Enhanced Quality Reconstruction of Erroneous Video Streams Using Packet Filtering Based on Non-desynchronizing Bits and UDP Checksum-Filtered List Decoding

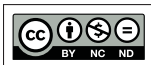
by

Firouzeh GOLAGHAZADEH

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
Ph.D.

MONTREAL, MAY 29TH, 2019

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC



BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Stéphane Coulombe, Thesis Supervisor
Department of Software Engineering and IT, École de technologie supérieure

Mr. François-Xavier Coudoux, Thesis Co-supervisor
Department of Electrical Engineering and Computer Science, Université Polytechnique Hauts-de-France

Mr. Patrick Corlay, Thesis Co-supervisor
Department of Electrical Engineering and Computer Science, Université Polytechnique Hauts-de-France

Mr. Ammar Kouki, President of the Board of Examiners
Department of Electrical Engineering, École de technologie supérieure

Mr. Carlos Vázquez, Member of the jury
Department of Software Engineering and IT, École de technologie supérieure

Mr. Marco Cagnazzo, External Independent Examiner
LTCI, CNRS, Télécom Paris Tech, Université Paris-Saclay

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON APRIL 10TH, 2019

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGEMENTS

I owe my deepest gratitude to my amazing husband Reza, who has always supported me and stood by me in my difficult moments. Reza, you gave me the confidence in myself because you believe in me and care about me. I would also like to extend my deepest gratitude to my brother and my sisters for their constant encouragements to pursue my academic goals. I'm deeply indebted to my parents who have sacrificed their own dreams to help me towards mine. I know this was always your one day dream, so, Mom, Dad, this is for you.

I cannot find the best words to express my thanks to my dear supervisor, Professor Stéphane Coulombe, for all his valuable advices, continuous encouragements during my long-term PhD studies. This accomplishment would not have been possible without his mentorship and invaluable involvement. Dear Professor Coulombe, I consider myself as a lucky person for the opportunity I had to work with you and I always consider it as a great honor.

I would like to express my gratitude to my co-supervisors, Professors François-Xavier Coudoux and Patrick Corlay, for their collaboration in the project and also their hospitality during my internship in France. Their participation in this project made my PhD experience richer technically and culturally.

I would like also to thank the members of my dissertation committee: Professors Ammar Kouki, Carlos Vázquez and Marco Cagnazzo for their time reviewing my thesis.

I would like to thank all my friends in our office, the Video Optimization lab, for their help and support: François, Luc, Jean-François, Reza, Esmaeil, Deepa, Neda and Nick.

Finally, I would like to thank, the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Fonds de recherche du Québec – Nature et technologies (FRQNT) for their financial support.

Qualité améliorée de la reconstruction des flux vidéos avec erreurs à l'aide d'un filtrage de paquets basé sur des bits non désynchronisants et une approche de décodage en liste filtrée par la somme de contrôle au niveau UDP

Firouzeh GOLAGHAZADEH

RÉSUMÉ

Les dernières normes de codage vidéo, telles que H.264 et H.265, sont extrêmement vulnérables dans les réseaux sujets aux erreurs. En raison de leurs outils sophistiqués de prédiction spatiale et temporelle, l'effet d'une erreur ne se limite pas à la zone erronée, mais il peut facilement se propager spatialement aux blocs voisins et temporellement aux images suivantes. Ainsi, les paquets vidéos reconstruits au décodeur peuvent présenter une dégradation significative de la qualité visuelle. La dissimulation d'erreurs et les corrections d'erreurs sont deux mécanismes qui ont été développés pour améliorer la qualité des trames reconstruites en présence d'erreurs.

Dans la plupart des approches existantes de dissimulation d'erreurs, les paquets corrompus sont ignorés et seules les informations correctement reçues des zones environnantes (dans l'espace et/ou dans le temps) sont utilisées pour récupérer la zone erronée. Cela est dû au fait qu'il n'existe aucun mécanisme de détection d'erreur parfait pour identifier correctement les blocs reçus dans un paquet corrompu, et aussi au problème de désynchronisation provoqué par les erreurs de transmission sur le code à longueur variable (VLC). Mais, comme de nombreuses études l'ont montré, les paquets corrompus peuvent contenir des informations précieuses pouvant être utilisées pour reconstruire correctement la zone perdue (par exemple, lorsque l'erreur est située à la fin d'une tranche).

D'autre part, les approches de correction d'erreur, telles que le décodage en liste, exploitent les paquets corrompus pour générer plusieurs paquets candidats transmis à partir du paquet reçu corrompu. Ils sélectionnent ensuite, parmi ces candidats, celui qui présente la probabilité la plus élevée d'être le paquet transmis sur la base des informations souples disponibles (par exemple, le rapport log-vraisemblance (LLR) de chaque bit). Cependant, les approches de décodage de liste souffrent d'un grand espace de solutions de paquets transmis candidats. Cela est aggravé lorsque les informations logicielles ne sont pas disponibles au niveau de la couche d'application; un scénario plus réaliste en pratique. En effet, comme on ignore quels bits ont des probabilités plus élevées d'avoir été modifiés au cours de la transmission, les paquets reçus candidats ne peuvent être classés par vraisemblance.

Dans cette thèse, nous proposons différentes stratégies pour améliorer la qualité des paquets reconstruits qui ont été légèrement endommagés lors de la transmission (par exemple au plus une erreur par paquet). Nous proposons d'abord un mécanisme simple mais efficace pour filtrer les paquets endommagés afin de conserver ceux qui sont susceptibles de conduire à une très bonne reconstruction et d'éliminer les autres. Cette méthode peut être utilisée en complément à la plupart des méthodes de dissimulation existantes pour améliorer leurs performances. La

méthode est basée sur le nouveau concept de bits non désynchronisants (NDBs) définis dans le contexte d'une séquence compressée à l'aide de codes à longueur variable (CAVLC) en H.264, en tant que bit dont l'inversion ne provoque pas de désynchronisation au niveau du flux binaire ni ne modifie le nombre de macroblocs décodés. Nous établissons que, sur des trains de bits codés typiques, les NDBs constituent environ un tiers (environ 30%) d'un train de bits et que l'effet sur la qualité visuelle du renversement de l'un d'eux dans un paquet est généralement insignifiant. Dans la plupart des cas (90%), la qualité du paquet reconstruit lors de la modification d'un NDB individuel est presque identique à celle du paquet intact. Nous démontrons ainsi que conserver, sous certaines conditions, un paquet corrompu en tant que candidat pour la zone perdue peut fournir une meilleure qualité visuelle que les méthodes de dissimulation. Nous proposons enfin un cadre de décodage non désynchronisé, qui conserve un paquet corrompu, à condition de ne pas provoquer de désynchronisation et de ne pas modifier le nombre de macroblocs attendus. Le cadre peut être combiné avec la plupart des approches de dissimulation actuelles. L'approche proposée est comparée à la copie de trame (FC) du logiciel JM (Joint Model) (JM-FC) et à une approche de dissimulation de pointe utilisant le mécanisme de l'algorithme d'adaptation de limite spatiotemporelle (STBMA), dans le cas d'un bit d'erreur, et fournit en moyenne respectivement un gain de 3,5 dB et 1,42 dB.

Nous proposons ensuite une nouvelle approche de décodage en liste appelée CFLD (checksum-filtered list decoding)) qui permet de corriger un paquet au niveau du train de bits en exploitant la valeur de somme de contrôle du protocole de datagramme utilisateur (UDP) du destinataire. L'approche proposée permet d'identifier les emplacements possibles d'erreurs en analysant le modèle de la somme de contrôle UDP calculée sur le paquet corrompu. Cela permet de réduire considérablement le nombre de paquets candidats transmis par rapport aux approches classiques de décodage en liste, en particulier lorsqu'aucune information souple n'est disponible. Lorsqu'un paquet composé de N bits contient un seul bit erroné, au lieu de considérer les paquets candidats au nombre de N, comme c'est le cas dans les approches de décodage en liste conventionnelles, l'approche proposée prend en compte environ $N/32$ candidats, entraînant une réduction de 97% du nombre de candidats. Cette réduction peut atteindre 99,6% dans le cas de deux bits erronés. Les performances de la méthode sont évaluées à l'aide de H.264 et H.265. Nous montrons que, dans le cas d'une séquence codée H.264, en moyenne, l'approche CFLD est capable de corriger le paquet 66% du temps. Elle offre également un gain de 2,74 dB sur JM-FC et des gains de 1,14 dB et 1,42 dB sur STBMA et un décodage par vraisemblance maximale en sortie dure (HO-MLD), respectivement. De plus, dans le cas de HEVC, l'approche CFLD corrige le paquet corrompu 91% du temps et offre des gains de 2,35 dB et 4,97 dB sur notre mise en œuvre de la dissimulation de FC dans le logiciel de modèle de test HEVC (HM-FC) pour les séquences des classes B (1920×1080) et C (832×480), respectivement.

Mots-clés: transmission vidéo, H.264, high efficiency video coding (HEVC), H.265, eléments de syntaxes, bit non désynchronisant (NDB), dissimulation d'erreurs, correction d'erreur vidéo, décodage en liste, somme de contrôle, protocole de datagramme utilisateur (UDP), checksum-filtered list decoding (CFLD)

Enhanced quality reconstruction of erroneous video streams using packet filtering based on non-desynchronizing bits and UDP checksum-filtered list decoding

Firouzeh GOLAGHAZADEH

ABSTRACT

The latest video coding standards, such as H.264 and H.265, are extremely vulnerable in error-prone networks. Due to their sophisticated spatial and temporal prediction tools, the effect of an error is not limited to the erroneous area but it can easily propagate spatially to the neighboring blocks and temporally to the following frames. Thus, reconstructed video packets at the decoder side may exhibit significant visual quality degradation. Error concealment and error corrections are two mechanisms that have been developed to improve the quality of reconstructed frames in the presence of errors.

In most existing error concealment approaches, the corrupted packets are ignored and only the correctly received information of the surrounding areas (spatially and/or temporally) is used to recover the erroneous area. This is due to the fact that there is no perfect error detection mechanism to identify correctly received blocks within a corrupted packet, and moreover because of the desynchronization problem caused by the transmission errors on the variable-length code (VLC). But, as many studies have shown, the corrupted packets may contain valuable information that can be used to reconstruct adequately of the lost area (e.g. when the error is located at the end of a slice).

On the other hand, error correction approaches, such as list decoding, exploit the corrupted packets to generate several candidate transmitted packets from the corrupted received packet. They then select, among these candidates, the one with the highest likelihood of being the transmitted packet based on the available soft information (e.g. log-likelihood ratio (LLR) of each bit). However, list decoding approaches suffer from a large solution space of candidate transmitted packets. This is worsened when the soft information is not available at the application layer; a more realistic scenario in practice. Indeed, since it is unknown which bits have higher probabilities of having been modified during transmission, the candidate received packets cannot be ranked by likelihood.

In this thesis, we propose various strategies to improve the quality of reconstructed packets which have been lightly damaged during transmission (e.g. at most a single error per packet). We first propose a simple but efficient mechanism to filter damaged packets in order to retain those likely to lead to a very good reconstruction and discard the others. This method can be used as a complement to most existing concealment approaches to enhance their performance. The method is based on the novel concept of non-desynchronizing bits (NDBs) defined, in the context of an H.264 context-adaptive variable-length coding (CAVLC) coded sequence, as a bit whose inversion does not cause desynchronization at the bitstream level nor changes the number of decoded macroblocks. We establish that, on typical coded bitstreams, the NDBs constitute about a one-third (about 30%) of a bitstream, and that the effect on visual quality of

flipping one of them in a packet is mostly insignificant. In most cases (90%), the quality of the reconstructed packet when modifying an individual NDB is almost the same as the intact one. We thus demonstrate that keeping, under certain conditions, a corrupted packet as a candidate for the lost area can provide better visual quality compared to the concealment approaches. We finally propose a non-desync-based decoding framework, which retains a corrupted packet, under the condition of not causing desynchronization and not altering the number of expected macroblocks. The framework can be combined with most current concealment approaches. The proposed approach is compared to the frame copy (FC) concealment of Joint Model (JM) software (JM-FC) and a state-of-the-art concealment approach using the spatiotemporal boundary matching algorithm (STBMA) mechanism, in the case of one bit in error, and on average, respectively, provides 3.5 dB and 1.42 dB gain over them.

We then propose a novel list decoding approach called checksum-filtered list decoding (CFLD) which can correct a packet at the bit stream level by exploiting the receiver side user datagram protocol (UDP) checksum value. The proposed approach is able to identify the possible locations of errors by analyzing the pattern of the calculated UDP checksum on the corrupted packet. This makes it possible to considerably reduce the number of candidate transmitted packets in comparison to conventional list decoding approaches, especially when no soft information is available. When a packet composed of N bits contains a single bit in error, instead of considering N candidate packets, as is the case in conventional list decoding approaches, the proposed approach considers approximately $N/32$ candidate packets, leading to a 97% reduction in the number of candidates. This reduction can increase to 99.6% in the case of a two-bit error. The method's performance is evaluated using H.264 and high efficiency video coding (HEVC) test model software. We show that, in the case H.264 coded sequence, on average, the CFLD approach is able to correct the packet 66% of the time. It also offers a 2.74 dB gain over JM-FC and 1.14 dB and 1.42 dB gains over STBMA and hard output maximum likelihood decoding (HO-MLD), respectively. Additionally, in the case of HEVC, the CFLD approach corrects the corrupted packet 91% of the time, and offers 2.35 dB and 4.97 dB gains over our implementation of FC concealment in HEVC test model software (HM-FC) in class B (1920×1080) and C (832×480) sequences, respectively.

Keywords: Video Transmission, H.264, high efficiency video coding (HEVC), H.265, Syntax Elements, non-desynchronizing bit (NDB), Error Concealment, Video Error Correction, List Decoding, Checksum, user datagram protocol (UDP), checksum-filtered list decoding (CFLD)

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

AR	Auto Regression
ASO	Arbitrary Slice Ordering
AVC	Advanced Video Coding
BEE	Bit Error Event
BMA	Boundary Matching Algorithm
CABAC	Context-Adaptive Binary Arithmetic coding
CAVLC	Context-Adaptive Variable-Length Coding
CBP	Coded Block Pattern
CFLD	Checksum-Filtered List Decoding
CPT	Checksum Pattern Type
CTU	Coding Tree Unit
DCT	Discrete Cosine Transform
DMVE	Decoder Motion Vector Estimation
EGC	Exponential Golomb Code
ESLD	Exhaustive Search List Decoding
FC	Frame Copy
FMO	Flexible Macroblock Ordering
HEVC	High Efficiency Video Coding

HM	HEVC Test Model
HO	Hard Output
HO-MLD	Hard Output Maximum Likelihood Decoding
I	Intra
IoT	Internet of Things
IP	Internet Protocol
JM	Joint Model
JSCD	Joint Source Channel Decoding
KDE	Kernel Density Estimation
LCU	Large Coding Unit
LLR	Log-Likelihood Ratio
LSB	Least Significant Bit
MAD	Mean Absolute Differences
MAP	Maximum a Posteriori
MB	Macroblock
MLD	Maximum Likelihood Decoding
MSB	Most Significant Bit
MV	Motion Vector
NAL	Network Abstraction Layer

NDB	Non-Desynchronizing Bit
P	Inter
PCA	Principal Component Analysis
PDE	Partial Differential Equation
PPS	Picture Parameter Set
PSNR	Peak Signal-to-Noise Ratio
QP	Quantization Parameter
RTP	Real-Time Transport Protocol
SAD	Sum of Absolute Differences
SDMCB	Sum of Distributed Motion-Compensated Blockiness
SGC	Signed-Exponential Golomb Code (EGC)
SO	Soft Output
SO-MLD	Soft Output Maximum Likelihood Decoding
SO/HO-MLD	Soft/Hard Output Maximum Likelihood Decoding
SPR	Spiral-like Pixel Reconstruction
SPS	Sequence Parameter Set
SSIM	Structural Similarity Index Measurement
STBMA	Spatiotemporal Boundary Matching Algorithm
TC	*TotalCoeff*
TCP	Transmission Control Protocol

TZ	*total_zeros*
UDP	User Datagram Protocol
UDP-Lite	Lightweight User Datagram Protocol
UGC	Unsigned-Exponential Golomb Code (EGC)
VLC	Variable-Length Code

LISTE OF SYMBOLS AND UNITS OF MEASUREMENTS

w_i	Reliability factor in MAD calculation
α	Weighting factor factor in STBMA cost function
S^*	Likeliest packet to the received corrupted packet
S_r	Received corrupted packet
S_t	Hypothetically transmitted slice
H	A set of all hypothetically transmitted slices (S_t)
$c_{i,j}$	The j-th codeword in i-the path
$s_{\{t,i\}}$	The i-th syntax element in a hypothetically transmitted slice (S_t)
X_i	The i-th bit in INFO part of EGC
Y_i	The i-th transform coefficient
V	A set of 16-bit values in hexadecimal format in an Abelian group
e	Identity element in Abelian group
W_i	The i-th word of the UDP packet
$w_{i,c}$	The c-th bit of word W_i
$\overline{W}_i$	Inverse of word W_i
$\widehat{W}$	Received version of word W
W_{cs}	Checksum value in the checksum field
$\widehat{W}_{cs}$	Received checksum value in the checksum field
C_T	Transmission side's checksum

C_R	Receiver side's checksum
TZ	Total number of bit 0 in the packet
TO	Total number of bit 1 in the packet
nz_c	Number of bit 0 in column c
no_c	Number of bit 1 in column c
ρ	Channel residual bit error rate

CHAPTER 1

INTRODUCTION

In recent years, digital video communication, especially in the form of high quality content delivery, has attracted considerable attention in a wide variety of application environments, such as mobile video streaming, video conferencing, telepresence, real-time monitoring, etc. According to the report published by Cisco (Cisco, 2017), mobile video traffic will grow ninefold between 2016 and 2021 reaching 38 exabytes[1] per month. It has been estimated that by 2021, mobile video traffic will comprise more than 78% of the mobile data traffic (see Figure 1.1). This is mostly due to the introduction of the high definition video content streaming (Cisco, 2018).

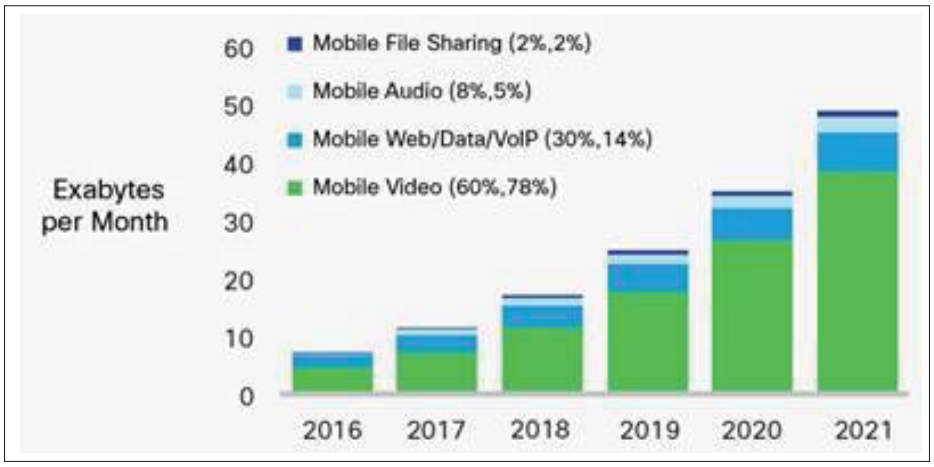


Figure 1.1 Mobile traffic growth between 2016 and 2021, reported by Cisco (Cisco, 2017).

Because of the huge size of a raw video file and other existing restrictions related to data storage, processing power, transmission cost, and communication speed, compression is a vital step in efficient processing of the video streams. Generally a codec, which stands for the compression and decompression technique, is responsible for reducing video file sizes while maintaining a desired level of quality. H.264/MPEG advanced video coding (AVC) (Interna-

[1] 1 exabyte = 1 billion gigabytes. This represents approximately 245 million DVDs of 4.38GB each.

tional Telecommunications Union, 2003) and high efficiency video coding (HEVC) (ISO/IEC JTC 1/SC 29/WG 11, 2013) are the two well-known video coding standards. In fact, H.264 is currently the most widely deployed video codec in a variety of applications and networks such as in broadcasting, streaming video sources, video conferencing, etc, due to its ability to provide a good compromise between coding efficiency and computational complexity (ITU-T-StudyGroups). HEVC, also known as H.265, is a very complex compression method that can provide better coding efficiency (50% bit-rate reduction for the same quality compared to H.264) but requires more computations and thus advanced hardware for deployment.

1.1 Problem statement

The high compression performance of the current video coding standards and moreover, the motion-compensated prediction techniques employed in the codecs, make the compressed video streams more vulnerable to transmission errors. In the real world of noisy channel communications (e.g. mobile networks), transmission errors are inevitable, which for video transmissions, leads to an unpleasant quality reduction of the reconstructed video sequences. For instance, a single-bit error in variable-length code (VLC) may cause the decoder to lose its synchronization with the corresponding encoder, and consequently decode incorrect code-words which eventually result in spatial error propagation. Even worse, because of the motion compensation techniques used in compression, the error can propagate from one frame to consecutive ones, and lead to severe visual artifacts (Tan *et al.*, 2008). Figure 1.2 illustrates the error propagation on four consecutive frames of an H.264 coded sequence.

Various error control mechanisms have been proposed to combat the visual quality degradation caused by transmission errors (Wang *et al.*, 2002). Among them, retransmission is one of the basic mechanisms for providing reliable communications. However, it is rarely used in real-time conversational or broadcasting/multicasting applications due to the added delay or lack of feedback channel involved (Sullivan & Wiegand, 2005). Error resilience, as an another approach, generally injects redundancies to the bitstream during the source encoding to make the streams more robust against the transmission errors. This may aid the decoder to

Figure 1.2 Illustration of error propagation (spatially and temporally).

better deal with the loss of information. It is worth noting that the H.264 standard includes new resilience tools like flexible macroblock orderings (FMO), arbitrary slice ordering (ASO), redundant slices, data partitioning, etc, which can be used to protect the compressed bitstream against the burst error or error propagation (e.g. enabling redundant slices tool). However, all error resilience mechanisms reduce the coding efficiency (by adding redundant bits while the compression goal is to remove redundancies) or sacrifice bit rate, especially when there is no transmission error (Wang *et al.*, 2000; Xiao *et al.*, 2013). On the other hand, error concealment and error correction approaches are the two post-processing mechanisms that strive to alleviate the effect of the transmission errors at the decoder side. Unlike the resilience techniques, They have the advantages of neither consuming extra bandwidth nor introducing retransmission delays.

Error concealment attempts to reconstruct the erroneous area by using the information of correctly received neighbouring areas. Most existing error concealment techniques are based on utilizing the inherent correlation among adjacent pixels. This can be performed by exploiting the spatial correlation (Liu *et al.*, 2015) between neighboring pixels in a specific area (block or slice or frame), the temporal correlation (Lie *et al.*, 2014) between consecutive frames, or combination of both correlations (Zhou *et al.*, 2017). The advantage of using concealment approaches is that they utilize the visual quality parameters (i.e. smoothness properties, boundary matching criteria) directly during the concealment process which makes the result more appealing for humans. However, the error concealment performance may suffer when lost areas have

less correlation (spatial or temporal) with the correctly received surrounding areas or when the lost regions are large.

Packets partially damaged due to transmission errors may contain valuable information that can be used to enhance the visual quality of the reconstructed video (Superiori *et al.*, 2006; Trudeau *et al.*, 2011). This is the case when the error occurs at the end of the packet or when the residual bit error rate (after channel decoding) is low. A standardized transport protocol, lightweight user datagram protocol (UDP-Lite), allows partially damaged packets to be delivered to the application layer instead of having them discarded upon reception (Larzon *et al.*, 2004). However, the application layer, in this case the decoder, is responsible to decide whether to keep those corrupted packets or discard them.

Utilizing the corrupted packet in concealment is always a challenge. The idea behind partial concealment, as shown in Figure 1.3, is to decode the uncorrupted macroblocks (MBs) within the corrupted packets, i.e. the MBs that are before the error location, and perform the concealment only on the others (Superiori *et al.*, 2006). Because of the way that video is encoded, the errors can not be detected at the actual location of their occurrence. Sometimes an error on a VLC can generate another valid, but wrong, codeword which may be detected later on by violating the following syntaxes or causing semantic errors. Such a distance between the error occurrence and error detection location can sometimes lead to a severe distortion on the reconstructed frame. Therefore, most existing error concealment approaches prefer to discard the corrupted packet (containing corrupted and uncorrupted MBs) and only utilize the correctly received information to reconstruct the lost area. Thus, error concealment treats a corrupted packet as it has been lost. In practice, network congestion results in packet loss, while wireless signal attenuation, fading, etc., result in corrupted packets. However, corrupted and lost packets must be handled differently.

In contrast, the goal of the error correction approaches is to utilize the corrupted packet and repair the errors directly in the bitstream [2]. In list decoding correction approaches, this is realized

[2] In the context of this thesis, by the term of bitstream, we mean packet bitstream or packet and the terms have used interchangeably.

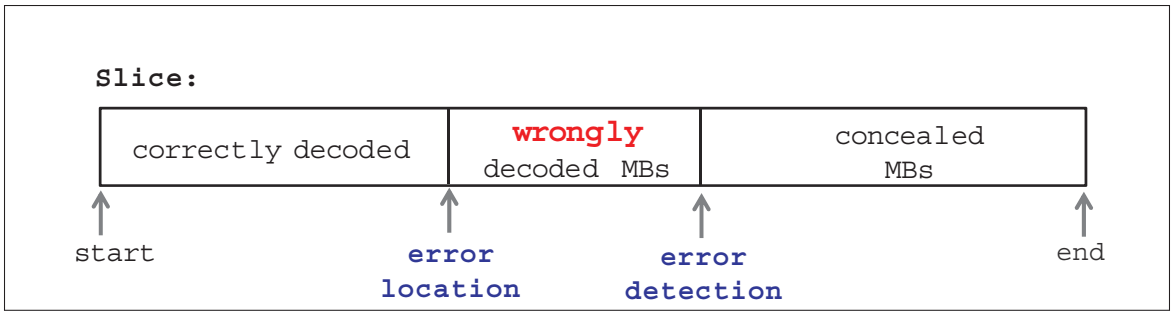


Figure 1.3 Partial error concealment (Superiori *et al.*, 2006).

by generating multiple candidate transmitted packets based on the received corrupted packet (by flipping the bits) and selecting one of them based on some constraints. This is possible due to the fact that for any finite length of a packet, there are a limited number of decodable candidates. However, the huge choice of error positions, i.e. high number of candidate bitstreams, and also defining the constraints to determine the best candidate bitstream are the two major challenges in correction approaches (Caron & Coulombe, 2015). Generally, in most cases, the soft information (e.g. log-likelihood ratio (LLR) available at the physical layer which is providing an indication of the reliability of each received bit), is propagated to the application layer to help ranking the candidates from most likely to least likely, and finally choosing a most likely bitstream that is validated by the decoder as the final candidate bitstream (Nguyen *et al.*, 2010).

Most traditional correction methods relying on the availability of soft information require exhaustive changes to the whole protocol stack to propagate such soft information, i.e. a fixed or floating point LLR value for each bit of the packet, from physical layer to the application layer. This issue will make them very complex to deploy in practice. This explains why although very effective correction methods exist in the literature, they are rarely deployed commercially. The lack of soft information for traditional list decoding approaches means that all the bits are having the same probability to be flipped; therefore all the candidate bitstreams are having the same probability. All candidates (without any preference) should go through the video decoder for more constraints. Therefore, all these approaches suffer from the major drawback of having a fairly large solution space for candidate packets, leading to a decoding process with

extremely high computational complexity. Indeed, a packet containing N bits has 2^N possible candidates when any number of errors is considered (or N candidates when a single-bit error is considered).

1.2 Objectives

The ultimate goal of this research project is to enhance the quality of the reconstructed frames in the presence of the transmission errors, particularly when the packets have been lightly damaged during the transmission. In order to achieve our goal, the following objectives were defined:

- The first objective was to study if there is a way to utilize the received corrupted packets instead of ignoring them at the reception. This requires a packet filtering mechanism which, under some specific conditions, retains the packet or discards it.
- The second objective was to correct the received packets that are damaged at the bit level, especially when the soft information (e.g. LLR of bits) is not available. This was targeted by exploiting other information in the protocol stack such as user datagram protocol (UDP) checksum value.

The first objective was achieved by exploiting the syntax elements in H.264 context-adaptive variable-length coding (CAVLC) coded sequences, and defining the concept of non-desynchronizing bits (NDBs). An NDB is a bit that does not cause any desynchronization at the bit level and more importantly, that does not have any impact on the number of decoded macroblocks. Based on this definition and various observations (effect of individual errors on NDB and visual quality), we proposed two conditions for which, if satisfied, the corrupted video packets are reliable-enough to be used in the reconstruction of the corrupted frame instead of discarding them and performing concealment. The proposed packet filtering approach allows the decoder to save not only all the error-free MBs before the error occurrence, but also the ones after that. This is possible since the error on NDBs does not cause any desynchronization at the bit level or semantic errors. Therefore, the effect of such errors is very small or,

in the case of propagation, it will be limited to a small area in the pixel domain. However, the visual difference from the intact packet in this case is much smaller than the one introduced by the concealment approaches.

The second objective was addressed by exploiting the receiver side UDP checksum value. The possible locations of errors in the packet can be identified by analyzing the pattern of the calculated UDP checksum. This allows our proposed checksum-filtered list decoding (CFLD) approach to alleviate the large solution space problem of conventional list decoding approaches. For instance, when a packet composed of N bits contains a single-bit error, instead of considering N candidate bitstreams, as is the case in conventional list decoding approaches, the proposed approach considers approximately $N/32$ candidate bitstreams, leading to a reduction of 97% of the number of candidates. Therefore, it is more likely for the proposed approach to finally select the best candidate (correct the error) and improve the quality of the reconstructed corrupted frame.

Both proposed methods (individually and moreover jointly), by utilizing the corrupted packet, are expected to reduce the area that requires to be concealed. The approaches are also expected to be more effective when there are only a few transmission errors. By few errors, we mean that the video stream contains only a small number of errors and each packet contains at most one-bit error most of the time.

1.3 Thesis structure

In order to facilitate the reading of this thesis, it is organized into three additionned chapters followed by a conclusion. A brief overview of each chapter is provided as follows:

In chapter 2, we review how the problem of visual quality degradation caused by transmission errors is addressed in the literature. We provide a comprehensive overview of the two post-processing approaches (implemented at the decoder side), error concealment and error correction, in separate subsections. For instance, for concealment, the approaches are moreover categorized into spatial, temporal and hybrid methods based on the utilized correlation

information during the concealing. The correction approaches are also classified into list decoding and joint source channel decoding (JSCD) approaches.

In chapter 3, we address our first objective by studying under what conditions it is more efficient to retain the corrupted packet instead of ignoring it. The chapter starts by analyzing and presenting all the syntax elements (along with their descriptions and role in encoding) in H.264 CAVLC coded sequences to specifically identify their corresponding NDBs. We also present the frequency of the NDBs in the typical coded sequence and the impact of flipping each NDB individually on the visual quality. Additionally, our proposed framework based on keeping corrupted packets (under some conditions), as well as the simulation results on H.264 baseline profile, are presented in this chapter. A preliminary result of this chapter (limited to a few syntax elements) has been presented in the 12th International Conference on Signal Processing and Communication Systems (ICSPCS) 2018 (Golaghazadeh *et al.*, 2018a).

The problem of correcting the corrupted video packet is addressed in chapter 4. The chapter begins with a detailed introduction to the UDP checksum and its calculation. Then, we explain how the checksum can be applied to error correction. This is done by defining different bit error events and calculating their corresponding checksum values. Our proposed CFLD approach is described in this chapter and it is validated on H.264 CAVLC and HEVC sequences. We have published a conference paper (Golaghazadeh *et al.*, 2017), a journal paper (Golaghazadeh *et al.*, 2018b) on this subject. Moreover a provisional patent (Golaghazadeh & Coulombe, 2017) has been filed on the topic.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we provide a comprehensive overview of two common error control mechanisms which are applied to various video coding standards, such as H.264, H.265. The ultimate goal of the error control approaches is to alleviate the influence of the transmission errors on the quality of a reconstructed video sequence. In the following, first, we review video error concealment techniques in Section 2.1, and then, in Section 2.2, different error correction approaches are discussed. Figure 2.1 shows the general implementation of these two approaches in a real-time video communication system.

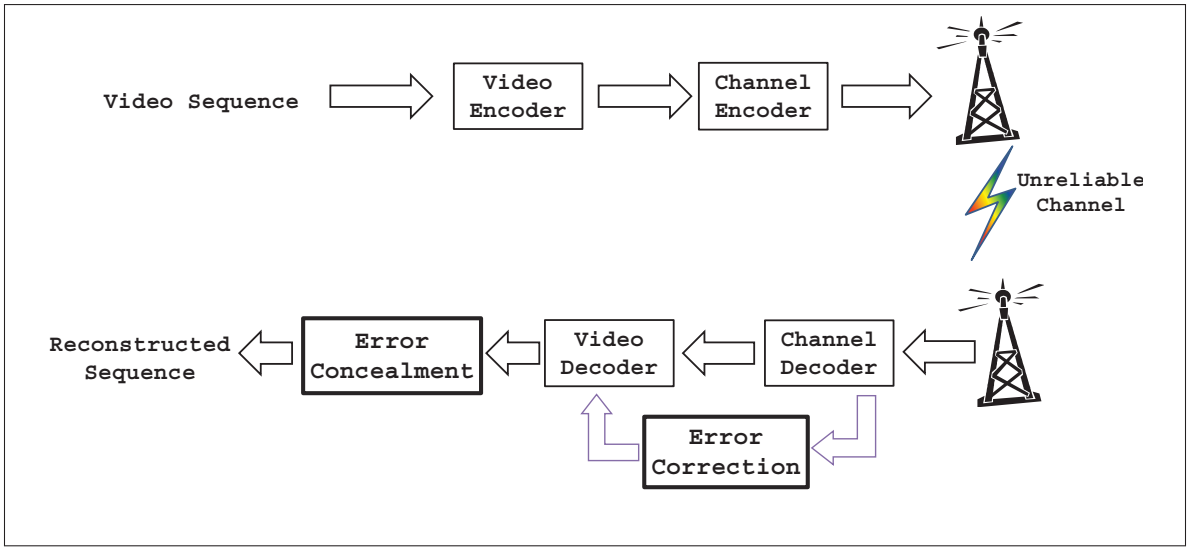


Figure 2.1 Illustration of the error concealment and error correction implementation in a real-time video transmission system.

2.1 Error concealment

The error concealment technique estimates lost areas by exploiting the inherent correlation between adjacent pixels. This can be done by making use of the spatial correlation between neighboring pixels in one frame or the temporal correlation in pixels or scenes in the successive

frames. In the following subsections, we will review the existing concealment techniques in three distinct groups as spatial, temporal and spatiotemporal error concealment.

2.1.1 Spatial error concealment

Generally, in smooth areas of a natural image, the pixel intensity values are very similar to each other. Spatial error concealment takes advantage of this smoothness property in neighboring pixels to reconstruct lost ones. The most basic and popular spatial error concealment technique performs pixel-wise interpolation, such as bilinear interpolation. Each pixel in the lost macroblock (MB) is interpolated using four nearest intact pixels in all four boundary MBs, (Salama *et al.*, 1995, 1998; Xiu *et al.*, 2006) as depicted in Figure 2.2. The inverse distance between the lost pixel and the received corrected neighboring ones is used as interpolation weight in such a way that the nearest intact pixel has more impact on the interpolation. As defined by Xiu *et al.*, the lost pixel $\widehat{P}$ can be estimated from equation 2.1,

$$\widehat{P} = \frac{\frac{P_R}{d_R} + \frac{P_L}{d_L} + \frac{P_T}{d_T} + \frac{P_B}{d_B}}{\frac{1}{d_R} + \frac{1}{d_L} + \frac{1}{d_T} + \frac{1}{d_B}} \tag{2.1}$$

where P_i, with index i being T, B, L and R indices, denotes the value of the closest intact pixel to lost pixel, $\widehat{P}$, in the top, bottom, left and right neighboring MBs, respectively. Similarly, d_i denotes the distance between the lost pixel $\widehat{P}$ and P_i for the four neighbors. This approach can work fairly well particularly in the monotone (or low frequency) areas of an image but not in the high-frequency regions when the intensities change rapidly or edge zones.

Sun & Kwok (1995) proposed to restore the edges in lost areas by considering the existing ones in large surrounding blocks. The proposed algorithm starts by determining the type of the lost blocks being a monotone block or an edge block. For the latter case, the edge orientation angle is calculated by using the Sobel convolution mask. Then, all the discovered edges are classified into one of the eight directional categories equally spaced in the range of 0–180 degrees (see Figure 2.3). In order to detect any edge, two parameters of magnitude (G) and angle (θ) are

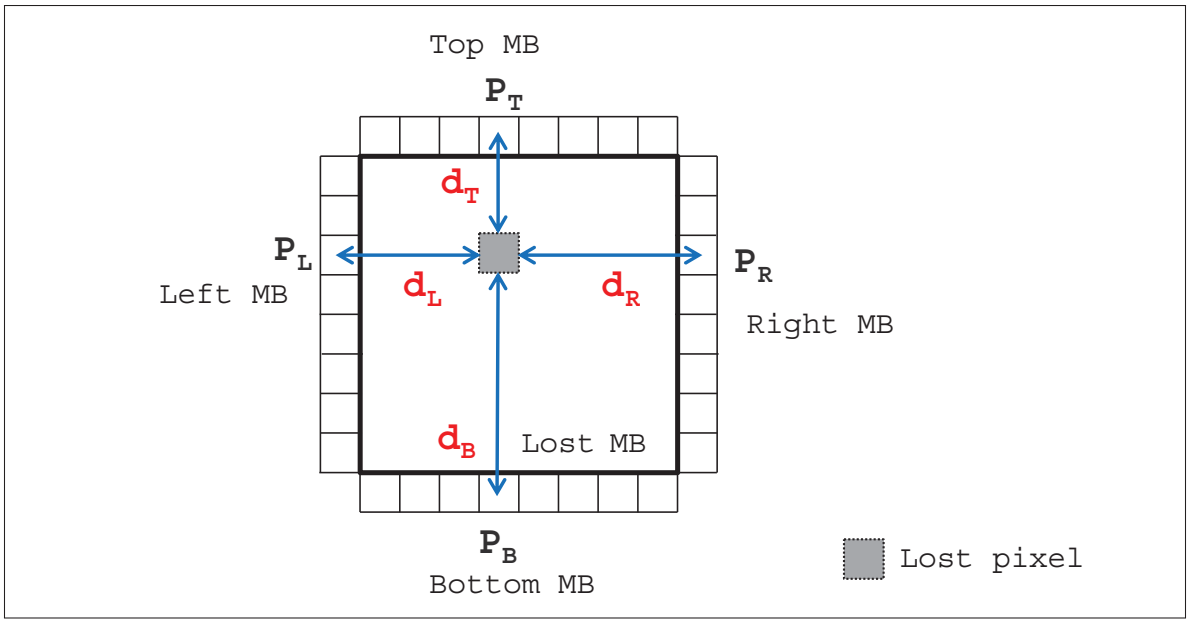


Figure 2.2 Bilinear interpolation by four nearest pixels located in the neighboring undamaged blocks (Xiu *et al.*, 2006).

computed by equation 2.2 for all the adjacent pixels to the lost area:

$$G = \sqrt{G_x^2 + G_y^2} \quad ; \quad \theta = \arctan(G_y / G_x) \tag{2.2}$$

where the approximations of the derivatives G_x and G_y are the Sobel operation mask on the neighboring areas of the lost block, denoted as $\mathbf{P}$, (Note that $\circledast$ is the convolution operation symbol):

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \circledast \mathbf{P} \quad ; \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \circledast \mathbf{P} \tag{2.3}$$

After classifying all the detected edges in the eight directions, only the predominant edge is chosen for the interpolation direction (if the line with specific angle passes through the missing

region). Finally, an iterative and complex method of projection is applied to reconstruct the missing monotone block or the edge block with a particular orientation.

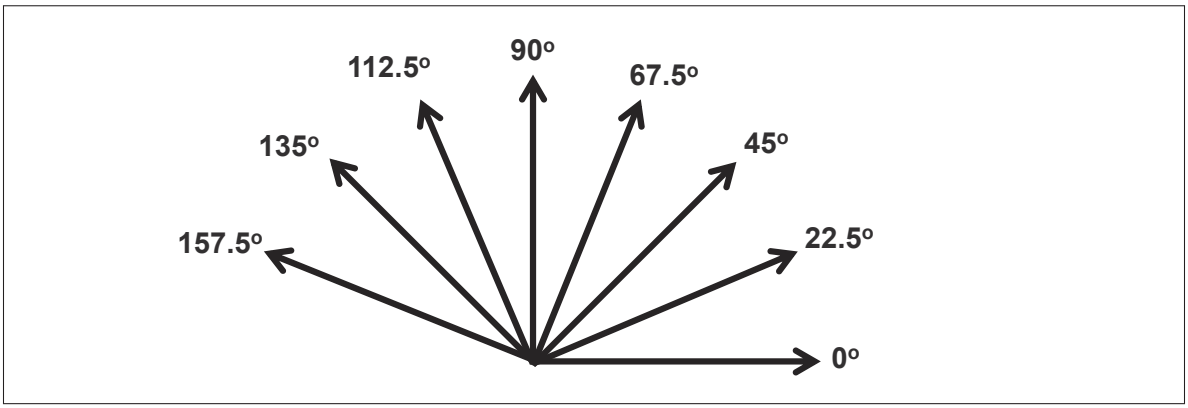


Figure 2.3 Eight directional edge categories (Sun & Kwok, 1995).

The proposed method was optimized and extended in Kwok & Sun (1993) to perform multi-directional edge interpolation instead of one strongest edge interpolation. Then, an image mixer step is employed to combine all the features obtained from the different directional interpolations into a single block for the lost area. Besides recovering lost pixels from the limited edge directional interpolations in the previously mentioned approaches, the major risks are ignoring a true edge or generating a false edge. Since the human visual system is very sensitive to the edge integrity, it is important in the spatial error concealment methods to deal with both issues. A considerable amount of research has been performed on spatial error concealment by focusing on the edge direction detection algorithms, irrespective of the number of surrounding edges, such as employing Hough transform-based technique to detect the relevant edges for directional interpolations (Robie & Mersereau, 2000; Gharavi & Gao, 2008; Koloda *et al.*, 2013b). In the work proposed by Robie & Mersereau, first, a 3×3 MB area around the lost MB is segmented into three different intensity groups using a clustering algorithm. Then, the Hough transform is applied to detect the edges in each group separately. A strong edge that crosses the lost area is used for interpolation. This approach has been extended by Gharavi & Gao to systematically connect the edges in lost areas and accordingly, divide the lost

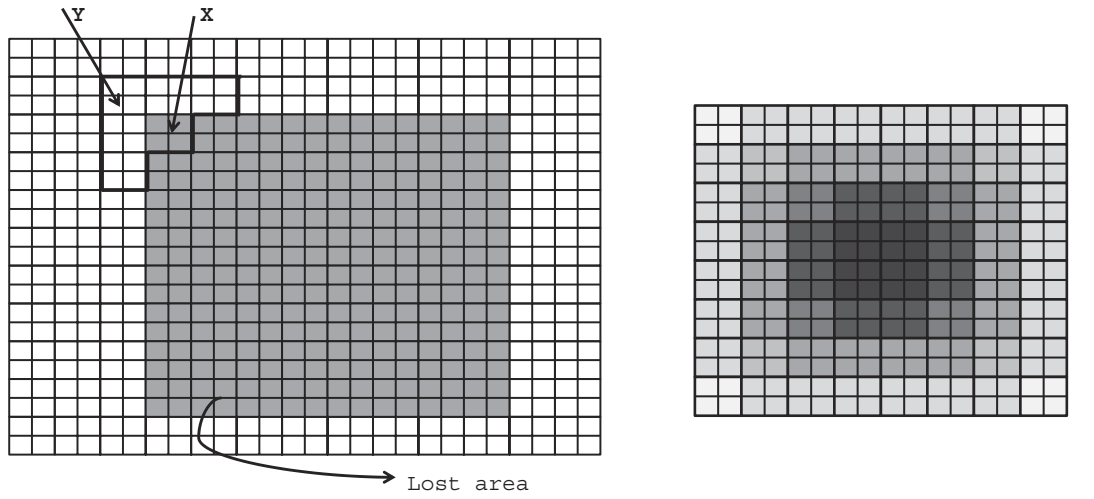
area into different regions for interpolations. Koloda *et al.* proposed to perform a weighted pixel by pixel directional interpolation on the visually clearest edge identified by the Hough transform. The weights are specified individually for each lost pixel based on its introduced visual clearness parameter. Kim *et al.* (2006) proposed a fine directional interpolation algorithm which attempts to extract the spatial direction vector sets from the edge structure in all the neighbors. Then, a pixel-wise interpolation is used to estimate the lost pixels in the dominant vector orientations (Kim *et al.*, 2006). Furthermore, a multi-directional interpolation algorithm was suggested (Asheri *et al.*, 2012) that ignores weak edges (by a fixed threshold value) in interpolations and only selects the strong edges that pass the adaptive defined thresholds.

Hsia & Hsiao (2016) proposed an algorithm that first classifies the lost blocks into four distinct groups of blocks based on the neighboring blocks' features. It then applies different methods of concealment on each group. As an advantage, the edge recovery process is only performed on one group of blocks not all the lost blocks. Although it seems they have reduced the computational complexity compared to the other traditional edge recovery approaches, their proposed approach brings other overheads from classifying the blocks.

There are some other researches that aim to reconstruct the missing pixels by applying kernel density estimation (KDE). The KDE approach estimates missing samples (a 2×2 patch shown as X in Figure 2.4) by considering all the available neighboring pixels whiting a 6×6 area centered by X (as identified by Y in Figure 2.4) and utilizing the kernel-based minimum mean square error. The lost area is reconstructed sequentially from outside toward the center (Koloda *et al.*, 2014, 2017). The main drawback of this approach is the computational complexity which the authors have made an effort to reduce in (Koloda *et al.*, 2017).

2.1.2 Temporal error concealment

Temporal error concealment approaches aim to restore the missing areas by exploiting the temporal redundancy between the adjacent frames. Due to the motion compensation coding, a lost motion vector can lead to severe visual distortion and it can simply propagate to the



a) Sample vectors for concealment

b) Reconstruction order from brighter to darker block

Figure 2.4 Proposed KDE for spatial error concealment (Koloda *et al.*, 2014, 2017).

following frames. The goal of most temporal concealment approaches is to recover the lost motion vectors. Therefore, a missing block can be substituted by the one that is pointed by the motion vector in the reference frame.

Generally, the temporal concealment approaches involve two steps: first determining the candidate blocks (or in other words, the candidate motion vectors), and then, selecting the best block among the candidates for the lost area. These two steps can be sequentially performed on all the blocks in the missing area.

A basic motion vector candidate list was proposed as follows in (Lam *et al.*, 1993).

- the zero motion vector
- the motion vector of the same block in the previous frame
- a motion vector from available neighboring blocks
- the median of all the available neighboring motion vectors

- the average of all the available neighboring motion vectors

As the second step, Lam *et al.* (1993) proposed a technique known as boundary matching algorithm (BMA) which is a metric to find the best blocks among the candidates. The approach compares the inner pixels of the candidate block with the outer pixels of the available neighbors in the lost block. An illustration of the BMA approach is shown in Figure 2.5. This comes from the fact that there must be a strong correlation between adjacent pixels in a frame. Therefore, the best motion vector is the one that minimizes the *BMA* value in equation 2.4. The *BMA* value is defined as the squared difference between the inner pixels of candidate blocks and the external boundary pixels of the lost block.

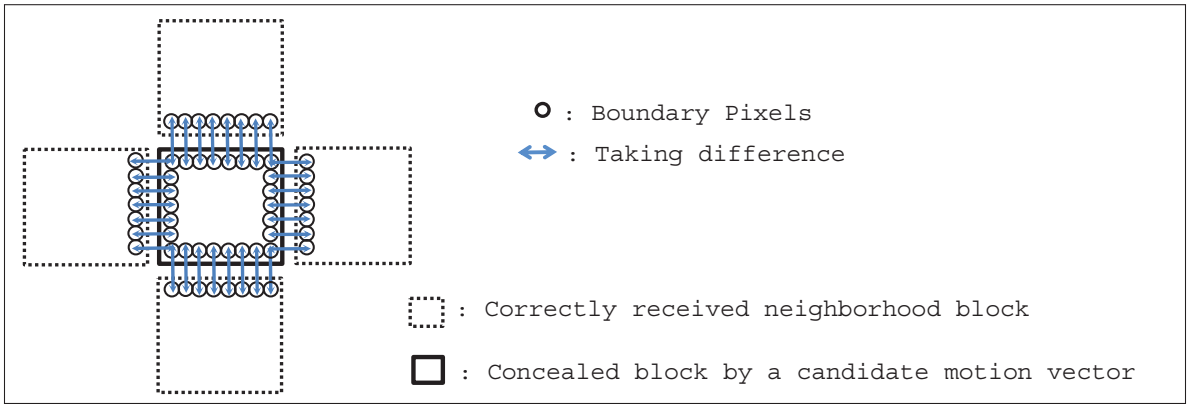


Figure 2.5 BMA technique illustration.

$$\begin{aligned}\mathrm{BMA}(x_0,y_0,mv_x,mv_y) = &\sum_{x=x_0}^{x_0+N-1}(f(x+mv_x,y_0+mv_y,t-1)-f(x,y_0-1,t))^2 \\ &+\sum_{x=x_0}^{x_0+N-1}(f(x+mv_x,y_0+mv_y+N-1,t-1)-f(x,y_0+N,t))^2 \\ &+\sum_{y=y_0}^{y_0+N-1}(f(x_0+mv_x,y+mv_y,t-1)-f(x_0-1,y,t))^2 \\ &+\sum_{y=y_0}^{y_0+N-1}(f(x_0+mv_x+N-1,y+mv_y,t-1)-f(x_0+N,y,t))^2\end{aligned} \tag{2.4}$$

where $f(x,y,t)$ and $f(x,y,t-1)$ stand for the pixel values at coordinate (x,y) for the current and previous frame, respectively. The pair value of (x_0,y_0) is the upper left coordinate of the lost $N\times N$ block and its candidate motion vector is defined as (mv_x,mv_y).

The BMA was popularly used in the literature with some modifications to consider additional pixels at the borders or refinement techniques for the replaced motion vector's block. Chen *et al.* (1997) proposed to utilize boundary matching distortion as a criterion along with their defined overlapped motion compensation weighted technique for the concealment. After finding the candidate block by distortion criteria, instead of directly placing it in the lost area, the overlapped block motion compensation is used. To do that, the lost block is divided into four sub-blocks and each sub-block is replaced by the weighted average of three predicted sub-blocks: one according to the candidate block (from distortion criteria), the second one according to the motion vector of the horizontally neighboring sub-blocks, and the third one according to the motion vector of the vertically neighboring sub-blocks. It is worth mentioning that the approach considers only the available neighboring motion vectors as the candidate motion sets, which can perform well only when the all neighbors are correctly received. The approach performs poorly when there are less neighbors available since they only considered the motion vectors of the available neighboring blocks as the candidates.

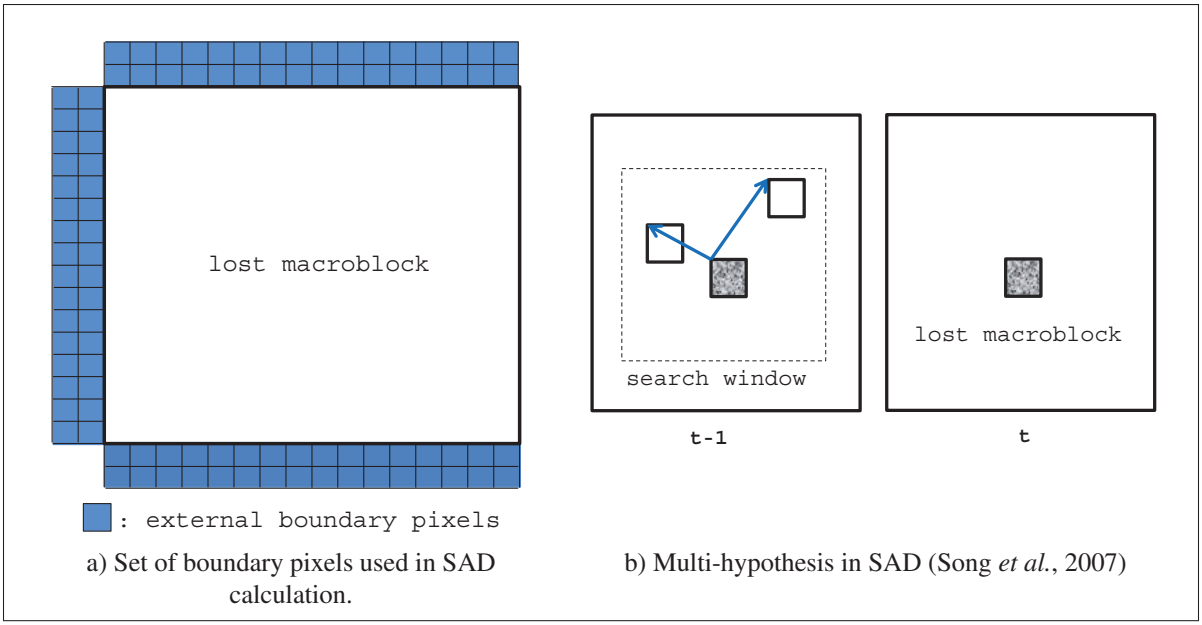
Unlike the BMA, which uses spatial correlation as a criterion, another popular approach, known as decoder motion vector estimation (DMVE), was introduced in (Zhang *et al.*, 2000) and uses temporal redundancy to recover the motion vectors. The DMVE algorithm employs a similar process as the encoder motion estimation at the decoder side. Generally, the lost block will be searched in a window in the previous frame. The best candidate block, in other words, the best motion vector, is the one that has the smallest sum of absolute differences (SAD) value. Therefore, $\mathrm{SAD}_{\mathrm{DMVE}}$ can be calculated as it is defined in equation 2.5.

$$\mathrm{SAD}_{\mathrm{DMVE}}(mv_x, mv_y) = \sum_{(x,y)\in\{T,B,L\}} |(f(x\ ,y\ ,t) - f(x+mv_x\ ,y+mv_y\ ,t-1)| \tag{2.5}$$

Where $f(x,y,t)$ is the pixel value of (x,y) coordinate in the current frame. Similarly, $f(.,.,t-1)$ stands for the pixel values in the previous frame. The SAD is calculated on a set of boundary pixels, two rows and columns pixels of the top (T), bottom (B) and left (L) border, as shown in the left side of Figure 2.6. Finally, the block with the minimum $\mathrm{SAD}_{\mathrm{DMVE}}$ value is replaced in the lost area.

More advanced approaches in this category suggested to search for the motion vectors in more frames or consider more blocks for replacement by SAD calculation. For instance, the multi-hypothesis concealment proposed in (Song *et al.*, 2007) selects multiple blocks with the lowest $\mathrm{SAD}_{\mathrm{DMVE}}$ value for replacement, unlike the conventional approach which only retains the block with minimum $\mathrm{SAD}_{\mathrm{DMVE}}$ value. Then the lost block is reconstructed based on all the selected blocks with a defined weight factor for each of them. Although these approaches perform well compared to the single-hypothesis, they all suffer from the computation complexity.

Wu *et al.* (2008) proposed a combination method for the motion recovery based on the number and the position of correctly received neighbors around the lost MB. The spatial correlation of the neighboring motion vectors is used to calculate the motion tendency between neighboring blocks. For instance, when more neighbors are available, the horizontal or vertical SAD calculation is used as a metric (temporal correlation) while in the case of less available neighbors, the BMA (spatial correlation) is employed.



a) Set of boundary pixels used in SAD calculation.

b) Multi-hypothesis in SAD (Song *et al.*, 2007)

Figure 2.6 Illustration of the SAD approach in motion vector recovery (Zhang *et al.*, 2000; Song *et al.*, 2007).

An adaptive error concealment order determination is proposed in (Qian *et al.*, 2009) to recover the motion vectors of the corrupted MBs when the lost MBs are connected. The approach is proposed especially when the flexible macroblock orderings (FMO) feature of the encoder is not used and the erroneous MBs are attached to each other. This can be observed in the raster scan mode, wipe scan mode, interleaved FMO, etc. They proposed a confidence factor for each neighboring blocks to deal with the recovery dependency problem in connected regions. This factor presents the reliability of a neighboring block based on whether it is a recovered block or a correctly received one. With the use of mean absolute differences (MAD), the difference between the M-width external boundaries of the lost block in the current frame and the reference frame, the best motion vector is estimated as the one that has minimum MAD value. We have:

$$\mathrm{MAD}(mv_x, mv_y) = \sum_{i \in \{T,B,L,R\}} w_i \, \mathrm{MAD}_i(mv_x, mv_y) \tag{2.6}$$

where w_i stands for the reliability factor and the letters T, B, L, and R are the short form of the top, bottom, left, and right blocks, respectively. Also, MAD_i for M-width external boundaries

of an $N \times N$ lost MB is defined as follows:

$$\mathrm{MAD}_i(mv_x, mv_y) = \sum_{(x,y)\in\{M \text{ width boundary of } i\}} |(f(x\,,y\,,t) - f(x+mv_x\,,y+mv_y\,,t-1)|/(M\times N) \tag{2.7}$$

It has been observed that the performance of BMA can be affected based on the lost area features such as including rotation, zoom, fast/slow scene changes, etc. Therefore, having a fixed set of candidate motion vectors for all the block types are not suitable. Zabihi *et al.* suggested to adaptively employ a different set of motion vectors in BMA based on the information from the neighboring MBs in the current and previous frames. Different factors have been defined to evaluate the movement type of a lost block, and determine whether the block belongs to an object or not. Regarding these factors, the motion vectors of the co-located MB, median of neighboring motion vectors or a set of neighboring motion vectors are used as the candidates (Zabihi *et al.*, 2017) .

Even though a series of modified BMA, DMVE have been proposed, they still have the drawback of not being able to estimate accurately complex motions. For example, when the motion vector of a lost block differs from its neighbors or fewer neighbors are available. Although the latter can be solved using the FMO resilience feature provided in the recent video standard such as H.264, generally, these approaches perform very well in homogeneous areas or when the motions are linear.

Motion vector extrapolation approaches are proposed and developed in (Peng *et al.*, 2002; Zhou *et al.*, 2011; Lin *et al.*, 2013a,b). In these approaches, first, the candidate motion vectors (or extrapolated blocks) are extrapolated from the last decoded frames to the corrupted one, as depicted in Figure 2.7. Then, for instance, Peng *et al.* proposed to select the best motion vector based on the overlapped area between extrapolated block and the lost area. The motion vector of the most overlapping block is chosen as the motion vector for the lost block. Several metrics or weights are proposed in the literature for choosing the best motion vectors. Zhou *et al.* proposed to interpolate the best motion vector using the motion vectors of extrapolated blocks for both the lost area and its neighbors and also the available neighboring motion vectors. This

approach has been extended in Lin *et al.* (2013a) to employ the residual information of the neighboring blocks as a weight, to assess the reliability of the neighboring motion vectors, in the estimation of the best motion vector. In addition to that, the distance between the lost block and the available neighbors is further considered as another weight. To alleviate the blocky artifacts caused by their approach, Lin *et al.* (2013b) proposed to consider the partition decisions of the MBs in the previous frames as an extra weight along with the overlapping weight, for the selection of best motion vector. With this additional weight, they take into account the object segmentation information in the estimation process (Lin *et al.*, 2013b). In a recent study, (Lin *et al.*, 2018), the proposed approach of (Zhou *et al.*, 2011) has been modified in three areas to provide a better technique for motion estimation. Instead of the conventional raster scan block recovery, the motion vector estimation starts from the corner block with the highest number of available neighbors and the process continues to estimate the motion vector of center blocks. Two prediction weights were defined and considered in the calculation of the best motion vector: first a horizontal and a vertical disparity weight that presents the consistency (reliability) of the adjacent motion vectors in that direction, and second, a weight defined as the difference between the motion compensated block and its decoded one for each neighbor. The authors believe that the motion vector of a block with a higher difference must have less weight in the prediction process. A combination of these two weights was employed in their proposed block recovery order approach to estimate the motion vector of each lost block (Lin *et al.*, 2018).

In another category of motion vector recovery, the lost motion vector is modeled and interpolated using the neighboring motion vectors. The idea behind these approaches is that the adjacent blocks may have the same motion tendency (correlation between motion vector values). For instance, a second-order polynomial model was proposed by Zheng & Chau (2005) to describe such a correlation between four neighboring motion vectors. The approach operates on the assumption that in interleaving techniques (such as FMO), all the neighboring motion vectors are available for interpolation. Lee *et al.* (2001) proposed utilizing an affine transform to consider complex motion behavior, like rotation, expansion, and contraction, ad-

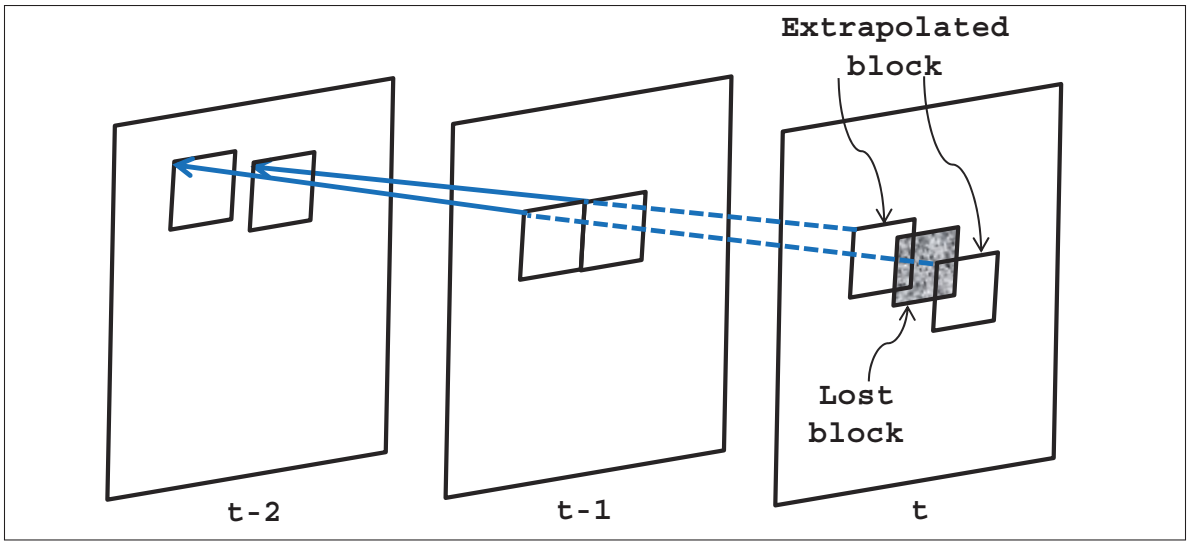


Figure 2.7 Motion vector extrapolation illustration (Peng *et al.*, 2002).

ditional to the shifting as parameters in the transform equation. The approach divides the lost MBs into triangles and considers six motion parameters. The parameters are estimated based on BMA on vertexes of the triangular patches. Lie *et al.* (2014) proposed to look at the BMA algorithm as an optimization problem for the case when a row of consecutive MBs (a row slice) is lost. Instead of recovering the motion vector for each MB independently, the proposed approach estimates globally all the motion vectors of the lost slice by using dynamic programming techniques. Although the motion vector modeling approaches are able to globally solve the problem, relying on only motion vector values and not considering the border continuity may cause blockiness artifacts.

In a recent study (Choe *et al.*, 2018), object recognition techniques and scene change features are used for temporal error concealment. Some characteristic of coded sequences, such as the number of bit in a frame (as a dynamic threshold), the discrete cosine transform (DCT) coefficients' block energy, and inter and intra prediction modes are considered in order to detect scene changes. The proposed temporal concealment algorithm is based on iteration on unknown convex set with object recognition principal component analysis (PCA) training model. Their compound approach was only simulated on low resolution, QCIF, sequences.

2.1.3 Spatiotemporal error concealment

Spatiotemporal error concealment approaches aim to reconstruct the lost areas as a combined approach by employing both spatial and temporal redundancies. For instance, an intra (I) frame's block may be better concealed by considering only spatial redundancy. While in the case of inter (P) blocks or more complex scenes, it is suitable to note all the valuable information (whether spatial or temporal) in the concealment process. This category further includes all the approaches that are using both redundancies in the estimation of the motion vectors.

Atzori *et al.* (2001) proposed a two-step concealment approach by combining both available temporal and spatial information. First, the lost block is temporally replaced by BMA. Then, a mesh-based transformation, based on an affine transformation in the spatial domain, is applied to best fit the replaced block with its correctly received surrounding area. With a little computational complexity overhead, the approach has better performance compared to the plain BMA approaches. In other approaches, different techniques are proposed for concealment of an I or a P frame. For instance, Kung *et al.* (2006) proposed to conceal an I-frame's lost MBs by the directional interpolation technique and a P-frame's one by temporal linear interpolation from three previous frames. Since a P frame can be coded with both I and P MBs, the proposed approach assumes that a lost MB surrounded by more than two I MBs is also an I one, which is not always true. Besides from that, the proposed approach requires to buffer previous frames, at least three of them, to track the error variance map of replacing different candidates. A hybrid concealment approach has been proposed in Xiu *et al.* (2006), which adaptively decides spatial or temporal concealment based on a boundary matching criteria. The proposed approach calculates the boundary pixel differences for two spatial techniques (bi-linear interpolation and average interpolation) and one temporal technique (motion estimation by classifying the motion relatively), and then selects the technique with a minimum boundary difference. Finally, an iterative maximum a posteriori (MAP) estimator is used to further smooth the reconstructed area. The performance of their approach depends on the number of iterations which adds extra complexity to the system.

Gaussian mixture model has also been employed in error concealment to estimate the lost pixel from the neighboring context (Persson *et al.*, 2008; Persson & Eriksson, 2009). The method first solves the Gaussian parameters offline and then uses the obtained model to estimate the lost blocks by considering both spatial and temporal surrounding pixels. A computationally lighter version of the approach is proposed in Persson & Eriksson (2009) where the parameters are estimated in an iterative algorithm in the least squares sense, while in (Persson *et al.*, 2008), the means of the expectation maximization algorithm has been used to compute the parameters. Moreover, an edge-directed spatiotemporal concealment approach was proposed in Ma *et al.* (2010) where strong edges around the lost region are first estimated based on the edges in both previous and current frames. Then, the lost pixels along the estimated edges are recovered by using both spatial and temporal surrounding pixel values. Finally, the remaining lost area is calculated with a patch-based filling approach. They assumed that the lost MB was surrounded with correctly received MBs. For this matter, the available error resilience technique in H.264, FMO, has been employed to fulfill the assumption.

To our knowledge, one of the most outstanding methods in the existing literature has been proposed by Chen *et al.* (2008). The approach describes reasonable boundary matching criteria to recover a motion vector or the lost area that can preserve both spatial and temporal continuities. Their proposed spatiotemporal boundary matching algorithm (STBMA) defines a cost function based on considering both spatial and temporal distortion, in a weighted manner, as follows:

$$\text{STBMA}(mv_x, mv_y) = \alpha \times D^{\text{temporal}} + (1-\alpha) \times D^{\text{spatial}} \tag{2.8}$$

Where $\text{STBMA}(mv_x, mv_y)$ presents the total distortion of replacing a candidate MB pointed to by (mv_x, mv_y). D^{spatial} and D^{temporal} are the two spatial and temporal distortion functions, respectively. The weighting factor α is a real number between 0 and 1 to control the blockiness artifacts of direct replacement. The temporal distortion function measures the difference between external boundaries of a lost block with the external boundary of a candidate block in the previous frame, as depicted in Figure 2.9. This value determines how well the temporal continuities are preserved. The spatial distortion function minimizes the gradient field of the

reconstructed block in a costly iterative approach. The best motion vector (candidate block) is chosen by minimizing the STBMA cost function. Moreover, they proposed to refine the discontinuity arising from replacing the candidate block by a partial differential equation (PDE) algorithm. At least ten iterations are required to achieve an acceptable performance which brings significant complexity to the system.

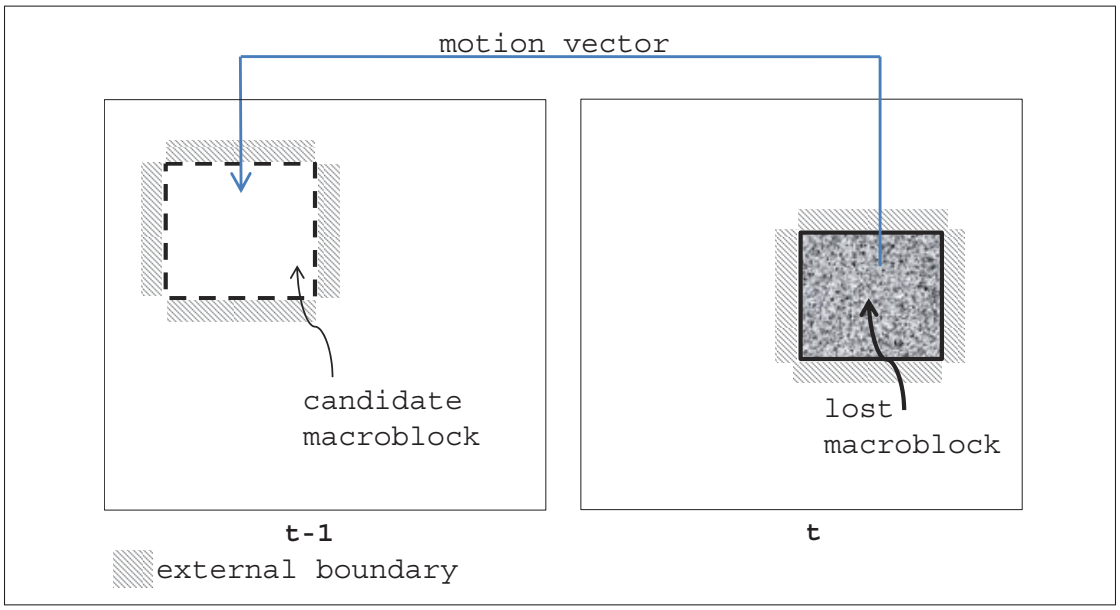


Figure 2.8 Temporal distortion used in (Chen *et al.*, 2008)

Similar to the previously mentioned approach, Xiang *et al.* (2011) proposed a more general distortion function for concealment. Their defined distortion function contains three weighted distortion terms. The first one, spatial distortion, calculates the distortion between the outer boundary of lost pixels in the current frame and the internal boundary pixels of candidate block in the previous frame in all four neighbors. This term is the same as the conventional BMA distortion measurement, as shown in equation 2.4 and Figure 2.5. Their employed temporal distortion, the second term, is similar to the one proposed in Chen *et al.* (2008). As the third term, they have considered the side match distortion between the lost block and its left neighbor as depicted in Figure 2.9. The smaller difference between internal and the external boundaries of the reference block in the previous frame presents the more smoothness at the borders.

The best motion vector is the one that minimizes the total distortion function (including three terms). Finally, the block pointed by the motion vector is replaced in the corresponding lost area. However, they assumed that the residual information is negligible which will affect the performance of their approach in low quantization parameter (QP) coded sequences.

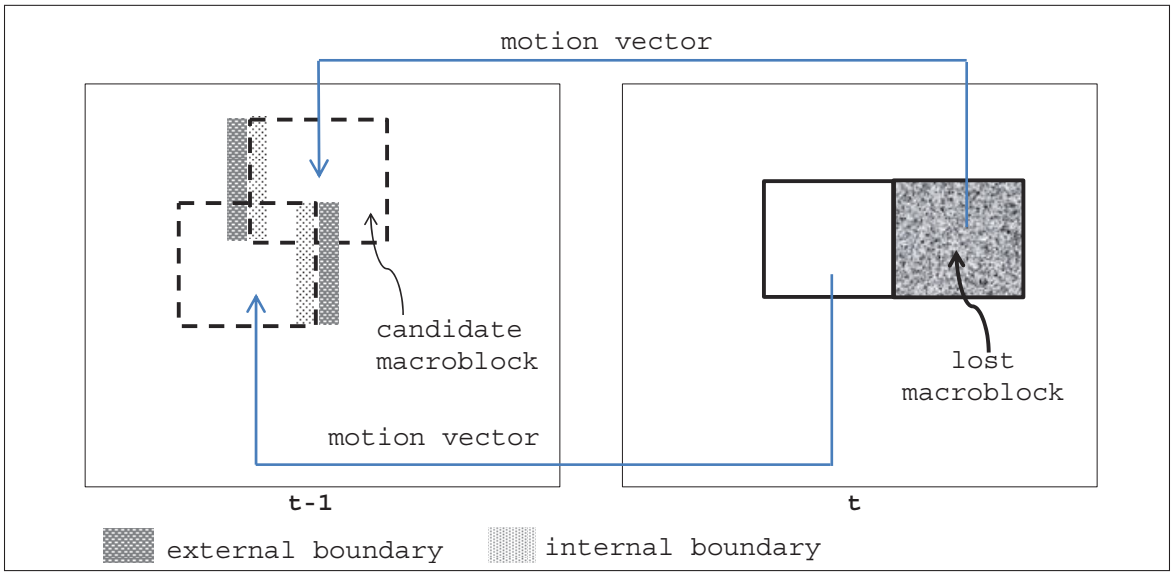


Figure 2.9 Side distortion used in (Xiang *et al.*, 2011)

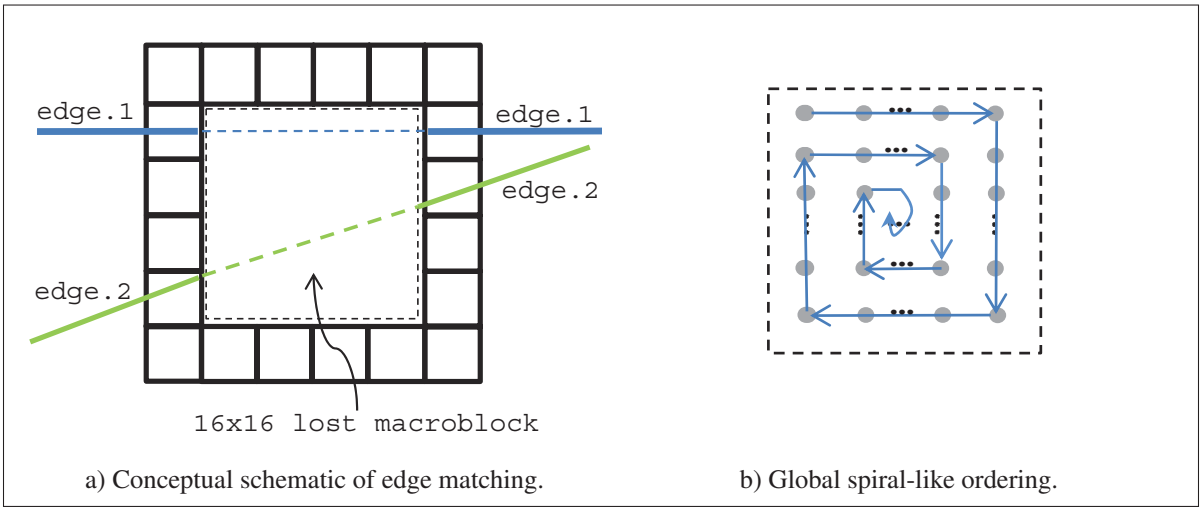
Zhang *et al.* (2012) proposed an auto regression (AR) model in concealment by considering both spatial and temporal information to refine the estimated pixels for the lost area. Their proposed approach is compatible with any motion vector recovery algorithm such as BMA or STBMA. After recovering the motion vector, each corrupted pixel, denoted as $\widehat{P}(x,y)$ in equation 2.9, is refined by a weighted summation of the corresponding pixels in a square area of $(2R+1)\times((2R+1))$ pixels, pointed by the motion vector in the previous frame. The linear sparse regression model for estimation of the lost pixels is expressed as:

$$\widehat{P}(x,y)=\sum_{k=-R}^{R}\sum_{l=-R}^{R}\alpha(k,l)\,f(x+mv_x+k\,,y+mv_y+l\,,t-1) \tag{2.9}$$

where $f(x,y,t-1)$ is the pixel value of (x,y) coordinate in the previous frame and the value of R defines the AR range of the surrounding pixels. The weighted AR coefficients $\alpha(k,l)$ are calculated based on spatial and temporal continuity constraints separately. Finally, the interpolation results of both coefficients are merged to restore lost pixels.

A similar work to the previous one is presented in (Lin *et al.*, 2017) where the sparse optimization theory and sparse characteristic of the image's nature are used to predict the weighted coefficient in equation 2.9. Both spatial and temporal estimated coefficients are combined to perform better estimation of lost pixels. Unlike Zhang *et al.*'s approach, the average motion vector of neighbors are considered as the initial motion vector. The performance of these approaches is highly dependent on the initial motion vector's estimation. Additionally the processing time for gathering suitable pixels and solving the optimization problem increases the complexity of these approaches. Koloda *et al.* (2013a) proposed to use convex optimization in the calculation of the coefficients' weights in equation 2.9. In order to reduce the processing time of optimization, they have modeled the coefficients with an exponential function for fast estimations. Their sequentially proposed algorithm can automatically decide between spatially, temporally or mixed approaches and it starts from the area with most available pixels, as shown in Figure 2.4. It must be noted that the considered pixels in the summations of equation 2.9, is different for each mode of concealment. For instance, both known and unknown (lost) pixels in spatially adjacent neighbors, as shown in Figure 2.4, are considered in spatial error concealment. However, in temporal mode, the pixels from neighboring blocks and the co-located one in the previous frame are considered. Correspondingly, in combined mode, both the spatial and temporal pixels are employed in the optimization problem. In the work of Zhou *et al.* (2017), the high dimensional video data is considered as a 3rd-order of a tensor model. The proposed approach consists of two parts: the first part creates a tensor model based on the corrupted block and its candidate blocks in the reference frames. The candidate blocks are chosen with the help of a flexible size version of BMA according to the size of the lost area. The second part estimates the lost pixels by tensor low rank approximation.

In a more recent study (Shih *et al.*, 2018), a spiral-like pixel reconstruction (SPR) has been proposed which compared to the conventional zigzag or scanline mode, is able to reference more relevant neighboring pixels in the estimation of the lost pixel. In the proposed algorithm, first, all the edges of 4×4 blocks surrounding the lost area are identified and matched based on their similarity directions. Then, all the lost pixels along the highest magnitude edges are reconstructed. And finally, the pixels in non-edge area part are estimated using the SPR method in the block. The SPR method starts recovering the pixels from the exterior part to the center in a clockwise direction as shown in Figure 2.10. Although the approach works well on a single or multiple edge areas, the grouping edges procedure, determining the similarity of edges, and ignoring the non-similar edges are still an issue.



a) Conceptual schematic of edge matching. b) Global spiral-like ordering.

Figure 2.10 Illustration of proposed SPR concealment in (Shih *et al.*, 2018).

Clearly, the performance of error concealment approaches reduces when lost areas have less correlation, whether spatially or temporally, with the correctly received surrounding areas, or in fast or complex motion scene, or when the lost areas are quite large. Most error concealment approaches treat a corrupted packet in the same manner as a lost one, where the corrupted packets are ignored and the missing areas are concealed. In practice, network congestion results in packet loss, while wireless signal attenuation, fading, etc., result in corrupted packets. However, corrupted and lost packets must be handled differently. Partially damaged packets may

contain valuable information that can be used to enhance the visual quality of the reconstructed video (Superiori *et al.*, 2006; Trudeau *et al.*, 2011). This is the case when the error occurs at the end of the packet or when the residual bit error rate (after channel decoding) is low. The novel user datagram protocol (UDP) such as UDP-Lite has been developed to deliver partially damaged packets to the application layer (Larzon *et al.*, 2004).

2.2 Error correction

Unlike the error concealment, the goal of video error correction approaches is to utilize the corrupted packet and recover the originally sent one rather than assuming that the whole packet is lost. In other words, these approaches attempt to correct errors by modifying the received bits into a most likely transmitted sequence of bits. A correction process may sometimes be expressed as an optimization problem. Knowing that the received packet is corrupted, the aim is to find the best one among the set of all candidates:

$$S^* = \underset{S_t \in H}{\arg\max} \{P(S_t|S_r)\} \approx \underset{S_t \in H}{\arg\max} \{P(S_r|S_t) \times P(S_t)\} \tag{2.10}$$

where S^* is the likeliest packet to the received corrupted packet S_r, and it is chosen among all the hypothetically transmitted slices S_t in the solution space, shown as H. The works on this topic can be categorized as list decoding and joint source channel decoding (JSCD) which will be deeply described in the following sub-sections.

2.2.1 List decoding

For a received corrupted packet, list decoding approaches generate multiple candidate packets. Generally, the candidates are produced by flipping individual bits in the corrupted packet. Then the candidates are ranked from the most likely to the least likely bitstream, based on the soft information or reliability parameters of each bit provided by the channel decoder. Each candidate is then checked for semantic and syntactic errors by the specific video decoder. Finally, the winning candidate is the first one that passes the decoder semantic verification.

A log-likelihood ratio (LLR), which expresses the probability that a bit 1 was sent over the probability that a bit 0 was sent, given the same noise, is used by the channel decoders to provide the soft information, as shown in equation 2.11 (Hagenauer *et al.*, 1996):

$$LLR(b_n) = \log\left(\frac{P(b_n = 1|y)}{P(b_n = 0|y)}\right) \tag{2.11}$$

where $LLR(b_n)$ is defined as the soft information of a transmitted bit b_n when y represents the received noisy bit.

Ma & Lynch (2004) used turbo code to provide the soft information of each transmitted bit. They proposed to choose a limited number of bits with the smallest LLR value as the candidate flipping bits. Later, their candidate generator flips some or all of the candidate bits to generate candidate packets. Finally, the first candidate packet, which correctly passes the syntax checker of the MPEG-4 decoder, is selected as the likeliest packet for the corrupted one.

In (Levine *et al.*, 2007), 300 likeliest candidates are generated based on the soft value of transmitted bits. The candidate packets are ranked based on the smallest sum of soft values of their flipped bits. Similar to other list decoding approaches, the video decoder, here H.264 context-adaptive binary arithmetic coding (CABAC) decoder, validates the candidates and chooses the likeliest one. Their proposed approach was further modified by Nguyen *et al.* (2010), to accelerate the semantic verification process of the video decoder. This has been performed by adding a virtual checking step into the proposed system. The virtual checker eliminates some non-valid candidate packets based on the information of the previously failed candidates. For instance, if flipping a bit at position k causes a semantic or syntactic error in the following syntaxes, the virtual checker will drop all the candidate packets that contain a flipped bit at position k. In this way, the proposed approach can speed up the process of eliminating non-valid candidates by not verifying all of them by video semantic checker. As it has been seen in their simulation results, using the virtual checker saves on average 24% of the processing time compared to when it has not been employed.

2.2.2 Joint source channel decoding

In the JSCD approaches, often, the problem of finding the likeliest packet is viewed as a problem of finding the shortest path, as modeled in Figure 2.11 in such a way that the codewords or the syntaxes are the nodes of a trellis.

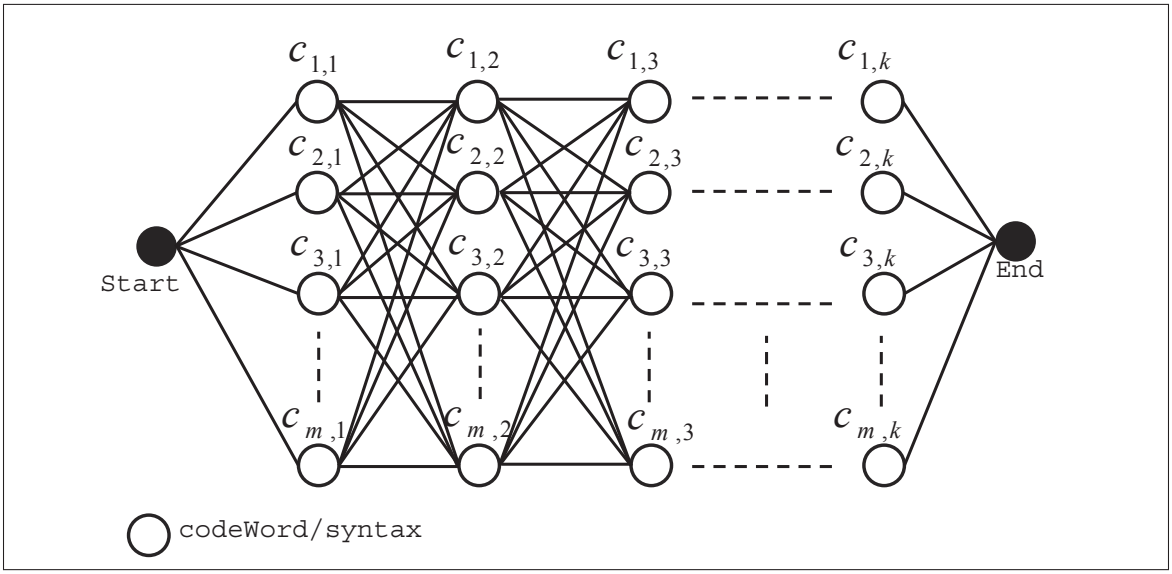


Figure 2.11 Trellis representation of the corrupted packet. $c_{i,j}$ represents the j-th codeword/syntax in i-the path.

Generally, the Hamming distance between the received corrupted codeword and the candidate codeword is considered as the weight for each path (Farrugia & Debono, 2008, 2010, 2011). Since the codewords have different length in bits, the generated paths may have a different number of codewords or different length as the number of consumed bits. Of course, it is obvious that some paths may die because of non-existing legal codewords in the variable-length code (VLC) tables. In order to eliminate the non-valid paths or prevent the exponential growth of the trellis, various constraints are considered in the literature. Farrugia & Debono (2011) proposed a trellis decoding strategy that consider M most probable paths with the smallest Hamming distance at each step of the decoding. In addition, they considered three source

constraints to validates the paths. The bitstream length, the number of MBs in the corrupted slice and the successful syntactic/semantic verification are checked to identify the likeliest path.

In some other works, JSCD was employed to only correct a specific part of the coded bitstream, such as residual information (Weidmann *et al.*, 2004; Wang & Yu, 2005) or motion vectors (Wang & Yu, 2005). In the approach proposed by Weidmann *et al.*, sequential decoding and the soft information provided by the channel decoder is used to find the best path for the residual coefficients coded with context-adaptive variable-length coding (CAVLC). Their proposed approach is based on the Extended profile of H.264 coded sequence. Furthermore, by utilizing its data partitioning feature, they assumed that other partitions are protected (error free), and only residuals are erroneous. The additional information provided by those protected partitions, such as packet length in bits and number of MBs in the slice, are used as the constraints to eliminate the non-valid paths. A MAP-based JSCD approach was proposed for the decoding of the motion vectors and CAVLC residual bitstream in (Wang & Yu, 2005) and (Bergeron & Lamy-Bergot, 2004), respectively. In (Wang & Yu, 2005), the authors modeled the neighboring motion vectors as a first order Markov process to utilize their high correlation features, and then, used data partitioning to transmit the horizontal and vertical motion vectors in separate partitions. Finally, at the decoder side, both motion vector partitions were decoded by their proposed MAP-Iterative JSCD approach. In (Sabeva *et al.*, 2006), JSCD combined with soft estimation techniques was adopted for correcting CABAC bitstreams of H.264 coded sequences. In all the previously-mentioned approaches (Wang & Yu, 2005; Bergeron & Lamy-Bergot, 2004; Sabeva *et al.*, 2006), the authors assumed that each packet contains a whole frame. This has been used as an extra constraint for eliminating the non-valid paths (since it indicates the number of coded MBs in each packet). However, such an assumption can dramatically increase the complexity of the approaches especially when the higher texture resolution sequences are considered and the packet size is extremely large.

Moreover, JSCD has been employed in the correction of the headers. Yen *et al.* (2012) proposed to exploit the syntaxes in two slice headers in H.264 coded sequences, known as sequence parameter set (SPS) and picture parameter set (PPS). Since the type and the number

of codewords in these slice headers are limited, therefore the trellis strategy can work well to find the best path and correct the errors in headers. In a more recent work, Perera *et al.* (2016) proposed a technique to correct the header information of high efficiency video coding (HEVC) coded sequence by using the syntactical conformance verification and soft information of turbo decoding.

However, all these approaches suffer from the major drawback of having a fairly large solution space for candidate packets, leading to a decoding process with extremely high computational complexity. Indeed, a packet containing N bits has 2^N possible candidates when any number of errors is considered. This issue alone restricts the use of these approaches in real-time applications (Caron, 2013). Recently, a significantly less complex approach has been proposed in (Caron & Coulombe, 2012, 2013, 2015), where the correction procedure has been considered at the syntax level instead of the whole slice. In their proposed maximum likelihood decoding (MLD) approach, the soft information of transmitted bits is used to select optimal syntax at each step of the decoding instead of listing the candidates for the whole slice. Therefore the solution space is limited to a set of valid syntaxes for each specific codeword. They extended the equation 2.10 in such a way that it measures the probability of finding likeliest syntax, s_i^*:

$$s_i^* = \underset{s_{\{t,i\}} \in C_i}{\arg\max} \left\{ P(S_r | s_{\{t,i\}}) \times \alpha \times P(s_{\{t,i\}} | s_{\{t,i-1\}}, s_{\{t,i-2\}}, ..., s_{\{t,1\}}) \right\} \tag{2.12}$$

where the transmitted slice, $S_t = \{s_{\{t,1\}}, s_{\{t,2\}}, ..., s_{\{t,N_t\}}\}$ is assumed to have N_t number of syntaxes. The weighting factor α is considered to compensate for the effect of variable length syntax candidates. The soft information of the transmitted bits (LLR) along with the channel bit error rate is used to calculate the first probability term in the equation. To evaluate the probability of each syntax in the second term, they modeled some of the syntaxes in the slice header and prediction syntaxes in the slice data and assumed that the non-modeled syntaxes (such as the residual syntaxes) have constant probability values. Their proposed greedy approach used in the correction process keeps only one syntax at each step of the decoding procedure. Although the greedy technique can further reduce the complexity of the procedure,

it increases the risk of reaching to an illegal or non-valid candidate syntax which causes to stop the correction process. In other words, any mistake in the decoding of a syntax will propagate to the following ones, and, there is no chance to correct the previously wrong decisions. They performed the simulations on a fixed packet size, 200 bytes and 300 bytes), and proposed to stop the correction process when the Hamming distance between the likeliest syntax and the received bits is greater than 1. In that case, the STBMA approach (Chen *et al.*, 2008) is integrated to conceal the remaining (not corrected) blocks. Their simulation results showed that both with and without soft information, their method referred as soft output (SO)-MLD and hard output (HO)-MLD respectively, can outperform the state-of-the-art error concealment (Chen *et al.*, 2008). The simulation results also confirmed that with the additional soft information better performance is expected compared to the HO-MLD approach. However, because of not modeling the residual syntaxes and utilizing a greedy approach, MLD performance will decrease with lower QP values or when the packet size is increased.

It is worth mentioning that an important issue with most error correction methods is the access (or lack of access) to the soft information. Propagating soft information, i.e. a fixed or floating point LLR value for each bit of the packet, throughout the protocol stack (from the physical up to the application layer), is complex to implement and deploy in practice.

2.3 Discussion

Although a lot of work was presented on error concealment and error correction approaches, there is still a lack of research on why keeping corrupted packet is useful and when it is more efficient to retain a corrupted packet. There are a few works in the literature that attempt to utilize the corrupted packet and perform partial error concealment. The idea is to recover all the correctly received MB before the error occurrence. This requires an error detection mechanism at the decoding stage which can be performed by the decoder itself. In the literature, much of the effort on the syntax analysis has been dedicated to error detection capability of the syntaxes (Superiori *et al.*, 2006; Barni *et al.*, 2000). Due to the fact that the sequences are coded using VLC, an error on a syntax element can create another wrong but valid codeword which

makes impossible for the decoder to detect the exact error location. In some cases, the error can be detected in 15 MBs after the error occurrence (Superiori *et al.*, 2006). Such a distance between error occurrence location and error detection location will drastically degrade the visual quality of reconstructed frame and its subsequent frames (since all the MBs between these two locations are wrongly decoded). Therefore, most error concealment approaches prefer to ignore the corrupted packet. In the next chapter, in chapter 3, we propose an approach where the decoder can fully use the corrupted packet for replacement in the lost area, and moreover, the proposed approach can be integrated with all the existing concealment approaches.

In the case of list decoding error correction approaches, we saw that the approaches suffer from the high solution space problem. Especially when the soft information is not available, all the bits have the same probability of being flipped and finding a final candidate is very complex. In chapter 4, we propose a novel checksum-filtered list decoding (CFLD) that can correct the corrupted packet by exploiting the UDP checksum value at the receiver side. Moreover, it will drastically reduce the candidate list compared to the conventional list decoding approaches from N to $N/32$ in the case of considering one bit in error.

CHAPTER 3

NON-DESYNC-BASED DECODING FOR H.264 CODED SEQUENCES

In this chapter, we describe how the corrupted packets can be exploited as a complement to error concealment approaches by filtering (keeping) those deemed to provide good reconstruction. We start by analyzing the context-adaptive variable-length coding (CAVLC) syntax elements in H.264 coded sequences and then propose filtering mechanism that can fully use the corrupted packet (under some conditions) for replacement in lost area.

There are a few works on exploiting syntaxes in the literature. Demirtas *et al.* (2011) examined the effect of an isolated error in each syntax element of H.264 on visual quality. By simulation on only low resolution QCIF sequences, they presented which syntax elements are less sensitive to an isolated error. And based on that, they concluded that if an error occurs on less sensitive syntax elements, decoding those corrupted packets leads to better quality than applying a slice level concealment approach. But the real problem is that when a corrupted packet is received, in most cases, it is not possible to find out which syntax element was actually hit by the error. Due to using variable-length codes (VLCs), the effect of an error may be detected in subsequent syntax elements. Therefore, their approach is not reliable. Imagine a scenario where the error happened on more sensitive syntax elements and detected only on a less sensitive one. Keeping (or decoding) the corrupted packet in this case will degrade the visual quality even more than performing concealment. Moreover, as it will be described later in this chapter, the sensitivity of a syntax to errors depends on how the syntax has been coded and how the following syntaxes depend on it. For instance, one isolated bit error on a specific part of an Exponential Golomb Code (EGC) syntax could cause direct desynchronization, which makes it highly sensitive to errors, while the other bits of the same syntax do not cause any desynchronization. They also showed that if an error hits these syntax elements, decoding those corrupted packets leads to better quality than using slice level concealment. Unlike Demirtas *et al.* (2011), we look for the least sensitive bits of each syntax element, the bits that errors on them will not cause any desynchronization.

In this chapter, the concept of non-desynchronizing bits (NDBs) is defined in the context of H.264 CAVLC coded sequences. An NDB is a bit that does not cause any desynchronization at the bit level and more importantly, that does not have any impact on the number of decoded macroblocks (MBs). In the first section, we identify the NDBs of most common syntax elements in H.264 baseline profile coded bitstreams. We determine what are the essential requirements for each bit to behave as an NDB. In the second section, we present the percentage of NDBs in typical video bitstreams. We then examine the effect of individual errors on the NDBs to confirm that they have a small impact on visual quality. The frequency of the NDBs and their effect on visual quality are also investigated. The proposed robust decoding approach, that retains the corrupted packets for which only NDBs are erroneous, is described in section three. Finally, the simulation results and a discussion are presented in the last section.

3.1 Non-desynchronizing bits in H.264 syntax elements

After prediction, transform and quantization, the H.264 video signal is represented as a series of transform coefficients along with prediction parameters. These values must be coded into a bitstream that can be efficiently transmitted or stored and can be decoded to reconstruct the video signal. There are several different mechanisms in H.264 to convert parameters into a compressed bitstream, namely: fixed length binary codes, variable length EGCs, CAVLC and context-adaptive binary arithmetic coding (CABAC).

As a consequence of the high coding efficiency, compressed bitstreams are extremely vulnerable to transmission errors. Note that there is no synchronization marker inside the slices, therefore, incorrect parsing of a codeword may bring desynchronization between the decoder and the encoder. But what is yet to be investigated is if all the errors on syntax elements lead to desynchronization. With this in mind, in this section, we look at the syntax elements in H.264 baseline profile to identify their NDBs. We concentrate on this profile since it is used by mobile terminals, which are more prone to transmission errors. In this work, a bit is identified as an NDB if after flipping it, it still satisfies the two following conditions:

- It does not cause desynchronization of the bitstream. And as will be seen later, this condition can be extended to the case where the desynchronization is restricted to just a few syntaxes.
- It does not change the number of decoded MBs in the slice.

In this section, we look at the syntax elements in H.264 baseline profile to identify their NDBs. We also specify the existing prerequisite for each particular syntax element to have an NDB. To simplify the procedure, note that, we assume that each slice, which is encoded and decoded independently from the others, contains one row of MBs. This condition will be relaxed later in this chapter.

It is worth mentioning that a similar analysis has been presented in (Lamy-Bergot & Bergeron, 2012) in which the authors identified *some* of the bits with less impact on the decoding, as the interchangeable bits, to use for the ciphering application. Although the concept of finding the NDB is slightly the same, here, we clearly defined the two conditions to determine an NDB, especially the second condition which we, later on in the following sub-section, explain its importance. Moreover, in this thesis, we are looking for *all* the possible NDBs in each syntax element of the Baseline profile with respect to some specific coding parameters, while in (Lamy-Bergot & Bergeron, 2012), the authors presented only some of the syntax elements. Therefore, some of our presented tables for NDBs in the following are exactly the same as (Lamy-Bergot & Bergeron, 2012) (such as Table 3.4) or slightly similar (such as Table 3.2 and Table 3.5). Since their study was targeting a different application and, more importantly, there was not any information about their encoding parameters, we are not able to use their identified sensitive bits results, and it was required to perform our own analysis and simulations based on our objective.

3.1.1 Guaranteed non-desynchronizing bits in H.264 syntax elements

The main syntax elements in H.264 are EGCs, which assign shorter codes to the more frequent values. The general structure of an EGC codeword is as follows:

$$\underbrace{0\,0\,0\,...\,0\,0}_{\substack{N\\ \text{zero-prefix}}} \quad \mathbf{1} \quad \underbrace{\mathrm{X}_1\,\mathrm{X}_2\,...\,\mathrm{X}_N}_{\substack{N\\ \text{INFO}}}, \qquad \mathrm{X}_i \in \{0,1\},\ \forall i \in [1,N] \tag{3.1}$$

The codeword starts with $N(\geq 0)$ zero bits as zero-prefix followed by a bit 1 (called here the middle bit), and then N bits of information as the INFO part. The complete codeword has length of $2N+1$ bits. The value of the codeword depending on the type of the EGC being signed or unsigned, named respectively as signed-EGC (SGC) and unsigned-EGC (UGC), can be decoded by these following steps (ITU-T SG16 Q.6 & ISO/IEC JTC 1/SC 29/WG11, 2003):

1. Count the number of zeros (N) until the first bit 1.
2. Read N bits after the first bit 1 (INFO).
3. $\mathrm{UGC} = 2^N + \mathrm{INFO} - 1$; $\mathrm{SGC} = (-1)^{UGC+1} \times \lceil \frac{\mathrm{UGC}}{2} \rceil$

where $\lceil\ \rceil$ is the ceil operator.

Some examples of EGC are presented in Table 3.1. It is obvious that if an error occurs in the zero-prefix part of an EGC or changes the middle bit "1" into 0, it will directly desynchronize the bitstream from that point forward because of incorrect parsing of the following syntax elements. However, an error in the INFO part (shown with X_i in equation 3.1), although leading to a different decoded value, does not have a direct desynchronization effect on the bitstream. Thus, all the INFO bits are *possible* NDBs.

The INFO bits of an EGC are not always categorized as NDBs. Suppose the bits "010" are related to the *mb_skip_run* syntax element which is coded by EGC. The value of the syntax indicates the number of skipped MBs before the next coded one. Note that for a skipped MB, no prediction or residual information is sent by the encoder. Therefore, in this example, the

Table 3.1 Examples of EGC mapping values to UGC and SGC. Bold bits are the *possible* NDBs of EGCs.

Bit Pattern	UGC	SGC	Bit Pattern	UGC	SGC
1	0	0	001**10**	5	3
01**0**	1	1	001**11**	6	-3
01**1**	2	-1	0001**000**	7	4
001**00**	3	2	0001**001**	8	-4
001**01**	4	-2	0001**010**	9	5

bits “010” means one MB should be skipped. By flipping the third bit “01**1**”, the number of skipped MB will change into two (as shown in Table 3.1). In this case, although there is no direct desynchronization at the bitstream level, there will be a shift of the successive MBs’ positions and the extra or reduced number of decoded MBs create a significant problem. This is often observed when the error happened on the *mb_skip_run* INFO part at the end of the slice). Therefore, we should consider this case in our definition of an NDB (as the second condition in the definition of NDB). Overall, based on the meaning of the syntax and how the subsequent syntaxes depend on it, the modification of an INFO bit may or may not cause bitstream desynchronization thereafter and this should be studied individually for each syntax element.

In the following, we will study the most common syntax elements in H.264 EGC-coded syntaxes. These syntax elements are: *mb_type* (inter/intra), *sub_mb_type*, *intra_chroma_pred_mode*, *coded_block_pattern* (inter/intra), *mvd_l0* and *mb_qp_delta*. A good overview of these syntax elements is presented in (Richardson, 2010).

***mb_type*:** The allowed MB types and their corresponding values for the *mb_type* syntax element in a P-Slice, which includes intra (I) and inter (P) prediction MBs, are listed in Tables 7-8 and 7-10 of the H.264 standard specification respectively (ITU-T SG16 Q.6 & ISO/IEC JTC 1/SC 29/WG11, 2003). In the standard, *mb_type* values 0 to 4 of unsigned EGC are assigned to the P–MB type and the values 5 to 30 are specified to the I–MB type syntax elements. Generally, a non-valid syntax value leads to a decoder crash. However, a valid but wrong *mb_type* value may cause syntax/semantic errors. For instance, if the type value describes a P–MB, mo-

tion syntaxes will be required at the next step of the decoding process, while there are no such syntaxes in the I–MB case. Similarly, the motion information in a P-16×16-MB is described by one motion syntax while for 16×8/8×16 and 8×8 MBs, two and four (without any subpartition block) motion syntaxes must be parsed respectively. For instance, in the *mb_type* case, the "00100" pattern describes a P–MB, and motion syntaxes are parsed at the next step of the decoding process, while no such syntaxes exist following the "00110" pattern, which describes an I–MB. Therefore, although the bold bit zero in "001**0**0" and the bold bit one in the "001**1**0" pattern are INFO bits of EGC, they are not categorized as NDBs in this case.

However, there are still some NDBs in *mb_type*. The bits are shown in Table 3.2 as bold bits. Consider the following example: flipping the least significant bit (LSB) bit of *mb_type*=01**0**, described as a P–16×8–MB (two blocks of 16×8) which later requires two motion syntaxes, into *mb_type*=01**1**, generates another valid syntax with P–8×16–MB requiring also two motion syntaxes. Thus these two syntax values are interchangeable which makes their LSB bit categorized as NDBs.

For the case of I–MB, different *mb_type* values result in different prediction modes, as well as, different values for the syntaxes related to the transform coefficient such as *cbp_chr* and *cbp_luma*. The Coded Block Pattern (CBP) value indicates, in the form of binary flags through the look up table, which transform blocks may have non-zero coefficients. And based on these two values (*cbp_chr*, *cbp_luma*), different syntaxes (such as *coeff_token*, *trailing_ones_sign_flag*, ...) must be decoded. Therefore, the NDBs should not cause any modification on the *cbp_chr* and *cbp_luma* value. In other words, the two interchangeable syntaxes value must have the same value for *cbp_chr* and *cbp_luma*. Moreover, changing the prediction mode may also result in desynchronization. It is not possible to change a type value from vertical prediction into horizontal prediction while there is not a left neighboring MB. This is an instance of semantic error that leads to decoder crash. In our case, only horizontal and DC prediction mode are interchangeable (since one row of MB is considered in each slice, so there is no top neighbor for vertical prediction).

According to the explanation given above, the *mb_type* value of 7(0001000) can change to 8(0001001) by flipping the LSB position without causing any desynchronization effect. This is because their CBP values for chroma and luma components are the same, as well as, their prediction mode is interchangeable (vertical to DC or vice versa). Therefore, their LSB is identified as NDBs. The other NDBs for an I–MB is shown in Table 3.2.

Although these NDBs do not have any effect on the bitstream it is obvious that it may cause some context modification because of the MB type changes but later (in Section 3.2), we will demonstrate that this modification is insignificant or restricted to a small area.

***sub_mb_type*:** In the H.264 standard, a P–8×8–MB can be split further into smaller block sizes as sub-partitions. This is addressed by another syntax element known as *sub_mb_type* (see Table 3.3). Like the *mb_type* for a P–MB, the *sub_mb_type* value implies the number of the motion vectors associated with the current MB. For example, an 8×8 block requires only one motion syntax while if it is partitioned into two 8×4 or 4×8 blocks, two motion syntaxes are needed. Similarly, the number of motion vector adds up to four when the block is split into four 4×4 sub-blocks. Therefore, only the two cases of *sub_mb_type*=1 and *sub_mb_type*=2 are interchangeable. We categorize their LSB bits as NDBs. These bits are represented in bold format in Table 3.3.

Table 3.2 NDBs of the *mb_type* syntax element for P–MBs and I–MBs in P–slices. For an I_16×16, the last column of the table shows the predMode, *cbp_chr* and *cbp_luma* value respectively. For an I–MB predMode can be Vertical(0), Horizontal(1), DC(2), Plane(3). Bold bits are NDBs.

mb_type value	Bit pattern	MB type description
P–MB		
0	0	P_L0_16×16
1	01**0**	P_L0_L0_16×8
2	01**1**	P_L0_L0_8×16
3	0010**0**	P_8×8
4	0010**1**	P_8×8ref0
I–MB		
5	00110	I_4×4
6	00111	I_16×16_0_0_0
7	000100**0**	I_16×16_1_0_0
8	000100**1**	I_16×16_2_0_0
9	0001010	I_16×16_3_0_0
10	0001011	I_16×16_0_1_0
11	000110**0**	I_16×16_1_1_0
12	000110**1**	I_16×16_2_1_0
13	0001110	I_16×16_3_1_0
14	0001111	I_16×16_0_2_0
15	000010000	I_16×16_1_2_0
16	000010001	I_16×16_2_2_0
17	000010010	I_16×16_3_2_0
18	000010011	I_16×16_0_0_1
19	000010100	I_16×16_1_0_1
20	000010101	I_16×16_2_0_1
21	000010110	I_16×16_3_0_1
22	000010111	I_16×16_0_1_1
23	000011000	I_16×16_1_1_1
24	000011001	I_16×16_2_1_1
25	000011010	I_16×16_3_1_1
26	000011011	I_16×16_0_2_1
27	000011100	I_16×16_1_2_1
28	000011101	I_16×16_2_2_1
29	000011110	I_16×16_3_2_1
30	000011111	I_PCM

Table 3.3 NDBs of *sub_mb_type* syntax element in P–MBs.

sub_mb_type value	Bit pattern	sub_MB type description
0	0	P_L0_8×8
1	01**0**	P_L0_8×4
2	01**1**	P_L0_4×8
3	00100	P_4×4

***mvd_l0*:** The motion vector is a key element for exploiting temporal redundancy in a P–MB and achieves significant compression. It is described as a pair of values to represent horizontal and vertical displacements of a block or MB, based on its position in the reference frame with *x* and *y* components. It is obvious that the adjacent blocks may have similar movement, which makes motion vector values correlated. Due to this property, H.264 encodes the difference motion vector instead of the actual one. The *mvd_l0* syntax (also known as *mvres*) is a pair of SGCs, one for the *x*, and the other for the *y* components, which represents the difference between actual calculated displacement and the predicted motion vector value from the available neighbors.

As mentioned previously, a bit error in the INFO part of the EGC (illustrated as X_i in equation 3.1) does not cause any desynchronization at the bit level. Therefore, all the INFO part bits of *x* and *y* components of the *mvd_l0* syntax are categorized as NDBs. However, a wrong motion vector can propagate to the neighboring motion vectors, since the difference motion vector has been coded and transmitted, and at the decoder side, the *mvd_l0* value extracted from the bitstream is added to the predicted motion vector value of neighbors (usually the median value of neighbors). Thus, an erroneous motion vector can easily change the following motion vector values without having any desynchronization effect on the bitstream. The error can thus affect significantly the visual quality. This kind of propagation will stop when there is an I–MB or when its contribution disappears in the median calculation process. This is discussed in detail in Section 3.2.

***coded_block_pattern*:** After inter/intra prediction, the residual blocks are coded using CAVLC. The *coded_block_pattern* syntax element is used to indicate which 8×8 blocks of a MB have

at least one non-zero transform coefficient. If there is such an 8×8 block, the CAVLC parsing process is started for each 4×4 sub-block of it. Although *coded_block_pattern* uses an EGC syntax, an error, even on the INFO part, can cause desynchronization. A different *coded_block_pattern* value means assigning the non-zero coefficients to the different 8×8 blocks. This can have an effect on the nC parameter which is calculated based on the number of non-zero coefficients in the top and left available neighboring blocks. The nC parameter value is important since it is later used in the decoding process to choose the lookup table for the following *coeff_token* syntax. In other words, any modification on the *coded_block_pattern* that changes the nC parameter may result in a desynchronization of the bitstream. Therefore, we are not able to identify any bit of the *coded_block_pattern* as NDB.

***coeff_token*:** The first parsing syntax element present in the block residual is *coeff_token*. The corresponding lookup table for decoding is selected between five different VLC tables based on the nC value. The pair value of the *coeff_token* syntax, as *TrailingOnes* and *TotalCoeff* (TC), signals respectively the number of coefficients with ± 1 values and the total number of non-zero coefficients (out of 16).

For each ± 1 value, a sign flag syntax must be later parsed. And, the rest of non-zero (and non ± 1) coefficients must be addressed by other "level" syntax elements. In addition, the value of TC is later used in the selection of the *total_zeros*' lookup table. Due to these dependencies, any changes in the *coeff_token* syntax can cause the decoder to lose its synchronization with the corresponding encoder either from the error position or from the following syntaxes.

***trailing_ones_sign_flag* (T1):** The sign of each *TrailingOnes* is signaled by a single bit (0/1) in *trailing_ones_sign_flag* (T1) syntax. The zero value (bit 0) of the syntax means that the corresponding coefficient is +1, otherwise, it is -1. Since *trailing_ones_sign_flag* syntax does not have any direct or indirect effect on the bitstream, its corresponding bits are categorized as NDBs.

***level_prefix*:** The value of each remaining non-zero coefficients (TC–T1) is specified by these two *level_prefix* and *level_suffix* syntaxes. The value of the *level_prefix* is determined from the

bitstream by the number of leading zero bits before the first "1". For example a "0001" bit patterns means *level_prefix* is equal to 3. As it can be seen from the example, any modification on the bits assigned to the *level_prefix* can easily desynchronize the bitstream from the error position and cause the error to propagate to the following syntaxes. Given that, we can say that there is no NDB in the *level_prefix* syntax. It is worth noting that there is a special case of modification on *level_prefix* for which the error propagation will be stopped shortly after parsing of a few syntaxes if some conditions are satisfied. This will be considered later as a special case.

***level_suffix*:** In the second step of coefficient level decoding, *n* bits are read to determine the value of the *level_suffix* syntax. The number of bits *n*, which is called levelSuffixSize in the standard, depends on the value of the previously decoded syntaxes such as *level_prefix* and another parameter SuffixLength which also depends on the *coeff_token* value (for more detail, see (International Telecommunications Union, 2003)). However, any modification on the *n* bits values themselves of the *level_suffix* syntax will not have any desynchronization effect on the bitstream. Thus all *level_suffix* bits are categorized as NDBs.

***total_zeros* (TZ):** Two different syntaxes, *total_zeros* (TZ) and *run_before*, are used to signal the remaining zero coefficients in each block. Due to the zig-zag scanning, the majority of the zero coefficients are at the end of the scan (in high frequencies) and H.264 does not actually code them (since they can be implicitly assigned by having the values of the previously decoded syntaxes). Only the zero coefficients from the last non-zero coefficients toward the start of the scanning (low frequencies) are coded and transmitted. The TZ syntax describes the total number of zeros before the last non-zero coefficient in the scanning order. The corresponding lookup table for TZ syntax is chosen based on the number of non-zero coefficients (TC value) in the current block. When there are more than one non-zero coefficients (TC>1), the *run_-before* syntax is also required, therefore different values of TZ can cause desynchronization at the parsing of the following *run_before* syntaxes. However, the TZ syntax element can have NDBs if the TC value is equal to 1. In this case, there is one non-zero coefficient and all the others are zeros. In other words, the TZ value demonstrates how many zeros are before the

non-zero coefficient. For instance, TZ=0011 means that there are three zero coefficients before the non-zero coefficient (Y), therefore twelve of them are after the non-zero coefficient.

0, 0, 0, Y, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

Flipping the LSB position of the TZ in the example above to TZ=0010 means that there are four and eleven zero coefficients before and after the non-zero coefficient respectively.

0, 0, 0, 0, Y, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

Therefore such a modification does not cause any desynchronization at the bitstream. Hence, the LSB of the two TZ cases (0011 and 0010) are categorized as NDBs. Note that, here, no *run_before* syntax is parsed. Table 3.4 presents all the identified NDBs of TZ syntax for both non-chroma-DC (sixteen coefficients of 4×4 blocks) and chroma-DC (four coefficients of 2×2 blocks) cases when TC=1.

Table 3.4 NDBs of the *total_zeros* syntax element as TC=1.

total_zeros value	Bit pattern	
	block 4×4 (non-chroma-DC)	block 2x2 (chroma-DC)
0	1	1
1	01**1**	01
2	01**0**	00**1**
3	001**1**	00**0**
4	001**0**	-
5	0001**1**	-
6	0001**0**	-
7	00001**1**	-
8	00001**0**	-
9	000001**1**	-
10	000001**0**	-
11	0000001**1**	-
12	0000001**0**	-
13	00000001**1**	-
14	00000001**0**	-
15	000000001	-

***run_before*:** When there are more than one non-zero coefficients in the block (and of course when TZ is bigger than zero), the zero transform coefficients between every two non-zero

coefficients are encoded by *run_before* syntax element. As it can be seen from the example in Figure 3.1, the TZ value expresses how many zero transform coefficients are present before the last non-zero coefficient (shown as Y_4) in reverse scan order, but it does not determine where they are exactly located. In the example, the TZ=7 specifies that the *run_before* can only take one of the eight values between 0 to 7. This is initialized into a parameter called zerosLeft (zerosLeft=7). The zerosLeft parameter, which describes the number of zeros preceding each non-zero coefficients, is used to choose the corresponding lookup table for the *run_before* syntax. The "100" bit pattern corresponding to the last column of Table 3.5 shows that the *run_before* is equal to 3, which means there are three zero transform coefficients between Y_4 and Y_3 and they are inserted as it can be seen from the example in Figure3.1. Then, this *run_before* value is subtracted from the zerosLeft value and the result (7-3=4) is re-assigned to the zerosLeft variable. This process continues until the zerosLeft is equal to zero (which means there is no other zero transform coefficient) or when we reach the first non-zero coefficient (Y_1). In the example of Figure 3.1, the latter happens. Note that the number of zero coefficients before Y_1 will not be encoded at all, since the final zeroLeft value demonstrates how many zero coefficients are present before Y_1 and they will be inserted.

In the example, we can see the effect of *run_before* value on zeroLeft parameter which furthermore can result in picking a different lookup table for the next *run_before* syntax. Thus, only the last *run_before* syntax, which decodes the zeros between Y_2 and Y_1, can have NDBs since there is no other *run_before* syntax after that. The NDBs of *run_before* syntax are represented in bold format in Table 3.5.

Let us consider another examples with flipping an NDB and non-NDBs of *run_before* syntax in Figure 3.1. If we receive the "1001000..." pattern instead of the "1001001...", the only difference between this case and the original one is the **value** of last *run_before* syntax which will change to 3 (corresponding "0**0**" bits at zeroleft=3) and accordingly the final zeroLeft will be 0. This modification does not cause any desynchronization at the bit level but inserts three zeros between Y_2 and Y_1 and no zero before Y_1 as follows:

$$Y_1, 0, 0, 0, Y_2, 0, Y_3, 0, 0, 0, Y_4, 0, 0, 0, 0, 0.$$

While a received pattern such as "**0**001001..." instead of "1001001..." can cause unpleasant desynchronization effect on the following syntaxes due to the remaining extra bits "001". The pattern is decoded as follows:

- Initialize zeroLeft=7, with the "**0**001" pattern, thus *run_before*=7 (seven zero transform coefficients are between Y_4 and Y_3).
- ZeroLeft=7-7=0 (which means there is no other zero transform coefficients).

$$Y_1, Y_2, Y_3, 0, 0, 0, 0, 0, 0, 0, Y_4, 0, 0, 0, 0, 0$$

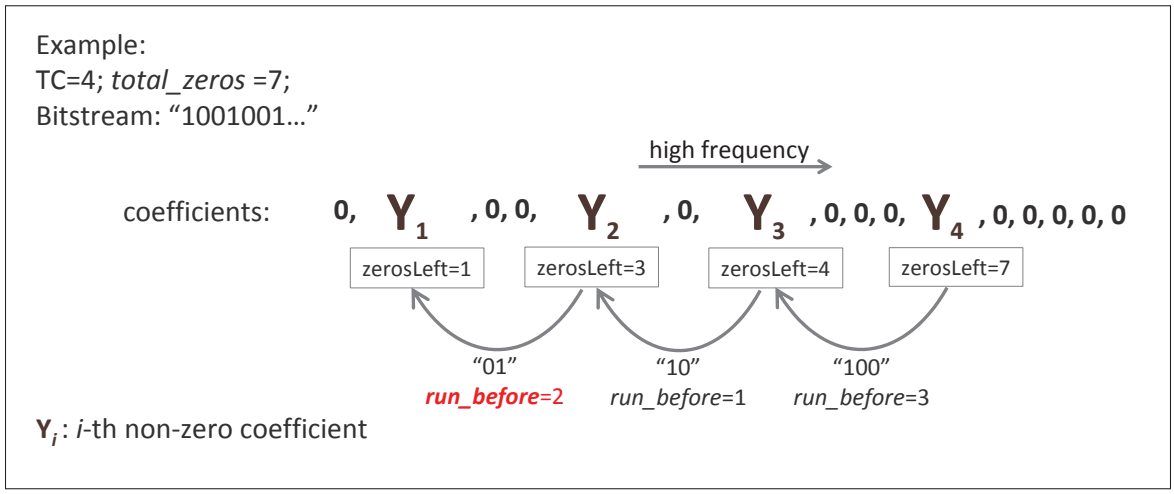


Figure 3.1 *run_before* example, the character Y_i specifies the *i*-th non-zero coefficient. Only the last *run_before* syntax can have non-desynchronizing feature. Its value can change by flipping the bitstream "1001001..." as "1001011..." (*run_before*=0) or "1001000..." or (*run_before*=3) without causing any desynchronization effect.

3.1.2 Contextual non-desynchronizing bits in H.264 syntax elements

In all the previous cases, the NDBs are considered for each syntax element individually. However, there is a possibility that the successive syntaxes can compensate for the desynchronization caused by each other. Therefore, the bitstream turns to be synchronized again and the error propagation will be restricted as well. This scenario happens under the condition that a

Table 3.5 NDBs of *run_before* syntax based on different zeroLeft values. NDBs are presented in bold format. Note that this is valid if the NDBs occur on the last *run_before*.

run_before value	**zeroLeft**						
	1	2	3	4	5	6	>6
0	**1**	1	**11**	**11**	1**1**	11	**111**
1	**0**	0**1**	**10**	1**0**	1**0**	**000**	**110**
2	-	0**0**	**01**	**0**1	0**11**	**001**	**101**
3	-	-	**00**	00**1**	0**10**	0**11**	1**00**
4	-	-	-	00**0**	0**01**	0**10**	**011**
5	-	-	-	-	0**00**	**101**	**0**10
6	-	-	-	-	-	**100**	**00**1
7	-	-	-	-	-	-	0001
8	-	-	-	-	-	-	00001
9	-	-	-	-	-	-	000001
10	-	-	-	-	-	-	0000001
11	-	-	-	-	-	-	00000001
12	-	-	-	-	-	-	000000001
13	-	-	-	-	-	-	0000000001
14	-	-	-	-	-	-	00000000001

combination of two or more successive syntaxes together, after one bit in error, still creates a semantically valid string of syntaxes which obviously should have the same length in total as pairs. Some of these group syntaxes are investigated in the following to identify their NDBs.

level_prefix and **TZ:** In fact, the *level_prefix* syntax element, individually, does not have any NDBs. However, when it is combined with the TZ syntax, some of its bits can behave as NDBs. When there is only one non-zero transform coefficient (TC=1) which is not ± 1 (*coeff_token*=(0,1)), there is no *level_suffix* syntax (since its parameter is zero). Therefore, after *level_prefix*, the TZ syntax is parsed. For specific values of the TZ, it can compensate the desynchronization error caused by flipping bits in *level_prefix* syntax. In other words, sometimes for a pair of (*level_prefix*, TZ), there is another pair that has the same length with only a single-bit difference at the bit level. Moreover, the other pair should not have any effect on the following syntaxes, such as leading to parse a different syntax than the one expected.

Let us consider the “00 01 1” bit pattern which describes the following syntaxes under the condition of *coeff_token*=(0,1). For the TZ and *level_prefix* bit patterns and values see Table 3.4 and Table 3.6, respectively.

- “00 01 1” parsed as: *level_prefix*=3 (“0001”); TZ=0 (“1”) → 5-bit length

The effect of flipping each leading zero bit in the *level_prefix*=3 is demonstrated here. The goal is to examine if there is another valid pair of (*level_prefix*, TZ) that can be replaced without causing any desynchronization effect.

1. “**1**0 01 1” parsed as: *level_prefix*=0 (“1”); TZ=3 (“0011”) → 5-bit length
2. “0**1** 01 1” parsed as: *level_prefix*=1 (“01”); TZ=1 (“011”) → 5-bit length
3. “00 **1**1 1” parsed as: *level_prefix*=2 (“001”); TZ=0 (“1”) → 4-bit length
4. “00 0**0** 1” parsed as: *level_prefix*=4 (“00001”) → more than 5-bit length; (need more bits to decode TZ syntax)

In the first two cases, the desynchronization effect caused by flipping bit in *level_prefix* syntax is compensated by the following TZ syntax, and after that point, it remains synchronized at the bit level. Therefore, these two bits are identified as NDBs in Table 3.6. In other words, the two pairs of *level_prefix*=3, TZ=0 (“0001,1”) and *level_prefix*=0, TZ=3 (“1,0011”) are interchangeable, and as expected, the two pairs are having the same length which is five bits here. Therefore, their different bit, first zero in *level_prefix*=3 and bit one in *level_prefix*=1, is identified as NDBs. The bits are shown in bold in the column of TZ=0 and TZ=3 of Table 3.6, respectively. Following the same logic, second zero bit in *level_prefix*=3 with TZ=0 and bit 1 in *level_prefix*=1 with TZ=1 are identified as NDBs.

In the two other cases, flipping bits in *level_prefix* will cause desynchronization from that syntax since they have a different length than the pair of (*level_prefix*=3, TZ=0). As it can be noticed from the third case, the TZ syntax is not able to terminate the error propagation

caused by flipping the third zero bit in *level_prefix*=3, and thus, the desynchronization effect, the remained bit '1', will continue to the following syntax elements.

The others NDBs for the *level_prefix* syntax, under the above-mentioned conditions, are presented in Table 3.6. Note that there is no NDB when the *level_prefix* value equals to 14 or 15 since it changes the parameter n corresponding to *level_suffix* syntax to a non zero value, and therefore the *level_suffix* syntax, not the TZ one, must be parsed after the *level_prefix*.

Table 3.6 NDBs of the *level_prefix* syntax element based on different value of TZ, if and only if *coeff_token*=(0,1).

level_prefix	TZ=0 "1"	TZ=1 "011"	TZ=3 "0011"	TZ=$2k+1<15$ "$\underbrace{00...0}_{k+1}11$"
0	1	**1**	**1**	**1**
1	01	0**1**	0**1**	0**1**
2	**0**01	00**1**	00**1**	00**1**
3	**00**01	000**1**	000**1**	000**1**
4	**000**01	0000**1**	0000**1**	0000**1**
5	**0000**01	00000**1**	00000**1**	00000**1**
6	**00000**01	000000**1**	000000**1**	
7	**000000**01	0000000**1**	0000000**1**	⋮
8	**0000000**01	00000000**1**	00000000**1**	*level_prefix* =
9	0**0000000**01	000000000**1**	000000000**1**	$13-(k+2)$
10	00**0000000**01	0000000000**1**	0000000000**1**	
11	000**0000000**01	00000000000**1**		
12	0000**0000000**01			
13	00000**0000000**01			

***mvd_l0* as pair:** We have mentioned earlier that a bit error in zero-prefix part of an EGC may cause desynchronization. But sometimes such error effect may be compensated by subsequent syntax elements. This can happen in *mvd_l0* syntax element. Since the syntax is coded as a pair value of (x,y), there is a possibility that the desynchronization effect produced by an incorrect value of the x component be compensated by the y component value. In fact, it is likely that a combination of two signed EGCs produces another signed EGC pair with the same length. Consider the following example that shows two pairs of interchangeable signed EGCs for the

mvd_l0 syntax.

$$mvd\_l0 = (-12, 0): \qquad 000\mathbf{0}11001, 1$$
$$mvd\_l0 = (-7, -1): \qquad 000\mathbf{1}110, 011$$

In this example, an incorrectly received value of -7 (instead of -12) begins desynchronization on the element *x*, but the value of -1 for the *y* component (instead of 0) will prevent the propagation of the bitstream level desynchronization to the following syntax elements. Obviously, this is because both pairs require the same total number of bits (10 bits) to describe such values. Therefore, their corresponding different bits (the bold bits shown in this example) can be identified as NDBs.

This can similarly be observed when two pairs of *mvd_l0* syntaxes must be parsed successively (i.e. in MB type of 16×8 or 8×16). The four successive signed EGCs have the chance to compensate for the desynchronization effect caused by one of their components, as shown in the following example.

$$mvd\_l0 = (1, -1); \quad mvd\_l0 = (0, 0): \qquad \mathbf{0}10, 011, 1, 1$$
$$mvd\_l0 = (0, 0); \quad mvd\_l0 = (-3, 0): \qquad \mathbf{1}, 1, 00111, 1$$

Thus, in this example, the bold bits behave as the NDBs.

total_zeros and ***run_before*****:** When two non-zero coefficients (TC=2) are available, the TZ syntax element and only just one *run_before* syntax (of course if TZ≠0) are parsed successively after decoding of the level value syntax elements. Therefore, some bit errors on the TZ syntax element, which cause desynchronization, may have a chance to be compensated by the next *run_before* syntax element. This is demonstrated in the following example (see Fig 3.1 and Table 3.5 for *run_before* and Table 3.7 for TZ bit patterns and values).

- "00**1**000001": TZ = 8 ("0010"); ZeroLeft=8, *run_before* = 8 ("00001")

- “00**0**000001”: TZ = 14 (“000000”); ZeroLeft=14, *run_before* = 6 (“001”)

As we see in the first example, the bold bit 1 can behave as an NDB only if the following *run_before* syntax element is equal to 8. This is because the remaining bits generate another valid value for the *run_before* which is 6, here in the example. In other words, the two pair values of (TZ=8, *run_before*=8) and (TZ=14, *run_before*=6) have the same length of bits, 9 bits, to represent the two syntaxes as bits. Hence, this new *run_before* value can terminate the desynchronization caused by an error in TZ, without having any effect on the following syntax elements. Note that any other value of *run_before* except 8, can not stop the desynchronization effect originated by the TZ in the example. Other NDBs of the TZ syntax element for the case of TC=2, along with the corresponding condition value on the succeeding *run_before* syntax elements are presented in Table 3.7.

This is also perceived when TC equals 3. In this case, it is possible again for the TZ to have NDBs if the two succeeding *run_before* hold some specific values to end desynchronization at the bit level. Consider the example below for this matter:

- “01**0**01101”: TZ = 4 (“0100”); ZeroLeft =4, *run_before*1 = 0 (“11”); ZeroLeft=4, *run_before*2 = 2 (“01”)
- “01**1**01101”: TZ = 7 (“011”); ZeroLeft=7, *run_before*1 = 4 (“011”); ZeroLeft=3, *run_before*2 = 2 (“01”)

Generally, any modification on TZ syntax element leads to a different value for ZeroLeft parameter. As a consequence, this may result in choosing different lookup table for the *run_before* (see Table 3.5). But as can be seen from the example, the desynchronization can be compensated under some specific values of the *run_before*. Table 3.8 shows the NDBs of TZ syntax element as TC=3.

Furthermore, the NDBs for the chroma–DC case is presented in Table 3.7. Note that in a 2×2 chroma–DC block, only four transform coefficients exist. Hence, when TC=2, TZ syntax

element can take value 0, 1 or 2. From our observation, only the two cases with TZ=1 and TZ=2 have NDBs, of course if their following *run_before* syntax element value is equal to 0. Otherwise, the desynchronization effect caused by flipping a bit in the TZ, which leads to a different lookup table for the succeeding *run_before* syntax element, will propagate to the following syntax elements. Similarly, as TC=3, the TZ can only be equal to 0 or 1. If the TZ=1, it requires *run_before* syntax element in the following decoding process, while in the other case (TZ=0) there is no such syntax element. Therefore, it is not possible to find any NDB in this case.

3.1.3 Other non-desynchronizing bits

Besides all the identified NDBs in the syntax elements presented in the previous subsections 3.1.1 and 3.1.2 as guaranteed and contextual NDBs respectively, there are still some other bits that irregularly behave as NDBs. In other word, depending on their neighboring decoded value or coding parameters, the modification of some bits will not cause any desynchronization while in some cases it will. This can be observed in syntax elements such as *coded_block_pattern* or *coeff_token*. Changing a *coded_block_pattern* value means assigning the non-zero coefficient to different 8×8 blocks. Depending on the number of non-zero coefficients in neighbors, the parameter nC will be calculated for each 4×4 block. nC is used as a lookup table index for the following *coeff_token* syntax element. Therefore, if this block replacement does not result in a different lookup table, then there will not be any desynchronization effect at the bit level. But this is not always the case and often the modification on the *coded_block_pattern* can lead to a completely different lookup table for *coeff_token*. Moreover, when more residual coefficients are available, the combinations of more than two successive syntax elements have the possibility to compensate the desynchronization effect caused by one another. In fact, it is not analytically easy to identify all the NDBs. However, we are still able to characterize the NDBs of the syntax elements by these definitions. If a bit is flipped and still satisfies the two conditions in the definition, which imply that 1) the corrupted packet is decod-

Table 3.7 The NDBs of the TZ when TC=2 and under some specific value of the succeeding *run_before* syntax, as presented.

TZ value	if *run_before* =	**TZ** bit pattern
non chroma-DC coefficients; 4×4 block size		
2	1, 2	10**1**
3	2, 3	10**0**
4	—	—
5	0, 2, 3, 4, 5	010**1**
6	0, 1, ..., 4	010**0**
7	0, 1, 2, ..., 6	001**1**
	7	00**11**
8	0, 1, 2, ..., 7	001**0**
	8	00**1**0
9	0, 1, 2, ..., 6	0001**1**
	7, 8, 9	000**11**
10	0, 1, 2, ..., 9	0001**0**
	6	00**0**10
11	0, 1, ..., 11	0000**11**
12	0, 1, ..., 11	0000**10**
	12	0000**1**0
	6, 7, 8	000**0**10
13	0, 1, ..., 11	0000**01**
	12, 13	00000**1**
14	0, 1, ..., 12	0000**00**
	13	00000**0**
	6	00**0**000
chroma-DC coefficients; 2×2 block size		
1	0	0**1**
2	0	0**0**

able, and 2) the number of decoded MBs in the packet is correct, then the bit is categorized as an NDB. In the following, we name all these bits as “Other NDBs”.

Table 3.8 The NDBs of the TZ when TC=3 and under some specific values of the succeeding *run_before* syntaxes, as presented.

TZ value	if (*run_before*1, *run_before*2) =	**TZ** bit pattern
2	(0, 2)	1**1**0
3	(0, 0)	10**1**
	(1, 2)	**1**01
4	(0, {0,1,2}); (1, {2,3})	01**0**0
5	(5,-)	**0**011
	(0, {2,3,4}); (1, {0,3,4}); (3, {1,2}); (4, {0,1})	0**0**11
	(2, {0,1,2}); (3, {1,2}); (4, 0)	001**1**
6	(6, -)	1**0**0
	(0, 0)	10**0**
7	(0, {4,5,6}); (1, {0,1,2}); (2, {4,5}); (3, {0,1})	0**1**1
	(4, {0,1,2}); (5, {1,2})	01**1**
8	(0, {7,8}); (1, 7)	00**1**0
	(4, {0,1,2}); (5, {2,3}); (6, 0)	001**0**
9	(0, {0,1,..,9}); (1, {0,1,...,8}); (2, {0,1,...,7}); (3, {2,3,...,6}); (4, {2,3,4}); (5, 3); (6, 0);	000**11**
	(4, {0,5}); (5, {0,1,4}); (6, {1,2}); (7, {1,2}); (8, 0)	0001**1**
10	(0, {7,8,9,10}); (1, {7,8,9}); (2, {4,7}); (3, {0,1})	0001**0**
	(0, {0,1,...,9}); (1, {0,1,...,8}); (2, {0,1,...,7}); (3, {2,3,...,6}); (4, {0,1,...,4}); (5, {0,1,4,5}); (6, {0,1,2}); (7, {2,3}); (8, 0)	0001**0**
11	(1, {6,7,8,9}); (3, {6,7,8}); (5, {0,2}); (6, {0,1});	000**00**1
	(1, 10)	00000**1**
12	(1, {6,7}); (3, 6)	00**0**01
	(0, {0,1,...,9}); (1, {0,1,...,8}); (2, {0,1,...,7}); (3, {2,3,...,6})	0000**1**
	(0, {7,8,...,11}); (1, {7,8,9}); (2, {4,7}); (3, {0,1}); (4, {7,8}); (5, 7); (7, {2,3,4}); (8, {0,3,4}); (9, {0,1}); (10, 1)	0000**1**
13	(1, {6,7}); (3, 6); (6, {4,5,6}); (7, {0,1,2}); (8, {0,1}); (9, 2)	00000**0**

3.2 Analysis of the non-desynchronizing bits

In this section, we will look first at the frequency of NDBs, and then at the effect of flipping each NDB on visual quality. To measure this, the first 60 frames of the following sequences, CIF (352×288) (*Foreman*), 4CIF (704×576) (*City*, *Crew*, *Ice)*, 720×480 (*Driving*, *Opening*-

ceremony, *Whale-show*) and 720×576 (*Walk*) are coded in IPPP... format (Intra refresh rate of 30 frames) using the H.264 Baseline profile with the Joint Model (JM) software, version 18.5 (Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, 2013). Each slice contains a single row of MBs, and is encapsulated into real-time transport protocol (RTP) packets.

3.2.1 Frequency of occurrence of the non-desynchronizing bits

In the first simulation, we sequentially invert all bits (flip individual bits one at a time) in each slice, and then the JM software is used to decode the corrupted packet. Each corrupted packet falls under one of the following categories:

1. It is not decodable (syntax/semantic error).
2. It is decodable, but the number of decoded MBs is not correct.
3. It is decodable, and the number of decoded MBs is correct.

Note that only the last category contains all the NDBs since it satisfies the two conditions in the definition of the NDB.

Figure 3.2 shows the percentage of each category for the *Crew* sequence at different quantization parameter (QP) values. As can be seen, flipping a single bit in the packet results in it being non-decodable in more than half of the cases. In other words, the flipping bit causes semantic or syntactic errors in the erroneous syntax element or maybe later in the following syntax elements. Nevertheless, this condition alone is not enough to detect the errors. For instance at QP=32, on average in 10% of the cases, flipping a bit does not cause any syntax or semantic error (the corrupted packet is decodable), but at the end, there are more or fewer than expected decoded MBs in the slice. This justifies the necessity for the second condition (decoding the right number of MBs) in our definition of the NDBs. This is important especially for higher QP values (for which there are less syntax elements compared to lower QPs), when the effect of an error can not be captured by only a syntax checker. As it can be seen in Figure 3.2, at

QP=37, around 16.3% of the error cases are detected by the second condition and this value reduces to 1% at QP=22.

On the other hand, on average, more than 30% of the bits belong to the third category, which means that flipping those individual bits will not cause any desynchronization at the bit level, and the number of decoded MBs is also correct. Similar percentages are observed for other video sequences.

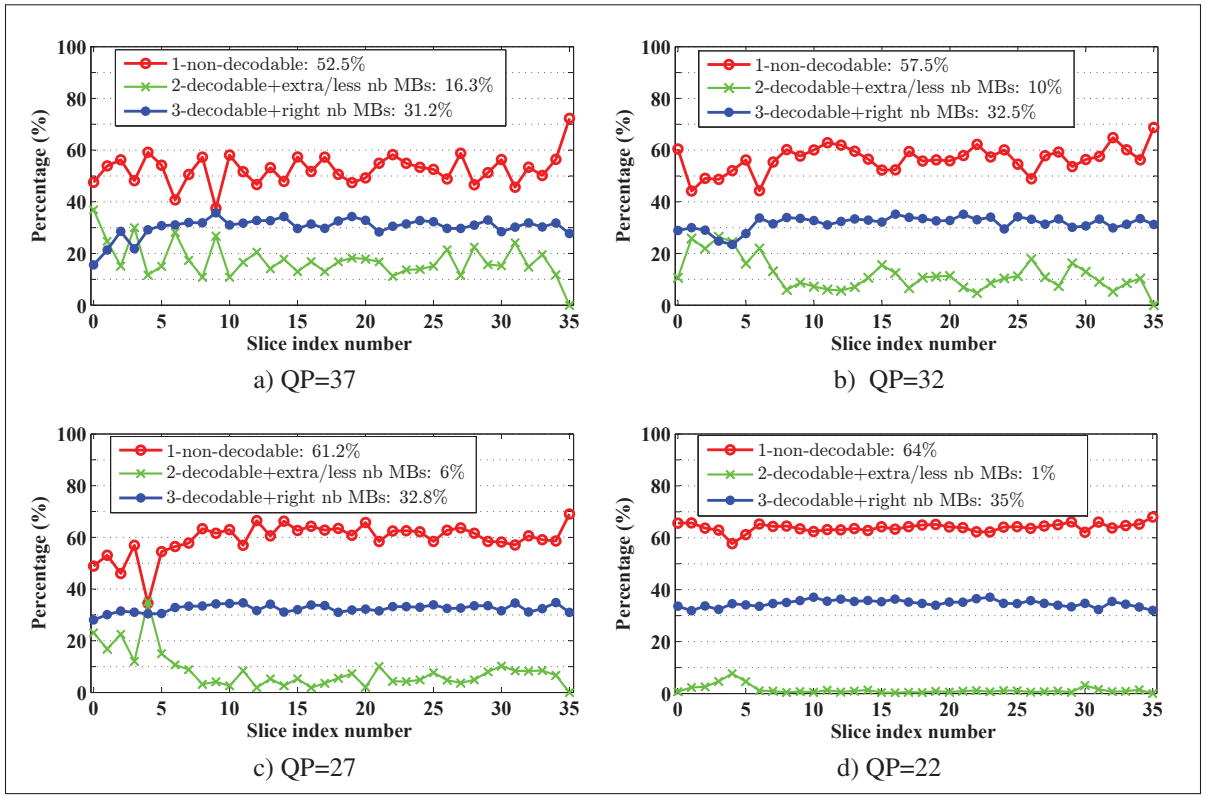


Figure 3.2 Percentage of the three different categories of a corrupted slice on frame index 44 of the *Crew* sequence at different QP values. The average percentage of each case is shown in the figure's legend.

Moreover, we divided the third category of "decodable + right number of MBs" into two sub-categories. The first sub-category includes the syntax bits which we have identified and described in the sub-sections 3.1.1 and 3.1.2 as guaranteed and contextual NDBs respectively, and we refer to them as "known–NDBs". The second sub-category, "other–NDBs", contains

all the other syntax bits which vary slice-by-slice and they do not constantly behave as NDBs (described in sub-section 3.1.3). As listed in Table 3.9, for a specific example, the first sub-category includes a predominant portion of the bits in the third category compared to the one for the second sub-category. On average on all QPs, 24.4% out of 32.9% of the bitstream belongs to the bits that we have identified as NDBs in the previous section (i.e. about 74% of all NDBs are identified NDBs). On the other hand, on average 8.5% of the bitstream are classified as the "other–NDBs". As can be seen, this percentage is higher in low QP values in which more residual syntax elements are available. This is because the desynchronization due to an erroneous bit in a residual syntax element is more likely to be canceled out by the following residual syntax elements. Therefore, the desynchronization will be limited to only a few syntaxes without violating the two conditions in the NDB definition. Overall, by averaging over different QP values for the *crew* sequence, we see that 32.9% of the bitstreams contain NDBs.

Table 3.9 Average percentage of the all NDBs (third decoding category) in two sub-categories on different QPs values of the *Crew* sequence at frame index 44.

Decoding Categories	**QP**				**Average**
	37	**32**	**27**	**22**	
known–NDBs (guaranteed and contextual)	25.4%	24.9%	23.8%	23.3%	24.4%
other–NDBs	5.8%	7.6%	9%	11.7%	8.5%
decodable+right nb of MBs (all NDBs)	31.2%	32.5%	32.8%	35%	32.9%

Furthermore, more detailed percentage values of "known–NDBs" sub-category are considered. In fact, the frequency of occurrence of each syntax element (i.e. their NDBs) has been investigated for the *Crew* sequence with different QP values and the results are presented in Table 3.10. As shown, the predominant NDBs are different for low and high QP values. For instance, at low QP values, they originate mostly from the residual syntax (T1), while at higher QP values the *mvd_l0* syntax constitutes the majority of the NDBs. Another noticeable simulation result is that the chroma bits constitute a very small portion of the "known–NDBs" bits, i.e. luma bits represent a strong majority.

Table 3.10 Frequency of occurrence of NDBs in each syntaxes for the *Crew* sequence at different QP values on frame index 44.

syntax element name	**QP**			
	37	**32**	**27**	**22**
guaranteed NDBs				
T1	5.1%	7.9%	10.4%	11.3%
mb_type	0.9%	0.8%	0.6%	0.3%
total_zeros	0.6%	1.2%	1.5%	1.2%
run_before	0.5%	1.5%	2.6%	3.5%
level_suffix	0.3%	0.4%	1.2%	3.1%
mvd_l0	17%	11.8%	6.2%	2.7%
nonsigned bit%, signed bit%	11.4%, 5.6%	7.5%, 4.3%	3.8%, 2.4%	1.5%, 1.2%
contextual NDBs				
level_prefix and TZ	0.1%	0.1%	0.1%	0.03%
mvd_l0 as pair	0.7%	0.7%	0.5%	0.2%
total_zeros and *run_before*	0.14%	0.5%	0.86%	1%
known–NDBs	25.4%	24.9%	23.8%	23.3%
Luma%, Chroma%	24.8%, 0.6%	24.2%, 0.7%	22.9%, 0.9%	22%, 1.3%

We have examined the frequency of occurrence of the NDBs in different video sequences and frames and similar results have been observed. Figure 3.3 presents the percentage value of "known–NDBs" in different frames of the *Crew* sequence at QP=32. The curve reveals that the percentage variation over all the P-frames (as shown for 31th frame to 48th frame) is very small. Therefore, it confirms that almost a quarter (25%) of the bitstream in the *Crew* are the "known–NDBs".

We conducted the same simulation for several other video sequences, and the percentage of "known–NDBs" are also presented in Table 3.11. Overall, the "known non-desync" bits constitute on average about 23% of a video bitstream.

3.2.2 Visual quality impact of erroneous non-desynchronizing bits

Although the NDBs do not have any effect on the bitstream level, they may cause some context modification. In this subsection, we evaluate the effect of flipping NDBs on the visual quality. An isolated error has been considered individually on each NDB in a coded sequence. Then,

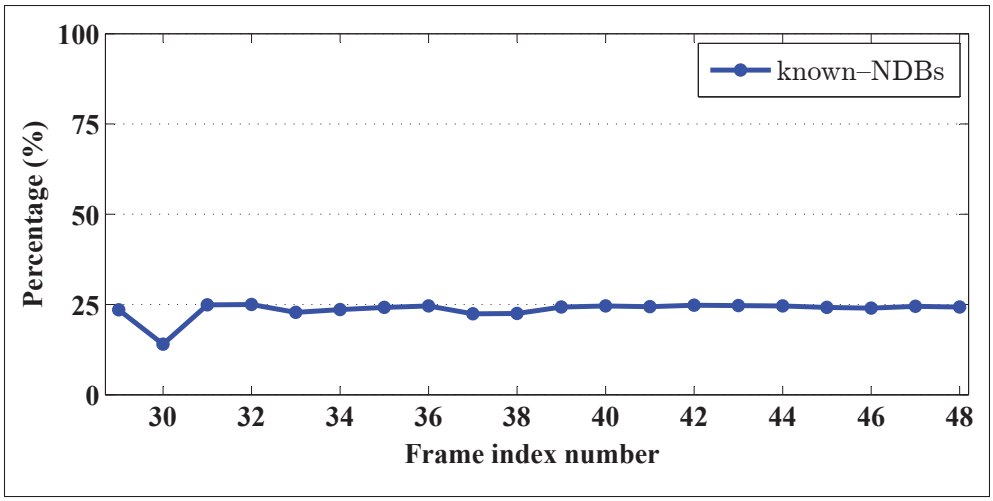


Figure 3.3 Percentage of "known–NDBs" bits on frame index 30 to 48 of the *Crew* sequence at QP=32. Note that frame index 30 is an I-frame with the refresh rate of 30 frames while the rests are P-frames.

Table 3.11 Average percentage of "known–NDBs" on frame index 44 for different sequences and different QP values.

sequence name (frame size)	**QP**			
	37	**32**	**27**	**22**
Crew (704×576)	25.4%	24.9%	23.8%	23.3%
City (704×576)	24.3%	24.1%	24.2%	24.6%
Ice (704×576)	25.0%	23.8%	22.9%	23.1%
Foreman (352×288)	26.7%	26.4%	24.0%	23.6%
Opening-ceremony (720×480)	23.0%	22.7%	22.6%	23.8%
Whale-show (720×480)	23.1%	22.2%	22.4%	24.2%
Driving (720×480)	23.8%	23.0%	22.7%	23.0%
Walk (720×576)	20.7%	19.8%	18.7%	19.6%
Mobcal (1280×720)	20.8%	23.7%	23.9%	23.2%
Average	23.7%	23.4%	22.8%	23.2%
Total average	**23.3%**			

the erroneous bitstream has been decoded and analyzed by calculating the peak signal-to-noise ratio (PSNR)[1] on the corrupted frame versus the original one. It is worth mentioning that since

[1] In this thesis, by PSNR of a method, we always mean the Y-PSNR with reference to the original frame.

the error is on an NDB, the corrupted packet is decodable and the number of decoded MBs in the corrupted packet is also correct; therefore the corrupted packet is kept.

Figure 3.4 depicts the percentage of packets having a specific range of PSNR degradation for each NDB (the difference between the PSNR of the corrupted and the intact frames calculated both with respect to the original one) in the *Crew* sequence. For QP values of 22, 27, 32 and 37, respectively, around 96%, 92%, 88% and 90% of the error events still lead to PSNR values very close to the intact value (with less than a 0.05 dB difference).

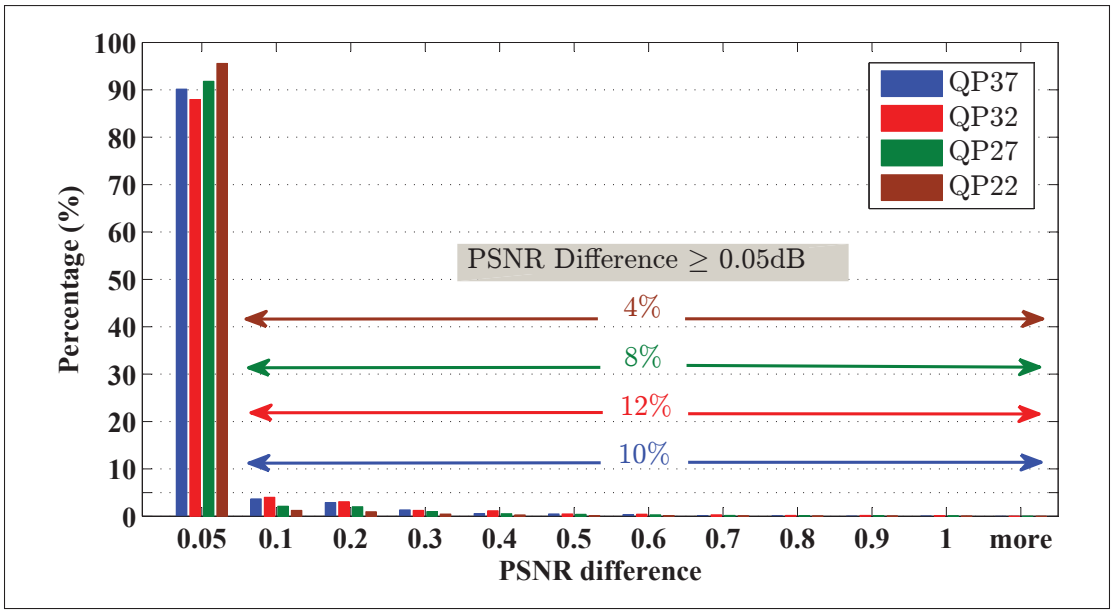


Figure 3.4 Percentage of PSNR difference of all NDBs against the intact case on frame index 44 of the *Crew* sequence.

As we can see, there are more cases with higher PSNR drops (>0.05dB) in higher QP values. In this context, Figure 3.5 represents a more detailed illustration of the PSNR differences for the case of high QP values such as 37 and 32. The figure depicts that the errors on "known–NDBs" are collectively more, for instance about 9% versus 3% at QP=32, compared to "other–NDBs" (since there are more "known–NDBs"). Therefore, it is helpful to look at the PSNR difference value of flipping each NDB in the syntax elements separately to identify the ones having more impact on the visual quality of the reconstructed frame.

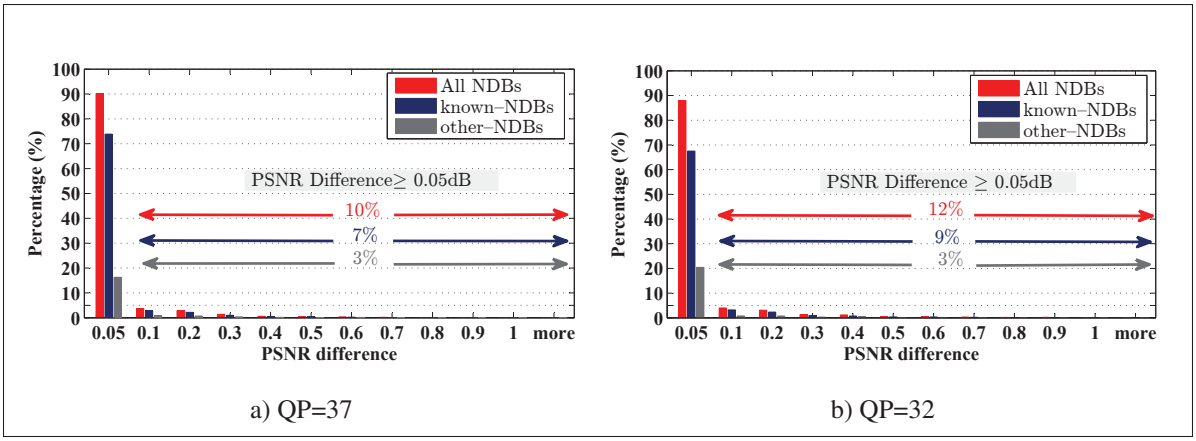


Figure 3.5 Percentage of all NDBs along with its two sub-categories of "known–NDBs" and "other–NDBs" on frame index 44 of the *Crew* sequence.

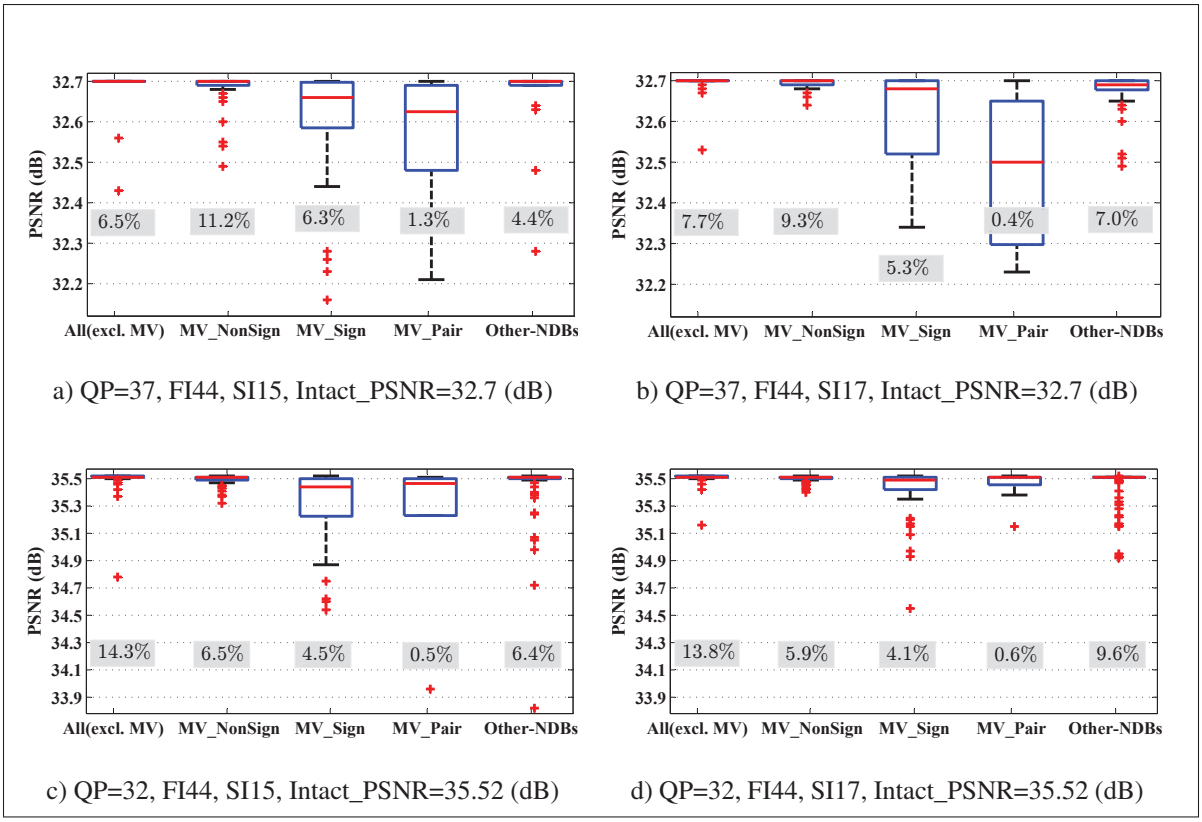
From different simulation results, we have observed that the NDBs of the motion vector syntax element (*mvd_l0*) are the most sensitive ones as compared to the other syntax elements in terms of the PSNR degradation. This is demonstrated in Figure 3.6. Correspondingly, we considered two distinct categories for the "known–NDBs" as (i) motion vector bits and, (ii) all the other bits (excluding motion vectors (MVs)), to be separately investigated. Furthermore, we divided the NDBs of the motion vector syntax into three sub-categories: sign bits (LSB), non-Sign bits and the special cases of *mvd_l0* as pair.

As we presented in the previous section, all the INFO bits of a *mvd_l0* syntax element are NDBs. And by knowing the fact that in the SGC, the LSB bit describes the sign value of the codeword, here in Figure 3.6, all the LSB bits of the *mvd_l0* syntax element are named as the MV-sign bits and all the other INFO bits of the *mvd_l0* are grouped as MV-NonSign bits. Moreover, the zero-prefix bits and middle bit one of a *mvd_l0* syntax element can sometimes behave as the NDBs, as described in sub-section 3.1.2, only when the following *mvd_l0* syntax element compensates for its propagation. Thus, all the NDBs regarding to this category named as MV-Pair in the figure.

Figure 3.6 shows the PSNR distributions (box plot) of two randomly selected slices in the *Crew* sequence for high QP values. As it can be seen in the box plot, an error on a motion vector (even on its NDBs) may strongly degrade the PSNR. This is because a wrong motion vector

value can simply propagate to the following motion vectors, which in turn comes from the fact that H.264 encodes the difference motion vector values of the adjacent blocks instead of the actual block movements (the difference between predicted motion vector and the median of the neighbors). This motion vector bit error propagation effect can be more severe when the error is on the sign bit. As can be seen for the case of error on the motion vector sign bits, there are more outliers (as shown with '+' red symbol) in the results, and the quality drops more severely as compared to the other cases. A wrong motion vector propagation effect can only stop when there is an I–MB or when the contribution of the wrong motion vector values disappears from the calculation of a subsequent motion vector.

Thus, an erroneous NDB of an MV, without having any desynchronization effect on the bitstream, may significantly affect the visual quality. However, it is worth mentioning that the percentage of motion vector sign bits is very small versus the whole bitstream, standing at around 4-6%, even in higher QPs (see the value inside Figure 3.6). Furthermore, not all of them can result in a significant PSNR degradation. As shown in Figure 3.6, the median value (red line in the middle of the box) of each box is very close to the intact one; as well, the lower and higher bands of boxes (25-75 percentile of the data) confirm that the effect of flipping NDBs on the visual quality is small and probably restricted to a very small area. Similar results as those presented in this section are obtained with other video sequences.



a) QP=37, FI44, SI15, Intact_PSNR=32.7 (dB)

b) QP=37, FI44, SI17, Intact_PSNR=32.7 (dB)

c) QP=32, FI44, SI15, Intact_PSNR=35.52 (dB)

d) QP=32, FI44, SI17, Intact_PSNR=35.52 (dB)

Figure 3.6 PSNR of all known–NDBs and other-NDBs on different slice of frame index 44 of the *Crew* sequence using box plots. All known–NDBs are depicted separately into four box plots in each figure. (a) frequency of known–NDBs=25.3%; (b) frequency of known–NDBs=22.7%; (c) frequency of known–NDBs=25.7%; (d) frequency of known–NDBs=24.4%. The breakdown percentage of the known–NDBs as well as the other-NDBs are shown inside the figures.

3.3 Proposed non-desync-based decoding framework

With all the new knowledge we have acquired on NDBs in the previous section, we now propose a robust decoding framework that retains the corrupted packets if the following two conditions are met:

1. the received corrupted packet is decodable (any non-valid syntax or semantic error will be detected in our modified decoder), and

2. the number of MBs in the corrupted slices is correct.

When those conditions are satisfied, which means there is no syntax or semantic error and the number of MBs is as expected, we decode and render the received corrupted packet (i.e. consider it as the best candidate). Otherwise, we discard it and perform error concealment.

The general schematic of the proposed approach is illustrated in Figure 3.7. We propose to add a new filtering step before each error concealment approach. First, the received corrupted packet is kept if a corrupted packet satisfies the two conditions (this can be simply validated by decoding the corrupted packet without any crash or error status), which means that, most probably, one of the NDBs in the packet is hit by the error. Since it is not likely that multiple errors each hit an NDB, and the quality associated with an NDB error is good, then we are pretty sure that quality is good as well. Therefore, we keep the corrupted packet only in this case instead of discarding it and performing concealment. Otherwise, one of the error concealment approaches is employed to handle the corrupted packet. It is worth mentioning that the proposed approach can be combined with any other error concealment approach.

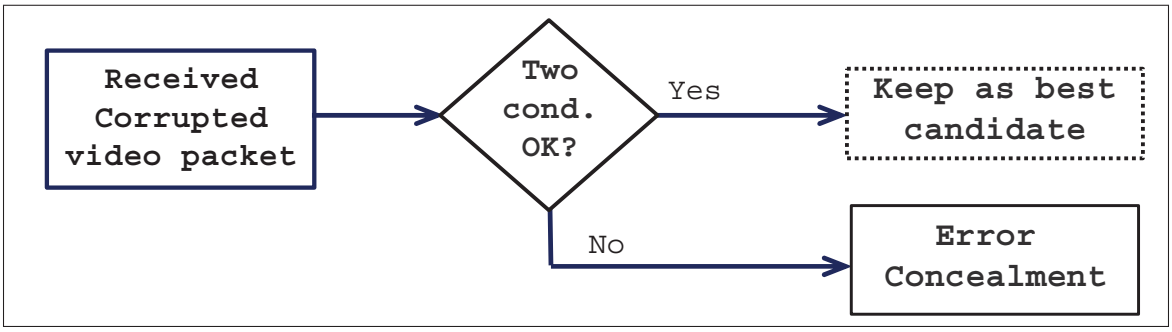


Figure 3.7 General schematic of the proposed approach.

In order to investigate the performance of the proposed approach on error concealment, two well-known error concealment approaches are employed during the simulation: (i) frame copy (FC) concealment by JM, (ii) a state-of-the-art concealment approach using the spatiotemporal boundary matching algorithm (STBMA) (Chen *et al.*, 2008). In Figure 3.8, we present various approaches which will be compared in the next section. The first approach (described as ① inside the figure) is the common frame copy concealment by JM in which the corrupted slice is

ignored and replaced by the same slice from the previous decoded frame. The second approach (described as ② inside the figure) is the proposed approach combined with JM–FC approach which will be named JM–FC$^+$. The approach ③ is a state-of-the-art concealment approach using the STBMA, which is a superior but complex method of MB-level error concealment (Chen *et al.*, 2008). The proposed strategy combined with STBMA is described as ④ in the figure which will be named as STBMA$^+$ in the following. The proposed design of the approaches JM–FC$^+$ and STBMA$^+$ helps us to find out the improvement of the proposed approach over each of those error concealment approaches separately. In other words, it will show how much gain each error concealment approach, individually, can get by retaining corrupted packets and our method deems adequate.

Note that when the received corrupted packet satisfies the two mentioned conditions, both proposed approaches JM–FC$^+$ and STBMA$^+$ are going to have the same performance on that packet. As another side note, when the corrupted packet does not satisfy the two conditions, the JM–FC$^+$ performs the same concealment as the JM–FC, therefore the two approaches will have the same performance on that packet (this is the same for STBMA$^+$ and STBMA).

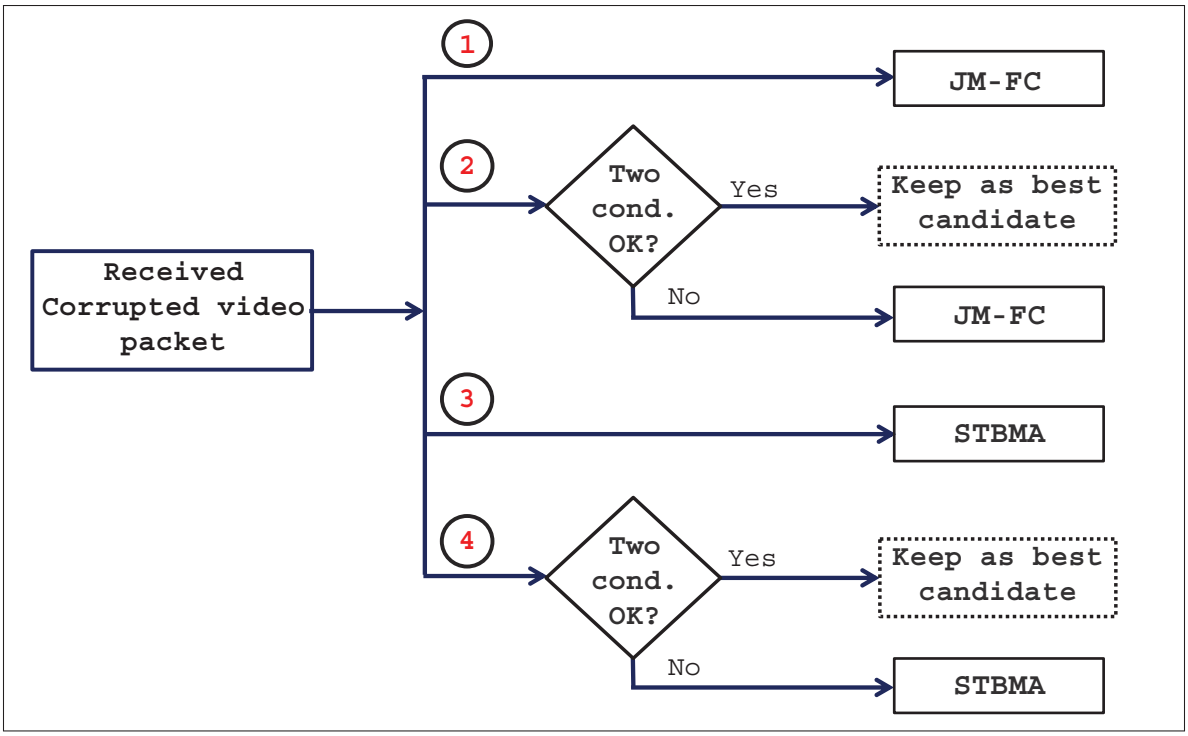


Figure 3.8 Schematic of different approaches that we use in our simulations. Approach ① and ③ are concealment by JM–FC and STBMA respectively. The approach ② and ④ named as JM–FC^{+} and STBMA^{+}, respectively, are the two proposed strategies for keeping a corrupted packet (under the two conditions) combined with the JM–FC and STBMA concealment respectively.

3.4 Simulation results

Since in our simulations, we coded the sequences with one row of MBs in each slice, we can easily verify the second condition presented at the beginning of section 3.3. Moreover, the number of MBs in the slice can be deduced from the information within other slices (the difference between the value of the *first_mb_in_slice* syntax element in the consecutive slices). Therefore, the proposed method applies even in the case where the number of MBs is not constant or known unless consecutive slices are damaged. The decoder validates the first condition by decoding the corrupted packet to find any syntax or semantic error.

For each QP, a random error (single-bit) is considered. In fact, there are several interleaving techniques that can combat the problem of burst errors in wireless communication channels by

fully randomizing the errors (Shi *et al.*, 2004; Kang & Sha, 2010). A single frame (between the 30th frame and the 60th frame) is randomly selected for error. Then, we apply a uniform error distribution on the bits of each packet with a channel residual bit error rate value varying between approximately 10^{-7} for small QPs, and 10^{-6} for large QPs, to obtain one bit in error when a packet is damaged. These residual bit error rates are much higher than those observed in some broadcasting systems, such as DVB-H and DVB-SH-A, in recommended operational conditions (Polák & Kratochvíl, 2011). The simulation is repeated 100 times for each QP, to ensure that the location of the erroneous bits did not bias our conclusions. The other characteristics of the encoded sequences are the same as mentioned in Section 3.2. Note that in this simulation, we compare the approaches by assuming that the corrupted packet contains *exactly* one erroneous bit. Later on in section 3.5, the effect of having more than one erroneous bit will be discussed.

Table 3.12 presents the average PSNR values for different error handling approaches. The last column in the table shows the percentage of times that the received corrupted packet satisfies the two conditions. As it can be seen, even in randomly applied error in different frames, on average 34% of the time the received packet was decodable and the number of MBs in the decoded corrupted packet was correct. As we mentioned earlier in the Section 3.2.2, these cases have very close PSNR to the intact one. Therefore, it is not reasonable to discard (or ignore) these good packets which occur frequently. The detailed results show that the proposed approaches JM–FC$^+$ and STBMA$^+$ outperform respectively JM–FC and STBMA approaches in all cases. Note that the PSNR difference between each method and JM–FC appears in parentheses in the table.

Figure 3.9 depicts the average PSNR gains of each approach over JM–FC for different QP values. We observe that keeping corrupted packets provides significant PSNR gains over JM–FC for all four QP values. For instance, when it is added to the STBMA approach (described as approach STBMA$^+$), it is more than 3.44 dB better than JM–FC at QP=22. On average, over all QPs, it offers a 1 dB gain improvement in approach JM–FC$^+$ and over 2.01 dB gain improvement in approach STBMA$^+$ over concealment in JM–FC.

Table 3.12 Comparison of the average PSNR (dB) of reconstructed corrupted frames versus the original one for different approaches. The PSNR gain differences between each method and approach JM–FC appear in parentheses. The last column shows the percentage of the cases that the received corrupted packet satisfied the two conditions.

Sequence	**QP**	**Average PSNR (dB)**					2 Conds.
		Intact	JM–FC	JM–FC$^+$	STBMA	STBMA$^+$	Met
City (704×576)	22	40.87	36.19	38.38 (2.19)	40.29 (4.10)	40.55 (4.36)	48%
	27	36.65	34.28	35.06(0.78)	36.45 (2.17)	36.50 (2.22)	37%
	32	33.05	32.04	32.33 (0.29)	32.98 (0.94)	33.00 (0.96)	31%
	37	30.05	29.55	29.66 (0.11)	30.01 (0.46)	30.01 (0.46)	25%
Crew (704×576)	22	41.78	39.21	39.93 (0.72)	40.64 (1.43)	40.97 (1.76)	28%
	27	38.53	37.09	37.61 (0.52)	38.03 (0.94)	38.22 (1.13)	38%
	32	35.69	34.96	35.23 (0.27)	35.44 (0.48)	35.53 (0.57)	31%
	37	33.00	32.64	32.73 (0.09)	32.86 (0.22)	32.89 (0.25)	28%
Ice (704×576)	22	43.70	39.18	40.58 (1.40)	41.74 (2.56)	42.28 (3.10)	36%
	27	41.44	38.00	39.27 (1.27)	40.05 (2.05)	40.52 (2.52)	30%
	32	39.00	36.50	37.51 (1.01)	38.15 (1.65)	38.50 (2.00)	35%
	37	36.43	34.37	34.95 (0.58)	35.77 (1.40)	35.98 (1.61)	34%
Foreman (352×288)	22	41.35	37.60	39.00 (1.40)	39.49 (1.89)	40.21 (2.61)	36%
	27	37.82	35.79	36.32 (0.53)	36.92 (1.13)	37.04 (1.25)	34%
	32	34.67	33.70	33.93 (0.23)	34.19 (0.49)	34.31 (0.61)	30%
	37	31.92	31.39	31.55 (0.16)	31.63 (0.24)	31.69 (0.30)	34%
Opening ceremony (720×480)	22	39.39	38.37	38.82 (0.45)	38.58 (0.21)	38.93 (0.56)	38%
	27	35.38	34.90	35.10 0.20)	35.02 (0.12)	35.16 (0.26)	34%
	32	31.39	31.20	31.27 (0.07)	31.26 (0.06)	31.30 (0.10)	33%
	37	27.69	27.64	27.65 (0.01)	27.65 (0.01)	27.66 (0.02)	31%
Whale show (720×480)	22	41.02	35.61	37.86 (2.25)	36.86 (1.25)	38.56 (2.95)	41%
	27	36.37	33.67	34.68 (1.01)	34.38 (0.71)	35.11 (1.44)	37%
	32	32.07	30.89	31.30 (0.41)	31.22 (0.33)	31.53 (0.64)	34%
	37	28.35	27.89	28.02 (0.13)	28.02 (0.13)	28.10 (0.21)	27%
Driving (720×480)	22	41.02	34.05	36.85 (2.80)	38.08 (4.03)	39.24 (5.19)	40%
	27	37.05	32.59	34.15 (1.56)	35.59 (3.00)	36.05 (3.46)	37%
	32	33.29	30.84	31.80 (0.96)	32.64 (1.80)	32.92 (2.08)	40%
	37	30.00	28.84	29.26 (0.42)	29.72 (0.88)	29.80 (0.96)	33%
Walk (720×576)	22	43.19	30.62	34.53 (3.91)	35.33 (4.71)	37.61 (6.99)	31%
	27	39.25	30.20	33.25 (3.05)	34.95 (4.75)	36.31 (6.11)	34%
	32	35.55	29.30	31.33 (2.03)	33.51 (4.21)	34.03 (4.73)	34%
	37	31.98	28.08	29.17 (1.09)	30.88 (2.80)	31.10 (3.02)	35%
Average PSNR gain over JM–FC (dB)				**1.00**	**1.60**	**2.01**	**34%**

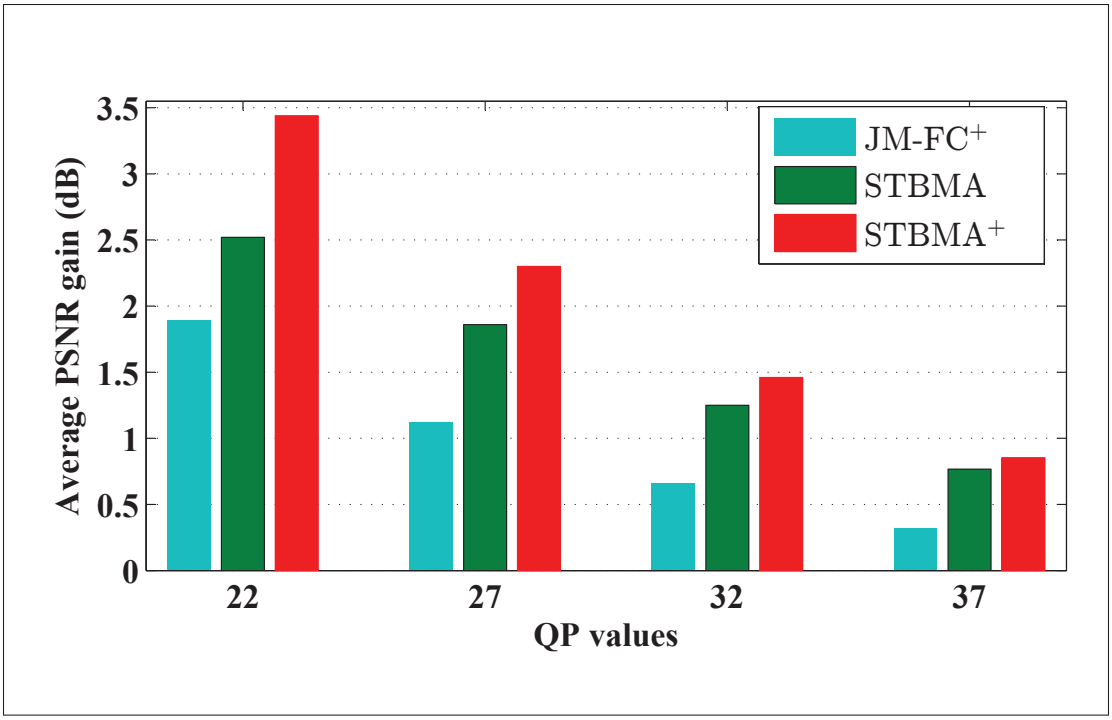


Figure 3.9 Average PSNR gain of all approaches over JM–FC for different QP values.

As it can be observed from Table 3.12, in some cases in higher QP values such as *Walk* sequence at QP=22, the average PSNR value of the proposed approach STBMA$^+$ or JM–FC$^+$ is still far from the intact one. This PSNR reduction mainly comes from the other 66% of the case, i.e. from those cases that the received corrupted packet does not satisfy the two conditions and the integrated concealment approaches such as STBMA or JM–FC are used in the proposed approach but failed to reconstruct well the lost information.

Figure 3.10 depicts the average PSNR drop in each approach compared to the intact frame when the received corrupted packet meets the two conditions (on average 34% of the case with random single-bit error). Note that in this case, approaches JM–FC$^+$ and STBMA$^+$ retain that packet as best candidate while JM–FC and STBMA will perform the concealment task. It is clear from the figure that the PSNR drop compared to the intact case, for the proposed strategy is less than 0.1 dB in all QP values while for a state-of-the-art concealment approach using the

STBMA alone it is between 1 dB to 1.8 dB. The reduction is even more, between 2.4 dB to 4 dB, as the basic simple concealment approach, JM–FC, is employed.

As separately presented in Table 3.13, on average on all QPs, the proposed approach brings an average gain of about 2.82 dB over JM–FC and 1.19 dB over STBMA (in 34% of the cases).

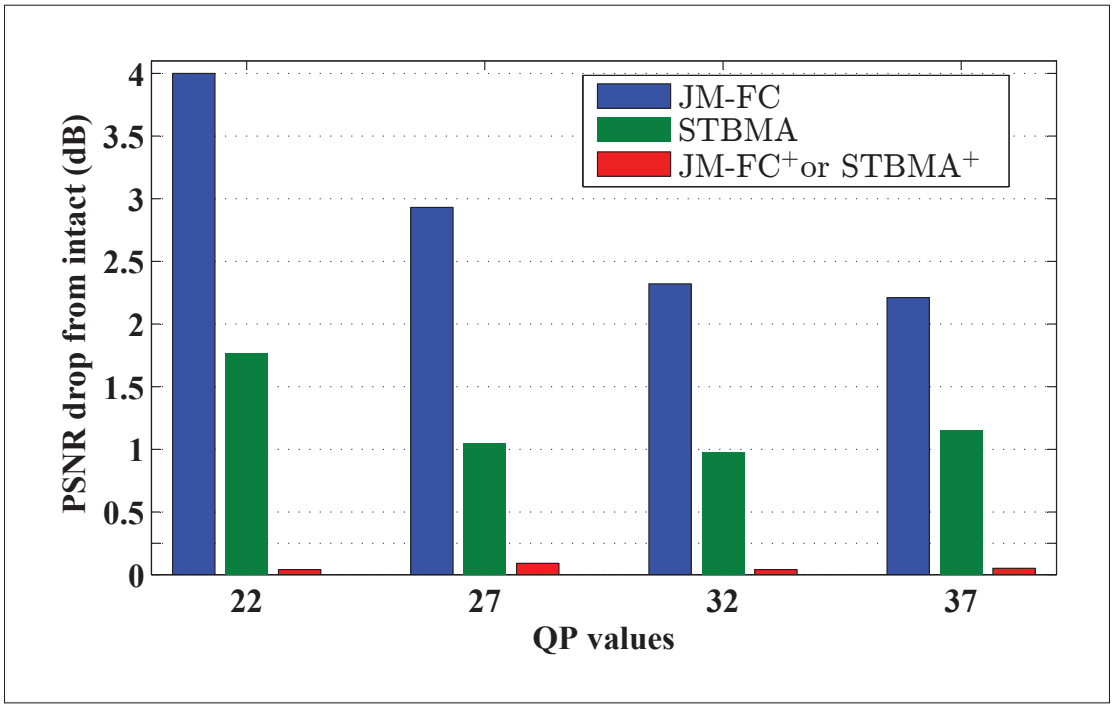


Figure 3.10 Average PSNR drop from the intact frame in all approaches for different QP values only when the two conditions are met.

The gain in visual quality is illustrated in Figure 3.11. In this case, a single bit error which occurred in frame 38, slice 15 and on the *coeff_token* syntax element. The received corrupted packet was decodable and the number of decoded MBs was correct (44 MBs in each slice). Since it satisfies the two conditions in the proposed strategy, both proposed approach JM–FC$^+$ and STBMA$^+$ will keep the corrupted packet as best candidate without doing any concealment. In contrast, JM–FC and STBMA will ignore the corrupted packet and perform only their individual concealment. The difference between the reconstructed frame and the intact one is also provided in the figure. Comparing the reconstructed frames by JM–FC and STBMA con-

Table 3.13 Average PSNR (dB) improvement over each error concealment method (JM–FC and STBMA) after considering to keep corrupted packets as in approach JM–FC$^+$ and STBMA$^+$ respectively.

QP	Two conditions met			All	
	Percentage	PSNR gain JM–FC$^+$ over JM–FC	PSNR gain STBMA$^+$ over STBMA	PSNR gain JM–FC$^+$ over JM–FC	PSNR gain STBMA$^+$ over STBMA
22	37%	3.97	1.74	1.89	0.92
27	35%	2.84	0.96	1.12	0.44
32	34%	2.28	0.94	0.66	0.22
37	31%	2.17	1.11	0.32	0.09
Avg.	**34%**	**2.82**	**1.19**	**1.00**	**0.42**

cealment approaches, it is clear that keeping the corrupted packet is a beneficial choice and it outperforms the two error concealment approaches. Furthermore, the error propagation effect is also shown in the figure (right side pictures), which confirms that the quality degrades drastically in the following frames (even after 10 frames) for the cases with error concealment. It is obvious that, keeping the corrupted packet (if it satisfies the two conditions) has a huge impact on the visual quality of the reconstructed corrupted frame, and more importantly, reduces the propagation of errors to subsequent frames due to the predictive coding.

From the results of all figures and tables, it can be inferred that for around one-third of the cases, the received corrupted packet is valid (satisfies two conditions), going through extra processes in concealment does not provide better results than the received one. As the results have shown, the received corrupted but valid packet is going to have a PSNR value very close to the intact one in most cases and we can not go beyond that. So, as a result, keeping the corrupted packet provides a significantly higher PSNR value and better quality compared to error concealment approaches alone. This is important not only for the corrupted frame, but for the following ones, as fewer visible drifting effects will result as shown in Figure 3.11.

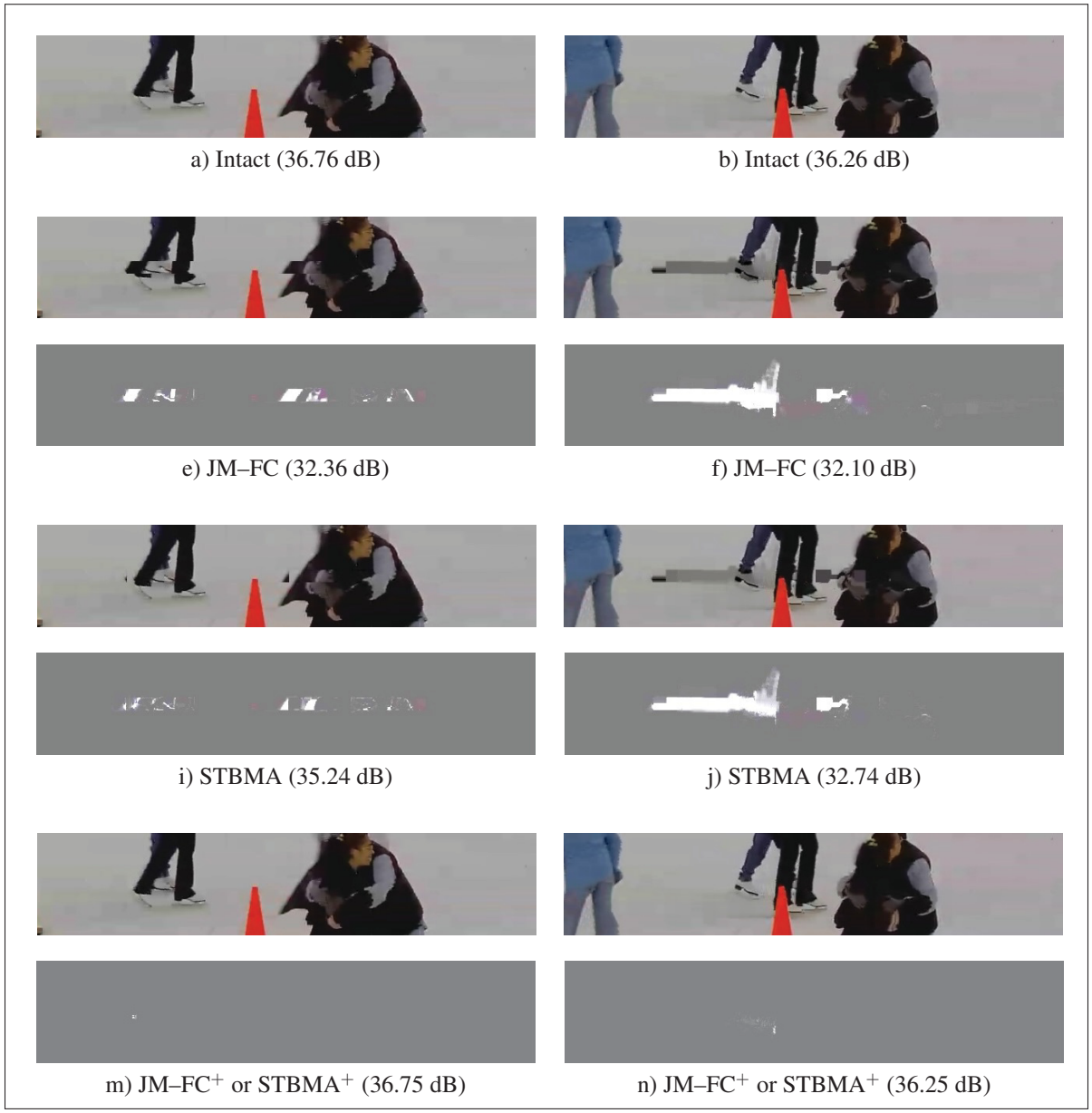


Figure 3.11 Visual comparison of a reconstructed frame in *Ice* sequence at QP=37 for different methods. In this case, the one bit error occurred in frame 38, slice 15 and on the *coeff_token* syntax element. The received corrupted packet satisfies the two conditions (it is decodable and the number of decoded MBs is correct (44 MBs)). The pictures on the left side of the figure are showing the reconstructed frame 38 by different approaches. In each case, the difference from intact are also captured by stream analyzer and provided. The pictures on the right side of the figure are showing the error propagation in each case. In fact, twenty frames after the corrupted frame, frame 59 is captured to demonstrate the effect of the error on the following frames.

3.5 Discussion

Although in this chapter we have considered the effect of one erroneous bit, the proposed approach can still work with the case with more erroneous bits in the slice. Due to the fact that the syntax elements have short lengths, and we are looking at low ρ (channel residual bit error rate) value, most of the time two different syntax elements will be hit by errors. For instance when there are two bits in error in a slice, the following cases are possible:

- Both errors are happening on NDBs. Therefore, the decoded corrupted packet must be decodable with the right number of encoded MBs (the two defined conditions in section 3.3 are met). The proposed approach can keep the corrupted packet. The PSNR drops of remaining the corrupted packet is still small, since both erroneous bits are on NDBs. As we presented in sub-section 3.2.1, on average 30% of the bits in a coded packet are belongs to NDBs, then the probability of having this case would be around $(0.3)^2 = 0.09$.

- Both errors are on the desynchronizing bits (the bits that are not NDBs). Therefore our modified decoder is able to detect any syntax violation and perform error concealment. Compared to the other cases, this case is very likely to happen with the probability of $(0.7)^2 = 0.49$.

- An NDB and a desynchronizing bit are hit by error. In this case with the probability of $2 \times 0.3 \times 0.7 = 42$, because of that desynchronizing bit, the two defined conditions will not be satisfied and the proposed approach will ignore the corrupted packet and perform concealment approach.

It is worth mentioning that, in the last two cases, there are a very small chance that the desynchronizing effect of one error be compensated by the other error, and at the end, the two defined conditions are met (like the case that we have discussed as “other–NDBs” in sub-section 3.1.3). Therefore, the error will be restricted to an area between the two bits. But as we shown in sub-section 3.2.1 this is very rare case even for one-bit in error case.

In general, when the bit error rates are higher and the residual error in the slices are more, it is very less probable ($(0.3)^n$ for n bit error) that the two conditions are met. Therefore the proposed approach will always end up performing concealment. This is reasonable, due to using VLC in H.264 coded sequences, the bitstreams are extremely sensitive to the errors and the effect of a wrong decision, even on one bit, can severely degrade the visual quality of the reconstructed frame. Thus, in higher error case, error concealment is always a better choice compared to correction approaches. Here in this chapter, we have tried to extract the non/less sensitive bits in coded sequences and demonstrated under what conditions it is more reliable to keep the corrupted packets and presented the maximum PSNR gain achieved (which happens in low bit error rates) by the proposed framework over some known concealment approaches. In higher error cases, the proposed approach will have the similar performance as the employed concealment one.

In the next chapter, we propose a correction approach (by exploiting the checksum value) that can correct erroneous packets, especially when the received corrupted packet is not satisfying the two conditions (the errors are not on NDBs).

CHAPTER 4

CHECKSUM-FILTERED LIST DECODING

In this chapter, we present our proposed novel list decoding approach for video error correction. The proposed approach exploits the receiver side user datagram protocol (UDP) checksum to alleviate the large solution space problem of the conventional list decoding approaches. We start the chapter with a detailed introduction of the internet checksum, the definition and properties, and then we focus on the UDP checksum and derive the essential mathematical expressions related to its definition and calculation. In the second section, we describe how UDP checksum can be applied to error correction. We define different bit error events, up to three-bit errors, and calculate their corresponding checksum values. In the third section, we mathematically prove the most probable bit error events, considering the observed checksum values. The proposed system for checksum-filtered list decoding (CFLD) is described in the fourth section. And finally in the last section, the performance of the proposed checksum-based approach for error correction, compared to other state-of-the-art methods is provided.

4.1 Internet checksum calculation and properties

Internet checksum is used by different standard protocols (Internet protocol (IP), transmission control protocol (TCP), UDP) for error detection (Braden *et al.*, 1989). Internet checksum, which is a fixed-length tag added to a message at the transmission side, enables the receiver to verify the integrity of the delivered message by recomputing the tag and comparing it with the tag that was sent. In this section, we present how the Internet checksum is computed, along with its mathematical properties, which will be exploited later. Although the following principles are applicable to other checksums, we will focus specifically on the UDP checksum.

4.1.1 Internet checksum definition and mathematical properties

The Internet checksum is a 16-bit field within the protocol header, and is defined as one's complement of the one's complement sum of all the 16-bit words in the computation data (Braden

et al., 1989). More specifically, the Internet checksum is calculated at the transmission side as follows:

- Divide the data into chunks of 16-bit words. If necessary, pad the data with one byte zero at the end to make it a multiple of 16 bits.
- Perform one's complement sum over all the words. If an overflow occurs during any sum, the ones' complement sum operation involves an "end-around carry". The end-around carry scheme routes the carry-out signal of the most significant bit (MSB) position c_n to the least significant bit (LSB) position, where it is used as a carry-in signal c_0 (Fall & Stevens, 2011).
- Flip all the bits of the final sum (one's complement).

Note that during the calculation of the checksum at the transmission side, the checksum value in the checksum field is set to zero, and after the calculation of the checksum, it is replaced by the computed one for transmission. The validation process at the receiver side is performed using the same algorithm, except that the received checksum field value is used in the computation of the checksum, rather than zeros. Received data is valid if the recomputed checksum at the receiver side is zero, otherwise the data is corrupted.

Mathematically, the set of 16-bit values, represented here in hexadecimal for the sake of convenience[1], $V = \{0000, 0001, \ldots, FFFF\}$ and the one's complement sum operation (denoted as +), together form an Abelian group (commutative group) (Dean, 1966).

An Abelian group is a set V with a binary operation + satisfying the following properties:

- For any $a, b \in V$, we have $a+b \in V$ (closure).
- For all $a, b, c \in V$, we have $a+(b+c) = (a+b)+c$ (associativity).
- For each $a \in V$, there exists an element $e \in V$ such that $a+e = a$ (identity element).

[1] Four-digit numbers in this thesis represents hexadecimal numbers

- For each $a \in V$, there exists $b \in V$ such that $a+b=e$, where $e=$ FFFF. We denote $b=\overline{a}$ as the inverse element of a, which is obtained by performing a one's complement on element a (flipping all its bits).
- For all $a,b \in V$, we have $a+b=b+a$ (commutativity).

Interestingly, in this Abelian Group, there are two identity elements, 0000 and FFFF, which correspond to the same zero (+0 and -0) value. In several references, it is mentioned that the identity element is unique. This is rather a consequence than a rule and since these identity elements correspond to the same value, the Abelian group's properties are still met.

There are yet other properties that can be deduced from the Abelian group, namely:

- **Property 1**: For all $a,b \in V$, we have $\overline{a+b}=\overline{a}+\overline{b}$.
  Proof: Let $c=a+b \ \rightarrow \ c \in V$ (closure property). Thus there is an inverse for c: $c+\overline{c}=e$ where $e=$ FFFF (inverse element). Then replacing c by $a+b$: $(a+b)+\overline{a+b}=e$. Adding $\overline{a}$ and $\overline{b}$ to each side of the equality gives: $(\overline{a}+\overline{b})+(a+b)+\overline{a+b}=(\overline{a}+\overline{b})+e \ \rightarrow \ (\overline{a}+a)+(\overline{b}+b)+\overline{a+b}=(\overline{a}+\overline{b})+e$. Using the inverse and identity properties, the expression leads to: $e+e+\overline{a+b}=\overline{a}+\overline{b}+e \ \rightarrow \ \overline{a+b}=\overline{a}+\overline{b}$.

From Property 1, it follows that:

- **Property 2**: For all $a_i \in V$, we have $\overline{\sum_i a_i}=\sum_i \overline{a}_i$
  Proof:

$$\overline{\sum_i a_i}=\overline{\sum_{i\neq 0} a_i+a_0}=\overline{\sum_{i\neq 0} a_i}+\overline{a}_0=\overline{\sum_{\substack{i\neq 0\\ i\neq 1}} a_i+a_1}+\overline{a}_0=\overline{\sum_{\substack{i\neq 0\\ i\neq 1}} a_i}+\overline{a}_1+\overline{a}_0=\ldots=\sum_i \overline{a}_i$$

4.1.2 User datagram protocol checksum definition and calculation

The UDP checksum is a 16-bit field in the UDP header, and it is the one's complement of the one's complement sum of the pseudo UDP header, the UDP header and the application data

message as it is well-defined in RFC 768 (Postel, 1980). Figure 4.1 shows the UDP datagram and its 12-byte prefix as a pseudo UDP header. The pseudo UDP header contains the source and destination IP addresses, the protocol, and the UDP length. This information initially comes from the IP header. The UDP checksum is calculated over all the segments shown in Figure 4.1.

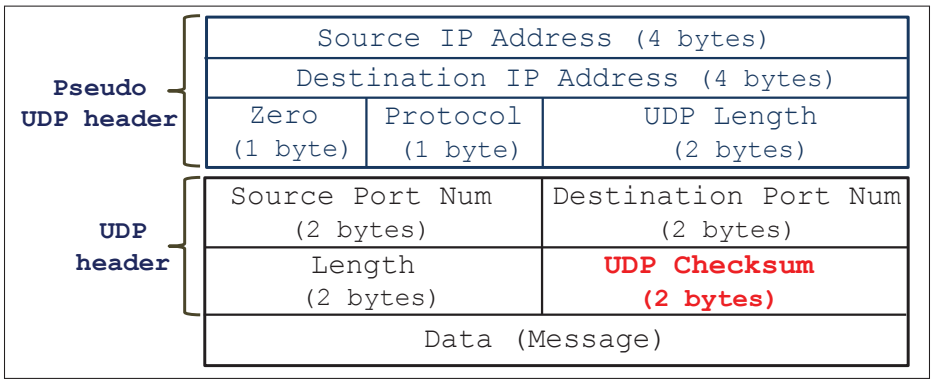


Figure 4.1 UDP datagram and pseudo header.

Like the Internet checksum, the checksum field of the UDP header should also be initialized to zero before the calculation, and then set to the calculated value prior to transmission. Since the UDP checksum is optional, a zero transmitted checksum value means that it was disabled. Therefore if the computed checksum is zero, as it mentioned in the standard, it should be transmitted as all ones (FFFF) (Postel, 1980). Note that the calculated checksum for a real packet can never be FFFF (i.e., the sum prior to the final ones' complement can never be zero) unless all the words in the packet are zeros (Fall & Stevens, 2011).

Let us assume that the UDP packet has a length of N bits (including padding), which is made up of m=N/16 16-bit words as $\{W_0, W_1, ..., W_{cs}, ..., W_{m-1}\}$ and W_{cs} is the checksum value in the checksum field. The i-th word and its inverse are respectively defined as:

$$W_i = \sum_{c=0}^{15} (w_{i,c} \times 2^c), \qquad \overline{W}_i = \sum_{c=0}^{15} (\overline{w}_{i,c} \times 2^c); \qquad w_{i,c}, \overline{w}_{i,c} \in \{0,1\} \tag{4.1}$$

where $\overline{w}_{i,c}$ represents the inverse of $w_{i,c}$, i.e., $\overline{w}_{i,c} = 1$ when $w_{i,c} = 0$, and $\overline{w}_{i,c} = 0$ otherwise.

The transmission side's checksum (C_T) can be expressed as shown in equation 4.2:

$$C_T = \overline{\sum_{i=0}^{m-1} W_i} = \overline{\sum_{\substack{i=0 \\ i \neq cs}}^{m-1} W_i} = \overline{\sum_{\substack{i=0 \\ i \neq cs}}^{m-1} \sum_{c=0}^{15} (w_{i,c} \times 2^c)} \tag{4.2}$$

The same process is performed at the receiver side to calculate the receiver side's checksum (C_R), except that instead of using zero in the checksum field ($W_{cs} = 0000$), the value of the received checksum ($\widehat{W}_{cs} = \widehat{C_T}$) is used during the calculation of C_R, as shown in equation 4.3:

$$C_R = \overline{\sum_{\substack{i=0 \\ i \neq cs}}^{m-1} \widehat{W}_i + \widehat{C_T}} = \overline{\sum_{\substack{i=0 \\ i \neq cs}}^{m-1} \widehat{W}_i + C_T} = \overline{\sum_{\substack{i=0 \\ i \neq cs}}^{m-1} \widehat{W}_i + \overline{\sum_{\substack{i=0 \\ i \neq cs}}^{m-1} W_i}} = \overline{\sum_{\substack{i=0 \\ i \neq cs}}^{m-1} (\widehat{W}_i + \overline{W}_i)} \tag{4.3}$$

where the received versions of W and C_T are denoted as $\widehat{W}$ and $\widehat{C_T}$, respectively, and assuming that the 16-bit checksum word is intact ($\widehat{C_T} = C_T$). The receiver verifies the packet by re-calculating the checksum. It is obvious from equation 4.3 that if there is no error, which means $\widehat{W}_i = W_i$, the C_R value will be zero:

$$C_R = \overline{\sum_{\substack{i=0 \\ i \neq cs}}^{m-1} (\widehat{W}_i + \overline{W}_i)} = \overline{\sum_{\substack{i=0 \\ i \neq cs}}^{m-1} (W_i + \overline{W}_i)} = \overline{\text{FFFF}} = 0000 \tag{4.4}$$

This is because the value of C_T, which is the inverse of the one's complement sum of all transmitted words, is included in the computation of C_R. Therefore, upon reception, when it is added to the one's complement sum of all words, the identity element FFFF is obtained. And after performing the final ones' complement operation, the final checksum value of a correctly received packet is turned to zero.

C_R from equation 4.3 can be expanded to:

$$C_R = \overline{\sum_{\substack{i=0 \\ i \neq cs}}^{m-1} (\widehat{W}_i + \overline{W}_i)} = \overline{\sum_{\substack{i=0 \\ i \neq cs}}^{m-1} \sum_{c=0}^{15} (\widehat{w}_{i,c} + \overline{w}_{i,c}) \times 2^c} \tag{4.5}$$

Note that here we assumed that the $\widehat{C_T} = C_T$ which means the 16-bit of the checksum are received correctly. We will justify this assumption later in our proposed approach. An important property of the above one's complement sum with an end-around carry expression is as follows (where *a mod b* means *a* modulo *b*):

$$(\widehat{w}_{i,c} + \overline{w}_{i,c}) \times 2^c = \begin{cases} 2^c, & \text{if no error in bit } c \text{ of word } i \\ \overline{w}_{i,c} \times 2^{(c+1) mod (16)}, & \text{if error in bit } c \text{ of word } i \end{cases} \tag{4.6}$$

This has been described in detail in Table 4.1. From the table, it follows that when there is no error in bit c of word i ($w_{i,c} = \widehat{w}_{i,c}$), then $\widehat{w}_{i,c} + \overline{w}_{i,c} = 1$; in the case of an error though, $\widehat{w}_{i,c} + \overline{w}_{i,c} = 0$, and a carry will be generated only if $w_{i,c} = 0$.

Table 4.1 Values of $\widehat{w}_{i,c} + \overline{w}_{i,c}$ for various error scenarios. 0→1 represents a 0 flipped to 1 (and 1→0 the opposite).

$w_{i,c}$	$\widehat{w}_{i,c}$	Condition	$\widehat{w}_{i,c} + \overline{w}_{i,c}$	Carry
0	0	no error	1	0
0	1	error (0→1)	0	1
1	0	error (1→0)	0	0
1	1	no error	1	0

Figure 4.2 contains an example of the checksum calculation at the transmission side and its validation procedure at the receiver side. In this example, the entire packet content is considered as having 32-bit length. The checksum calculation steps are performed to establish the C_T. First, the data is divided into two 16-bit words, and then the ones' complement sum is performed over all the 16-bit words. Lastly, all the bits of the one's complement sum are flipped. As can be seen, at the reception side, the value of C_T is used during the calculation of the C_R. The zero value of C_R validates the received packet.

TRANSMISSION

Bit Position	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Packet 1	0	0	1	0	0	1	1	0	1	1	0	1	0	0	0	1	1	1	0	0	1	1	0	0	1	0	0	1	1	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Second 16-bit word	0	0	1	0	0	1	1	0	1	1	0	1	0	0	0	1
First 16-bit word	1	1	0	0	1	1	0	0	1	0	0	1	1	0	0	0
One's compl. Sum	1	1	1	1	0	0	1	1	0	1	1	0	1	0	0	1
Checksum (C_T)	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**

RECEPTION

Bit Position	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Packet 1	0	0	1	0	0	1	1	0	1	1	0	1	0	0	0	1	1	1	0	0	1	1	0	0	1	0	0	1	1	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Second 16-bit word	0	0	1	0	0	1	1	0	1	1	0	1	0	0	0	1
First 16-bit word	1	1	0	0	1	1	0	0	1	0	0	1	1	0	0	0
One's compl. sum	1	1	1	1	0	0	1	1	0	1	1	0	1	0	0	1
C_T	0	0	0	0	1	1	0	0	1	0	0	1	0	1	1	0
One's compl. sum	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Checksum (C_R)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Figure 4.2 UDP checksum calculation example on a packet data at the transmission side and its validation process at the reception side (transmission of the checksum is not shown).

4.2 Exploiting checksum for error correction: relationship between C_R and error location

The goal of this section is to study the C_R values in different error situations in order to demonstrate how a bit error can affect the C_R value and change its zero pattern. This will help determine the potential error locations based on observed C_R values. In the following, we go through different bit error events (BEEs) and calculate their corresponding C_R values. The values are then grouped into different checksum pattern types (CPTs) based on their similarity patterns. Different BEEs will create different bit patterns of C_R. The better we classify the C_R value the easier it becomes to determine the error locations. Now, we will study all the different BEEs by considering one, two and three bits in error. This is reasonable since, in practice, the residual error after channel decoding should be low. In each BEE, two different bit modification cases are possible:

- $1_j \rightarrow 0_j$, which means a bit 1 was flipped to 0 in column j of a word.
- $0_j \rightarrow 1_j$, which means a bit 0 was flipped to 1 in column j of a word.

The C_R value is calculated for each case of bit modification separately. We start with one bit in error.

4.2.1 One-bit error

The following table shows all the bit error patterns for the case of one bit in error.

Table 4.2 BEEs definitions for one bits in error.

BEEs	Definition
BEE=1	1 bit in error (e.g. $1_j \rightarrow 0_j$ or $0_j \rightarrow 1_j$)

4.2.1.1 BEE=1

In this type of event, there is only one erroneous bit in the packet. If $1_j \rightarrow 0_j$, where $w_{i,j} = 1$ and $\widehat{w}_{i,j} = 0$, as shown in Table 4.1, $\widehat{w}_{i,j} + \overline{w}_{i,j} = 0$ for column j, and for the other columns $c \neq j$, $\widehat{w}_{i,c} + \overline{w}_{i,c} = 1$. Then $\overline{C_R}$ will have a bit 0 in column j and 1 for the others. By considering the final one's complement operation in equation 4.5, which flips all the bits, C_R will have a bit 1 in column j and 0 for the others. This is illustrated in the top part of Figure 4.4.

Figure 4.3 contains an example of this case on the same packet of Figure 4.2. A single bit 1 was flipped to 0 in column 26 of the packet, which corresponds to column 10 of the second word (26 modulo 16 equals 10). As can be seen, C_R has a bit 1 in column 10 and 0 in the others.

TRANSMISSION

Bit Position	31	30	29	28	27	**26**	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Packet 1	0	0	1	0	0	**1**	1	0	1	1	0	1	0	0	0	1	1	1	0	0	1	1	0	0	1	0	0	1	1	0	0	0

RECEPTION

Bit Position	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Packet 1	0	0	1	0	0	**0**	1	0	1	1	0	1	0	0	0	1	1	1	0	0	1	1	0	0	1	0	0	1	1	0	0	0

	15	14	13	12	11	**10**	9	8	7	6	5	4	3	2	1	0
Second 16-bit word	0	0	1	0	0	**0**	1	0	1	1	0	1	0	0	0	1
First 16-bit word	1	1	0	0	1	1	0	0	1	0	0	1	1	0	0	0
Ones's compl. sum	1	1	1	0	1	**1**	1	1	0	1	1	0	1	0	0	1
C_T	0	0	0	0	1	1	0	0	1	0	0	1	0	1	1	0
One's compl. sum	1	1	1	1	1	**0**	1	1	1	1	1	1	1	1	1	1
Checksum (C_R)	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Figure 4.3 UDP checksum validation procedure example at the reception side on a received packet data having one bit error at bit position 26.

For the case of $0_j \rightarrow 1_j$ ($w_{i,j}=0$ and $\widehat{w}_{i,j}=1$), $\widehat{w}_{i,j}+\overline{w}_{i,j}=0$ for column j with an extra *carry* and $\widehat{w}_{i,c}+\overline{w}_{i,c}=1$ for the other columns ($c \neq j$). The extra carry generated in column j will affect the value of column $(j+1)mod(16)$ and change its value from 1 to 0 and also generate a carry which should be added to the next column $(j+2)mod(16)$. This carry propagation will continue and change all the bits 1 to 0, all the way, up to a column with a zero value. Since column j has a zero value, the carry propagation will finally stop there, and change its value from 0 to 1. That means there will be a 1 in column j of $\overline{C_R}$, while the other columns will have a 0. Therefore, C_R will have a 0 in column j and a 1 for all the other columns. This is illustrated in the bottom part of Figure 4.4.

All the C_R values for these two cases are summarized in Figure 4.4. As can be seen, C_R values for these two cases are the inverse of each other. Depending on the error column, which can be one of the 16 columns in a word, C_R can have different patterns. All the 16 different values of C_R in the case of flipping $1_j \rightarrow 0_j$ have the similar pattern of fifteen bits 0 and a single bit 1. The column location of the bit 1 in the pattern, signals the potential error locations in the words. On the other hand, all the 16 different values of C_R in the case of flipping $0_j \rightarrow 1_j$ have the similar pattern of fifteen bits 1 and a single bit 0. All the 16 patterns of C_R for a $1_j \rightarrow 0_j$ flip, as well as the 16-bit patterns of C_R for a $0_j \rightarrow 1_j$ flip are grouped as CPT=1. CPT=1 is defined as the set of all C_R patterns that have fifteen bits 0 and a single bit 1, or vice versa.

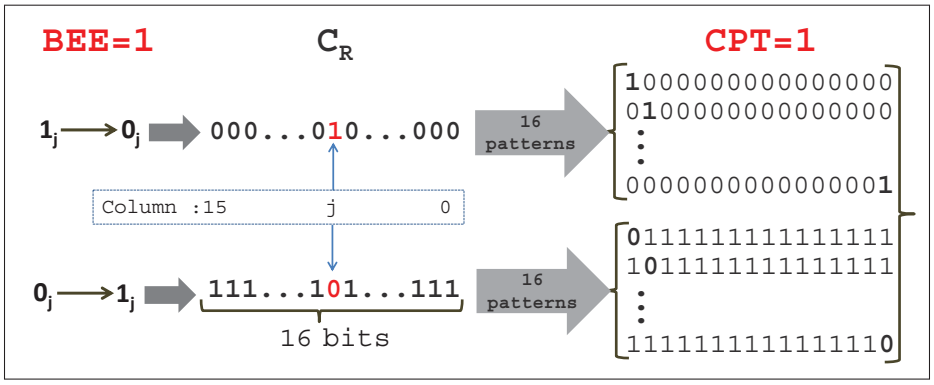


Figure 4.4 BEE=1 and its corresponding 32 patterns of C_R forming CPT=1. Bold bits in CPT=1 indicate the error column.

Let us revisit the example of Figure 4.3. The non-zero value of C_R demonstrates that the received packet is corrupted. In addition, the location of bit 1 in the C_R pattern, column 10, signals that the potential error locations are the 10th bit of each of the words in the packet. So, in this example, the 10th bit and the 26th bit of the packet (10th bit of the first word and 10th bit of the second word respectively) are the two potential error locations. Moreover, the observed pattern of C_R, fifteen bits 0 and a single bit 1, indicates that a bit 1 was flipped to 0 (see in the top part of Figure 4.4). Then, all the potential error locations having a bit value of 0 will constitute the final set of potential error locations. In this case, only the 26th bit of the packet has a value of 0, and is the final error location (in large packets, the list of candidates usually contains more elements).

Therefore, in BEE=1, the potential error location, j, in 16-bit words is indicated by the column of the bit, which is different from the others in the C_R value. In other word:

- If C_R has only one bit 1 and zeros for the others, then BEE=1, which indicate that the potential error location, j, is the column of the bit 1. Moreover, the C_R indicates that a bit 1 changed to 0.
- If C_R has only one bit zero and ones for the others, then BEE=1, which indicate that the potential error location, j, is the column of the bit 0. Moreover, the C_R indicates that a bit 0 changed to 1.

4.2.2 Two-bit errors

In the case of two-bit error, four different BEEs are possible. Two erroneous bits can be in the same column or in different columns; as well, the two flipped bits can have the same values (both 0 or both 1) or have different values (one 0 and the other, 1). Table 4.3 lists the definition of each BEE for this case.

All these BEEs and their corresponding generated C_Rs and CPTs are calculated and defined in the following.

Table 4.3 BEEs definitions for two bits in error. j and k are two different column indexes in modulo 16 for which $0 \leqslant j \neq k \leqslant 15$. Note that, for instance, 1_j1_k means $1_j \rightarrow 0_j$ and $1_k \rightarrow 0_k$.

BEEs	**Definition**
BEE=2	same bit value, different columns, (e.g. 1_j1_k , 0_j0_k)
BEE=3	same bit value, same column, (e.g. 1_j1_j , 0_j0_j)
BEE=4	different bit values, different columns, (e.g. 1_j0_k , 0_j1_k)
BEE=5	different bit values, same column, (e.g. 1_j0_j , 0_j1_j)

4.2.2.1 BEE=2

In this type, two same bits in different columns are flipped. If $1_j \rightarrow 0_j$ and $1_k \rightarrow 0_k$, with $j \neq k$, $\widehat{w}_{i,c} + \overline{w}_{i,c} = 0$ for $c \in \{j,k\}$, and for the other columns, $\widehat{w}_{i,c} + \overline{w}_{i,c} = 1$. So, the corresponding C_R will have bits 1 in column j and k and bits 0 in other columns. For the case of $0_j \rightarrow 1_j$ and $0_k \rightarrow 1_k$, with $j \neq k$, $k < j$, $\widehat{w}_{i,c} + \overline{w}_{i,c} = 0$ for $c \in \{j,k\}$ plus two extra carries in columns k and j. As explained for BEE=1, an extra carry in column k will propagate and generate zeros all the way (from column $(k+1)mod(16)$ up to column $(j-1)mod(16)$), and will stop at column j by changing its value from 0 to 1. The extra carry in column j also propagates, and will stop in column k and change its value to 1. Finally, for $\overline{C_R}$, there should be two 1s in columns j and k and zeros for the others. In this case, C_R will have bits 0 in column j and k, and 1 in the other columns. Depending on which two columns of the words are hit by errors (2 out of 16 columns), the positions of the two bits 1 in the C_R pattern will change. We grouped all the C_R patterns with fourteen bits 0 and two bits 1, which are 120 different patters, (plus their inverse) as CPT=2, as shown in Figure 4.5. Furthermore, the CPT=2 is divided into two sub-groups, CPT=2.1 and CPT=2.2, as we will see later, CPT=2.1 will be observed in other BEEs.

The error column in CPT=2 is indicated by the column of the two bits, which are different from the others in the C_R value. Therefore in BEE=2, the potential error location, j, and k can be identified from C_R pattern as follows:

- If C_R has only two bits 1 and zeros for the others, the potential error location j and k are the column of the two bits 1. Moreover in both cases a bit one was changed to 0.

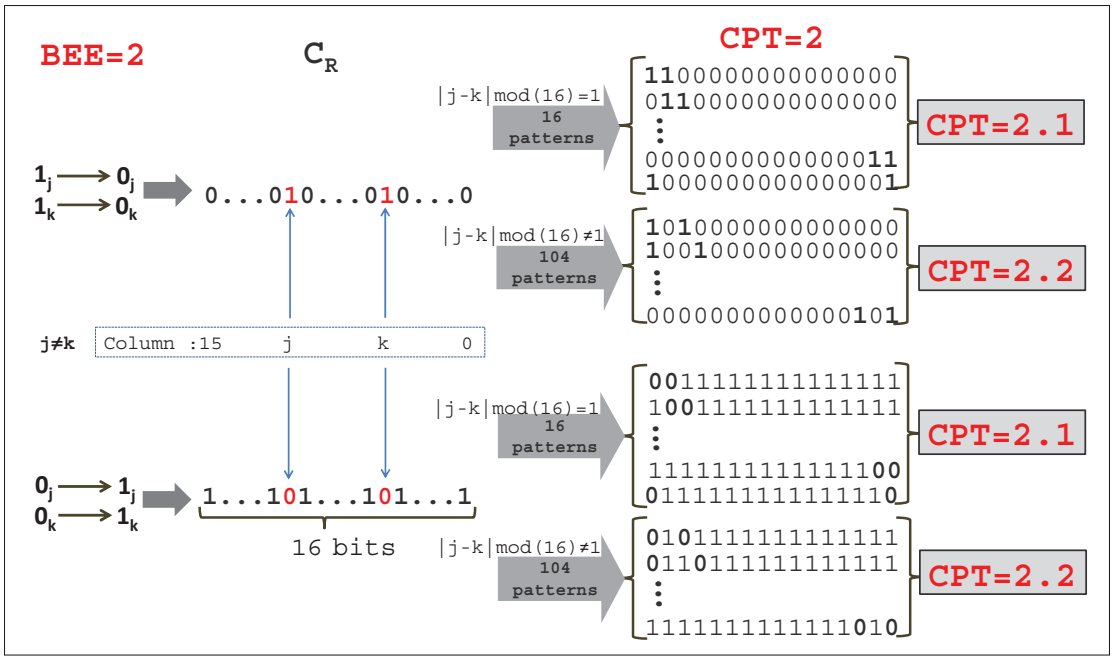


Figure 4.5 BEE=2 and its corresponding 240 patterns of C_R forming CPT=2. The CPT=2 is divided further into two sub-groups as CPT=2.1 and CPT=2.2. All the patterns with two successive bit 1 (or 0) are grouded as CPT=2.1 and the rest are in CPT=2.2. Bold bits in CPT=2 indicate the error columns.

- If C_R has only two bits 0 and ones for the others, the potential error location j and k are the column of the two bits 0. Moreover in both cases a bit zero was changed to 1.

4.2.2.2 BEE=3

In this type, two same bits in the same column are flipped. As shown in Figure 4.6, this BEE generates the same pattern as BEE=1, which is CPT=1. As mentioned earlier, when there is no error $\widehat{w}_{i,c}+\overline{w}_{i,c}=1$ for all 16 columns. When two $0_j \rightarrow 1_j$, then two extra 1s are obtained in column j, and this generates an additional carry. Then, column $(j+1)mod(16)$ will receive the extra carry, and its value will change to 0 with an additional carry. In fact, such a carry will propagate and change all 1s, al1 the way up to a column with a 0 value. Since the value of column $(j+1)mod(16)$ is now 0, the carry propagation will stop there and change its value to 1. Therefore, for $\overline{C_R}$, all columns should be 0, except for column $(j+1)mod(16)$. In this case, C_R will have a 0 in column $(j+1)mod(16)$ and 1 in the other columns. In the other case,

when two $1_j \rightarrow 0_j$, we are missing a carry which should have been generated by column j, and therefore, column $(j+1) mod(16)$ will contain a 0 instead of a 1. The C_R value will have a bit 1 in column $(j+1) mod(16)$ and 0 in the other columns. Like the other BEEs, the calculated C_R of the two cases are the inverse of each other (see Figure 4.6).

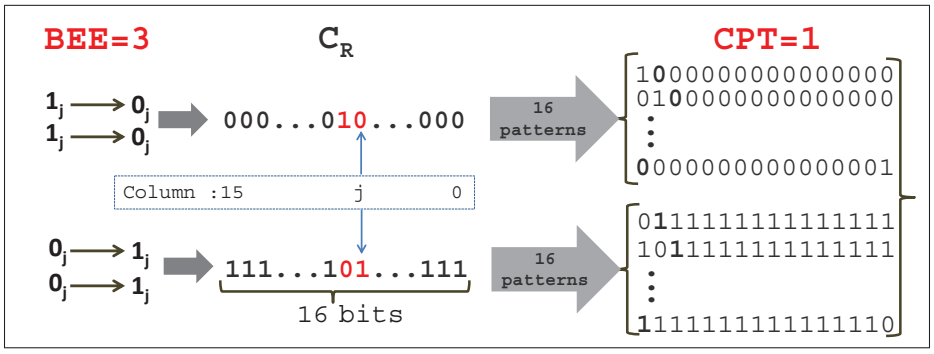


Figure 4.6 BEE=3 and its corresponding 32 patterns of C_R forming CPT=1. Bold bits in CPT=1 indicate the error column.

It is interesting to note that although this type leads to the same CPT as BEE=1, the location and type of errors in each case are quite different. For instance, if the observed C_R pattern has a bit 0 in column j and 1 for the others, the potential error column for the case of BEE=3 is the column $(j-1) mod(16)$, while in the case of BEE=1 the error column is j.

Therefore in BEE=3, the potential error location, j, can be identified from C_R pattern as follows:

- If C_R has one bit 1 in column i and zeros for the others, the potential error location, j, is the column of $(i-1) mod(16)$. Moreover in both error cases a bit one was changed to 0.
- If C_R has one bit 0 in column i and ones for the others, the potential error location, j, is the column of $(i-1) mod(16)$. Moreover in both error cases a bit zero was changed to 1.

4.2.2.3 BEE=4

In this type, two different bits in different columns are flipped. If $1_j \rightarrow 0_j$ and $0_k \rightarrow 1_k$, with $j \neq k$, then $\widehat{w}_{i,j} + \overline{w}_{i,j} = 0$ and $\widehat{w}_{i,k} + \overline{w}_{i,k} = 0$, with a carry in column k, while for the other columns ($c \notin \{j,k\}$), $\widehat{w}_{i,c} + \overline{w}_{i,c} = 1$. The generated carry in column k will propagate and change all 1s to 0s, all the way up to the next 0 value, which is in column j, where it will stop by changing column j's value into 1. So, for $\overline{C_R}$, all the bits between columns k and j (moving circularly from right to left from k to j), excluding j, become 0, while the others remain 1. In this case, the C_R will have $|j-k| mod(16)$ bits 1 between column k and j (including k, but excluding j) and bits 0 in the others.

Depending on which two columns are hit, C_R can have different patterns. If the two columns j and k are next to each other in modulo 16, i.e., $|j-k| mod(16) = 1$, C_R has a single 1 and fifteen 0s, which has been defined as CPT=1. But when $|j-k| mod(16) = 2$, the generated pattern of C_R, which has two bits 1 beside each other and fourteen zero (or vice versa), is the same as CPT=2.1. In the other cases, when $3 \leqslant |j-k| mod(16) \leqslant 13$, where there are between three and thirteen bits 1 between column k and j, the C_Rs are grouped as CPT=3 (see Figure 4.7).

Therefore in BEE=4, the potential error location k (indicating column of $0_k \rightarrow 1_k$) and j (indicating column of $1_j \rightarrow 0_j$) can be identified from C_R pattern as follows:

- The location k is the column of first continuous bit ones from the right to the left.
- The location j is the column of first 0 bit after all the consecutive bit ones from right to left.

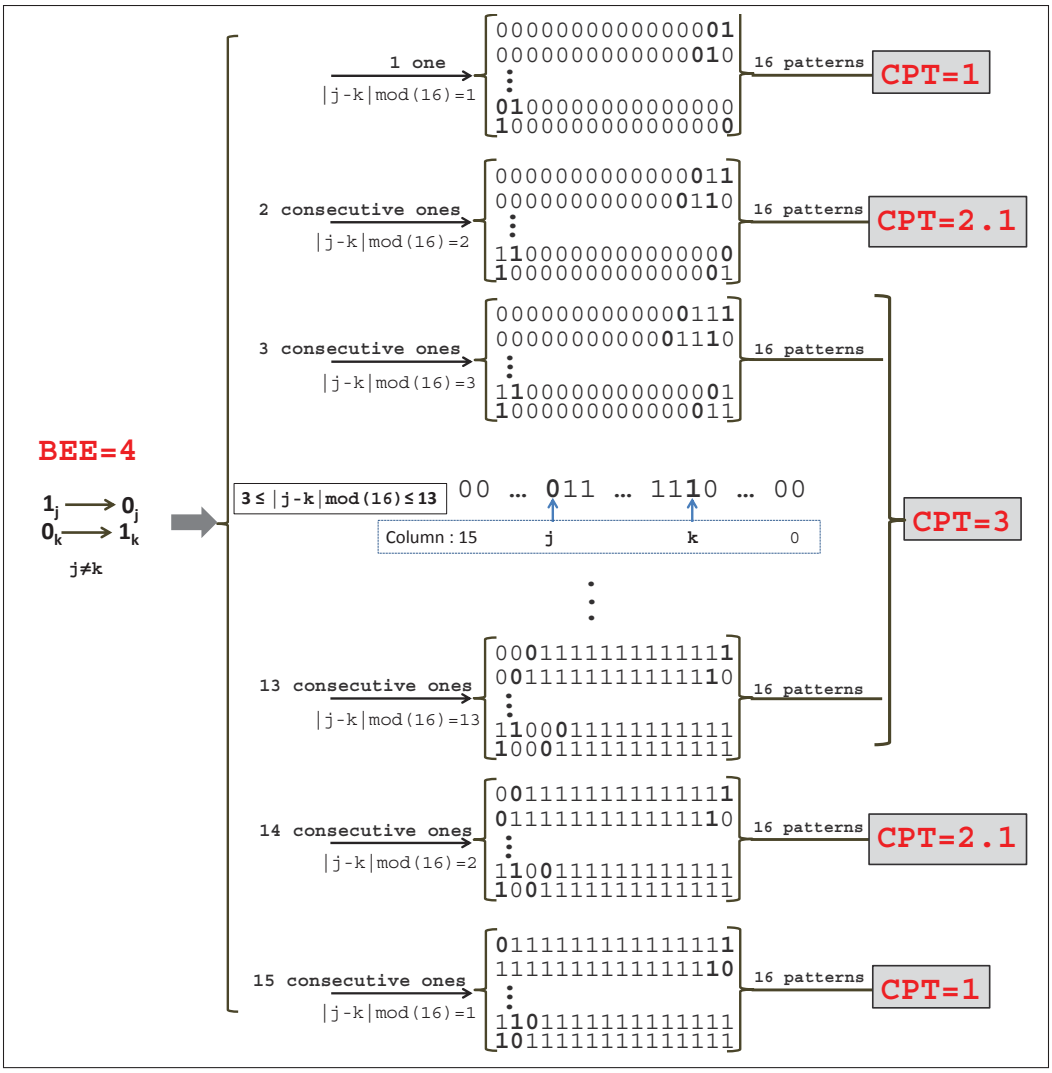


Figure 4.7 BEE=4 and its corresponding 240 patterns of C_R forming CPT=1, CPT=2.1 and CPT=3. The column of bold bits 0 and 1 indicates the columns of bit $1\rightarrow0$ and $0\rightarrow1$, respectively.

4.2.2.4 BEE=5

In this type, two different bits in the same column are flipped. When $0_j\rightarrow1_j$ and $1_j\rightarrow0_j$, the first modification will add an extra 1 in column j, while the second one will remove a 1 in the same column. They will therefore cancel each other's effect and column j's value will not change. Therefore, $\widehat{w}_{i,j}+\overline{w}_{i,j}=1$ for all columns, and consequently, C_R will be zero, which is grouped as CPT=4 in Figure 4.8. In this case, the observed pattern of C_R is exactly the same as

the intact one. If information from the other layers shows that the received packet is corrupted, observing such a pattern indicates that BEE=5 has occurred. Only general information about the possible locations of the errors is available. We know that the two erroneous bits are in the same column, and that they are different bits.

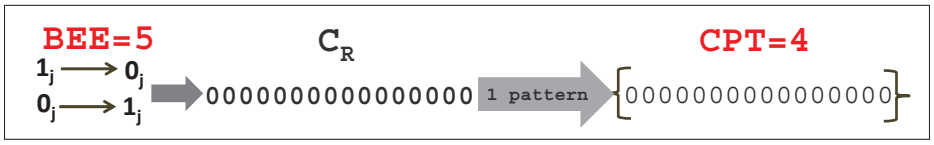


Figure 4.8 BEE=5 and its only C_R pattern forming CPT=4.

Table 4.4 summarizes the definition of all the observed CPT in one and two bits in error. The computation of C_R for a received corrupted packet leads to one of the CPTs defined in Table 4.4. Based on the CPT value, it is possible to determine the corresponding BEEs, as shown in Figure 4.9. For each BEE, the C_R pattern will indicate the error columns and the type of modified bits (1→0 or 0→1).

Table 4.4 Summary of CPT definitions.

CPTs	**Definition**
CPT=1	one bit 1 and fifteen bits 0 or vice versa
CPT=2	two bits 1 and fourteen bits 0 or vice versa
CPT=2.1	two (**successive**) bits 1 and fourteen bits 0 or vice versa
CPT=2.2	two (**non-successive**) bits 1 and fourteen bits 0 or vice versa
CPT=3	three to thirteen consecutive bits 1 between zeros
CPT=4	sixteen bits 0

4.2.3 Three-bit error

The same process can be followed to describe the behavior of three bits in error. The errors may occur in the same or different columns, and as well as, the type of the bits in error can have different patterns. Having all these possibilities in mind, in the case of three bits in error, seven different BEEs can be defined as presented in Table 4.5.

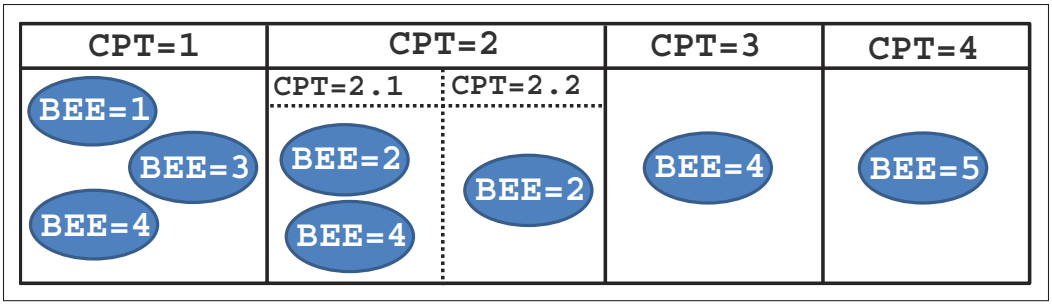


Figure 4.9 Summary of observed CPTs and their corresponding BEEs for one and two bits in error.

Table 4.5 BEEs definitions for three bits in error. j, k and l denote the three different column indexes in modulo 16 for which $0 \leqslant j \neq k \neq l \leqslant 15$. Note that the subdivisions have not been considered.

BEEs	**Definition**	**CPT**
	–Three different columns	
BEE=6	same bit value, (e.g. $1_j1_k1_l$, $0_j0_k0_l$)	3, 5
BEE=7	two same bit value, (e.g. $1_j1_k0_l$, $1_j0_k1_l$, $0_j1_k1_l$, $0_j0_k1_l$, $0_j1_k0_l$, $1_j0_k0_l$)	1, 2, 3, 5
	–Two bits in the same columns	
BEE=8	with same bit values, (e.g. $1_j1_j1_k$, $0_j0_j0_k$)	1, 2
BEE=9	with different bit value than the other column, (e.g. $1_j1_j0_k$, $0_j0_j1_k$)	1, 2, 3, 4
BEE=10	with different bits in the same columns, (e.g. $0_j1_j1_k$, $0_j1_j0_k$)	1
	–Three in the same columns	
BEE=11	with same bit values (e.g. $1_j1_j1_j$, $0_j0_j0_j$)	2
BEE=12	with two same bit value (e.g. $1_j1_j0_j$, $0_j0_j1_j$)	1

Some of the defined BEEs for three bits in error will map to the defined CPTs, and additional CPTs will be also observed. Here for simplicity, we categorize all new generated class patterns as CPT=5. For example in the case of BEE=6, if $1_j \rightarrow 0_j$, $1_k \rightarrow 0_k$ and $1_l \rightarrow 0_l$, with $j \neq k \neq l$, then $\widehat{w}_{i,c} + \overline{w}_{i,c} = 0$ for $c \in \{j,k,l\}$, and for the other columns, $\widehat{w}_{i,c} + \overline{w}_{i,c} = 1$. So, the corresponding C_R will have bits 1 in columns j, k and l and bits 0 in other columns. BEE=6 can generate 560 (a 3-column combination among 16 columns: $\binom{16}{3} = \frac{16!}{3!13!}$) different patterns. Note that when j, k and l are three successive columns, which means $|j-k| mod(16) = 1$ and $|l-k| mod(16) = 1$, then there will be three successive ones between zeros. These sixteen patterns have been already defined as part of CPT=3 with $|j-k| mod(16) = 3$ in Figure 4.7. Therefore BEE=6 generates 16 patterns of CPT=3 (a subdivision of it) and 540 new patterns that we assign to

CPT=5. The generated pattern for all the other BEE cases are shown in the last column of Table 4.5. A more detailed analysis and subdivision should be done to classify the patterns. For instance, BEE=11 should be mapped to CPT=2.1, but in Table 4.5 we have ignored the subdivisions on patterns. We ignore the presentation and definition for the new generated patterns for the case of more than two bits in error because, as will be seen later, the probability of having more than two bits in error is dramatically less than that of having a single-bit error in the applications of interest.

Figure 4.10 summarizes all the observed CPTs and their corresponding BEEs for the case up to three bits in error. When the computed C_R for a received corrupted packet leads to one of the defined CPTs, based on the CPT value, it is possible to determine the corresponding BEEs, as shown in Figure 4.10. For each BEE, the C_R pattern will indicate the error columns and the type of modified bits ($1 \rightarrow 0$ or $0 \rightarrow 1$).

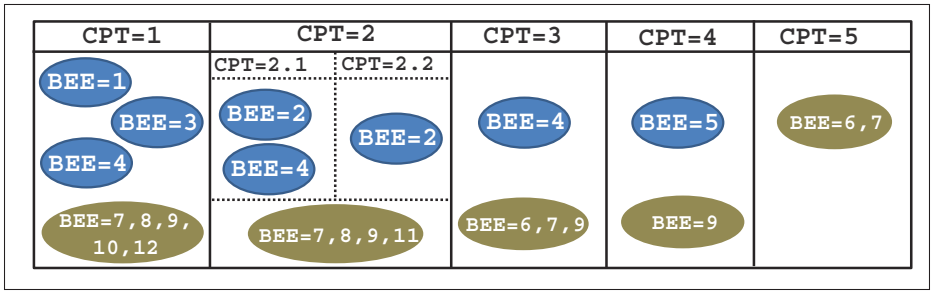


Figure 4.10 Summary of observed CPTs and their corresponding BEEs for the case of one, two and three bits in error.

For instance, if the calculated C_R is "0000 0000 0010 0000", which has one bit 1 in column 5, it belongs to CPT=1, as defined in Table 4.4. This pattern can be generated by BEE=1 (considering one bit error), BEE=3 or BEE=4 (considering two bits error), BEE=7, 8, 9, 10 or 12 (considering three bits error) as shown in Figure 4.10. Based on each BEE, the C_R pattern will have different meanings. In the case of BEE=1, the C_R pattern indicates that there is a single-bit error in column 5 of a word, and it is $1_5 \rightarrow 0_5$. Then, all the bits 0 in column 5 are the potential error locations in this case. In the case of BEE=3, the pattern indicates that there are

two bits in error, and both are $1_4 \rightarrow 0_4$, as presented in Figure 4.6. In this case, the number of candidates is a 2-combination of the number of zeros in column 4. In the case of the BEE=4, the pattern indicates that there are two bits in error and $1_6 \rightarrow 0_6$; $0_5 \rightarrow 1_5$, as presented in Figure 4.7. The full potential candidate list is presented in Table 4.6.

Table 4.6 All the possible BEEs in the case of one to three bits in error when C_R equals to a pattern in CPT=1 as "0000 0000 0010 0000" (a non-zero bit in column 5). Note that nz_c and no_c represent the number of bits 0 and 1 in column c, respectively.

BEEs	**Error Locations in columns**	**number of candidates**
one bit in error		
BEE=1	$1_5 \rightarrow 0_5$	$\binom{\text{nz}_5}{1}$
two bits in error		
BEE=3	$1_4 \rightarrow 0_4$ $1_4 \rightarrow 0_4$	$\binom{\text{nz}_4}{2}$
BEE=4	$1_6 \rightarrow 0_6$ $0_5 \rightarrow 1_5$	$\binom{\text{nz}_6}{1}\binom{\text{no}_5}{1}$
three bits in error		
BEE=7	$1_7 \rightarrow 0_7$ $0_6 \rightarrow 1_6$ $0_5 \rightarrow 1_5$	$\binom{\text{nz}_7}{1}\binom{\text{no}_6}{1}\binom{\text{nz}_5}{1}$
BEE=8	$1_4 \rightarrow 0_4$ $1_3 \rightarrow 0_3$ $1_3 \rightarrow 0_3$	$\binom{\text{nz}_4}{1}\binom{\text{nz}_3}{2}$
BEE=9	$1_6 \rightarrow 0_6$ $0_4 \rightarrow 1_4$ $0_4 \rightarrow 1_4$	$\binom{\text{nz}_6}{1}\binom{\text{no}_4}{2}$
BEE=10	$1_5 \rightarrow 0_5$ $0_j \rightarrow 1_j$; $0 \le j \le 15$; $j \ne 5$ $1_j \rightarrow 0_j$;	$\binom{\text{nz}_5}{1}\sum_{\substack{j=0 \\ j\neq 5}}^{15}\binom{\text{no}_j}{1}\binom{\text{nz}_j}{1}$
BEE=12	$1_5 \rightarrow 0_5$ $1_5 \rightarrow 0_5$ $0_5 \rightarrow 1_5$	$\binom{\text{nz}_5}{2}\binom{\text{no}_5}{1}$

As shown in the example, it is possible to have more than one BEE for an observed CPT. In the next section, we mathematically demonstrate which one of the candidate BEEs is more likely than the others.

4.3 Probability of BEEs given observed CPTs: $\Pr(\mathbf{BEE}=i|\mathbf{CPT}=j)$

As can be seen from Figure 4.10, several BEEs can cause the same observed CPT. For instance, if the observed C_R value belongs to the patterns in CPT=1, then it could possibly be due to one of the three BEEs (BEE=1, 3 or 4) if considering only one and two bits in error. In this section, we show mathematically the probability of each event to determine which one of these possible BEEs is more likely. The goal here is to find the probability associated to each BEE based on the observed CPT, which is defined as $\Pr(\text{BEE}=i|\text{CPT}=j)$. To compute this, we use the conditional probability and the law of total probability (Pfeiffer, 2013), as shown in equation 4.7.

$$\begin{aligned}\Pr(\text{BEE}=i|\text{CPT}=j) &= \frac{\Pr(\text{BEE}=i,\text{CPT}=j)}{\Pr(\text{CPT}=j)} \\ &= \frac{1}{\Pr(\text{CPT}=j)} \times \sum_{k=0}^{N} \left\{ \Pr(\text{BEE}=i,\text{CPT}=j|\text{nbErr}=k) \times \Pr(\text{nbErr}=k) \right\}\end{aligned} \tag{4.7}$$

The probability of having k bits error in a packet of N bits with a channel residual bit error rate (ρ) can be expressed as:

$$\text{P}_k = \Pr(\text{nbErr}=k) = \rho^k \times (1-\rho)^{N-k} \tag{4.8}$$

Assuming that ρ is very small (e.g. $\rho \leq 10^{-5}$), then the probability of having more than two bits in error (P_k for $k>2$), even for large packet sizes, will be so small that the terms of the summation for $k>2$ can be ignored. That is the reason we ignore considering more than two bits in error in the rest of the calculation. Accordingly, equation 4.7 can be approximated with

equation 4.9:

$$\Pr(\text{BEE}=i|\text{CPT}=j) \approx \frac{1}{\Pr(\text{CPT}=j)} \times \sum_{k=0}^{2} [\Pr(\text{BEE}=i, \text{CPT}=j|\text{nbErr}=k) \times \rho^k \times (1-\rho)^{N-k}] \quad (4.9)$$

By using the chain rule (Pfeiffer, 2013), the first probability in the previous equation can be expressed as:

$$\Pr(\text{BEE}=i, \text{CPT}=j|\text{nbErr}=k) = \Pr(\text{BEE}=i|\text{nbErr}=k) \times \Pr(\text{CPT}=j|\text{BEE}=i \cap \text{nbErr}=k) \quad (4.10)$$

Therefore $\Pr(\text{BEE}=i|\text{CPT}=j)$ can be finally expressed as:

$$\begin{aligned} \Pr(\text{BEE}=i|\text{CPT}=j) \approx & \frac{1}{\Pr(\text{CPT}=j)} \times \\ & \sum_{k=0}^{2} [\Pr(\text{BEE}=i|\text{nbErr}=k) \times \Pr(\text{CPT}=j|\text{BEE}=i \cap \text{nbErr}=k) \times \text{P}_k] \end{aligned} \quad (4.11)$$

The above two probabilities $\Pr(\text{BEE}=i|\text{nbErr}=k)$ and $\Pr(\text{CPT}=j|\text{BEE}=i \cap \text{nbErr}=k)$ will be calculated in the following sections. Note that P_k will be calculated from the equation 4.8.

4.3.1 $\Pr(\textbf{BEE}=i|\textbf{nbErr}=k)$

Assuming a packet with length of N bits, the packet is divided into words of sixteen bits, as shown in Figure 4.11. For simplicity, the packet size is considered a multiple of 16 bits. Let nz_c and no_c represent the number of bits 0 and 1 in column c, respectively. Therefore we define TZ and TO as total number of bit 0 and 1 in the packet as follows:

$$\text{TZ} = \sum_{c=0}^{15} \text{nz}_c; \qquad \text{TO} = \sum_{c=0}^{15} \text{no}_c \quad (4.12)$$

In the following expressions, the probability value of $\Pr(\text{BEE}=i|\text{nbErr}=k)$ is calculated for the case of one and two bits in error ($k=1,2$). If we know that there is only one bit in error,

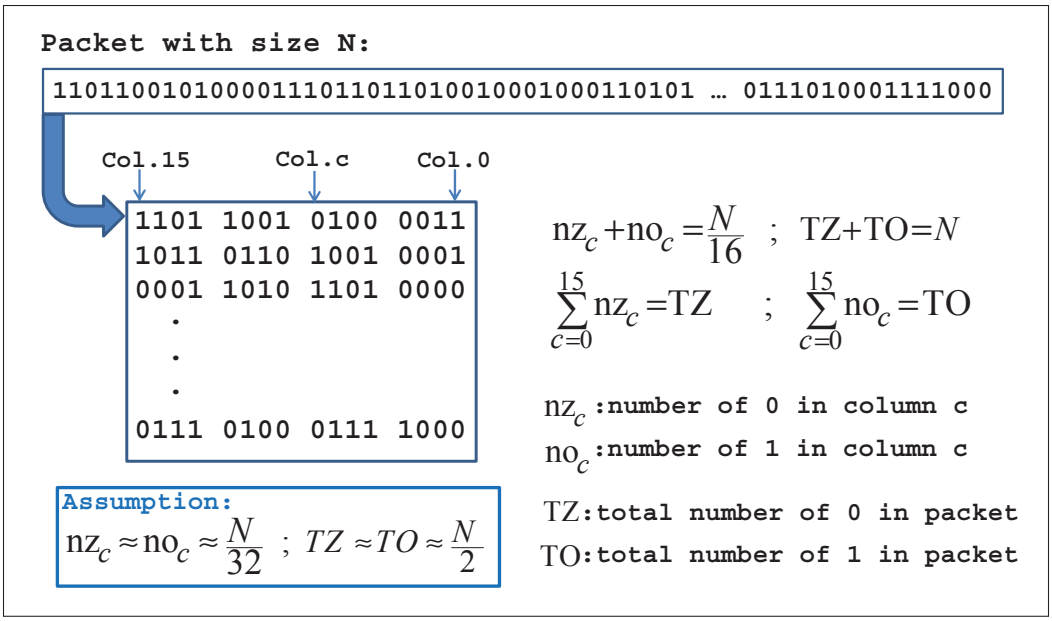


Figure 4.11 Example of packet division into 16 bits.

obviously the BEE=1 was happening. While if two bits were erroneous, the probability of occurrence of BEE=1 is zero. These have been demonstrated in the following probability expressions:

$$\Pr(\text{BEE}=1|\text{nbErr}=1)=1; \qquad \Pr(\text{BEE}=1|\text{nbErr}=2)=0 \tag{4.13}$$

By definition, all the other BEEs from 2 to 5 are for two-bit error, and therefore, these BEEs cannot occur when the number of bits in error is one. However, they have values for a two-bit error. The probability value of each one can be calculated by the definition of each BEE in Table 4.2 and 4.3 and considering the number of bits 0 and 1 in each column. The following equations reflect the number of possible combinations of taking two bits, same or different type, in the same or different columns. Assuming $nz_c \approx no_c$, which means the number of bits 0

and 1 in each column are the same, the expression can, however, be simplified as follows:

$$\Pr(\text{BEE}=i|\text{nbErr}=1)=0; \qquad i\in\{2,3,4,5\}$$

$$\Pr(\text{BEE}=2|\text{nbErr}=2)=\frac{1}{2}\times\frac{\sum_{c=0}^{15}\binom{\text{nz}_c}{1}\binom{\text{TZ}-\text{nz}_c}{1}+\sum_{c=0}^{15}\binom{\text{no}_c}{1}\binom{\text{TO}-\text{no}_c}{1}}{\binom{N}{2}}\approx\frac{15N}{32(N-1)}$$

$$\Pr(\text{BEE}=3|\text{nbErr}=2)=\frac{\sum_{c=0}^{15}\binom{\text{nz}_c}{2}+\sum_{c=0}^{15}\binom{\text{no}_c}{2}}{\binom{N}{2}}\approx\frac{N-32}{32(N-1)} \tag{4.14}$$

$$\Pr(\text{BEE}=4|\text{nbErr}=2)=\frac{1}{2}\times\frac{\sum_{c=0}^{15}\binom{\text{nz}_c}{1}\binom{\text{TO}-\text{no}_c}{1}+\sum_{c=0}^{15}\binom{\text{no}_c}{1}\binom{\text{TZ}-\text{nz}_c}{1}}{\binom{N}{2}}\approx\frac{15N}{32(N-1)}$$

$$\Pr(\text{BEE}=5|\text{nbErr}=2)=\frac{\sum_{c=0}^{15}\binom{\text{nz}_c}{1}\binom{\text{no}_c}{1}}{\binom{N}{2}}\approx\frac{N}{32(N-1)}$$

Note that from the assumption, the value of nz_c and no_c are replaced by $\frac{N}{32}$, and as well as $\text{TZ}\approx\text{TO}\approx\frac{N}{16}$ to simplify the probability value of each case.

4.3.2 $\Pr(\text{CPT}=j|\text{BEE}=i\cap\text{nbErr}=k)$

Here, the second probability of equation 4.11 will be examined. From the definition of BEE=1 and all its generated patterns from the Figure 4.4, only CPT=1 is observed. Therefore, it is clear that when there is a single-bit error, the following value is obtained:

$$\Pr(\text{CPT}=j|\text{BEE}=1\cap\text{nbErr}=1)=\begin{cases}1 & j=1\\ 0 & j\in\{2,3,4\}\end{cases} \tag{4.15}$$

For the case of two bits in error, for instance BEE=2 only CPT=2 is generated which has 240 different patterns. As it can be seen from the Figure 4.5, 32 of the patterns belong to the class

pattern CPT=2.1 and the others belong to the CPT=2.2.

$$\Pr(\text{CPT}=j|\text{BEE}=2\cap\text{nbErr}=2)=\begin{cases}1 & j=2 \quad\Longrightarrow\begin{cases}\frac{32}{240} & j=2.1\\ \frac{208}{240} & j=2.2\end{cases}\\ 0 & j\in\{1,3,4\}\end{cases} \tag{4.16}$$

For the case of two bits in error as BEE=3 and BEE=5, the following probability values are obtained:

$$\begin{aligned}\Pr(\text{CPT}=j|\text{BEE}=3\cap\text{nbErr}=2)&=\begin{cases}1 & j=1\\ 0 & j\in\{2,3,4\}\end{cases}\\ \Pr(\text{CPT}=j|\text{BEE}=5\cap\text{nbErr}=2)&=\begin{cases}1 & j=4\\ 0 & j\in\{1,2,3\}\end{cases}\end{aligned} \tag{4.17}$$

And finally for the case of BEE=4,

$$\Pr(\text{CPT}=j|\text{BEE}=4\cap\text{nbErr}=2)=\begin{cases}\frac{32}{240} & j=1\\ \frac{32}{240} & j=2.1\\ 0 & j=2.2\\ \frac{176}{240} & j=3\\ 0 & j=4\end{cases} \tag{4.18}$$

BEE=4 comprises 240 different patterns, as shown in Figure 4.7, 32 of which belong to CPT=1. Hence, the probability of having CPT=1 given BEE=4 is $32/240$. Similarly, the probability values for the other cases can be computed. Note that in BEE=4, there are 32 patterns of CPT=2.1 and there is not any CPT=2.2.

4.3.3 Estimation of $\Pr(\text{BEE}=i|\text{CPT}=j)$

By substituting the probability values of the two previous sections, Section 4.3.1 and Section 4.3.2, into equation 4.11, we obtain the $\Pr(\text{BEE}=i|\text{CPT}=j)$ for each case as follows:

$$\begin{aligned}\Pr(\text{BEE}=1|\text{CPT}=1) \approx & \frac{1}{\Pr(\text{CPT}=1)} \times \\ & [\Pr(\text{BEE}=1|\text{nbErr}=1) \times \Pr(\text{CPT}=1|\text{BEE}=1 \cap \text{nbErr}=1) \times \text{P}_1 \\ & + \Pr(\text{BEE}=1|\text{nbErr}=2) \times \Pr(\text{CPT}=1|\underbrace{\text{BEE}=1 \cap \text{nbErr}=2}_{\varnothing}) \times \text{P}_2\,] \\ \approx & \frac{1}{\Pr(\text{CPT}=1)} \times [1 \times 1 \times \text{P}_1 + 0 \times 0 \times \text{P}_2] \\ \approx & \frac{\text{P}_1}{\Pr(\text{CPT}=1)}\end{aligned} \tag{4.19}$$

Note that P_1 and P_2 can be computed from equation 4.8.

$$\begin{aligned}\Pr(\text{BEE}=2|\text{CPT}=2.1) \approx & \frac{1}{\Pr(\text{CPT}=2.1)} \times \\ & [\Pr(\text{BEE}=2|\text{nbErr}=1) \times \Pr(\text{CPT}=2.1|\overbrace{\text{BEE}=2 \cap \text{nbErr}=1}^{\varnothing}) \times \text{P}_1 \\ & + \Pr(\text{BEE}=2|\text{nbErr}=2) \times \Pr(\text{CPT}=2.1|\text{BEE}=2 \cap \text{nbErr}=2) \times \text{P}_2\,] \\ \approx & \frac{1}{\Pr(\text{CPT}=2.1)} \times [0 \times 0 \times \text{P}_1 + \frac{15N}{32(N-1)} \times \frac{32}{240} \times \text{P}_2] \\ \approx & \frac{\text{P}_2}{\Pr(\text{CPT}=2.1)} \times \frac{N}{16(N-1)}\end{aligned} \tag{4.20}$$

The same replacement procedure can be done to calculate the probability for each case. All the probability values are summarized in Table 4.7. For simplicity, instead of showing the value of $\Pr(\text{BEE}=i|\text{CPT}=j)$, the value of $\Pr(\text{BEE}=i|\text{CPT}=j) \times \Pr(\text{CPT}=j)$, which equals to

$$\sum_{k=0}^{2} [\Pr(\text{BEE}=i, \text{CPT}=j|\text{nbErr}=k) \times \text{P}_k] \qquad \text{(see equation 4.9)}$$

are shown in the table. As can be seen from the table, when the first row (BEE=1) is multiplied by the probability value of P_1, and the other rows (BEE=2 to 5) by probability value of P_2, the probability value of $\Pr(\text{BEE}=i|\text{CPT}=j)\times\Pr(\text{CPT}=j)$ is obtained. It should be straightforward to normalize the latter probabilities within each CPT$=j$ to obtain $\Pr(\text{BEE}=i|\text{CPT}=j)$, but this is not required since in an error correction scheme, it is the relative probabilities among the various BEEs which are of interest.

Table 4.7 Array of $\Pr(\text{BEE}=i,\text{CPT}=j|\text{nbErr}=k)$ and its approximate value for large packet size. Multiplying each cell by P_1 or P_2 gives $\Pr(\text{BEE}=i|\text{CPT}=j)\times\Pr(\text{CPT}=j)$.

BEE	**CPT =1**	**CPT =2**		**CPT =3**	**CPT =4**	
		CPT =2.1	**CPT =2.2**			
1	1	-	-	-	-	$\times\mathbf{P_1}$
2	0	$\frac{N}{16(N-1)}\approx 0.063$	$\frac{13N}{32(N-1)}\approx 0.406$	0	0	
3	$\frac{N-32}{32(N-1)}\approx 0.031$	0	0	0	0	
4	$\frac{N}{16(N-1)}\approx 0.063$	$\frac{N}{16(N-1)}\approx 0.063$	0	$\frac{11N}{32(N-1)}\approx 0.344$	0	$\times\mathbf{P_2}$
5	0	0	0	0	$\frac{N}{32(N-1)}\approx 0.031$	

When comparing the two probability values P_1 and P_2, with the values in Table 4.7, one can deduce that the probability of having more than two bits in error is dramatically less than that of having a single-bit error for a small ρ. The table also illustrates that when a CPT=1 is observed, BEE=1 is much more likely, and BEE=4 or BEE=3 are possible albeit at a very low probability (about $10/\rho$ times smaller).

To verify the probability values, we conducted a simulation on different sequences with different packet sizes. In each bitstream, one or two bits were randomly flipped, and the simulation was repeated 10,000 times to estimate the empirical probability value of $\Pr(\text{BEE}=i,\text{CPT}=j|\text{nbErr}=k)$. Table 4.8 presents an example of the simulation results for the *Crew* video sequence which is encoded by an H.264 codec at quantization parameter (QP) of 27. As can be observed, the simulation results are similar to the values in Table 4.7. Fig-

Table 4.8 Empirical probability value of $\Pr(\text{BEE}=i, \text{CPT}=j|\text{nbErr}=k)$ for the *Crew* sequence encoded by H.264, packet size=1432 bits. $\rho=10^{-6}$, so $P_1\approx10^{-6}$ and $P_2\approx10^{-12}$.

BEE	CPT =1	CPT =2		CPT =3	CPT =4	
		CPT =2.1	CPT =2.2			
1	1	-	-	-	-	$\times\mathbf{P_1}$
2	0	0.060	0.409	0	0	$\times\mathbf{P_2}$
3	0.030	0	0	0	0	
4	0.063	0.060	0	0.349	0	
5	0	0	0	0	0.029	

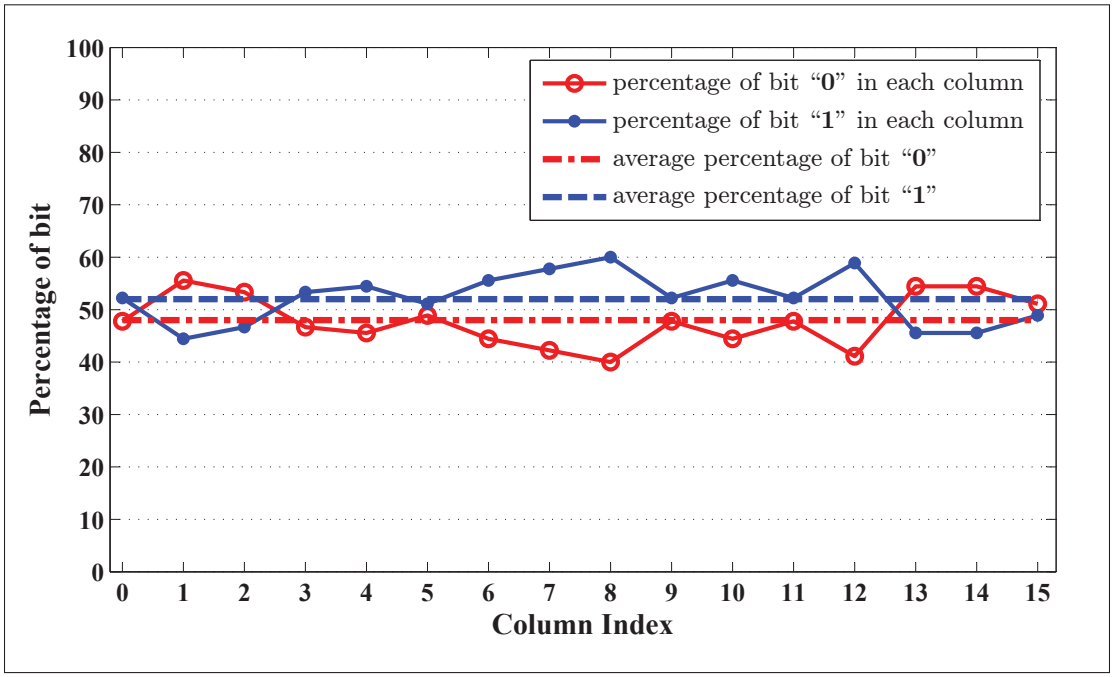


Figure 4.12 Percentage of bits 0 and 1 in each column of slice index 29 and frame index 32 of the *Crew* sequence coded by H.264 at QP=27, packet size=1432 bits.

ure 4.12 shows the distribution of bits 0 and 1 in each column of the simulated packet in Table 4.7. These results demonstrate that the assumption of having an equal number of bits 0 and 1 in each column is a reasonable assumption and, if they are the same on average, then the results

will perfectly match the theoretical results. Similar results have been obtained on other video sequences with different QPs as shown in Table 4.9.

Table 4.9 Average percentage of zero and one in each column. The letters F and S in the first column show the frame and the slice number in each case of simulation.

sequence info	average percentage of "0" and "1"	
	"0"	"1"
*City*_qp37_F35_S17	49	51
*Crew*_qp27_F32_S29	48	52
*Ice*_qp32_F39_S10	55	45
*Foreman*_qp22_F42_S12	52	48
*Opening ceremoney*_qp27_F37_S20	47	53
*Whale show*_qp22_F47_S5	47	53
*Driving*_qp32_F31_S26	51	49
*Walk*_qp27_F40_S19	52	48

4.4 Proposed checksum-filtered list decoding approach for video error correction

In the previous section, we showed that the UDP checksum of corrupted packets exhibits specific bit patterns. Observing these specific patterns helps us to identify the potential error locations. In this section, we present our novel CFLD approach which exploits the receiver side UDP checksum to remove non-valid candidate bitstreams and alleviate the large solution space problem of the conventional list decoding approaches. The checksum pattern allows us to find the potential locations of the erroneous bits in the bitstream, by having the information about the possible error column(s) in the words and the erroneous value (a 0 or a 1). Figure 4.13 shows the general schematic of the proposed method. When a packet is received, if it is intact (depending on the UDP checksum value), it will go directly to the video decoder, otherwise it will go through the error correction process. Since the UDP checksum is calculated over the pseudo header, the header and the payload, it is helpful to identify whether an error indicated by C_R, is from the headers or from video data. Therefore, the first step of the correction process is to fix the headers.

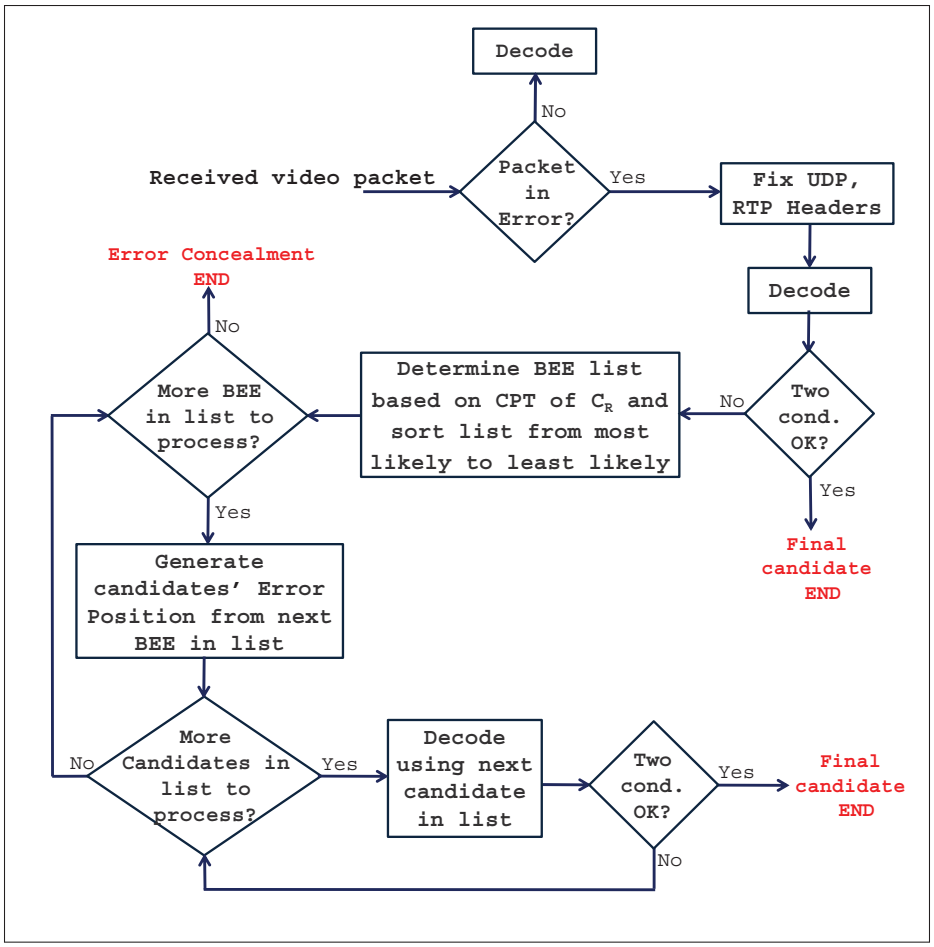


Figure 4.13 Proposed CFLD system.

4.4.1 Header correction process

Figure 4.14 shows the UDP packet encapsulation in H.264 coded sequences. The size of a UDP header is 8 bytes and the real-time transport protocol (RTP) header has 12 bytes in length. Since the C_R is calculated over all these bits, it is desirable, to identify if there is an error which is indicated by C_R, whether it is coming from the headers or the video data. So, the first step of the correction process is to fix the headers. Figure 4.1 and Figure 4.15 show the format of UDP and RTP headers as they are defined in the standard. Some fields of the UDP/RTP headers are static during the transmission (e.g., Source/Destination Port Num in UDP header and almost the first two-byte fields in RTP header), and some other parts are easily predictable (e.g., Sequence number and Timestamp in RTP header) because of the redundant information

in the headers (Postel, 1980; Schulzrinne *et al.*). The next layer, network abstraction layer (NAL), has one byte header which consists of three syntaxes: *forbidden_zero_bit* (1-bit), *nal_ref_idc* (2-bit field) and *nal_unit_type* (5-bit field) and their corresponding values are fixed for the same frames and it is possible to be predicted from the other received intact packets.

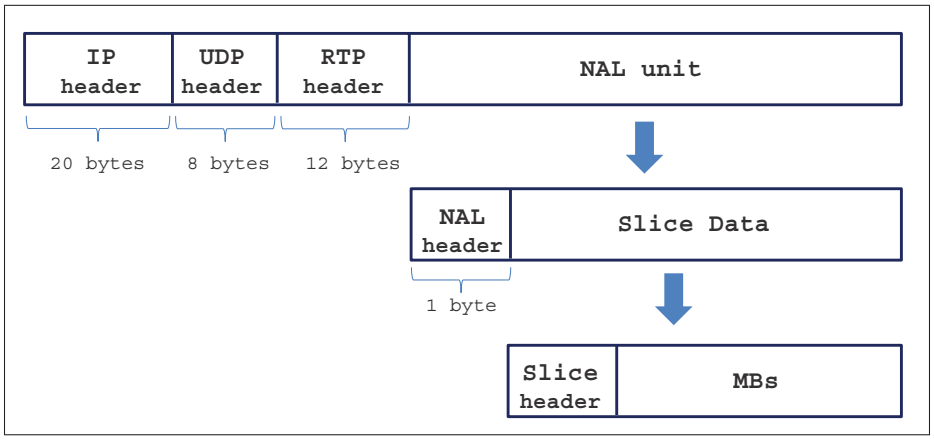


Figure 4.14 UDP encapsulation for H.264 coded sequences.

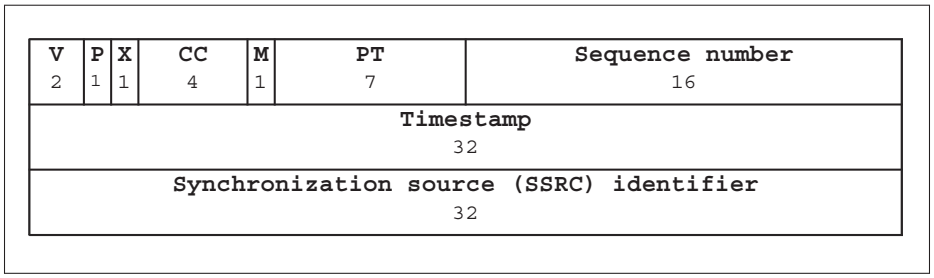


Figure 4.15 RTP header format. The numbers are showing the length of each field.

4.4.2 Video data correction process

The next step after fixing all the headers is to decode the bitstream. Here, we consider two conditions (as defined in section 3.3) which must be satisfied:

1. the sequence should be decodable (no semantic or syntactical errors are detected),

2. and the number of blocks in the corrupted slices should be correct.

This step helps save the sequences which had errors somewhere in the headers, but not in the video payload. Thus, they are not put through the correction process. It is assumed that the number of blocks in the packet is known. That is the case in several systems where the number of macroblocks (MBs) or coding tree units (CTUs) in a packet is constant (for instance, sending a row of MBs in each slice) or can be deduced from the information within other packets (for instance, the *first_mb_in_slice* syntax element in H.264). During the simulations, it was observed that because of the high compression properties of the encoding process, the coded bitstreams were very sensitive to errors and, in many cases, even a single-bit error can desynchronize the whole packet. This desynchronization creates non-valid syntax or semantic errors in the decoding process. This property is used to differentiate between decodable and non-decodable bitstreams. A decodable bitstream has syntactically/semantically valid codewords. Since it has been observed that decodable bitstreams can nevertheless still be fairly damaged, the constraint on the number of MBs, in the case of H.264 sequences, or CTUs, in the case of high efficiency video coding (HEVC) sequences, further eliminates corrupted candidates.

If the sequence does not satisfy the two above-mentioned conditions, that means there are errors somewhere in the video payload. Consequently, the packet should be further processed by the following method:

- Based on the observed CPT value of C_R, all the possible BEEs are determined and ordered from most likely to least likely, according to the results of Table 4.7.

- Starting with the most probable BEE, a candidate list is generated. This list includes the potential error locations, based on the observed CPT, which provides the potential error column(s) and the type of flipped bits at issue ($1\rightarrow0$ or $0\rightarrow1$). For each potential error location, a candidate bitstream is generated.

- Each candidate bitstream passes through the video decoder until one is found that satisfies the two conditions (the sequence is decodable; and the number of MBs, in the case of H.264

sequences, or the number of CTUs, in the case of HEVC, is correct), and from that the final candidate bitstream is determined.

- If none of the candidate bitstreams meets these two conditions, we restart the process of generating a candidate list of potential error locations with the next most probable BEE.

In summary, the method finds the first candidate bitstream that satisfies the two conditions, starting with the most probable BEEs. When there is no probable BEEs, or none of the candidate bitstreams meet two conditions of the decoder, the approach falls back to error concealment. Note that any error concealment approach can be employed. There could be a case where, at the end, more than one candidate bitstream would satisfy the decoder's conditions. The system could thus possibly be modified to have an extra step for ranking the bitstreams that satisfies these conditions by likeliness. For instance, a pixel domain approach, such as boundary matching or border checking, could help in selecting a *final* candidate between those candidates that meet the decoder's conditions.

The difficulties of accessing soft information at the application layer in existing video communication systems make the approaches using only hard information very appealing to build robust video error correction systems. But ignoring the soft information in traditional list decoding approaches makes them highly inefficient (as the following simulations will show). Indeed, since all the bits then have the same probability of being flipped, as a result, all the candidate bitstreams have the same probability of being the final one. The final candidate would then be chosen through an exhaustive (brute force) search on all the candidates without any order preference. We name this method as exhaustive search list decoding (ESLD) approach. In the following simulations, we used ESLD approach as another benchmark for comparison against the proposed CFLD to represent the performance of list decoding methods that would not have access to soft information to order their candidates.

4.5 Candidate reduction

Using the checksum value in the error correction process provides a notable reduction in the number of candidates to be considered in list decoding approaches. The receiver side's checksum value allows the determination of the potential error column in the words and in the type of the flipped bits (a bit 0 changed to 1 or a bit 1 changed to 0). The total number of candidates depends on the packet size (or the number of words in the packet) and on the number of errors. Generally, in list decoding approaches (e.g. ESLD), for a packet of containing N bits, there are N possible candidate bitstreams for the case of a single-bit error, whereas our CFLD approach will reduce it to only an average of $N/32$ candidates. This is because the C_R value provides extra information about the error column in the words and the type of the flipped bit. Since the packet is divided into 16-bit words, there are $N/16$ bits in each column and, assuming that half of the bits in each column are zeros and half of them are ones, the total number of candidates will therefore be approximately $N/32$. This means that in the case of a single-bit error, there is a 97% reduction in the number of candidate bitstreams, and only about 3% should be considered, as compared to other list decoding approaches. This reduction is even higher when the number of bits in error is increased. For instance, in the case of a two-bit error, about 99.6% of non-valid candidates can be eliminated by considering the C_R validation process in the proposed CFLD approach. Note that for a packet of N bits, there are $\frac{N(N-1)}{2}$ candidates in the case of a two-bit error (generally $\frac{N!}{K!(N-K)!}$ candidates for a k-bits error). Table 4.10 presents the average number of candidates for different packet lengths in the cases of one and two bits in error by using the checksum verification.

Table 4.10 Average number of candidates for different observed packet lengths from a simulation using H.264 Baseline packets.

Packet length	**One-Bit Error**		**Two-Bit Errors**	
	Average number of candidates by			
	ESLD	CFLD	ESLD	CFLD
272	272	9	36856	142
880	880	28	386,760	1526
1112	1112	35	617,716	2549
2240	2240	70	2,507,680	9,531
5272	5272	165	13,894,356	56,991
Eliminated candidates(%)	**97%**		**99.6%**	

4.6 Experimental results

In this section, we present the experimental results of our proposed approach. We only consider a single-bit error since for small values of ρ (e.g., 10^{-6}), the probability of having two or more bits in error is extremely low. After describing the experimental setup in detail, we will demonstrate the superior performance of the proposed approach in comparison with other state-of-the-art approaches. To illustrate this point, the approaches are compared from the standpoint of visual quality and complexity. In the simulations, we assume that the checksum is intact and the error is in the video payload. This is reasonable for 10,000-bit video packets since we will have 1 chance out of 625 (i.e., 10000/16) that the checksum is hit instead of the video payload. Furthermore, in the last subsection, we will consider adding the complementary approach proposed in the previous chapter (if the received corrupted packet is decodable and the number of decoded MBs is right, keep that as the final candidate) into the CFLD approach to determine how much improvement it will bring to CFLD. Therefore, if only the checksum is erroneous, the two conditions will be met and the packet will be kept.

4.6.1 Simulation setup

We carry out the simulations using the H.264 Baseline profile, which is typically used in conversational services and mobile applications, and the HEVC Low Delay P Main profile. We use the Joint Model (JM) software, version 18.5 (Joint Video Team (JVT) of ISO/IEC MPEG

and ITU-T VCEG, 2013) for H.264 and the HEVC Test Model (HM) software, version 15 (HEVC Test Model Software, 2016), for HEVC. The first 60 frames of NTSC (720×480) sequences (*Driving*, *Opening-ceremony*, *Whale-show*), 4CIF (704×576) sequences (*City*, *Crew*, *Ice*), CIF (352×288) sequence (*Foreman*) and PAL (720×576) sequence (*Walk*) are coded with JM 18.5. The sequences are coded in IPPP... format (Intra refresh rate of 30 frames) at a 30 Hz frame rate. Each slice contains a single row of MBs, and is encapsulated into RTP packets. We also carry out the simulation on HEVC sequences. The first 50 frames of five class B (1920×1080) sequences (*BasketballDrive, BQTerrace, Cactus, Kimono* and *ParkScene*) and four class C (832×480) sequences (*BasketballDrill*, *BQMall*, *PartyScene* and *RaceHorses*) are coded by HM. The slicing mode is chosen to fix the number of CTUs in a slice. One row of 64×64 CTUs is considered per slice. All the sequences are encoded with different QP values, namely, 22, 27, 32, and 37.

For each QP, a single frame is randomly selected for error. Then, we apply a uniform error distribution on the bits of each packet with a ρ value varying between approximately 10^{-7} for small QPs to 10^{-6} for large QPs to obtain one bit in error. These residual bit error rates are much higher than those observed in some broadcasting systems, such as DVB-H and DVB-SH-A, in recommended operational conditions (Polák & Kratochvíl, 2011). Moreover, for this range of bit-error rates, having more than one bit in error is extremely infrequent. To simplify the simulations, we just consider the errors in the payload part. Also, the UDP checksum is only calculated on the UDP payload, which is an RTP packet. In our transmission simulations, the corrupted slices are identified prior to their decoding by verifying the checksum. The simulation is repeated 100 times for each QP value, to ensure that the location of the erroneous bits does not bias our conclusions.

In H.264 cases, four different approaches are then used to handle the corrupted sequences: (i) frame copy (FC) concealment by JM (in which a corrupted slice is replaced by the same collocated slice from the previous frame), (ii) state-of-the-art spatiotemporal boundary matching algorithm (STBMA) (Chen *et al.*, 2008), (iii) error correction using hard output maximum likelihood decoding (HO-MLD) (Caron & Coulombe, 2015), and (iv) the proposed CFLD approach.

The first 30 frames are kept intact to allow the HO-MLD approach to gather video statistics. When the CFLD approach falls back to error concealment, here we consider STBMA to be fair with other approaches (HO-MLD relies on STBMA when it can not correct the packet). However, our method never reached the point of calling error concealment during the simulations. In the case of HEVC sequences, the corrupted packets are handled by (i) implemented FC error concealment in HM and (ii) the CFLD approach.

Like all other list decoding approaches, the generated candidate bitstreams should go through additional constraints. First, all the candidate bitstreams should be checked for syntax or semantic errors. We perform this by decoding each candidate bitstream to see whether or not it is decodable. Secondly, we check whether or not the number of MBs or CTUs in the decoded bitstream is valid (Nguyen *et al.*, 2010; Farrugia & Debono, 2011). Since we coded the sequence with one row of MBs (or fixed number of CTUs) in each slice, we already know about the second condition. Moreover, the number of MBs in the slice can be deduced from the information within other slices (for instance, the *first MB in slice* syntax element in H.264 coded sequence). The first bitstream that satisfies these two mentioned constraints is considered as the final candidate.

4.6.2 Simulation Results

Table 4.11 shows the candidate reduction at each step of the proposed approach, for H.264 and HEVC sequences. As can be observed, with the CFLD method, the checksum helps eliminate about 97% of the candidates. Then, as a complementary step, the two conditions are successively applied on candidate bitstreams. The last two columns in the table present the extent to which the two conditions are excluding non-valid candidates. There are some cases where, at the end of the process, more than one candidate is present. We observed that this happens less frequently in HEVC, where sequences are coded using context-adaptive binary arithmetic coding (CABAC), versus with H.264 context-adaptive variable-length coding (CAVLC). We conjecture that the use of CABAC is the reason why HEVC is much more sensitive to errors (easier to desynchronize) than the H.264 Baseline. We expect that the H.264 Main profile,

using CABAC, would be more sensitive to errors than the Baseline profile, and therefore, lead to the elimination of more candidates.

Table 4.11 Candidate reduction at each step of the CFLD method for H.264 *City* sequence, and HEVC *BasketballDrive* sequence. The letters F, S, B in the first column showing the frame, slice and bit that are hit by an error.

Error location	**Packet size** (bits)	**Number of candidates with valid...**		
		① = **checksum**	② = ① + **syntax/ semantic**	② + **number of MBs/CTUs**
H.264, *City*, QP=27 and 44 MBs per slice				
F35_S7_B2872	2952	87	4	1
F53_S16_B4312	4384	134	54	52
F35_S34_B2784	2856	96	1	1
F52_S22_B3925	4000	126	3	1
F35_S22_B823	3760	113	1	1
F51_S32_B3475	3544	110	2	1
F48_S13_B4675	4712	138	61	44
F42_S23_B304	2160	66	1	1
F44_S10_B400	2392	84	1	1
F41_S1_B1251	1360	51	19	3
HEVC, *BasketballDrive*, QP=22 and 30 CTUs per slice				
F25_S8_B11190	18016	564	3	1
F46_S10_B57355	58232	1815	51	2
F40_S7_B33218	55328	1758	21	2
F37_S7_B11757	19968	616	4	1
F45_S2_B5339	9520	294	2	1
F4_S3_B13211	28304	891	10	1
F38_S12_B19672	25496	820	13	1
F14_S11_B428	26152	815	365	1
F38_S4_B4266	6680	221	1	1
F13_S8_B10614	16192	517	7	1

For performance evaluation, we calculated the peak signal-to-noise ratio (PSNR)[2] of the corrupted frames after reconstruction, using various approaches in order to compare their objective video quality. The structural similarity index measurement (SSIM) (Wang *et al.*, 2004)

[2] In this thesis, by PSNR of a method, we always mean the Y-PSNR with reference to the original frame.

was also evaluated as it is well-known that SSIM is better correlated to human visual judgment than PSNR. Table 4.12, Table 4.13 and Table 4.14 display the average PSNR value for different error handling approaches on H.264, HEVC class B and C sequences, respectively. The column ① in the tables demonstrates the percentage of times the proposed CFLD approach was able to fully correct the packet (in other words the bit in error was corrected). This percentage value was affected by considering the first valid candidate as the final one in the proposed CFLD approach and, of course, this could be higher if more than one candidate were considered. Moreover, the last column, named as ②, shows the percentage of times that the PSNR of the reconstructed bitstream is almost the same as the intact one (with less than a 0.01 dB difference). This latter value will confirm that although in some cases the bit error was not corrected, at the end, the first valid candidate by CFLD has satisfactory results in most cases. The simulation was repeated 100 times for each sequence for different QP values. The results for the H.264 sequences indicate that the proposed approach outperforms JM-FC, STBMA and HO-MLD in all cases.

Figure 4.16 shows the average PSNR gains of each approach in the case of H.264 coded sequences at different QP values. We observe that the proposed approach provides significant PSNR gains over JM-FC for all four QP values. For instance, it is more than 5 dB better than JM-FC at QP=22. As shown in Table 4.12 for the H.264 case, on average, over all QPs, the CFLD approach was able to correct the bitstream 66% of the time compared to HO-MLD with only 6% in our simulations. Also, it offers a 2.74 dB gain over JM-FC and 1.14 dB and 1.42 dB gains over STBMA and HO-MLD, respectively, as shown in Table 4.12.

In the case of HEVC, the CFLD approach corrects the corrupted bitstream 91% of the time, and offers 2.35 dB and 4.97 dB gains over HM-FC in class B and C sequences, respectively (see Table 4.13 and Table 4.14).

Table 4.12 Comparison of the average PSNR of reconstructed corrupted frames for different methods in H.264. The differences between each method and the JM-FC appear in parentheses. The column ① shows the percentage of packets that were fully corrected by CFLD approach and the column ② shows the percentage of the cases with less than 0.01 dB PSNR difference between CFLD and Intact (with respect to the original one).

Seq.	**QP**	**Average PSNR of reconstructed corrupted frame**					CFLD	
		Intact	JM-FC	STBMA	HO-MLD	CFLD	①	②
City (704×576)	22	40.87	36.19	40.29 (+4.10)	39.27 (+3.08)	40.77 (+4.58)	51%	84%
	27	36.65	34.28	36.45 (+2.17)	35.65 (+1.37)	36.55 (+2.27)	61%	85%
	32	33.05	32.04	32.98 (+0.94)	32.66 (+0.62)	33.00 (+0.96)	66%	72%
	37	30.05	29.55	30.01 (+0.46)	29.91 (+0.36)	30.01 (+0.46)	76%	82%
Crew (704×576)	22	41.78	39.21	40.64 (+1.43)	40.28 (+1.07)	41.76 (+2.55)	71%	89%
	27	38.53	37.09	38.03 (+0.94)	37.90 (+0.81)	38.52 (+1.43)	59%	86%
	32	35.69	34.96	35.44 (+0.48)	35.36 (+0.40)	35.66 (+0.70)	69%	81%
	37	33.00	32.64	32.86 (+0.22)	32.85 (+0.21)	32.99 (+0.35)	72%	83%
Ice (704×576)	22	43.70	39.18	41.74 (+2.56)	41.66 (+2.48)	43.56 (+4.38)	69%	82%
	27	41.44	38.00	40.05 (+2.05)	40.07 (+2.07)	41.25 (+3.25)	71%	75%
	32	39.00	36.50	38.15 (+1.65)	38.08 (+1.58)	38.95 (+2.45)	74%	80%
	37	36.43	34.37	35.77 (+1.40)	35.71 (+1.34)	36.42 (+2.05)	83%	88%
Foreman (352×288)	22	41.35	37.60	39.49 (+1.89)	39.05 (+1.45)	41.01 (+3.41)	63%	71%
	27	37.82	35.79	36.92 (+1.13)	36.74 (+0.95)	37.65 (+1.86)	72%	82%
	32	34.67	33.70	34.19 (+0.49)	34.09 (+0.39)	34.58 (+0.88)	71%	73%
	37	31.92	31.39	31.63 (+0.24)	31.65 (+0.26)	31.88 (+0.49)	77%	83%
Opening ceremony (720×480)	22	39.39	38.37	38.58 (+0.21)	38.67 (+0.30)	39.32 (+0.95)	60%	90%
	27	35.38	34.90	35.02 (+0.12)	35.06 (+0.16)	35.34 (+0.44)	62%	85%
	32	31.39	31.20	31.26 (+0.06)	31.26 (+0.06)	31.38 (+0.18)	70%	86%
	37	27.69	27.64	27.65 (+0.01)	27.64 (+0.00)	27.69 (+0.05)	80%	94%
Whale show (720×480)	22	41.02	35.61	36.86 (+1.25)	36.86 (+1.25)	40.63 (+5.02)	53%	64%
	27	36.37	33.67	34.38 (+0.71)	34.42 (+0.75)	35.11 (+1.44)	60%	74%
	32	32.07	30.89	31.22 (+0.33)	31.13 (+0.24)	32.06 (+1.17)	65%	81%
	37	28.35	27.89	28.02 (+0.13)	27.98 (+0.09)	28.33 (+0.44)	74%	85%
Driving (720×480)	22	41.02	34.05	38.08 (+4.03)	37.39 (+3.34)	40.96 (+6.91)	58%	77%
	27	37.05	32.59	35.59 (+3.00)	34.75 (+2.16)	36.79 (+4.20)	62%	71%
	32	33.29	30.84	32.64 (+1.80)	32.17 (+1.33)	33.22 (+2.38)	57%	74%
	37	30.00	28.84	29.72 (+0.88)	29.49 (+0.65)	29.96 (+1.12)	63%	76%
Walk (720×576)	22	43.19	30.62	35.33 (+4.71)	34.65 (+4.03)	42.87 (+12.25)	65%	74%
	27	39.25	30.20	34.95 (+4.75)	33.93 (+3.73)	39.21 (+9.01)	63%	78%
	32	35.55	29.30	33.51 (+4.21)	32.50 (+3.20)	35.42 (+6.12)	64%	75%
	37	31.98	28.08	30.88 (+2.80)	30.38 (+2.30)	31.95 (+3.87)	64%	83%
Average PSNR gain over JM-FC			**0**	**+1.60**	**+1.32**	**+2.74**	**66%**	**80%**

Table 4.13 Comparison of the average PSNR of reconstructed corrupted frames for different methods in HEVC class B sequences. The differences between the CFLD and HM-FC methods appear in parentheses. The column ① shows the percentage of packets that were fully corrected by the proposed approach and the column ② shows the percentage of the cases with less than 0.01 dB PSNR difference between CFLD and Intact, both computed with respect to the original one.

Seq. (class B) (1920×1080)	**QP**	**Average PSNR of reconstructed corrupted frame**			CFLD	
		Intact	HM-FC	CFLD	①	②
BQ Terrace	22	38.89	35.16	35.76 (+0.60)	58%	58%
	27	36.30	34.32	35.68 (+1.36)	82%	82%
	32	33.76	32.37	33.66 (+1.29)	92%	92%
	37	31.17	30.26	31.15 (+0.89)	98%	98%
Basketball Drive	22	39.89	32.53	38.49 (+5.95)	84%	84%
	27	38.23	32.28	37.67 (+5.39)	90%	92%
	32	36.70	31.81	36.47 (+4.66)	96%	96%
	37	34.80	31.51	34.80 (+3.29)	100%	100%
Cactus	22	39.20	36.82	37.89 (+1.07)	76%	76%
	27	36.74	34.59	36.25 (+1.66)	88%	88%
	32	34.65	33.56	34.59 (+1.03)	98%	98%
	37	32.31	31.55	32.03 (+0.48)	96%	96%
Kimono	22	42.15	36.69	41.62 (+4.93)	90%	92%
	27	40.04	36.10	39.81 (+3.71)	96%	96%
	32	38.20	34.78	38.07 (+3.29)	98%	98%
	37	35.30	33.40	35.30 (+1.90)	98%	98%
Park Scene	22	40.11	37.39	39.63 (+2.24)	82%	82%
	27	37.33	35.42	37.19 (+1.77)	96%	96%
	32	34.83	33.86	34.74 (+0.88)	94%	94%
	37	32.17	31.58	32.17 (+0.59)	100%	100%
Average PSNR gain over HM-FC						
All Sequences	22	-	0	+2.96	78%	78%
	27	-	0	+2.78	90%	91%
	32	-	0	+2.23	96%	96%
	37	-	0	+1.43	98%	98%
Average		-	**0**	**+2.35**	**91%**	**91%**

As mentioned earlier, in the proposed system, we select the first candidate which satisfies the two conditions but it is not always the optimal one, i.e., the one with a corrected bitstream. Some of the first valid candidates have very low PSNR values, which has a negative impact on the average PSNR values shown in Table 4.12, Table 4.13 and Table 4.14.

Table 4.14 Comparison of the average PSNR of reconstructed corrupted frames for different methods in HEVC class C sequences. The differences between the CFLD and HM-FC methods appear in parentheses. The column ① shows the percentage of packets that were fully corrected by the proposed approach and the column ② shows the percentage of the cases with less than 0.01 dB PSNR difference between CFLD and Intact, both computed with respect to the original one.

Seq. (class C) (832×480)	**QP**	**Average PSNR of reconstructed corrupted frame**			CFLD	
		Intact	HM-FC	CFLD	①	②
Basketball Drill	22	40.44	31.90	39.91 (+8.01)	94%	94%
	27	37.41	30.84	37.06 (+6.22)	94%	94%
	32	34.66	30.07	34.56 (+4.49)	98%	98%
	37	32.11	29.21	32.00 (+2.80)	98%	98%
BQ Mall	22	39.84	31.04	39.16 (+8.12)	92%	94%
	27	36.91	30.03	36.23 (+6.20)	92%	94%
	32	33.86	29.69	33.48 (+3.79)	94%	94%
	37	30.68	27.83	30.50 (+2.67)	92%	94%
Party Scene	22	38.14	32.57	35.00 (+2.43)	72%	72%
	27	34.66	31.32	33.52 (+2.20)	84%	84%
	32	31.07	29.38	30.98 (+1.60)	96%	96%
	37	27.76	26.94	27.47 (+0.53)	94%	94%
Race Horses	22	39.29	26.01	35.94 (+9.93)	70%	74%
	27	36.21	25.48	35.16 (+9.68)	90%	90%
	32	32.60	25.80	32.18 (+6.38)	92%	94%
	37	29.44	24.98	29.32 (+4.34)	96%	96%
Average PSNR gain over HM-FC						
All Sequences	22	-	0	+7.12	82%	83%
	27	-	0	+6.08	90%	90%
	32	-	0	+4.07	95%	96%
	37	-	0	+2.59	95%	95%
Average		-	**0**	**+4.97**	**91%**	**91%**

For further analysis, we also perform some simulations based on the brute force search or ESLD approach. In the ESLD approach, all candidates will sequentially go through the video decoder and the first candidate that satisfies the decoder's two conditions is chosen as the final candidate. The candidates are generated by sequentially flipping the bits of the received packet from the first to the last one. We use the same order for CFLD but only considering the potential bit error locations.

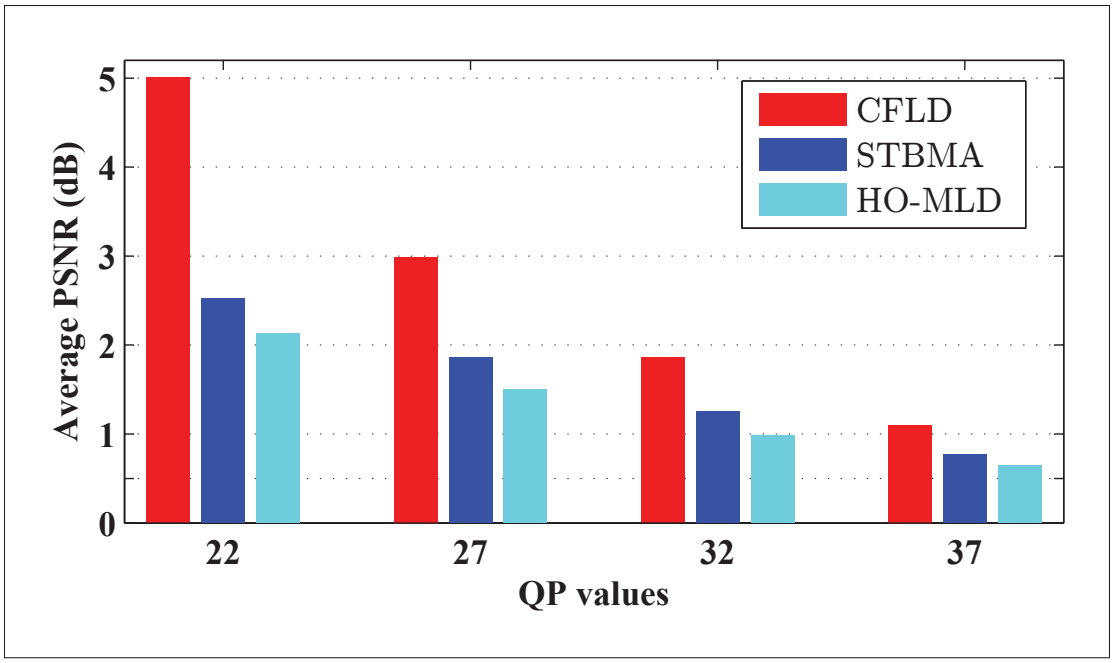


Figure 4.16 Average PSNR gains of HO-MLD, STBMA and CFLD method over JM-FC for different QP values of H.264 sequences on different frames.

Figure 4.17 presents the PSNR and SSIM distributions (box plots) of four sequences having a low percentage of fully corrected slices in Table 4.12. As can be observed from the figures, for all the sequences, the median value (red line in the middle of the box) of PSNR and SSIM for CFLD is exactly the same as the intact one and also the lower and higher bands of boxes (25-75 percentile of the data) confirm that in most cases the CFLD has the same or closest value to the intact one which is obviously higher than the other approaches. This has a huge impact on the visual quality of the reconstructed corrupted frame and, more importantly, prevents the propagation of errors to subsequent frames due to the predictive coding. In fact, a few decibels PSNR difference on the reconstructed corrupted frame increases to several dBs on subsequent frames due to this drift. Since in the simulations, we choose the first satisfied candidate as the final one, there are some outliers (as shown with '+' red symbol) in the CFLD results.

The detailed information of this simulation for 100 runs is presented in Table 4.15. From the table, we observe that the CFLD can outperform ESLD approach in all cases. As an instance, the ESLD perfectly corrects damaged H.264 packet 36% and 32% of the time for *Foreman*

and *Ice* sequences, respectively, while the values for CFLD are 66% and 61%. On top of that, the CFLD search is much less complex than the ESLD search and it significantly reduces the number of candidates from N (for ESLD) to $N/32$. This is shown in Figure 4.18. For example, in the case of *Foreman* sequence at QP 22, there are 100 to 500 non valid candidates (detected by decoding semantic errors) before reaching the first valid candidate while in CFLD it can be achieved by only 5 to 15 decoding of candidates. In fact, if CFLD fails to fully correct the packet, for sure ESLD will fail. This is because ESLD will always retain a candidate that either comes before that of CFLD or the same one. Therefore, it is not possible for ESLD to select a fully corrected packet without CFLD also selecting it.

Table 4.15 Detailed information of the box plot of Figure 4.17: average PSNR and SSIM values, percentage of fully corrected packets.

Seq.		Intact	JM-FC	STBMA	HO-MLD	ESLD	CFLD
Foreman QP=22	PSNR	41.37	38.47	39.78	39.41	41.14	41.33
	SSIM	0.0.9691	0.9632	0.9663	0.9659	0.9687	0.969
	fully corrected packets (%)				–	36%	66%
	less than 0.01 dB PSNR difference (%)				6%	61%	79%
	less than 0.05 dB PSNR difference (%)				9%	80%	93%
Ice QP=27	PSNR	41.53	38.06	40.00	39.96	41.28	41.33
	SSIM	0.9833	0.9802	0.9821	0.982	0.9831	0.9832
	fully corrected packets (%)				–	32%	61%
	less than 0.01dB PSNR difference (%)				15%	66%	70%
	less than 0.05dB PSNR difference (%)				15%	80%	80%
City QP=32	PSNR	33.43	32.01	33.36	33.04	33.06	33.38
	SSIM	0.954	0.9435	0.9535	0.9511	0.9513	0.9537
	fully corrected packets (%)				–	19%	69%
	less than 0.01dB PSNR difference (%)				8%	47%	79%
	less than 0.05dB PSNR difference (%)				30%	59%	88%
Driving QP=37	PSNR	29.61	28.45	29.34	29.04	29.50	29.57
	SSIM	0.8908	0.8768	0.8876	0.8841	0.8897	0.8904
	fully corrected packets (%)				–	22%	56%
	less than 0.01dB PSNR difference (%)				1%	51%	69%
	less than 0.05dB PSNR difference (%)				1%	77%	87%

A visual quality inspection of the results is illustrated in Figure 4.19. The figure depicts the gain in subjective quality of frame 45 of *Ice* sequence at QP 37 for different approaches. Compar-

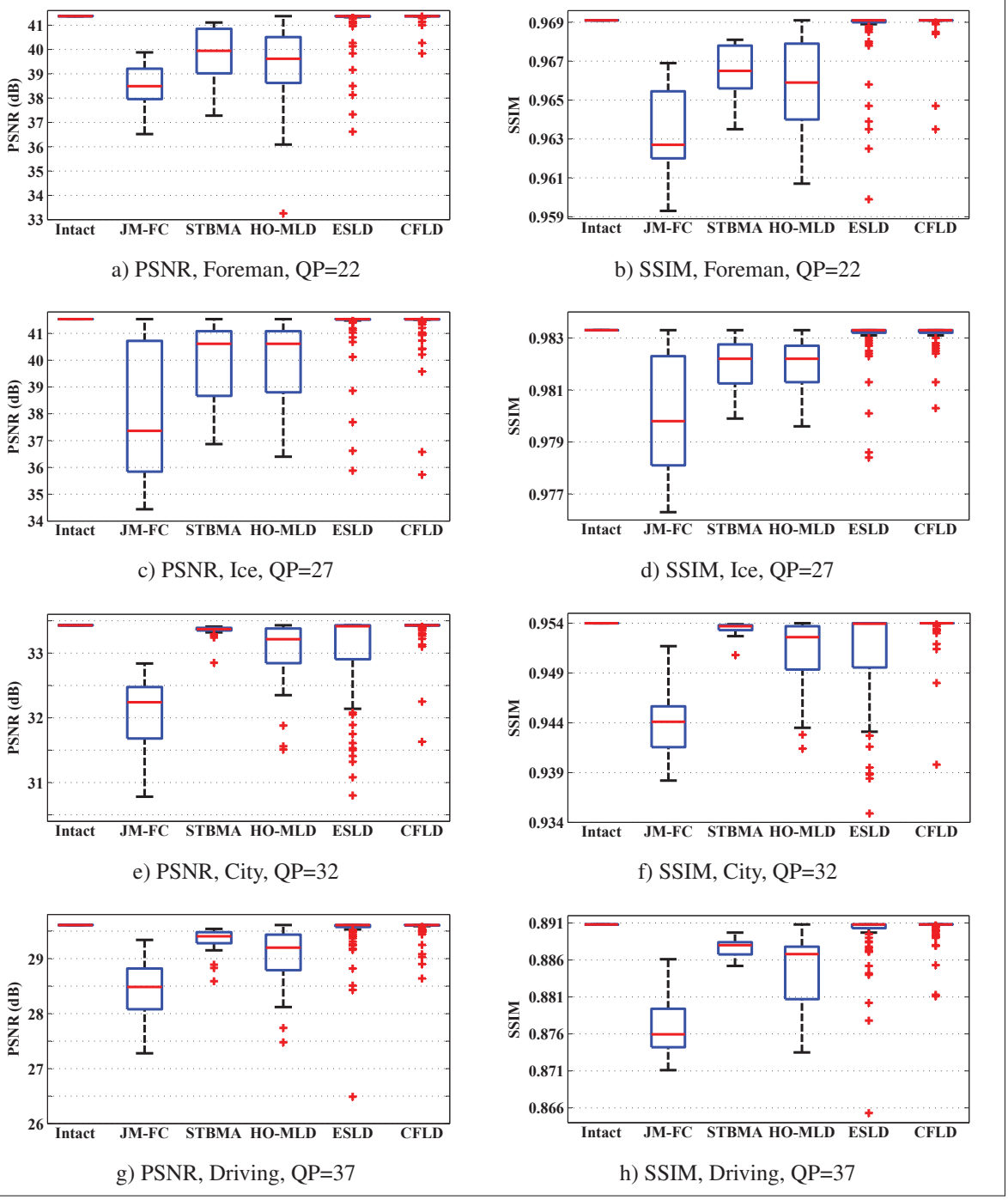


Figure 4.17 PSNR and SSIM distributions of 100 runs on frame 45 of H.264 sequences.

ing the reconstructed frame and the luminance difference, it is clear that the CFLD approach

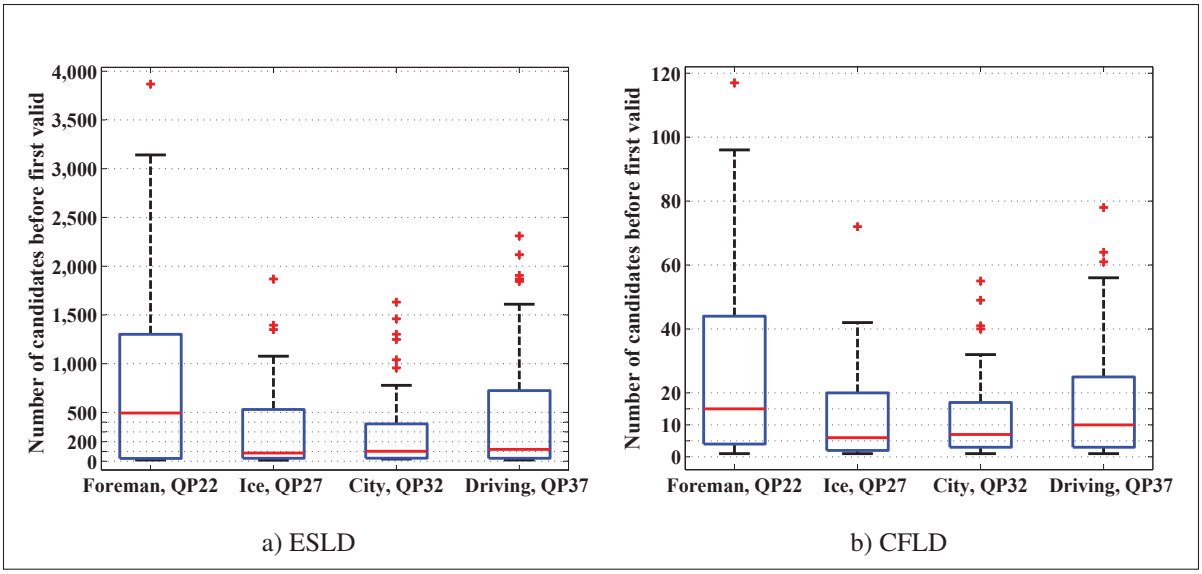


Figure 4.18 Number of candidates before the first valid candidate in each case of CFLD and ESLD approach.

outperforms the other approaches and further confirms the robustness and superiority of the proposed method.

From the results of all figures and tables, it can be inferred that the proposed CFLD approach can effectively remove non-valid candidates in comparison to conventional list decoding approaches, leading to a reduction of 97% of the number of candidates for the case of one bit error and in nearly 88% of the cases in H.264, and 91% of the cases in HEVC, the reconstructed sequence can have very close PSNR value to the intact one. So, as a result, the proposed CFLD provides a significantly higher PSNR value and better quality compared to other approaches. This is important not only for the corrupted frame, but for the following ones, as fewer visible drifting effects will result. As we mentioned, since the first candidate which satisfies the two conditions is kept as the final one, there are some cases with low PSNR value. However, we believe that most cases which have very low PSNR, can be eliminated by adding an additional pixel-domain step (such as boundary matching or border checking) in our system. Indeed, instead of selecting the first candidate which satisfies the two conditions, we could rank all candidates satisfying the two conditions using a yet-to-be-defined pixel-domain likeliness measure or other likeliness measure based on the decoded information (e.g., motion vectors). For in-

Figure 4.19 Visual comparison of a reconstructed frame with H.264 *Ice* sequence at QP=37 by different methods. One bit was flipped in frame 45, slice 22 and bit 381. The packet contains 472 bits. The proposed checksum provides 11 candidates. The first valid candidate which satisfies the two mentioned conditions is picked as CFLD output and the error in the packet is perfectly corrected. The PSNR and SSIM values of each approach are as follows, respectively: Intact (36.49 dB, 0.9681), JM-FC (34.12 dB, 0.9649), STBMA (34.37 dB, 0.9659), HO-MLD (34.39 dB, 0.966) and CFLD (36.49 dB, 0.9681).

stance, for all the candidates satisfying the two conditions, we could use a pixel-domain metric such as the one based on the sum of distributed motion-compensated blockiness (SDMCB) proposed in (Trudeau *et al.*, 2011) to rank them. We thus could select the candidate having

the highest likeliness (e.g., lowest SDMCB value). But this is outside the scope of the current work.

4.6.3 Comparison of CFLD and CFLD$^+$

Our observation on the received corrupted packet from the simulations on H.264 sequences reveal that in more than 30% of the case, the received corrupted packets, from different sequences at different QPs, were decodable and the number of decoded MBs were right (see Figure 4.20). Therefore, in those cases, we can keep the corrupted packet as the final candidate instead of going through the CFLD approach. In this subsection, the same error patterns used in the previous simulations on H.264 sequences are used to compare the performances of the CFLD approach when it integrates with the proposed method in the previous chapter (section 3.3). The integrated proposed CFLD approach is described here as CFLD$^+$ as follows: when a corrupted packet is received, if it satisfies the two conditions (decodable and having the correct number of decoded MBs), it is kept as the final candidate; otherwise the final candidate is chosen by the CFLD approach. Note that in the previous simulations, all the corrupted packets (no matter if the conditions are met on the received corrupted packet or not) went through the CFLD approach to find the final candidate.

We compare the two mentioned approaches from the complexity and PSNR viewpoints. For the complexity comparison, the number of non-valid candidates, in other words, number of extra decodings, before finding the first valid candidate, is considered for each approach. Figure 4.22 presents the number of non-valid candidates before the first valid one, in the CFLD approach for two different H.264 coded sequences. From the left box plot of the *Foreman* sequence, it seems that in 25-75 percentile of the cases (the lower and higher bands of boxes), the number of non-valid candidates of CFLD is between 5-45 candidates, with a median of 15 candidates. In order to better compare the approaches, the left box plot is divided into two separate cases based on if the received corrupted packet meets the conditions or not. The middle box plot depicts the number of candidates for CFLD, when the received corrupted packet satisfies the two conditions and the right box plot (shown as “Others”) depicts when it does not meet the

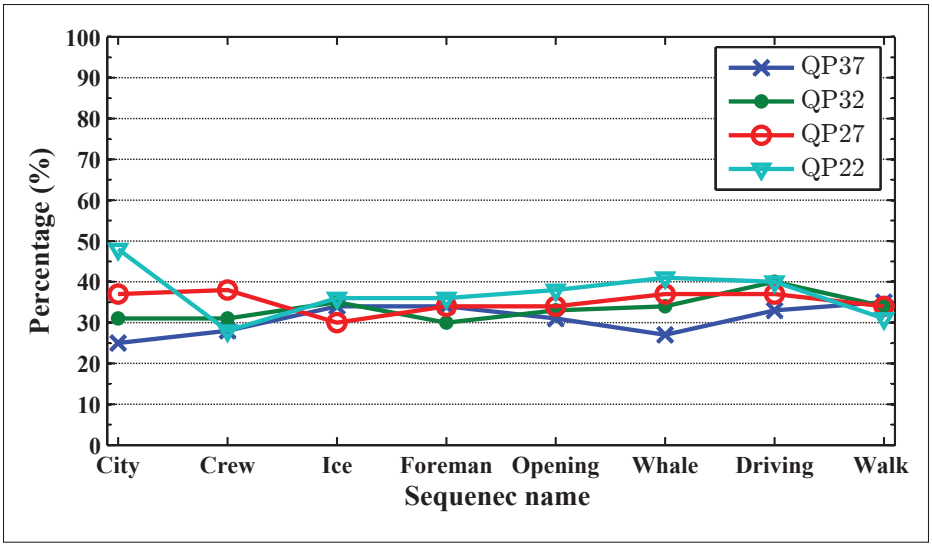


Figure 4.20 Percentage of the cases that the received corrupted packet satisfies the two conditions.

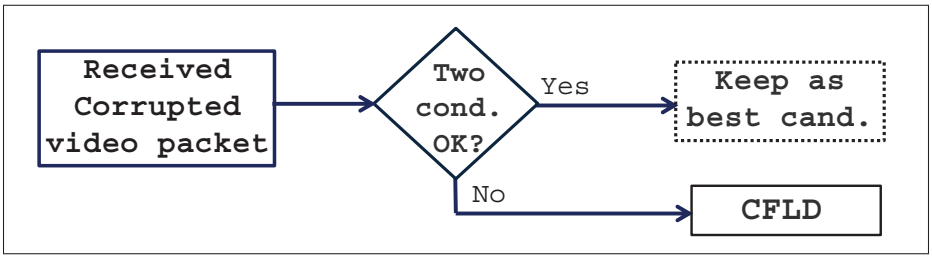


Figure 4.21 Proposed CFLD^{+} approach.

two conditions. As it can be seen in Figure 4.21 and explained before, the two approaches behave differently only when the received corrupted packet meets the conditions. Thus the middle box plot identifies their difference in complexity. As can be observed from the middle box plot, when the received corrupted packet meets the two conditions, the CFLD finds the first valid candidate sooner than in the other cases, this will bring overhead of, on average, three extra decoding compared to CFLD^{+} approach in around 30% of the cases.

From the performance viewpoint, the CFLD^{+} approach can also improve the quality (PSNR) of the reconstructed frame, overall on all sequences, by around 0.3 dB at QP=22 to 0.1 dB at QP=37, specifically when the two conditions are met (see Figure 4.23). Note that in the case

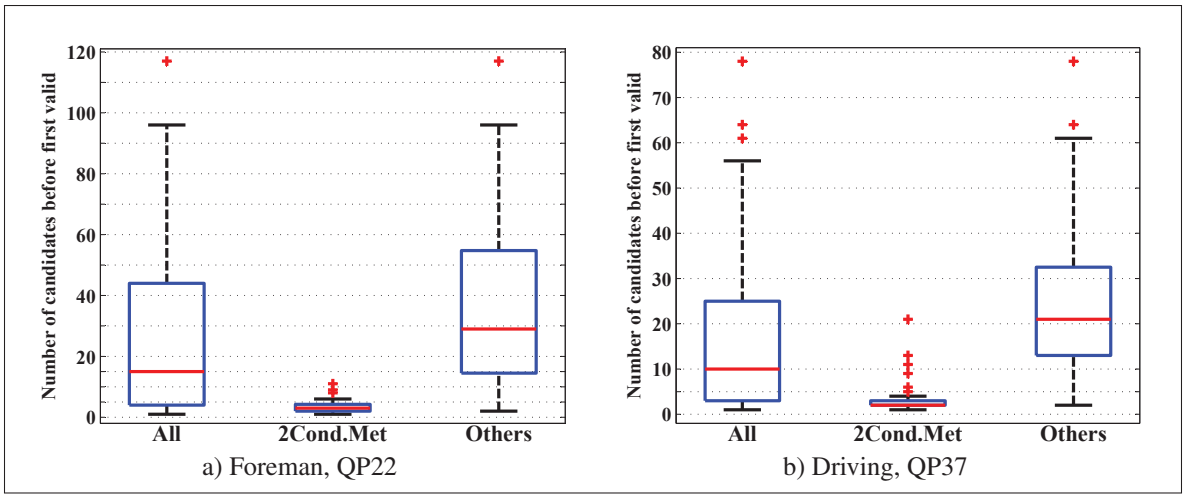


Figure 4.22 Number of non-valid candidates (extra decodings) before the first valid candidate in CFLD approach. The left box plot contains all the cases for 100 run simulations and is further divided into two separate box plots: when the received corrupted packet meets the two conditions (middle box plot) and when it is not (right box plot).

of the CFLD approach, when the received corrupted packet satisfies the two conditions at the reception, it is very unlikely to correct the error (unless the error is at the very beginning of the packet). Therefore CFLD will end up with a candidate that has two bits in error while, in the CFLD^{+} approach, the final candidate has only the one bit error (the actual received erroneous one). For example, in *Ice* sequence at QP=27, as illustrated in Figure 4.24, from 100 simulations, in 30 cases, the received corrupted packet satisfies the two conditions and the CFLD approach was able to (fully) correct error in only 4 cases. In the remaining 70 cases ("Others"), where the two conditions are not meet, the packet was perfectly corrected in 67 cases. Similar results have been observed for the other sequences and QP values. The results support that the two conditions constitute a proper constraint to remove non-valid candidates provided by the checksum while, when the received corrupted packet is decodable and the number of MBs is right, the two constraints for filtering non-valid candidates will not perform very well. But remember that, as we presented in the previous chapter, if the received corrupted packet satisfies the two conditions, in more than 90% of the case for one bit in error, it has very close PSNR to the intact one therefore there is not need to process further the packet.

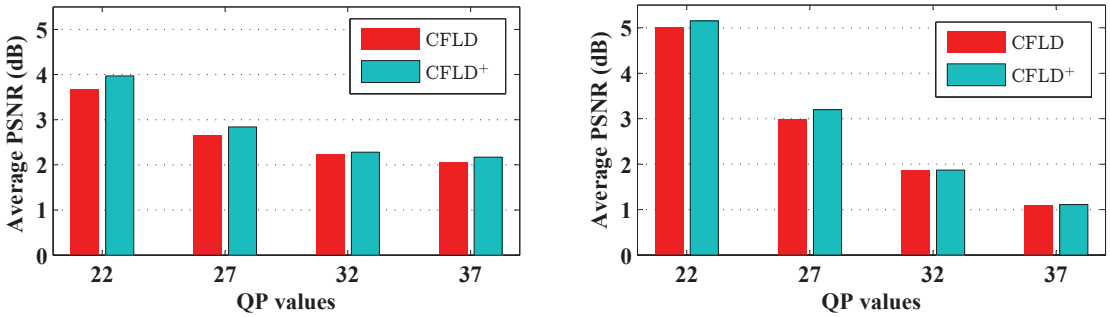


Figure 4.23 Average PSNR (dB) gain over JM-FC for CFLD and CFLD$^+$ on different sequences at different QP values.

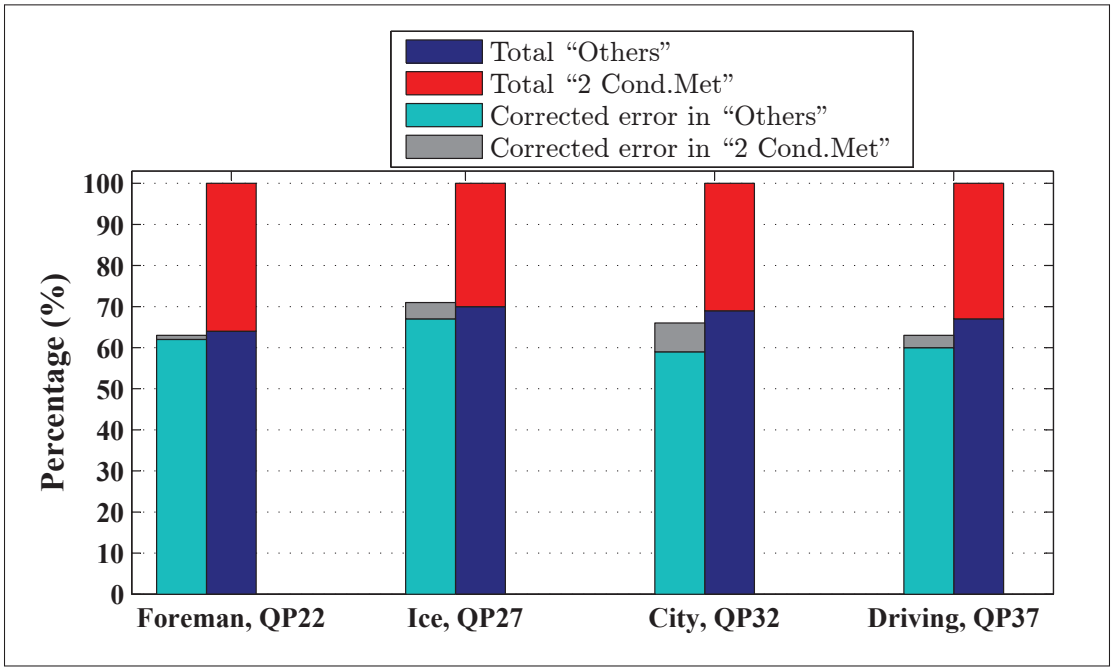


Figure 4.24 Percentage of the times that error was corrected by the CFLD approach in two separate cases: when the received packet meets two conditions (1% at QP=22, 4% at QP=27, 7% at QP=32 and 3% at QP=37) and when it does not, described as "Others", (62% at QP=22, 67% at QP=27, 59% at QP=32 and 60% at QP=37).

The same strategy can be used when there are more than one bit in error. As we have discussed in section 3.5 for two bits in error, the probability of having both errors on non-desynchronizing bits (NDBs) is very low, but even in those cases, CFLD$^+$ can have better performance com-

pared to CFLD. In other words, retaining the corrupted packet as a final candidate is better than going through the correction process of CFLD since There are more chances that the CFLD will damage more (not able to fix the errors then add other errors) than fix the packet. Moreover, as the results have shown, when the received packet is not satisfying the conditions, it is more likely that the CFLD will be able to fix the errors. This is because, the non-valid candidates will flip other bits (adding more errors in the packet) which increases the chances of desynchronization and non-valid syntaxes. Therefore, more non-valid candidates will be filtered by the two conditions and probably the final one would be the best candidate.

CONCLUSION AND RECOMMENDATIONS

In this thesis, we presented two different mechanisms to enhance the quality of reconstructed corrupted frames in the presence of transmission errors. This has been achieved by proposing different methods applicable in the context of error concealment and error correction. Our contributions can be summarized as follows:

- We have addressed the reliability of received corrupted video packets in concealment. We have identified the most common NDBs syntax elements in H.264 CAVLC coded sequences. It was observed that, on average, the NDBs make up one-third of all bitstreams. The simulation results revealed that the effect of NDBs on context modification is insignificant and that the majority of them (90%) provide PSNR values that are highly comparable to the intact value. Our proposed approach keeps the corrupted packets if only the NDBs are erroneous. The visual difference from intact in this case is much smaller than the one introduced by concealment approaches. The main advantage of the proposed approach is that it can be combined with any available concealment approach and moreover it can reduce the complexity of the complex concealment approaches by up to 30%, by decoding the received corrupted packet, and lead to a better quality. In this work, we were more focused on the Baseline profile of H.264, but the method can be applied to other profiles with CABAC coded sequences. However, the corrupted packets in those cases may more likely cause desynchronization at the bit level. Moreover, the NDBs can be useful in actual error correction at the bit level such as list decoding. If the received corrupted packet satisfies the proposed conditions, most probably only the NDBs are erroneous and the received corrupted packet can become a candidate in list decoding approaches.

- Our proposed CFLD can correct the received corrupted packet with the use of receiver side UDP checksum value. Unlike the other existing list decoding approaches, it does not require soft information (e.g. log-likelihood ratio (LLR) of bits) during the correction.

The proposed approach can reduce the problematically large solution space of the conventional list decoding approaches. For instance, when a packet composed of N bits contains a single-bit error, instead of considering N candidate bitstreams, as is the case in conventional list decoding approaches, the proposed approach considers approximately $N/32$ candidate bitstreams, leading to a reduction of 97% of the number of candidates. For two bits in error, this reduction reaches 99.6%. Such a filtering of the candidates as proposed, supplemented by checksum information dramatically reduces the complexity of the list decoding approaches. Although, current applications do not typically have access to soft information, the proposed CFLD approach can also be applied to that context, allowing it to perform even better by enabling it to exploit the soft information to rank the candidate bitstreams in each BEE. We also expect a further increase in performance by exploiting pixel domain information to select the best decodable candidate rather than selecting the first decodable candidate which can be the subject of future research.

LIST OF REFERENCES

Asheri, H., Rabiee, H. R., Pourdamghani, N. & Ghanbari, M. (2012). Multi-directional spatial error concealment using adaptive edge thresholding. *IEEE Transactions on Consumer Electronics*, 58(3), 880-885.

Atzori, L., De Natale, F. G. & Perra, C. (2001). A spatio-temporal concealment technique using boundary matching algorithm and mesh-based warping (BMA-MBW). *IEEE Transactions on Multimedia*, 3(3), 326–338.

Barni, M., Bartolini, F. & Bianco, P. (2000). On the performance of syntax-based error detection in H.263 video coding: a quantitative analysis. *Electronic Imaging, SPIE Conf. Image and Video Communications*, 3974, 949–957.

Bergeron, C. & Lamy-Bergot, C. (2004). Soft-input decoding of variable-length codes applied to the H.264 standard. *Proc. IEEE 6th workshop Multimedia Signal Process.*, pp. 87–90.

Braden, R. T., Borman, D. A. & Partridge, C. (1989). Computing the internet checksum. IETF, RFC 1071, [Online]. Available: https://www.rfc-editor.org/rfc/rfc1071.txt.

Caron, F. (2013). *A maximun likelihood approach to video error correction applied to H.264 decoding*. (Ph.D. thesis, École de Technologie Supérieure, Montreal, Canada).

Caron, F. & Coulombe, S. (2012). A maximum likelihood approach to video error correction applied to H.264 decoding. *Proc. IEEE 6th Int. Conf. Next Gen. Mob. Appl., Serv., Technol.*, pp. 1–6.

Caron, F. & Coulombe, S. (2013). A maximum likelihood approach to correcting transmission errors for joint source-channel decoding of H.264 coded video. *Proc. IEEE 20th Int. Conf. Image Process.*, pp. 1870–1874.

Caron, F. & Coulombe, S. (2015). Video Error Correction Using Soft-Output and Hard-Output Maximum Likelihood Decoding Applied to an H.264 Baseline Profile. *IEEE Trans. Circuits Syst. Video Technol.*, 25(7), 1161–1174.

Chen, M. J., Chen, L. G. & Weng, R. M. (1997). Error concealment of lost motion vectors with overlapped motion compensation. *IEEE Trans. Circuits Syst. Video Technol.*, 7(3), 560–563.

Chen, Y., Hu, Y., Au, O. C., Li, H. & Chen, C. W. (2008). Video error concealment using spatio-temporal boundary matching and partial differential equation. *IEEE Trans. Multimedia.*, 10(1), 2–15.

Choe, G., Nam, C. & Chu, C. (2018). An effective temporal error concealment in H. 264 video sequences based on scene change detection-PCA model. *Multimedia Tools and Applications*, 1–15.

Cisco. (2017). Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021 White Paper.

Cisco. (2018). Cisco Visual Networking Index: Forecast and Trends, 2017–2022. Accessed: 2018-12-12.

Dean, R. A. (1966). *Elements of abstract algebra.* John Wiley & Sons Inc.

Demirtas, A., Reibman, A. & Jafarkhani, H. (2011). Performance of H.264 with isolated bit error: Packet decode or discard? *Proc. IEEE 18th Int. Conf. Image Process.*, pp. 949–952.

Fall, K. R. & Stevens, W. R. (2011). *TCP/IP illustrated, volume 1: The protocols*. Addison-Wesley.

Farrugia, R. A. & Debono, C. J. (2010). A hybrid error control and artifact detection mechanism for robust decoding of H. 264/AVC video sequences. *IEEE Transactions on Circuits and Systems for Video Technology*, 20(5), 756–762.

Farrugia, R. A. & Debono, C. J. (2011). Robust decoder-based error control strategy for recovery of H.264/AVC video content. *IET Communications*, 5(13), 1928–1938.

Farrugia, R. A. & Debono, C. J. (2008). Robust transmission of H.264/AVC sequences using list decoding and source constraints. *IEEE 14th Mediterranean Electrotechnical Conference (MELECON)*, pp. 885–889.

Gharavi, H. & Gao, S. (2008). Spatial interpolation algorithm for error concealment. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1153–1156.

Golaghazadeh, F. & Coulombe, S. (2017). Checksum-filtered decoding, checksum-aided forward error correction of data packets, forward error correction of data using bit erasure channels and sub-symbol level decoding for erroneous fountain codes. PCT/CA2017/051046, 2017-09-07, [Available] https://patents.google.com/patent/WO2018045459A1/en.

Golaghazadeh, F., Coulombe, S., Coudoux, F. & Corlay, P. (2017). Low complexity H.264 list decoder for enhanced quality real-time video over IP. *IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 1-6.

Golaghazadeh, F., Coulombe, S., Coudoux, F. & Corlay, P. (2018a). The Impact of H.264 Non-desynchronizing Bits on Visual Quality and its Application to Robust Video Decoding. *Presented in IEEE 12th International Conference on Signal Processing and Communication Systems (ICSPCS)*.

Golaghazadeh, F., Coulombe, S., Coudoux, F. & Corlay, P. (2018b). Checksum-Filtered List Decoding Applied to H.264 and H.265 Video Error Correction. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(8), 1993-2006.

Hagenauer, J., Offer, E. & Papke, L. (1996). Iterative decoding of binary block and convolutional codes. *IEEE Transactions on information theory*, 42(2), 429–445.

HEVC Test Model Software. (2016) (Version 15) [Software]. Consulted at https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/branches/.

Hsia, S.-C. & Hsiao, C. H. (2016). Fast-efficient shape error concealment technique based on block classification. *IET Image Processing*, 10(10), 693–700.

International Telecommunications Union. (2003). ITU-T Recommendation H.264: Advanced video coding for generic audiovisual services.

ISO/IEC JTC 1/SC 29/WG 11. (2013). High Efficiency Video Coding [Recommendation]. ITU-T H.265.

ITU-T SG16 Q.6 & ISO/IEC JTC 1/SC 29/WG11. (2003). *ITU-T Recommendation H.264: Advanced video coding for generic audiovisual services* (Report n°International Telecommunications Union).

ITU-T-StudyGroups. ITU-T SG 16 standardization on visual coding – the Video Coding Experts Group (VCEG). [Online; accessed 20-January-2018]. Available: https://www.itu.int/en/ITU-T/studygroups/2017-2020/16/Pages/video/vceg.aspx.

Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG. (2013). H.264/AVC JM Reference Software (Version 18.5) [Software]. Consulted at http://iphome.hhi.de/suehring/tml/.

Kang, K. & Sha, L. (2010). An interleaving structure for guaranteed QoS in real-time broadcasting systems. *IEEE Trans. Computers*, 59(5), 666–678.

Kim, W., Koo, J. & Jeong, J. (2006). Fine directional interpolation for spatial error concealment. *IEEE Transactions on Consumer Electronics*, 52(3), 1050–1056.

Koloda, J., Ostergaard, J., Jensen, S. H., Sanchez, V. & Peinado, A. M. (2013a). Sequential error concealment for video/images by sparse linear prediction. *IEEE Trans. Multimedia.*, 15(4), 957–969.

Koloda, J., Sánchez, V. & Peinado, A. M. (2013b). Spatial error concealment based on edge visual clearness for image/video communication. *Circuits, Systems, and Signal Processing*, 32(2), 815–824.

Koloda, J., Peinado, A. M. & Sánchez, V. (2014). Kernel-based MMSE multimedia signal reconstruction and its application to spatial error concealment. *IEEE Transactions on Multimedia*, 16(6), 1729–1738.

Koloda, J., Seiler, J., Peinado, A. M. & Kaup, A. (2017). Scalable kernel-based minimum mean square error estimator for accelerated image error concealment. *IEEE Transactions on Broadcasting*, 63(1), 59–70.

Kung, W.-Y., Kim, C.-S. & Kuo, C.-C. (2006). Spatial and temporal error concealment techniques for video transmission over noisy channels. *IEEE transactions on circuits and systems for video technology*, 16(7), 789–803.

Kwok, W. & Sun, H. (1993). Multi-directional interpolation for spatial error concealment. *IEEE Trans. Consumer Electronics*, 39(3), 455–460.

Lam, W. M., Reibman, A. R. & Liu, B. (1993). Recovery of lost or erroneously received motion vectors. *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 5, 417–420.

Lamy-Bergot, C. & Bergeron, C. (2012). Video H.264 encryption preserving synchronization and compatibility of syntax. US Patent 8,160,157, Apr. 17, 2012, https://www.google.com/patents/US8160157.

Larzon, L. A., Degermark, M., Pink, S., Jonsson, L. E. & Fairhurst, G. (2004). The lightweight user datagram protocol (UDP-Lite). https://www.rfc-editor.org/rfc/rfc3828.txt. IETF, RFC 3828.

Lee, S.-H., Choi, D.-H. & Hwang, C.-S. (2001). Error concealment using affine transform for H. 263 coded video transmissions. *Electronics Letters*, 37(4), 218–220.

Levine, D., Lynch, W. E. & Le-Ngoc, T. (2007). Iterative Joint Source-Channel Decoding of H.264 Compressed Video. *Proc. IEEE Int. Symp. Circuits Syst.*, pp. 1517-1520.

Lie, W.-N., Lee, C.-M., Yeh, C.-H. & Gao, Z.-W. (2014). Motion vector recovery for video error concealment by using iterative dynamic-programming optimization. *IEEE Transactions on Multimedia*, 16(1), 216–227.

Lin, T.-L., Chen, W.-C. & Lai, C.-K. (2013a). Recovery of lost motion vectors using encoded residual signals. *IEEE Transactions on Broadcasting*, 59(4), 705–716.

Lin, T.-L., Yang, N.-C., Syu, R.-H., Liao, C.-C. & Tsai, W.-L. (2013b). Error concealment algorithm for HEVC coded video using block partition decisions. *Signal Processing, Communication and Computing (ICSPCC), 2013 IEEE International Conference On*, pp. 1–5.

Lin, T.-L., Ding, T.-L., Fan, C.-Y. & Chen, W.-C. (2017). Error concealment algorithm based on sparse optimization. *Multimedia Tools and Applications*, 76(1), 397–413.

Lin, T.-L., Wei, X., Wei, X., Su, T.-H. & Chiang, Y.-L. (2018). Novel pixel recovery method based on motion vector disparity and compensation difference. *IEEE Access*, 44362-44375.

Liu, J., Zhai, G., Yang, X., Yang, B. & Chen, L. (2015). Spatial Error Concealment With an Adaptive Linear Predictor. *IEEE Trans. Circuits Syst. Video Technol.*, 25(3), 353–366.

Ma, M., Au, O. C., Chan, S. G. & Sun, M. T. (2010). Edge-directed error concealment. *IEEE Trans. Circuits Syst. Video Technol.*, 20(3), 382–395.

Ma, X. F. & Lynch, W. E. (2004). Iterative joint source-channel decoding using turbo codes for MPEG-4 video transmission. *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on*, 4, iv–iv.

Nguyen, N. Q., Lynch, W. E. & Le-Ngoc, T. (2010). Iterative Joint Source-Channel Decoding for H.264 video transmission using virtual checking method at source decoder. *Proc. IEEE 23rd Can. Conf. Electr. Comput. Eng.*, pp. 1–4.

Peng, Q., Yang, T. & Zhu, C. (2002). Block-based temporal error concealment for video packet using motion vector extrapolation. *Communications, Circuits and Systems and West Sino Expositions, IEEE 2002 International Conference on*, 1, 10–14.

Perera, R., Arachchi, H. K., Imran, M. A. & Xiao, P. (2016). Extrinsic information modification in the turbo decoder by exploiting source redundancies for HEVC video transmitted over a mobile channel. *IEEE Access*, 4, 7186–7198.

Persson, D. & Eriksson, T. (2009). Mixture model-and least squares-based packet video error concealment. *IEEE Transactions on Image Processing*, 18(5), 1048–1054.

Persson, D., Eriksson, T. & Hedelin, P. (2008). Packet video error concealment with Gaussian mixture models. *IEEE Transactions on Image Processing*, 17(2), 145–154.

Pfeiffer, P. E. (2013). *Concepts of Probability Theory: Second Revised Edition* (ed. 2). New York, United states: Dover Publications Inc.

Polák, L. & Kratochvíl, T. (2011). DVB-H and DVB-SH-A performance and evaluation of transmission in fading channels. *IEEE 34th Int. Conf. Telecommunications and Signal Processing (TSP)*, pp. 549-553. doi: 10.1109/TSP.2011.6043669.

Postel, J. (1980). *User datagram protocol, RFC 768.* Consulted at https://www.rfc-editor.org/rfc/rfc768.txt.

Qian, X., Liu, G. & Wang, H. (2009). Recovering connected error region based on adaptive error concealment order determination. *IEEE Transactions on Multimedia*, 11(4), 683–695.

Richardson, I. E. (2010). *The H.264 advanced video compression standard* (ed. 2nd). West Sussex, United Kingdom: John Wiley & Sons Ltd.

Robie, D. L. & Mersereau, R. M. (2000). The use of Hough transforms in spatial error concealment. *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 4, 2131–2134.

Sabeva, G., Jamaa, S. B., Kieffer, M. & Duhamel, P. (2006). Robust decoding of H.264 encoded video transmitted over wireless channels. *Proc. IEEE 8th workshop Multimedia Signal Process.*, pp. 9–13.

Salama, P., Shroff, N. B., Coyle, E. J. & Delp, E. J. (1995). Error concealment techniques for encoded video streams. *IEEE International Conference on Image Processing, 1995. Proceedings.,*, 1, 9–12.

Salama, P., Shroff, N. B. & Delp, E. J. (1998). Error concealment in encoded video streams. In *Signal Recovery Techniques for Image and Video Compression and Transmission* (pp. 199–233). Springer.

Schulzrinne, H., Casner, S., Frederick, R. & Jacobson, V. RTP: A transport protocol for real-time applications. IETF, RFC 3550, Jul. 2003. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3550.txt.

Shi, Y. Q., Zhang, X. M., Ni, Z.-C. & Ansari, N. (2004). Interleaving for combating bursts of errors. *IEEE Circuits and Systems Magazine*, 4(1), 29–42.

Shih, H.-C., Wang, C.-T. & Huang, C.-L. (2018). Spiral-Like Pixel Reconstruction Algorithm for Spatiotemporal Video Error Concealment. *IEEE Access*, 6, 6370–6381.

Song, K., Chung, T., Kim, C.-S., Park, Y.-O., Kim, Y., Joo, Y. & Oh, Y. (2007). Efficient multi-hypothesis error concealment technique for H. 264. *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, pp. 973–976.

Sullivan, G. J. & Wiegand, T. (2005). Video compression-from concepts to the H.264/AVC standard. *Proceedings of the IEEE*, 93(1), 18–31.

Sun, H. & Kwok, W. (1995). Concealment of damaged block transform coded images using projections onto convex sets. *IEEE Trans. Image Process.*, 4(4), 470–477.

Superiori, L., Nemethova, O. & Rupp, M. (2006). Performance of a H.264/AVC Error Detection Algorithm Based on Syntax Analysis. *Proc. MoMM*, pp. 49–58.

Tan, W. T., Shen, B., Patti, A. & Cheung, G. (2008). Temporal propagation analysis for small errors in a single-frame in H.264 video. *IEEE 15th Int. Conf. Image Process.*, pp. 2864–2867.

Trudeau, L., Coulombe, S. & Pigeon, S. (2011). Pixel domain referenceless visual degradation detection and error concealment for mobile video. *Proc. 18th IEEE Int. Conf. Image Process.*, pp. 2229–2232.

Wang, Y., Wenger, S., Wen, J. & Katsaggelos, A. K. (2000). Error resilient video coding techniques. *IEEE Signal Process. Mag.*, 17(4), 61–82.

Wang, Y., Ostermann, J. & Zhang, Y. Q. (2002). *Video processing and communications*. Prentice Hall Upper Saddle River.

Wang, Y. & Yu, S. (2005). Joint source-channel decoding for H.264 coded video stream. *IEEE Trans. Consum. Electron.*, 51(4), 1273–1276.

Wang, Z., Bovik, A. C., Sheikh, H. R. & Simoncelli, E. P. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Process*, 13(4), 600–612.

Weidmann, C., Kadlec, P., Nemethova, O. & Al Moghrabi, A. (2004). Combined sequential decoding and error concealment of H.264 video. *Proc. IEEE 6th workshop Multimedia Signal Process.*, pp. 299–302.

Wu, J., Liu, X. & Yoo, K. Y. (2008). A temporal error concealment method for H.264/AVC using motion vector recovery. *IEEE Trans. Consum. Electron.*, 54(4), 1880–1885.

Xiang, Y., Feng, L., Xie, S. & Zhou, Z. (2011). An efficient spatio-temporal boundary matching algorithm for video error concealment. *Multimedia Tools and Applications*, 52(1), 91–103.

Xiao, J., Tillo, T., Lin, C., Zhang, Y. & Zhao, Y. (2013). A real-time error resilient video streaming scheme exploiting the late-and early-arrival packets. *IEEE Trans. Broadcast.*, 59(3), 432–444.

Xiu, X., Zhuo, L. & Shen, L. (2006). A hybrid error concealment method based on H.264 standard. *IEEE 8th International Conference on Signal Processing*, 2. doi: 10.1109/ICOSP.2006.345630.

Yen, K., Sun, S., Sethakaset, U., Tan, P. H., Li, Z. & Zheng, J. (2012). A joint source-channel decoder for h. 264 sps and pps headers. *Wireless Personal Multimedia Communications (WPMC), 2012 15th International Symposium on*, pp. 443–447.

Zabihi, S. M., Ghanei-Yakhdan, H. & Mehrshad, N. (2017). Adaptive temporal error concealment method based on the MB behavior estimation in the video. *Computer and Knowledge Engineering (ICCKE), 2017 7th International Conference on*, pp. 193–198.

Zhang, J., Arnold, J. F. & Frater, M. R. (2000). A cell-loss concealment technique for MPEG-2 coded video. *IEEE Transactions on Circuits and Systems for Video Technology*, 10(4), 659–665.

Zhang, Y., Xiang, X., Zhao, D., Ma, S. & Gao, W. (2012). Packet video error concealment with auto regressive model. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(1), 12–27.

Zheng, J. & Chau, L.-P. (2005). Efficient motion vector recovery algorithm for H. 264 based on a polynomial model. *IEEE Transactions on Multimedia*, 7(3), 507–513.

Zhou, J., Yan, B. & Gharavi, H. (2011). Efficient motion vector interpolation for error concealment of H.264/AVC. *IEEE Trans. Broadcast.*, 57(1), 75–80.

Zhou, Z., Dai, M., Zhao, R., Li, B., Zhong, H. & Wen, Y. (2017). Video error concealment scheme based on tensor model. *Multimedia Tools and Applications*, 76(14), 16045–16061.