Universitat Politecnica de Catalunya - EPSC
Technical University of Vienna - INTHFT
Institute of Communications and Radio-Frequency Engineering

Master of Telecommunications

Robust Error Detection Methods for H.264/AVC Videos

by

Eva Rodriguez Rodriguez

Supervisors
Markus RUPP, Univ.Prof.
Olivia NEMETHOVA, Dr.techn.
Luca SUPERIORI, Dipl.-Ing.

Vienna, 2008

Contents

Acknowledgements

A mi familia,
por el apoyo de todos estos años

To Luca,
for the priceless daily support

To Olivia,
for the motivating conversations
and the effective "remote corrections"

To Prof. Markus Rupp,
for providing all the facilities and an
excellent working environment at the Institute

Finally, "a la gente del lab",
for helping me to laugh when frustration arose

Chapter 1

Introduction

1.1 Motivation

Nowadays, there is an increasing number of services and transmission media, like Cable Modem, xDSL or UMTS, that provide lower data rates when compared with broadcast channels like HDTV or DVB-T. These, together with the advances in processing power and memory, and in video coding technology, claim for an efficient video compression representation with increased coding efficiency and robustness to support different network environments [1][2][3]. As a response to these needs, the International Telecommunication Union (ITU) together with the ISO/IEC (International Electrotechnical Commission) created the Joint Video Team (JVT) of experts, that defined the H.264 or Advanced Video Coding standard [1], as an evolution of previous standards (H.263 [4] and MPEG-4 part 2 [5]). Therefore, H.264/AVC main features are, on the one hand to provide high compression for a wide range of applications: from videoconferencing to Internet streaming; and on the other hand to bring flexibility over different network environments.

The 3rd generation of mobile systems is mainly focused on enabling multimedia services such as video streaming, video call and conferencing. In order to achieve this, the Universal Mobile Telecommunications System (UMTS), is the standard that has been developed by the 3rd Generation Partnership Project (3GPP) in Europe, including the baseline profile of H.264/AVC in the specification. With the union of both technologies a great improvement on video transmission over mobile networks, and even modification of the user habits towards the use of the mobile phone is expected. Nevertheless, video transmission has always been related to wired networks and unfortunately the migration to wireless networks is not as easy as it seems.

In real time applications the delay is a critical constraint. Usually, transmission

protocols without delivery warranties, like the User Network Protocol (UDP) for IP based networks, are used. This works under the assumption that in real time applications dropped packets are preferable to delayed packets. Moreover, in UMTS the network needs to be treated in a different way, thus the wireless channel is a prone error channel due to its high time variance. Typically, when transmitting video, the receiver checks whether the information packet is corrupted (by means of a checksum) or if its temporal mark exceeds the specified delay, and if affirmative, the whole packet is discarded. Nevertheless, this approach is suboptimal, due to the fact that perhaps the video information is not damaged and could still be used.

Instead, residual redundancy on the video stream can be used to locate the errors in the corrupted packet, increasing the granularity of the typical upper-layer checksum error detection. Based on this, the amount of information previous to the error detection can be decoded as usually. In [2] several error detection methods are presented and evaluated through simulation.

The aim of this thesis is to combine some of the more effective methods proposed in [2], and define a common set of simulation to exhaustively compare they performance. Concretely, Syntax check, Watermarking and Checksum schemes have been reformulated, combined and simulated. The rest of this first chapter is devoted to explain the basics of the H.264/AVC, the problems encountered when it is used in a wireless environment, the parameters used in simulation and the performance indicators used to evaluate the obtained results.

1.2 H.264/AVC standard

H.264/AVC [1] is the newest standard for video coding. Its last version was published on November 2007 by the ITU-T as Recommendation H.264 [1] and for the ISO/IEC as International Standard ISO/IEC 14 496-10 (MPEG-4 part 10) in December 2005 [6]. Compared to its predecessors, ITU-T H.261, H.262 (MPEG-2), and H.263, H.264/AVC has been created in response to the growing need for higher compression of moving pictures, comprising applications such as digital storage media, television broadcasting, Internet streaming and real-time audiovisual communication. It has also been designed to enable the use of the coded video representation in a flexible manner for a wide variety of network environments. To achieve these, the design of the system is split into the Video Coding Layer (VCL), that represents the video content in an efficient way, and the Network Abstraction Layer (NAL), responsible of the format of the VCL representation and that provides header information to have a proper interaction with different transport layers and/or storage media. Figure 1.1 shows the mentioned structure. After the encoding process, the video content is

grouped in NALUs (NAL Units) to be sent over the determined network.

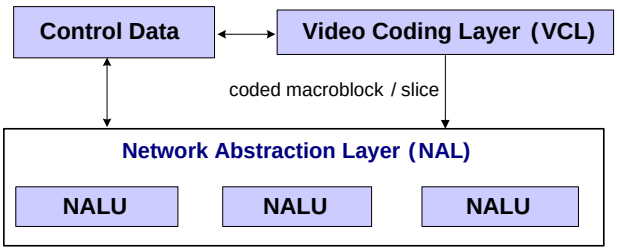


Figure 1.1: H.264/AVC structure. Coexistence of Video Coding Layer (VCL) and Network Abstraction Layer (NAL)

Although the standard just specifies the decoding process, the encoding process works in a similar way. The steps followed by the H.264/AVC encoder are described in Figure 1.2. To understand the process, the next subsections are devoted to explain, at a high level, the different parts of this encoding process.

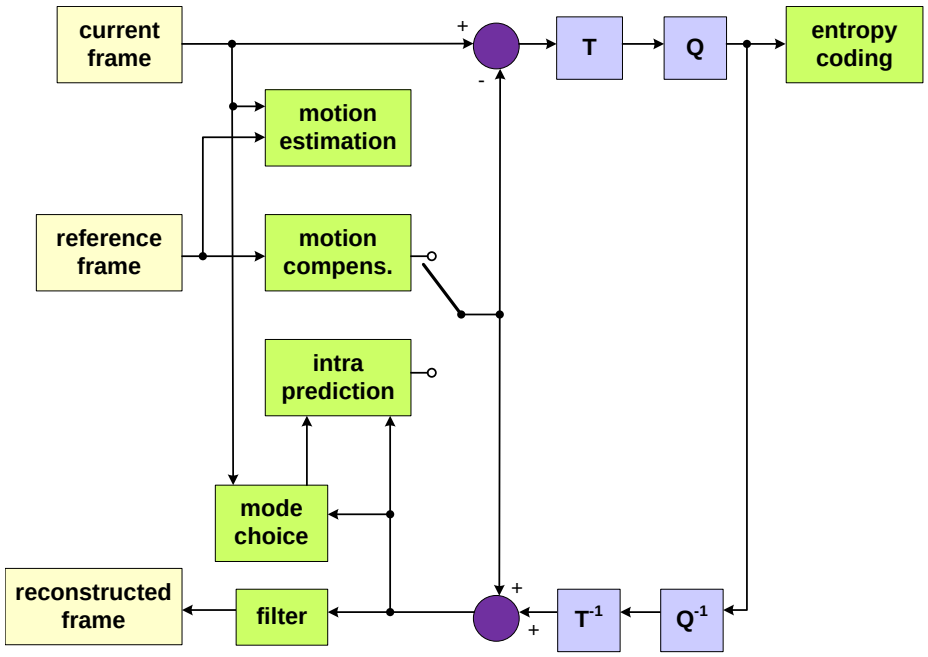


Figure 1.2: H.264/AVC encoder structure

1.2.1 Video sampling

To obtain a video stream, video cameras sample in the temporal and spatial domain. In the temporal domain, the result is a set of pictures per second. These pictures are called frames, thus the frame rate (frequency) of a video is expressed as the number

Table 1.1: Common picture resolutions for internet and mobile video applications

Name	*Resolution*	*Description*
VGA	640x480	Video Graphics Array
QVGA	320x240	Quarter Video Graphics Array
CIF	352x288	Common Intermediate Format
QCIF	176x144	Quarter Common Intermediate Format

of frames per second (f/s), or can also be expressed in Hertz (Hz). In the spatial domain, the sampling provides the the number of pixels in each of the frames (picture resolution). Depending on the type of picture, different types of pixels can be found. Pixels of intensity, for black and white videos, are scalar values; whereas pixels of color pictures are represented by coordinates in the relevant color space. Finally, the output of the digital video camera consists in a series of RGB (red-green-blue) frames, represented by $M \times N$ color component matrices. The samples of the three color matrices are typically represented by 8 or 16 bits. Depending on the number of pixels, different resolutions are obtained. Some of the more common resolutions for internet and mobile video are found in Table 1.1.

The input video to the H.264/AVC encoder is a YUV signal, created from an RGB source. As the human eye is less sensitive to color than to brightness, an initial reduction in size can be achieved by storing more luminance than color information. To create the YUV signal, the video signal is divided into luminance (denoted as Y, and called luma) and two color difference (chrominance) components, denoted as U and V (or Cb and Cr, respectively), called chroma. The most common way of subsampling, called 4:2:0 reduces the number of samples in both the horizontal and vertical dimensions by a factor of two, i.e., for four luma pixels there is only one blue and one red chroma pixel. Unfortunately, the rate of compression achieved at this point is not sufficient for much applications. The H.264/AVC encoder is able to highly increase the compression of the information. The steps followed to do this can be seen on Figure 1.3, and are explained in the following subsections.

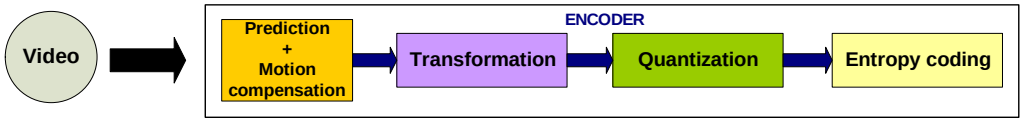


Figure 1.3: H.264/AVC encoder compression steps

1.2.2 Prediction and motion compensation

The first step of encoding is to divide the frame in a squared grid. Each subdivision is called macroblock (MB) and has 16x16 luma, and 8x8 of each chroma samples, respectively (prediction in H.264/AVC also performs for non squared blocks of 8x16 pixels, or smaller sub-blocks). Subsets of macroblocks that can be decoded independently are organized in slices. There are different types of frames, depending on how they are predicted:

- **Intra-coded frames:** encoded either using spatial prediction, just using the information contained in the picture itself, or non prediction at all. Intra-coded frames are shortly called *I frames*.
- **Inter-coded frames:** encoded using temporal prediction, using the information contained in previous encoded pictures.

Typically, the first picture of a video sequence is intra-coded. Each macroblock in an intra- coded frame is predicted using spatially neighboring samples of previously coded macroblocks (but macroblocks without encoded neighbours, like the first macroblock, that are directly encoded). The encoder decides which and how neighboring samples are used for intra prediction, and the selected mode is then signalized within the stream. A group of pictures (GOP) can be configured at the encoder, and determines the periodicity of intra-coded frames, i.e., in a GOP of 10, every 9 inter-coded frames, 1 frame is intra-coded.

Different inter-coded frame types are defined, depending on how the prediction is performed. In this work, the type of frames used are I and P.

- **Predictive (P) frames:** coded as a prediction of the last I or P image.
- **Bi-predictive (B) frames:** coded as a prediction of last I/P image and the next I/P.
- **Switching P (SP) slice:** provides efficient switching between different precoded pictures.
- **Switching I (SI) slice:** allows exact match of a macroblock in an SP slice.

For the inter-coded frame process a motion estimation of each block is performed by searching the best matching region from the previous frame, and taking it as a prediction of the encoded block. The comparison is made pixel by pixel over the quantized and filtered block (thus preventing artifacts during the reconstruction process). The information about motion (motion vector) is also signalized, to have a proper reconstruction at the decoder. When there is no movement between pictures, or some static parts, no information needs to be sent. For this reason, a SKIP mode

allows for skipping a signalized number of P macroblocks. In this case, at the decoder, the corresponding macroblock information for the previous frame is taken.

After this procedure, the luma and chroma samples of each MB are spacially or temporally predicted. The differences between the actual macroblock samples and the prediction are called *prediction residuals*, this values are the ones encoded and further transmitted.

1.2.3 Transformation

In the obtained prediction residuals stream, a lot of correlation can still be found. To compress the information, a de-correlation needs to be performed. The Karhunen-Loève Transform (KLT) is optimal to this purpose, nevertheless the pre-training for the specific content as well as the computational complexity make it complicated to use it. The Discrete Cosinus Transform (DCT) is a very good approximation to it, and therefore, is the one used in H.264/AVC. Its basic property is that it is able to concentrate the energy of a given set of values in a region. After applying the DCT to the residuals of each sub-macroblock (each of the four 4x4 blocks within a MB), the resulting is a matrix of coefficients that represent the different frequencies. The DC coefficient corresponds to the lower frequency, it is located on the left upper corner of the matrix, and concentrates the most important information. The remaining coefficients are denoted as AC coefficients, and represent the high frequencies. The result of the transformation is strongly dependent on the content, i.e.; in pictures with a lot of edges, and fast movement, AC information is predominant; whereas pictures with smooth transitions and low movement have the energy mainly concentrated around the DC value. At this point, no compression has been performed yet, but information has been prepared for it.

1.2.4 Quantization and Entropy coding

At this step, for each macroblock, the obtained matrix of coefficients is quantized. Basically, and speaking at a high level, the values of the matrices are divided by the quantization parameter (QP), rounded to integer values, and later ordered in a vector following a zig-zag scan order. The quantization parameter can have a value from 0 to 52. The level of compression is directly dependent on the value of QP; higher QP values imply higher compression, but also higher distortion on the resulting video. This procedure is illustrated in Figure 1.4, and consists in a preparation for further entropy coding.

The entropy coding in H.264/AVC can be performed in two ways:

- **Contex-Adaptive Binary Arithmetic Coding (CABAC):** provides better coding efficiency than CAVLC. Even though, due to its complexity and the

fact that it is not efficient for small block sizes, it is not supported by all profiles.

- **Context-Adaptive Variable-Lenght Coding (CAVLC):** does not perform as good as CABAC, but it is supported by all profiles. At the moment just the baseline profile is accepted for the 3GPP, therefore this work is focused on it and just refers to the CAVLC.

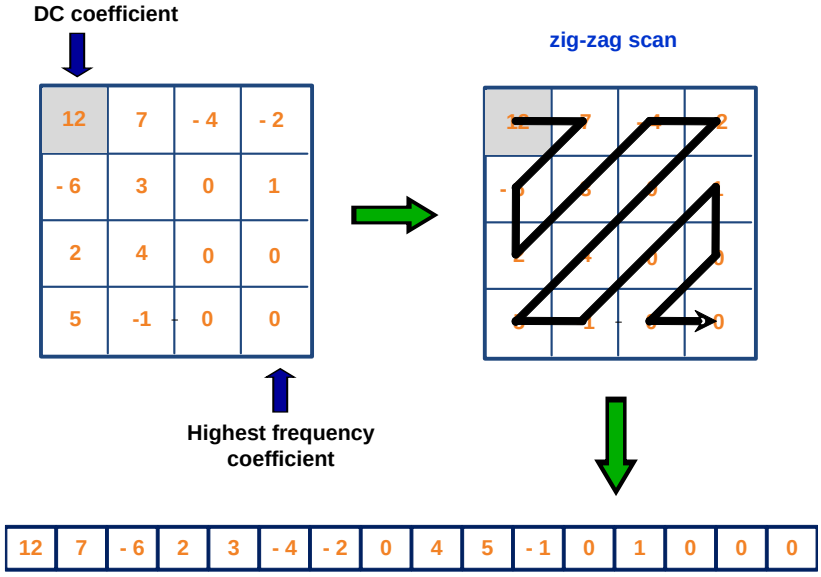


Figure 1.4: Zig-zag scan on a sub-macroblock and vector storage of the coefficients

The basic idea of CAVLC is to use Variable Length Codes (VLC), with a universal exponential Golomb code (exp-Golomb). Nevertheless, an important point, that will be referred in further sections, is the encoding of the transformed residuals. The resulting vector after the zig-zag scan has all the important values at the first positions, and for high frequencies, typically zero coefficients and trailing ones are founded at the last positions. These values, the number of zero coefficients and trailing ones (up to three of them, if more trailing ones are founded, those are encoded as normal coefficients), are encoded specially, chosing one of four look-up tables. The sign of the trailing ones is encoded separately. Then the rest of nonzero coefficients are encoded choosing one of the seven VLC tables, context adaptively selected. After this point, the compression is finished. The resulting H.264/AVC video stream, is then transformed in the NAL.

1.3 Network abstraction layer (NAL)

Once the video is encoded, in order to be efficiently sent through the network, the NAL is defined. Video data is encapsulated in NAL units (NALU). The format of a NALU is shown in Figure 1.5. The first byte is the header, and the rest is payload. On this first byte, the first bit is always a zero; the second and thirds bits are the NRI (NAL Reference Identification) that signalizes the importance of the NAL unit for reconstruction purposes. The remaining bits determine the type of NALU, and have up to 32 different values, that depend on the type of data contained into the NALU. NALUs can be encapsulated into different transport protocols and file formats, this work is related to NALUs encapsulated into RTP (Real-Time Protocol) packets.

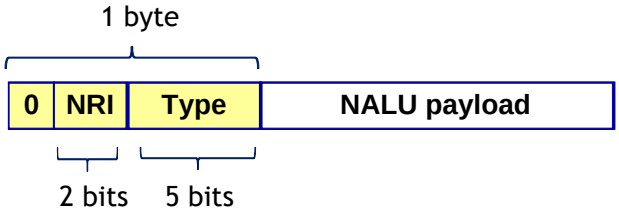


Figure 1.5: NAL Unit packet format

1.4 Video streaming over wireless networks

Typically, video streaming has been used in wired networks, in which bandwidth is abundant and the transmission channel provides very low bit error rates. Nevertheless, the wireless channel is a prone error channel, and because of that, the received video can be damaged after a transmission. On the other hand, both the compression scheme in H.264/AVC, and the transmission system, do not help to improve the effect of the errors, but the opposite.

Because of the encoding system in H.264/AVC a single bit error can cause error propagation, thus affecting more than its actual MB, but slices, and frames. The three possible sources of error propagation are the following:

- **Spatial prediction:** as the encoded MBs require information from their spatially neighbors, if those are wrong, also the reconstructed MB will be distorted.
- **Temporal prediction:** because of the temporal prediction, if an error occurs in a frame,the following frames using the erroneous frame as reference will be affected by the error.
- **Entropy coding:** as variable length codes are used, an error in a codeword can have impact in the following codewords, if the codeword boundaries are

determined incorrectly. Then desynchronization when reading the next words appears, with the decoder being unable to distinguish among codewords.

Another thing that affects the amount of information lost when errors, is the transmission system. Typically, NALUs are encapsulated on RTP, over UDP and IP packets. The use of UDP makes sense in real-time applications because it provides less overhead than TCP and less delay, as it does not have acknowledgment packets. Usually, error detection is performed at the UDP level, by the correctness of the checksum field, computed over the whole UDP packet. When this value is wrong, the whole packet is discarded. Nevertheless, this approach is not optimal. In case the error is located within the NALU, the information located before the error is not damaged, and thus could be recovered, decreasing the amount of lost information. As illustrated in Figure 1.6.

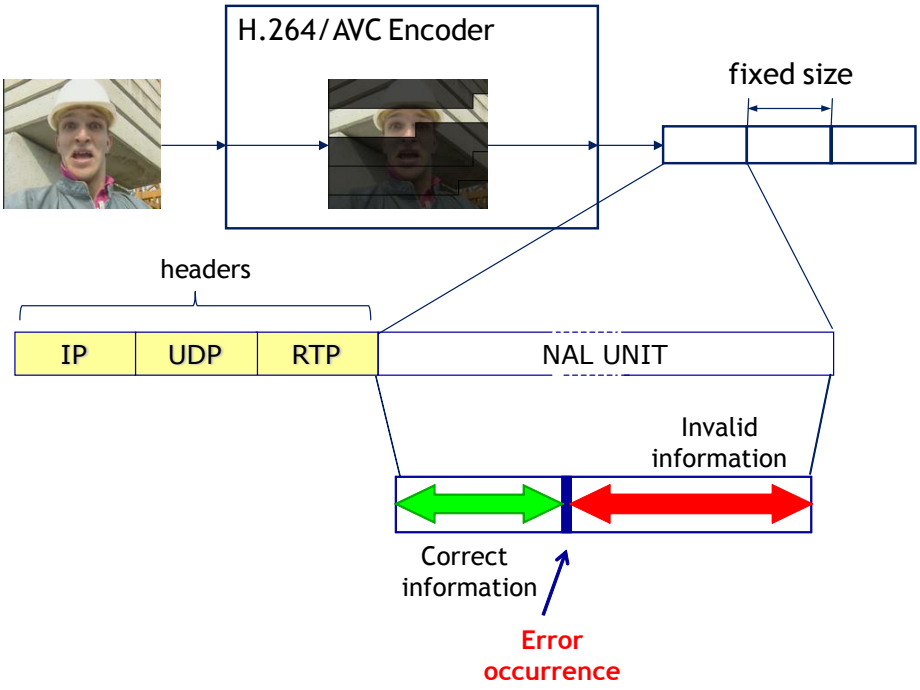


Figure 1.6: Stream structure with protocols and uncorruped information when having errors within the video payload

To improve the degradation caused by errors at the receiver three steps can be followed. *Error detection*, trying to detect errors on the NALU and extracting the possible correct content. *Error concealment*, determines the error visibility in the decoded stream, i.e.; even though the error cannot be corrected, the method tries to reconstruct the image using the information it has. Finally, *Cross-layer dessign*, that proposes improvement for H.264/AVC video transmission over UMTS, by means

of sharing the error information between layers. The presented thesis is focused on error detection methods, therefore neither error concealment nor cross-layer design strategies will be explained.

1.5 Overview of previous error detection methods and new approaches

Error detection methods are designed to efficiently locate errors, in this case, within the video stream. Hence, the more errors found and the faster, the better performance the method has. Previous work on error detection methods as well as references to alternative error detection methods can be found in [2]. Among those, the methods studied here are the proposed Syntax check on [7] and Watermarking [8].

1.5.1 Principles of Syntax check

Syntax check takes advantage of the residual redundancy existing in the video stream after encoding. Concretely, exploiting the variable length code codewords, range and significance of the H.264/AVC information elements. As briefly explained previously in Section 1.2.4, due to the use of CAVLC, a bit error can easily cause desynchronization making impossible to distinguish the boundaries of the following codewords. When this happens, it may even be impossible to decode the video stream, and if possible, visual impairments are recognized at the decoded video.

Regarding the results obtained in [2] and [7], Syntax check seems to be a good method when detecting errors. Nevertheless, its performance may be improved when combined with other strategies. In previous work, it was shown that typical detection delay for I frames was one or two MBs. Considering those results, an easy way to improve quality at the receiver might be a direct previous concealment of one or two macroblocks before error occurrence. This approach is implemented and tested in Chapter 2. Moreover, as Syntax check provides a low implementation complexity, all the different methods that are proposed in this work are implemented on top of Syntax check.

1.5.2 Principles of Watermarking

Watermarking consists in adding a hidden redundancy into the video at the encoder side, in order to locate errors at the receiver. When a video sequence is watermarked, different information values can be changed: pixel values, coefficient values, residuals, etc. A Force Even Watermarking (FEW) scheme was first introduced for H.263 in [9]. In this work, a fragile watermark is forced onto quantized DCT coefficients at the encoder; the decoder checks the correctness of this mark and is able to detect errors at the MB level. The main problem of this approach though, is that even when invisible Watermarking is used, an initial distortion is introduced in the video

sequence. Lately, the watermarking scheme was introduced also in [8], applying FEW to H.264/AVC and proposing a Relation Based Watermarking (RBW) approach.

The results obtained in both approaches showed great improvement in the final quality obtained at the receiver. Taking these into account, this work proposes a combination of Syntax check together with different Watermarking schemes, to see the grade of enhancement that can be achieved when both methods cooperate. Moreover, different scenario conditions are simulated, applying more realistic error patterns to the video transmission. Chapter 3 describes the implementation of the method.

1.5.3 Principles of Checksum

The basic idea of checksum consists in adding redundancy bits to a content in order to be able to detect errors. Typically, some function is calculated over the whole content to protect, or over part of it; this result is also transmitted to the receiver, that performs the same operation over the content, if the results differ, an error is assumed. Checksums can be implemented in several ways, by changing the amount of content to protect, the operation performed over it, and the in or out-band transmission. Depending on these, the error detection probability is higher or lower. Unfortunately, the overhead introduced by the additional checksum information is strongly related with the good performance of the detection, i.e. the amount of information sent by a checksum that applies over a whole frame is less than applying a checksum per macroblock, nevertheless, with the second approach the granularity of the detection is higher.

A method applying parity bits over a group of MBs in H.264/AVC videos is already proposed in [10]. In this work, based on the results obtained in Chapter 3, an alternative Checksum scheme, considering the protection of critical information elements is proposed, implemented together with Syntax check and evaluated with simulation in Chapter 4.

A comparison of the methods, by comparing the simulation results, can be found on Section 3.3 for WM and on Section 4.3 for Checksum. To sum up, Figure 1.7 shows at which level of the encoding/decoding scheme the different methods are applied. Syntax check works on the bitstream domain and is applied directly at the entropy decoding. Watermarking works adding redundancy at the pixel domain and is applied both in the encoder and decoder, after quantization and before inverse quantization, respectively. Finally, Checksum is applied after the whole encoding process, sent out-of-band, and checked just when the decoding starts.

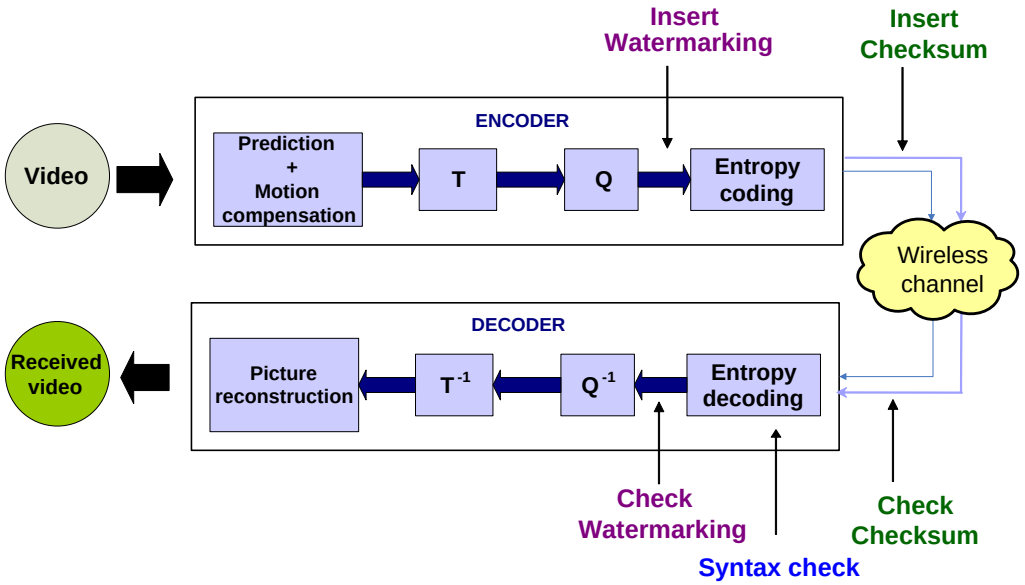


Figure 1.7: Error detection methods studied. Position of application at the encoder / decoder system. Syntax check, Watermarking and Checksum

1.6 Simulation, software, and performance indicators

All the methods proposed in this work have been first programmed, then simulated with predefined chosen parameters, and finally the results have been evaluated. The exception is Syntax check, that has just been simulated because it had already been successfully programmed in [7]. The Force Even Watermarking implemented in [8] had to be programed together with the Syntax check, thus just the programing of the combination was considered at first. Nevertheless, some modifications had to be done over that watermarking approach, and finally a new FEW was reprogramed together with Syntax check.

1.6.1 Reference software and simulation settings

The different error detection methods for H.264 have been programed by modifying the Joint Model (JM) reference software [11]. The software is distributed for free, and includes both the encoder and the decoder, compliant with the H.264/AVC. The different settings are passed to the program from the command line and/or the configuration files `encoder.cfg` and `decoder.cfg`.

In order to be able to compare the simulation results in a proper way, common simulation scenarios need to be defined. In this work, the simulations are performed with the parameters described in Table 1.2. Table 1.3 shows the different Bit Error Rates simulated, and the number of simulations performed for each one. Errors are generated using a Binary Symmetric Channel (BSC) model.

Table 1.2: Common simulation parameters

Parameter	**Value**	**Description**
Video sequence	foreman.yuv (400 frames)	Common reference sequence used in video compression
Resolution	QCIF (176x144)	Applicable to mobile phone screens
Frame rate	30 f/s and 10 f/s	30 f/s applied to method tests, and 10 f/s to provide more realistic results in the methods comparative
Quantization parameter (QP)	20, 26, 30	26 used as standard. 20 and 30 for the rate-distortion tests
Group of pictures (GoP)	10	One I frame, every nine P frames
Packet size	800 bytes	The packet size correspond as well to the size of the slice, thus one packet containing a whole slice
Concealing method	Copy paste	Conceals erroneous MBs by copy-pasting the correspondent spatial MB of the previous uncorrupted frame.

Table 1.3: Bit Error Rates (BER) and number of simulations (N)

BER	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}
N	60	100	130	170	200

Note that, given a frame rate, simulations for each QP are performed, and for each QP, all Bit Error Rates are simulated. Moreover, the concealing method in this work is the most simple: a simple copy-paste of the last uncorrupted frame, for a given MB, is performed.

1.6.2 Performance indicators

In order to objectively evaluate the performance of the proposed error detection methods, performance indicators need to be defined. In the proposed work, the end-user distortion in transmission with errors, the error detection probability and the error detection delay are evaluated.

- **Distortion**

To measure the distortion caused by errors on the received video, the Mean Square Error (MSE) and the Peak Signal to Noise Ratio (PSNR) can be used. The MSE measures the distortion comprised within the nth video frame $\underline{\underline{F}}_n$ and the distortion-free reference frame $\underline{\underline{R}}_n$, as stated in:

$$\mathsf{MSE}[n] = \frac{1}{M \cdot N \cdot |\mathcal{C}|} \sum_{c \in \mathcal{C}} \sum_{i=1}^{N} \sum_{j=1}^{M} [\underline{\underline{F}}_n^{(n)}(i,j) - \underline{\underline{R}}_n^{(n)}(i,j)]^2, \tag{1.1}$$

where $N \times M$ corresponds to the size of the frame and $\mathcal{C}$ are the color components (for example, for YUV, $\mathcal{C} = \{Y, U, V\}$). Indexes i and j determine the individual elements of the color component matrix, per row and column respectively.

Nevertheless, for image distortion it is more common to use the PSNR, defined for one frame as:

$$\mathsf{PSNR}[n] = 10 \cdot \log_{10} \frac{(2^q - 1)^2}{\mathsf{MSE}[n]} [\mathsf{dB}], \tag{1.2}$$

where q represents the number of bits used to express the color component values. For a decoded video sequence, the JM typically provides the PSNR value for each color component (luminance, and two chrominance). As chrominance is usually smoother than luminance, and specially considering the Watermarking method (that changes luminance coefficient), just luminance distortion is compared in this work. In consequence, $\mathcal{C} = \{Y\}$. Luminance PSNR is denoted as Y-PSNR$[n]$.

Distortion can be studied by analyzing the evolution over time or over BER. The distortion over time is calculated by obtaining the average value of distortion for each frame, of the whole set of simulations. To obtain the final distortion values for each BER, the average over the whole sequence first, and over the whole batch of simulations later, is performed. The formally correct is to average over linear values, for this reason the given PSNR values per frame are transformed to MSE, using Equation 1.2 with $q = 8$. Having the results of MSE per frame, then averaging can be performed using Equation 1.3. The final averaged value can be then transformed to PSNR for comparison, by means of the Equation 1.4.

$$\overline{\mathsf{MSE}} = \frac{1}{F} \sum_{n=1}^{F} \mathsf{MSE}[n], \tag{1.3}$$

$$\mathsf{PSNR} = 10 \cdot \log_{10} \frac{(2^q - 1)^2}{\overline{\mathsf{MSE}}} [\mathsf{dB}], \tag{1.4}$$

- **Error detection probability**

Due to desynchronization, a single error on a video stream can cause several errors while decoding. To compute the number of error detections it is important to have a correspondance between the inserted error and the actual detection. To do this, simulations introducing one error per slice are performed. To compute this value the Equation 1.5 is used.

$$\text{Error detection probability} = \frac{\text{Number of errors detected by method}}{\text{Number of inserted errors}} \tag{1.5}$$

Detection error probabilities can be extracted for the different methods alone and for the combination of them with Syntax check. The focus in this work is the combination of methods, thus the error detection probability of Syntax check interacting with each of the methods is analyzed. When an error is detected, by one of the combined methods in the simulation, the concealment starts, and from this point the whole slice is concealed.

- **Error detection delay**

Because of the fact that errors can propagate, they can affect more than one MB. Thus errors are detected at the same MB where they occur, or some MBs later. As explained, once the error is located, and because of a possible desynchronization, the rest of the MBs of the slice are concealed. This applies specifically to this work, because the payload of one NALU is a slice.

To compare the performance of the different methods in terms of error detection delay, the number of macroblocks from the error occurrence until the error detection is stored. In order to compare the methods, a cumulative distribution function (CDF)of the detection delay is used, and computed as a cumulative histogram M_i as:

$$M_i = \sum_{j=1}^{i} m_j, \tag{1.6}$$

where given a normalized histogram m_i, M_i counts the accumulated amount of times that a certain detection delay has occurred. For example, considering the normalized vector of detection delays $V = [0.75, 0.15, 0.10]$ and assuming detection delay values are within zero to two macroblocks, the resulting cumulative distribution function is $M = [0.75, 0.90, 1]$; meaning that 75% of the detections are on the actual macroblock where the error ocurrs, 90% within the actual or first MB, and 100% within the two first MBs.

Chapter 2

Direct concealment of previous MBs with Syntax check

2.1 Introduction to Syntax check principles

At the end of the encoding process, the obtained H.264/AVC bitstream has a determined structure, formed from different information elements. Considering this, invalid bit structures (due to errors) can be located by exploiting the codewords, range and significance of the H.264/AVC information elements. An approach based on this concept for H.263 was presented in [9]; later, a proposal for error detection on H.264/AVC based on syntax analysis was presented in [7]. The examined bitflows in [7], refer to encoding sequences using QCIF, file mode RTP, and the JM codec in baseline profile. The work presented in this thesis completely relies on that implementation.

Syntax check for H.264/AVC, also encounters the limitations already present on the syntax analysis for H.263, that depends on the encoding scheme itself. Entropy coding and a lack of synchronization words between macroblocks cause the errors to propagate until the end of slice, if not detected previously. As most codewords are entropy encoded and can be decoded without the need of a look-up table, a contextual analysis of each information element is needed in order to detect errors.

The structure of the VCL NALU payload is composed by the Slice Header (SH) and coded macroblocks. The slice header contains basic information about the slice, thus errors affecting the header might make it impossible to decode the entire slice. The H.264/AVC decoder is differentiated in two steps; a reading phase, that reads and partitions the raw bitstream into codewords, and the decoding phase, that transforms the codewords into information elements, used to reconstruct the slice. Depending on the information they represent, parameters are encoded in the following different

ways:

- **Fixed Length codewords (FL):** composed by a known number of bits.
- **Exp-Golomb coded codewords (EG):** exponential Golomb Codes, adopted by H.264/AVC, characterized by a regular logical structure consisting of a predetermined code pattern and no requirement of decoding tables.
- **Tabled codewords (TE):** the VLC words to be found in a look-up table. H.264/AVC defines several VLC tables for different syntax elements and contexts.
- **VLC level codewords (VL):** context adaptive coding style, characteristic for the residual levels encoding.

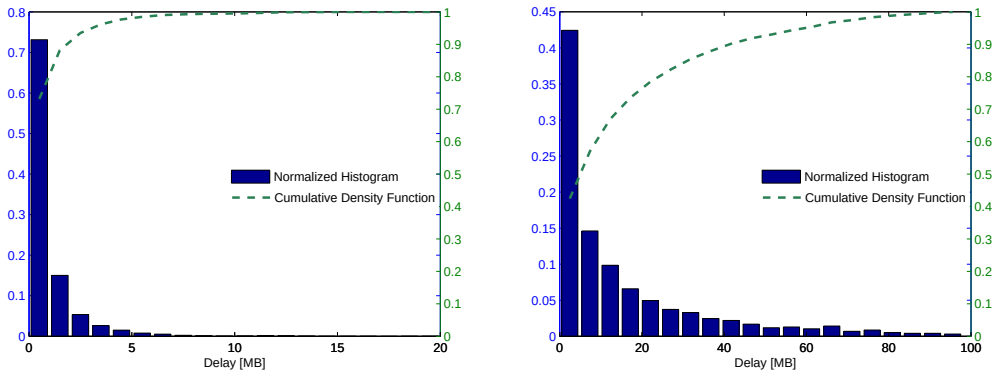
In order to decide that a codeword is erroneous, the characteristics of the codeword are exploited. Based on this, erroneous codewords are classified as follows:

- **Illegal Codeword (IC):** the codeword does not find correspondence in the look-up table. IC occurs during the reading process for tabled, exp-Golomb coded and fixed length codewords.
- **Out of Range Codeword (OR):** decoded values are outside the specified range. This is identified during the reading process, and applies to all type of codewords.
- **Contextual Error (CE):** the decoded word causes the decoder to perform illegal actions. It arises during decoding, for tabled, exp-Golomb coded and fixed length encoded parameters.

According to the simulations performed in [2] and [7], the detection distance is higher for P than for I frames; and about 65% of the errors are detected within two macroblocks after the error occurrence for I frames when using Syntax check, as shown in Figure 2.1. Considering this, a direct previous macroblock concealment, of 1MB or 2MBs, is implemented and tested respectively. It is important to notice that this approach does not represent an error detection method itself. The direct concealment of macroblocks, previous to the error detection, might also conceal correct macroblocks, thus degrading the quality at some points.

2.2 Implementation

The implementation of the method is performed at the decoder and unified with the Syntax check code. Concretely, the code is added to the function `decode_one_slice`, located in the `image.c` file. The procedure for the concealment of two macroblocks previous to the error detection (2MB Prev.Conc.) is illustrated in Figure 2.2. When



(a) I frames detection delay for Syntax check (b) P frames detection delay for Syntax check

Figure 2.1: Detection delay for I and P frames, with Syntax check. Normalized histogram and Cumulative Density Function

the first error in a slice is detected by Syntax check, the MB number in which the error is detected, N, is stored (being $N = 1$ the first macroblock number in a slice); and from that point, the rest of the slice is concealed. At the end of the process, the concealment of two macroblocks previous to the error detection (in case $N > 2$)

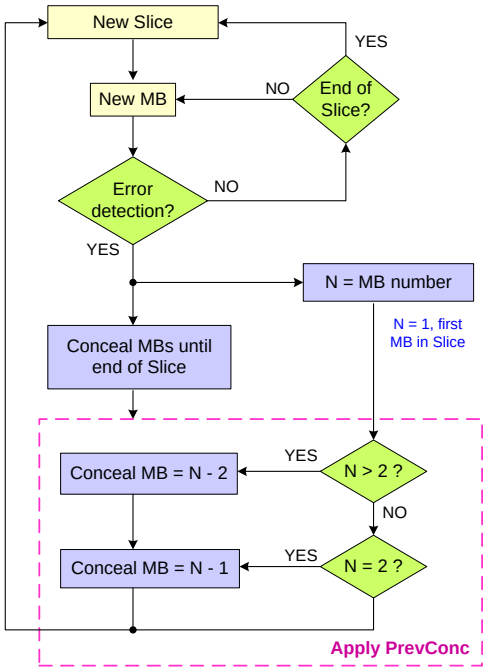


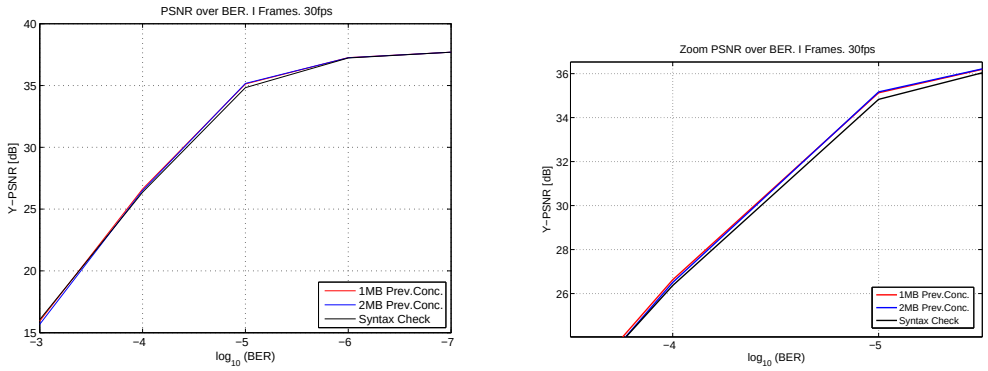
Figure 2.2: Concealment of two macroblocks previous to the error detection occurrence (2MB Prev.Conc.)

is performed. If the error detection occurs in the second macroblock of the slice ($N = 2$) the concealment of one previous macroblock is still possible, and thus the first MB of the slice is concealed.

Note that the method only conceals the previous macroblocks, and the concealment of the slice from the error occurrence until the last MB is performed by the Syntax check function. This can be done independently because the concealment method is the copy paste of the corresponding block of the previous image, and the concealment of a macroblock does not depend on the actions taken on the concealment of the neighboring macroblocks. The concealment of one macroblock previous to the error detection (1MB Prev.Conc.) follows the same steps shown in Figure 2.2, but just checking if $N = 1$, and in that case, as the detection is in the first MB of the slice no previous concealment is performed.

2.3 Simulations and results

Simulations are performed with the initial settings described in Section 1.6, for Syntax check alone, and together with one or two previous macroblock concealment.



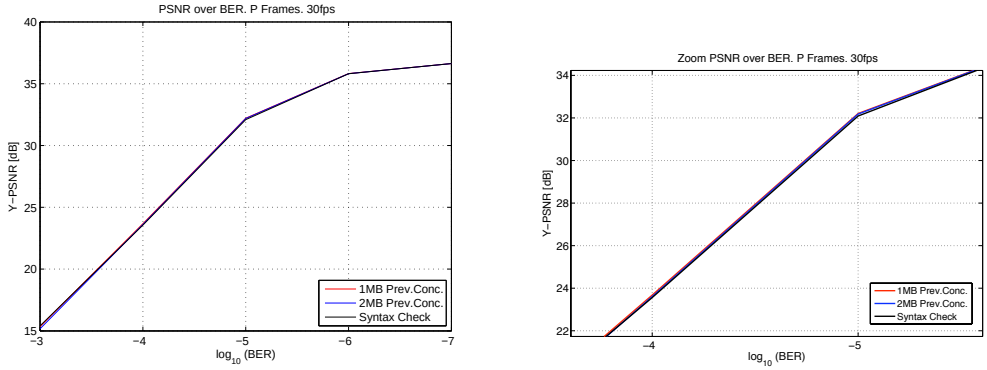
(a) Quality over BER for I frames

(b) Detail for BER from 10^{-4} to 10^{-5}

Figure 2.3: Received quality over BER for I frames. Comparison of Syntax check, 1MB Prev.Conc. and 2MB Prev.Conc.

Figure 2.3 and Figure 2.4 show the results of simulations performed in this work, where the distortion at the receiver versus the BER is plotted for I frames and P frames respectively. The quality over BER for I frames is higher than for P frames. This can be explained by the fact that detection distances for P frames are bigger than for I frames, thus the effect of concealing one or two macroblocks is slightly perceived. Moreover, for all the methods the quality improvement for I frames is bigger than for P frames. This is derived from the fact that P frames are inter-coded, so the initial quality after encoding is lower than for I frames.

When comparing the performance of each of the methods, different behaviors are observed depending on BER values, as illustrated on Figure 2.3 for I frames. On the



(a) Quality over BER for P frames

(b) Detail for BER from 10^{-4} to 10^{-5}

Figure 2.4: Received quality over BER for P frames. Comparison of Syntax check, concealment of 1MB Prev.Conc. and 2MB Prev.Conc.

one hand, for higher values of BER (10^{-3}) the concealment of the two macroblocks previous to the error (2 MB PrevConc) achieves the worse performance. At this point, as a great amount of errors are placed on the stream, the number of detections is also higher. In most of the cases the errors are detected on the MB where they occur, thus the concealment of previous MBs is actually the concealment of non erroneous MBs. On the central zone, when the BER equals 10^{-5}, an improvement of 0.3dBs over Syntax check alone is achieved for 1MB PrevConc. For low BER (10^{-7}), all three methods achieve approximately the same quality. In this case, as the amount of errors is really small nearly no concealment is performed, thus the decoded quality is comparable to the quality of the error-free transmission.

Chapter 3

Watermarking

3.1 Introduction and theoretical approach

Watermarking (WM) consists in adding redundancy at the pixel level in the encoder in order to locate errors at the receiver. A Force Even Watermarking (FEW) scheme was first introduced for H.263 in [9]. In this work, a fragile watermark is forced onto quantized DCT coefficients at the encoder; the decoder checks the correctness of this mark and is able to detect errors at the MB level. The main problem of this approach though, is that even when invisible Watermarking is used, an initial distortion is introduced in the video sequence. Lately, a watermarking scheme was used also in [8], applying FEW to H.264/AVC and proposing a Relation Based Watermarking (RBW) approach.

The results obtained in both approaches shown great improvement in the final quality obtained at the receiver. Taking these into consideration, a combination of Syntax check together with different Watermarking schemes is proposed, to see the grade of enhancement that can be achieved when both methods cooperate. One of the most important differences when comparing the presented work with the mentioned before, is the type of errors. In [9] and [8] errors are introduced directly and exclusively on DCT coefficients, whereas in the presented work errors are randomly distributed over the whole H.264/AVC video. This represents a more realistic approach, since also effects like desynchronization are considered.

FEW consists in changing the values of one or more AC coefficients, from a given position p inside a sub-macroblock, and following the zig-zag scan until the end of the given sub-macroblock. Considering a sub-macroblock of size $N \times N$ (4×4 in our case) with DCT coefficient $a_n, n \in [2, N^2]$[1], each a_i with $i = p \dots N^2$, the resulting $a_i^{(w)}$ watermarked coefficient follows

[1]The first coefficient corresponds to the DC value, and this is never watermarked in this work.

$$a_i^w = \begin{cases} a_i & ; \quad |a_i| \text{mod } 2 = 0 \\ a_i - \text{sign}(a_i) & ; \quad |a_i| \text{mod } 2 = 1; \end{cases} \tag{3.1}$$

where mod 2 stands for the operation modulo 2 and sign(a_i) for

$$a_i^w = \begin{cases} 1 & ; \quad a_i \geq 0 \\ -1 & ; \quad a_i < 0. \end{cases} \tag{3.2}$$

This study is focused on the combination of Watermarking techniques together with Syntax check. As Syntax check achieves good error detection results in I frames, and because of the fact that watermarking I frames causes high distortion on the encoder, the watermark is only applied to P frames. Even when just watermarking P frames, FEW with low values of p still causes a high initial degradation. Theoretically, the lower the value of p, the higher the error detection probability is; nevertheless, if the initial distortion is too high, it is hard for the concealing methods to overcome it. An example of this initial distortion when watermarking with FEW and different values of p is show in Figure 3.1.

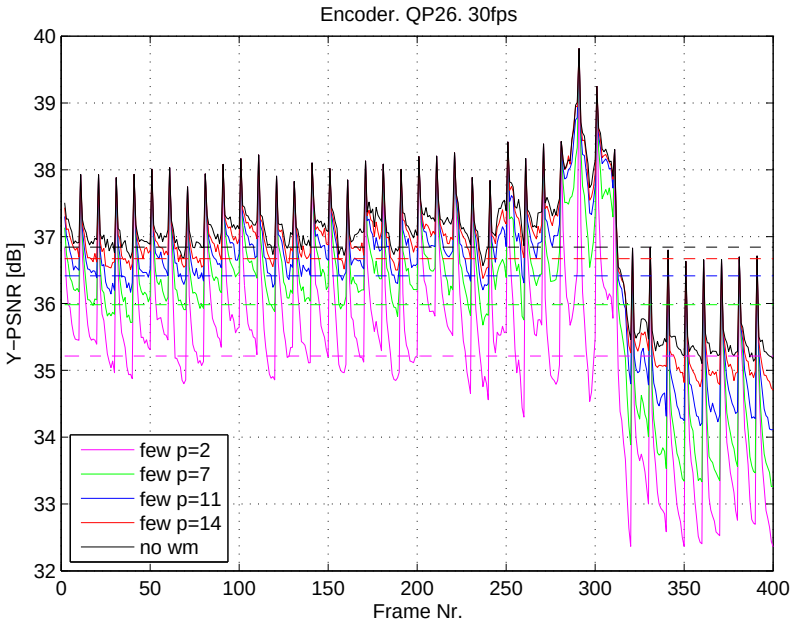


Figure 3.1: FEW initial distortion at encoder for values of $p = 2, 7, 11, 14$ vs. non-watermarked sequence

The effect of the initial degradation is mainly caused by the fact that forcing coefficients to even numbers in P frames, most of the time is the same as turning the trailing ones of the sub-macroblock to zero. This has also the implication that the compression is higher, indirectly reducing the size of the final compressed video. Considering these, a Force Odd Watermarking (FOW) scheme is also implemented together with Syntax check and evaluated. FOW performs exactly in the same way as FEW, but changing DCT coefficients to odd values (thus preserving the trailing ones). Considering a sub-macroblock of size $N \times N$ with DCT coefficient $a_n, n \in [2, N^2]$, each a_i with $i = p \dots N^2$, and $a \neq 0$, the resulting $a_i^{(w)}$ watermarked coefficient follows what is stated in Equation 3.3, with sign(a_i) defined in Equation 3.2. Coefficients with 0 value are not modified.

$$a_i^w = \begin{cases} a_i & ; \quad |a_i| \text{mod } 2 = 1 \\ a_i - \text{sign}(a_i) & ; \quad |a_i| \text{mod } 2 = 0; \end{cases} \tag{3.3}$$

Using FOW, under the same considerations of Figure 3.1 decreases the initial encoding distortion considerably, as show in Figure 3.2. On the other hand, FOW does not achieve the higher compression that FEW is able to provide. An evaluation in terms of distortion over bitrate is to be found in Section 3.3.

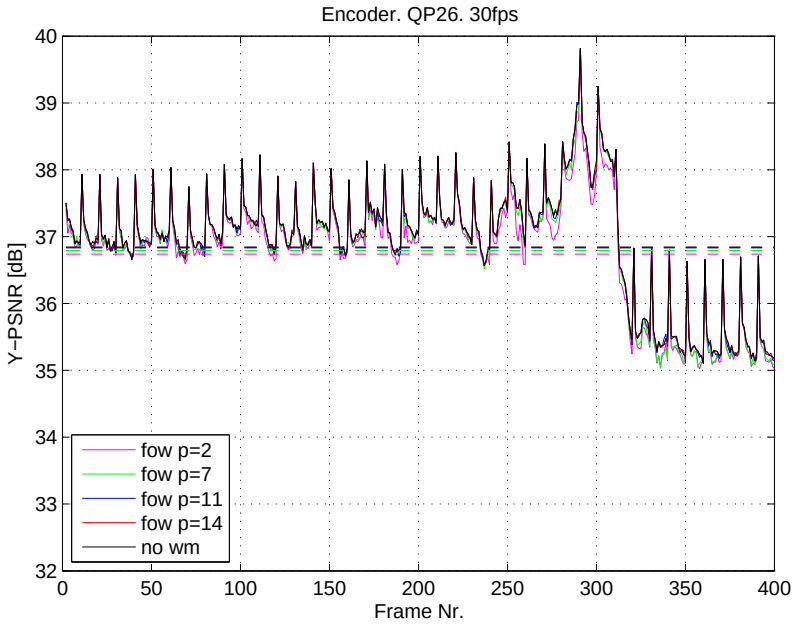
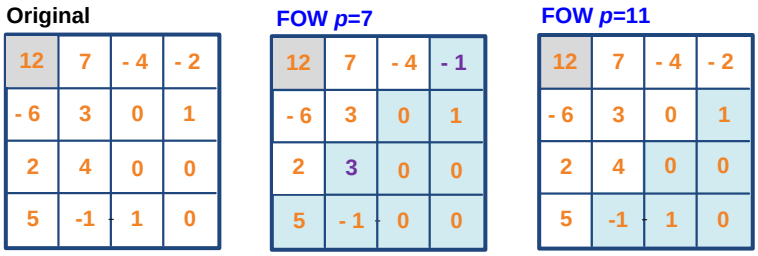


Figure 3.2: FOW initial distortion at encoder for values of $p = 2, 7, 11, 14$ vs. non-watermarked sequence

The differences between both FEW and FOW watermarking are shown in Figure 3.3

(a) Watermarking with FEW



(b) Watermarking with FOW

Figure 3.3: Watermarking differences of FEW and FOW. Example using values of $p = 7$ and $p = 11$, for the same original sub-macroblock

3.2 Implementation

The implementation of FEW and FOW is performed at the encoder and decoder, unified with the Syntax check code. At the encoder, the functions `dct_luma` and `dct_luma_16x16`, in the file `block.c` are modified. The function `dct_luma` performs the transformation, and quantization; moreover, it performs an optimization to select the most appropriate coefficient levels, discarding expensive values. The function `dct_luma_16x16` does the same but for intra coded macroblocks.
At the decoder, the functions `decode_one_slice`, in the file `image.c` as well as the function `readCBPandCoeffsFromNAL`, in the file `macroblock.c` are modified. The function `readCBPandCoeffsFromNAL` is responsible for the extraction and interpretation of the values received from an H.264 NALU, and `decode_one_slice` decodes a slice, reading macroblock per macroblock.

3.2.1 WM encoder

The implementation of the JM encoder has an inner-decoder, used for the prediction. When calculating the DCT coefficients, the values are stored both in the final

H.264/AVC file but also in the buffer of the inner-decoder. The modification of the coefficients because of the watermarking is performed together with the optimization, meaning that the decisions on the prediction consider the already watermarked coefficients, and in consequence, WM can be applied to the same coefficient more than once. For this reason, all the modifications made by WM are applied to both to the H.264/AVC file and to the inner-decoder buffers.

The implementation on `dct_luma` uses a loop for the 16 coefficients of a 4×4 sub-macroblock. Each time, a *level* (DCT coefficient) is calculated, and stored into a vector called *ACLevel*. *ACLevel* contains all the non-zero values for a sub-macroblock, following the zig-zag order. In order to know where the zeros are placed, the vector *ACRun* is defined. *ACRun* is synchronized with ACLevel. The positions where ACRun has a positive value (non-zero) determine the number of zeros to insert until the next value; i.e. *ACRun[1]* with value 2, means that after value of *ACLevel[2]*, 2 zeros must be inserted. This is signalised with the variable *run*. In the case of FEW, some coefficients can be turned to zero, then the correct modifications of these variables is crucial to assure that all the coefficients are correctly stored and signalized. This procedure is shown for FEW in Figure 3.4.

3.2.2 WM decoder

The procedure at the decoder is more simple than the one at the encoder. In the function `readCBPandCoeffsFromNAL`, a check is performed after each DCT coefficient extraction. For P frames, and when the coefficient positions is equal or bigger than p, if the value received differs from even (for FEW) or odd (for FOW), an error detection is signalized for the given MB. The errors detected by WM are at the macroblock reading step. The management of the errors is performed in the function `decode_one_slice`. When an error is detected in a MB, either by Syntax check or by WM, the concealment procedure for the whole slice starts. At this step, the computation of the number of error detections as well as the detection distance, is also performed.

3.3 Simulations and results

This section evaluates the results obtained through simulation using the parameters described in Section 1.6. First, an evaluation in terms of degradation (Y-PSNR) depending on the BER and depending on time, is performed. Then, results of distortion over bitrate are presented, comparing FEW and FOW. The final part of the section is devoted to analyze the results obtained for error detection probability and for error detection delay.

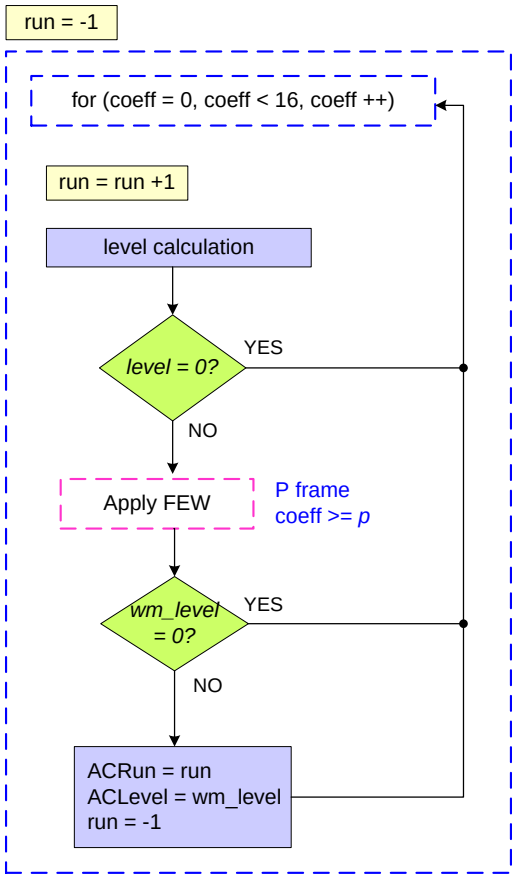


Figure 3.4: Encoder procedure for implementation of the watermarking with FEW at the JM

3.3.1 Distortion evaluation

Simulations are performed for different BER values, for both FEW and FOW. The results shown are for 30 f/s.

In Figure 3.5 and Figure 3.6, the received quality depending on the BER, for different values of p and compared with Syntax check alone, is shown for FEW and FOW respectively. In both cases, the quality improves gradually as the number of errors is reduced (as the BER is decreased). When taking a look to high BERs $(10^{-3}, 10^{-4})$ FEW with small values of p is able to improve the quality, compared to Syntax check alone and to FOW. The opposite occurs for low values of BER $(10^{-6}, 10^{-7})$: the effect of the initial degradation for low BERs in FEW is predominant and therefore FEW is not able to overcome Syntax check alone nor FOW for any value of p. As the initial watermarking degradation is not that severe in FOW, the received quality for low BERs with FOW is similar for the different values of p, and in some cases overcomes the received quality for Syntax check alone.

In order to take a deeper look at the mentioned performances of FEW and FOW

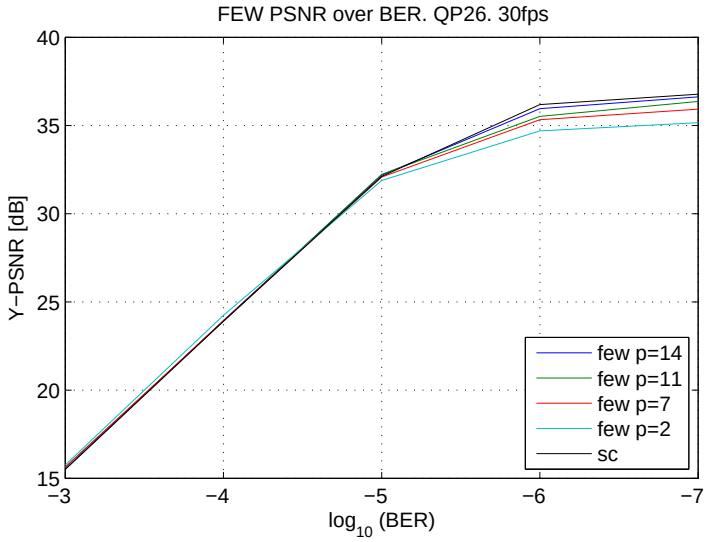


Figure 3.5: FEW distortion over BER at the decoder

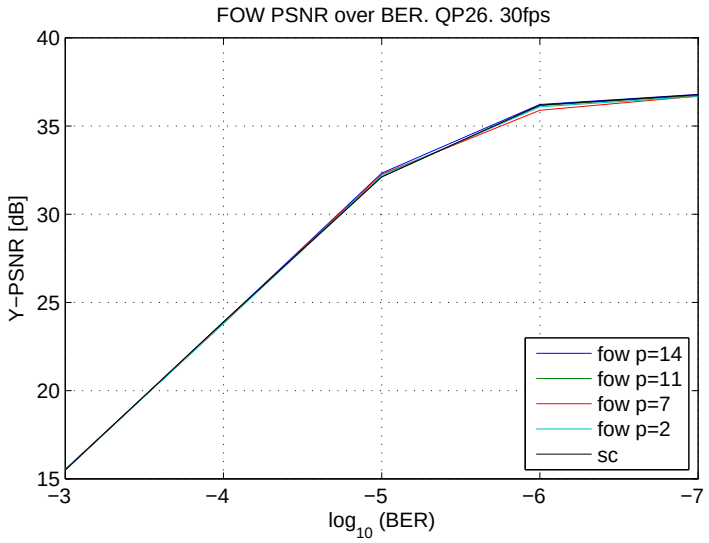


Figure 3.6: FOW distortion over BER at the decoder

depending on the BER, a comparison of the received video quality over time for the three methods (Syntax check, FEW and FOW), depending on p, for a high BER (10^{-4}) and for a lower BER (10^{-6}) is provided in Figure 3.7 and Figure 3.8 respectively.

When comparing quality over time, the instant quality per frame can be seen. In the chosen sequence, for example, the quality experience a great decrease at the frames from 250 to 350 due to a fast camera movement. The methods can achieve

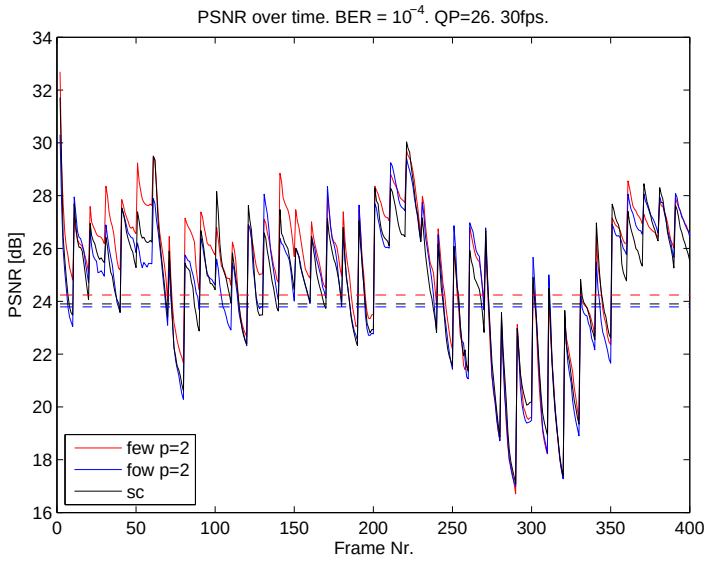


Figure 3.7: Distortion over time. BER $= 10^{-4}$. Comparison of Syntax check, FEW and FOW with $p = 2$

better or worse results depending on the frame but in order to be evaluated, the mean quality is considered. In Figure 3.7 for a high BER (10^{-4}) FEW with $p = 2$ overcomes the performance of Syntax check and FOW, in terms of mean received

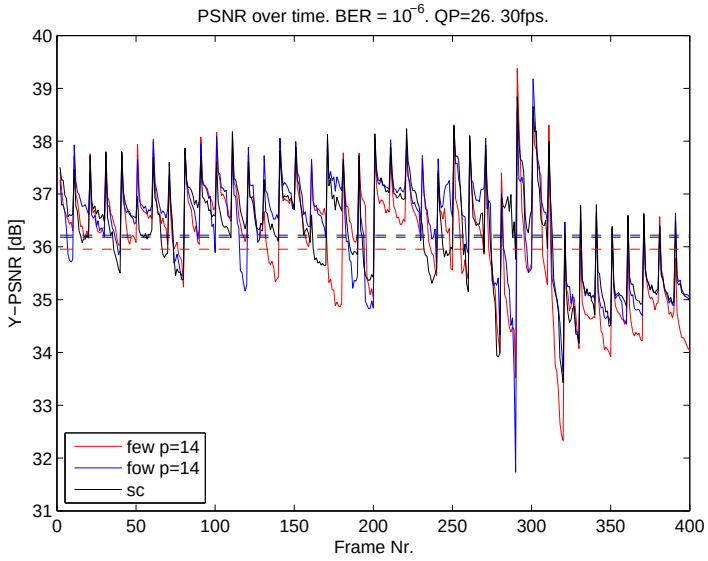


Figure 3.8: Distortion over time. BER=10^{-6}. Comparison of Syntax check, FEW and FOW with $p = 14$

quality, with an increase of 0.34dB with respect to Syntax check. For a BER of 10^{-6} and with a high value of p (Figure 3.8), FOW achieves the best results, with a little improvement of 0.04dB with respect to Syntax check. Considering the results, the general conclusion consists in FEW performing better with high BERs and low values of p and FOW sometimes improving the quality for low BERs and high values of p.

3.3.2 Distortion over bitrate analysis

FEW provides a higher level of compression due to the conversion of the trailing ones to zero. Considering this, it is interesting to compare the distortion over bitrate for the different watermarking (FEW and FOW) with Syntax check, with and without errors. To do this, the video sequence is encoded with different quantization parameters QP = [20,26,30]. Results for both 30f/s and 10f/s are shown.

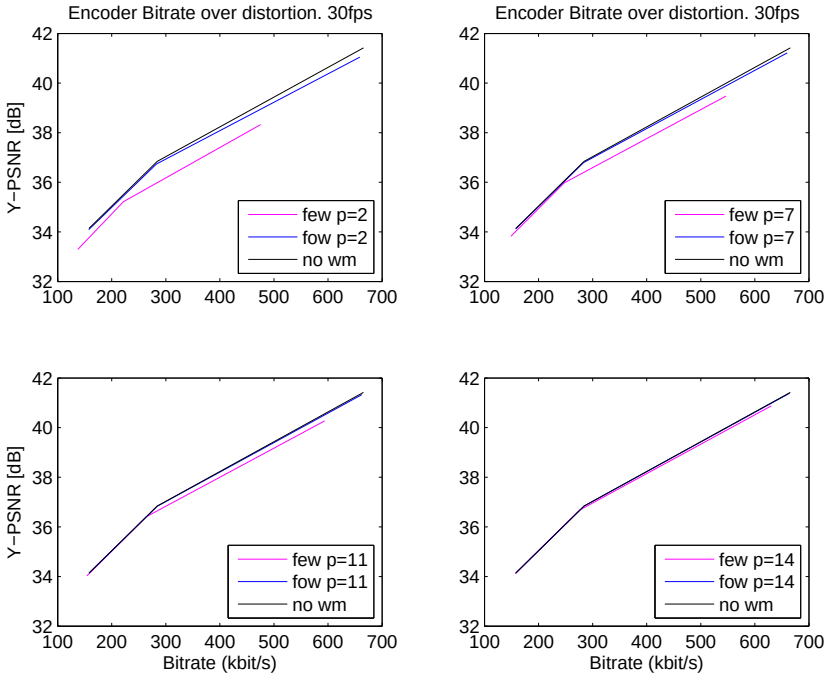


Figure 3.9: Encoder distortion over bitrate for different values of p and 30f/s. Comparison for FEW, FOW and the non-watermarked sequence

Figure 3.9 shows a comparison of the distortion over bitrate, depending on the value of p. The method achieving best results is encoding without watermarking, because in this case the original video content is not changed. In contrast, FEW achieves the worse results for lower values of p, where the initial distortion is more severe. The interesting aspect as this point is to see whether this behavior can change at the decoder.

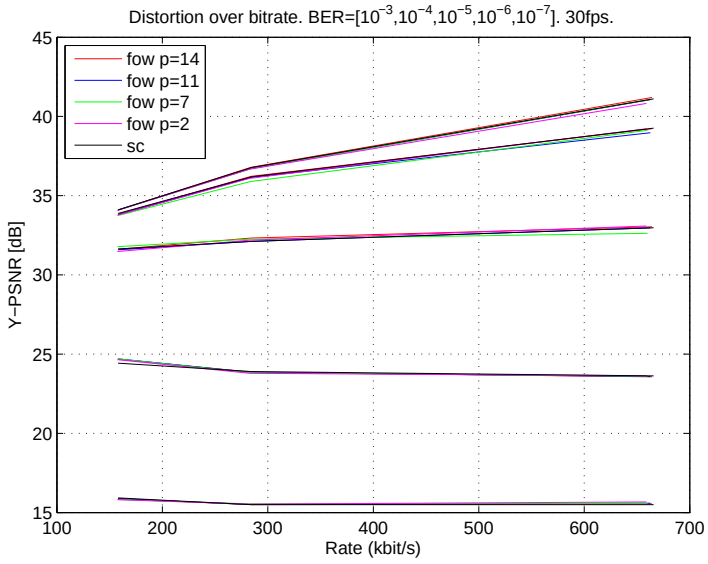


Figure 3.10: Distortion over bitrate for FOW and Non-WM videos for all BERs and 30f/s

When introducing errors, the overall behavior for both FEW and FOW is the same and varies depending on the BER. This is shown in Figure 3.10, for FOW. This figure needs to be read from upside-down, the upper plots corresponding to lower BERs (higher quality at the decoder) until higher BERs at the bottom. For high BERs (10^{-3} and 10^{-4}) low bitrates achieve higher quality than higher bit rates. This behavior does not look coherent at first sight, and could explained by the fact that fixed length packets are used. Lower bitrates mean higher QPs, thus more MBs for a given packet, or a higher granularity. When having the same amount of errors per packet in average, but more and smaller MBs, the amount of information lost is also smaller, thus slightly increasing the quality at the receiver. In contrast, for lower BERs the errors affect a bigger area, thus having the same shape observed at the encoder side in Figure 3.9, where the quality increases along with the bitrate.

Nevertheless, in order to see the differences within the methods, an analysis for different BER is performed. Figure 3.11 shows the distortion over bitrate for different p values with a BER of 10^{-3} and 30f/s. It can be observed that the best results are obtained for FEW when $p = 2$, and the differences of FEW with respect to FOW and Syntax check are reduced as p is increased; to the point that, when $p = 14$ all the methods have a similar behavior. When the frame rate is reduced to 10 f/s (Figure 3.12) FEW performance overcomes FOW and Syntax check, in this case with independence on the value of p.

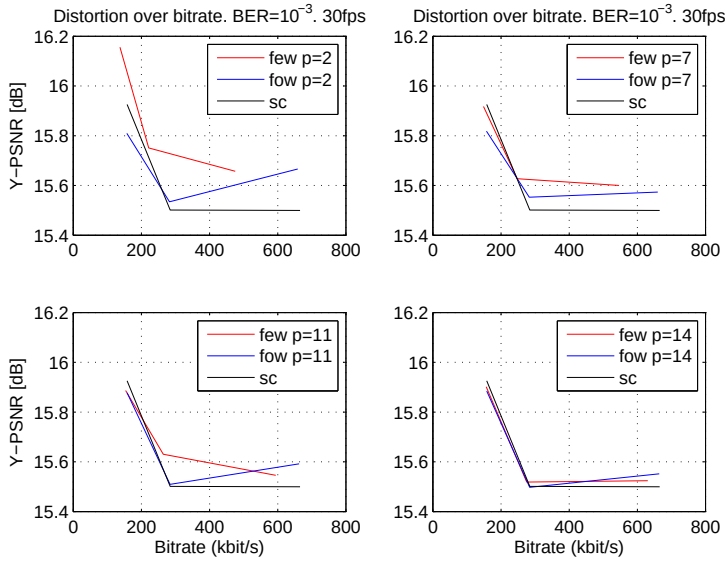


Figure 3.11: Distortion over bitrate depending on p for BER=10^{-3} and 30 f/s

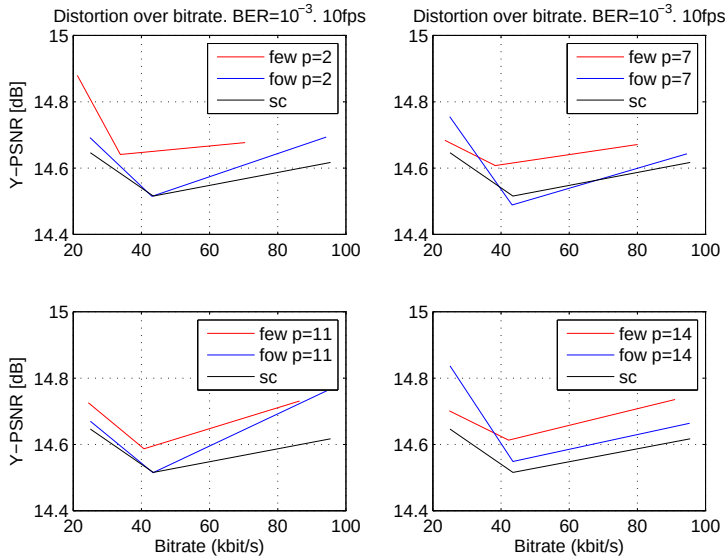


Figure 3.12: Distortion over bitrate depending on p for BER=10^{-3} and 10 f/s

When considering low values of BER, for 30f/s (Figure 3.13) the general trend is that FOW improves the behavior of FEW and Syntax check, specially for $p = 14$. For the case of 10f/s (Figure 3.14), the superiority of FOW is predominant, overcoming the results of the other two methods, with independence of p. This effect is the same as observed with high BER for 10f/s (Figure 3.12).

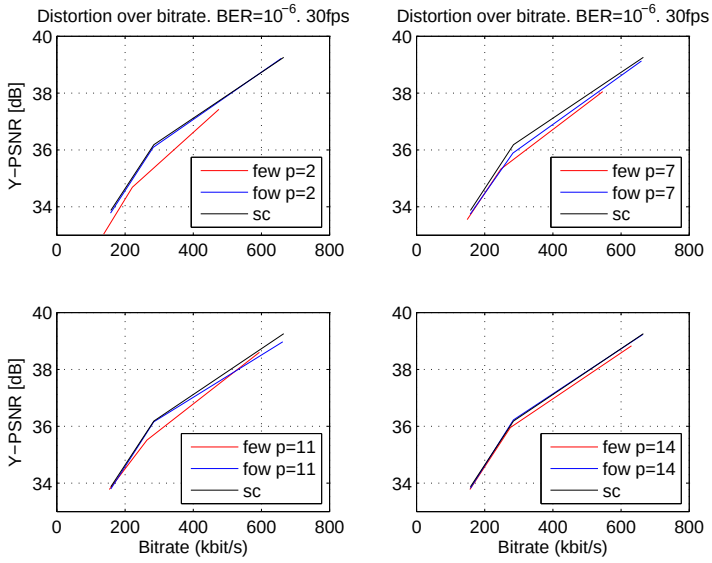


Figure 3.13: Distortion over bitrate depending on p for BER=10^{-6} and 30 f/s

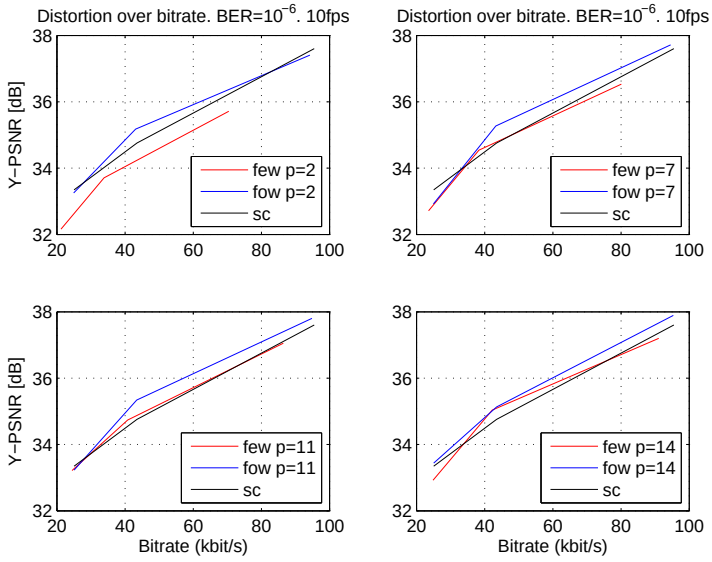


Figure 3.14: Distortion over bitrate depending on p for BER=10^{-6} and 10 f/s

Considering these, the results obtained for the distortion-bitrate comparative with 30f/s support the already observed with the distortion over BER and time; in general, FEW provides better quality for high BERs and low values of p, whereas FOW does it for low BERs and high values of p. When having 10f/s, this behavior is also accomplished, but with independence on the value of p. It is important to notice, that the results are strongly dependent on the placement of errors; thus simulations

with different errors could lead to fluctuations in these results.

In order to provide realistic results, several simulations are performed and averaged. Moreover, errors need to be generated in a random way (in this work, this is achieved with a BSC); and this implies that some errors can be placed in critical parts of the H.264/AVC file, causing great fluctuations in the final quality at the decoder. Finally, it is important to consider that Watermarking introduces an extra compression, thus each watermarked video has different size depending on the value of p and on the content itself; for this reason, even if the same error traces were considered in all the experiments, the effect of the errors would be different in each H.264/AVC file.

3.3.3 Error detection probability and error detection delay

The error detection probability permits to compare the percentage of errors detected for a given error detection method. In this case, the interest is focused on the error detection probability of the combination of watermarking and Syntax check, resulting in the methods FEW and FOW. In the simulations for the error detection computation, one single error per slice is introduced. On the one hand, because in this way an easier computation of the detected and undetected errors can be performed, and on the other hand because once an error is located, the concealment is performed until the end of the slice. Table 3.1 shows the error detection probability results.

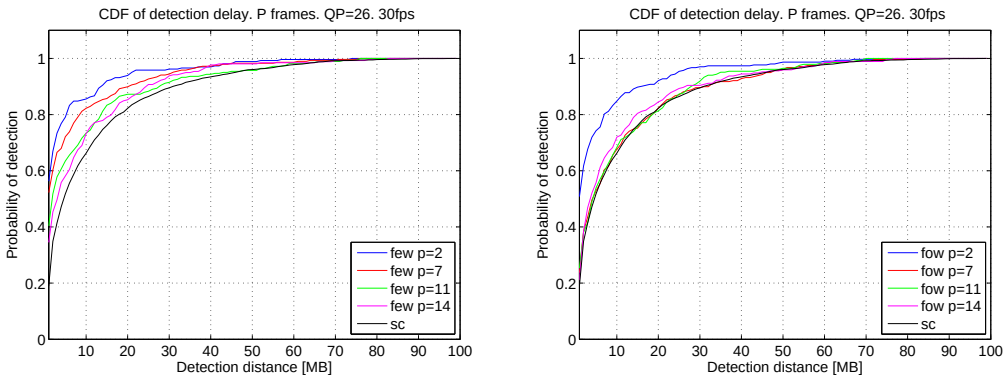
Table 3.1: Error detection probabilities for FEW and FOW (QP = 26)

Error detection probabilities (QP=26)					
30 f/s			10 f/s		
p	FEW	FOW	p	FEW	FOW
2	65.6%	57.96%	2	64.39%	62.93%
7	65.33%	52.48%	7	62.21%	53.28%
11	59.1%	53.49%	11	51.44%	49.23%
14	50.87%	51.47%	14	50.36%	49.80%

Considering the error detection probabilities for 30f/s and for FEW, the smaller the value of p, the higher is the error detection probability. Achieving the the best result for $p = 2$ with an error detection probability of 65.6%. The error detection probabilities for FOW and 30f/s are lower, achieving a 57.96% as the maximum, for $p = 2$. Moreover, FOW seems not to be dependent on the p value for $p > 2$, thus achieving better results for $p = 11$ than for $p = 7$, for example. This behavior can be explained by the fact that for higher p values the watermarked coefficients are mainly trailing ones, and thus the odd watermarking in this region introduces a random component strongly dependent on the content.

The results obtained for 10f/s are comparable to ones the obtained for 30f/s and follow the same trend. For FOW with $p = 2$, a detection probability of 62.93% is achieved, overcoming the result obtained for 30f/s. Nevertheless, the most probable explanation to this are the fluctuations of the results depending on the type of errors inserted. In any case, and comparing the results with the error detection probability of Syntax check (that lies around the 50%), to use watermarking together with Syntax check represents a great improvement in terms on error detection, specially for FEW with low values of p.

The cumulative distribution function (CDF) of the detection delay shows which of the method is able to locate more errors sooner. Figure 3.15 shows the CDF obtained for FEW and FOW, for all p values, and compared with the results obtained with Syntax check alone.



(a) FEW detection delay

(b) FOW detection delay

Figure 3.15: Cumulative distribution function (CDF) of the detection delay. Comparison of watermarking with Syntax check alone, for all values of p. Results for P frames, QP=26 and 30f/s

The results for FEW tally with the observed for the error detection probability: the lower the value of p, the shorter is the detection distance. When $p = 2$ FEW achieves the best result, with an 85% of error detections with detection distance below 10 MBs, compared with the 65% of Syntax check alone. As p increases, the detection delay probability for FEW decreases gradually as well. In FOW, the decrease in error detection delay when comparing $p = 2$ with higher values of p is blunter. FOW with $p = 2$ achieves similar results to the observed by FEW, as shown in Figure 3.16.

The results of the CDF for 10f/s are shown in Figure 3.17, for the compared methods and different values of p. In this case, the best result is achieved for WM when $p = 2$, with more than 90% of error detections with detection distance below

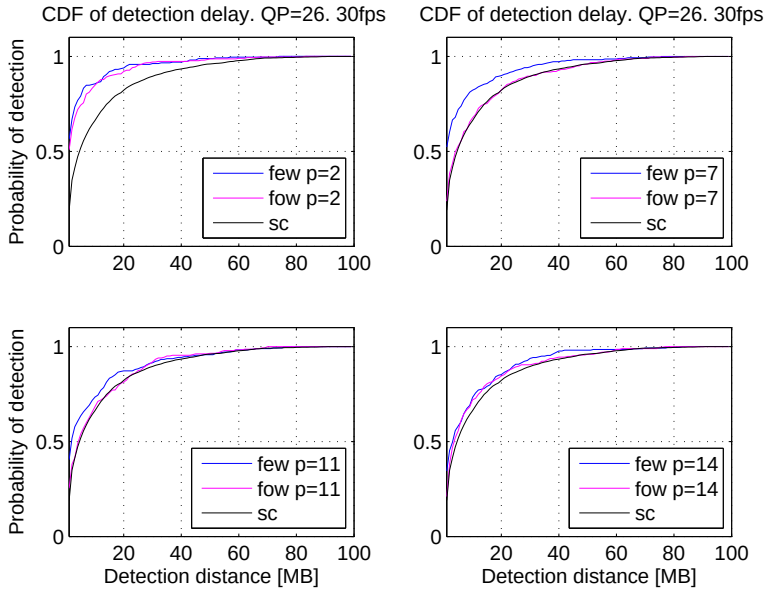


Figure 3.16: Cumulative distribution function (CDF) of the detection delay. Comparison of FEW, FOW and Syntax check alone for different p values and 30f/s.

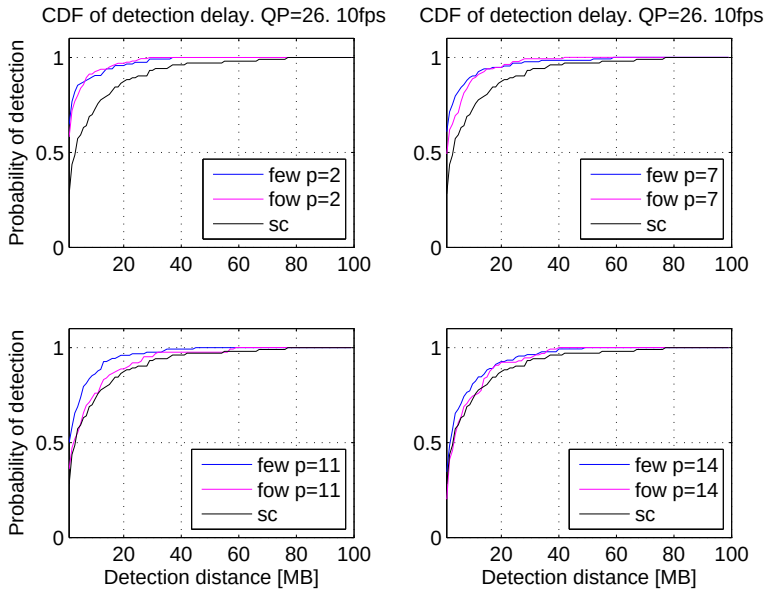


Figure 3.17: Cumulative distribution function (CDF) of the detection delay. Comparison of FEW, FOW and Syntax check alone for different p values and 10f/s.

10 MBs, compared to the 70% of Syntax check. Nevertheless, it is interesting to notice that in this case the differences between FEW and FOW are reduced; thus in general for a given p both methods have approximately the same performance.

Considering the presented results of distortion, error detection probability and detection delay it is possible to state that WM together with Syntax check improves the performance of Syntax check alone. In general, FEW provides better results with high BERs and using low values of the parameter p; whereas FOW is able to outperform FEW and Syntax check with lower BERs and specially with higher values of p. When the frame rate is reduced from 30 f/s to 10 f/s, two effects are observed: the results for WM in terms of distortion and detection delay seem to be independent of the value of p and the results of FEW and FOW become very similar. Taking these into account, FOW could be a good candidate when working with low frame rates, because it is able to reach the performance of FEW but reducing the initial degradation cost.

Chapter 4

Checksum

4.1 Introduction and theoretical approach

Checksum (CH) consists in adding redundancy bits to a content transmitted over an error prone channel, in order to be able to detect errors on reception. At the encoder side, some function is calculated over some of the data bits to be send; the obtained result is also transmitted to the receiver, that performs the same operation over the same bits. An error is detected if the results (checksum received and value obtained for the decoder) differ. Checksums can be implemented in several ways, by changing the amount of content to protect, the operation performed over it, and the in or out-band transmission. Depending on these, the error detection probability is higher or lower. Unfortunately, the overhead introduced by the additional checksum information is strongly related with the good performance of the detection, i.e. the amount of information sent by a checksum that applies over a whole frame is less than applying a checksum per macroblock, nevertheless, with the second approach the granularity of the detection is higher. A method applying parity bits over a group of MBs in H.264/AVC videos is already proposed in [10]. In this chapter, based on the results obtained in Chapter 3, an alternative Checksum scheme is implemented together with Syntax check and evaluated through simulation.

4.1.1 Motivation

The Checksum implementation proposed in this chapter responds to a behavior observed on the simulations performed with Watermarking, in Section 3.3. Both the H.264/AVC encoder and decoder output the files `trace_enc.txt` an `trace_dec.txt` respectively. The trace files contain all the values that the encoder writes on the H.264/AVC file and the decoder reads. In an error-free transmission, written and read values are the same. In presence of errors, the decoder interprets some values

wrongly, and thus trace files at the encoder and decoder are different at some points.

In order to analyze in which cases the Watermarking failed to detect errors on the Luma field, a brief analysis of the trace files both in encoder and decoder was performed. The main conclusion was that the method was able to recognize errors, when any detectable change on the Luma coefficients happens. Nevertheless, two more effects made some errors undetectable by Watermarking:

- The effect of errors caused the decoder not to decode some macroblocks, thus the WM information contained on those MBs was useless.
- Luma values were decoded as other type of information, like Chroma values, for example.

These behaviors happened due to errors affecting the elements `mb_skip_run` and `coded_block_pattern` (cbp) respectively. Analyzing the codification of these elements, and the important repercussion of errors on them, an specific Checksum scheme to protect these values is defined. In this chapter, Subsection 4.1.2 describes the encoding of these two elements, and the repercussions of errors on them; whereas Subsection 4.1.3 describes the method, the information protected and the chosen encoding method for the Checksum values.

4.1.2 Characteristics of `mb_skip_run` and `coded_block_pattern`

The `mb_skip_run` field is defined in [1] as:

"mb_skip_run specifies the number of consecutive skipped macroblocks for which, when decoding a P or SP slice, mb_type shall be inferred to be P_Skip and the macroblock type is collectively referred to as a P macroblock type. {...} The value of mb_skip_run shall be in the range of 0 to PicSizeInMbs - CurrMbAddr, inclusive".

The skip mode is useful in frames using prediction (P frames), as explained in Section 1.2.2. The value of `mb_skip_run` determines the number of MBs to skip, and is present in every MB of P frames. For non-skipped MBs, the value of `mb_skip_run` consists in one bit with value 0. Considering desynchronization, the probability of error in this field for MBs containing information is very high. In consequence, signalizing the presence of the skip mode makes special sense to improve the error detection capabilities.

The field `coded_block_pattern` (also know as "cbp") determines which submacroblocks contain coded coefficients and the type of these [12]. The `coded_block_pattern` is encoded using a variable length Exp-Golomb entropy coding, where shorter codewords are assigned to the most frequently values. The most common values are assigned one bit with value 1.

```
************ Pic: 2 (I/P) MB: 71 Slice: 0 ***********

************ Pic: 2 (I/P) MB: 71 Slice: 0 ***********

************ Pic: 2 (I/P) MB: 72 Slice: 0 ***********

************ Pic: 2 (I/P) MB: 72 Slice: 0 ***********

************ Pic: 2 (I/P) MB: 73 Slice: 0 ***********

************ Pic: 2 (I/P) MB: 73 Slice: 0 ***********

838187 mb_skip_run                                                  011 (  2)
838190 mb_type (P_SLICE) ( 7, 6) =   1                                1 (  0)
838191 mvd_l0 (0) =   0  (org_mv   8 pred_mv   8)                     1 (  0)
838192 mvd_l0 (1) =  -1  (org_mv   4 pred_mv   5)                   011 ( -1)
838195 CBP ( 7, 6) =   0                                              1 (  0)
```

(a) Correct MB skip without errors (`trace_enc.txt`)

```
************ POC: 4 (I/P) MB: 71 Slice: 0 Type 0 ***********
838214  mb_skip_run                                               011  (  2)

************ POC: 4 (I/P) MB: 72 Slice: 0 Type 0 ***********

************ POC: 4 (I/P) MB: 73 Slice: 0 Type 0 ***********
838217  mb_type                                                     1  (  0)
838218  mvd_l0                                                      1  (  0)
838219  mvd_l0                                                    011  ( -1)
838222  coded_block_pattern                                         1  (  0)
```

(b) Correct MB skip without errors (`trace_dec.txt`)

Figure 4.1: Example of a correct `mb_skip_run` of two macroblocks (MB = 71 and 72) at the P frame 2 (Pic: 2).

The illustration of this behavior is shown in Figure 4.1 and in Figure 4.2 for the case of the `mb_skip_run`. Figure 4.1 shows the desired behavior for the traces of both encoder and decoder[1] for frame 2, where macroblocks 71 and 72 are skipped. The format at encoder and decoder differs a bit: at the encoder side, no information is written for the two skipped MBs, and this is signalized in the following MB with data (MB: 73); at the decoder side the skip is signalized at the first MB that is actually skipped (MB:71).

Figure 4.2 shows the case in presence of errors. At the encoder trace, a shortened version of the MBs 87, 88, and 89 are shown. These MBs are non-skipped MBs containing Luma and other information corresponding to the MB. Observing the decoder side, due to errors in the transmission, the `mb_skip_run` value at the macroblock 87 indicates that 10 MBs need to be skipped, and so does the decoder. Inserting redundancy inside the video content results useless in these cases, for this reason, and taking the described effects into consideration, a specific Checksum to protect the `mb_skip_run` and `coded_block_pattern` is presented.

[1] At the encoder, the number of frame appears as POC: 4. This actually refers to frame 2 (the same as the encoder), and just responds to a different format of the `trace_dec.txt`.

```
*********** Pic: 2 (I/P) MB: 87 Slice: 0 ***********

@38842 mb_skip_run                                                    1 (  0)
@38843 mb_type (P_SLICE) (10, 7) =   8                            00101 (  4)
@38848 8x8 mode/pdir( 0) =   5/0                                    010 (  1)
...

@38920 CBP (10, 7) =  35                                      000010101 ( 35)
@38929 Delta QP (10, 7) =   0                                         1 (  0)
@38930 Luma # c & tr.1s(0,0) vlc=0 #c=0 #t1=0                         1 (  0)
@38931 Luma # c & tr.1s(1,0) vlc=0 #c=0 #t1=0                         1 (  0)
@38932 Luma # c & tr.1s(0,1) vlc=0 #c=0 #t1=0                         1 (  0)
@38933 Luma # c & tr.1s(1,1) vlc=0 #c=5 #t1=3                   0000100 (  5)
@38940 Luma trailing ones sign (1,1)                              110 (  6)
@38943 Luma lev (1,1) k=1 vlc=0 lev= -1                            01 ( -1)
@38945 Luma lev (1,1) k=0 vlc=1 lev=  1                            10 (  1)
...

*********** Pic: 2 (I/P) MB: 88 Slice: 0 ***********

*********** Pic: 2 (I/P) MB: 88 Slice: 0 ***********

@39166 mb_skip_run                                                    1 (  0)
@39167 mb_type (P_SLICE) ( 0, 8) =   8                            00101 (  4)
@39172 8x8 mode/pdir( 0) =   6/0                                    011 (  2)
...

@39232 CBP ( 0, 8) =  21                                    00000100110 ( 21)
@39243 Delta QP ( 0, 8) =   0                                         1 (  0)
@39244 Luma # c & tr.1s(0,0) vlc=0 #c=0 #t1=0                         1 (  0)
@39245 Luma # c & tr.1s(1,0) vlc=0 #c=0 #t1=0                         1 (  0)
@39246 Luma # c & tr.1s(0,1) vlc=0 #c=5 #t1=0               00000000111 (  5)
@39257 Luma lev (0,1) k=4 vlc=0 lev= -2                            01 ( -1)
@39259 Luma lev (0,1) k=3 vlc=1 lev= -1                           11 ( -1)
@39261 Luma lev (0,1) k=2 vlc=1 lev=  2                           010 (  2)
...

*********** Pic: 2 (I/P) MB: 89 Slice: 0 ***********

*********** Pic: 2 (I/P) MB: 89 Slice: 0 ***********

@39378 mb_skip_run                                                    1 (  0)
@39379 mb_type (P_SLICE) ( 1, 8) =   3                              011 (  2)
@39382 mvd_l0 (0) =   3  (org_mv   9 pred_mv   6)                 00110 (  3)
@39387 mvd_l0 (1) =   4  (org_mv   9 pred_mv   5)                0001000 (  4)
@39394 mvd_l0 (0) =   2  (org_mv   8 pred_mv   6)                 00100 (  2)
@39399 mvd_l0 (1) =   2  (org_mv   6 pred_mv   4)                 00100 (  2)
```

(a) Sequence of MBs containing valid information at the encoder (`trace_enc.txt`)

```
*********** POC: 4 (I/P) MB: 87 Slice: 0 Type 0 ***********
@38858  mb_skip_run                                          0001011  ( 10)

*********** POC: 4 (I/P) MB: 88 Slice: 0 Type 0 ***********

*********** POC: 4 (I/P) MB: 89 Slice: 0 Type 0 ***********

*********** POC: 4 (I/P) MB: 90 Slice: 0 Type 0 ***********

*********** POC: 4 (I/P) MB: 91 Slice: 0 Type 0 ***********

*********** POC: 4 (I/P) MB: 92 Slice: 0 Type 0 ***********

*********** POC: 4 (I/P) MB: 93 Slice: 0 Type 0 ***********

*********** POC: 4 (I/P) MB: 94 Slice: 0 Type 0 ***********

*********** POC: 4 (I/P) MB: 95 Slice: 0 Type 0 ***********

*********** POC: 4 (I/P) MB: 96 Slice: 0 Type 0 ***********

*********** POC: 4 (I/P) MB: 97 Slice: 0 Type 0 ***********
@38865  mb_type                                              00101  (  4)
@38870  sub_mb_type                                          00101  (  4)
@38875  sub_mb_type                                            010  (  1)
@38878  sub_mb_type                                          00100  (  3)
@38883  sub_mb_type                                            010  (  1)
```

(b) Wrong skip of 10 MBs at the decoder due to errors in the transmission (`trace_dec.txt`)

Figure 4.2: Example of wrong interpretation of `mb_skip_run` at the decoder due to errors

4.1.3 Checksum encoding

Considering the particularities of H.264/AVC, a specific Checksum for this codec is defined. A Checksum code is generated for each macroblock for all frames. A variable

length encoding is used, as described in Table 4.1.

In the encoding, the first bit ($Bit1$) determines wheter the MBs is skipped or

Table 4.1: Checksum encoding

$Bit1$	$Bit2$	`coded_block_pattern`	*Otherwise*
0	0	Nr. of 1's = odd	`mb_type` nr. of 1's = odd
0	1	Nr. of 1's = even	`mb_type` nr. of 1's = even
1		Macroblock skipped	

not. In the case it is not skipped, a field of the MB is protected; the value of $Bit2$ indicates the parity of the field to protect. At MBs where the `coded_block_pattern` value is not present, the `mb_type` is protected instead. The encoding of the parity of `mb_type` introduces a random component, because depending on the video content this value is protected in more or less MBs. This provides a more robust protection than just signalizing the presence of the `coded_block_pattern` because it restricts the possibilities of missing detections. It is important to remark that the usage of two bits for the signalization of non-skipped macroblocks is necessary. Otherwise the parity of the field could not be computed, and moreover, the distinction between skipped and non-skipped macroblocks would not be possible.

4.1.4 Channel and network considerations

Once defined the encoding procedure for the Checksum values, the placement of these values needs to be considered. It is important to notice that it makes no sense to include the Checksum values on the H.264/AVC stream because, on the one hand it would imply to break the standard, and on the other hand the Checksum information would suffer the effect of the errors. Considering the second statement, and the fact that the Checksum coding has been designed using a variable length code, transmitting the redundancy over a prone error channel could result in a decrease in the final performance, because the effect of errors and desynchronization would also be present in the Checksum information.

Taking the previous into consideration, the need to use an error-free channel arises. A simple approximation to this is encapsulating the Checksum information in TCP packets, instead of using UDP. These packets would be synchronized with the UDP transmission of the H.264/AVC, and as TCP allows retransmission, possible losses of Checksum information could be recovered. To have an error-free channel, theoretically infinite retransmission should be allowed, and this is not the case in TCP. Nevertheless, the use of a TCP channel is sufficient to provide a lower error probability for the presented implentation. Moreover, depending on the protocol used to send the information, the following issues need to be considered:

- **Overhead:** there is an explicit overhead due to the Checksum information itself. Even though, an extra overhead due to the transmission protocol in packetizing needs to be considered. The amount of overhead depends on both the protocols used to encapsulate the information and on the periodicity with which the information is sent.

- **Delay:** although Checksum can be generated on the fly, it is not possible to avoid a certain buffering delay in the packet construction. Moreover, two different delays need to be considered at the decoder side. These consist of the waiting time for Checksum packets to arrive (and thus to be able to decode the H.264/AVC file); and in case of errors on the Checksum packets, the retransmission delay.

Both overhead and delay are directly related at the encoder side. In order to reduce the delay the frequency of Checksum transmitted packets can be increased, although this causes an increase in the overhead. The delay due to retransmissions on the decoder side is complicated to control, nevertheless, the buffering time for decoding can be reduced by smartly synchronizing the data path (transmission of the H.264/AVC packets) and the control path (transmission of the Checksum packets). In this document a simple approach just considering layering overhead is used. Even though, a detailed analysis of the protocol configurations in order to have an equilibrium between overhead and delay, and a further evaluation through simulation should be performed in order to define the optimal format packet and periodicity. This last part is left for further investigation.

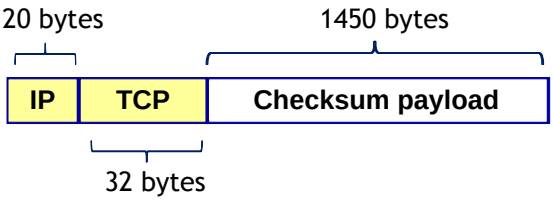


Figure 4.3: Checksum packet format. Encapsulation over IP and TCP.

The chosen implementation considers the data to be directly encapsulated in TCP over IP packets, by dividing the total number of Checksum bits by the MTU and thus obtaining the number of packets. The packet format is shown in Figure 4.3. In this case, the MTU (Maximum Transfer Unit) is that of an Ethernet network, thus 1500 bytes, although a final value of 1450 bytes of payload is adopted. Typical headers values, of 20 bytes for IP and 32 bytes for TCP are considered. This values are used on Subsection 4.3.2 for the rate calculation in the distortion over rate simulations. This implementation is too optimistic in terms of overhead, thus the maximum TCP

payload size is used; but results pessimistic in terms of delay at the decoder, because the Checksum information of several frames is grouped together.

4.2 Implementation and performance

In the presented implementation, the Checksum values are generated from a trace file of an already encoded video. Considering this, no modifications need to be done at the encoder side. Nevertheless, an implementation "on the fly" is totally feasible; this, together with a detailed analysis of the introduced delay, is left for further investigation.

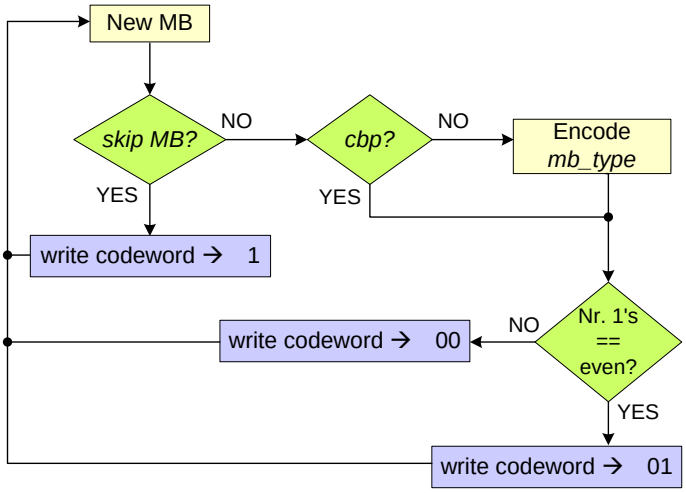


Figure 4.4: Process for the Checksum file generation, values generation from trace file

The generation of the Checksum values is achieved by processing the trace file with a simple MATLAB®program, that outputs a text file with the result. The block diagram describing the generation process is shown in Figure 4.4. The information for each macroblock is read, if the MB is skipped, the codeword "1" is written. When the MB is not skipped, the parity of the `coded_block_pattern` is coded; if the `coded_block_pattern` field is not present in the MB, the same procedure is performed for the `mb_type` field.

The process for the decoding is shown in Figure 4.5. While decoding the H.264/AVC file, the decoder reads the textfile with the Checksum values. For each MB one Checksum bit is read, in case it is "1", the decoder checks if the MB is said to be skipped on the H.264/AVC file, if they differ, an error is detected. When the first read value is a "0", a second bit is read. In this case, if the `coded_block_pattern` is present, the parity is checked; if the `coded_block_pattern` is not present, then the parity

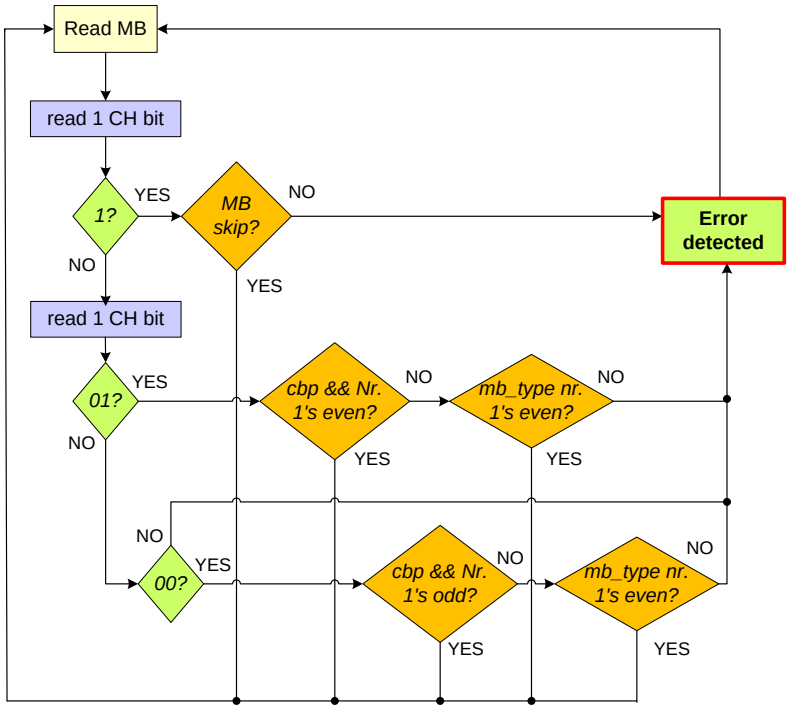


Figure 4.5: Process for Checksum decoding

of the `mb_type` is checked; if some incongruence is found, an error is detected and signalized.

4.3 Simulations and results

In this section an evaluation of the results obtained through simulation is performed. The simulation parameters are described in Section 1.6. Due to timing issues, the evaluation for Checksum is just performed for 30 f/s. First, an evaluation in terms of degradation (Y-PSNR) depending on the BER and depending on time, is performed. Then, results of distortion over bitrate are presented. Finally, an analysis of the results obtained for error detection probability and for error detection delay is performed. In all the sections, the obtained results are compared with the obtained for Watermarking.

4.3.1 Distortion evaluation

Considering different values, and for 30 f/s, the distortion over BER for Checksum compared with Syntax check alone is plotted in Figure 4.6. It can be observed, that the Checksum improves the performance obtained for Syntax Check alone; the major improvement is achieved for a BER of 10^{-5}.

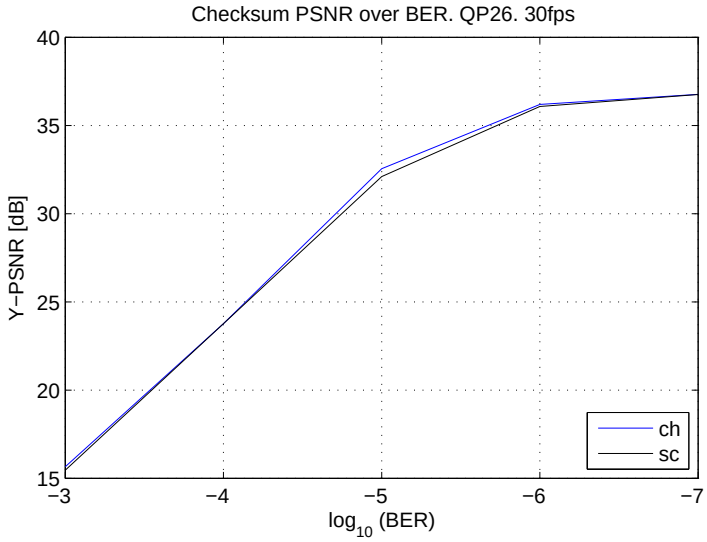


Figure 4.6: Checksum distortion over BER at the decoder.

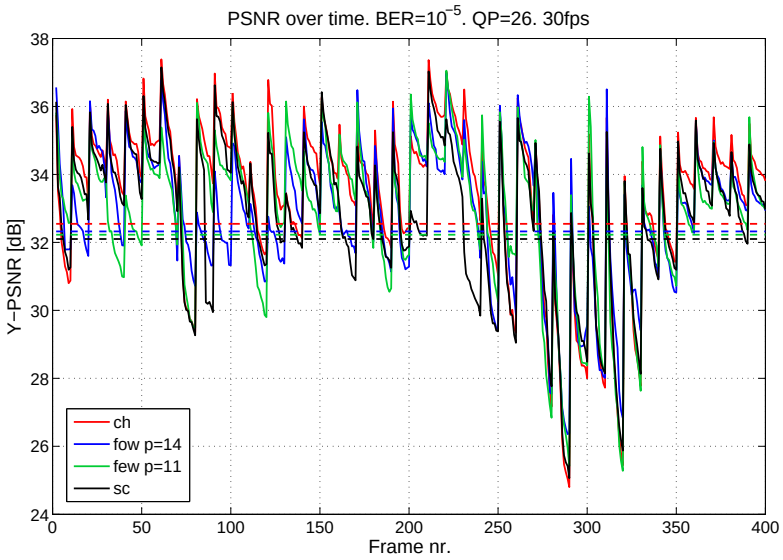


Figure 4.7: Distortion over time. BER=10^{-5}. Comparison of Syntax check, FEW $p = 11$, FOW $p = 14$ and Checksum

A comparison, plotting distortion over time for a BER of 10^{-5}, for Checksum, Watermarking and Syntax check is shown in Figure 4.7. In this case, the best results obtained for a BER= 10^{-5} for FEW and FOW are plotted; corresponding to FEW with $p = 11$ and FOW with $p = 14$. In this case, the quality achieved by Checksum overcomes the other methods; and it is followed by FOW and FEW.

4.3.2 Distortion over bitrate analysis

The use of Checksum implies an increase on the amount of information sent, for this reason it is specially interesting to perform a distortion over bitrate analysis. With the use of Checksum an extra amount of information (Checksum values plus headers associated to the Checksum packets) needs to be computed in order to provide real result. The bitrate provided by the JM is computed as shown in Equation 4.1.

$$\text{Bitrate} = \frac{\text{Total number of bits} \times \text{Frame rate}}{\text{Number of frames}} \tag{4.1}$$

Considering Equation 4.1, the real bitrate can be defined as:

$$\text{Real bitrate} = \frac{(\text{H.264/AVC bits} + \text{Checksum bits} + \text{IP/TCP headers}) \times \text{Frame rate}}{\text{Number of frames}} \tag{4.2}$$

Table 4.2: Overhead size considering Checksum information

QP	Bits video	Bits Checksum	Nr. IP packets	Bits total overhead
20	8876360	77115	7	84112
26	3791528	71775	7	84112
30	2108056	67250	6	72096

In order to calculate the bitrate as stated in Equation 4.2, the number of bits of the H.264/AVC file, Checksum file, and IP/TPC headers is computed. The values are summarized in Table 4.2. These values are obtained considering the "foreman.yuv" of 400 frames encoded sequence, payload of 1450 bytes for the Checksum packets and IP/TCP packets with 52 bytes of headers each. The field "Bits total overhead" considers the sum of the bits due to Checksum and the bits due to headers, for each QP.

Table 4.3: Checksum rates comparison

QP	Ideal rate (kb/s)	Real rate (kb/s)	Increase (%)
20	665.73	672.04	0.95%
26	284.36	290.67	2.21%
30	158.10	163.51	3.42%

The differences between the ideal and the real bitrates are summarized on Table 4.3. The ideal rate is the one obtained directly from the encoder, thus exactly the same that is obtained for Syntax check alone; whereas the real rate, considers the overhead introduced by the Checksum values and their encapsulation in IP/TCP packets.

Figure 4.8 shows the difference between the ideal and real distortion over bitrate at the decoder, for a transmission with errors and a BER of 10^{-5}. It is shown that the differences are not very significative, and for this reason, the remaining distortion over bitrate results consider the ideal case.

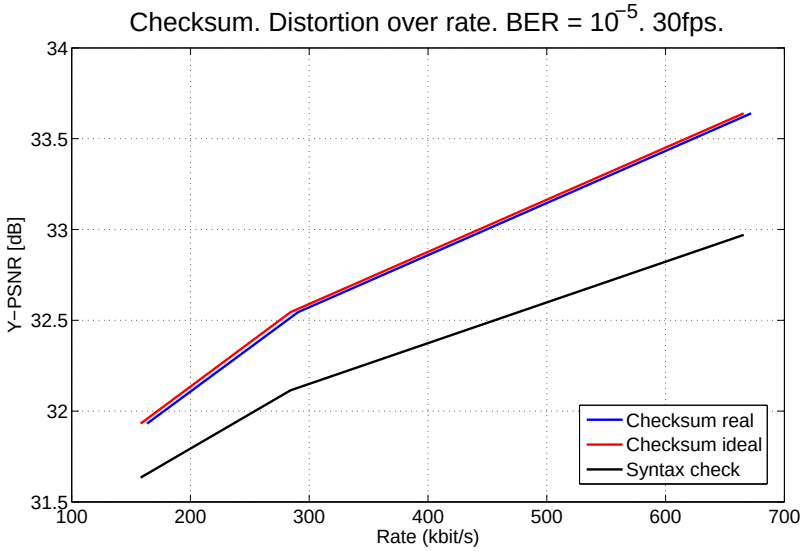


Figure 4.8: Distortion over bitrate. Ideal and Real Checksum compared with Syntax check

Figure 4.9 compares the distortion over bitrate for Checksum, Watermarking and Syntax check, for values of BER $= 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}$. The chosen Watermarking values are those that achieved the best results for FEW and FOW, depending on the BER. For a BER of 10^{-3} the best results are achieved for FEW with $p = 2$ and Checksum, where FEW performs better for rates between 150-180kb/s and 280-350kb/s; when the BER is 10^{-4} the behavior is similar, in general Checksum improves the performance of the rest of the methods, but for the range between 280 and 350 kb/s, FEW with $p = 2$ achieves better results. For 10^{-5}, Checksum achieves a great improvement in the quality, improving in 0.67dB the results of Syntax check alone. Finally, when the BER is equal to 10^{-6}, Checksum outperforms the results of Watermarking for bit-rates higher than 400 kb/s; for lower rates, FOW with $p = 14$ behaves slightly better than Syntax check alone.

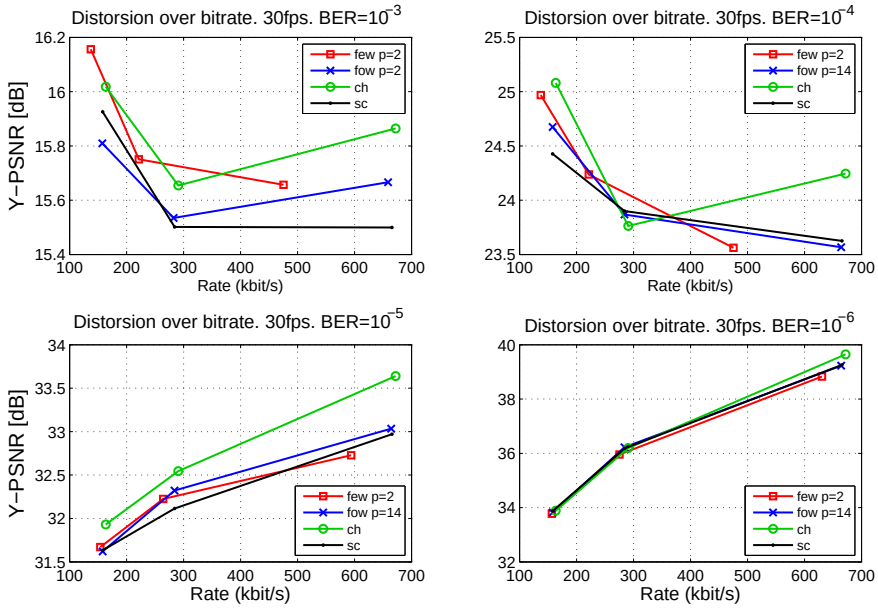


Figure 4.9: Distortion over bitrate. Comparison of Checksum, Watermarking and Syntax check for different BERs

4.3.3 Error detection probability and error detection delay

The error detection probability for the Checksum method (Syntax check together with a cheksum computation) is computed in the same way as done in the Watermarking. Error detection probability for Checksum, is compared with the best results obtained for FEW and FOW with 30 f/s and QP=26. This is summarized in Table 4.4.

Table 4.4: Error detection probability. Comparison of Checksum and Watermarking (30f/s)

Error detection probabilities (QP=26)	
Checksum	64.54%
FEW $p = 2$	65.6%
FEW $p = 7$	65.33%
FOW $p = 2$	57.96%
FOW $p = 7$	52.48%

The error detection probability achieved for the Checksum method is comparable to the best error detection probabilities achieved with Watermarking. FEW with $p = 2$ and $p = 7$ achieves an error detection probability of about 65.5%, whereas the obtained error detection probability for Checksum is around the 64.5%.

In order to compare the error detection delay, the cumulative distribution function (CDF) is used. In Figure 4.10 the best CDFs of Watermarking (when $p = 2$) are compared with the results obtained for Checksum, and with Syntax check alone. The simulations are performed with QP=26 and 30 f/s; and results for P frames are shown.

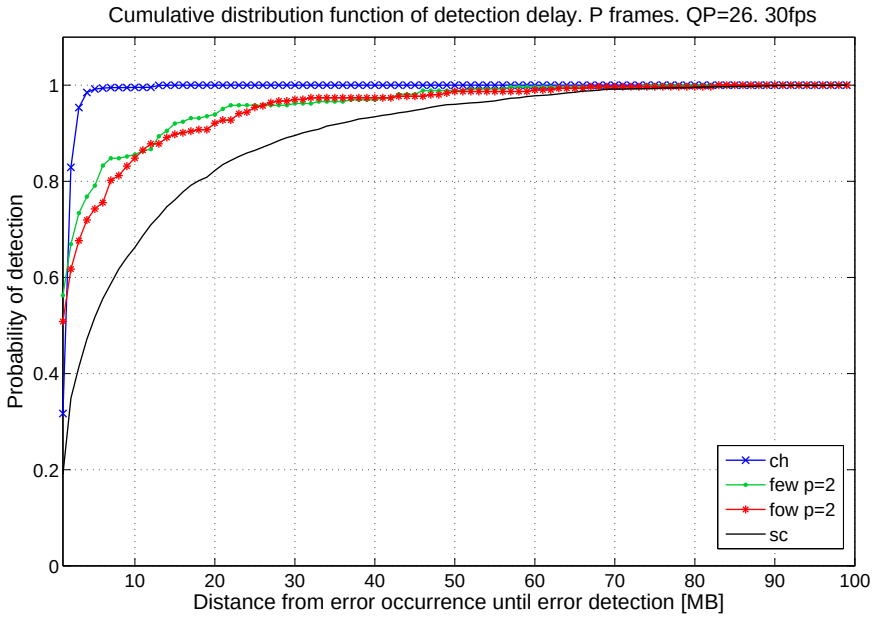


Figure 4.10: Cumulative distribution function (CDF) of the detection delay. Comparison of Checksum, Watermarking and Syntax check alone with 30f/s.

The obtained results show a great improvement of Checksum over the rest of the methods. Watermarking is able to provide an 85% of error detections with detection distance below 10 MBs, compared with the 65% of Syntax check alone. With Checksum, 99% of the errors detections are performed within the 10 first macroblocks. Considering this last experiment, it is possible to state that Checksum results are comparable to the best results obtained for Watermarking, with the difference that Checksum does not alter the video information. Checksum is able to overcome the rest of the methods in most of the simulations, specially in the case of error detection delay. This points out Checksum as a good error detection method candidate to be considered.

Chapter 5

Conclusions

The presented master thesis investigates different error detection methods for H.264/AVC videos. All the presented methods are combined with the already proposed Syntax check [7], e.g. "FEW" stands in this work for an implementation of a Force Even Watermarking on top of Syntax check. In particular, two types of approaches are defined, implemented and tested: Watermarking based methods and Checksum. Experiments are performed using a Binary Symmetrical Channel (BSC) considering different bit error rates, and results of received quality at the decoder (PSNR), distortion over bitrate, error detection probability and error detection delay are evaluated.

Taking as a reference the Watermarking approach presented in [8], an improved Watermarking has been implemented together with Syntax check, and tested. Apart from the Force Even Watermarking (FEW) a Force Odd Watermarking (FOW) is presented. In contrast to existing literature analyzing watermarking as an error detection method, this thesis considers transmission errors in all parts of the video stream, i.e. also in information elements other than coded coefficients. Such scenario corresponds better to typical transmission conditions for video streaming, and thus provides more realistic performance evaluation of the proposed and tested error detection methods. As an observed result, the BSC channel causes desynchronization at the decoder. This effect turned out to be especially harmful for some of the H.264/AVC video fields, concretely for the elements containing information about the skipped macroblocks (`mb_skip_run`) and the type of coefficients (`coded_block_pattern`). Considering this, a specific Checksum protecting these fields was defined. The presented Checksum method is based on a variable length coding, and it is sent out of band using a TCP connection, which can be seen as a type of nearly error free channel. For a given rate, comparison of the methods in terms of quality has been performed.

Analyzing the obtained results for Watermarking, it is possible to state that for

high values of BER, FEW is able to recover the video stream with better quality at the receiver than Syntax check alone, but fails to overcome the initial degradation for low bit error rates. The opposite behavior is observed in FOW, which is able to provide good results for low BERs. Watermarking overcomes the performance of Syntax check alone in terms of error detection probability and delay; the highest error detection probability and error detection delay for Watermarking are achieved for FEW when watermarking is performed for all AC coefficients.

The obtained results for Checksum point out that the method is comparable to the best configurations of Watermarking. Checksum is capable of overcoming the Watermarking in terms of distortion over bitrate at the decoder. In terms of error detection probability, Checksum is a 1% below the best results obtained for Watermarking; whereas it really improves the error detection delay, achieving around a 15% of improvement over the best configuration of Watermarking, and more than a 30% compared to Syntax check alone.

Taking the previous results into consideration, both Watermarking and Checksum resulted capable of overcoming the performance of Syntax check alone. More specifically, Checksum appears to be a good solution as an error detection method, because it provides an error detection probability close to the best results achieved by Watermarking (but without an initial video degradation) and strongly improves the error detection delay compared to the other methods. Nevertheless, it is important to notice that other Checksum configurations, protecting a different number or set of elements, should be tried and compared with the proposed configuration. Moreover, and in order to provide an optimal configuration of Checksum, a detailed study about the introduced delay (Checksum generation, packet encapsulation and process at the decoder) and overhead should be performed. This last point is not covered in the presented thesis and thus is left for further investigation.

Bibliography

[1] ITU-T H.264. Series H: Audiovisual and multimedia systems, Infrastructure of audiovisual services - coding of moving video. Advanced video coding for generic audiovisual services, November 2007. URL `http://www.itu.int/rec/T-REC-H.264`. [cited at p. 3, 4, 42]

[2] Olivia Nemethova. *Error Resilient Transmission of Video Streaming over Wireless Mobile Networks*. PhD thesis, Institut für Nachrichten- und Hochfrequenztechnik, 2007. [cited at p. 3, 4, 12, 20]

[3] Thomas Wiegand, Gary J. Sullivan, Gisle Bjøtegaard, and Ajay Luthra. "Overview of the H.264/AVC video coding standard". *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, July 2003. [cited at p. 3]

[4] ITU-T H.263. Series H: Audiovisual and multimedia systems, Infrastructure of audiovisual services - coding of moving video. Video coding for low bit rate communications, January 2005. [cited at p. 3]

[5] ISO/IEC-14496-2. Coding of Audio-Visual Objects, part 2: visual, 2001. [cited at p. 3]

[6] ISO/IEC. International Standard ISO/IEC 14 496-10. Information technology – Coding of audio-visual objects – part 10: Advanced Video Coding (MPEG-4 part 10), 2005. URL `http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=43058`. [cited at p. 4]

[7] Luca Superiori, Olivia Nemethova, and Markus Rupp. "Performance of a H.264/AVC Error Detection Algorithm Based on Syntax analysis". *4th Int. Conf. on Mobile Computing and Multimedia (MoMM) Yogiakarta, Indonesia,*, pages 49–58, December 2006. [cited at p. 12, 14, 19, 20, 55]

[8] Olivia Nemethova, Gonzalo Calvar Forte, and Markus Rupp. "Robust Error Detection for H.264/AVC Using Relation Based Fragile Watermarking". *Proc. of 13th Int. Conf. on Systems, Signals and Image Processing, Budapest, Hungary*, September 2006. [cited at p. 12, 13, 14, 25, 55]

[9] Minghua Chen, Yun He, and R.L.Lagendijk. "A Fragile Watermark Error Detection Scheme for Wireless Video Communications". *IEEE Transactions on Multimedia*, 7(2): 201–211, April 2005. [cited at p. 12, 19, 25]

[10] Olivia Nemethova, Jacob Canadas, and Markus Rupp. "Improved Detection for H.264 Encoded Video Sequences over Mobile Networks". *Proc of Int. Symp. on Communication Theory and Applications (ISCTA), Ambleside, UK*, July 2005. [cited at p. 13, 41]

[11] H.264/AVC Software Coordination, "Joint Model Software", ver. 13.0, availabe at http://iphome.hhi.de/suehring/tml/. [cited at p. 14]

[12] Iain Richardson. H.264 Variable Length Coding tutorial. URL `http://www.vcodex.com/h264.html`. [cited at p. 42]

List of Abbreviations

Abbreviation	Description
3GPP	3rd Generation Partnership Project
AC	Alternating Current
dB	Decibel
BSC	Binnary Symmetrical Channel
BER	Bit Error Rate
CABAC	Context-Adaptive Binary Arithmetic Coding
CAVLC	Context-Adaptive Variable-Length Coding
cbp	Coded Block Pattern
CDF	Cumulative Distribution Function
CE	Contextual Error
CH	Checksum
CIF	Common Intermediate Formate
DC	Direct Current
DCT	Discrete Cosine Transform
DVB-T	Digital Video Broadcasting - Terrestrial
EG	Exp-Golomb codeword
exp-Golomb	Exponential-Golomb code
FEW	Force Even Watermarking
FL	Fixed Length codeword
FOW	Force Odd Watermarking
GOP	Group Of Pictures
HDTV	High-Definition television
IC	Illegal codeword
IP	Internet Protocol
ISO/IEC	International Organization for Standardization / International Electrotechnical Commission
ITU	International Telecommunication Union
JM	Joint Model
JVT	Joint Video Team
KLT	Karhunen Leve Transform
MB	Macroblock

Abbreviation	Description
MPEG	Motion Picture Expert Group
MSE	Mean Square Error
MTU	Maximum Transfer Unit
NAL	Network Abstraction Layer
NALU	Network Abstraction Layer Unit
NRI	NAL Reference Identication
OR	Out of Rage codeword
Prev.Conc.	Concealment of macroblocks previous to the error detection
PSNR	Peak to Signal-to-Noise Ratio
QCIF	Quarter Common Intermediate Format
QP	Quantization Parameter
QVGA	Quarter Video Graphics Array
RBW	Relation Based Watermarking
RGB	Red, Green, Blue
RTP	Real Time Protocol
SH	Slice Header
TCP	Transmission Control Protocol
TE	Tabled codeword
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
VCL	Variable Coding Length
VGA	Video Graphics Array
VL	VLC level codeword
WM	Watermarking method
xDSL	Digital Subscriber Line technologies
Y-PSNR	Luminance PSNR

List of Figures

List of Tables