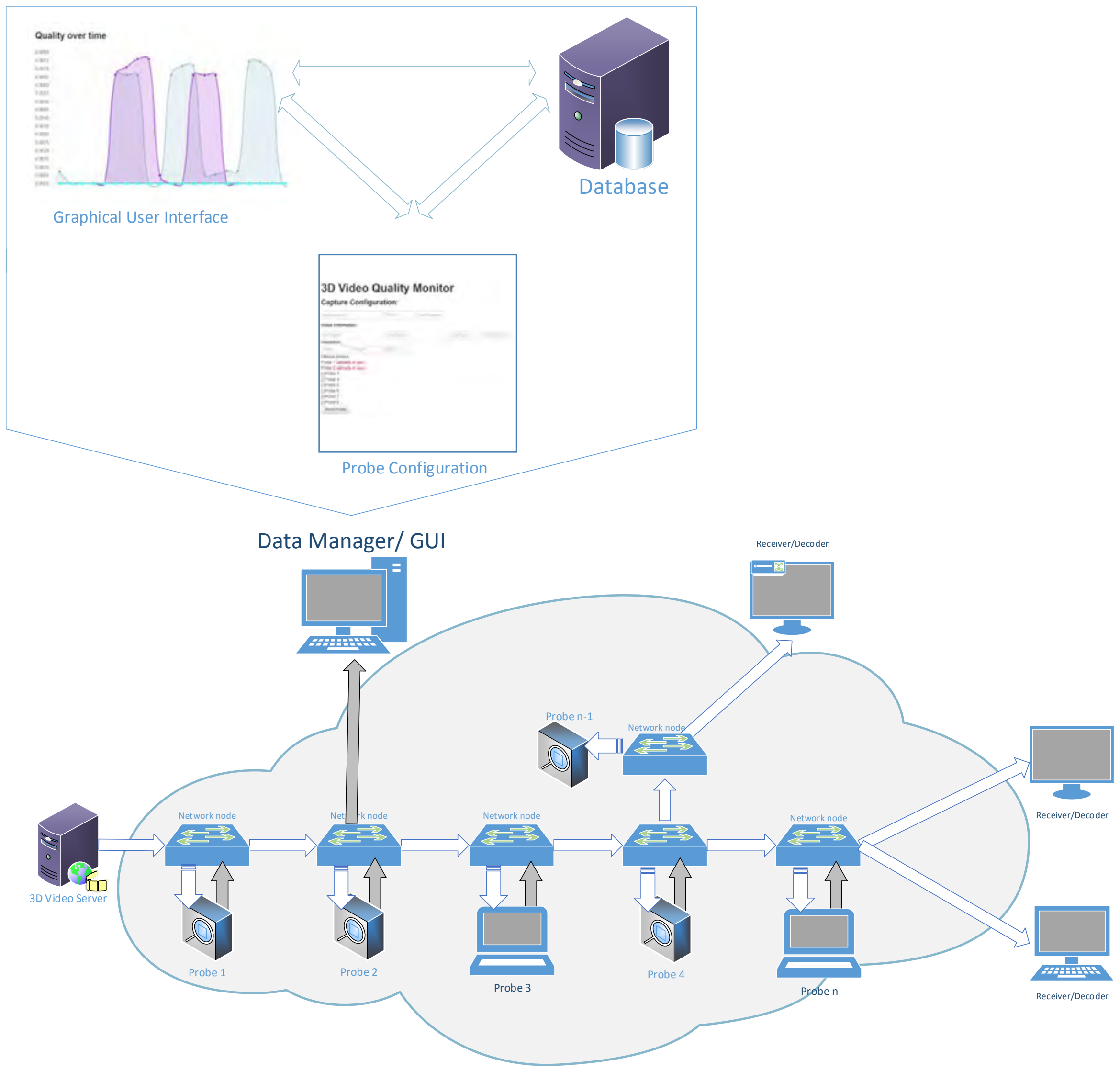


Figure 5.1: Proposed system architecture overview: white arrows - video/network traffic; grey arrows - Probe report data

Due to the IP network architectures, the Probes must be connected to a network hub or a network switch with port mirroring capabilities.

The hub connects multiple devices to each other and works by broadcasting any signal received in any port to every other ones, making packet monitoring possible to a Probe connected to it, since even if a video stream is not destined for the Probe the hub will output it to everyone of its ports. This type of device is now practically obsolete since it only operates in half-duplex and if the devices connected to it increase, the available bandwidth decreases. For testing purposes in early development stages, this option could be considered, although a network switch is a better option.

The network switch serves the same purpose as the hub, that is to connect several devices together in a computer network, but with the difference that a switch uses packet switching schemes to forward data more efficiently to the destination. It only transmits received messages to the device for which the message was intended, making this a more secure device. These features are great for a network implementation but they are not particularly helpful for the development of the current project because the Probes need to listen to network traffic that is not intended for them. To enable packet monitoring of traffic passing through, the switch must have Port Mirroring capabilities. Port Mirroring is used to send a copy of the network traffic in one port to another, enabling passive Probe monitoring (Figure 5.2). Furthermore, the network adapter in the Probe must be set to promiscuous mode in order to receive network packets that are not addressed to it.

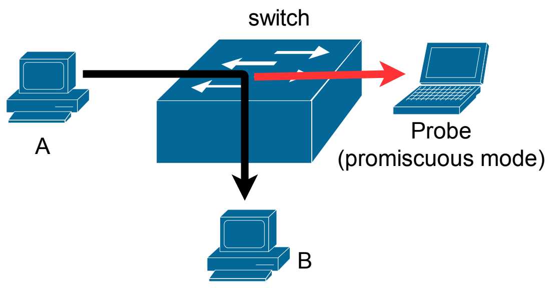


Figure 5.2: Port Mirroring illustration

In summary, 3D or 2D video is streamed from the Video Streamer to a Receiver. On its path there are a series of IP network switches with port mirroring capabilities, where Probes are connected. Each one of the Probes is monitoring all network traffic looking for MPEG2-TS packets. Once found, the Probe starts to parse packet data and gather statistics that are reported to a server. The server stores and processes the received statistical data, computes the desired quality metric and displays the video quality metric results.

5.2 Probe

The Probes are scattered throughout the target IP network, monitoring all traffic and compiling important information about the video passing through a particular node. They are the base of the system since they feed data to a central point (Server) for storage, further processing and presentation to the user. Probes are in reality a computer running a Packet Monitor and Parser software written in C++ capable of implementing all the previously described tasks.

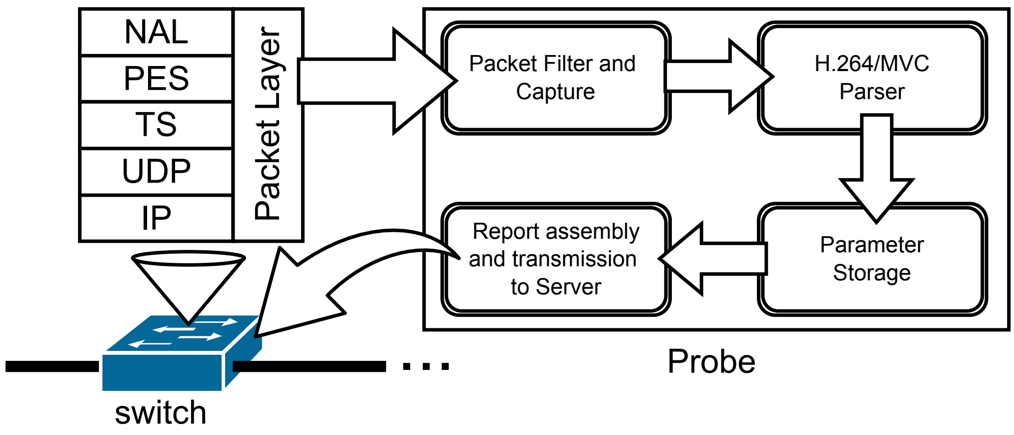


Figure 5.3: Probe high level block diagram

A Probe high level diagram is presented in Figure 5.3. A Probe starts by connecting to the server and waiting for a reply containing a packet filter and a time window value for video analysis. Once these parameters are received the filter is set, the Probe starts the network monitoring procedures by capturing and storing packets complying with the filter and storing then. Meanwhile another thread starts to analyse the captured packets to determine if the video packets belong to a 3D video stream. After this parsing and analysis for packet loss events, the Probe starts capturing and organizing all data to be reported to the server. Another thread is monitoring the quantity of stored data, constraining it to the amount correspondent to the last moments of video defined by the defined time window. Finally, yet another thread running in parallel with all previous ones, sends the data resulting from packet monitoring to the Server upon request. A Probe operation flowchart can be observed in Figure 5.4 and may help to understand the following subsections, where all main program threads are thoroughly described.

5.2.1 Packet Monitoring

Given that video bitrates are in general variable and can be very high, software implementations must be as fast, efficient, reliable and modular as possible. To accomplish this, the most critical stages of development had to be identified at an early stage. For obvious reasons, packet retrieval is the most important and the base module for the whole project. Timely and robust packet capture routines capable of supporting 3D video bitrates should be implemented before any kind of network packet frame analysis is done.

To implement such software the *WinPcap* driver and library was used. This library provides low-level network access allowing, for example, to choose the network hardware to use, to change hardware settings to allow a computer network card to operate in promiscuous mode (accept all traffic even if is not destined for it), packet filtering and capture.

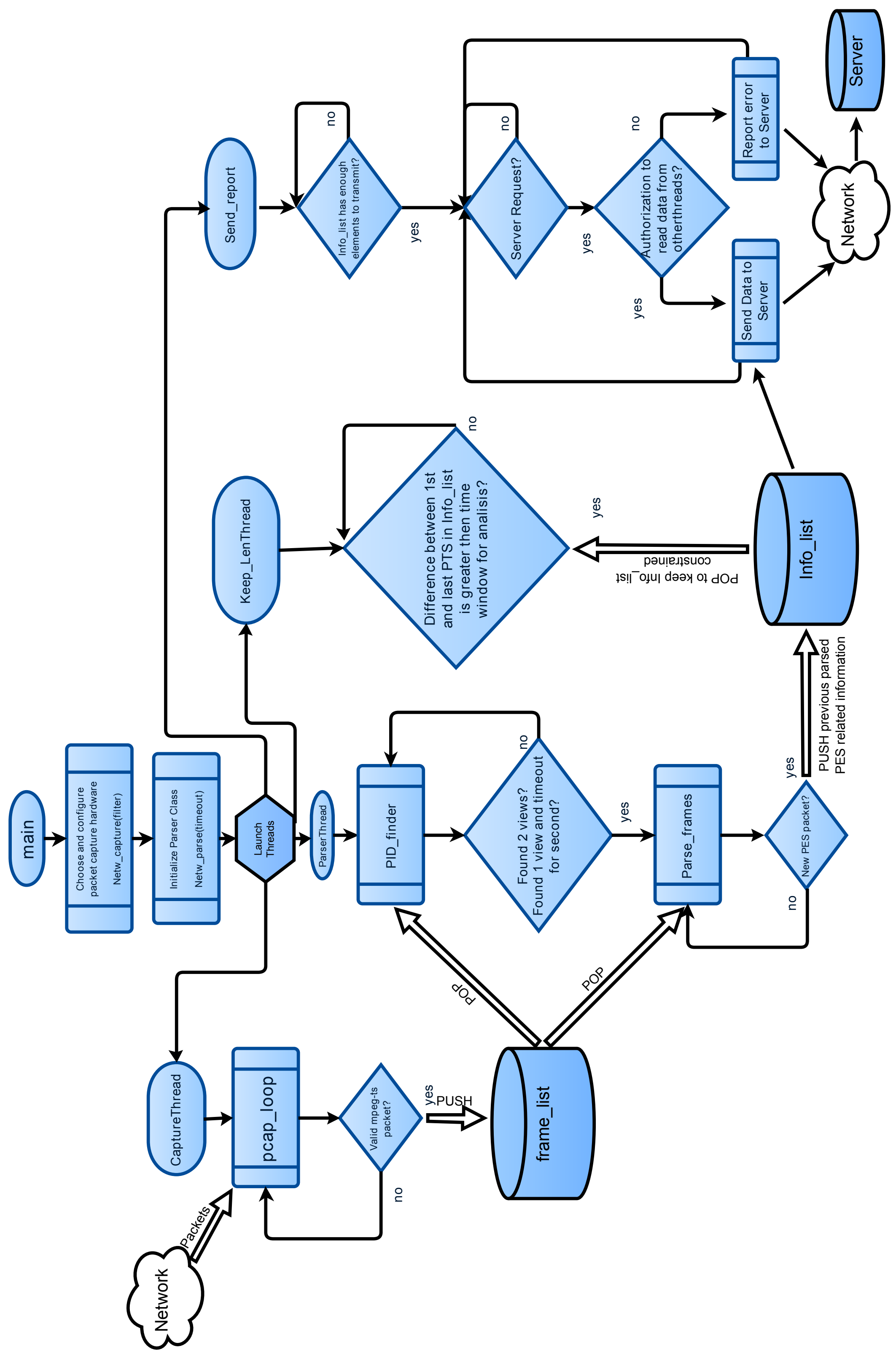

main
Choose and configure packet capture hardware
Netw_capture(filter)
Initialize Parser Class
Netw_parse(timeout)
Launch Threads
CaptureThread
ParserThread
Keep_LenThread
Send_report
Network
Packets
pcap_loop
Valid mpeg-ts packet?
yes
no
PUSH
frame_list
POP
POP
PID_finder
Found 2 views? Found 1 view and timeout for second?
yes
no
Parse_frames
New PES packet?
yes
no
PUSH previous parsed PES related information
Info_list
POP to keep Info_list constrained
Difference between 1st and last PTS in Info_list is greater then time window for analisis?
yes
no
Info_list has enough elements to transmit?
yes
no
Server Request?
yes
no
Authorization to read data from otherthreads?
yes
no
Send Data to Server
Report error to Server
Network
Server

Figure 5.4: Probe operation flowchart

Based on this library, a main program thread (pcap_loop) was developed with the single objective of storing retrieved UDP packets into memory. A filter is set, usually with the destination IP address and PORT of the video being transmitted (other combinations are possible) and when a packet that fulfils the filter conditions arrives, it is stored. This is the only action this thread is responsible for, in order to reduce the chances of packet retrieval failure due to possible time spent on any other task. Packet storage is performed by adding a pointer, pointing to the start of the retrieved network packet, to a list of pointers making this a **producer thread**.

Each member of the resulting list of pointers is a structure of type *frame* (defined on Code Example 5.1) with members pointing to other structures, defined to correspond to each packet header type. So, to do a memory mapping of the captured raw information and ease future parsing and validation actions during development four structures were created (IP, UDP, TS and PES), defining each of their major fields with the size of the correspondent data type. In this way all fields are easily and automatically accessible just by pointing the first element of the structure to the location in memory where the captured packet is. Also, using this scheme enables an efficient sweep through all TS packets contained in a UDP one, by hopping in 188 memory byte strides, instead of sweeping all data, byte by byte.

```
/* 4 bytes IP address */
typedef struct ip_address{
    u_char byte1;
    u_char byte2;
    u_char byte3;
    u_char byte4;
}ip_address;

/* IPv4 header */
typedef struct ip_header{
    u_char  ver_ihl;         // Version (4 bits) + header length (4 bits)
    u_char  tos;             // Type of service
    u_short tlen;            // Total length
    u_short identification;  // Identification
    u_short flags_fo;        // Flags (3 bits) + Fragment offset (13 bits)
    u_char  ttl;             // Time to live
    u_char  proto;           // Protocol
    u_short crc;             // Header checksum
    ip_address  saddr;       // Source address (4 byte struct)
    ip_address  daddr;       // Destination address (4 byte struct)
    u_int   op_pad;          // Option + Padding
}ip_header;

/* UDP header*/
typedef struct udp_header{
    u_short sport;          // Source port
    u_short dport;          // Destination port
    u_short len;            // Datagram length
    u_short crc;            // Checksum
}udp_header;

/**Pointers to specific parts of captured nework frame***/
struct frame{
  ip_header *ip;
  udp_header  *udp;
  TS_header *TS;  //may not exist.First TS in UDP, others every 188 byte
     strides
  PES_header  *PES; //may not exist.Point first...test after.
};
```

Code Example 5.1: Packet structure example, missing definition of MPEG-TS and PES for simplicity

In early development stages, to confirm the implemented thread packet retrieval reliability, tests were performed involving 2D video streaming using ***VLC Media Player***, the network traffic analyser software ***Wireshark*** and two separate computers: one running *VLC* (streaming) and *Wireshark*; the other running also *VLC* (playback) and the developed packet monitoring software. Since *Wireshark* is also based on *WinPcap*, identical filtering options could be used. As expected, the number of packets detected by *Wireshark* was the same as detected by the developed monitoring software.

5.2.2 Finding Video PIDs

Now that packet retrieval is assured, further analysis into their content can be performed. With the packet filter defined above, there is no guarantee that every retrieved packet is MPEG2-TS, so the first byte of every TS packet candidate (sync_byte == 0x47) must be checked.

Since it is assumed that the Probes know nothing about the incoming video, and thus not knowing the actual point in the transmission (monitoring may start at any point in a video streaming session), a second thread (PID_finder) starts looking for video PIDs in the captured packets (**packet consumer**) in order to identify what kind of video is present (2D or 3D). This is done by sweeping through all TS packets contained in UDP packets, and looking for a **Payload Unit Start Indicator** (Section 3.1.4) flag, that indicates the start of a PES packet. If that PES packet contains video information (**stream_id** equal to **0xe0**) the PID of that TS packet is stored for future use by other threads. This is done until two different PIDs from TS packets containing video are found or a specific time without finding a second PID has passed. The later condition meaning that 2D video is being transmitted through the network. This operation discards every packet it processes, but the number of discarded packets versus the packets needed for video quality assessment is negligible. Now, that video PIDs are available, all packets containing unwanted PIDs can be automatically discarded, easing even more the parsing task for the rest of the monitoring session.

5.2.3 Packet Parser

The Parser thread (Parse_frames) is the more complex thread since it parses the bitstream, analyses it and organizes all the information in a coherent way to be handed over to the Server. Figure 5.5 represents this function in a block diagram.

The information is gathered between PES packets in the following manner: a PES packet header is identified and information of PID, lenght (if present), PTS and DTS (if present) is stored; all following TS packets are scanned for NAL Units, storing its Type and Nal Ref IDC; then, in between NAL Units all lost TS packets are counted. The result is a list containing members with the structure in Code Example 5.2 for each received/identified PES packet:

```
/*After all netw. frames are parsed, resulting
info is stored on a list of type PES_info */

typedef struct PES_info{
  u_short PID;
  u_short lenght;
  u_char  totalNal;
  float PTS;
  float DTS;
  u_char  NalRefIDC[20];
  u_char  NaluType[20];
  u_short Lost_pkts[20];
}PES_info;
```

Code Example 5.2: Parsed information structure

Packet Loss Detection

Packet losses are responsible for the main contribution to video quality degradation, so detecting them is very important. They are detected using the Continuity Counter (CC) present in TS packet headers. A discontinuity in this 4 bits circular counter means that some information was lost. The number of packet losses can only be estimated by guessing the difference between the value of CC for the previous packet and for the present one of the same stream. For 3D video there are two different CCs per view, so if a CC from a previous packet is 3 and the present one is 5, one can say that at least one packet was lost but it is also correct to say that maybe 16 packets were lost due to counter overflow and restart.

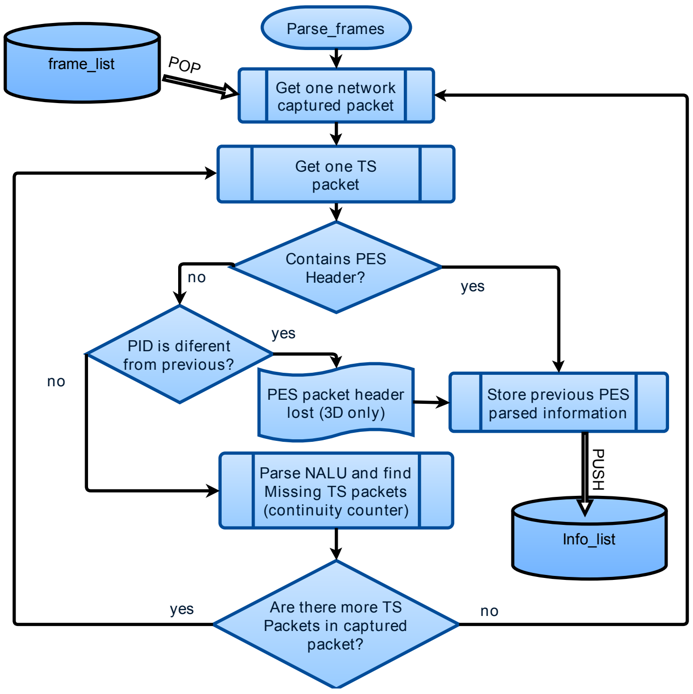

Parse_frames
frame_list
POP
Get one network captured packet
Get one TS packet
Contains PES Header?
no
yes
PID is diferent from previous?
yes
no
PES packet header lost (3D only)
Store previous PES parsed information
PUSH
Info_list
Parse NALU and find Missing TS packets (continuity counter)
Are there more TS Packets in captured packet?
yes
no

Figure 5.5: Parse_frames program flowchart

Also, lost packet categorization is needed, i.e. a packet loss must be associated with a NAL Unit Type and NAL_REF_IDC because depending on these parameters, the type of packets lost (containing I, B or P slice information) have a different impact on the perceived video quality.

To categorize NAL Units, a byte by byte sweep was implemented looking for the start sequence signalling the presence of NAL Unit header (3 bytes 0x00, 0x00, 0x01) to retrieve the NALU Type and the NAL_REF_IDC of NALUs containing VCL information (NALU Type 1, 5 or 20). There is no way to know if a TS packet containing a NAL Unit header has been lost so the number of lost packets is added to the previously detected one.

To cope with the possible loss of a PES packet header, or when its ”length” field is 0, some additional measures were implemented: if a packet PID changes and a PES packet header is not present, this means that a TS packet that contained a PES header was lost, thus the PTS cannot be determined. Additionally, if the length also cannot be promptly retrieved, it can be estimated by counting the number of received and lost TS packets between the present and the next PES packets and multiplying it by 184 (TS packet size minus header). Thus if a report arrives to the server without PTS, this means that a packet containing a PES header was lost and that the reported length is an estimation.

5.2.4 Information constraint

As the video keeps being streamed packets are being parsed and information being stored. The number of stored members in the resulting list starts to increase and to avoid filling too much memory, this list has to be constrained.

This is done by yet another thread that keeps enough number of elements in the list, corresponding to the information of the last moments of transmitted video. This parameter (last moments of video) is set in seconds at the time of Probe initialization. The transmission to the central server starts only when enough elements to meet this requirement are in the list. Stored information constraint is done by using the difference between PTS values of the last and first elements in list. If this difference is greater than the set value, list elements are discarded until the PTS difference is lower or equal than the set time window of video to analyse. In this way the network Probes can dynamically send the information to the server, upon request, using a sliding time frame, independently of the duration of the video being transmitted.

5.2.5 Report to Server

Now that all parsed information is available, it can be transmitted to the server. This is done via network sockets, because they are easily implemented in several development languages and system architectures, making it possible to treat a packet monitor Probe as a **black box**, that provides the information necessary to be analysed on request. Data transmission of the last available parsed data is done upon server request. The information amount to be sent increases proportionally with the set size for the time window of video to analyse.

Since the list storing the parsed information is constantly mutating, there is the chance that the information at server request (transmission start) is not the same at transmission end. So, to keep data consistency and prevent data access concurrency between program threads, a copy of the information list has to be made before transmission starts. For this reason, if a request for information is made to the Probe, it may reply that sending a report is not possible. This happens because the sender thread has to wait for permission to access the report data and, if it's not promptly accessible, the Probe informs the requester in order to not keeping it waiting. Please bare in mind that all threads are running simultaneously in parallel, sharing variables. If done differently, the Probes could not handle timely packet retrieval.

The report has the following format for each one of the PES packets identified by a Probe within the specified time window: **Base view PID**; **PID**; **PES Length**; **number of NALUs in PES**; **PTS**; **DTS**; **Lost_packets** (array with size equal to the number of NALUs); **Nal_REF_IDC**(array with size equal to the number of NALUs); **NALU type** (array with size equal to the number of NALUs). The requester has the task to organize all reported data for further processing upon reception as explained in Section

5.3.1.

5.2.6 Usage and Versions

Once connected to a port mirroring switch, the user must assign an identification number to the Probe, set the IP and PORT number of the server and choose the capture network interface card to use (TS_network_parser.exe ID IP PORT net_interface). In this case, a custom Server provides the means to automatically interact with the Probes, but this can also be done manually, using any SSH client, for example PuTTY.

A Probe software variant was developed to allow the parsing and a analysis of local TS files, helping in the validation of the parser results before network deployment. There is also a stand-alone version of this software capable of calculating the video quality assessment metric from Section 4.1 and reporting this value only. This version may also be used to implement any NR video quality assessment model, although such implementation may not be as straightforward as the one explained in Section 5.3.1.

5.3 Server

The server is the system node responsible for initial configuration of the network distributed Probes, receive the bulk information from them, store it on a database and show the information being received in a human readable format as shown in Figure 5.6. In reality it is comprised of two servers: one socket server and one web server (Python Tornado). The first one is responsible for handling the Probes and their data; the later renders the web interface and enables the use of WebSockets. These two servers are running in parallel sharing variables and establishing the bridge between the interface and the Probes.

On start it waits for contact from the Probes requesting for basic configuration information such as time window size for analysis, video source/destination IP and PORT combinations (for filtering purposes). The Probes wait for packets as explained in Scction 5.2 and start transmitting data to the server, when enough information is gathered and the server requests it.

To avoid processing competition between threads, a round-robin approach on data request to the Probes was adopted. In this way a scalable system could be implemented and future changes to the code become easier.

For each one of the server connected Probes, data is requested and then collected until the Probe signals the end of transmission. Then, the data is processed and stored into a database. Before asking another probe for more information, the video quality assessment metric is calculated and presented to the user on the web interface.

In the meanwhile, the server is always checking if a new Probe has checked in. If so, a new thread is launched to enable configuration of the new Probe while the round-

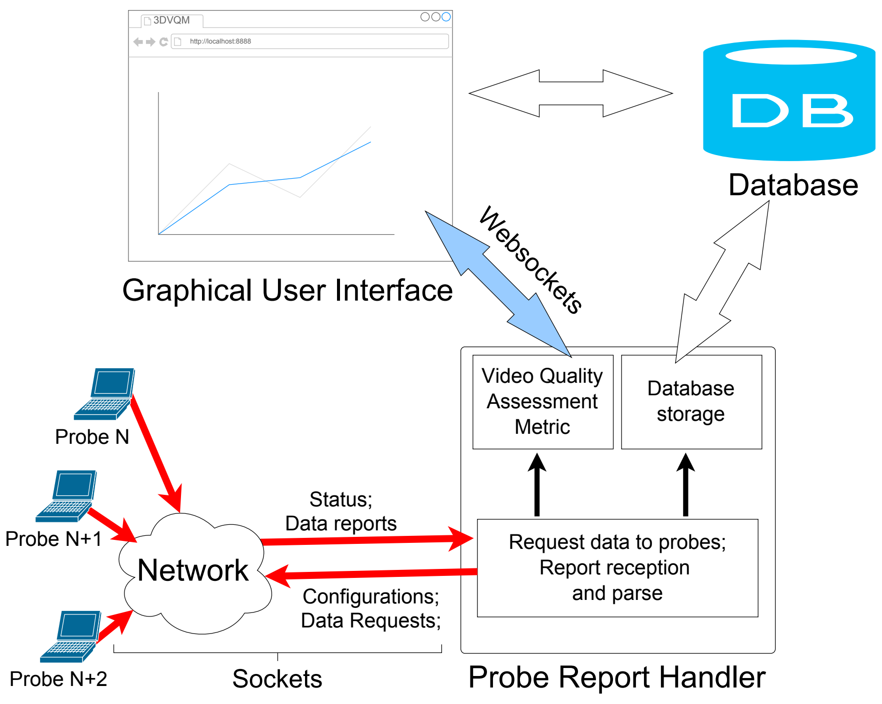


Figure 5.6: Server functional blocks

robin routine continues. Once configured the new Probe is added to the list of Probes to be asked for data. The server supports virtually a limitless number of Probe connections, meaning that database storage and video quality metric calculation is assured. In the current implementation the video interface supports the simultaneous display for 8 Probes.

5.3.1 Video Quality Assessment calculation

Since the server receives all data it is only natural to do the video quality assessment metric calculations at this point. One of the objectives of this work was to develop a way for easy implementation of any video quality assessment metric. This is done in a well defined function within the server code where all variables are easily available for use, as exemplified in Code Example 5.3. The code is in Python that is easily learnt due to being a very high level programming language. In addition, any video quality assessment algorithm can be tested in a real scenario with short metric implementation times. Code Example E.1 shows how the video quality assessment metric explained in Section 4.1 was implemented within the Server.

```
def metric_calc(data):

    metric_result = 0
#iterate through all elements
```

```
    for index, PES in enumerate(data):

        PES['Probe']
        PES['Base_PID']
        PES['PID']
        int( PES['Lenght'] )
        int( PES['total_NAL'] )
        float( PES['PTS'] )
        float( PES['DTS'] )

        for NAL in PES['NALS']:
            int( NAL['Lost_packets'] )
            NAL['Nalu_type']
            NAL['Nal_REF_IDC']

    return metric_result
```

Code Example 5.3: Example of Probe data variables access

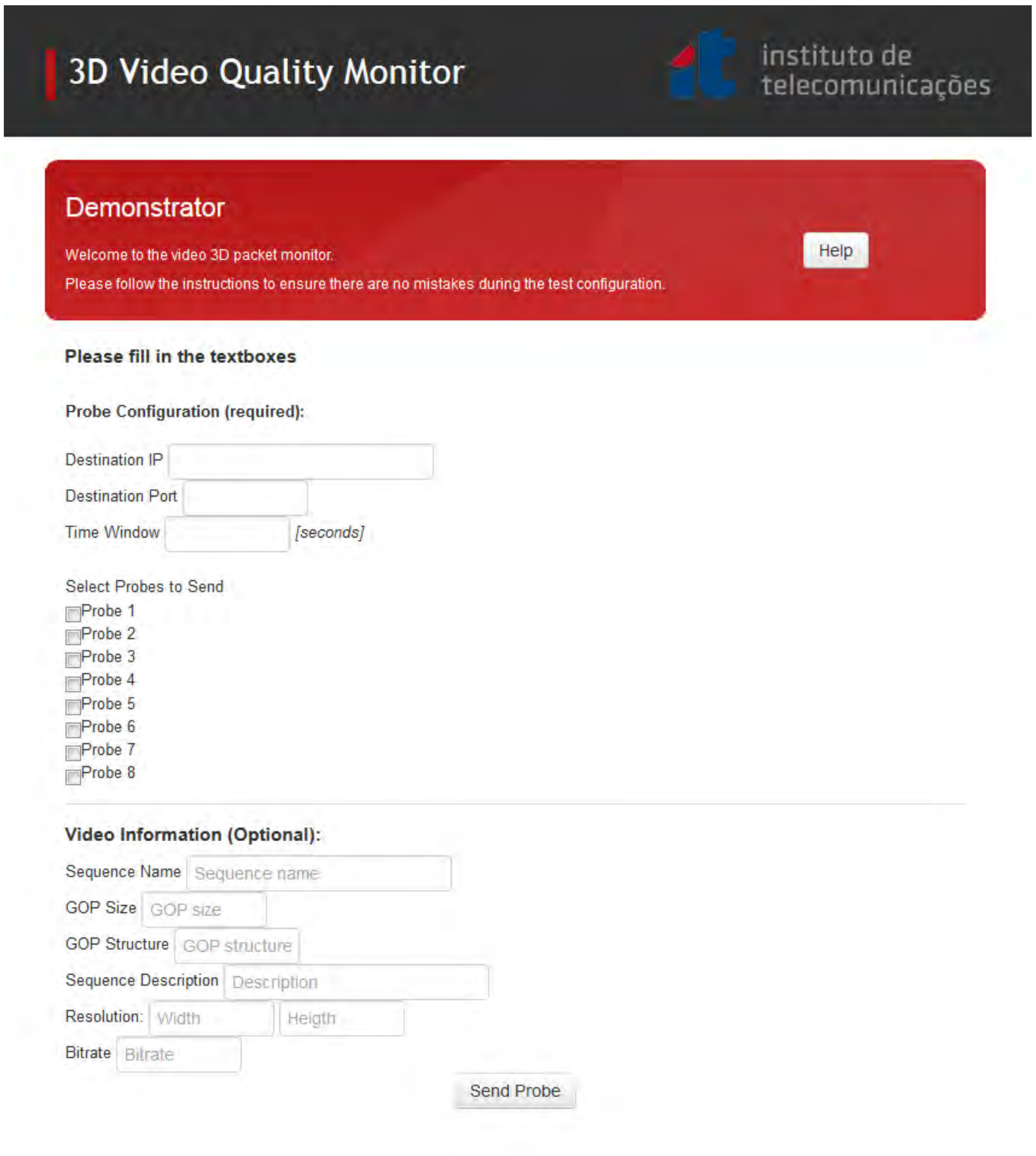
The video quality calculation function is called when a burst of data from one Probe is received, and thus calculations are done for the last moments of video passing through one Probe. All received data is organized into a list of dictionaries for easy iteration between list elements and each element of the dictionary can be called by its name. With this approach the user may center his efforts on video quality assessment metric implementation instead of worrying about how information is received and organized.

Please note that there are some differences between this implementation and the described NR model due to some implementation constraints. For example, one single TS packet cannot be lost since, if a loss event happens, it will affect the UDP packet that in turn transports 7 TS packets.

5.4 Graphical User Interface

To enable simple interaction with all system components, a Graphical User Interface was created. It is possible to set all Probe related settings, start monitoring sessions and have real time feedback of Video Quality, determined by any Video Quality Metric that is being object of study. It was designed with simplicity in mind, to keep the user away from the complexity of Probe connection and all the processing involved in receiving all the information.

The interface communicates with the server program using the recent WebSocket protocol, enabling fast and bidirectional information exchange between the web browser and the server program. So all data received from the Probes can be processed and presented to the user without being previously stored into a database, making it possible a near rea-time video quality monitoring and Probe control.


3D Video Quality Monitor

instituto de telecomunicações

Demonstrator

Welcome to the video 3D packet monitor.

Please follow the instructions to ensure there are no mistakes during the test configuration.

Help

Please fill in the textboxes

Probe Configuration (required):

Destination IP

Destination Port

Time Window *[seconds]*

Select Probes to Send

- Probe 1
- Probe 2
- Probe 3
- Probe 4
- Probe 5
- Probe 6
- Probe 7
- Probe 8

Video Information (Optional):

Sequence Name: Sequence name

GOP Size: GOP size

GOP Structure: GOP structure

Sequence Description: Description

Resolution: Width Heigth

Bitrate: Bitrate

Send Probe

Figure 5.7: Interface: Probe Status and Configuration

The interface has three main areas: Connection and Probe Status; Probe configuration and general video information; Monitoring area.

5.4.1 Connection and Probe Status

The area in the GUI destined for connection and Probe status verification is represented in Figure 5.7. It shows the health of the WebSocket connection between the server program and the browser and signals any new incoming information. Generally speaking is a way to detect if the server program is actually running.

The Probe status field shows any complementary information about the Probes and their state. For example, if a new Probe connects to the server, disconnects, reports an error occurrence or when it is ready to start transmission to the server.

5.4.2 Capture Configuration

This is a group of form fields that are used for Probe and session configuration (Figure 5.7). The mandatory fields, corresponding to a packet filter that will be used by a probe, are ”destination IP”, ”destination PORT” and ”time_window”. However, it is recommended to fill all fields since they are stored into the database and can help in future data retrieval and analysis. All fields correspond to one or several Probes, depending on user selection.

The ”Manual commands” field is for single Probe communication and debugging purposes. The user may send a command to a particular Probe, bypassing all implemented automated procedures.

5.4.3 Video Quality Assessment Graphs

After all Probes are configured, the user may start a monitoring session. Once in this mode the server will send the assigned configuration to the corresponding Probes that may have already established a connection with the server. The server will then wait for Probe feedback, stating that enough video information has been parsed to produce a report. If a new Probe establishes a connection to the server in the meanwhile, it will be configured accordingly, meaning that a Probe may be configured without being previously connected.

The number of graphs showing the calculated video quality assessment metric present in the interface will vary depending on the number of Probes previously configured. They show the evolution of the last 40 quality values (Figure 5.9), enabling instant visual comparison. Lastly, a bar graph shows the last calculated video quality assessment metric value, by Probe (Figure 5.8). Furthermore, the quality values are dependent of the quality metric being tested, for example in the case of the N-R quality model proposed in Section 4.1.1, a value of 0 means that video is being transmitted without any errors and, if any variation happens, it means that a probe detected a video degradation.

Bellow these two graphs is a table (Figure 5.10) with general information about the video that was retrieved and that does not change that often, such as :

- **Base_PID**: PID value of base view;
- **PID** : secondary view PID;
- **Avg_Nal** : average number of NALU's per PES packet;
- **Avg_Length** : average PES packet length;
- **Lost I, P and B** : number of lost TS packets ordered by type, from the last Probe to report data.

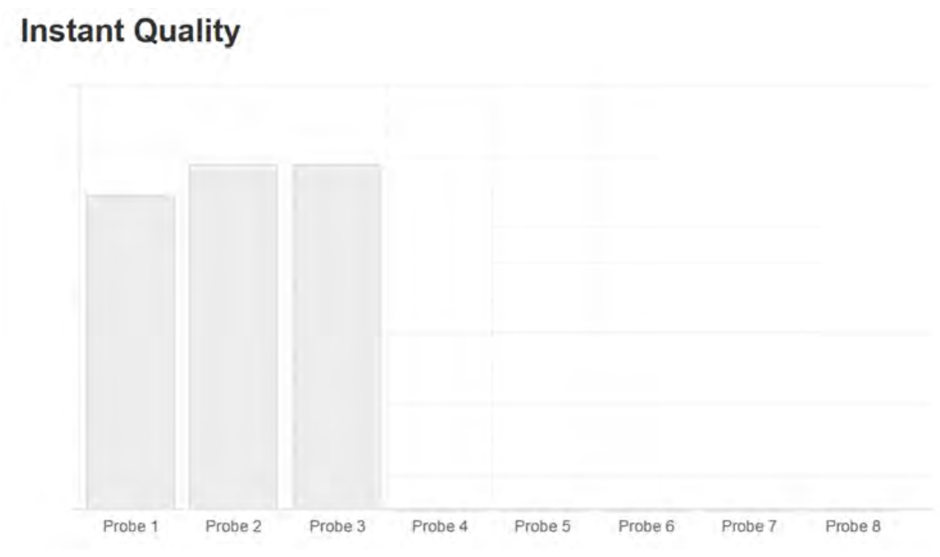


Figure 5.8: Interface: Instant quality

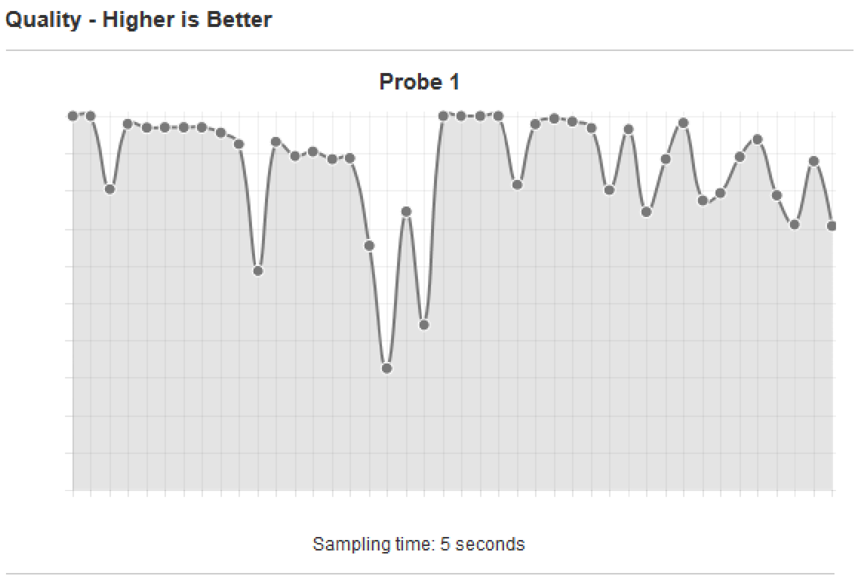
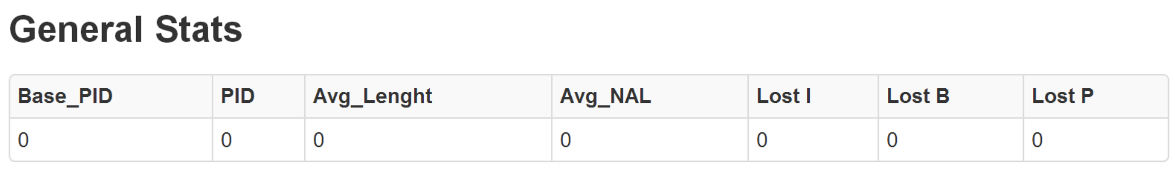


Figure 5.9: Interface: Quality progress for a single Probe



General Stats

Base_PID	PID	Avg_Lenght	Avg_NAL	Lost I	Lost B	Lost P
0	0	0	0	0	0	0

Figure 5.10: Interface: Table with general information from the captured video

5.5 Database

A database was designed to store all gathered information for future reference (Figure 5.11), so it had to be general purpose and independent of any specific implemented Video Quality Metric. It could store quality information calculated in a particular experimental session but the used metric must be well identified. The most correct way to do it is to simply store all data from the Probes along with reception timestamps and other relevant information regarding the particular video being object of study in a particular session. This generic storage scheme means that the available data for future analysis is kept flexible and can be used to test new VQA schemes without constraints. The disadvantage is the database will ending up storing a huge number of members in its, tables very fast, since all data from all the Probes is gathered.

All information can be available just by queering the database for the wanted data to perform off-line tests for a real case scenarios or for various network testing conditions.

5.5.1 Table description

This section describes each one of the tables that form the database, and how they relate to each other to form a coherent and redundancy free database (Figure 5.11):

Session

This table stores general information about a monitoring session.

- **idsession**: (primary key) identifies a particular monitoring session;
- **description**: field describing a session with complementary information;
- **video_transmited**: (foreign key) links to video_info table;
- **src_ip**: source IP of video stream;
- **src_port**: source PORT of video stream;
- **dst_ip**: destination IP of video stream;
- **dst_port**: destination PORT of video stream;

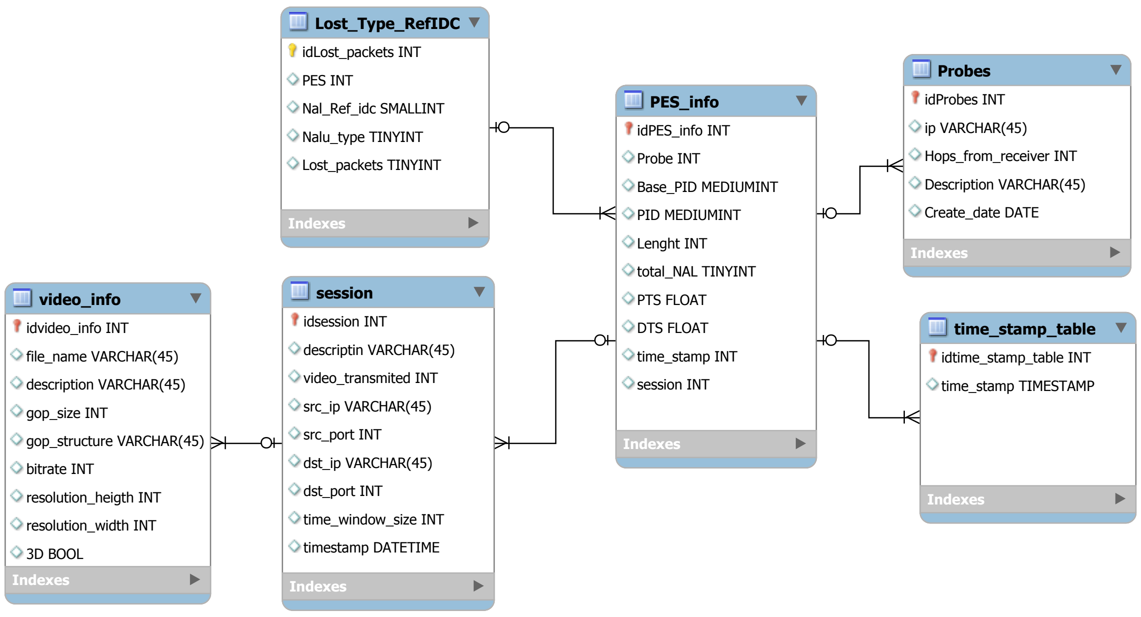


Figure 5.11: Entity Relationship Diagram of the system database

- **time_window_size**: time window size of the last moments of video to be gathered by the Probes;
- **timestamp**: (foreign key) links to time_stamp_table. States the session start.

Video_info

To help in future analysis of the stored information, the streamed videos have to be identified. Also there are important intrinsic characteristics that can be used in video quality assessment.

- **idvideo_info**: (primary key) identifies a particular video to be streamed and its characteristics;
- **file_name**: file name of the video;
- **description**: more particular information other than the present fields;
- **gop_size**: distance between two I-frames;
- **gop_structure**: specifies the order intra- and inter-frames are arranged within a video stream;
- **bitrate**: encoding bitrate;
- **resolution_height**;
- **resolution_width**;
- **3D**: boolean to state if the video is 3D or 2D.

Probes

Their identification and geographical distribution is important, and it is stored into this table.

- **idProbes**: (primary key) probe unique identification number;
- **IP**: probe IP address;
- **hops_from_receiver**: contains the distance from the receiver a probe is, in network node hops;
- **description**: additional data about Probe location;
- **create_date**: date it was added to the system.

Time_stamp_table

Since there will be intensive use of repeated timestamps throughout the database, this simple table helps to prevent information redundancy.

- **id_time_stamp_table**: (primary key) identifies a particular timestamp;
- **time_stamp**: actual timestamp value;

PES_info

Each burst of information sent by the Probes contain several PES specific data:

- **idPES_info**: (primary key) unique identification for each received PES related data;
- **Probe**: (foreign key) link to **Probes** table. Identifies which Probe sent this piece of data;
- **Base_PID**: base view PID;
- **PID**: actual PID of this particular PES. If Base PID is equal to PID, means that this PES is from base view;
- **Lenght**: PES lenght;
- **Total_NAL**: Number of NAL units in PES;
- **PTS**: Presentation Timestamp. If 0 this PES header was not present and its length was a prediction;
- **DTS**: Decoding Timestamp. May be 0;
- **time_stamp**: (foreign key) links to **Time_stamp_table**;
- **session**: (foreign key) links to **Session**. Points to the session this data refers to.

NALUS

Every entry of this table stores data referring to individual NAL Units.

- **idNALU**: (primary key) identifies every NAL unit;
- **PES**: (foreign key) links to **PES_info**, relating a PES with the several NALUs it contains;
- **Lost_packets**: lost TS packets in a particular NALU;
- **Nalu_type**: NALU type;
- **Nalu_ref_idc**: NAL reference IDC.

After some test runs of the system, the database had enough entries to test its structure by querying for some interesting values. Off-line data processing is out the scope of this work, but for future reference, some examples of data querying are presented here. All values resulting from these examples are from tests done during debugging and system reliability tests so they may not have a specific practical meaning.

Query 5.4 presents the amount of lost packets sorted by type in a particular session:

```sql
select probes_info.pes_info.PID,
sum(probes_info.lost_type_refidc.lost_packets) as 'lost packets',
    probes_info.lost_type_refidc.Nalu_type from pes_info
left join probes_info.lost_type_refidc
on probes_info.pes_info.idPES_info = probes_info.lost_type_refidc.PES
group by probes_info.lost_type_refidc.Nalu_type;
```

Code Example 5.4: Example of a Database query

5.6 Future HEVC implementation

This software code can be used for a similar implementation for the latest Video Standard or any other, being MPEG-TS packetization the only constraint to prevent an over complicated adaptation.

If the HEVC bitstream is packetized into MPEG-TS, only the function responsible for NAL Unit parsing and identification has to be modified, because only the NALU header will be different. Figure 5.12 shows the NALU header fields. They are described in Annex B of [4] and [41]:

- **forbidden_zero_bit** : this bit must be 0. This bit exists to enable transport of HEVC video over MPEG-2 transport systems (avoidance of start code emulations);

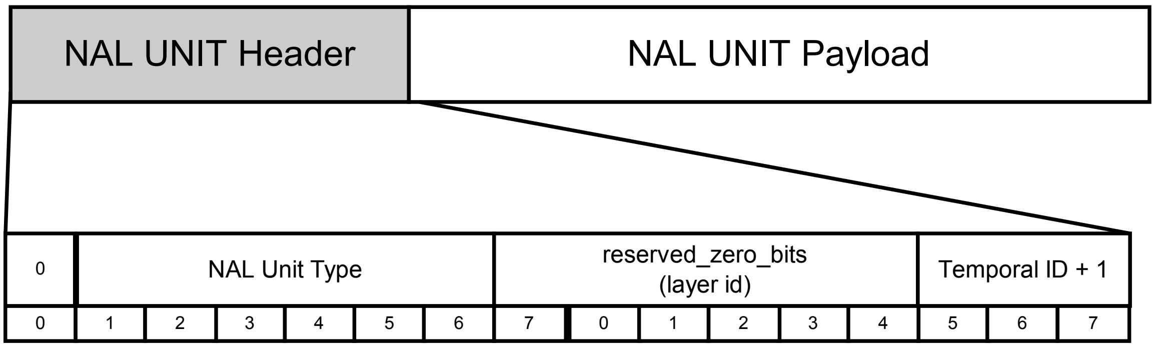


Figure 5.12: HEVC NALU header fields

- **NAL Unit Type** (6 bits): this field specifies the NAL Unit type. If this value is less than 32, means that a NAL Unit is a VCL NAL Unit and carries video information;
- **layer id** (6 bits): equal to zero (for now). Anticipating to be used in the future for scalable or 3D video coding extensions, and may identify additional layers for example, spatial scalable layer, a quality scalable layer, a texture view, or a depth view;
- **Temporal ID + 1** (3 bits): this field specifies the temporal identifier of the NAL unit plus 1. A TID value of 0 is illegal to ensure that start code emulation does not happen.

From the above information it seems premature to do an implementation with 3D video quality assessment in mind, since the standard does not yet define the presence of this type of video at NALU level. This leaves 2D as the only option but, from a development point-of-view, parsing the HEVC NALU header is easily implemented and only a Probe code function needs to be changed. With this change, information to retrieve also changes but is also simple to add new variables to the structure responsible for storing the data from the network parsed frames. The rest of the Probe code should work with minor changes while server and database must also be changed to accommodate the new variables.

Other obvious requirement is the quality metric that has to be fitted for HEVC in a similar way as the one described in Chapter 4 in order to understand the effects packet loss have on decoded video.

Another problem is the lack of tools available for HEVC MPEG2-TS packetization, streaming and reproduction over the network.However, as soon as these problems are solved, a version similar to the developed system should work with minor changes.

Chapter 6

Functional tests

This Chapter firstly describes a MPEG2-TS 3D video streamer, specially developed to enable testing of the Probe software during the development stages as well as final system testing. The implemented video quality assessment metric is also tested for several video sequences and network packet losses, analysing the resulting video impairments. Furthermore, the test setup and results for the final version of the network monitoring system are presented in order to assess reliability and solution behaviour upon deployment.

6.1 MPEG2-TS 3D Video Streamer

In order to test the video stream parser code on early development stages, only 2D video streams were used since a public domain 3D video streamer solution did not exist. This was done until the point in development that, for further advances, 3D video streams had to be used. Thus to be able to continue with the project, a basic MPEG-TS video streamer was created. It took multiplexed 3D MPEG-TS video files and read the TS packets contained within them, 7 at a time (the maximum a UDP packet can carry as explained in Section 3.1.5) and streamed them via an UDP socket to the receiver. This was done having no concerns on packet transmission timings, meaning all data was being dumped to the network at the fastest possible speed the network or interface network card could support. This meant the video playback at the receiver had poor quality because its buffer was overflowing with video data and packets were being discarded upon reception. Also the packet retrieval Probes and *Wireshark* had problems to recognize all packets at such speeds. To attenuate the problem, and since the objective was to detect packet losses, another version of this software was created to delay packet transmission. Transmission rate was simply determined by dividing the number of packets to transmit by the video duration. Such approach also produced bad video quality at the receiver because not all packets have the same importance or impact on the video being played, thus they cannot be treated as equals. From a development point-of-view this was good enough for testing purposes because all packets were being streamed and received within the same time as the video duration, enabling 3D video tests.

After more progress was made on 3D video packet monitoring, there was the need to test the implemented code with an actual streamer capable of produce an acceptable video playback at the receiver. 2D Video could be properly tested using *VLC* or *FFMPEG*, but there was no way to know if all developments could cope with Full HD 3D Video bitrates. Some very time consuming attempts to modify the *FFMPEG* source code to make it capable of streaming 3D video were also conducted. This proved to be too complex for the scope of this particular project and should maybe considered as a project on itself, since all H264 MVC standard would have to be implemented into this code. Trying to bypass the checks *FFMPEG* does before start streaming also proved to

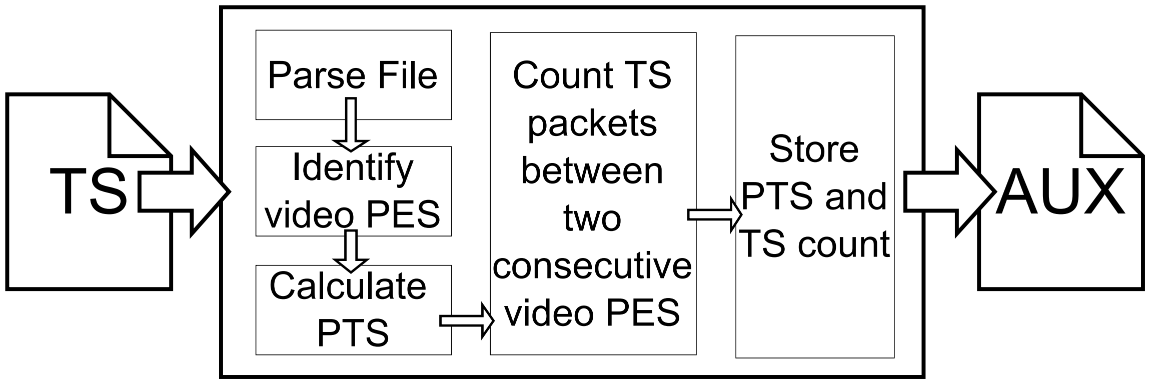


Figure 6.1: Auxiliary file generation for 3D video Streamer

The two solutions left were to buy a professional 3D streaming solution or develop a streamer capable of producing acceptable results. Of course the faster solution was the second since it could be based on the first versions of the previously developed software. It has to be capable to deliver a satisfactory video quality independently of the video being transmitted. This meant developing a program capable of interpret the MPEG-TS file prior to transmission to understand the packet transmission timing. The program (Figure 6.1) analyses the file to transmit (only the first time the file is submitted for transmission) and creates an auxiliary file containing two columns: the first contains the PTS of every PES packet in the file; the second has the number of TS packets between two consecutive PES packets. This file is then used to get the difference between two different consecutive PTS values and to see if the video is 2D or 3D (if two consecutive PTS values are equal the video is 3D). The difference between the PTS values gives the presentation periodicity of a PES packet, so all the data of one PES has to be transmitted on that time frame independently of its size, resulting in UDP packet bursts to the network. Knowing if a video is 3D or 2D is necessary since (in this implementation) transmission periodicity of a 3D PES packet is half periodicity fo a 2D packet. This does not mean, that 3D PES packets contains double the information but that two PES packets need

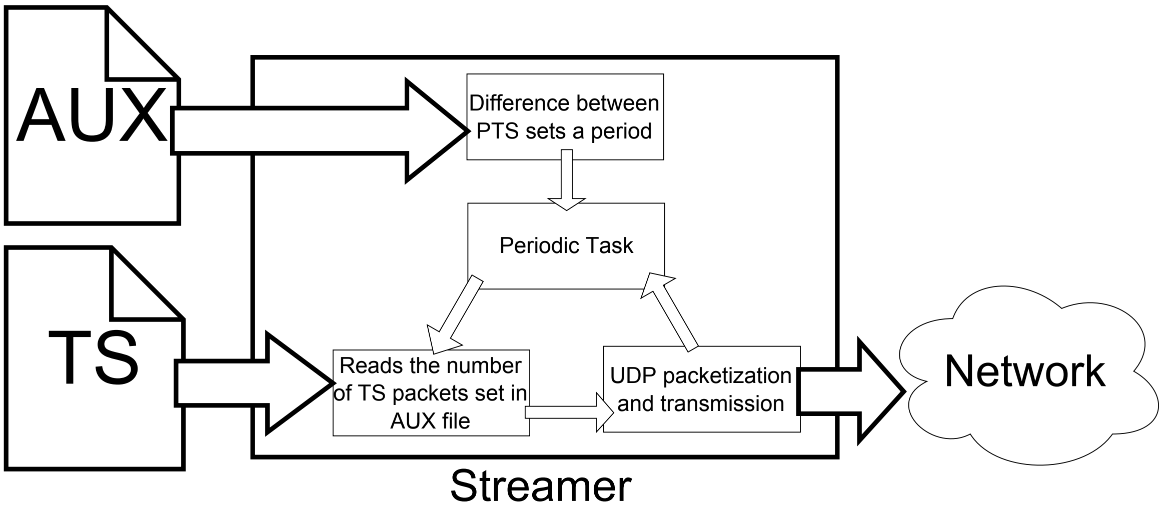

AUX
TS
Difference between PTS sets a period
Periodic Task
Reads the number of TS packets set in AUX file
UDP packetization and transmission
Network
Streamer

Figure 6.2: 3D video Streamer block diagram

This code is also capable of simulating random and periodic packet losses, a feature that proved to be very useful to test the Probe reliability. The streamer signals when a packet is ”lost” (not transmitted) and the Probes have to react accordingly.

6.1.1 Packet loss using the Gilbert-Elliot model

In the early stages of the 3D video implementation, a very simple packet loss scheme was implemented to ease the debugging task, by simulating periodic packet losses at several user defined packet loss percentages. Of course this does not accurately characterize a lossy network, so a way to simulate packet losses had to be studied and implemented. Based on [42] the Gilbert-Elliot packet loss model was chosen, since packet losses usually occur in bursts and not in a Bernoulli random manner, thus making this a great model to use for demonstration purposes.

This model was introduced in [43] and [44], and is a stochastic packet loss model based on a two-state Markov process (Figure 6.3). It simply has a Bad (packet not transmitted) state, a Good (packet transmitted) state and the transition probabilities p and q, between these two states. In short, once a packet is transmitted the system transits to the Bad (S_1) state or remains in the Good (S_0) state; if a packet is not transmitted the system transits to the Good state or remains in the Bad state. Since the

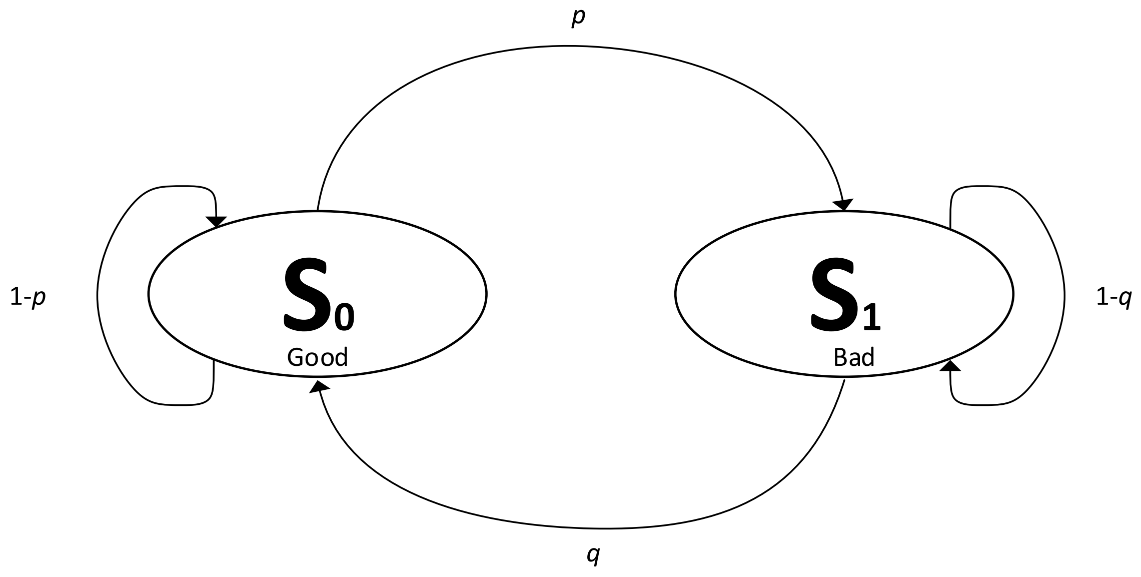


Figure 6.3: Gilbert-Elliot model representation

model only retains the previous state, the probability of a packet not being sent depends on the state the system is currently at. The transition probabilities may be calculated from the average packet loss rate (PLR) and the mean burst length (MBL). From [45]:

$$p = P(S_{i+1} = 1 \mid S_i = 0) = \left[MBL \cdot \left(\frac{1}{PLR} - 1 \right) \right]^{-1} \quad (6.1)$$

$$q = P(S_{i+1} = 0 \mid S_i = 1) = \frac{1}{MBL} \quad (6.2)$$

$$frac1PLR > 1 + \frac{1}{MBL}, \quad 0 < PLR < 1 \quad , \quad MBL \geq 1 \quad (6.3)$$

The streamer version using this model loops through a list of *PLR* ranging from 0.5% to 4% , that give enough degradation for demonstration purposes and a *MLB* of 3 UDP packets since the Continuity Counter of a TS packet ranges from 0 to 15 and a UDP packet carries 7 TS packets, a burst length of 3 is already too much for an accurate packet loss detection, so there is no point in making this value greater. The streamer is very similar to the previously developed versions until the transmission point, where the model decides if an already assembled UDP packet is transmitted or discarded (packet loss simulation). A simple Python script in Annex E shows an example of the Gilbert-Elliot model implementation.

6.2 Test Environment

To properly test the system and prevent error occurrence due to external conditions it had to be isolated from the existing network infrastructure. There is huge network pollution with all kinds of broadcast packets being transmitted at any given time. The objective of network isolation is to make sure the only traffic passing through is video from the transmitter to the receiver. Also this is the only way to test more than two Probes at a time. To have more than two Probes operating, the test scenario presented in Figure 6.4 was created, where:

- **1**: Receiver 1;
- **2**: Probe 1;
- **3**: Server: Quality Monitor / 3D video Streamer;
- **4**: Probe 2;
- **5**: Receiver 2.
- **6**: Network switch with port mirroring capabilities (HP ProCurve 2810-24G).

Prior to testing, the switch had to be configured to assign a mirroring port, the two other ports to be mirrored and the switch IP address. After this, all satellite components could be connected to form a working network, paying attention to connect the Streamer and Receiver to the mirrored ports and the remaining Probe to the mirror port. The Server can be connected to any other of the remaining switch ports.

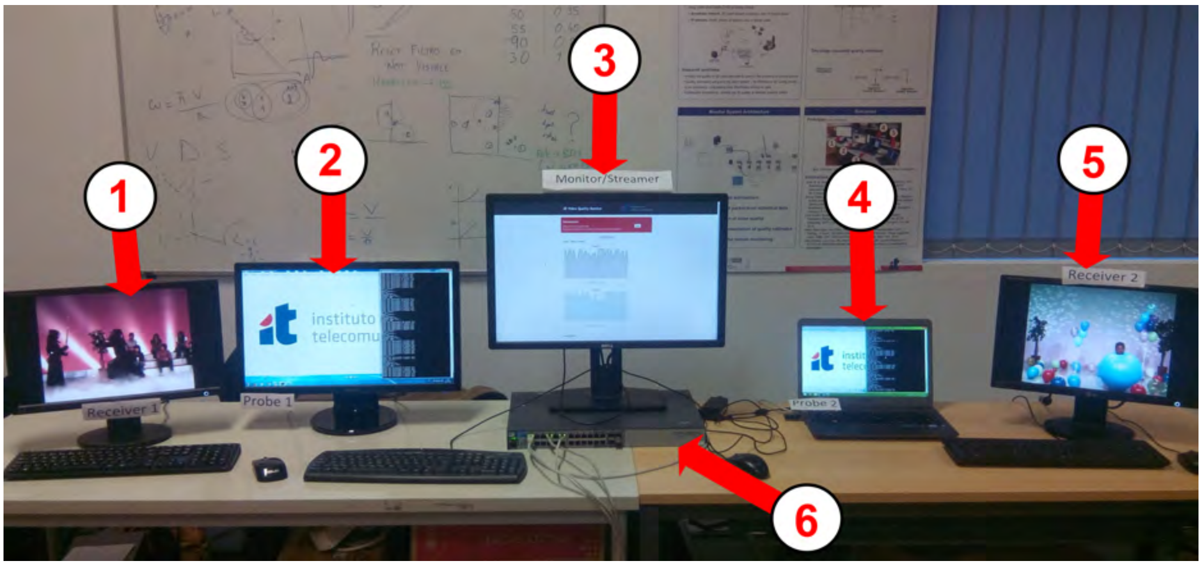


Figure 6.4: Experimental setup

6.3 Reliability tests

The most basic test was to stream a Full HD 3D video sequence to the receiver and visually confirm, using *VLC*, if it was being played without any artefacts or jitter. Although 3D video is transmitted, *VLC* plays it discarding all data from secondary views.

Another test that had to be performed was Probe connection to the Server and associated configuration. This is done by submitting the mandatory configuration values set in the interface (streaming destination IP, PORT and video time window to analyse) and connect every Probe by assigning an Identification number, Server IP and PORT and select a network interface. The interface should show that there are new Probe connections and show their identification numbers as well as information about configuration status.

Now that every component is tested individually, it is time for a full test with all Probes reporting to the Server, and to observe the video quality assessment metric being determined in 3 different points of the network. As expected, no packet losses were detected by the Probes (for the Streamer node it is obvious that such thing cannot happen since all outgoing data is captured) and as a result all results were shown accordingly in the GUI.

Reliability was tested with all the following combinations: stopping/starting streaming at random, Probes react by considering packet losses until MPEG-2 TS Continuity Counter synchronization is retained again; disconnect/connect Probes from/to the Server, assigning a different ID and/or keeping the same. In what concerns the server, if a Probe is disconnected, a timeout is raised when data is to be requested to that Probe and it is removed from the request queue. When a Probe reconnects with the same ID or a new one it will be added to the request queue only if the ID is not already part of the queue.

Overall, the system behaves well under the above conditions, although it is possible that a never foreseen condition could cause an unexpected failure or data loss.

6.4 NR Video Quality Model Tests

In order to determine the effect packet losses have the on video being transmitted, and to assess if the implemented metric would work in a real case scenario, some tests were performed using only one probe located on the receiver and a streamer/server running the version of the MPEG2-TS 3D Video Streamer capable of network packet loss simulation.

6.4.1 NR Model Parameters

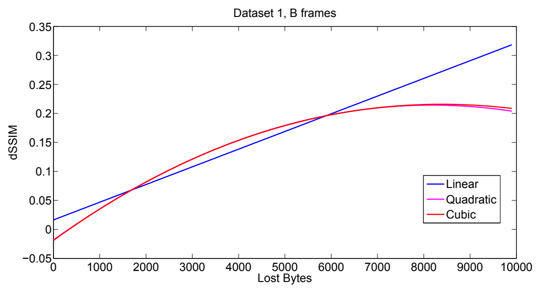
In Section 4.1.1 the proposed N-R model is defined by Equation 4.5 whose parameters had to be chosen from Tables C.1 and C.2. To decide on a Dataset of parameters to use, every one of the possibilities for Equation 4.5 was plotted in function of the number of bytes lost (Figure 6.5). This allowed to evaluate which one of the combinations would better behave once implemented in the video quality monitor. Keeping in mind that a PES packet may be comprised of several TS packets that can be as much as 700, meaning that for this case a PES packet length is 128000 bytes (700×184 TS packet payload size). One has to guarantee that if a portion of a packet with these characteristics is lost the metric will still be able to compute a valid result. For example, all Cubic functions from all Datasets cannot be used because at a certain point they would start measuring video improvements with packet loss increase. The same can be said for Quadratic functions in Figures 6.5a, 6.5c,6.5f and 6.5f, for greater packet loss values, meaning that a pair of Quadratic dSSIM functions per Dataset that could be used does not exist. This leaves only the Linear functions from which Dataset 2 was selected, leaving Equations 6.4 and 6.5 to be implemented on the video quality assessment monitor.

$$dSSIM_P = 2.61 \times 10^{-05} L_{fs} - 0.04488 \tag{6.4}$$

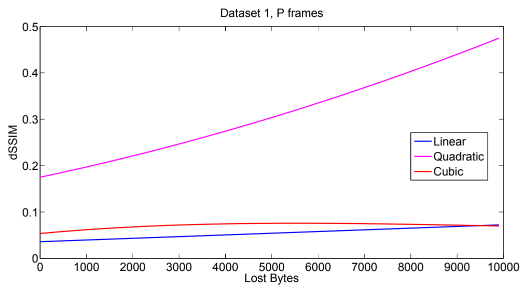
$$dSSIM_B = 4.38 \times 10^{-05} L_{fs} - 0.006689 \tag{6.5}$$

6.4.2 NR Model Tests

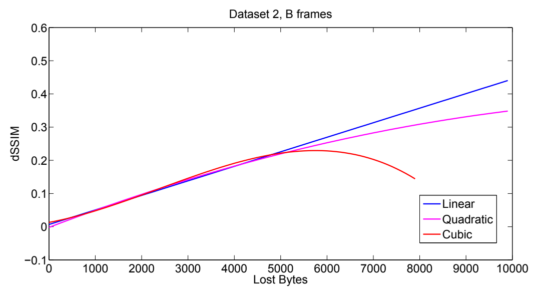
For these tests the video sequences used were Dog (Figure 6.6a) , Ballons (Figure 6.6b), Bullinger (Figure 6.6c) and ESTG Bus Stop (Figure 6.6d) were used. Dog and Balloons



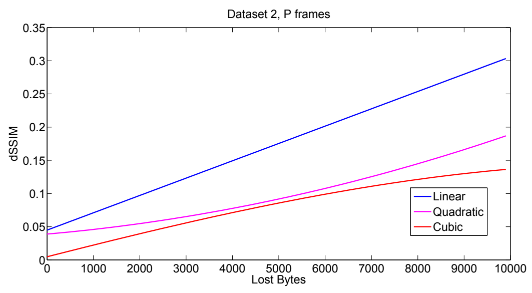
(a) Dataset 1, B frames



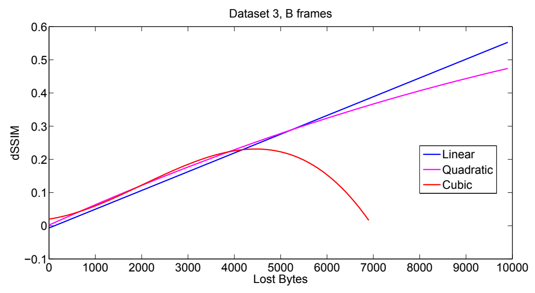
(b) Dataset 1, P frames



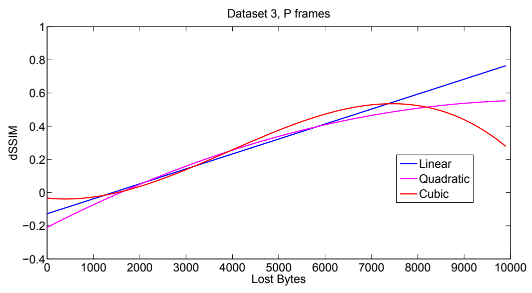
(c) Dataset 2, B frames



(d) Dataset 2, B frames



(e) Dataset 3, P frames

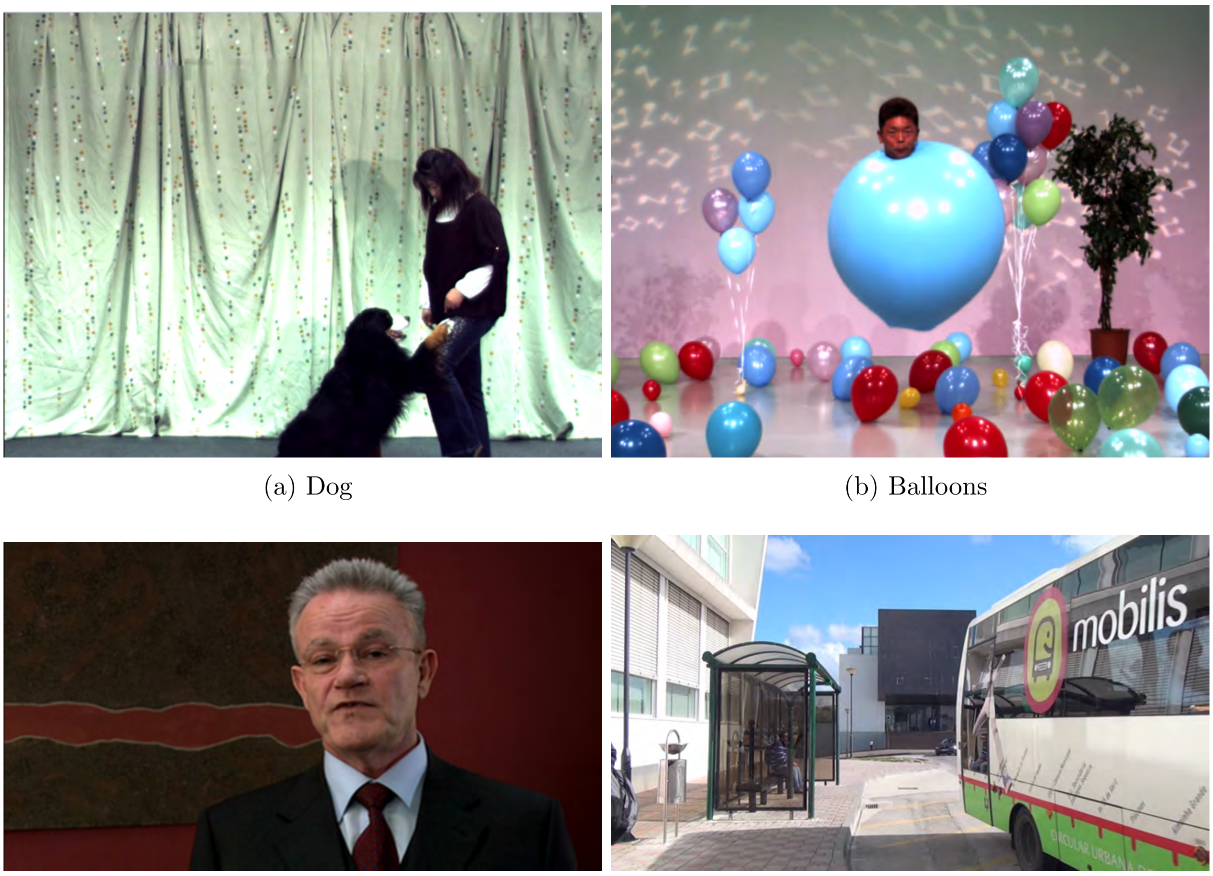


(f) Dataset 3, B frames

Figure 6.5: dSSIM in function of packet losses

sequences are originally too short for these tests (300 frames, 10 seconds), so each one of the views was concatenated with itself to form a 2 minutes sequence. Bullinger and ESTG Bus Stop already had enough length. To encode the sequences Dog, Balloons and Bullinger into H.264/AVC MVC-3D *FRIM Encoder* software from *Intel* was used enabling the creation of H.264 elementary streams (ES) that were then multiplexed into MPEG2-TS using *tsMuxeR* software. ESTG Bus Stop sequence is different from the others because it was produced by Instituto de Telecomunicações using a 3D Digital HD Video Camera Recorder (Sony HXR-NX3D1U NXCAM) that produces already packetized MPEG2-TS video stream files that can be used without any further processing. The videos used have their main characteristics enumerated in Table 6.1.

To test the video quality model each one of the TS encapsulated video streams was transmitted using the 3D Video Streamer software version capable of simulating UDP packet losses, with varying packet loss percentage of 0.1%, 0.5%, 1%, 2%, 4% and 8%. There is no need for testing further from 8% because subjective video quality degradation



(a) Dog

(b) Balloons

(c) Prof. Bullinger

(d) ESTG bus stop

Figure 6.6: Video sequences used for metric evaluation

at this point is already too much. The Probe was configured to analyse the last 5 seconds of the transmitted video. The Server, upon receiving a Probe report calculates the dSSIM for the base and secondary views as follows: for each report entry, lost packets from each type (I, P or B) are counted; dSSIM is calculated for P and B using Equations 6.4 and 6.5; for I type packets, dSSIM is 1 (total loss); the average of all calculated dSSIMs (by view) is performed and stored for further processing. The same will happen for a 2D video, and the base-view dSSIM can be used as a 2D video quality indicator for devices supporting only H.264 2D video streams. Figure 6.7 is the result of these procedures where results are organized by Probe report.

The computed results are then averaged to produce an average dSSIM value for a given UDP packet loss percentage. Figure 6.9 shows these results and Figure 6.8 compares the base and secondary views for one single video sequence.

From analysing the Figures 6.9a and 6.9b one can conclude that comparison between the determined video quality of the videos is not possible because they all evolve in the same way with packet losses. The quality from a video passing through the network can only be compared with the quality from the same video, or one with the same characteristics, passing at another network point.

Table 6.1: Video sequence characteristics

Sequence	Resolution	Frames	Bitrate (Mbbps)	Framerate (fps)
Dog	1280x960	1800	20	30
Balloons	1024x768	3000	20	30
Bullinger	432x240	1622	20	25
ESTG Bus	1920x1080	1152	22	23.976

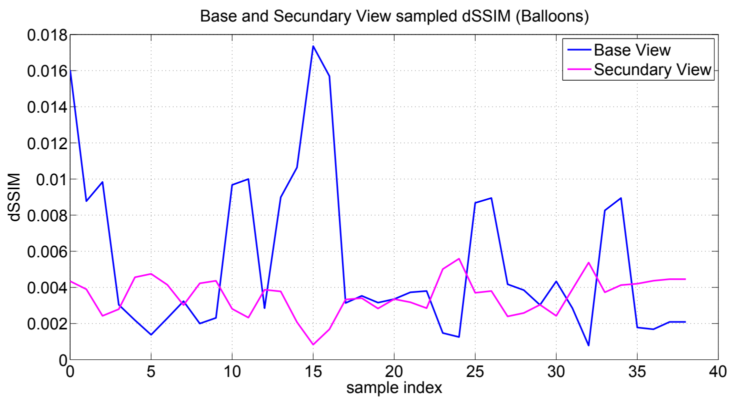


Figure 6.7: Calculated dSSIM values by Probe report

Still images taken from the video sequences, present in Annex F, help to understand the impact that the UDP packet losses have on the quality of the video being transmitted.

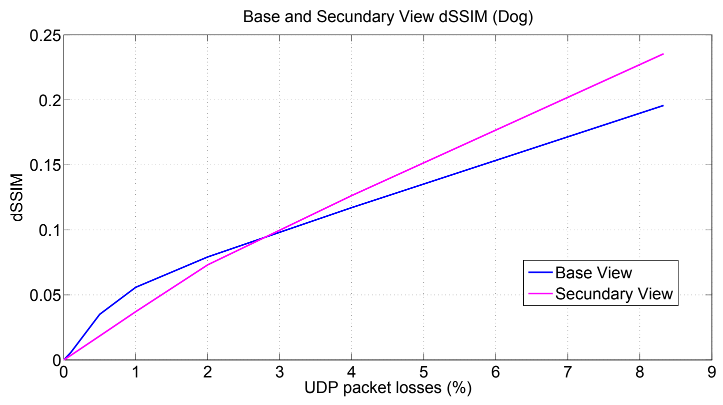
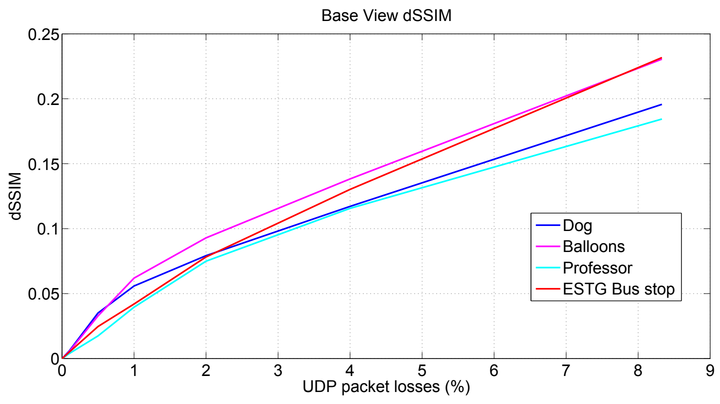
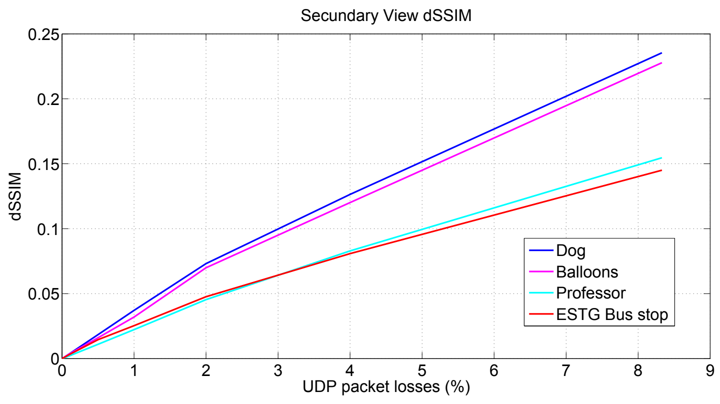


Figure 6.8: Base and secondary views dSSIM comparison



(a) Base view dSSIM



(b) Secondary view dSSIM

Figure 6.9: dSSIM in function of UDP packet loss percentage

Chapter 7

Conclusions and future work

This chapter concludes this project report, presenting some conclusions about this research and development work and some future research perspectives in the field of 3D video network quality monitoring.

7.1 Conclusions

The subject addressed in this project description was the creation of a set of tools to help further develop the research in the field of objective quality evaluation of 3D video over packet-loss-prone transmission channels. All the objectives proposed in Section 1.2 were achieved.

The developed system performs well, is flexible, reliable, scalable, easily deployable and customizable. The modular architecture enables the Probes to be treated as a black box that gives feedback about the content of the video stream passing through a given IP network point, enabling the deployment as a stand-alone monitoring device or the integration on a distributed monitoring system. The Server is the bridge between the Probes and the GUI. It is responsible for establishing communication with the Probes and ask for their monitoring results in order to calculate any video quality assessment metric. The Server also supports virtually unlimited Probe connections at any point of a monitoring session, storing all received data and presenting it in a GUI. The GUI was created with functionality in mind and was kept as simple as possible. It enables setting all parameters for Probe configuration as well as parameters for the network monitoring session. Although this project only covers data storage into a database, it was created to reduce data redundancy and keep all tables and relations as coherent and simple as possible, thus easing the future task of data querying.

One of the main objectives is to enable the implementation of any video quality metric for testing. In Section 5.3.1 there is an example showing how all variables from the Probes are accessible within the Server, abstracting future users from Probe report acquisition an processing, leaving them the task of implementing their own video quality model. In addition, in Annex B, the metric explained in Section 4.1 is fully implemented for illustrative purposes.

As a complement to the developed system, required for system development and test, a 3D video streamer software was developed. It produces a good video quality at the receiver and was of great help for test and validation. A version that simulates packet losses was used, in order to assess the effect that the packet loss percentage has on the video quality, as well as to calculate video quality metric results.

7.2 Future work

Although this research work presents a powerful set of software tools that may help in future developments regarding 3D video quality assessment, there are several aspects that can be addressed in the future and that may lead to a higher performance of the network monitor or to widen this field of study.

- **Probe support for more network packetization schemes**. As explained in Section 3.1.5 there are several packetization schemes that could be implemented in the Probe source code. This would enable, for example, the study of the impact that packet losses for each one of the transmission methods has in perceived video quality.

- **Inclusion of depth-enhanced video formats**. Implement identification and parsing of 3D video using depth information for view rendering, as specified in Annex I of H.264 specification [16].

- **Implement other NR video quality assessment metrics** for several other video formats such as V+D or MVD and compare relative video impairment for the same network conditions.

- **Analyse the data captured by the Probes**. The data from the database can be used off-line to conceive new video quality assessment metrics, to be subjected to data mining, or for neuronal network training.

- **HEVC implementation and support**. This subject was already addressed and explained in detail on Section 5.6. Objective video quality assessment metrics for this standard are yet to be explored and a tool like the one developed on the scope of this work would help a lot.

- **Embedded Probe Version**. Port the Probe code to an embedded platform to improve mobility, installation requirements and the reduction of power consumption for long monitoring sessions.

Bibliography

[1] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, Jul. 2003.

[2] I. E. G. Richardson, *THE H.264 ADVANCED VIDEO COMPRESSION STANDARD*, 2nd ed. UK: Wiley, 2010.

[3] T. Wiegand, G. J. Sullivan, J.-R. Ohm, and W.-J. Han, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, Dec. 2012.

[4] *High efficiency video coding*, ITU-T Recommendation H.265, Apr. 2013.

[5] A. Vetro, T. Wiegand, and G. J. Sullivan, "Overview of the Stereo and Multiview Video Coding Extensions of the H.264/MPEG-4 AVC Standard," 2011.

[6] B. T. dos Santos Feitor, "3DVQM: 3D Video Quality Monitor," Master's thesis, Escola Superior de Tecnologia e Gestao, Instituto Politecnico de Leiria, Nov. 2013.

[7] *The Zettabyte Era - Trends and Analysis*, Cisco Visual Networking Index, Cisco, May 2013.

[8] F. Battisti, M. Carli, and A. Neri, "No reference quality assessment for MPEG video delivery over IP," *EURASIP Journal on Image and Video Processing*, Jan. 2014.

[9] M. Jacobs and J. Probell, "A Brief History of Video Coding," *ARC International*, 2007.

[10] *Video Codec for Audio Visual Services at p x 64kbit/s*, ITU-T. Recommendation H.261, 1990.

[11] *Information Technology - Digital Compression and Coding of Continuous-Tone Still Images (JPEG)*, ISO/IEC IS 10918, 1994.

[12] *Information Technology-Coding of Mooving Pictures and Associated Audio for Digital Storage Media up to about 1.5 Mbit/s (MPEG-1) - Part 2: Coding of Moving Picture Information*, ISO/IEC. 11172-2, 1993.

[13] *Information Technology-Generic Coding of Moving Pictures and Associated Audio Information (MPEG-2) - Part 2: Video*, ISO/IEC. 13818-2, 1995.

[14] *Video Coding for Low Bitrate Communication*, ISO/IEC. 13818-2, 1995.

[15] F. Lopes, "Motion Estimation for Very Low Bitrate Video Coding," *University of Essex*, Jul. 2002.

[16] *Advanced video coding for generic audiovisual services*, ITU-T Recommendation H.264, Apr. 2013.

[17] T. L. Silva, L. V. Agostini, and L. A. da Silva Cruz, "Fast HEVC Intra Prediction Mode Decision Based On Edge Direction Information," in *20th European Signal Processing Conference (EUSIPCO 2012)*, Bucharest, Romania, Aug. 2012, pp. 43–53.

[18] G. Correa, P. Assuncao, L. Agostini, and L. A. da Silva Cruz, "Coding Tree Depth Estimation for Complexity Reduction of HEVC ," in *Data Compression Conference*, 2013, pp. 1214–1218.

[19] V. Nozick, "Camera array image rectification and calibration for stereoscopic and autostereoscopic displays," *Ann. Telecommun.*, pp. 581–596, Nov. 2013.

[20] P. Merkle, K. Müller, and T. Wiegand, "3D Video: Acquisition, Coding, and Display," *IEEE Transactions on Consumer Electronics*, vol. 56, pp. 946 – 950, May 2010.

[21] Y.-K. Wang and T. Schierl, "RTP Payload Format for MVC Video," *Internet Draft*, Apr. 2010.

[22] J. Postel, "Internet Protocol," *RFC 791*, Sep. 1981.

[23] ——, "User Datagram Protocol," *RFC 768*, Aug. 1980.

[24] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," *RFC 3550*, Jul. 2003.

[25] Y.-K. Wang, R. Even, T. Kristensen, and R. Jesup, "RTP Payload Format for H.264 Video," *RFC 6184*, May 2011.

[26] *Generic coding of moving pictures and associated audio information: Systems*, ITU-T Recommendation H.222.0, ISO/IEC 13818-1, Oct. 2000.

[27] B. G. Haskell, A. Puri, and A. N. Netravali, *Digital Video: An Introduction to MPEG-2*, Kluwer Academic Publishers, 2000.

[28] D. Hoffman, G. Fernando, V. Goyal, and M. Civanlar, "RTP Payload Format for MPEG1/MPEG2 Video," *RFC 2250*, Jan. 1998.

[29] *Digital Video Broadcasting (DVB); Transport of MPEG-2 Based DVB Services over IP Based Networks*, DVB IP Phase 1 handbook, ETSI TS 102 034, Mar. 2005.

[30] A. Kordelas, T. Dagiuklas, and I. Politis, "ON THE PERFORMANCE OF H.264/MVC OVER LOSSY IP-BASED NETWORKS," *EUSIPCO 2012*, pp. 1149–1153, Aug. 2012.

[31] B. W. Micallef and C. J. Debono, "An analysis on the effect of transmission errors in real-time H.264-MVC Bit-streams," *MELECON 2010*, pp. 1215 – 1220, Apr. 2010.

[32] Q. Huynh-Thu, M. Barkowsky, and P. L. Callet, "The Importance of Visual Attention in Improving the 3D-TV Viewing Experience: Overview and New Perspectives," *IEEE TRANSACTIONS ON BROADCASTING*, vol. 57, p. 421–431, Jun. 2011.

[33] M. Urvoy, M. Barkowsky, and P. L. Callet, "How visual fatigue and discomfort impact 3D-TV quality of experience: a comprehensive review of technological, psychophysical, and psychological factors," *Ann. Telecommun.*, p. 641–655, Sep. 2013.

[34] C. Sun, X. Liu, X. Xu, and W. Yang, "An effcient quality assessment metric for 3D video," *Computer and Information Technology, 2012 IEEE 12th International Conference*, pp. 209–213, 2012.

[35] J. G. Apostolopoulos and A. R. Reibman, "The Challenge of Estimating Video Quality in Video Communication Applications," *IEEE SIGNAL PROCESSING MAGAZINE*, pp. 160, 156 – 158, Mar. 2012.

[36] Z. Wang, L. Lu, and A. C. Bovik, "Video quality assessment based on structural distortion measurement," *SIGNAL PROCESSING: IMAGE COMMUNICATION*, vol. 19, Jan. 2004.

[37] Z. P. Sazzad, S. Yamanaka, Y. Kawayoke, and Y. Horita, "STEREOSCOPIC IMAGE QUALITY PREDICTION," *QoMEX 2009*, 2009.

[38] M. Solh and G. AIRegib, "A NO-REFERENCE QUALITY MEASURE FOR DIBR-BASED 3D VIDEOS," *IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6, 2011.

[39] I. Sedano, K. Brunnstrom, M. Kihl, and A. Aurelious, "Full-reference video quality metric assisted the development of no-reference bitstream video quality metrics for real-time network monitoring," *EURASIP Journal on Image and Video Processing 2014*, 2014.

[40] J. R. S. Soares, "Quality Evaluation of 3D Video Subject to Transmission Channel Impairments," Master's thesis, Faculdade De Ciências e Tecnologia Da Universidade De Coimbra, Sep. 2013.

[41] Y.-K. Wang, Y. Sanchez, T. Schierl, S. Wenger, and M. M. Hannuksela, "RTP Payload Format for High Efficiency Video Coding," *Internet Draft*, Aug. 2014.

[42] M. M. Nasralla, C. T. E. R. Hewage, and M. G. Martini, "Subjective and Objective Evaluation and Packet Loss Modeling for 3D Video Transmission over LTE Networks," *International Conference on Telecommunications and Multimedia (TEMU)*, pp. 254–259, 2014.

[43] E. N. Gilbert, "Capacity of a burst-noise channel," *Bell system technical journal*, vol. 39, p. 1253–1265, 1960.

[44] E. Elliott, "Estimates of error rates for codes on burst-noise channels," *Bell system technical journal*, vol. 42, pp. 1977–1997, 1963.

[45] R. Skupin, C. Hellge, T. Schierl, and T. Wiegand, "Packet level video quality evaluation of extensive H.264/AVC and SVC transmission simulation," *Journal of Internet Services and Applications*, vol. 2, p. 129–138, Jun. 2011.

Appendix A

NAL Unit Types

Table A presents all NALU types for H.264. MVC corresponds to the column containing **Annex G and Annex H NAL unit type class**.

Table A.1: NAL unit type codes, syntax element categories, and NAL unit type classes

nal_unit _type	Content of NAL unit and RBSP syntax structure	Annex A NAL unit type class	Annex G and Annex H NAL unit type class	Annex I NAL unit type class
0	Unspecified	non-VCL	non-VCL	non-VCL
1	Coded slice of a non-IDR picture slice_layer_without_partitioning_rbsp( )	VCL	VCL	VCL
2	Coded slice data partition A slice_data_partition_a_layer_rbsp( )	VCL	not applicable	not applicable
3	Coded slice data partition B slice_data_partition_b_layer_rbsp( )	VCL	not applicable	not applicable
4	Coded slice data partition C slice_data_partition_c_layer_rbsp( )	VCL	not applicable	not applicable
5	Coded slice of an IDR picture slice_layer_without_partitioning_rbsp( )	VCL	VCL	VCL
6	Supplemental enhancement information (SEI) sei_rbsp( )	non-VCL	non-VCL	non-VCL
7	Sequence parameter set seq_parameter_set_rbsp( )	non-VCL	non-VCL	non-VCL
8	Picture parameter set pic_parameter_set_rbsp( )	non-VCL	non-VCL	non-VCL
9	Access unit delimiter access_unit_delimiter_rbsp( )	non-VCL	non-VCL	non-VCL
10	End of sequence end_of_seq_rbsp( )	non-VCL	non-VCL	non-VCL
11	End of stream end_of_stream_rbsp( )	non-VCL	non-VCL	non-VCL
12	Filler data filler_data_rbsp( )	non-VCL	non-VCL	non-VCL
13	Sequence parameter set extension seq_parameter_set_extension_rbsp( )	non-VCL	non-VCL	non-VCL
14	Prefix NAL unit prefix_nal_unit_rbsp( )	non-VCL	suffix dependent	suffix dependent
15	Subset sequence parameter set subset_seq_parameter_set_rbsp( )	non-VCL	non-VCL	non-VCL
16..18	Reserved	non-VCL	non-VCL	non-VCL
19	Coded slice of an auxiliary coded picture without partitioning slice_layer_without_partitioning_rbsp( )	non-VCL	non-VCL	non-VCL
20	Coded slice extension slice_layer_extension_rbsp( )	non-VCL	VCL	VCL
21	Coded slice extension for depth view components /*specified in Annex I */ slice_layer_extension_rbsp( ) /* specified in Annex I */	non-VCL	non-VCL	VCL
22..23	Reserved	non-VCL	non-VCL	VCL
24..31	Unspecified	non-VCL	non-VCL	non-VCL

Appendix B

Function responsible for quality metric calculation

This Annex contains the portion of the Server code responsible for video quality calculation. It should be treated as an example to help future implementations and to understand the dynamics of this code, since this particular function should be edited to accommodate the calculation of any new metric. The entry parameter for this function is a report (list of data) from a single probe. The returned values from this function is sent to the GUI.

```python
def metric_calc(data_burst):

    dSSIM_B = 0 #Base
    LostP_B = 0
    LostB_B = 0
    LostI_B = 0
    dSSIMP_B = 0
    dSSIMB_B = 0
    dSSIMI_B = 0
    B_elements = 0

    dSSIM_A = 0 #Auxiliar
    LostP_A = 0
    LostB_A = 0
    LostI_A = 0
    dSSIMP_A = 0
    dSSIMB_A = 0
    dSSIMI_A = 0
    A_elements = 0

    dSSIMT_B = 0 #Final result base
    dSSIMT_A = 0 #Final result auxiliar

    for index, PES in enumerate(data_burst):

        if PES['Base_PID'] == PES['PID']: #base view
```

```

            B_elements = B_elements + 1

            LostP_B = 0
            LostB_B = 0
            LostI_B = 0

            for NAL in PES['NALS']:

                if int( NAL['Lost_packets'] ) > 0:

                    if ((int(NAL['Nal_REF_IDC']) > 0) and ( int(NAL['Nalu_type']) != 5 )):#P slice
                        LostP_B = LostP_B + int( NAL['Lost_packets'] )

                    elif ((int(NAL['Nal_REF_IDC']) == 0) and ( int(NAL['Nalu_type']) != 5 )):#B slice
                        LostB_B = LostB_B + int( NAL['Lost_packets'] )

                    elif ((int(NAL['Nal_REF_IDC']) > 0) and ( int(NAL['Nalu_type']) == 5 )):#I slice
                        LostI_B = LostI_B + int( NAL['Lost_packets'] )

            if LostP_B > 0 :
                if int( PES['Lenght'] ) <= 0 : #not possible (in theory), just prevention
                    dSSIMP_B = 1
                else:
                    dSSIMP_B = (LostP_B*188) * 0.0000261 + 0.0448

                dSSIMT_B = dSSIMT_B +  dSSIMP_B

            elif LostB_B > 0:
                if int( PES['Lenght'] ) <= 0 : #not possible (in theory), just prevention
                    dSSIMB_B = 1
                else:
                    dSSIMB_B = (LostB_B*188) * 0.0000438 + 0.006689

                dSSIMT_B = dSSIMT_B +  dSSIMB_B

            elif LostI_B > 0:

                dSSIMT_B = dSSIMT_B +  1.0 #ALL IS LOST!

        else: #auxiliar view

            A_elements = A_elements + 1
            LostP_A = 0
            LostB_A = 0
            LostI_A = 0

            for NAL in PES['NALS']:

                if int( NAL['Lost_packets'] ) > 0:

                    if ((int(NAL['Nal_REF_IDC']) > 0) and ( int(NAL['Nalu_type']) != 5 )):#P slice
                        LostP_A = LostP_A + int( NAL['Lost_packets'] )

                    elif ((int(NAL['Nal_REF_IDC']) == 0) and ( int(NAL['Nalu_type']) != 5 )):#B slice
                        LostB_A = LostB_A + int( NAL['Lost_packets'] )

                    elif ((int(NAL['Nal_REF_IDC']) > 0) and ( int(NAL['Nalu_type']) == 5 )):#I slice
                        LostI_A = LostI_A + int( NAL['Lost_packets'] )


            if LostP_A > 0 :
                if int( PES['Lenght'] ) <= 0 : #not possible (in theory), just prevention
                    dSSIMP_A = 1
                else:
                    dSSIMP_A = (LostP_A*188) * 0.0000261 - 0.0448

                dSSIMT_A = dSSIMT_A +  dSSIMP_A

            elif LostB_A > 0:
                if int( PES['Lenght'] ) <= 0 : #not possible (in theory), just prevention
                    dSSIMB_A = 1
                else:
                    dSSIMB_A = (LostB_A*188) * 0.0000438 + 0.006689

                dSSIMT_A = dSSIMT_A +  dSSIMB_A

            elif LostI_A > 0:

                dSSIMT_A = dSSIMT_A +  1.0 #ALL IS LOST!


    dSSIMT_A = dSSIMT_A / A_elements
    dSSIMT_B = dSSIMT_B / B_elements
    return [dSSIMT_B, dSSIMT_A]
```

Code Example B.1: Example of VQA metric calculation

Appendix C

VQA Metric parameter tables

Table C.1: Curve fitting coefficients for P-frames

	Linear	Quadratic	Cubic
Dataset-1	p0=0.03596	p0=0.03596	p0=0.05365
	p1=3.66E-06	p1=3.66E-06	p1=9.29E-06
	-	p2=9.26E-10	p2=-1.19E-09
	-	-	p3=4.22E-14
Dataset-2	p0=-0.04488	p0=3.89E-02	p0=4.74E-03
	p1=2.61E-05	p1=6.11E-06	p1=1.78E-05
	-	p2=8.92E-10	p2=-1.87E-10
	-	-	p3=2.72E-14
Dataset-3	p0=-0.1276	p0=-0.2097	p0=-0.03292
	p1=9.01E-05	p1=1.43E-04	p1=-2.92e-05
	-	p2=-6.66E-09	p2=3.86E-08
	-	-	p3-3.28E-12

Table C.2: Curve fitting coefficients for B-frames

	Linear	Quadratic	Cubic
Dataset-1	p0=0.01636	p0=-1.83E-02	p0=-1.90E-02
	p1=3.05E-05	p1=5.69E-05	p1=5.78E-05
	-	p2=-3.48E-09	p2=-3.77E-09
	-	-	p3=2.57E-14
Dataset-2	p0=0.006689	p0=-2.04E-03	p0=1.30E-02
	p1=4.38E-05	p1=5.34E-05	p1=2.57E-05
	-	p2=-1.80E-09	p2=1.07E-08
	-	-	p3=-1.50E-12
Dataset-3	p0=-0.006671	p0=2.07E-03	p0=2.01E-02
	p1=5.65E-05	p1=6.29E-05	p1=2.13E-05
	-	p2=-1.54E-09	p2=2.23E-08
	-	-	p3=-3.69E-12

Appendix D

Graphical User Interface

This Annex shows how the GUI when viewed on a Web Browser.

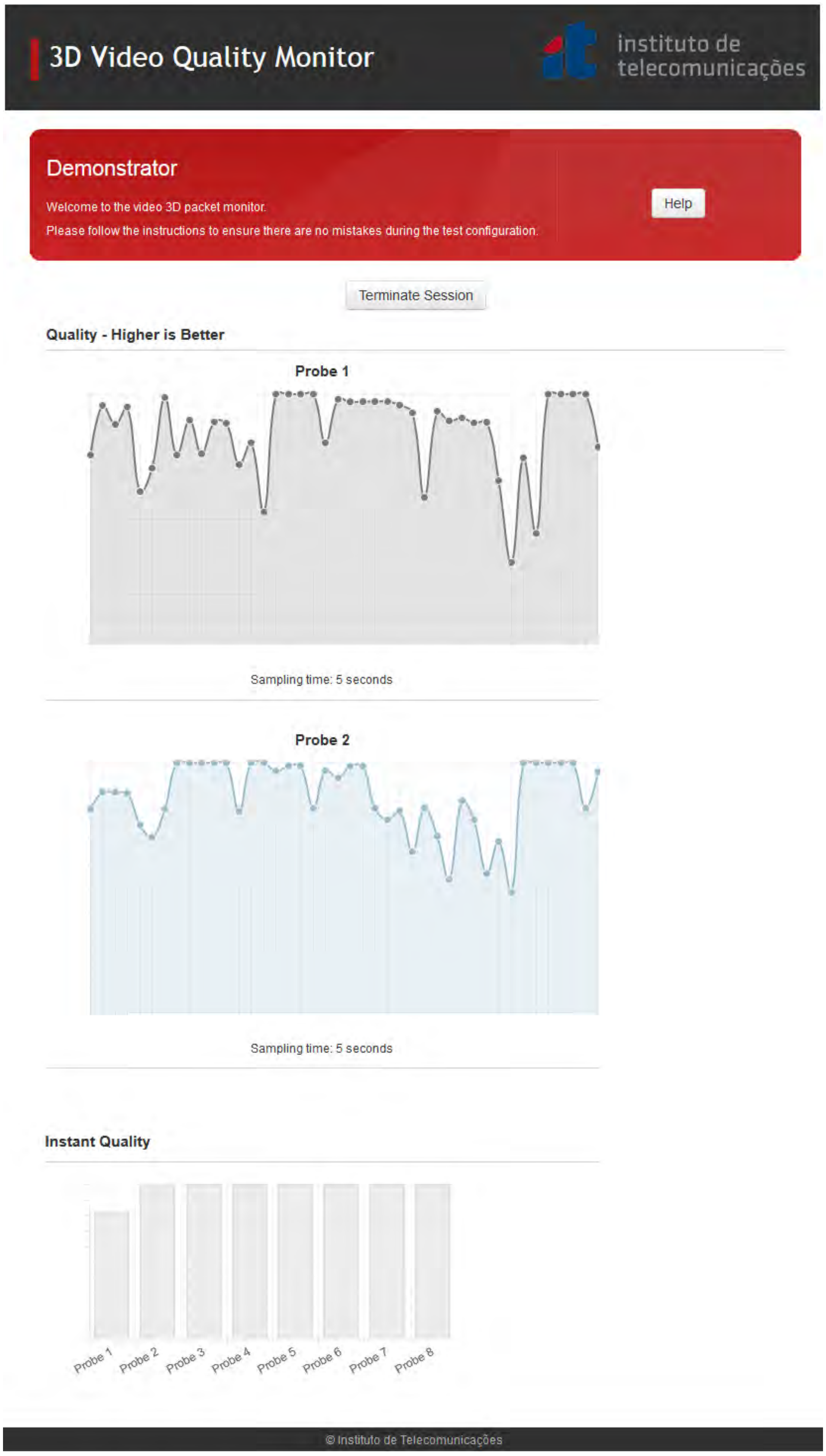

3D Video Quality Monitor
instituto de telecomunicações
Demonstrator
Welcome to the video 3D packet monitor.
Please follow the instructions to ensure there are no mistakes during the test configuration.
Help
Terminate Session
Quality - Higher is Better
Probe 1
Sampling time: 5 seconds
Probe 2
Sampling time: 5 seconds
Instant Quality
Probe 1
Probe 2
Probe 3
Probe 4
Probe 5
Probe 6
Probe 7
Probe 8
© Instituto de Telecomunicações

Figure D.1: Graphical User Interface Layout

Appendix E

Function implementing the Gilbert-Elliot model

```
import random
#variables to change

BL = 3
PLR = 0.05

######################


maxBL = 7

p = 1/(BL*((1/PLR)-1))  #p calculation
r= 1.000/BL             #q calculation

print p, r

burst = 0
loss = False

while 1:

    if loss == 0:
        burst = 0
        print "LOSS:" , loss #show state
        loss = (random.random() < p )

    elif loss == 1:
        burst = burst + 1

        if burst <= maxBL :
            print "LOSS:" , loss #show state
            loss = (random.random() < (1-r) )
        else:
            loss = False
    else:
        print "ERROR"
```

Code Example E.1: Example of Gilbert-Elliot model implementation

Appendix F

Pictures ilustrating video degradation

This Annex shows still images taken from video being transmitted for several packet loss levels, as explained in Section 6.4. Some less noticeable impairments are signalled.

F.1 Dog sequence

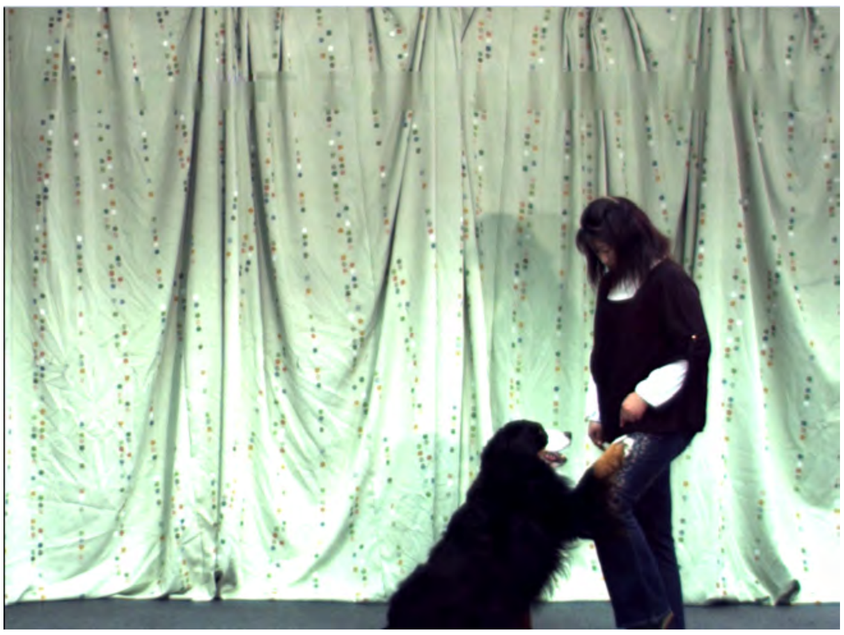


Figure F.1: Still image from DOG sequence, for 0.1% UDP packet losses

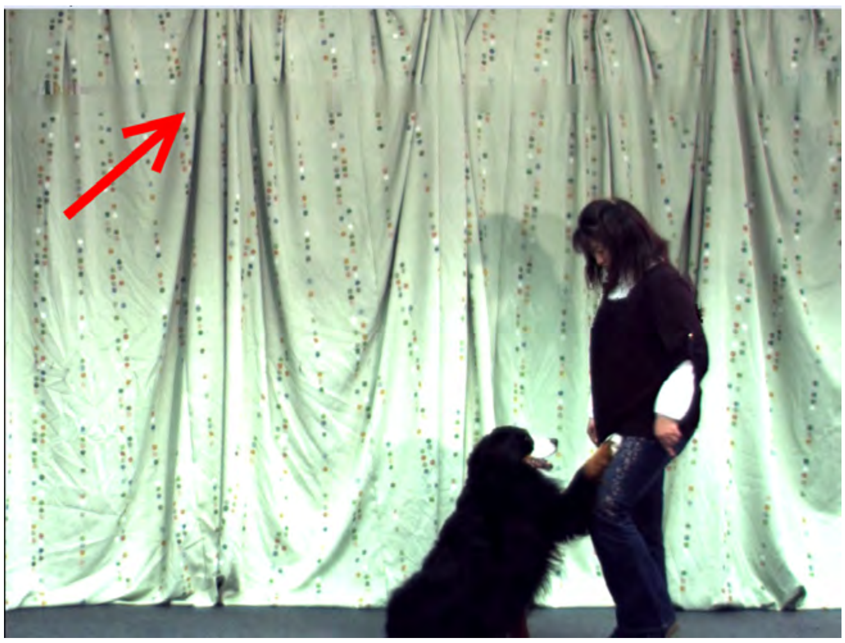


Figure F.2: Still image from DOG sequence, for 0.5% UDP packet losses

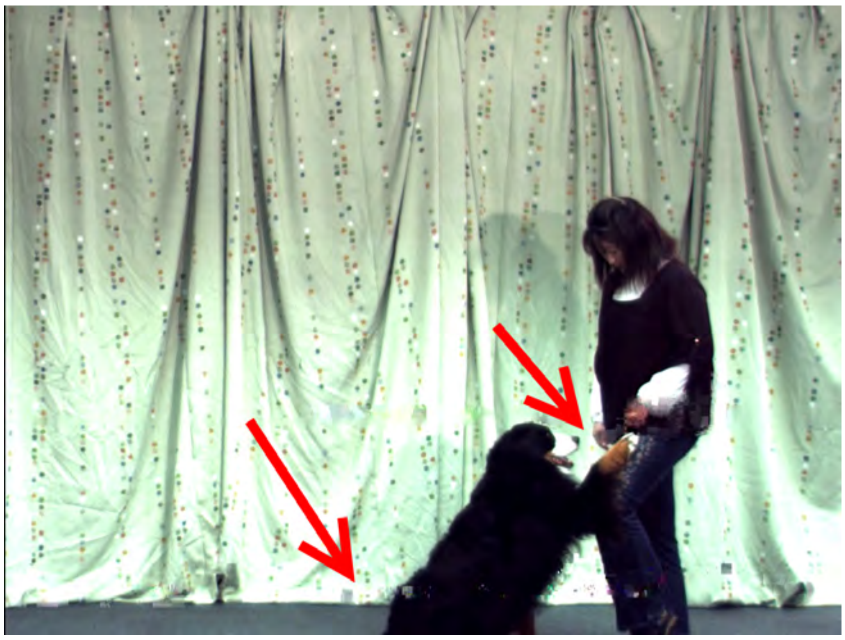


Figure F.3: Still image from DOG sequence, for 1% UDP packet losses

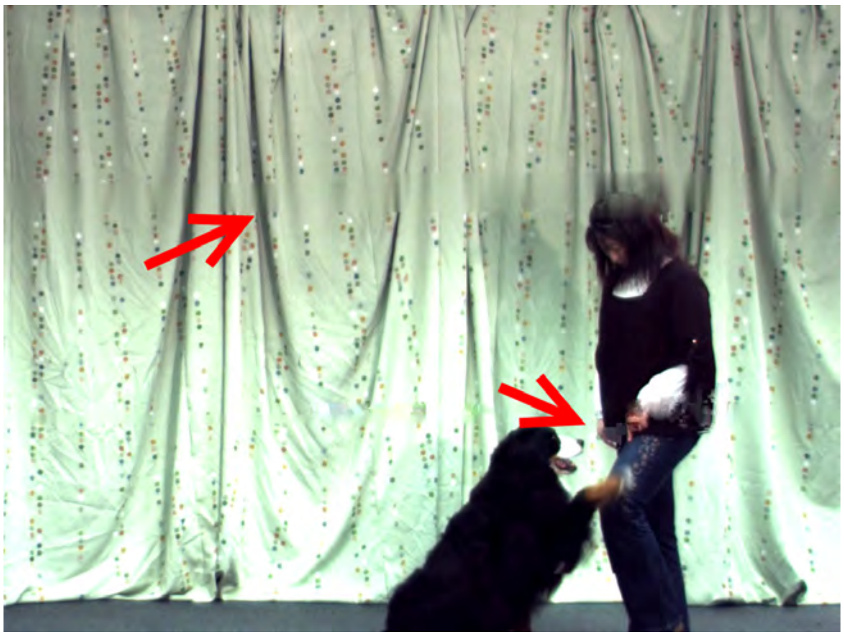


Figure F.4: Still image from DOG sequence, for 2% UDP packet losses

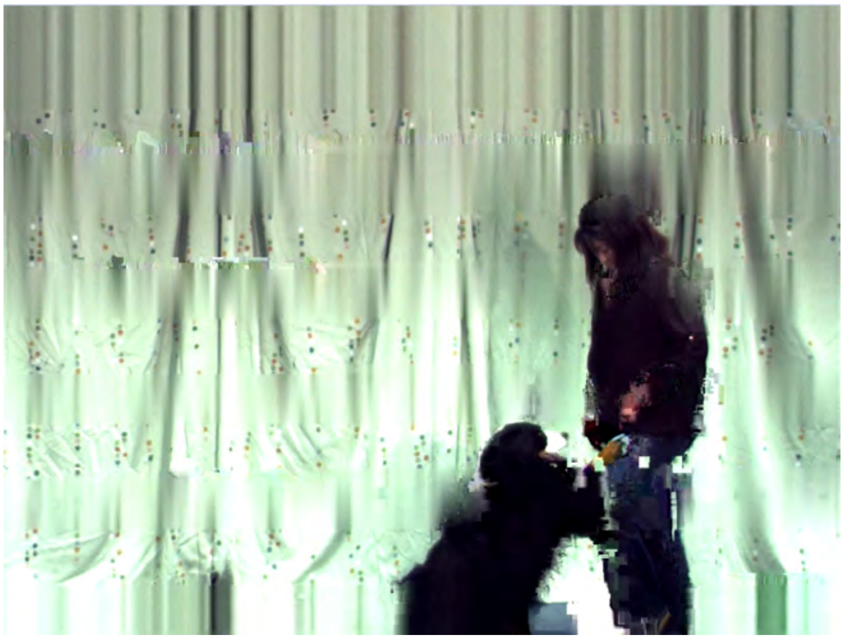


Figure F.5: Still image from DOG sequence, for 4% UDP packet losses

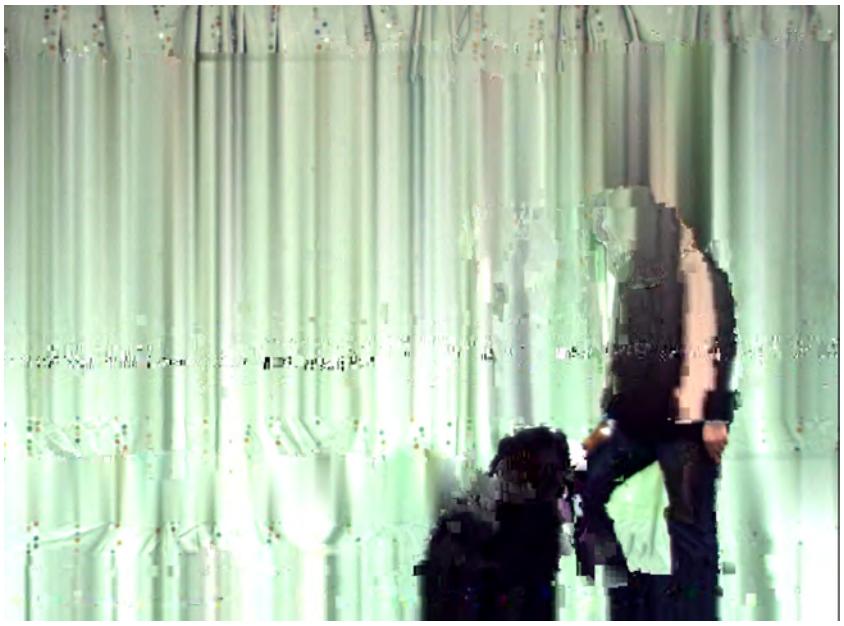


Figure F.6: Still image from DOG sequence, for 8% UDP packet losses

F.2 Balloons sequence

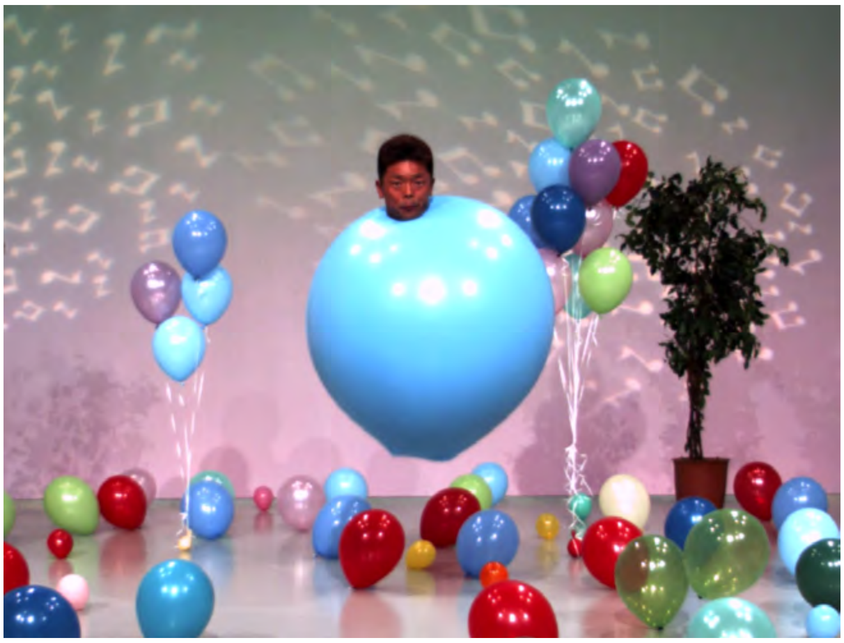


Figure F.7: Still image from Balloons sequence, for 0.1% UDP packet losses

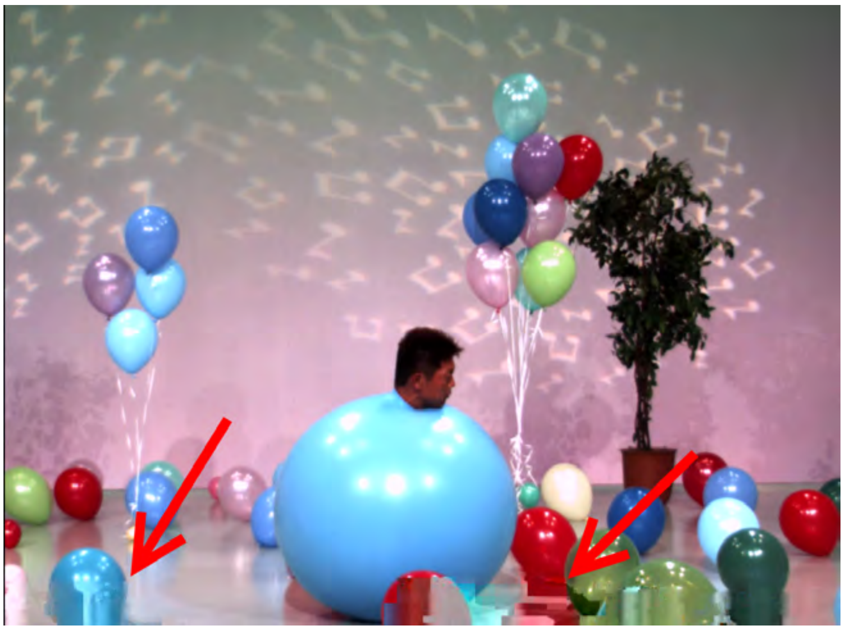


Figure F.8: Still image from Balloons sequence, for 0.5% UDP packet losses

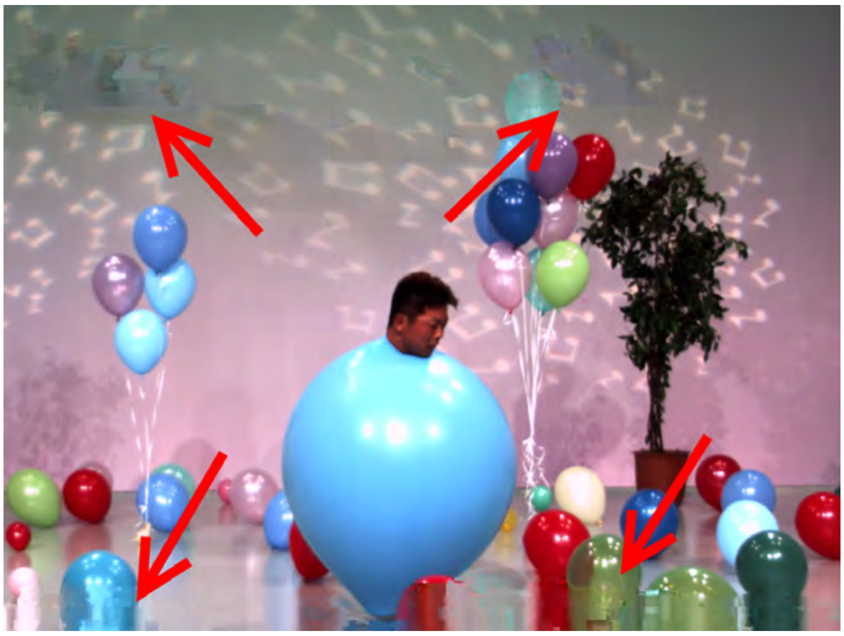


Figure F.9: Still image from Balloons sequence, for 1% UDP packet losses

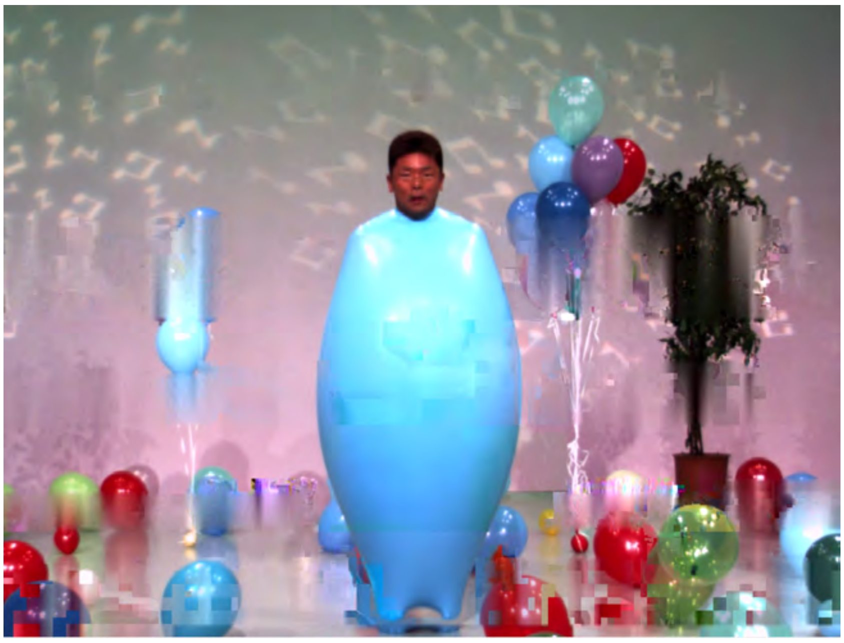


Figure F.10: Still image from Balloons sequence, for 2% UDP packet losses

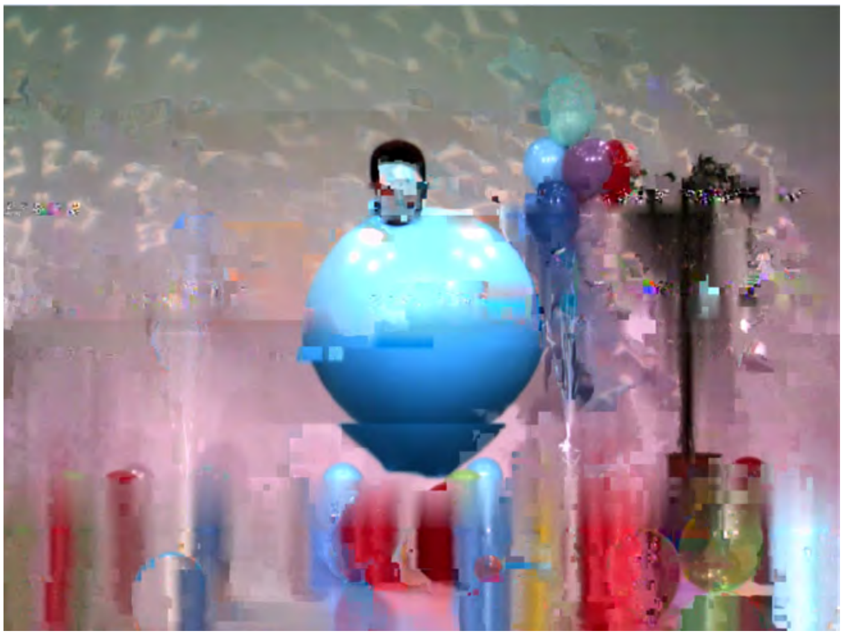


Figure F.11: Still image from Balloons sequence, for 4% UDP packet losses

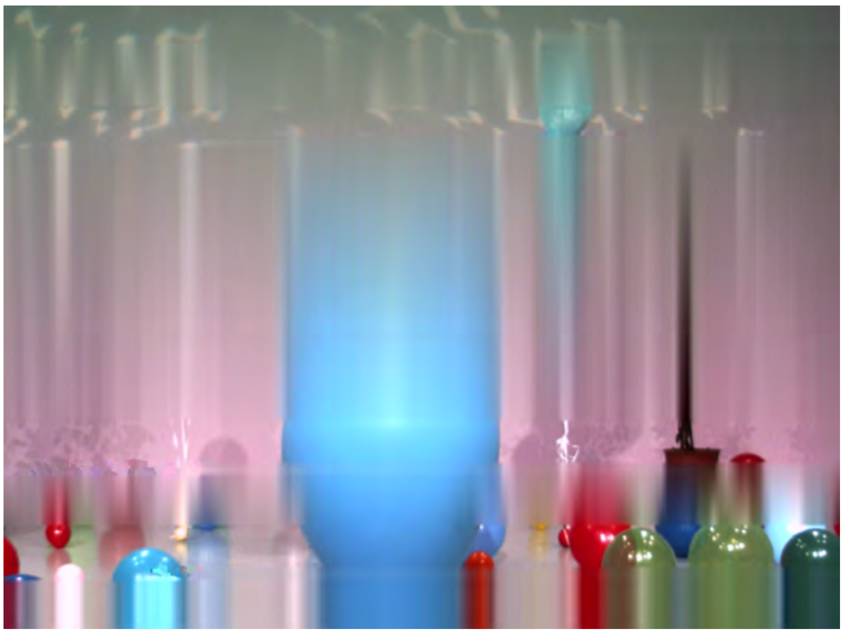


Figure F.12: Still image from Balloons sequence, for 8% UDP packet losses

F.3 Bullinger sequence

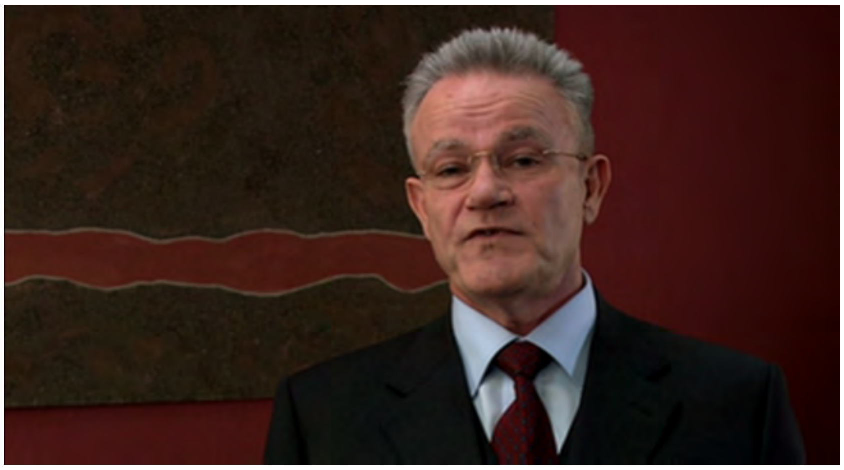


Figure F.13: Still image from Bullinger sequence, for 0.1% UDP packet losses

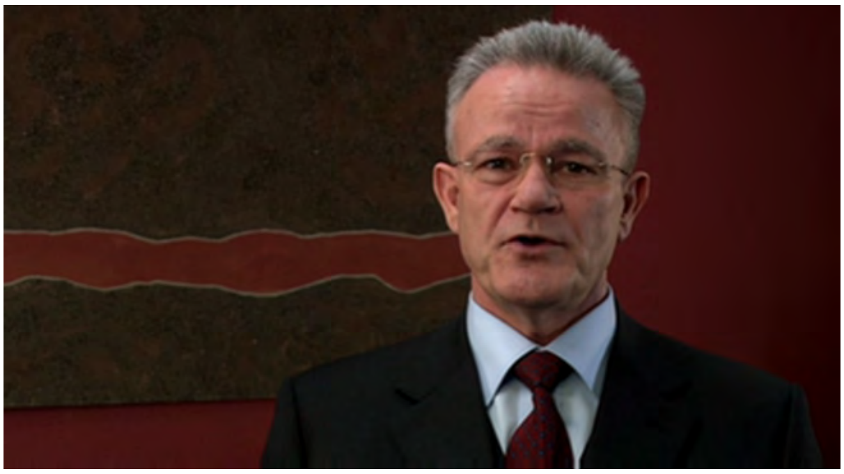


Figure F.14: Still image from Bullinger sequence, for 0.5% UDP packet losses

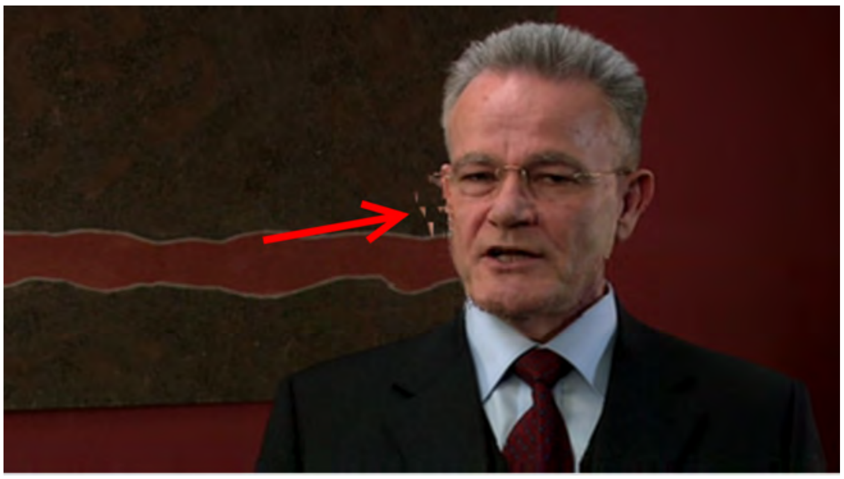


Figure F.15: Still image from Bullinger sequence, for 1% UDP packet losses

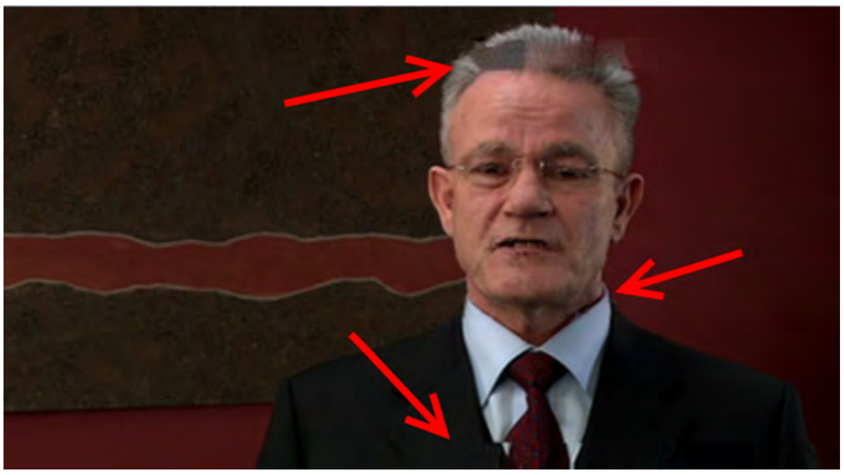


Figure F.16: Still image from Bullinger sequence, for 2% UDP packet losses

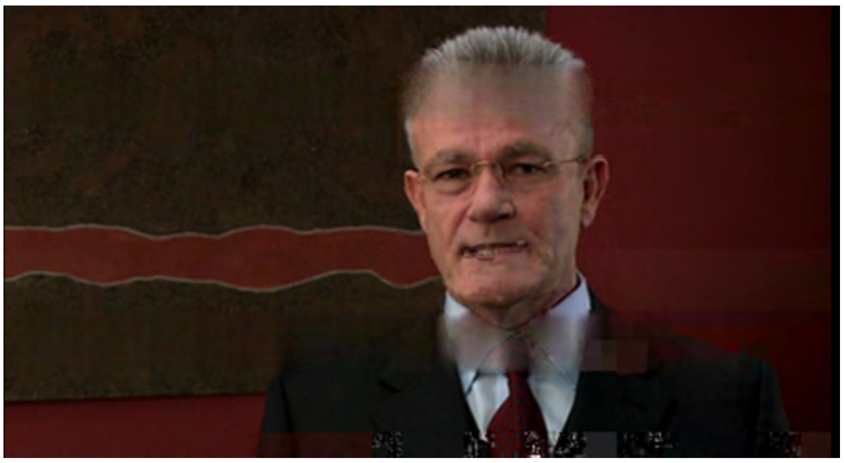


Figure F.17: Still image from Bullinger sequence, for 4% UDP packet losses

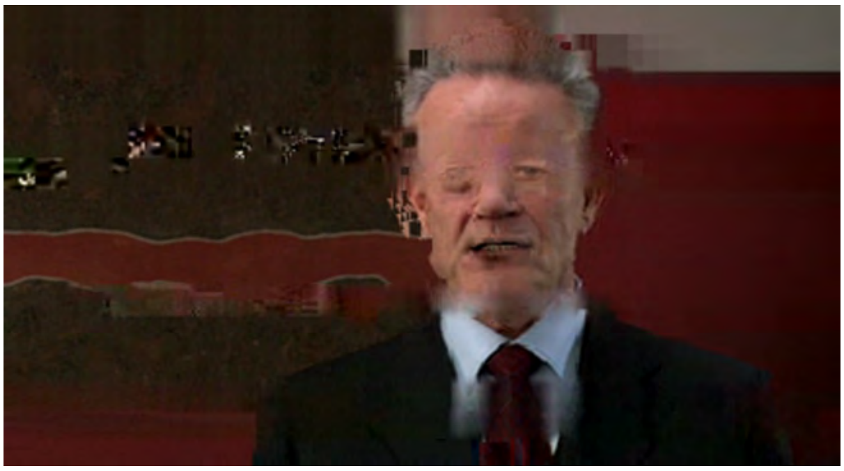


Figure F.18: Still image from Bullinger sequence, for 8% UDP packet losses

F.4 ESTG Bus Stop sequence

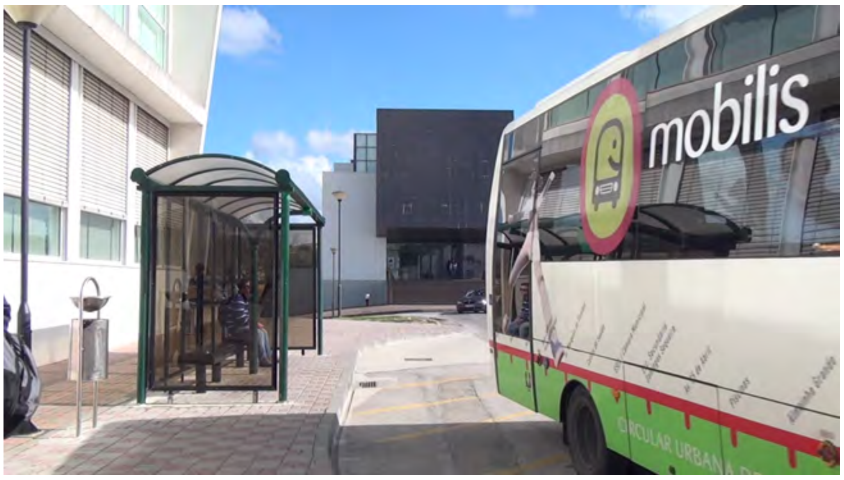


Figure F.19: Still image from ESTG Bus Stop sequence, for 0.1% UDP packet losses

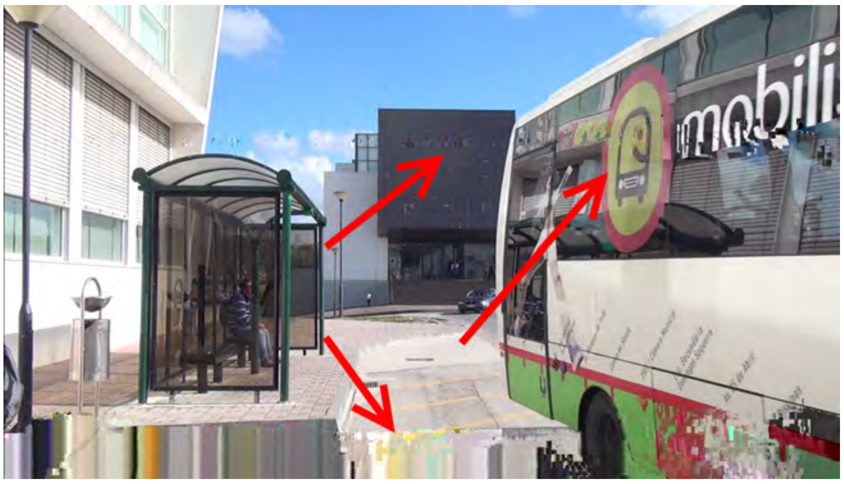


Figure F.20: Still image from ESTG Bus Stop sequence, for 0.5% UDP packet losses

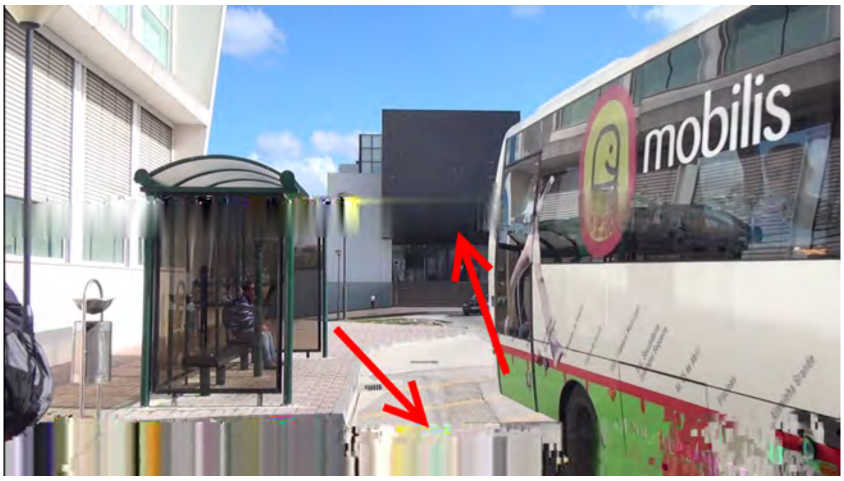


Figure F.21: Still image from ESTG Bus Stop sequence, for 1% UDP packet losses

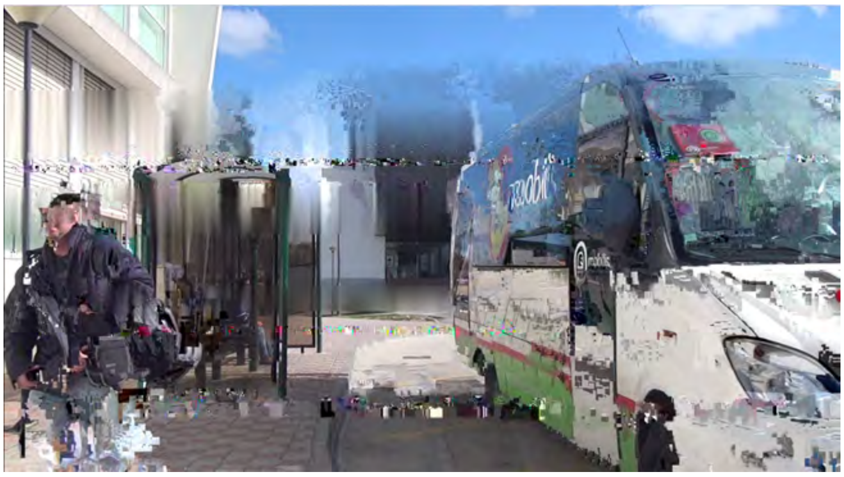


Figure F.22: Still image from ESTG Bus Stop sequence, for 2% UDP packet losses

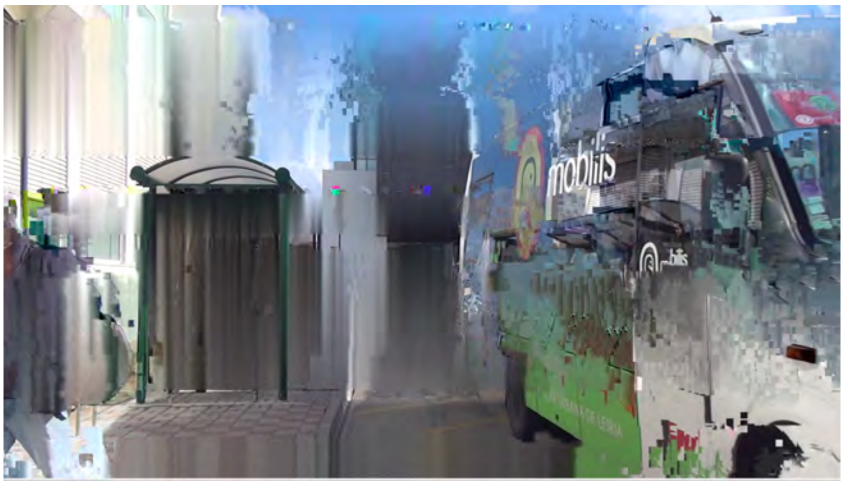


Figure F.23: Still image from ESTG Bus Stop sequence, for 4% UDP packet losses

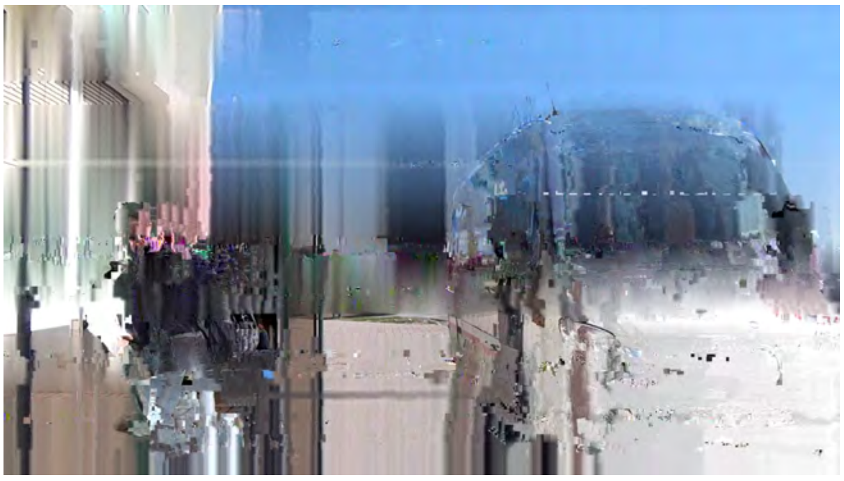


Figure F.24: Still image from ESTG Bus Stop sequence, for 8% UDP packet losses