

Thèse de doctorat
Pour obtenir le grade de Docteur de
l'UNIVERSITE POLYTECHNIQUE HAUTS-DE-FRANCE
l'INSA HAUTS-DE-FRANCE
et de l'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

Discipline: Électronique, spécialité: Télécommunications

Présentée et soutenue par Vivien BOUSSARD

Le 23/02/2021, à Valenciennes

Ecole doctorale : Sciences Pour l'Ingénieur (ED SPI 072)

Equipe de recherche, Laboratoire : Institut d'Electronique, de Micro-Electronique et de
Nanotechnologie - Département d'Opto-Acousto-Electronique (IEMN DOAE - UMR 8520)

**Méthodes et algorithmes de correction d'erreurs basés
sur CRC appliqués aux communications vidéo sans fil
dans des environnements véhiculaires et IoT**

JURY

Président du jury : M. Georges KADDOUM, Professeur, ÉTS Montréal

Rapporteurs : M. Pierre DUHAMEL, Directeur de recherche, L2S Paris-Saclay
Mme. Fabienne NOUVEL, MCF HDR, IETR Rennes

Examineurs : Mme. Maria TROCAN, Professeure, ISEP Paris
M. Carlos VAZQUEZ, Professeur, ÉTS Montréal

Co-directeurs : M. Stéphane COULOMBE, Professeur, ÉTS Montréal
M. François-Xavier COUDOUX, Professeur, UPHF Valenciennes
M. Patrick CORLAY, Professeur, UPHF Valenciennes



This Creative Commons license allows readers to download this work and share it with others as long as the author is credited. The content of this work cannot be modified in any way or used commercially.

REMERCIEMENTS

Je profite de ce manuscrit pour exprimer ma gratitude envers mes parents, merci de savoir toujours être là pour moi. Je pense également à ma famille, à mes amis, et à toutes les personnes qui ont su m'entourer et me soutenir pendant cette période de ma vie. J'étais parfois loin d'eux mais ils sont toujours parvenus à me donner le courage et la force nécessaires pour aller de l'avant et mener ce travail à bien. Un remerciement tout spécial à Justine, pour m'avoir encouragé au quotidien et motivé en tout temps.

Je n'oublie pas mes collègues, que ce soit de ce côté ou de l'autre de l'Atlantique, sans qui certaines journées auraient été plus difficiles et avec qui j'ai eu plaisir à travailler tout au long de ma thèse.

Je tiens tout particulièrement à remercier mes encadrants, les Professeurs Stéphane Coulombe, François-Xavier Coudoux et Patrick Corlay. Ils m'ont permis d'aborder mon travail de recherche avec bienveillance, ont toujours cru en mes capacités à réussir et je peux dire aujourd'hui que je suis heureux d'avoir passé ces trois années sous leur supervision.

Merci à Pierre Duhamel, Professeur Émérite et Directeur de Recherche CNRS, ainsi qu'à Fabienne Nouvel, Maître de Conférence HDR, d'avoir pris le temps d'être rapporteur et rapportrice de mon travail de recherche.

Je remercie également les Professeurs Maria Trocan, Georges Kaddoum et Carlos Vazquez d'avoir accepté de faire partie de mon jury de thèse.

Pour finir, je souhaite remercier le Conseil de Recherches en Sciences Naturelles et en Génie du Canada (CRSNG) ainsi que l'Université Polytechnique Hauts-de-France (UPHF) pour leur soutien financier.

Méthodes et algorithmes de correction d'erreurs basés sur CRC appliqués aux communications vidéos sans fil dans des environnements véhiculaires et IoT

Vivien BOUSSARD

RÉSUMÉ

La transmission de contenus vidéo constitue aujourd'hui l'essentiel des flux de données transmis dans le monde. La qualité de ces contenus est sans cesse en augmentation, du fait du déploiement de réseaux permettant de supporter davantage de trafic à des débits plus élevés, couplé à des stratégies visant à réduire l'information nécessaire à la transmission d'une séquence vidéo, par le biais de techniques de compression vidéo de plus en plus performantes. Néanmoins, la qualité visuelle d'un contenu vidéo peut être dégradée pour l'utilisateur final lorsque surviennent des erreurs lors de la transmission. En effet, un paquet peut être perdu ou corrompu lors de la transmission, dû aux perturbations inhérentes au canal de diffusion. Afin de recouvrer l'information manquante, une retransmission du paquet corrompu peut être envisagée. Cependant, cette option n'est pas toujours valide sous des contraintes de temps réel, comme lors de la diffusion de contenus vidéo en direct, ou afin de ne pas augmenter la charge réseau. Il est alors possible de mettre en œuvre des méthodes de correction d'erreurs au niveau du récepteur pour récupérer les données erronées.

Dans le cadre de cette thèse, nous proposons des méthodes de correction d'erreurs situées au niveau du récepteur, exploitant les codes de détection d'erreurs Cyclic Redundancy Check (CRC) pour la correction d'erreurs. Les méthodes que nous proposons utilisent le syndrome d'un paquet corrompu pour dresser la liste exhaustive des patrons d'erreurs ayant pu produire ce syndrome, pour un nombre d'erreurs inférieur ou égal à celui défini en paramètre d'entrée. Nous proposons plusieurs approches pour parvenir à ce résultat, tout d'abord en utilisant une approche arithmétique, basée sur des opérations logiques effectuées à la volée et ne nécessitant donc pas de stockage mémoire. La deuxième approche propose une table optimisée, dans laquelle sont stockés les calculs répétitifs de l'approche arithmétique de manière efficace et appropriée à la méthode proposée, permettant une exécution bien plus rapide de la correction, au prix d'un stockage mémoire. La validation de la correction s'effectue par un processus en deux étapes, visant à croiser la liste des candidats obtenus avec un autre code de détection d'erreur, la somme de contrôle. L'ultime étape vise la vérification de la syntaxe du flux vidéo encodé en testant sa décodabilité. Nos méthodes ont été testées sur des simulations de transmission de contenus vidéo compressés selon les standards H.264 et HEVC sur des canaux Wi-Fi 802.11p et Bluetooth Low Energy. Ce dernier cas offre les taux de correction les plus significatifs, amenant la vidéo à être reconstruite quasiment intacte même lorsque la qualité du canal de transmission commence à fléchir.

Mots-clés: Transmission sans fil, Wi-Fi, 802.11p, Bluetooth Low Energy, Cyclic Redundancy Check, correction d'erreurs

CRC-based error correction methods and algorithms applied to video communications over vehicular and IoT wireless networks

Vivien BOUSSARD

ABSTRACT

Video content transmission constitute the main category of data transmitted in the world nowadays. The quality of the transmitted content is ever increasing, thanks to the deployment of networks able to support huge traffic loads at high speeds, along with strategies to reduce the amount of data necessary to carry video information, based on more efficient video encoders. However, the quality of the video stream perceived by the end user can be greatly degraded by transmission errors. In fact, a packet can either be corrupted or lost during the transmission due to channel impairments, which result in missing video information that must be recovered. Several strategies exist to recover such information. Retransmission of the damaged packet can be performed. However, this option is not always valid under real time constraints as in video streaming, or to avoid increasing the global network load. To recover missing information, error correction methods can be applied at the receiver's side.

In this thesis, we propose error correction methods at the receiver's side based on the properties of the widely used error detection code Cyclic Redundancy Check (CRC). These methods use the syndrome of a corrupted packet computed at the receiver to produce the exhaustive list of error patterns that could have resulted in such syndrome, containing up to a defined number of errors. We propose different approaches to achieve such error correction. First, we present an arithmetic-based approach which performs logical operations (XORs) on the fly and does not need memory storage to operate. The second approach we propose is an optimized table approach, in which the repetitive computations of the first method are precomputed prior to the communication and stored in efficiently constructed tables. It allows this method to be significantly faster at the cost of memory storage. The error correction validation is performed through a two-step process, which cross-checks the candidate list with another error detection code, the checksum, and then validates the syntax of the encoded packet to test its decodability. We test these new methods with wireless transmission simulations of H.264 and HEVC compressed video content over Wi-Fi 802.11p and Bluetooth Low energy channels. The latter allows the most significant error correction rates and the reconstruction of a near-optimal video even when the channel's quality starts to decrease.

Keywords: Wireless communication, Wi-Fi, 802.11p, Bluetooth Low Energy, Cyclic Redundancy Check, error correction

TABLE OF CONTENTS

	Page
INTRODUCTION	1
CHAPTER 1 LITERATURE REVIEW	5
1.1 Error concealment	5
1.1.1 Temporal Error Concealment	6
1.1.2 Spatial Error Concealment	10
1.1.3 Spatio-temporal approaches	13
1.2 Bit-stream level error correction	17
1.2.1 Standard approach	18
1.2.2 Syntax considerations	20
1.3 Protocol-aided error correction	23
1.3.1 Single protocol approach based on checksums or CRCs	23
1.3.2 Cross-layer management	27
1.4 Wireless communications for mobility and IoT	29
1.4.1 Cellular-based communications	29
1.4.2 WiFi-based technologies	32
1.4.3 Bluetooth-based technologies	35
1.5 Discussion	38
CHAPTER 2 ARTICLES PREAMBLE	41
CHAPTER 3 TABLE-FREE MULTIPLE BIT-ERROR CORRECTION USING THE CRC SYNDROME	47
3.1 Abstract	47
3.2 Introduction	47
3.3 Proposed Method	51
3.3.1 Fundamentals	51
3.3.2 Single-Error Correction	55
3.3.3 Double-error correction	60
3.3.3.1 Problem with straightforward extension of Algorithm 3.1	60
3.3.3.2 Proposed double-error correction approach	61
3.3.4 N-error correction	64
3.4 Simulation and results	71
3.4.1 Correction rate	71
3.4.2 Memory requirements	74
3.4.3 Computational Time Comparison	76
3.4.4 Application to IoT	78
3.4.5 Future Work	81
3.5 Conclusion	81

CHAPTER 4	ENHANCED CRC-BASED CORRECTION OF MULTIPLE ERRORS WITH CANDIDATE VALIDATION	83
4.1	Abstract	83
4.2	Introduction	83
4.3	Related works	87
4.3.1	CRC-based single error correction using tables	87
4.3.2	CRC-based single error correction using arithmetic operations	87
4.3.3	Validation of CRC-based error correction	89
4.3.4	CRC-based multiple error correction	91
4.4	Proposed method	94
4.4.1	List handling: validation of candidates	95
4.4.2	Optimizing the method	98
4.4.2.1	Exploiting syndrome parity	98
4.4.2.2	Reducing memory requirements	101
4.4.3	Proposed approach implementation	103
4.5	Simulation and results	108
4.5.1	Wi-Fi 802.11p	109
4.5.2	Bluetooth Low Energy	112
4.6	Conclusion	119
CHAPTER 5	CRC-BASED CORRECTION OF MULTIPLE ERRORS USING AN OPTIMIZED LOOKUP TABLE	121
5.1	Abstract	121
5.2	Introduction	122
5.3	Related works	125
5.4	Proposed method	128
5.4.1	Single error correction	129
5.4.2	Double error correction	133
5.4.3	N-error correction	140
5.4.4	Cycles of <i>next</i> elements	144
5.4.5	Syndromes with no solution for single error	148
5.5	Performance and complexity	151
5.5.1	Computational complexity	152
5.5.2	Memory requirements	155
5.5.3	Application to the correction of multiple errors	157
5.6	Conclusion and perspectives	159
CONCLUSION AND RECOMMENDATIONS		161
BIBLIOGRAPHY		163

LIST OF TABLES

	Page
Table 1.1	Corresponding values of a_i and b_i (Atzori <i>et al.</i> , 2001). ©2001, IEEE 15
Table 1.2	List of all Bit Error Events (BEE) considered for the method of (Golaghazadeh <i>et al.</i> , 2017a). ©2017, IEEE 24
Table 2.1	Table generated for the correction of double bit errors for a CRC-4 of generator polynomial $g(x) = x^4 + x + 1$ 46
Table 3.1	Memory requirements for storing the lookup tables considering a payload of 1500 bytes for several Cyclic Redundancy Check (CRC) lengths and number of errors considered with implicit and explicit error positions 75
Table 3.2	Error distribution in real environment for Bluetooth Low Energy (BLE) and IEEE 801.15.4 and two packet sizes 79
Table 4.1	Cycle lengths for widely used CRCs: CRC-8-CCITT with $g(x) = x^8 + x^2 + x + 1$, CRC-16-CCITT with $g(x) = x^{16} + x^{12} + x^5 + 1$, CRC-24-BLE and CRC-32. 94
Table 4.2	Average number of candidates before and after Checksum Validation (CV) for different Bluetooth Low Energy channel qualities with $N = 3$ 97
Table 4.3	Addition and multiplication truth tables for finite field GF(2). 99
Table 4.4	Average processing time with (P) and without (NP) considering the syndrome's parity for different BLE channel qualities with $N = 3$ 100
Table 4.5	Average number of errors per corrupted packet for different channel Signal to Noise Ratio (SNR) values in an 802.11p environment. 110
Table 4.6	Average number of error per corrupted packet for different channel conditions. 112
Table 4.7	Average peak signal-to-noise ratio (PSNR) comparison for different sequences and Quantization Parameters (QPs) over different BLE channel conditions for AVC encoded sequences. The tested decoding methods are the following: ①: intact sequence, ②: Joint Model Frame Copy (JM-FC) concealed sequence, ③: Spatio-Temporal Boundary Matching Algorithm (STBMA) concealed sequence, ④: CRC-ECA1 and ⑤: proposed CRC-ECCV. 118

Table 5.1	Example of lookup table for single error correction as proposed in literature (Shukla & Bergmann, 2004), applied to CRC8-CCITT of generator polynomial $g(x) = x^8 + x^2 + x + 1$ considering a 10-bit payload.	126
Table 5.2	Single error position table generated for a CRC-5 of generator polynomial $g(x) = x^5 + x^4 + x^2 + 1$	131
Table 5.3	Table generated for double-bit error correction for a CRC-5 of generator polynomial $g(x) = x^5 + x^4 + x^2 + 1$	141
Table 5.4	Value of syndrome exceptions for commonly used generator polynomials (CRC-8-CCITT, CRC-16-CCITT, CRC-24-BLE and CRC-32-Ethernet)	151
Table 5.5	Comparison of the memory required for the tables in the CRC-ECEXP (Shukla & Bergmann, 2004) and the proposed CRC-ECOT approaches. The packet considered here has a length of 1500 bytes. Note that the arithmetic method is table-free, and thus, its memory requirements remain negligible for any case described in this table	155

LIST OF FIGURES

		Page
Figure 1.1	Relationship between the boundaries of the missing block in the current frame and those of the reference block in the reference frame (Chen <i>et al.</i> , 2008). ©2008, IEEE	6
Figure 1.2	Overview of the proposed approach in (Sankisa <i>et al.</i> , 2018). ©2018, IEEE	9
Figure 1.3	Representation of the eight potential edge orientations used in multi-directional interpolation (Kwok & Sun, 1993). ©1993, IEEE	10
Figure 1.4	Splitting pattern of the block before mesh-based deformation (Atzori <i>et al.</i> , 2001). ©2001, IEEE	14
Figure 1.5	Representation of a standard decoding system (dotted lines) and the joint source channel decoder used in method (Lakovic <i>et al.</i> , 1999). ©1999, IEEE	19
Figure 1.6	Representation of each Bit Error Event (BEE) and their associated CPTs (Golaghazadeh <i>et al.</i> , 2017a). ©2017, IEEE	24
Figure 1.7	Matching table that illustrates the possible BEEs corresponding to each Checksum Pattern Type (CPT) (Golaghazadeh <i>et al.</i> , 2017a). ©2017, IEEE	25
Figure 1.8	Illustration of the header recovery correction based on intra and inter-layer redundancies (Marin <i>et al.</i> , 2010). ©2010, IEEE	28
Figure 1.9	Different 5G scenarios. (a) Standard management, (b) Proximity Services, and (c) Locally managed (Shah <i>et al.</i> , 2018). ©2018, IEEE	30
Figure 1.10	Segmentation of a large transport block into several code blocks with a maximum size of 6144 bits each. Every chunk is protected by a CRC code	31
Figure 1.11	802.11p frame format at the physical (PHY) and medium access control (MAC) layers (Wen <i>et al.</i> , 2009). ©2009, IEEE	33
Figure 1.12	Protocol stack used in Dedicated Short-Range Communication (DSRC). Reprinted by permission from Springer Nature Customer Service Centre GmbH: Springer Nature (Festag, 2015) ©2015, Springer Verlag Wien	34

Figure 1.13	Protocol stack used in ITSG5 (Festag, 2015). Reprinted by permission from Springer Nature Customer Service Centre GmbH: Springer Nature (Festag, 2015) ©2015, Springer Verlag Wien 36
Figure 1.14	Bluetooth Low Energy protocol stack (Gomez <i>et al.</i> , 2012) / CC BY 4.0 36
Figure 1.15	Bluetooth Low Energy packet (Collotta <i>et al.</i> , 2018). ©2018, IEEE 38
Figure 2.1	Example of single error correction on a 10-bit packet with generator polynomial $g(x) = x^4 + x + 1$ and computed syndrome $s(x) = x^3 + x^2$. We can see a single error candidate after the first XOR performed, resulting in a single error at position x^8 42
Figure 2.2	Example of visual reconstruction using frame copy (JM-FC), spatio-temporal boundary matching algorithm (STBMA), CRC based single error correction (CRC1e) and CRC-based triple error correction with checksum validation (CRC3e+CV) on H.264 encoded Mobcal sequence (1280x720) at QP27, over BLE environment with Eb/No=9dB 44
Figure 3.1	Illustration of the binary vector representation: each polynomial $g(x)$ with binary coefficients can be seen as a binary vector \mathbf{g} . Multiplying $g(x)$ by x^n corresponds to a left shift of \mathbf{g} by n positions 52
Figure 3.2	Flowchart of the proposed method's algorithm to correct a single error in the packet. Numbers show the corresponding steps in Algorithm 3.1 58
Figure 3.3	Structure of the initial error vector $\mathbf{e} = \mathbf{0} \oplus \mathbf{s}$ 58
Figure 3.4	Illustration of the single-error search applied to CRC-4-ITU, where $g(x) = x^4 + x + 1$ (yellow cells) with a syndrome $s(x) = x^2 + 1$ (grey cells). In this notation, \sum represents sum(\mathbf{e}). A single-error pattern is found at bit position 8 at step t_3 . In this example: $E_1 = \{8\}$ (i.e., the red cell) 59
Figure 3.5	Illustration of the error range applied to CRC-CCITT-8 ($n = 8$). Canceling Least Significant Bit (LSB) non-zero values by performing an XOR operation with a generator polynomial of width $n + 1$ bits produces an error range of n bits. 62
Figure 3.6	Visual example of the proposed algorithm applied to double-error correction, performed over CRC-4-ITU (yellow cells) protecting 6 data bits with a syndrome $s(x) = x^3 + x^2 + 1$ (grey cells). Each

	forced bit position, represented as a black cell, is tested throughout the process to get the exhaustive list of double-error patterns. In this example there are three such cases at steps t_1 , t_{13} and t_{17} , thus $E_2 = \{(0, 6); (3, 8); (5, 7)\}$ 63	63
Figure 3.7	Illustrative example of the proposed algorithm performed over CRC-8-CCITT (yellow cells) protecting 10 data bits, where $N = 3$ and $s(x) = x^6 + x^4 + x^2 + x + 1$ (grey cells). Forced bit positions are represented as black cells in the vector \mathbf{e} . Three solutions are valid candidates in this example, where \sum , representing $\text{sum}(\mathbf{e})$, equals 3 (shown in red font). Here, $E_3 = \{(0, 1, 18); (1, 6, 16); (2, 8, 18)\}$ 68	68
Figure 3.8	Flowchart of the proposed method algorithm for correcting multiple errors in the packet. Numbers show the corresponding steps in Algorithm 3.2 69	69
Figure 3.9	Single Candidate Ratio (SCR) for a payload length from 0 to 500 bits protected by a CRC-16-CCITT (Association, 2011) of generator polynomial $g(x) = x^{16} + x^{12} + x^5 + 1$, considering up to 4 errors 72	72
Figure 3.10	SCR for a payload length from 0 to 10 000 bits protected by a CRC-24-BLE (SIG, 2013) of generator polynomial $g(x) = x^{24} + x^{10} + x^9 + x^6 + x^4 + x^3 + x + 1$, considering up to 4 errors 73	73
Figure 3.11	Examples of the explicit design of a lookup table containing all triple-error patterns for packet length up to 12000 bits 74	74
Figure 3.12	Evolution of the normalized time ratio to run the proposed method compared to lookup table-based approaches (explicit and implicit) for single and double error in the packet depending on the length of the packet. The normalized time ratio corresponds to $155\mu\text{s}$ in this case 76	76
Figure 3.13	Error correction rate of the proposed method compared to two methods recently proposed in (Tsimbalo <i>et al.</i> , 2016) for different number of errors in the packet 80	80
Figure 4.1	Example illustrating the single error correction method as described in (Boussard <i>et al.</i> , 2020b), using a CRC-4-ITU with generator polynomial $g(x) = x^4 + x + 1$ over 10 data bits, when the computed syndrome at the receiver is $s(x) = x^2 + 1$ 89	89
Figure 4.2	The system proposed in (Golaghazadeh <i>et al.</i> , 2018) allows a validation of the reconstructed bitstream based on a two step validation process. 90	90

Figure 4.3	Illustration of each bit position management for multiple bit error correction using the CRC syndrome. The error vector \mathbf{e} is initialized to zero and its LSB are set as the syndrome \mathbf{s} . For each position, the current value is checked and the procedure depends on the elements in the set of forced position \mathcal{F} . The current position is flipped if it has to be forced and is currently 0 or has to be canceled and is currently set to 1.	91
Figure 4.4	Flowchart illustrating the correction process using both CRC and checksum to reduce the number of candidates in the output list.	96
Figure 4.5	Illustration of the memory reduction induced by considering the error vector as only its error range. The set of forced positions must be stored in order to identify error positions if a candidate is found.	100
Figure 4.6	Comparison of sequence PSNRs for AVC encoded <i>Ice</i> sequence (4CIF 704x576) at QP 32 and different channel SNRs (vehicular Rural Line-of-Sight (LOS) scenario).	111
Figure 4.7	Heatmap representing the value of the number of errors to consider, N , to handle 75% of the error cases for a given channel quality and packet length over a BLE channel. If N is set to the value in the corresponding box for such parameters, then it can successfully correct 75% of the corrupted packets.	113
Figure 4.8	Comparison of sequence PSNRs for AVC encoded <i>Ice</i> and High Efficiency Video Coding (HEVC) encoded <i>Crew</i> sequences (4CIF 704x576) at different channel conditions in a Bluetooth Low Energy environment.	114
Figure 4.9	PSNR evolution through time for H.264 sequence <i>Ice</i> (4CIF) at QP32. The channel used has a SNR per bit (E_b/N_o) of 9 dB.	114
Figure 4.10	Visual comparison of the literature methods to the proposed approach on AVC encoded <i>Ice</i> sequence (4CIF) at QP32 for an E_b/N_o value of 8 dB.	116
Figure 5.1	Flowchart representing the steps to build the table containing the whole list of syndromes along with their associated single error positions	129
Figure 5.2	Flowchart representing the steps to obtain the single error position from a computed syndrome (step 3 in Fig 5.1). In step 1, $\mathbf{0}$ represents the null vector	130

Figure 5.3	Illustration of the next elements generation for the two possible cases. In case 1, the newly forced value is already set to 1 after the first XOR of $g(x)$. In case 2, the newly forced position is 0 after the first XOR, which requires a second XOR of $g(x)$ to force this position to 1. The red boxes represent the new position to force.	134
Figure 5.4	Flowchart representing the steps to generate the table T containing single error position and next element to handle double error correction	137
Figure 5.5	Flowchart representing the steps to follow to generate the next element from the computed syndrome and the generator polynomial (step 6 of Fig. 5.4, illustrated in Fig. 5.3)	138
Figure 5.6	Example of cycles and exceptions when the generator polynomial is $g(x) = x^5 + x^4 + x^2 + 1$	145
Figure 5.7	Representation of the self-loop <i>next</i> elements for the two syndromes exceptions in Figure 5.6	146
Figure 5.8	Single error correction method performed on syndrome $s(x) = x^4 + x + 1$ using a generator polynomial $g(x) = x^5 + x^4 + x^2 + 1$ (even parity)	149
Figure 5.9	Comparison of the average processing time per syndrome for single and double error corrections for state-of-the-art CRC-based error correction (Boussard <i>et al.</i> , 2020a) (CRC-ECA), the proposed optimized table (CRC-ECOT), the implicit table (CRC-ECIMP) and explicit table (Shukla & Bergmann, 2004) (CRC-ECEXP) applied to CRC16-CCITT	154
Figure 5.10	Evolution of the SCR and post checksum validation SCR for CRC-8-CCITT as a function of the packet length	157

LIST OF ALGORITHMS

	Page
Algorithm 3.1 SingleErrorCorrection($\mathbf{s}, \mathbf{g}, n, m$)	57
Algorithm 3.2 N -ErrorPatternsGeneration($\mathbf{s}, \mathbf{g}, n, m, N$)	65
Algorithm 3.3 UpdateForcedPositions(\mathcal{F}, m)	66
Algorithm 3.4 PositionsToVector(\mathcal{F})	67
Algorithm 4.1 SingleErrorCorrection($\mathbf{s}, \mathbf{g}, n, m$)	103
Algorithm 4.2 N -ErrorPatternGeneration($\mathbf{s}, \mathbf{g}, n, m, N, p_R$)	104
Algorithm 4.3 UpdateForcedPositions(\mathcal{F}, m)	105
Algorithm 5.1 SingleErrorCorrection($T[2^n], \mathbf{s}, n, m, cycle$)	133
Algorithm 5.2 TwoErrorCorrection($T[2^n][2], \mathbf{s}, n, m, cycle$)	139
Algorithm 5.3 NErrorCorrection($T[2^n][2], \mathbf{s}, n, m, cycle, N$)	142
Algorithm 5.4 UpdateForcedPositions(\mathcal{F}, m)	143

LIST OF ABBREVIATIONS

ADAS	Advanced Driver Assistance Systems
ADMM	Alternate Direction Method Multiplier
BEE	Bit Error Event
BER	Bit Error Rate
BLE	Bluetooth Low Energy
BMA	Boundary Matching Algorithm
BP	Belief Propagation
BS	Base Station
BSS	Base Service Set
CAVLC	Context-Adaptive Variable Length Coding
CB	Code Block
CFLD	Checksum Filtered List Decoding
CNN	Convolutional Neural Network
CPT	Checksum Pattern Type
CRC	Cyclic Redundancy Check
CV	Checksum Validation
D2D	Device-to-Device
DSRC	Dedicated Short-Range Communication
E-UTRAN	Evolved Universal Terrestrial Radio Access Network

Eb/No	SNR per bit
EPC	Evolved Packet Core
EMVI	Efficient Motion Vector Interpolation
FCS	Frame Check Sequence
FMIS	First_MB_In_Slice
GOP	Group of Pictures
HD	High Definition
HEISI	Hybrid Exemplar Inpainting and Spatial Interpolation
HEVC	High Efficiency Video Coding
IoT	Internet of Things
IoMT	Internet of Medical Things
IP	Internet Protocol
JM-FC	Joint Model Frame Copy
JSCD	Joint Source Channel Decoding
LLR	Log-Likelihood Ratio
LOS	Line-of-Sight
LSB	Least Significant Bit
LSTM	Long Short-Term Memory
LTE	Long Term Evolution
LUT	Look Up Table

MAC	Medium Access Control
MAD	Mean Absolute Difference
MAP	Maximum A Posteriori
MB	Macroblock
MBW	Mesh-Based Warping
ML	Maximum Likelihood
MMEs	Mobility Management Entities
MSB	Most Significant Bit
MV	Motion Vector
MVE	Motion Vector Extrapolation
NALU	Network Abstraction Layer unit
NLOS	Non-Line-of-Sight
OCB	Out of Context of BSS mode
OF	Optical Flow
PHY	Physical
PSDU	Physical layer Service Data Unit
PSNR	Peak Signal-to-Noise Ratio
QoS	Quality of Service
QP	Quantization Parameter
RAN	Radio Access Network

RSSI	Receiver Signal Strength Indicator
RSU	Road Side Unit
RTP	Real-Time Protocol
SCR	Single Candidate Ratio
SE	Syntax Element
SNR	Signal to Noise Ratio
SoC	System on a Chip
STBMA	Spatio-Temporal Boundary Matching Algorithm
TB	Transport Block
TCP	Transmission Control Protocol
TTL	Time to Live
UDP	User Datagram Protocol
UE	User Equipment
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
V2X	Vehicle-to-Any
VLC	Variable Length Coding
VS-IoT	Video Surveillance over Internet of Things
VVC	Versatile Video Coding
WAVE	Wireless Access in Vehicular Environment
WSMP	Wave Short Message Protocol

LIST OF SYMBOLS AND UNITS OF MEASUREMENTS

s	Bit vector associated with the CRC syndrome
s'	Updated syndrome vector
g	Bit vector associated with the generator polynomial
q	Bit vector associated with the quotient of the long division
e	Error vector
e'	Updated error vector
0	Null vector
n	Bit length of the CRC syndrome
m	Bit length of the packet payload
m'	updated remaining payload length
N	Maximum number of consider in multiple error search
k	Local number of error considered
T	Table built from optimized table approach
P_1	Single error position
\mathcal{F}	Set of forced bit positions
F_i	The i^{th} forced position in set \mathcal{F}
E_i	Set of of error patterns containing i errors or less
S	Set of all possible syndromes
S_o	Set of syndromes produce by an odd number of errors

\mathcal{S}_E	Set of syndromes produce by an even number of errors
d_T	Transmitted data
d_R	Received data
p_T	Transmitted packet
p_R	Received packet
m_{v_x}	Horizontal component of a motion vector
m_{v_y}	Vertical component of a motion vector
\oplus	XOR operator between binary vector
$+$	XOR operator between binary polynomials
\ll	Binary left-shift
\gg	Binary right-shift
\leftarrow	Affectation operator

INTRODUCTION

Video quality experience has greatly improved over the past years, thanks to the advent of high definition (HD) video and the emergence of 4K content. As a consequence, video streams tend to embed a larger amount of data. To limit the size of such video streams, video compression solutions were developed in order to represent the same information with fewer bits (Sullivan & Wiegand, 2005; Wiegand *et al.*, 2003; Sullivan *et al.*, 2012). These methods contributed to the transmission of HD video streams over standard channels but also increased the impact of transmission errors on such streams. In fact, as each bit is carrying more information, a single error can have a significant visual impact for the end viewer. Moreover, since compression standards are based on temporal and spatial redundancies, the reconstructed pictures at the decoder are based on the previously decoded ones. Therefore, an error will propagate and spread through time, which will increase its impact on the video quality. There already exists several error correction codes in the lowest layers of the protocol stack, such as Reed Solomon (Wicker & Bhargava, 1999) or convolutional codes (Dholakia, 2012). These codes are able to correct multiple error occurring in a packet. However, some corrupted packets still contain error after passing through these error correction mechanisms. Additional error correction techniques should thus be developed to handle such packets.

Transmission errors can occur due to several factors, such as network congestion and environmental conditions (Li *et al.*, 2004). In addition, some communication schemes are more prone to transmission errors than others. Wireless communications such as vehicular communications or the Internet of Things (IoT), which are the targeted fields of application of our research, are challenging usecases to consider.

Vehicular communications (often referred to as vehicle-to-any (V2X) communications) deployment is a key factor that will contribute to increase road safety over the next few years. Already, existing advanced driver assistance systems (ADAS) such as lane departure warning and blind spot information are limited to the available information from the vehicle's sensors. Cooperative

ADAS will enable neighbouring vehicles to share data in real time, such as their speed and status. In a vehicular context, real-time video streaming can be used by non-safety applications such as infotainment broadcasting as well as safety applications, to reveal hidden objects that can cause a collision (Gomes *et al.*, 2012). The IoT is also widely used today for many applications, from small sensors sending packets of few bytes for monitoring temperature, humidity with a low power consumption to bigger packets for transmitting video content over IoT channels, for video surveillance purposes, for instance. In such environments, increasing reliability and maintaining low latency are critical requirements that can be achieved through error correction.

Generally, reliability can be ensured by the retransmission of the corrupted data, thanks to error detection tools within the protocol stack and packet reception acknowledgement systems, implemented in the widely used Transmission Control Protocol (TCP) (Postel *et al.*, 1980a) for instance. This approach is typically used in data communications, where there are no severe delay constraints to consider. However, these techniques are inefficient in real-time environments. In such cases, an erroneous packet cannot be received in time to be processed due to the delay added by the retransmission. In order to address this loss of information and to avoid error propagation in real-time communications, error management mechanisms such as error correction and concealment (Yao Wang & Qin-Fan Zhu, 1998; Nafaa *et al.*, 2008) must be performed at the receiver. Several approaches and strategies have been developed in order to approximate or correct the corrupted information as accurately as possible.

According to these considerations, our research problematic is to develop new methods to **improve the error correction capacity at the receiver side** in order to ensure a high video quality while maintaining reasonable complexity in an error-prone environment. These constraints induce a restriction on the computational complexity of the solution proposed in order to offer a method having a low processing time. In addition, the proposed approach must recover as much information as possible, ideally all the information, from the damaged packet.

In order to achieve such objectives, we propose in this research work error correction methods based on the CRC syndrome to recover information from mildly corrupted packet. The main contributions of our research work are the following:

- **Arithmetic based CRC error correction:** we developed a method based on the definition of the CRC field computation allowing to output the exhaustive list of error patterns corresponding to the computed syndrome at the receiver, given the generator polynomial used in the transmission. The proposed method works for any packet length and/or generator polynomial used, and does not need any lookup table or significant memory storage. The tests and simulations show that the complexity remains reasonable when considering few errors in the packet. When applied on packets with small payload and strong generator polynomials such as in BLE, the proposed approach outperforms the state-of-the-art methods.
- **Performance enhancement through cross validation:** The limitation of a candidate list approach comes from the number of entries in the output list. Instant correction may be performed when a single candidate remains at the end of the process. However, it was observed that when increasing the number of errors considered, the average candidate list's size increases as well. To address such issue, we propose additional validation steps applied to video communication that allow to drastically reduce the number of candidates in the list while ensuring decodability of the video stream.
- **Optimized table design:** we propose an alternative way to perform our CRC-based error correction, based on the generation of optimized tables. We store in those tables the precomputed results of the operations performed by the arithmetic-based approach. By doing so, we can achieve significant complexity gains as all the process is performed within the table. We organized the tables such that their size does not exponentially increase with the number of errors to consider, as state-of-the-art table would do. Thus, by proposing both table-free and optimized table CRC-based error correction methods, we offer several solutions to

perform error correction, depending on the constraints of the targeted environment, whether they are complexity or memory storage constraints.

These original contributions led to the publication of several of our works. Throughout our research, we published an article in the IEEE Access journal (Boussard *et al.*, 2020a) and submitted two other articles for publication in journals (Boussard *et al.*, 2021a,b). We also participated to the 2020 IEEE International Conference on Image Processing (ICIP) (Boussard *et al.*, 2020b) and filed a patent on both the CRC-based methods described in this thesis (Boussard & Coulombe, 2020).

This manuscript is organized as follows: in chapter 1, we review literature methods for error handling in video communications, from concealment of errors to actual correction of a corrupted packet. We also review the wireless environments commonly used today for vehicular, mobility and IoT communications. The remaining chapters are the journal articles we submitted and/or published during this thesis. Each of these articles are based on one of the main contributions of our research. Chapter 2 is the article presenting the table free CRC-based error correction of multiple errors, using arithmetic operations to generate the list of error patterns having a determined number of errors that correspond to the computed syndrome. In chapter 3, the article focusing on cross validation to increase the correction rate applied to video communications is presented. Chapter 4 is the article containing the description of the optimized table approach that allow faster processing speed thanks to precomputed tables. We conclude this manuscript by presenting an overview of our research results and perspectives.

CHAPTER 1

LITERATURE REVIEW

In this chapter, we review the state-of-the-art error management methods and the vehicular and IoT communication technologies relevant to this work. According to the mechanisms used to recover the damaged data, these error management methods can be classified into three categories:

- the first category is **error concealment** methods, generally applied at the application layer of the protocol stack, that aim at generating the missing video part with spatial and or temporal video information.
- rather than concealing an erroneous packets, the second category of approaches, **bit-stream level error correction**, aims at recovering the originally transmitted packet. Such methods are generally performed at lower layers of the protocol stack.
- The last category we review in this part is **protocol-aided error correction**. Those methods are based on the characteristics of the protocol used to transmit the packet, the different codes and redundancy is conveys, to reconstruct the intact packet. Those approaches are mainly used from Link to Transport layers of the protocol stack.

1.1 Error concealment

Error concealment methods (Shirani *et al.*, 1999; Kung *et al.*, 2006) reduce the visual degradation induced by the loss of information by providing a pixel-domain approximation of the corrupted areas. These approaches are based on the observation of the correctly received video data in the current frame or in the previous frame, depending on the method used. Error concealment cannot guarantee the recovered pixels to have the exact value of the original data but still helps to increase the quality of the damaged video and limit error propagation. The concealment can be performed using temporally (Al-Mualla *et al.*, 1999) and/or spatially (Sun & Kwok, 1995; Liu *et al.*, 2015) neighbouring data to approximate the missing area.

1.1.1 Temporal Error Concealment

Temporal error concealment is mainly based on motion vectors (MVs) recovery of the damaged area (Wu *et al.*, 2008). The purpose of this type of error concealment is to approximate as accurately as possible the missing motion vectors of the current frame to retrieve the damaged information by using the corresponding pixel values in the reference frame.

The Boundary Matching Algorithm (BMA) is one of the most used temporal error concealment method in video. It has been described in 1993 by Lam *et al.* (Lam *et al.*, 1993) and has been extended throughout the years (Chen *et al.*, 2003) to be still used in recent methods performing error concealment. BMA is based on the smoothness property of natural video content. The concept of this method is to reconstruct a missing block by finding its equivalent in a reference frame. Assuming that a good candidate match is found in the reference frame, its pixel values are then duplicated in the current frame to replace the missing block.

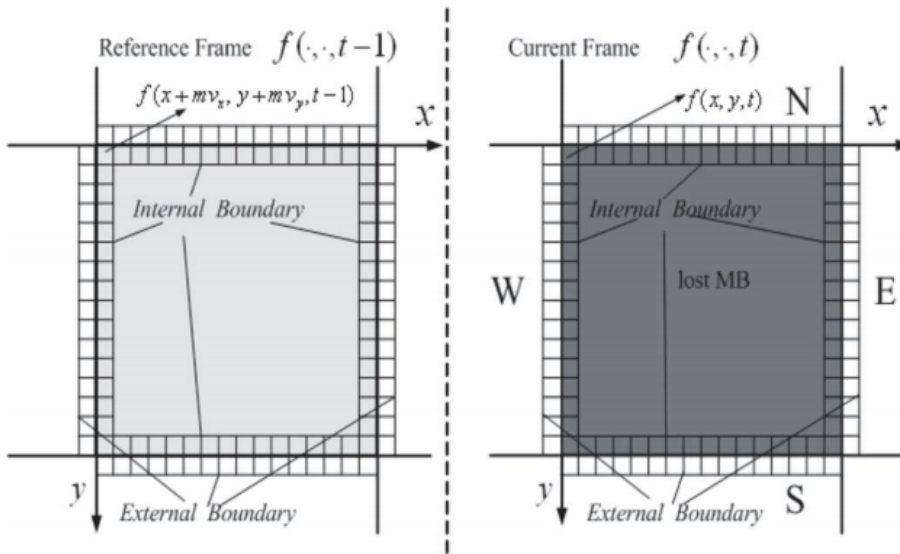


Figure 1.1 Relationship between the boundaries of the missing block in the current frame and those of the reference block in the reference frame (Chen *et al.*, 2008). ©2008, IEEE

The smoothness property of the video content induces a low probability of having brutal variations between adjacent pixels within a picture. Assuming that the neighbouring blocks of the missing block in the current frame are correctly received, the external boundaries of

the missing block are known. The purpose of this method is to find the candidate block in the reference frame which minimizes the absolute difference between the value of its internal boundaries and the external boundaries of the missing block, as it can be seen in Figure 1.1 and denoted D_{sm} in the following equation:

$$\begin{aligned}
 D_{sm} = & \frac{1}{(w_N + w_S + w_W + w_E)M} \\
 & \times \left[w_N \sum_{i=0}^{M-1} |f(x+i, y-1, t) - f(x+mv_x+i, y+mv_y, t-1)| \right. \\
 & + w_S \sum_{i=0}^{M-1} |f(x+i, y+M, t) - f(x+mv_x+i, y+mv_y+M-1, t-1)| \\
 & + w_W \sum_{i=0}^{M-1} |f(x-1, y+i, t) - f(x+mv_x, y+mv_y+i, t-1)| \\
 & \left. + w_E \sum_{i=0}^{M-1} |f(x+M, y+i, t) - f(x+mv_x+M-1, y+mv_y+i, t-1)| \right] \quad (1.1)
 \end{aligned}$$

Where M is the size of the missing block, in pixels, w_X is a weighting factor for each direction, respectively North, South, West and East equal to 0 or 1 depending on the existence of the external boundary in the specified direction. The result will not be distorted by a missing external boundary. $f(x, y, t)$ is the value of the pixel with coordinates (x, y) in the frame t . mv_x and mv_y are the horizontal and vertical components of the tested MV.

Motion Vector Extrapolation (MVE) concealment techniques were developed by Peng et al. in 2002 (Qiang Peng *et al.*, 2002). The concept is to enable the recovery of the entire lost frame with a minimal complexity. The approach is composed of three distinct steps:

- **Extrapolation of the motion vector.** Assuming that the movement within the picture is constant from the last decoded frame to the current, the MVs can be duplicated in the current frame as candidate MVs.
- **Estimation of the overlapped areas** between the damaged block and motion estimated blocks.

- **Choice of the best extrapolated block.** Considers the best candidate and copies its MVs in the current sub-block.

In 2011, Zhou et al. (Zhou *et al.*, 2011) proposed an improvement to this method called Efficient Motion Vector Interpolation (EMVI) which increases the accuracy of MVE by using processes from the Lagrange interpolation (Jinghong Zheng & Lap-Pui Chau, 2003) and performing interpolation on each 4×4 block.

In 2012, Liu et al. proposed an adapted version of the MVE algorithm (Liu *et al.*, 2012) to operate with HEVC (Sullivan *et al.*, 2012), a compression standard which allows a more flexible block management. It was also motivated by the increasing resolution of video content due to the deployment of HD. In fact, MVE is performed on very small block sizes, and thus the computational complexity of this algorithm is strongly related to the resolution of the video content. This method is composed of two steps:

- **Decision of the block size in the lost frame.** The block size is initialized with the biggest block size managed by HEVC. The local distortion within the block is computed, using eight MVs surrounding the block to determine if a further division is needed. This process further splits the resulting blocks until the distortion is acceptable or the minimum block size is attained.
- **Value of the reconstructed MVs.** The value of the restored MV is set according to the average value of all of the neighbouring MVs at the end of the process.

This adaptation of MVE leads to a significant reduction in computational complexity thanks to a more efficient and flexible selection of the block size to consider based on the local MVs distortion.

Recent works on temporal error concealment use deep neural networks in order to predict the content of the missing regions. However, due to the high dimensionality of video content from complex texture information and motion, deep learning approaches cannot take into account the whole set of parameters on the whole sequence. In 2018, the authors of (Sankisa *et al.*, 2018) proposed a deep learning based approach with reasonable complexity. Such method aims at

reconstructing the missing regions of the frame with optical flow (OF) prediction, based on the three previous OFs obtained from an OF generator. This method handles the dimensionality problem on three different aspects:

- Reducing the complexity by minimizing the layers within the proposed machine learning approach.
- Limit the approach to three previously decoded frames. Thus, no bidirectional prediction is performed and the solution does not need an image buffer.
- No full frame: by predicting only a portion of the OF in each frame, corresponding to the damaged portion, the approach reduces the global complexity.

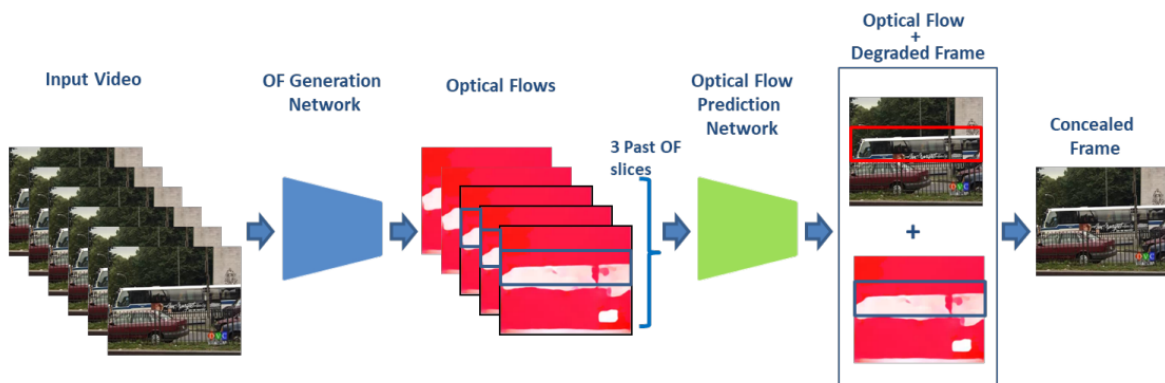


Figure 1.2 Overview of the proposed approach in (Sankisa *et al.*, 2018). ©2018, IEEE

The horizontal and vertical motion fields are predicted separately through two parallel neural networks. The proposed network uses both convolutional neural network (CNN) and Long Short-Term Memory (LSTM) layers. The former is suited for extracting spatial correlation in images and the latter is more efficient on temporal sequences such as videos. The complete concealment process is presented in Figure 1.2, where we can see that the OF of the three past slices is generated and selected to produce the predicted OF in the current frame. Based on the correctly received content of the degraded frame and the predicted OF, the unknown area is concealed.

1.1.2 Spatial Error Concealment

Spatial error concealment takes advantage of the information within the current frame to recover the missing data. This category of error concealment can be based on whole-block (Alka-chouh & Bellanger, 2000) or pixel-wise (Koloda *et al.*, 2013) recovery processes, respectively known as parallel and sequential recovery methods. Generally, block-based recovery processes are less computationally complex than pixel-based methods but the latter offers a better accuracy on the recovered values.

Multi-directional interpolation is a block-based method for reconstruction of damaged blocks first defined by Kwok and Sun in 1993 (Kwok & Sun, 1993).

This method is based on the detection of the edges cutting through the missing block. The process is to first analyze the neighbouring blocks of the missing block to estimate the position and orientation of edges traversing the block when they exist. The restoration of the missing values is then based on the interpolation of each candidate edge mixed together. This method offers a set of eight orientation values available for edge reconstruction as shown in Figure 1.3.

To determine if an edge is cutting through the block, this method classifies the missing block using a gradient measure for each pixel. Each gradient orientation value is rounded to the nearest 22.5° to correspond to a valid orientation. A voting mechanism is set over all these measures to determine which edges will be considered in the restoration process. The restoration uses spatial

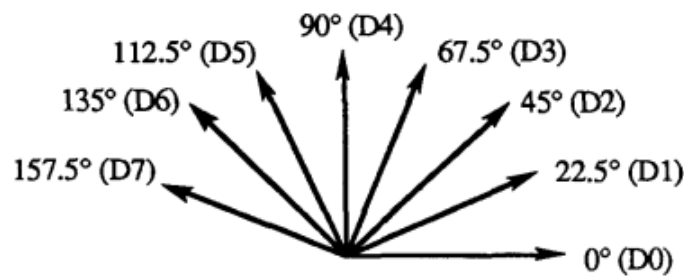


Figure 1.3 Representation of the eight potential edge orientations used in multi-directional interpolation (Kwok & Sun, 1993). ©1993, IEEE

interpolation and is conducted over all the candidate edges. For each candidate, interpolations are performed along the corresponding orientation.

A method to recover the missing block sample values sequentially was proposed in 2002 by Li and Orchard (Xin Li & Orchard, 2002). This method uses Orientation Adaptive Interpolation (OAI) to recover the missing values. The main principle of this method is to recover each sample value successively by using a Maximum A Posteriori (MAP) process. The purpose of MAP estimation is to find the set of parameter values which will maximize the following equation, considering (X_1, \dots, X_K) a set of K unknown values and (Y_1, \dots, Y_L) a set of L known neighbouring values:

$$\text{MAP}_k = p_{\text{MAX}}(X_k | X_1, \dots, X_{k-1}, Y_1, \dots, Y_L) \quad (1.2)$$

Each value can thus be interpolated according to the following assumptions:

- The part of the image processed can be seen as a stationary Gaussian process. It will help to compute the MAP estimation as the probability distribution function can be characterized by its covariance.
- The image source satisfies the N th-order Markov property. It will reduce the size of the set of values considered at each step of the MAP estimation, as taking into account the N previous values is equivalent to considering the whole set of previous values.

In order to reduce the error propagation due to a specific scanning order of the interpolation process, this approach proposes to conduct the process on each of the eight available scanning orientation to get the average value of each sample. This step is called the Linear Merge Strategy. Each orientation is associated to a weighting factor to increase or reduce its importance in the averaging process. The weighting factors are set according to the following assumptions:

- The spatial correlation between two sample values decreases according to their distance. Two spatially neighbouring pixels are more likely to be highly correlated than the two pixels at the top-left and bottom-right corners of a block.

- The weighting factor must take into account the reliability of the pixel. The first pixel to be evaluated does not depend on previously decoded values. The error probability increases with the position of the pixel in the scanning order. The first values must be assigned with a higher weighting factor.

Most recent works propose to improve the traditional multi-directional interpolation, as the method proposed by Byongsu et al. in 2019 (Byongsu *et al.*, 2019). It is based on the original method (Kwok & Sun, 1993). The magnitude and direction of the gradient of each pixel within the missing block is computed and used to determine how significantly every edge direction is cutting through the block. The most significant edges are then selected with an adaptive threshold based on the content of neighbouring areas (edge magnitude in a given direction in the correctly received neighbouring data). While reconstructed pixel values are traditionally weighted through a similarity measure between two boundary pixels, the proposed approach uses a higher weighting factor for boundary pixels having the same direction as interpolation direction along the computed significant edges. The computing of the approximate value p_k^* of each pixel along the k^{th} edge is performed as:

$$p_k^* = \frac{d_2 \times p_1 + d_1 \times p_2}{d_1 + d_2} \quad (1.3)$$

where p_1 and p_2 represent the boundary pixels in the k^{th} edge direction, d_1 and d_2 represent the distance from the current pixel to the boundary pixel p_1 and p_2 , respectively.

The weighting process is performed through the following steps:

- First, the gradient of five neighbouring pixels of p_1 and p_2 , is calculated.
- Then, for each significant edge k , the sums of magnitude of gradients with the same quantization direction level as the edge k , denoted g_{k_1} and g_{k_2} are computed.
- The weighting factor is computed as the ratio between the gradient magnitude of the current edge k over all the significant edges in the set of significant edge E_T :

$$w_{g_k} = \frac{(G_k + g_k)}{\sum_{j=1}^{|E_T|} (G_j + g_j)} \quad (1.4)$$

where $|E_T|$ is the number of significant edges and G_k is the gradient magnitude of the edge k in E_T .

- The final step is the reconstruction of each pixel using the approximated pixel value and the weighting factor:

$$p^* = \sum_{k=1}^{|E_T|} w_{g_k} \times p_k^* \quad (1.5)$$

Thanks to such weighting process, this approach allows to slightly improve the performance of traditional multi-directional interpolation on all the videos tested in their paper. However, such approach is designed to operate on isolated lost MBs, i.e. small unknown regions. Moreover, as it is based only on spatial correlation, the temporal consistency can not be ensured. Thus, methods combining spatial and temporal error concealment and able to operate on larger corrupted regions can offer better reconstruction.

1.1.3 Spatio-temporal approaches

Some methods of error concealment take advantage of both previously discussed approaches to proceed with a spatio-temporal error concealment method.

Most part of these methods consist in the adaptation of a temporal and a spatial error concealment method so that they can be used together and increase the accuracy of the concealment. In the most relevant hybrid methods, the BMA (Lam *et al.*, 1993) is used as a first step of these processes since it gives an acceptable estimation of the missing motion vectors at a reasonable computational cost.

In 2001, Atzori et al. (Atzori *et al.*, 2001) proposed to enhance the BMA by adding a refining step which consists in a spatial approach based on Mesh Warping. This method was motivated by the impossibility for the BMA to predict correctly complex movements such as rotations or zooms. The concept of this method is to perform a traditional BMA on the damaged block and to refine the result obtained by deforming its content to match the surrounding block by a Mesh-Based Warping (MBW) process. MBW consists in applying a mesh structure to the damaged block with a definite number of nodes. There are two types of nodes: 12 control nodes

(C_i) which are located on the edges of the missing block and 4 internal blocks (P_i) located in the inner part of the missing block. These nodes are linked to get a triangular shaped mesh, as shown in Figure 1.4. Each of these nodes will be displaced by a specific distance and a specific orientation according to a matching algorithm. The purpose of this displacement is to reduce the image distortion and the tiled effect that can be introduced by the BMA since it only considers linear translation movements. To perform the MBW method, the BMA block is processed in four main steps:

- **Estimation of the necessity to perform MBW** by computing the average MVs magnitude among the surrounding blocks. If the average value is less or equal to a fixed threshold, the movement between the previous and the current frames is not judged significant enough to perform MBW.
- **Motion estimation of the control nodes C_i .** The control nodes movement estimation is set by comparing the internal and external boundaries of the BMA-reconstructed block. This comparison is performed to ensure a better edge continuity and also to avoid blocking artifacts due to misplaced BMA reconstruction. When one of the four control nodes located in a corner of the block must be displaced, we can note the appearance of a quadrangular shape, which does not match with the other triangular polygons in the block and could

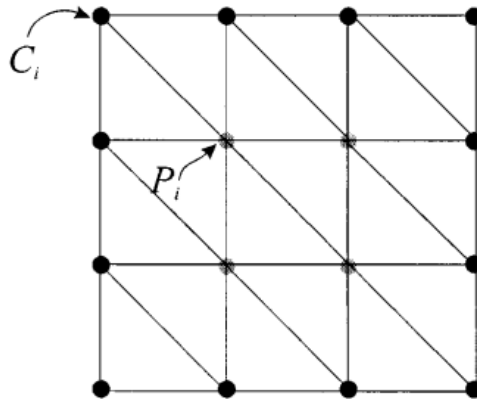


Figure 1.4 Splitting pattern of the block before mesh-based deformation (Atzori *et al.*, 2001). ©2001, IEEE

increase the complexity. In this case, a new control node denoted A_i is added in the corner of the block to restore an all-triangular-shaped mesh.

- **Motion estimation of internal nodes P_i .** The internal nodes cannot be managed as the control nodes, by comparing them to the external boundaries of the reconstructed block. Internal nodes are far from these boundaries and thus are less prone to spatial correlation. According to this, the motion estimation of the internal nodes is processed by a direct interpolation of the previously computed control nodes displacements. Hence, internal nodes positions are set to be the average displacement vector of their neighbour, weighted by their respective distance to each control node.
- **Block Texture Warping.** The final step of the MBW method after mapping the new position of each node is to perform the transform from the original block values to the warped values. A transform from a generic triangular form to another is a complex task to conduct. To avoid such complexity, the authors propose to decompose this step into two transforms, the first going from the original polygonal element to a regular element and the second going from this regular element to the final polygonal element. The transform from a regular element (denoted *master*) and an irregular element (denoted *target*) is much easier to perform. The mapping operation from the *master* to the *target* is the following:

$$w(s, t) = \begin{cases} x = a_1 + a_2s + a_3t \\ y = b_1 + b_2s + b_3t \end{cases} \quad (1.6)$$

Where the parameters a_i and b_i depend on the position of each edge of the triangular shape according to Table 1.1. The inverse mapping is obtained by computing:

Table 1.1 Corresponding values of a_i and b_i (Atzori *et al.*, 2001). ©2001, IEEE

$a_1 = x_3$	$b_1 = y_3$
$a_2 = x_1 - x_3$	$b_2 = y_1 - y_3$
$a_3 = x_2 - x_3$	$b_3 = y_2 - y_3$

$$w^{-1}(x, y) = \begin{cases} s = \frac{1}{J} [(x_2 y_3 - x_3 y_2) + (y_2 - y_3)x + (x_3 - x_2)y] \\ t = \frac{1}{J} [(x_3 y_1 - x_1 y_3) + (y_3 - y_1)x + (x_1 - x_2)y] \end{cases}$$

with

$$J = x_1 y_2 + x_2 y_3 + x_3 y_1 - y_1 x_2 - y_2 x_3 - y_3 x_1 \quad (1.7)$$

To finally retrieve the original value of each pixel (x^0, y^0) in the block, these two operations are performed sequentially on the deformed values (x^d, y^d) :

$$(x^0, y^0) = w(w^{-1}(x^d, y^d)) \quad (1.8)$$

This method takes advantage of BMA to reconstruct in a fast way the missing blocks when only few movements occurred between the previous and the current frames. In addition to this well-known algorithm, the process adds a MBW which can refine and even correct the BMA reconstruction, by managing several new object movements such as zoom or rotation instead of only considering linear translation, which is the case of standard BMA. This method is adapted to real time as it does not add significant computational complexity to the whole process. However, low-delay constraints can lead to erroneous restorations as it means a smaller node vector and search window.

More recently, warping, also denoted registration, is still performed in newly proposed methods. In 2020, a bi-sequential concealment method using adaptive homography based registration has been proposed (Chung & Yim, 2020). This method is performed through four steps on the group of frames hit by errors:

1. Temporal error concealment is sequentially performed in forward direction. Such method combines two kinds of warping models: the image-based (global) and the patch-based (local) models. While global models use the information from the whole frame, local models only use the neighbouring area of the unknown region. For each model, feature points are extracted from the reference and target frames. The homography matrices and warped image/patch of each model is generated. The comparison of the two models is

performed through calculation of the mean absolute difference (MAD). The model with the least MAD is selected. If such MAD value is less than a determined threshold, the missing region is reconstructed using the winning model warped patch. Else, temporal error correction is not performed in this region. If multiple regions are present within a frame, the process is repeated. The reconstructed image is then used as a source for the following frames.

2. When all the frame in the group of pictures (GOP) have been processed, the last frame is first checked for remaining missing regions. If any remain, these regions are reconstructed using spatial error concealment Hybrid Exemplar Inpainting and Spatial Interpolation (HEISI) (Chung & Yim, 2014).
3. The temporal error concealment is then performed backward, from the last to the first frame of the GOP. The same process as step 1 is conducted, but this time the reference frame is the frame following the target frame.
4. At each frame, if the winning MAD of an unknown region is still less than a threshold, it is reconstructed using spatial error concealment HEISI method.

This method is able to handle larger lost regions than standard error concealment methods, which are traditionally performed on small unknown regions, typically corresponding to missing MBs. State-of-the-art methods to recover larger areas are inpainting methods, but require more computations and do not use temporal information. The homography-based method proposed in (Chung & Yim, 2020) is a reasonable tradeoff to handle relatively large unknown regions while maintaining low complexity. By considering both global and local registration models, such approach is more efficient to reconstruct complex and less natural movements in the scene.

1.2 Bit-stream level error correction

Rather than discarding and concealing damaged chunks of a video stream, some methods propose error correction tools to recover a damaged received packet based on its bit-stream information. These methods make use of both channel and source decoding and are often called Joint Source Channel Decoding (JSCD) in the literature. The purpose of such methods is to use jointly the

knowledge of the channel, such as noise estimation and the knowledge of the source, which can be based on the symbol alphabet used and also syntax and semantic checks. These methods can be categorized according to their algorithm characteristics, such as Constrained JSCD or Iterative-JSCD but are often categorized by their outputs. In such cases, the methods are called List Decoding when they produce a list of candidates to validate as their output.

1.2.1 Standard approach

In 1999, Lakovic et al. proposed a joint method to decode video streams transmitted over noisy channels (Lakovic *et al.*, 1999). This method assumes that the theoretical optimality of two-stage systems demonstrated by Shannon (Shannon, 2001) achieves optimal coding performance for sufficiently large block lengths. In practical situations, we generally face short-length blocks and thus it can be less computationally complex to perform a Joint Source-Channel Decoding approach. In 2005, Wang and Yu (Yue Wang & Songyu Yu, 2005) proposed to adapt this method to the decoding of motion vectors in H.264 (Suhring, 2015) coded video streams since the error correction scheme proposed in the reference H.264 codec is provided to add robustness to the stream but is not built to correct errors. As shown in Figure 1.5, a standard decoding system is constituted of a convolutional decoder using the Viterbi algorithm (Forney, 1973) which feeds a Huffman decoder with a sequence of hard decision bits to produce the sequence of decoded symbols. The basic approach to JSCD is to perform both of these processes simultaneously, which is illustrated here by integrating an implicit soft Huffman decoding into the convolutional decoding step. Let us consider a source alphabet A such as $A = \{a_0, a_1, \dots, a_N\}$ and a source coder alphabet B such as $B = \{b_0, b_1, \dots, b_M\}$. The initial information sequence W_i is composed of L elements from A : $W_i = (w_{i1}, w_{i2}, \dots, w_{iL}), w_{ik} \in A$. This information sequence is then encoded with a variable length encoder to generate X_i composed of L elements from B : $X_i = (x_{i1}, x_{i2}, \dots, x_{iL}), x_{ik} \in B$. This sequence then passes through the convolutional encoder to produce Y_i , the sequence to be transmitted through the channel. The received sequence is denoted \widehat{Y}_i and is corrupted by noise. The purpose of the decoder is to find Y_b , the most probable sequence that has been sent over all Y_i possible to retrieve the original data W_i after

the decoding process. To do so, a common and convenient method consists in using a trellis representation of the information at the decoder's side.

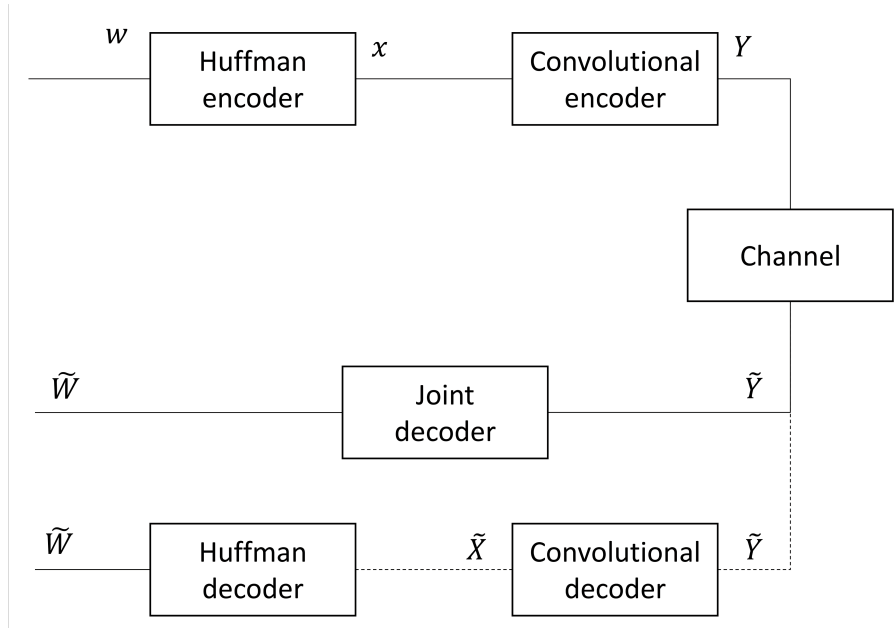


Figure 1.5 Representation of a standard decoding system (dotted lines) and the joint source channel decoder used in method (Lakovic *et al.*, 1999). ©1999, IEEE

Trellis representation is widely used in JSCD to consider and sort a large number of possible paths, each representing a candidate sequence. In standard trellis representations, state-to-state transitions are associated with symbols from the source coder alphabet B , which is usually a binary alphabet. Hence, each state-to-state transition is operated over $C_{m',m}^{b_l}$, which is the sequence corresponding to the transition from state $S_{m'}$ to state S_m for the input b_l . However, the method proposed here uses a codeword approach to the trellis representation, which means that if a sequence composed of symbols from B represents a Huffman-coded symbol a_l , such as $a_l = (b_{l,1}, b_{l,2}, \dots, b_{l,L})$, then the state-to-state transition is operated over $C_{m',m}^{a_l}$, which is the sequence corresponding to the transition from state $S_{m'}$ to state S_m for the input codeword a_l .

1.2.2 Syntax considerations

To reduce the complexity of the MAP estimation, trellis optimization techniques have been proposed (Moonseo Park & Miller, 2000; Ming Jia *et al.*, 2005), based on approximated search algorithms, such as the M-Algorithm (Anderson & Lin, 1986) and the Stack algorithm (Jelinek, 1969). Other approaches brought more accuracy in the candidate's generation step by taking into account the specific syntax of the video compression standard used to reduce the list of candidates by providing only valid sequences to the decoder. These methods can be classified into two categories: the syntax checking methods (Sabeva *et al.*, 2006; Levine *et al.*, 2010; Nguyen *et al.*, 2010), which use a syntax verification at the end of the process to ensure the validity of the candidate sequence and the integrated syntax methods (Nguyen & Duhamel, 2004, 2009; Farrugia & Debono, 2011), which take into account the syntax within the candidate generation process to reduce the complexity by restricting the search to the set of feasible sequences.

A semantic approach to error correction was introduced by Nguyen and Duhamel in 2004 (Nguyen & Duhamel, 2004) and adapted to radio-mobile channels in 2009 (Nguyen & Duhamel, 2009) which integrates the semantic check into the main process. The method proposes to use the inherent redundancy within the source stream by using the Variable Length Coding (VLC) structure of the sequence. This approach consists in finding the most likely candidate among the set of all feasible sequences. A feasible sequence is a semantically compliant sequence of symbols with a bit-length equal to the length of the received sequence. This method is iterative and thus updates, at each iteration, the likelihood estimation of each bit element. The survivor path is selected thanks to a conventional Viterbi metric. Some constraints are established to validate or invalidate each candidate:

- Only candidates of same bit length λ and same number of DCT coefficients r are considered
- Only one highest likelihood survivor is selected, the others are discarded.
- Reliability information is stored into two vectors, which are computed from the probabilities of all associated paths, including the discarded ones.

- The two vectors associated with the final survivor of length N provide the value of the marginal probabilities $P(x_i = \pm 1 \mid y)$.

The semantic constraints operations of this method are complex but the associated cost is offset by the resulting reduced number of nodes in the search algorithm.

In 2011, Farrugia and Debono (Farrugia & Debono, 2011) introduced an error correction method which employs a list decoder strategy to recover the most feasible bit stream with highest likelihood. This method uses soft information in addition to source constraints to minimize the computational complexity of the process. A contextual module is added to keep track of changes in the VLC table. The error control mechanism of this method is based on residual source redundancy. Since the purpose of video compression is to remove redundancy from the original data to reduce the amount of data needed to transmit a video sequence, in an ideal world the residual redundancy would not exist. However, in practice, the redundancy is not completely removed and the decoder can exploit it to recover missing information. There are some fields that one can check to ensure that a received bit stream contains a valid slice:

- The length of the VLC codeword is equal to N bits;
- The number of decoded macroblocks is as expected;
- Each symbol must comply to the syntax rules specified by the compression standard, which is the H.264 standard in this thesis.

If each of these conditions is met, the received sequence is considered as feasible.

Another source of redundancy that can be exploited was added during the entropy coding. The Context-Adaptive Variable Length Coding (CAVLC) used by the H.264 standard encodes quantized coefficients according to a VLC table. Since this is a context-adaptive coding, the VLC tables are switched depending on the value of previously decoded syntax elements. This method proposes to identify which table will be used next according to the received sequence to take advantage of this information. This is the role of the contextual module introduced here.

A trellis-based representation of VLC symbols is used to generate the list of candidates. This method keeps a low computational cost thanks to a non-iterative structure while offering good performance when the number of considered states at each step is small (denoted by parameter M). In their simulations, the authors demonstrated that with M set to 5, this algorithm can outperform the modified Viterbi algorithm, offering a significant PSNR gain of 1.2 dB with 14 times fewer arithmetic operations.

A more recent approach was proposed in 2013 by Caron and Coulombe (Caron & Coulombe, 2013) and uses soft output maximum likelihood decoding to estimate the probability of a sequence of codewords based on the probability of specific syntax elements. This method is applied to the H.264 standard but can easily be adapted to any other standards. The probability of each Syntax Element (SE) is evaluated by using the properties and constraints associated to them. This specific step is what enables this method to work with any standard. The only modification to conduct is to adapt each SE to its specific constraints. In the H.264 standard, some syntax elements are constant for the entire stream, such as `pic_parameter_set_id`, which are thus easy to retrieve.

For non-constant syntax elements, the probability is adapted to each syntax element, as the example of the field. `First_MB_In_Slice` (FMIS) illustrates it:

- **FMIS** : this syntax element is a part of the slice header. Its value corresponds to the index of the first macroblock (MB) in the current Network Abstraction Layer unit (NALU). At the start of a new picture, $\text{FMIS} = 0$. Hence, the value of consecutive FMIS fields is separated by the number of MBs within the corresponding NALU. However, the number of MBs carried by each NALU is not constant, so we cannot predict exactly its value. In this paper, the authors assumed that this number follows a normal distribution, thus conducting to the following equation:

$$P(\text{FMIS} = \hat{c}_i \mid \Omega_{c,i}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(\hat{c}_i - (\text{fmis}_{\text{Prev}} + \mu))^2}{2\sigma^2}\right) \quad (1.9)$$

where $\Omega_{c,i}$ represents the set of variables which hold the context necessary to decode the i -th syntax element, $\text{fmis}_{\text{Prev}}$ is the previously observed value of FMIS, and where the average

number of MBs μ and the standard deviation σ are obtained by analyzing the difference between the FMIS values in consecutive slices.

The same process is conducted over each syntax element available in the coding standard and results in a manageable estimation of the most likely sequence to recover. Since the whole process is based on correct semantic, the resulting sequence will always be decoder-compliant. On the other hand, it can happen that a valid codeword is too unlikely to be considered as a valid candidate. In such cases, an early termination process takes place to stop the correction process.

1.3 Protocol-aided error correction

Video transmission over wired (Wenger, 2003) and wireless (Stockhammer *et al.*, 2003) IP networks regroups a large set of applications, such as digital broadcast television, conversational applications, video downloads or IP-based streaming. The use of such networks offers the possibility to take advantage of their layered protocol management to retrieve missing information. Some works are conducted on a specific layer, such as Transport Layer or Application Layer and other works developed methods to use jointly the redundancies of each layer of the protocol stack to produce a cross-layer approach.

1.3.1 Single protocol approach based on checksums or CRCs

An approach to error correction using the transport layer protocol information was proposed by Golagazadeh *et al.* in 2017 (Golagazadeh *et al.*, 2017a). This method is based on the User Datagram Protocol (UDP) (Postel *et al.*, 1980b) checksum calculation (Braden *et al.*, 1989) and generates a list of candidates when an error is detected by the checksum.

To do so, the authors conducted an analysis on the erroneous patterns according to every possible scenario. To reduce the complexity of the method, this analysis assumes that the maximum number of errors that can possibly occur is two. To justify this assumption, the authors mention that in several practical situations, the channel residual bit error rate ρ is very small ($\rho \leq 10^{-5}$) and thus the probability of having more than two errors occurring is negligible. Five scenarios

of two errors or less, denoted BEEs, have been identified as shown in Table 1.2. The purpose of this method is to identify the corresponding erroneous checksum patterns at the receiver's side. Hence, for each BEE, all the possible erroneous patterns and their number have been taken into account to match BEEs and Checksum Pattern Types (CPTs). At the end of this analysis process, four different CPTs have been identified as shown on Figure 1.6. Since multiple BEEs can correspond to the same CPT as the matching table illustrates in Figure 1.7, the Checksum Filtered List Decoding (CFLD) method proposes to calculate the probability of appearance of each BEE for an observed CPT.

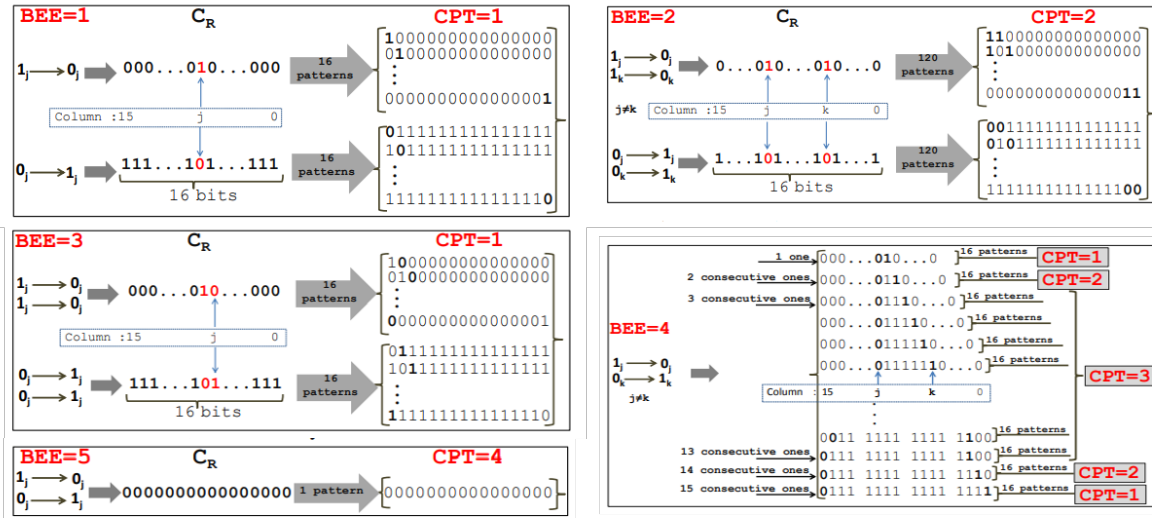


Figure 1.6 Representation of each BEE and their associated CPTs (Golaghazadeh *et al.*, 2017a). ©2017, IEEE

Table 1.2 List of all Bit Error Events (BEE) considered for the method of (Golaghazadeh *et al.*, 2017a). ©2017, IEEE

BEEs	Definition
BEE = 1	1 bit in error
BEE = 2	2 bits in error, same bit value, different columns
BEE = 3	2 bits in error, same bit value, same column
BEE = 4	2 bits in error, different bit values, different columns
BEE = 5	2 bits in error, different bit values, same column

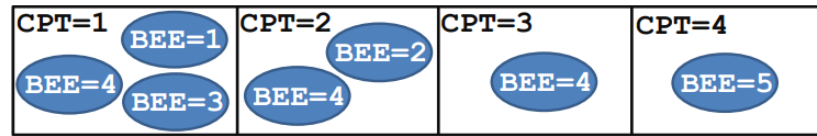


Figure 1.7 Matching table that illustrates the possible BEEs corresponding to each CPT (Golaghazadeh *et al.*, 2017a). ©2017, IEEE

The list decoding itself is composed of four elementary steps, all made possible thanks to the computation step above:

- **Header fixing** : The first step is to check the constant and predictable fields within the UDP and Real-Time Protocol (RTP) (Schulzrinne *et al.*, 1996) headers for errors since an erroneous checksum value does not indicate if the error is located in the header or in the payload of the packet.
- **Syntax Check** : Since the payload data is what contains the valuable information, the method first checks the payload of the packet to determine if it is a valid candidate. The payload is considered as valid and the process stops if it passes two following verifications:
 - The sequence is decodable.
 - The number of macroblocks is correct.
- In any other cases, the correction process takes place, based on the observed CPT to generate an ordered candidates list from the most likely BEEs.
- The final step is the syntax check. Each candidate passes through the syntax verification and the first one to meet the two required syntax verifications is considered as the winning candidate. If no candidate passes these tests, another candidates list is generated based on the second most probable BEE associated with the observed CPT. If there is no BEE left to generate a list, the packet is discarded and an error concealment process takes place.

Other approaches are based on the recovery of corrupted data protected by a CRC (Sobolewski, 2003), an error detecting code which is performed at the physical layer and at the network layer of the protocol stack. In 2004, Shukla et al. (Shukla & Bergmann, 2004) proposed a method to correct single errors occurring within the CRC-protected data. Since the erroneous CRC

pattern generated by single errors are unique, this approach is based on a lookup table which contains all the possible single bit error positions and their associated erroneous CRC pattern. As a consequence, when an erroneous CRC is detected, the algorithm will compare the received value with the different patterns from the table and if it corresponds to an entry of the table, the bit located at the corresponding error position is flipped. Such CRC-based single error correction has been implemented on FPGA in 2017 for CRC-8 and CRC-16.

Improvements of this lookup table based approach have been proposed in 2009 by Babaie et al. (Babaie *et al.*, 2009) in order to correct up to 2-bit errors. These methods are useful to correct a few number of errors but several drawbacks are introduced when one aims at correcting a larger number of errors. In fact, the size of the lookup tables is strongly dependent on the length of the protected data and the number of errors considered. In addition, the cyclic aspect of CRCs induces that several error patterns will produce the same erroneous CRC value, decreasing the reliability of such approaches.

CRC-based error correction has also been proposed in 2017 by Tsimbalo et al. (Tsimbalo *et al.*, 2016) This method uses a maximum likelihood (ML) approach with the log-likelihood ratio (LLR) of each bit (i.e. the confidence index of each bit based on the soft value received) to correct erroneous bit sequence. Their approach aims at finding the most probable sent sequence \mathbf{x} given the observed received sequence \mathbf{r} , the soft information on each bit and subject to compliance with the parity check matrix $\mathbf{H}\mathbf{x}$ generated for the CRC generator polynomial used. The reconstructed sequence can be expressed as:

$$\hat{\mathbf{x}} = \arg \max_{\mathbf{x}: \mathbf{H}\mathbf{x}=0} P(\mathbf{x}|\mathbf{r}) \quad (1.10)$$

In their work, they compare two different approaches to solve this problem, based on two optimization techniques which are Alternate Direction Method Multiplier (ADMM) (Boyd *et al.*, 2011) and Belief Propagation (BP) (Gallager, 1962). It was shown in their paper that ADMM offers a slightly better correction rate. As it is an iterative process, the performance increases with the number of iterations, which requires larger processing times to be able to offer significantly high correction rates. The authors suggest at least 100 iterations to be efficient

and based their tests on 1000 iterations, which yields processing times of 85 ms for packets of a few bytes. The maximum packet length tested in their paper is 39 bytes.

1.3.2 Cross-layer management

A cross-layer approach to JSCD has been proposed in (Duhamel & Kieffer, 2009) and implemented in (Marin, 2009). This approach uses the redundancy within the protocol stack to jointly detect and correct errors. This approach is based on a permeable layer mechanism, which means that the (soft) information must be available at each layer of the protocol stack and from a layer to its upper and lower neighbours.

Since each packet needs to have valid headers at each layer of the protocol stack to be forwarded to the application level, a correction mechanism must be applied to these headers. In fact, this will enable this method to repair many more packets than the previously discussed ones, since it is able to process packets which would have not even have reached the application layer.

The header recovery mechanism is illustrated in Figure 1.8 and applied to the PHY and MAC layers in (Marin *et al.*, 2010). We can see that at each layer of the protocol stack, the header uses information from upper and lower layer fields as well as the previously decoded headers from each layer. This step is conducted to take advantage of the predictable fields of the header. Note that predictable fields are a particular category of header fields and one can use the particularity of each field to select the best approach to recover the corrupted data.

At each layer, each field of the header is thus classified into one of four categories to determine the way it will be corrected:

- **Known fields**, denoted k . These fields are static standardized protocol fields, such as the "Version" field in the IP header or the "Service" field of the PHY header in the 802.11 standard, which is always set to 0x00.
- **Predictable fields**, denoted p . These fields are not constant but can be predicted from the value of a previously decoded packet or another field from another layer. For example, the

sequence number field of the RTP protocol is incremented by 1 at each new transmitted packet.

- **Unknown fields**, denoted u . It is the set of essential fields to the transmission that we cannot predict or evaluate. For example, the server or terminal port number at the UDP layer cannot be predicted but is necessary for the information to be transmitted to the receiver.
- **Other fields**, denoted o . These fields are unknown and not essential to the transmission. In other words, an error in such fields will not affect the communication. Among these fields, we can mention the Time to Live (TTL) field at the IP layer, which gives information on the lifetime of the current datagram. It is thus not necessary to recover those fields, but they can still be protected by a checksum or a CRC and must be taken into account in some particular cases.

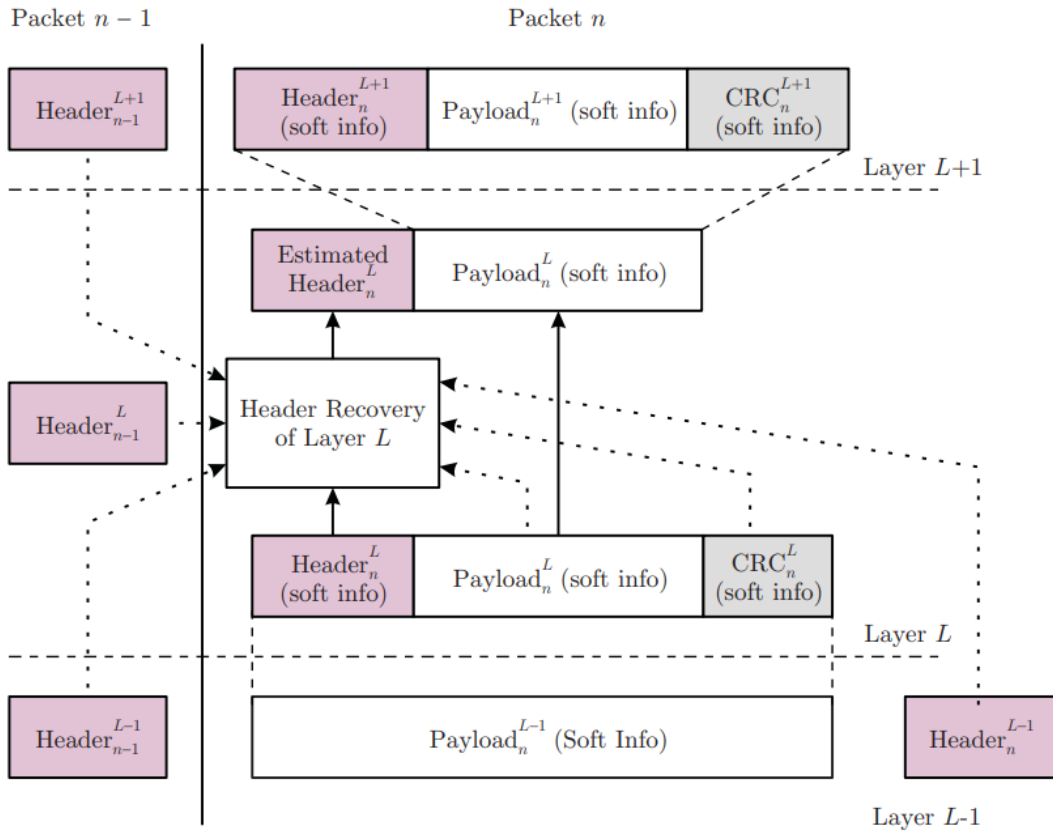


Figure 1.8 Illustration of the header recovery correction based on intra and inter-layer redundancies (Marin *et al.*, 2010). ©2010, IEEE

The strategy adopted by this method is to recover the Unknown fields with a high priority. Concepts of JSCD such as MAP estimation can be used to estimate the most likely u given a received corrupted packet. Considering $Y = [y_k, y_p, y_u, y_o, y_c]$, the actual received packet and R the redundancy information available, one must maximize:

$$\hat{u} = \arg \max_u P(u \mid k, p, R, y_u, y_o, y_c) \quad (1.11)$$

to obtain the header candidate. If the set of values of u is restricted, such as the "Signal" field in the PHY header (which can only take two possible values), the process will have a very low computational complexity. The specificities of each layer, such as the correcting code used and the properties of each field, will help the receiver to take a corrupted sequence to the application layer, where an estimation of the most probable sequence can be conducted.

1.4 Wireless communications for mobility and IoT

Wireless communication technologies such as for vehicular and IoT environments are designed to provide a reliable way to transmit information from vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I) in the case of vehicular communications or towards sensors, machines and devices in IoT. Different communication technologies are currently considered for environments such as V2X and IoT, namely, cellular networks, WiFi-based and Bluetooth-based technologies. In this section, we describe these architectures and their associated protocol stacks. We then discuss the pros and cons of each category.

1.4.1 Cellular-based communications

In 2018, 4G Long Term Evolution (LTE) (Dahlman *et al.*, 2013) is a widely spread cellular technology for mobile telecommunications. According to (ETSI, 2017), the average 4G availability is more than 60% in most countries, exceeding 80% in the United States and Canada. 4G and its successor 5G (Gupta & Jha, 2015) are considered for vehicular communications.

In terms of architecture, cellular networks are based upon three main components:

- **User Equipment (UE)**, which is the mobile device intended to communicate.
- **Evolved Universal Terrestrial Radio Access Network (E-UTRAN)**, which is the Radio Access Network (RAN) composed of antennas, Base Stations (BSs) and links to core network.
- **Evolved Packet Core (EPC)**, which is the core network composed of gateways, servers and Mobility Management Entities (MMEs), which manage to signal messages and subscriber profile information.

Figure 1.9 illustrates the architecture of cellular networks in three scenarios. Scenario (a) is the standard management, where UEs exchange data with their BS that is connected to the core network. Standard management includes the coordination of transmissions to produce fewer collisions. Devices must request transmission permission to the BS and must wait until the BS schedules the transmission. In this architecture, vehicles must be synchronized and communications are not possible out of coverage, such as in tunnels for instance. Moreover, it is not possible to perform a direct transmission between devices without going through the cellular network, which introduces latency. To solve this issue, some device-to-device (D2D)

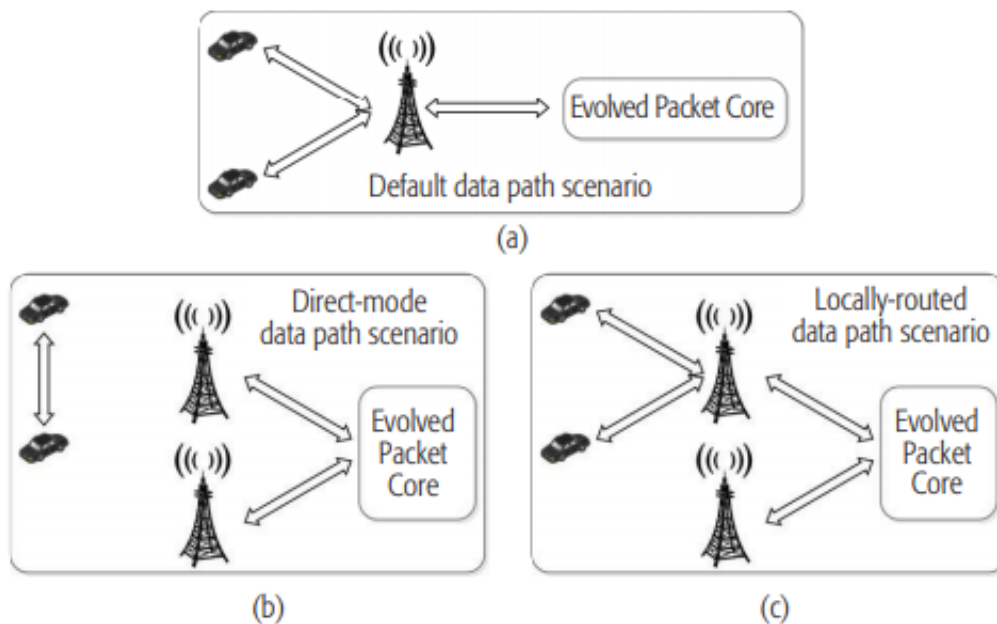


Figure 1.9 Different 5G scenarios. (a) Standard management, (b) Proximity Services, and (c) Locally managed (Shah *et al.*, 2018). ©2018, IEEE

(also referred as Proximity Services: ProSe) solutions have been developed and allow the neighbouring devices to exchange data directly (illustrated in Figure 1.9 (b)). This process is known as *sidelink* communication. D2D cycle is composed of two phases:

- **Proximity detection** to discover the devices in the proximity range and consider the interference constraints;
- **Communication phase** to manage the direct communication between the two end devices.

4G LTE uses a traditional upper layer protocol stack, including IP TCP/UDP (Nshimiyimana *et al.*, 2016). In the lower layers, the packet is protected using a CRC-24 ($\mathbf{g}_A(x)$ in Eq.1.12). If the transport block (TB) exceeds 6144 bits, it is segmented into several code blocks (CBs) and a second CRC-24 ($\mathbf{g}_B(x)$ in Eq.1.12) protection is attached to each CB. In such configuration, each CB is protected individually and the overall transport block is protected as well (ETSI, 2017). The management of large TBs is illustrated on Figure 1.10.

$$\begin{aligned}\mathbf{g}_A(x) &= x^{24} + x^{23} + x^{18} + x^{17} + x^{14} + x^{11} + x^{10} + x^7 + x^6 + x^5 + x^4 + x^3 + x + 1 \\ \mathbf{g}_B(x) &= x^{24} + x^{23} + x^6 + x^5 + x + 1\end{aligned}\quad (1.12)$$

5G cellular network is the upcoming cellular communication technology currently being deployed. It is not designed to change the current 4G architecture but to extend its capabilities. The key features concerning vehicular communications associated with this new generation are the following (Panwar *et al.*, 2016):

- **Integration of existing systems** (3G, 4G, WiFi, ZigBee, Bluetooth, etc.) to allow devices to select the most suitable network to meet the application needs.

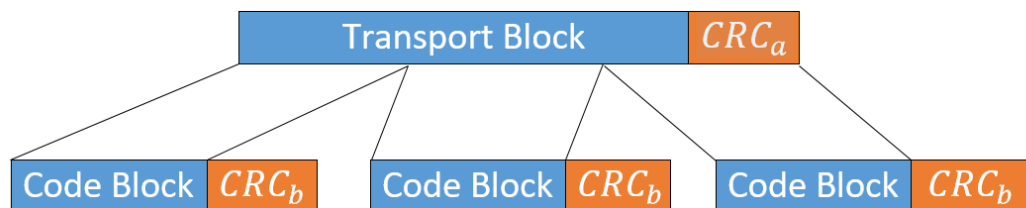


Figure 1.10 Segmentation of a large transport block into several code blocks with a maximum size of 6144 bits each. Every chunk is protected by a CRC code

- **Higher data rate** to allow for a massive number of connected devices with higher amount of data exchanged (real-time video, audio and multimedia content) thanks to small cell deployment and network densification.
- Significantly **reduced latency** thanks to a generalization of D2D communications and edge-caching methods, closer to the end user.

Cellular networks allow for almost full radio coverage and are already deployed around the globe, which gives this technology a significant advantage compared to state-of-the-art WiFi based vehicular technologies. Some concerns regarding 802.11p discussed in the next sub-section are due to the small spectrum capacity compared to the increasing number of connected vehicles and applications targeted by vehicular communications. These concerns are alleviated in cellular networks, which are high capacity compliant. 4G supports high mobility of nodes with speed up to 350 km/h, where 802.11p ensures reliability only up to 60 km/h (Mumtaz *et al.*, 2015). However, the standard management of LTE implies that the transmitted message must wait for the BS to schedule transmission, which involves some real-time issues. The cellular network introduces transmission latency since only Proximity Services involve direct communication between UEs. This lack of support for low-latency applications (mostly safety-related) significantly impacts the interest for using cellular networks for V2X applications. 5G provides solutions to these issues but will take some time to be widely available.

1.4.2 WiFi-based technologies

WiFi-based vehicular communication technologies emerged in the early 2000s with the DSRC frequency band allocated to Intelligent Transport Systems in the United States (Kenney, 2011). This frequency band covers seven 10 MHz wide channels for safety and non-safety communications, in the frequency range from 5.85 GHz to 5.95 GHz. Four service channels are available for both safety and non-safety usage, one control channel is restricted to safety usage and the remaining two channels are for special use, namely "*Critical Safety of Life*" and "*High Power Public Safety*."

In Europe, the intelligent transport systems' frequency band is defined in the European standard ETSI EN 302 663 (ETSI, 2013) and is commonly denoted Cooperative Intelligent Transport Systems (C-ITS) (Sjoberg *et al.*, 2017). The standard defines one control channel and seven fixed service channels, in the frequency ranges from 5.470 GHz to 5.725 GHz and 5.855 GHz to 5.925 GHz. The spectrum is divided into four parts from A to D (ETSI, 2013).

- **ITS-G5A:** ITS road safety and traffic efficiency applications (30MHz band from 5.875 GHz to 5.905 GHz).
- **ITS-G5B:** ITS non-safety applications (20 MHz band from 5.855 GHz to 5.875 GHz).
- **ITS-G5C:** RLAN shared (355 MHz band from 5.470 GHz to 5.725 GHz).
- **ITS-G5D:** Future ITS applications (20 MHz band from 5.905 GHz to 5.925 GHz).

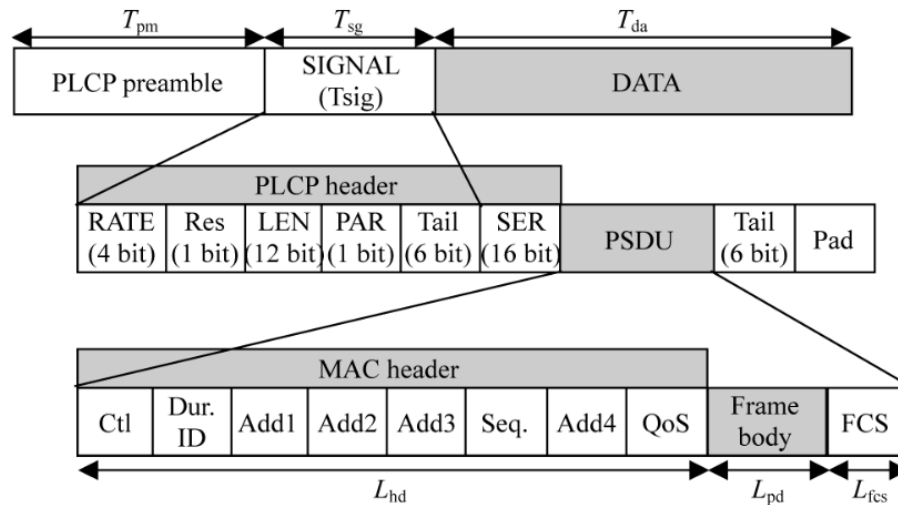


Figure 1.11 802.11p frame format at the PHY and MAC layers (Wen *et al.*, 2009). ©2009, IEEE

Both of these standards are based upon an amendment of 802.11 WiFi dedicated to vehicular communications for PHY and MAC layers. This amendment is 802.11p and appears in the 2012 release of the 802.11 standard. More specifically, PHY and MAC layers of 802.11p are derived from the WiFi standard 802.11a, which operates in the 5 GHz band. However, in 802.11p, the frequency range is shifted from regular WiFi to dedicated DSRC or C-ITS channels. In traditional WiFi standards, each device is connected to a Base Service Set (BSS) which regroups a set of stations. 802.11 devices need to be a member of the BSS to exchange messages.

Joining a BSS implies management procedures like channel scanning and association. This implies a delayed transmission which is not acceptable in critical application such as road safety. The devices must be able to exchange data immediately. To overcome this problem, 802.11p specified an Out of Context of BSS mode (OCB) which allow the devices to directly transmit their messages by disabling the control procedures of the BSS. The frame format of PHY and MAC layers is illustrated in Figure 1.11. The physical layer specifies the rate and the length of the transmitted packet. Other fields are either reserved, service field or padding bits (all set to null fields). The parity bit checks the validity of the frame. The Physical layer Service Data Unit (PSDU) is the payload corresponding to the MAC frame. At MAC layer, the header is composed of the traditional 802.11 MAC header fields: Control, Duration of the frame, MAC addresses of both transmitter and receiver as well as the Quality of Service (QoS). The entire frame (header and frame body) is protected by a Frame Check Sequence (FCS), which is a 4-byte CRC (Sobolewski, 2003) with generator polynomial:

$$g(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 \quad (1.13)$$

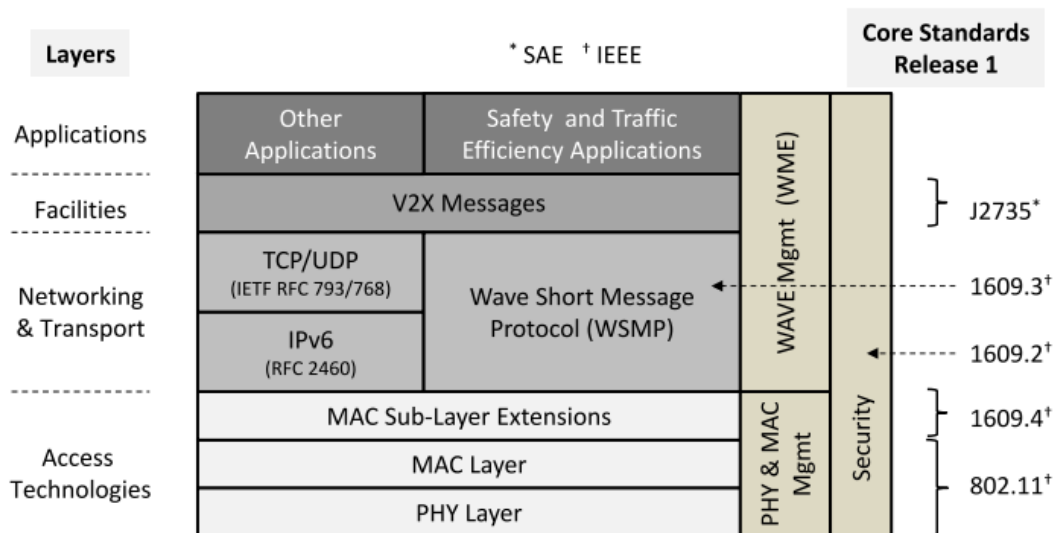


Figure 1.12 Protocol stack used in DSRC. Reprinted by permission from Springer Nature Customer Service Centre GmbH: Springer Nature (Festag, 2015) ©2015, Springer Verlag Wien

DSRC and C-ITS vehicular communication technologies differ at the upper layers of the protocol stack. For non-safety applications, both DSRC and C-ITS use the standard internet protocol stack, namely IP in the network layer and TCP/UDP at the transport layer.

For safety applications the US standard 802.11p is used in addition to IEEE 1609 standards (Association, 2010) as illustrated in Figure 1.12. The combination of these two standards is known as Wireless Access in Vehicular Environment (WAVE). IEEE 1609.4 is used for channel switching and 1609.3 is used for network services, such as Wave Short Message Protocol (WSMP). WSMP can replace IP and TCP/UDP in safety applications. The purpose of this protocol is to reduce the overhead of traditional IP, which includes routing information that is not necessary in vehicular environment, where packets can be sent directly from the source to the destination. The minimum packet overhead of 52 bytes in UDP/IPv6 packet is reduced to 5 bytes with WSMP.

In the European standard, as illustrated in Figure 1.13, the upper protocol stack for safety applications differs from WAVE and is referred to as ITS-G5. In safety applications, C-ITS uses GeoNetworking, which is an ad hoc routing protocol for multi-hop communication. This protocol uses geographical coordinates for addressing and forwarding. IPv6 can also be used over GeoNetworking, denoted as GN6. This protocol has more technical features than DSRC but with an increased protocol complexity and overhead.

WiFi-based vehicular communications are today mature technologies that are planned for almost a decade. Field trials started the in US and Europe and semi-conductor companies are currently designing 802.11p compliant products. However, the deployment of such technology will require Road Side Units (RSUs) working as access points to be installed on the side of the roads. Since there is no pre-installed infrastructure today, a global deployment of WiFi-based vehicular communications would require a significant investment.

1.4.3 Bluetooth-based technologies

Bluetooth technology was first developed in 1994 and is today widely used to pair two devices for entertainment purposes, by connecting audio sources to Bluetooth speakers for instance,

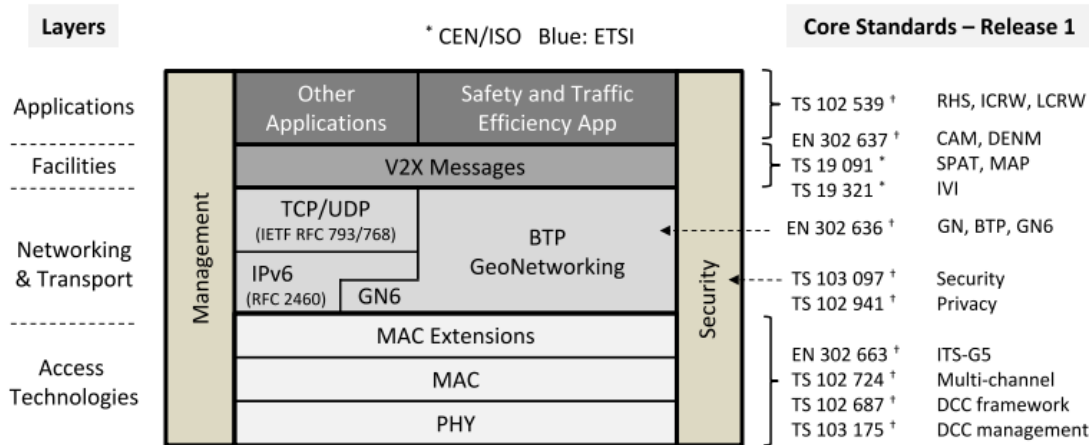


Figure 1.13 Protocol stack used in ITSG5 (Festag, 2015). Reprinted by permission from Springer Nature Customer Service Centre GmbH: Springer Nature (Festag, 2015) ©2015, Springer Verlag Wien

and is a pillar of IoT connectivity by providing a continuous connection between devices and sensors. In order to be efficient in IoT applications, the standard had to reduce its power consumption to be adaptive to battery-powered objects such as wearable devices. To achieve such objective, BLE was proposed in the Bluetooth 4.0 standard, released in 2010 as a low

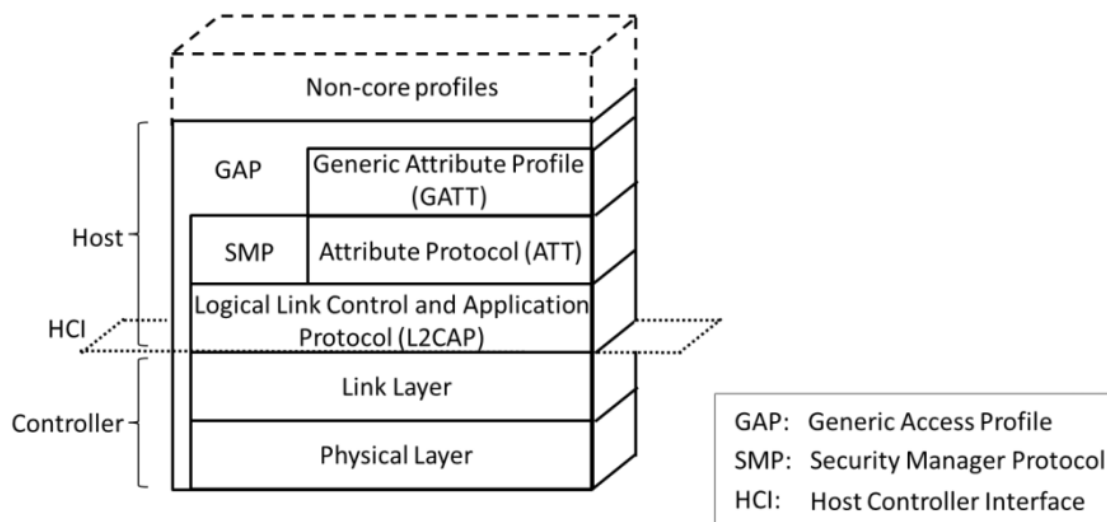


Figure 1.14 Bluetooth Low Energy protocol stack (Gomez *et al.*, 2012) / CC BY 4.0

consumption alternative to traditional Bluetooth, permitting 10 times less energy consumption. The maximum payload size was 31 bytes until the emergence of Bluetooth 4.2 in 2014, which increased such limit to 256 bytes, which helped to significantly decrease the downloading time of large transmitted data. Bluetooth 5 was launched at the end of 2016 and allows to increase the range of BLE from 100 m to 200 m outdoor (20 m to 40 m indoor), along with doubling the maximum theoretical bitrate from 1 Mbps to 2 Mbps. The applications of BLE range from healthcare devices communications, Internet of Medical Things IoMT (IoMT), for collection and analysis of health data through remote monitoring to transportation systems, with traffic management and vehicular communication. In 2020, BLE also plays a significant role in contact tracing in CoViD-19 pandemic as proximity tracing was used by smartphone applications to track the other phones encountered when a positive case is detected (Whaiduzzaman *et al.*, 2020).

Figure 1.14 shows the standard protocol stack of BLE. The standard is defined for physical and link layers. At the physical layer, it operates in the same 2.4 GHz band as traditional Bluetooth but differs by defining 40 channels with 2 MHz channel spacing, while Bluetooth defines 79 channels with 1 MHz channel spacing. Channels are divided into two categories in BLE. There are three advertising channels and 37 data channels. At the link layer, the connection is established between two devices and a 4-byte access code is randomly generated for this specific connection. Each packet will thus be identified as belonging to the connection if its access code matches the connection one.

As illustrated in Figure 1.15, a BLE packet at the link layer is composed of preamble and header fields followed by the payload and CRC protection. The payload is limited to the maximum size of 256 bytes. Each packet is protected with a CRC-24 of the generator polynomial:

$$g(x) = x^{24} + x^{10} + x^9 + x^6 + x^4 + x^3 + x + 1 \quad (1.14)$$

The CRC field protects the entire payload length in such communications.

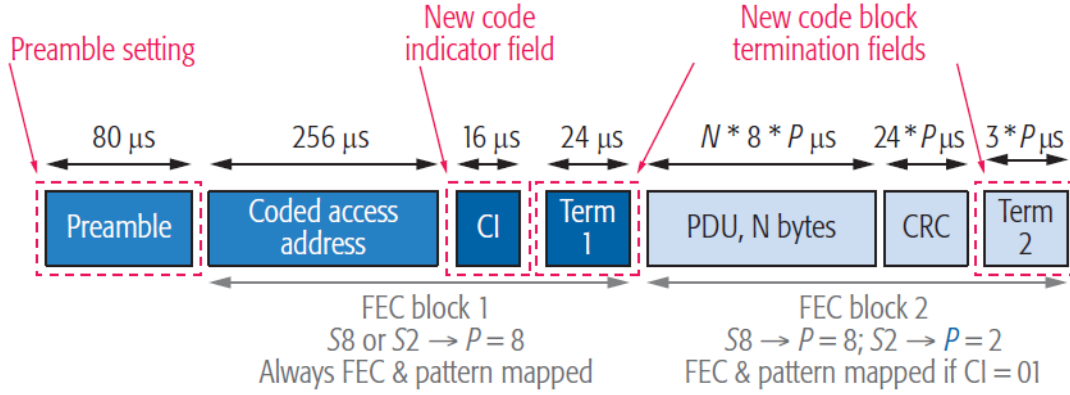


Figure 1.15 Bluetooth Low Energy packet (Collotta *et al.*, 2018). ©2018, IEEE

The BLE standard offers a good alternative to transmit relatively small data packets. Moreover, the recent improvement brought by the emergence of Bluetooth 5 increases the possibilities of such technology by increasing the data rate and packet length for more complex device monitoring and video surveillance applications, for example. The strong protection offered by the CRC-24 on packets with maximum payload 256 bytes can also be an advantage as it offers a strong protection while bringing a small overhead. This solution is thus a good candidate in CRC-based error correction methods.

1.5 Discussion

We saw throughout this literature review that there exists many strategies to handle a corrupted packet in video communications. However, such approaches suffer from limitations, such as error concealment techniques. Indeed, error concealment is able to generate video content in the case of a lost packet, but the systematic discarding of a corrupted packet can lead to loss of useful information. Moreover, the visual quality of the concealed unknown video region is dependent from the content of the video itself, as complex, fast and/or unnatural movements in the scene will lead to less accurate results. Recent error concealment approaches achieve significantly good results thanks to the use of inpainting and neural networks techniques. However, such approaches bring a higher complexity, which results in higher processing times.

Bit-stream level reconstruction methods are independent from the video context, and can perform

error correction on highly corrupted packets. However, such approaches are computationally greedy as they are generally based on iterative search. Moreover, these techniques requires the same processing time for mildly and highly corrupted packets.

In this thesis we present methods to correct packets that we assume mildly corrupted. Discarding corrupted packets that were hit by only few errors seems like a waste of valuable information. By proposing approaches to correct few errors using the CRC syndrome of the corrupted packet, we aim at increasing the correction capability at the receiver while maintaining a tractable complexity. We will see in the remaining of this thesis that some applications are well-suited for that strategy due to their error distribution and thus offer significant gains over the literature methods.

The following chapters are the articles describing in depth the methods and algorithms we propose to achieve such objective.

CHAPTER 2

ARTICLES PREAMBLE

We decided to produce a manuscript using a thesis by articles format, which means that the forthcoming chapters are actually the articles produced from our research works, as they were submitted and/or accepted by their respective journals. In this chapter, we describe the research directions of these articles along with the research problematic they address.

Based on the state-of-the-art approaches for error concealment and error correction described in the literature review chapter, our main research problematic is to increase the error correction capabilities at the receiver side and thus improve the visual quality of video content corrupted during their transmission over error-prone networks. We propose to address this by developing solutions using the CRC codes as error correction codes, which were originally designed for error detection purposes only. Moreover, considering the applications of our work, we must also consider a constraint on the complexity of the proposed solution, so that each corrupted packet can be processed in a reasonable amount of time.

Chapter 3 contains the article published in the journal IEEE Access in August 2020 that describes the first CRC-based error correction approach we proposed. Such method aims at generating the exhaustive list of error patterns containing at most N errors, where $N \geq 1$ is arbitrary. From the definition of the computation of the CRC field, it has been shown that a binary error polynomial $e(x)$ leading to a computed non-null syndrome $s(x)$ can be expressed as:

$$e(x) = q(x).g(x) + s(x) \quad \forall q(x) \in \text{GF}(2^m) \quad (2.1)$$

Where $g(x)$ is the generator polynomial used to compute the CRC field, m is the length of the protected data and $q(x)$ can be any binary polynomial of highest degree less or equal to m . In our research work, we assumed that the corrupted packets we wish to correct are only lightly corrupted. In fact, lightly corrupted packets still contain a huge amount of valuable information to extract by handling only few errors. Thus, we propose algorithms to generate the list of all

CRC-compliant error patterns, i.e. compatible with the computed syndrome, for a determined number of errors.

To handle a single error, the process consists in successively canceling the least significant non-null position by XORing the generator polynomial $g(x)$ at such position (i.e. adding a properly left-shifted version of the generator polynomial). By doing so, we build the polynomial $q(x)$ term after term and avoid considering the most part of the solutions to Eq. 2.1 that would lead to error polynomial $e(x)$ with a large number of non-null positions (i.e., a large number of errors in the packet).

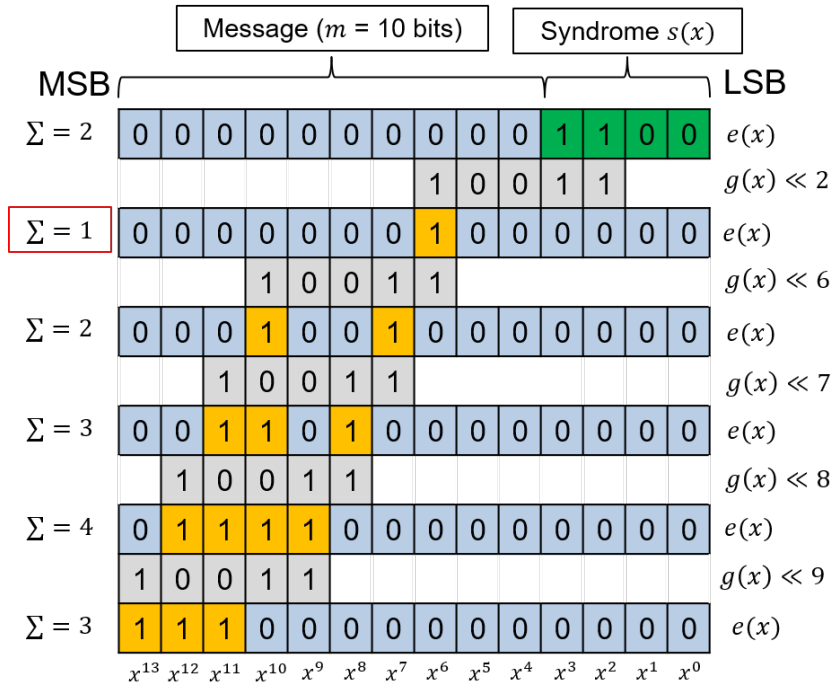


Figure 2.1 Example of single error correction on a 10-bit packet with generator polynomial $g(x) = x^4 + x + 1$ and computed syndrome $s(x) = x^3 + x^2$. We can see a single error candidate after the first XOR performed, resulting in a single error at position x^8

From the example in Figure 2.1, we can see that the range of non-null values at any time during the process is no larger than n position, n being the bit length of the syndrome. To extend such approach to handle more than one error, the method we propose is to force positions to 1 during the process, still by performing XORs with the generator polynomial, to raise the range limitations introduced by the single error approach. By successively forcing $N - 1$ positions in

the error polynomial, we can generate the exhaustive list of error patterns containing N errors of less, leading to the computed syndrome at the receiver.

This method can be used as a standalone process if the approach produces a candidate list with a single entry at the end of the process. By evaluating a proposed Single Candidate Ratio (SCR) metric, we were able to illustrate the percentage of error cases, for a given number of errors, that would result in lists containing a single candidate.

The proposed approach offers significant results on small packets such as those used in Bluetooth Low Energy (BLE) environments. However, we can also observe two main drawbacks of such approach. First, when the considered number of errors N increases, the SCR tends to greatly decrease as the considered packet length increases. Thus, the proposed method will produce lists containing a large number of candidates and introduce ambiguity, as all these candidates are CRC-compliant. Additional validation steps are necessary to reduce the number of candidates to a single one and perform error correction. The second drawback is the computational complexity of this method, which greatly increases with the number of errors considered, as its complexity is $O(m^N)$. In order to handle large packets and error patterns containing several errors, solutions to reduce this complexity should be investigated.

Chapter 4 contains the article submitted to Signal Processing: Image Communication in March 2021, in which we present our solution to address the problem of ambiguity when several candidates are present in the list, applied to video communications over vehicular 802.11p and IoT BLE environments. We propose a cross-layer approach which jointly uses the CRC-based error correction method and error detection tools available in other layers of the protocol stack. We decided to use the User Datagram Protocol (UDP) checksum at the transport layer as a validation step after generating the whole list of CRC-compliant candidates. We compute the checksum of each reconstructed sequence and discard from the list every error pattern leading to an erroneous checksum. We further validate the remaining candidates by trying to decode the corresponding video sequence. If the decoder fails, the candidate is discarded. At the end of this validating process, we note a significant reduction of the number of candidates in the

list, which enables the error correction of packets (i.e. a single candidate remains in the single) for most corrupted packets when setting the number of errors to consider to $N = 3$. This is illustrated in Figure 2.2, where we can observe the reconstruction of a video sequence with four different strategies, namely, error concealment frame copy (JM-FC) and spatio-temporal boundary matching algorithm (STBMA), the proposed CRC-based error correction searching for 1 error (CRC1e) and the combination of CRC error correction searching for 3 errors or less and the checksum validation (CRC3e+CV).

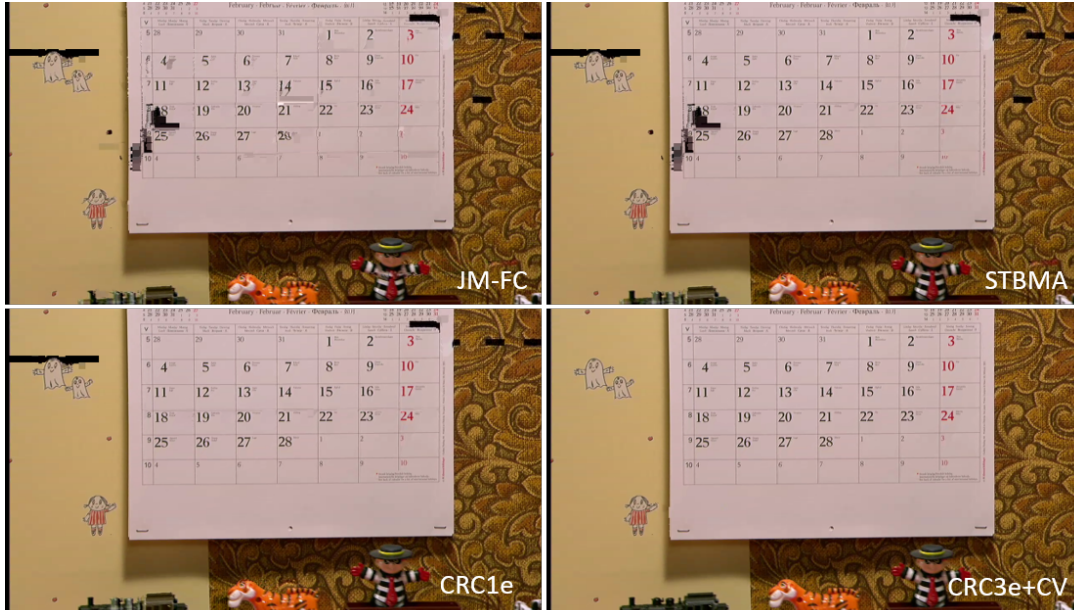


Figure 2.2 Example of visual reconstruction using frame copy (JM-FC), spatio-temporal boundary matching algorithm (STBMA), CRC based single error correction (CRC1e) and CRC-based triple error correction with checksum validation (CRC3e+CV) on H.264 encoded Mobcal sequence (1280x720) at QP27, over BLE environment with $E_b/N_0=9\text{dB}$

This article also illustrates the significant difference in error correction rate depending on the transmission environment selected. In this paper, we compare the performance of our method in both vehicular and IoT environments. The results show that the error distribution in the former environment is not compatible with the hypothesis we made when developing our CRC-based error correction methods, which was that we only had to deal with lightly corrupted packets. In such environment, it was observed that most of the corrupted packets contained

more than 5 errors, regardless of the channel quality, which leads to moderate gains of 0.2 dB over state-of-the-art error concealment methods. On the other hand, we show that when the environment is adequate, we can achieve significant visual gains over literature methods. In IoT environments using BLE, the vast majority of corrupted packets contain fewer than 3 errors when the channel quality is high enough. In such cases, we observe gains from 1.6 dB to 6 dB compared to state-of-the-art error concealment.

Chapter 5 contains the article submitted for publication in January 2021, addressing the computational complexity issue raised by the first method proposed, which is referred as the arithmetic approach. This method is based on the observation that for a given computed syndrome and generator polynomial, the arithmetic approach systematically performs repetitive operations to generate the list of candidates. Thus, such operations can be stored in a table designed to produce the candidate list in a fast way. For the case of a single error, we proposed to build a table containing the position of the single error position for each possible syndrome for a given generator polynomial. By indexing such table according to the value of the syndrome, the single error position can be accessed with a single table call. To handle the correction of multiple errors, the mechanisms introduced in the arithmetic approach, which consist in forcing some bits to 1 during the process, have been applied to the table approach. In our work we defined the next element of a syndrome, which corresponds to the update of the syndrome when going from a forced position the next one. For multiple error management, the tables of both the single error position and the next element for all possible syndromes must be generated and stored prior to the transmission, as illustrated in Table 2.1.

As we observed in our first works, m positions should be forced throughout the process to handle a packet of m bits. Thus, it would need m table calls to generate the list of candidates error patterns for the correction of two errors using such optimized table approach. This approach offers significant gains in terms of processing speed compared to the proposed arithmetic method, from $17\times$ to $3000\times$ faster for single error correction, depending on the packet length, from 36 bits to 2500 bytes, and from $1.5\times$ to $2300\times$ faster for correction of double bit error, for packet lengths of 36 bits and 2500 bytes, respectively. From a practical point of view, it takes an average

Table 2.1 Table generated for the correction of double bit errors for a CRC-4 of generator polynomial $g(x) = x^4 + x + 1$

Index	P_1	$next$	Index	P_1	$next$
0	-1	4	8	3	0
1	0	13	9	14	9
2	1	5	10	9	1
3	4	12	11	7	8
4	2	6	12	6	2
5	8	15	13	13	11
6	5	7	14	11	3
7	10	14	15	12	10

of 642 μ s to generate the candidate list for up to 2 errors for the largest packet size of 2500 bytes, which corresponds to 1550 packets per second, hence having a max bitrate in such case of $2500 \times 8 \times 1550 = 31$ Mbits/s. Given a video frame rate of 60 frames per second, our approach can conduct real-time error correction if each frame is composed of $\lfloor \frac{1550}{60} \rfloor = 25$ packets or less.

To summarize our work, we developed CRC-based methods to correct multiple errors through the generation of the exhaustive list of error patterns corresponding to a given computed CRC syndrome, containing no more than N errors. Throughout our research, we presented two different approaches to generate such list:

- The first one is based on arithmetic operations on the syndrome to generate the list. Such approach is table free and thus does not need memory to be performed.
- The second one is based on optimized tables to generate the list only by navigating within tables.

The complete description and implementation details of each method are presented and discussed in their corresponding chapter.

CHAPTER 3

TABLE-FREE MULTIPLE BIT-ERROR CORRECTION USING THE CRC SYNDROME

Vivien Boussard^{1,2}, Stéphane Coulombe¹, François-Xavier Coudoux², Patrick Corlay²

¹ Département de génie logiciel et des technologies de l'information,
École de technologie supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3

² Univ. Polytechnique Hauts-de-France, CNRS, Univ. Lille, ISEN, Centrale Lille, UMR 8520 - IEMN - Institut d'Électronique de Microélectronique et de Nanotechnologie, DOAE
Département d'Opto-Acousto-Électronique, F-59313 Valenciennes, France

Article published in IEEE Access, vol. 8, pp. 102357-102372, Aug. 2020

3.1 Abstract

In this paper, we propose a novel method for correcting multiple errors in data packets, using the Cyclic Redundancy Check (CRC) syndrome present in low layers of protocol stacks. The proposed method generates the whole list of error patterns, leading to a received syndrome containing up to a given maximum number of errors. Our approach is table-free, is computationally efficient, and can instantly correct erroneous packets when the output list contains a single element. A performance study is conducted, and shows that the proposed approach outperforms existing ones in Bluetooth Low Energy (BLE) as it can correct all single- and double-error patterns as well as most triple-error cases when considering small payloads used in Internet of Things (IoT) applications.

3.2 Introduction

CRC codes constitute a well-known special case of checksum functions, which are typically used for packet error detection in a wide variety of low-layer protocols (Sobolewski, 2003). Their main purpose is to validate the integrity of received packets. If an error is detected by such

codes, the corrupted packet is normally discarded and a data recovery mechanism can be set, as implemented in protocols such as the TCP (Postel *et al.*, 1980a), where reliability is ensured through retransmission of the corrupted data. In order to avoid systematic retransmission, which would lead to an increased amount of data and extra delays within the network, error correction methods have been proposed at the receiver side. In addition, error detection codes such as CRCs and Checksums (Braden *et al.*, 1989) have also been demonstrated to allow error correction (Tsimbalo *et al.*, 2015; Golaghazadeh *et al.*, 2017a; Niu & Chen, 2012; Tsimbalo *et al.*, 2016; Shukla & Bergmann, 2004; Babaie *et al.*, 2009; Liu *et al.*, 2018; Aiswarya & George, 2017). The principle of CRC error detection is based on the computation of a so-called CRC field at the transmitter side. The value of this field is the remainder of the long division of the protected bit sequence, the data, which we will refer to as the *payload*, denoted $d(x)$, by a generator polynomial (a binary polynomial of degree n defined by the protocol used, denoted $g(x)$). The payload is left-shifted by n positions before the division. In (3.1), the remainder is denoted $r(x)$ and $q(x)$ represents the quotient of the long division (Sobolewski, 2003):

$$d(x).x^n = q(x).g(x) + r(x) \Rightarrow \text{CRC} = r(x) \quad (3.1)$$

The computed CRC field $r(x)$ is then appended to the payload and sent to the receiver. The transmitted packet, comprising the payload and its associated remainder, is denoted $p_T(x) = d(x).x^n + r(x)$.

At the receiver, a long division by $g(x)$ is performed on the received packet, denoted $p_R(x)$, in order to check its integrity. An error-free packet (i.e., $p_R(x) = p_T(x)$) is thus a multiple of $g(x)$ and the remainder is zero. In the contrary case, an error will modify $p_T(x)$ and produce a non-zero value as the remainder. The result is called the syndrome of the CRC, denoted $s(x)$. The standard management here consists in automatically discarding a received packet with a non-null syndrome. However, such management leads to a waste of information. In real-time applications such as video conferencing, packet retransmission is unavailable. One would then benefit from extracting as much information as possible from a received corrupted packet. Our

approach is to propose algorithms to attempt to repair such corrupted data using the actual syndrome value.

CRC-based error correction techniques have been explored in previous works, and can be divided into two main categories:

1. **Methods based on estimators (Duhamel & Kieffer, 2009; Tsimbalo *et al.*, 2016; Caron & Coulombe, 2013):** These approaches use statistical estimators, such as the MAP estimator, and aim at finding the most probable binary sequence that has been sent, considering the received erroneous sequence. The CRC is used to check the validity of the MAP sequence or can be part of the estimation process. Such methods can use optimization techniques such as the (ADMM) (Boyd *et al.*, 2011) or BP (Gallager, 1962). These approaches to MAP are costly, and generally use LLR (Duhamel & Kieffer, 2009) and provide information on the confidence of the received bit, expressed as a real value between $-\infty$ and $+\infty$, based on the received soft values. Unfortunately, today's TCP/Internet Protocol (IP) and glsudp/IP protocol stacks are essentially designed to deal with hard values (decoded bits), and consequently, such approaches cannot be implemented in current architectures without great effort.
2. **Lookup table approaches (Niu & Chen, 2012; Shukla & Bergmann, 2004; Liu *et al.*, 2018; Babaie *et al.*, 2009; Aiswarya & George, 2017):** These approaches implement lookup tables prior to the communication, in which each entry contains the syndrome resulting from one (Shukla & Bergmann, 2004) or two (Babaie *et al.*, 2009) errors at specific positions. Upon reception, when a CRC check results in a non-null syndrome, the table is scanned. If a match is found, the corresponding bit positions are flipped to correct the packet. By definition, the CRC codes are designed such that each single error leads to a unique syndrome within the period of the generator polynomial used. The period of a generator polynomial is thus the number of different syndromes it can output for single errors. If the packet length surpasses the period of the generator polynomial, several single-error positions could lead to the same syndrome, thereby introducing ambiguity. Recently, some CRC-aided error correction methods have implemented a lookup table approach to increase

their correction capacities (Liu *et al.*, 2018). Besides high memory requirements for storing the table, such approaches raise two main issues:

- Lack of flexibility: lookup tables must be generated prior to the transmission, and cannot be dynamically modified to support multiple generator polynomials and larger packet sizes than those for which they were designed.
- Memory constraints: memory requirements for lookup table-based approaches rapidly increase with the number of errors to consider. In fact, such methods must store the entire set of possible error patterns and their associated syndrome.

In this paper, we propose a novel approach to error correction that outputs the exhaustive list of CRC-compliant binary sequences containing up to N errors. This method does not need any lookup table, which thus reduces the memory resources needed and allows the algorithm to be flexible as it can be used for any number of errors and any payload length without the need to rebuild a lookup table. Whereas CRC-aided ML methods (Niu & Chen, 2012) typically use CRC to check the validity of the candidates at the end of the MAP process, our method uses the CRC syndrome itself to produce the list of candidates, thus ensuring the CRC integrity of every candidate. The output list can be used to instantly correct the packet if it contains a single element or it can be used along with error correction or validation methods from upper layers of the protocol stack in order to reduce the list of candidates.

The paper is organized as follows. In section 3.3, we give a detailed description of the proposed method in three distinct parts. The first one describes the concept of the approach and its application to single-error correction. Challenges encountered with the double-error correction are then introduced and generalization to any number of errors is explained. In section 3.4, we present the proposed algorithm's performance as compared to state-of-the-art approaches. Tests are conducted according to different standards used in targeted applications, such as Wi-Fi (Association, 2016) and BLE (SIG, 2013). Simulation results demonstrate the superiority of the proposed solution in terms of error correction rate, computational complexity and memory usage. They show that for small-sized packets such as those found in IoT, we can achieve a 100%

correction rate for corrupted packets containing two errors or less, as well as high correction rates for three errors. In section 3.5, we conclude and give an overview of future research works.

3.3 Proposed Method

The proposed method uses the CRC syndrome value $s(x)$ computed at the receiver to list all the possible error patterns that lead to such a specific syndrome, considering a maximum number of errors. The resulting list can contain one or several entries at the end of the process. Each entry represents the positions of the bits to be flipped to recover a CRC-valid packet, i.e., it reveals the error positions. When the list contains only one element, we can instantly correct the packet, but when it contains several entries, additional information is required in order to identify the actual error pattern among the candidates. The proposed approach is flexible, and lists the whole set of possible error patterns with up to N errors, where the parameter N can be set according to the observed channel conditions, for instance. In this section, we first introduce the basic theoretical concepts of the proposed method for the single-error case. Then, we extend the method to double-error patterns, followed by multiple-error patterns.

3.3.1 Fundamentals

For convenience, it is common to use a binary vector representation of binary polynomials as described in (Arndt, 2011) and illustrated in Fig. 3.1. Using the vector representation, the degree of a coefficient corresponds to the bit position of the associated element in the vector. The length, in bits, of a vector is equal to the degree of the polynomial increased by 1, due to the existence of degree 0 in the polynomial (i.e., a polynomial of highest degree x^{15} will be represented as a 16-bit vector). Vectors allow a better understanding of operators such as *exclusive or* (XOR) and *binary left shifts*. Throughout this paper, specific notations will be used. The following is a list of such notations based on (Boyd & Vandenberghe, 2018) and their definitions:

- \mathbf{a} : binary vector $[a_k, \dots, a_0]$ of length $k+1$ associated with the binary polynomial $a(x)$ of degree k
- a_i : i^{th} bit (entry) of binary vector \mathbf{a} , starting from LSB

- m : payload length in bits
- n : syndrome length in bits
- M : total packet length in bits ($M = m + n$)
- N : number of errors searched
- P_1 : error position obtained from the single-error correction algorithm (Algorithm 3.1)
- \mathcal{F} : sorted list (F_1, \dots, F_{k-1}) of $(k - 1)$ bit positions forced to 1, such that $F_i < F_{i+1}, \forall i$
- $\text{len}(\mathcal{F})$: number of elements in the list \mathcal{F}
- E_i : set of valid error patterns containing i errors or less
- $\text{sum}(\mathbf{a})$: number of non-zero elements in a binary vector \mathbf{a} ; also denoted \sum when the context is clear
- \oplus : XOR operator between binary vectors
- $+$: XOR operator between polynomials
- \ll : left shift operator
- \leftarrow : affectation operator
- t_i : i^{th} step

We will frequently use the following binary vectors:

- $\mathbf{0}$: null vector (the length depends on the context)
- \mathbf{g} : generator polynomial vector of length $n + 1$ with $g_0 = 1$ and $g_n = 1$, given its definition (Bhatnagar, 2018)
- \mathbf{s} : syndrome vector of length n
- \mathbf{e} : error vector of length $M = n + m$

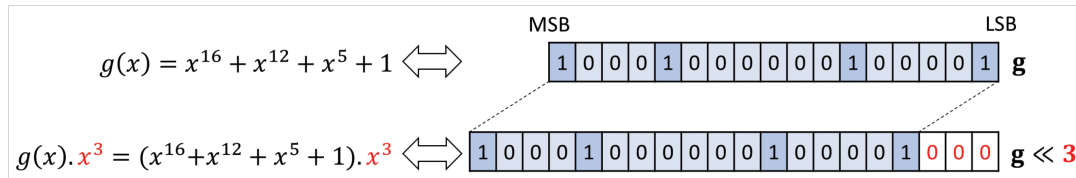


Figure 3.1 Illustration of the binary vector representation: each polynomial $g(x)$ with binary coefficients can be seen as a binary vector \mathbf{g} . Multiplying $g(x)$ by x^n corresponds to a left shift of \mathbf{g} by n positions

According to the definition of the CRC (Sobolewski, 2003), we know that the syndrome $s(x)$ is computed at the receiver as the remainder of the division of the received packet by the generator polynomial, which can be expressed as:

$$s(x) = p_R(x) \bmod g(x) \quad (3.2)$$

When no error occurs, the syndrome $s(x)$ is equal to a null polynomial. If we consider an error pattern $e(x)$, the syndrome of the received packet can be expressed as:

$$s(x) = (p_T(x) + e(x)) \bmod g(x) \quad (3.3)$$

where $s(x)$ is a non-null syndrome. A given syndrome value can be the result of several different error patterns $e(x)$, containing different numbers of errors. We denote $E_M(s(x))$ the set of all valid error patterns leading to the syndrome $s(x)$. In order to lighten the notation, we will use E_M since we are interested in a single syndrome value throughout the process. The error patterns in E_M contain between 1 and M errors (all bits of the packet are erroneous in the latter case). We denote E_i the subset of E_M comprising error patterns with i errors or less ($1 \leq i \leq M$). We thus have:

$$E_i \in E_{i+1} \quad \forall \quad 1 \leq i \leq M - 1 \quad (3.4)$$

where the E_i are not disjoint sets. The number of elements in E_M and in each subset E_i depends on the syndrome, the generator polynomial used and the packet length. We aim at finding the actual error pattern $e_A(x)$ among the set of all error patterns leading to the computed syndrome value $s(x)$. Given the definition of the modulo operator and (3.3), all error patterns of E_M are defined as:

$$E_M = \{e(x) \in \text{GF}(2^M) \mid e(x) = s(x) + q(x).g(x) \quad (3.5)$$

with $q(x) \in \text{GF}(2^m)\}$

where m is the payload length and $\text{GF}(2^m)$ is the Galois Field of order 2^m (i.e., the set of binary polynomials of length m (Arndt, 2011)). In other words, the error pattern corresponding to the syndrome can be any binary polynomial of highest degree $m - 1$ (that we denoted as $q(x)$) multiplied by the generator polynomial, with $s(x)$ added. The set E_M is called the equivalence class containing $s(x)$. Each element is equivalent under mod $g(x)$ operation since adding any multiple of $g(x)$ to $s(x)$ does not affect the result. Every possible value of $q(x)$ in this equation will produce a CRC-compliant error pattern $e(x)$ (i.e., an element of E_M). The degrees of the non-zero coefficients in the resulting $e(x)$ correspond to the erroneous positions in the corrupted packet. Assuming that packets are not too damaged, the straightforward approach to identifying candidates having a maximum number of errors would be to test every possible value of $q(x)$ and to count the number of non-zero coefficients in the resulting error polynomial $e(x)$. If this number, denoted $\text{sum}(\mathbf{e})$, is greater than a fixed threshold, the candidate is discarded. Otherwise, it is appended to the list of valid candidates. This method is computationally complex, and would require 2^m tests to consider all the possible values of $q(x)$. Such a complex process is therefore prohibitive to conduct in real-time scenarios, such as videoconferencing, for instance.

It can be verified that most of the possible values of $q(x)$ produce error polynomials $e(x)$ containing many errors (i.e., corresponding to highly corrupted packets cases, where $e(x)$ and its associated vector \mathbf{e} contain a significant amount of non-null values). Considering the whole set of possibilities would only increase the complexity of the method. We make the hypothesis that highly corrupted packets are too damaged to be recovered. Thus, in the rest of this paper, we focus on recovering slightly corrupted packets that are worth extracting information from. Some indicators such as the Receiver Signal Strength Indicator (RSSI), included in the 802.11 standard (Association, 2016), can be used to indicate the degree of corruption of a received packet. In the remainder of this section, we first describe the single-error correction method (the search for all elements in E_1), and then we introduce the double-error correction and extend it to any number N of errors (i.e., we determine the elements of E_N).

3.3.2 Single-Error Correction

We exploit the knowledge on both the generator polynomial and the way the syndrome is computed to reversely find the position of the single error at the receiver side. With such an approach, we are not testing possible values of $q(x)$, but rather, are gradually building a specific polynomial $q(x)$, one coefficient at a time. If a single candidate is identified at the end of the process, the packet can be corrected. If not, some additional processes must be used to determine the only candidate to consider.

We now demonstrate that the proposed approach is guaranteed to identify single errors. Suppose that the error is at position P_1 (i.e., $e(x) = x^{P_1}$). We know from the definition of (3.5) that:

$$x^{P_1} = s(x) + q(x).g(x) \text{ for a } q(x) \in \text{GF}(2^m) \quad (3.6)$$

It is clear that $q(x)$ must be constructed such that $s(x) + q(x).g(x)$ has zero coefficients for all positions $i \neq P_1$. Having coefficients at positions $i < P_1$ is ensured by successively determining, from LSB to Most Significant Bit (MSB), the coefficient values of $q(x)$ meeting this condition. For simplicity, in the following derivations, we can consider $s(x)$ of degree $m - 1$

with $s_i = 0, i > n - 1$. We have:

$$\begin{aligned}
x^{P_1} &= \sum_{i=0}^{m-1} s_i x^i + \left(\sum_{i=0}^{m-1} q_i x^i \right) \left(\sum_{j=0}^n g_j x^j \right) \\
&= \sum_{i=0}^{m-1} s_i x^i + \sum_{i=0}^{m-1} q_i \cdot \sum_{j=0}^n g_j x^{i+j} \\
&= \sum_{i=0}^{m-1} s_i x^i + \sum_{i=0}^{m-1} q_i \cdot \sum_{r=i}^{i+n} g_{r-i} x^r \\
&= \sum_{i=0}^{m-1} s_i x^i + \sum_{i=0}^{m-1} \left(q_i \cdot g_0 x^i + q_i \cdot \sum_{r=i+1}^{i+n} g_{r-i} x^r \right) \\
&= \sum_{i=0}^{m-1} \left((s_i + q_i \cdot g_0) x^i + q_i \cdot \sum_{r=i+1}^{i+n} g_{r-i} x^r \right) \\
&= \sum_{i=0}^{m-1} \left((s_i + q_i) x^i + q_i \cdot \sum_{r=i+1}^{i+n} g_{r-i} x^r \right), \text{ since } g_0 = 1
\end{aligned} \tag{3.7}$$

From (3.7), it is clear that for every value of i in the main summation, $(s_i + q_i \cdot g_0) x^i$ is of a lower degree than $q_i \cdot \sum_{r=i+1}^{i+n} g_{r-i} x^r$. Thus, for $i = 0$ and each successive value of i , we can easily determine the q_i value resulting in the desired result, namely, zero coefficients for positions $i < P_1$. Of course, setting q_i to 1 creates terms that must be considered in subsequent positions. If $s(x)$ was generated by a single error, performing the process on increasing values of i would eventually lead to a monomial (i.e., x^{P_1} for a certain value of P_1). This must happen, otherwise, after position $i = P_1$, adding $\sum_{r=i+1}^{i+n} g_{r-i} x^r$ (i.e., $q_i \neq 0$) would add a coefficient at position $i + n$ (the MSB) that cannot be canceled without adding a coefficient of even higher degree.

The search for single-error patterns is illustrated in Algorithm 3.1 and the corresponding flowchart is given in Fig. 3.2. Each step of the algorithm is identified in Fig. 3.2 using binary notations. We provide further details in the following steps:

3: We first initialize the error \mathbf{e} to a zero vector of length $M = m + n$ and replace the n LSB values with the computed syndrome \mathbf{s} , as shown in Fig. 3.3. We can note that it corresponds in (3.5) to $e(x) = q(x) \cdot g(x) + s(x)$, where $q(x)$ is equal to zero. Such initialization allows to

Algorithm 3.1 SingleErrorCorrection($\mathbf{s}, \mathbf{g}, n, m$)

Inputs:

\mathbf{s} : the syndrome vector
 \mathbf{g} : the vector associated with the generator polynomial
 used to compute the CRC
 n : the length of the syndrome vector
 m : the length of the payload vector

Output:

E_1 the list of valid error patterns for a single bit error

```

1:  $E_1 \leftarrow \{\}$ 
2: Let  $\mathbf{e}$  be a vector of length  $m + n$ 
3:  $\mathbf{e} \leftarrow \mathbf{0} \oplus \mathbf{s}$ 
4: if  $\text{sum}(\mathbf{e}) = 1$  then
5:   Add  $\mathbf{e}$  to  $E_1$ 
6: end if
7: for  $j = 0$  to  $m - 1$  do
8:   if  $e_j = 1$  then
9:      $\mathbf{e} \leftarrow \mathbf{e} \oplus (\mathbf{g} \ll j)$ 
10:    if  $\text{sum}(\mathbf{e}) = 1$  then
11:      Add  $\mathbf{e}$  to  $E_1$ 
12:    end if
13:  end if
14: end for
15: Return  $E_1$ 

```

comply with (3.5) and maintains its equivalence relation as we are adding shifted versions of $g(x)$ to build $q(x)$ in step 9.

4-5: At this point, we compute the sum of non-zero elements in $\mathbf{e} = \mathbf{s}$, denoted $\text{sum}(\mathbf{e})$, equivalent to the number of errors in the computed syndrome. If it contains only one element to 1, then it is itself a suitable candidate as a single-error pattern.

7: We scan the m first payload positions from 0 to $m - 1$. We do not consider the last n positions since they correspond to the range of the XOR operation to perform. Hence, it reaches the end of the payload at position $m - 1$ and beyond this position would be out of the payload range.

8-9: For each scanned position, we check the j^{th} bit value of the current error vector. If this

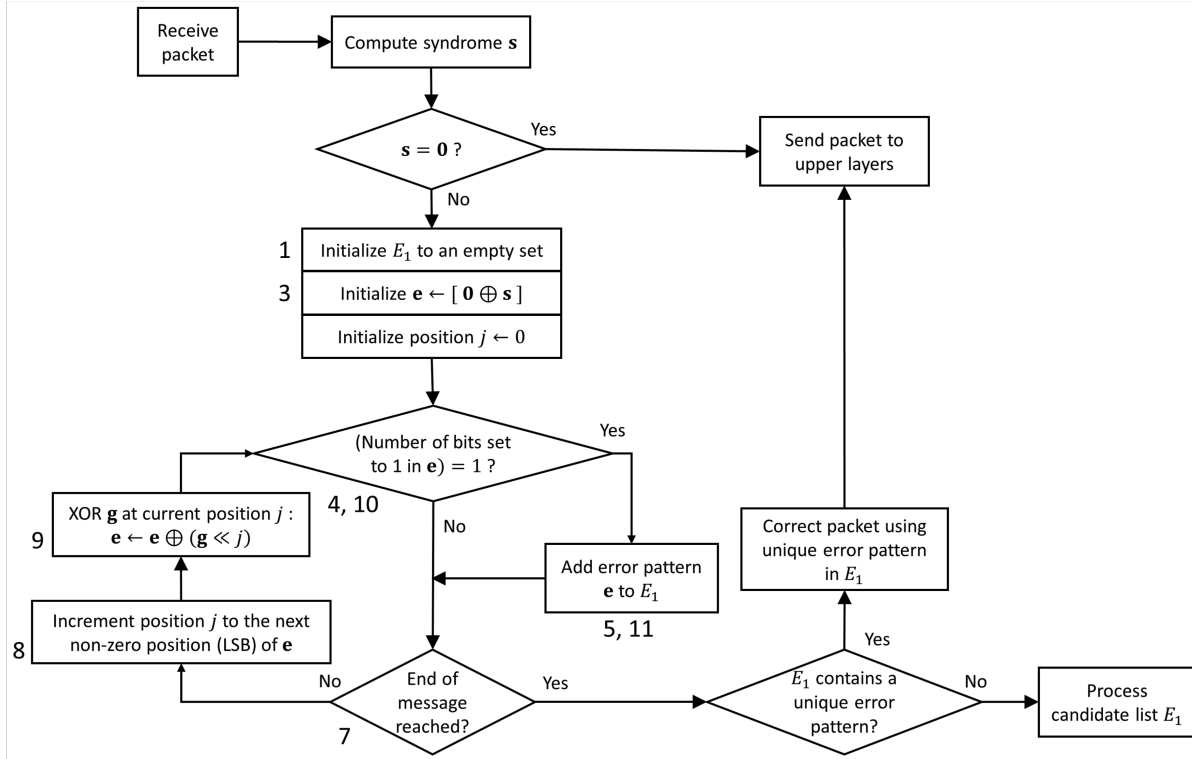


Figure 3.2 Flowchart of the proposed method's algorithm to correct a single error in the packet. Numbers show the corresponding steps in Algorithm 3.1

value is 0, we simply jump to the next element. If it is 1, we cancel the non-zero value by performing an XOR operation with \mathbf{g} at this position, as its LSB is 1 (i.e., $g_0 = 1$). Note that for clarity, we simplified this step in the figures and flowcharts by directly incrementing the current position to the next element set to 1. Each time we perform an XOR operation at position j , a 1 is added at MSB position $j + n$ since $g_n = 1$. If the error pattern is a single error at position k , the proposed method will reveal this since the XOR operations will be able to cancel all bits at positions $j < k$, and all bits at positions $j > k$ are already set to zero. The strategy is to

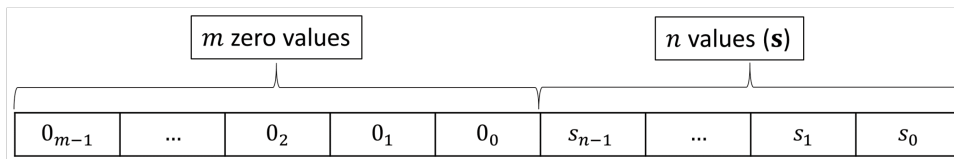


Figure 3.3 Structure of the initial error vector $\mathbf{e} = \mathbf{0} \oplus \mathbf{s}$

cancel every LSB non-zero element until the end of the packet is reached, and thus not miss any single-error candidate.

10-11: After each cancelation, we check the number of non-zero coefficients in the error vector **e**. If this number is equal to 1, a valid single-error candidate is identified and its position is appended to the list.

At the end of the whole process, if the algorithm does not provide any candidate, it means the syndrome was caused by multiple errors in the packet.

So, depending on the packet size and syndrome, there can be zero, one or multiple candidates. This latter case occurs in long enough packets due to the periodic aspect of generator polynomials, as discussed in the introduction. The whole single-error search process is illustrated in Fig. 3.4. In the present case, the payload consists of 10 data bits and the CRC-4-ITU where a generator polynomial $g(x) = x^4 + x + 1$ is applied. At the receiver, the computed syndrome is $s(x) = x^2 + 1$, represented in dark grey boxes at step t_0 .

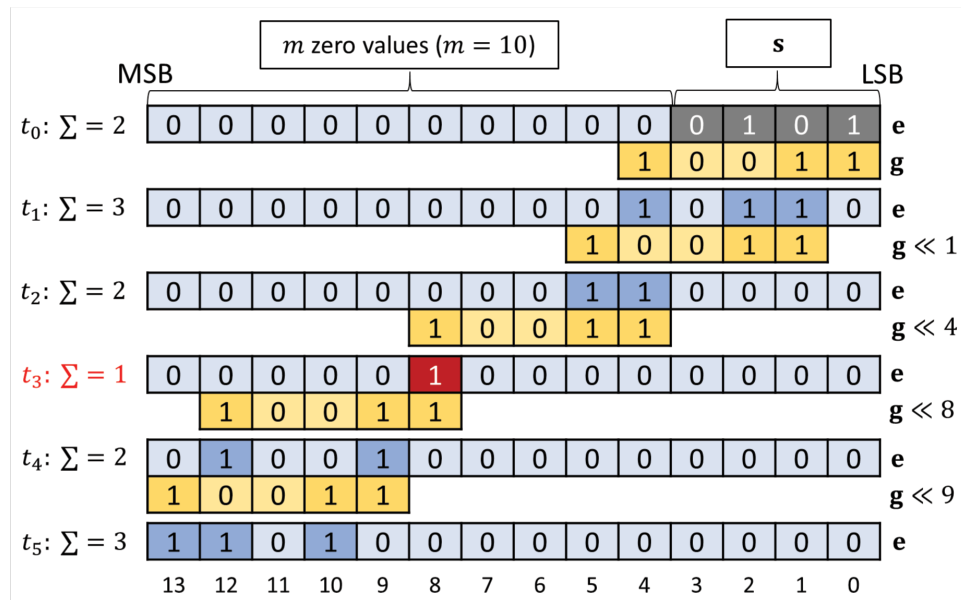


Figure 3.4 Illustration of the single-error search applied to CRC-4-ITU, where $g(x) = x^4 + x + 1$ (yellow cells) with a syndrome $s(x) = x^2 + 1$ (grey cells). In this notation, Σ represents $\text{sum}(\mathbf{e})$. A single-error pattern is found at bit position 8 at step t_3 . In this example: $E_1 = \{8\}$ (i.e., the red cell)

At step t_0 , the error vector \mathbf{e} is initialized to m zeros, m being the length of the protected data, and the syndrome \mathbf{s} is appended. We first check the number of non-zero values in the error vector to verify if the syndrome itself is a valid candidate (i.e., if the syndrome is corrupted). Since the sum of non-zero values in \mathbf{s} is greater than 1, the syndrome does not contain a valid single-error pattern, and is thus not a candidate. At each step until we reach the end of the packet, we successively perform an XOR operation with \mathbf{g} at each non-null position and check the resulting number of 1s in the updated error vector \mathbf{e} . If this sum is equal to 1, the candidate is appended to the list. Such a candidate is found at time t_3 , since there is only one bit set in the error vector. A first single-error candidate is thus identified, containing an error at position 8. Since there could be several candidates, we continue the scanning of the packet until the end. At step t_5 , the algorithm reaches the end of the packet and the list of candidates contains a single entry. Flipping the bit at position 8 in the corrupted packet is the only valid correction if a single error has occurred.

3.3.3 Double-error correction

3.3.3.1 Problem with straightforward extension of Algorithm 3.1

The method described in the previous section produces as output the exhaustive list of single-error patterns corresponding to a non-null syndrome at the receiver, given the generator polynomial used and the length of the protected data. To deal with double-error patterns, a straightforward method would be to run the exact same algorithm while appending all the error vectors \mathbf{e} with two coefficients set to 1 to the candidate list. This approach would be able to output double-error patterns, but cannot ensure that an exhaustive list of such error patterns is provided. Actually, only one specific type of double-error patterns will be output, namely, those in which the double-error pattern covers n bits or less (i.e., are close to each other). The single-error search aims at canceling non-zero values from LSB to MSB. This cancelation is performed thanks to an XOR operation between a shifted version of the generator polynomial and the constantly updated error vector. We can observe that there cannot be more than n bits between the 1 located

at the LSB position and the 1 at the MSB position, as illustrated in Fig. 3.5, which represents the error vector during the single-error search. The MSB zeros correspond to the positions still in the original state of \mathbf{e} , initialized as a null vector, and the LSB zeros correspond to the already canceled positions. Between these two null subvectors we have the possible non-zero positions, with a maximum width of n bits. We will refer to the maximum distance between the first and last non-zero coefficients as the error range of the method. At step t_9 of Algorithm 3.1, the update of the error vector \mathbf{e} can be expressed as:

$$\begin{aligned}
 \sum_{i=0}^{m+n-1} e_i x^i &\leftarrow \sum_{i=0}^{m+n-1} e_i x^i + \sum_{i=j}^{j+n} g_{(i-j)} x^i \\
 &= \sum_{i=j+1}^{j+n} (e_i + g_{(i-j)}) x^i \\
 &= x^{(j+n)} + \sum_{i=j+1}^{j+n-1} (e_i + g_{(i-j)}) x^i
 \end{aligned} \tag{3.8}$$

From (3.8), it is clear that the whole set of non-null values in the error vector covers n positions at most. In fact, all the values in \mathbf{e} up to position j are already canceled and set to 0, due to the design of the proposed algorithm, and all positions above $j + n$ are also set to 0 due to the initialization of the error vector (i.e., $\mathbf{e} = \mathbf{0} \oplus \mathbf{s}$). As the highest degree term of the generator polynomial is 1, we can see that position x^{j+n} is set. The other non-null positions are subject to the values of the error vector at its current state and the other terms of the generator polynomial. The range of non-null values is denoted the error range, and is illustrated in Fig. 3.5. In conclusion, a straightforward extension of Algorithm 3.1 would only yield error patterns in which errors are within a range of n bits. A different approach is thus required.

3.3.3.2 Proposed double-error correction approach

To obtain the exhaustive list of error patterns, we aim at expanding the error range to have it cover the entire length of the protected data. The method we propose is to force a bit to 1 during the process. Forcing a position consists in setting it (or leaving it) to 1 during the single-error

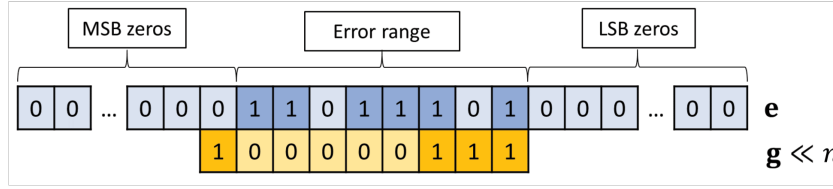


Figure 3.5 Illustration of the error range applied to CRC-CCITT-8 ($n = 8$). Canceling LSB non-zero values by performing an XOR operation with a generator polynomial of width $n + 1$ bits produces an error range of n bits.

search. In other words, it is equivalent to making the hypothesis that a specific position is actually erroneous in the packet. Hence, we force one bit to 1 at position F_1 during the process and run the single-error algorithm on the remaining length of the packet. If the bit is already 1, we leave it untouched. Otherwise, setting a bit to 1 is done by applying an XOR operation with $g(x)$ at position F_1 in order to maintain the equivalence relation. Throughout the cancelation process with the forced bit set, if a single-error position (denoted hereafter P_1) is obtained from the single-error correction algorithm, we determine a double-error pattern with errors at positions F_1 and P_1 .

As we want to get the whole list and we do not know the actual position of the first error, we test each possible forced position in order to output all the double-error patterns associated with the computed syndrome. In the proposed algorithm, we suggest forcing positions starting from LSB to MSB. Moreover, starting from LSB at each tested forced position would lead to a cancelation of the same first positions several over and degrade the computational efficiency. To avoid verifying the same possibilities repeatedly, we store the value of e when a bit is forced and recall this state to start from it and save computations for the next forced position to test.

Fig. 3.6 illustrates the complete process for listing the double-error patterns corresponding to the syndrome $s(x) = x^3 + x^2 + 1$ applied to a CRC-4-ITU of generator polynomial $g(x) = x^4 + x + 1$ protecting 6 data bits. In this figure, forced positions are represented as black boxes through time. At step t_0 , the error vector e is initialized as a null vector, to which syndrome vector $s(x) = x^3 + x^2 + 1$ is appended. As we start at position 0, and e_0 is already set to 1, we simply jump to the next element and start the single-error search from the next non-zero position,

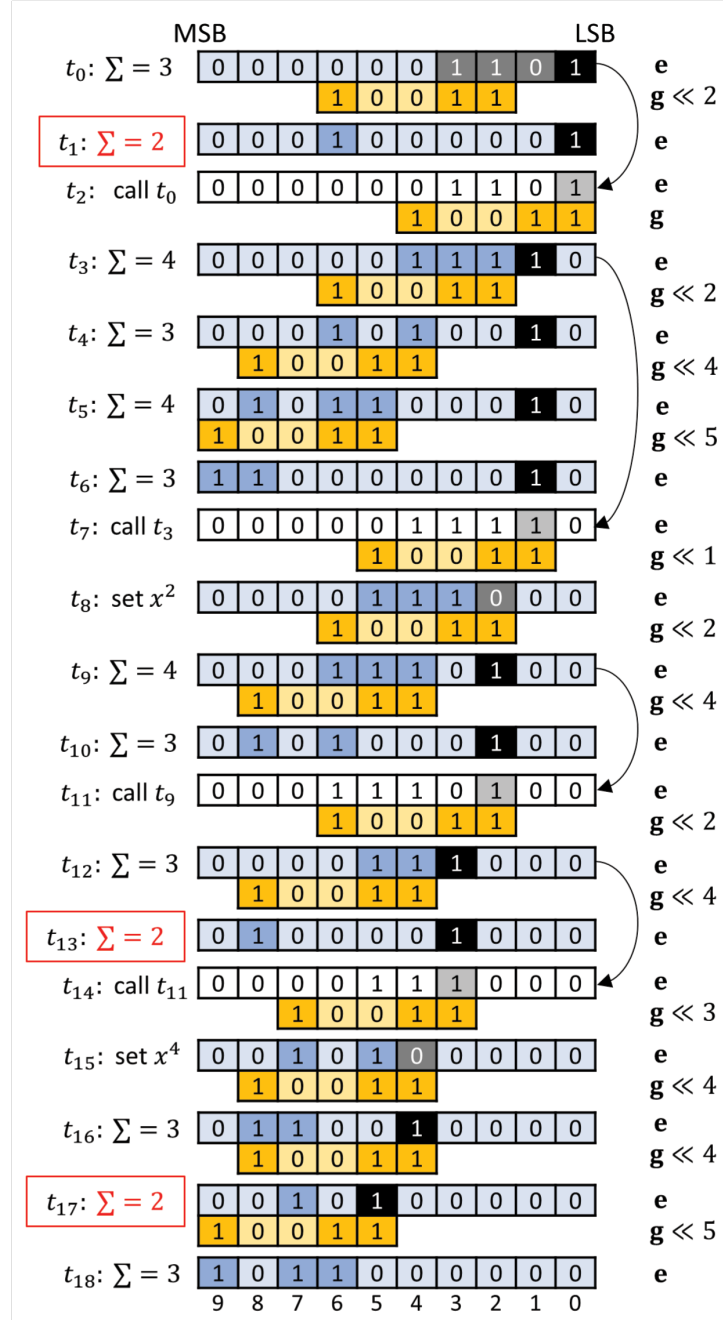


Figure 3.6 Visual example of the proposed algorithm applied to double-error correction, performed over CRC-4-ITU (yellow cells) protecting 6 data bits with a syndrome $s(x) = x^3 + x^2 + 1$ (grey cells). Each forced bit position, represented as a black cell, is tested throughout the process to get the exhaustive list of double-error patterns. In this example there are three such cases at steps t_1 , t_{13} and t_{17} , thus $E_2 = \{(0, 6); (3, 8); (5, 7)\}$

checking at each step if the number of non-zero coefficients in \mathbf{e} equals 2. A first candidate appears at t_1 , corresponding to errors at positions $(F_1=0, P_1=6)$. Canceling the next non-zero value would move the operation out of the range of the packet, thus ending the search for a single error for this forced position. At step t_2 we recall the initial state of the error vector from t_0 . The next forced bit to test is at position 1. Hence, we cancel position 0 and let position 1 be set to 1. From t_3 to t_6 , we perform the single-error algorithm on the remaining length. No new error pattern is found. At time t_7 , we recall the previous state from t_3 and cancel the former forced bit at position 1. At t_8 , the next forced position to test, position 2, is not yet set to 1. We thus have to perform an XOR operation with \mathbf{g} at this position to set it. We identify such cases as dark grey boxes in Fig. 3.6, at steps t_8 and t_{15} . We continue this process until reaching the last forced bit position, corresponding to the m^{th} position starting from LSB. We can see at step t_{18} that we cannot perform any XOR operation without going out of the range of the packet. Hence, the algorithm is stopped at t_{18} and outputs the list of error patterns containing two errors corresponding to the received syndrome. The sums of errors in such cases are shown in red font in Fig. 3.6. The output list contains the following error patterns: $(F_1=0, P_1=6)$, $(F_1=3, P_1=8)$ and $(F_1=5, P_1=7)$. The proposed approach for double-error correction is exemplified in Fig. 3.6 and presented in Algorithm 3.2 using $N = 2$.

3.3.4 N-error correction

We can further extend the proposed method to deal with any number N of errors in a packet. The strategy applied is the extension of the double-error correction approach described in the previous section.

Much as we forced one position and scanned the remaining length of the packet using the single-error search, we can manage the N -error search. In such cases, we set $(N - 1)$ forced bits in the error vector, corresponding to the first $(N - 1)$ errors in the packet, and scan the remaining length using the single-error search to identify the position of the last error in the packet, if it exists. The $(N - 1)$ forced binary errors have to be tested in the packet. The proposed

Algorithm 3.2 N -ErrorPatternsGeneration($\mathbf{s}, \mathbf{g}, n, m, N$)

Inputs:

\mathbf{s}, \mathbf{g} : the syndrome and generator polynomial binary vectors, respectively
 n, m : the lengths of the syndrome and payload vectors
 N : the maximum number of bit errors considered

Output:

E_N the list of valid error patterns up to N bit errors

```

1:  $E_N \leftarrow \{\}$ 
2: Let  $\mathbf{e}$  be a vector of length  $m + n$ 
3:  $\mathbf{e} \leftarrow \mathbf{0} \oplus \mathbf{s}$ 
4: Let  $\mathbf{v}$  be a vector of length  $m$ 
5: if  $\text{sum}(\mathbf{e}) \leq N$  then
6:   Add  $\mathbf{e}$  to  $E_N$ 
7: end if
8:  $k \leftarrow N$ 
9: while  $k \geq 1$  do
10:  if  $k = 1$  then
11:    Add SingleErrorCorrection( $\mathbf{s}, \mathbf{g}, n, m$ ) to  $E_N$ 
12:  else
13:    Let  $\mathcal{F} \leftarrow (0, \dots, k - 2)$ 
14:     $\mathbf{v} \leftarrow \text{PositionsToVector}(\mathcal{F})$ 
15:    while  $\mathcal{F} \neq (m - (k - 1), \dots, m - 1)$  do
16:       $\text{start} \leftarrow \max(F_1 - 1, 0)$ 
17:      for  $j = \text{start}$  to  $m - 1$  do
18:        if  $e_j \neq v_j$  then
19:           $\mathbf{e} \leftarrow \mathbf{e} \oplus (\mathbf{g} \ll j)$ 
20:          if  $\text{sum}(\mathbf{e}) \leq N$  then
21:            Add  $\mathbf{e}$  to  $E_N$ 
22:          end if
23:        end if
24:        if  $j = F_1$  then
25:           $\mathbf{e}' \leftarrow \mathbf{e}$ 
26:        end if
27:      end for
28:       $\mathcal{F} \leftarrow \text{UpdateForcedPositions}(\mathcal{F}, m)$ 
29:       $\mathbf{v} \leftarrow \text{PositionsToVector}(\mathcal{F})$ 
30:       $\mathbf{e} \leftarrow \mathbf{e}'$ 
31:    end while
32:  end if
33:   $\mathbf{e} \leftarrow \mathbf{0} \oplus \mathbf{s}$  and
34:   $k \leftarrow k - 1$ 
35: end while
36: Remove duplicate elements in  $E_N$ 
37: Return  $E_N$ 

```

Algorithm 3.3 UpdateForcedPositions(\mathcal{F}, m)

Inputs:

\mathcal{F} : sorted list (F_1, \dots, F_{k-1}) of $(k - 1)$ bit positions
 forced to 1, such that $F_i < F_{i+1}, \forall i$
 m : the length of the payload vector

Note that $k = \text{len}(\mathcal{F}) + 1$, with $\text{len}(\mathcal{F})$ being the number of
 elements in the list \mathcal{F}

Output:

\mathcal{F}' : the updated sorted list of forced positions

```

1: if  $F_{k-1} < (m - 1)$  then
2:    $F_{k-1} \leftarrow F_{k-1} + 1$ 
3:   Return  $\mathcal{F}' \leftarrow (F_1, \dots, F_{k-1})$ 
4: else
5:   for  $i = k - 2$  to 1 do
6:     if  $F_i < F_{i+1} - 1$  then
7:        $F_i \leftarrow F_i + 1$ 
8:        $j \leftarrow i$ 
9:       while  $j < k - 1$  do
10:         $F_{j+1} \leftarrow F_j + 1$ 
11:         $j \leftarrow j + 1$ 
12:      end while
13:      Return  $\mathcal{F}' \leftarrow (F_1, \dots, F_{k-1})$ 
14:   end if
15: end for
16: end if

```

method to generate the list of potential error patterns containing up to N errors is illustrated in Algorithm 3.2.

We now present the key steps of the proposed algorithm, while the corresponding flowchart is given in Fig. 3.8:

3: The binary vector of length M representing the error vector \mathbf{e} is initialized to m zeros, followed by n values, corresponding to the computed syndrome \mathbf{s} .

5-6: We first check if the number of non-zero values in this initial vector \mathbf{e} is less than or equal

Algorithm 3.4 PositionsToVector(\mathcal{F})

Inputs:
 \mathcal{F} : sorted list (F_1, \dots, F_{k-1}) of $(k - 1)$ bit positions forced to 1, such that $F_i < F_{i+1}, \forall i$.

Note that $k = \text{len}(\mathcal{F}) + 1$, with $\text{len}(\mathcal{F})$ being the number of elements in the list \mathcal{F}

Output:
 \mathbf{v} : the corresponding vector of forced positions

```

1:  $\mathbf{v} \leftarrow \mathbf{0}$ 
2: for  $i = 1$  to  $k - 1$  do
3:    $v_{F_i} \leftarrow 1$ 
4: end for
5: Return  $\mathbf{v}$ 

```

to the targeted number of errors N . If so, a first candidate is added to the list E_N .

8-9: The local variable k represents the current number of errors considered. k is initialized to N , then decreased at each main loop of the algorithm to consider every number of errors from N to 1.

10-11: In the last loop, the variable k equals 1. In this case, no forced position must be set and the single-error correction algorithm is performed. The output candidate list is then appended to the global candidate list E_N .

13: The sorted list of forced positions \mathcal{F} is initialized to the $(k - 1)$ LSB values at the first iteration. At this step, the set of forced positions $\mathcal{F} = (F_1 = 0, F_2 = 1, \dots, F_{k-1} = (k - 2))$. The $(k - 1)$ forced positions in the set \mathcal{F} are ordered such that $F_1 < F_2 < \dots < F_{k-1}$.

14: The binary vector is set according to the forced positions in \mathcal{F} . In Algorithm 3.4, the bits in \mathbf{v} corresponding to forced positions in \mathcal{F} are set to 1. The other bits in \mathbf{v} are set to 0.

15: The forced positions will then be updated to cover the entire set of possible fixed error positions (until the forced positions are the $(k - 1)$ MSB positions). For a packet of M bits, there are $\binom{M-n}{k-1}$ such positions, thanks to the range of the XOR operation performed. After setting these forced positions, we are aiming at finding the last error by conducting the single-error

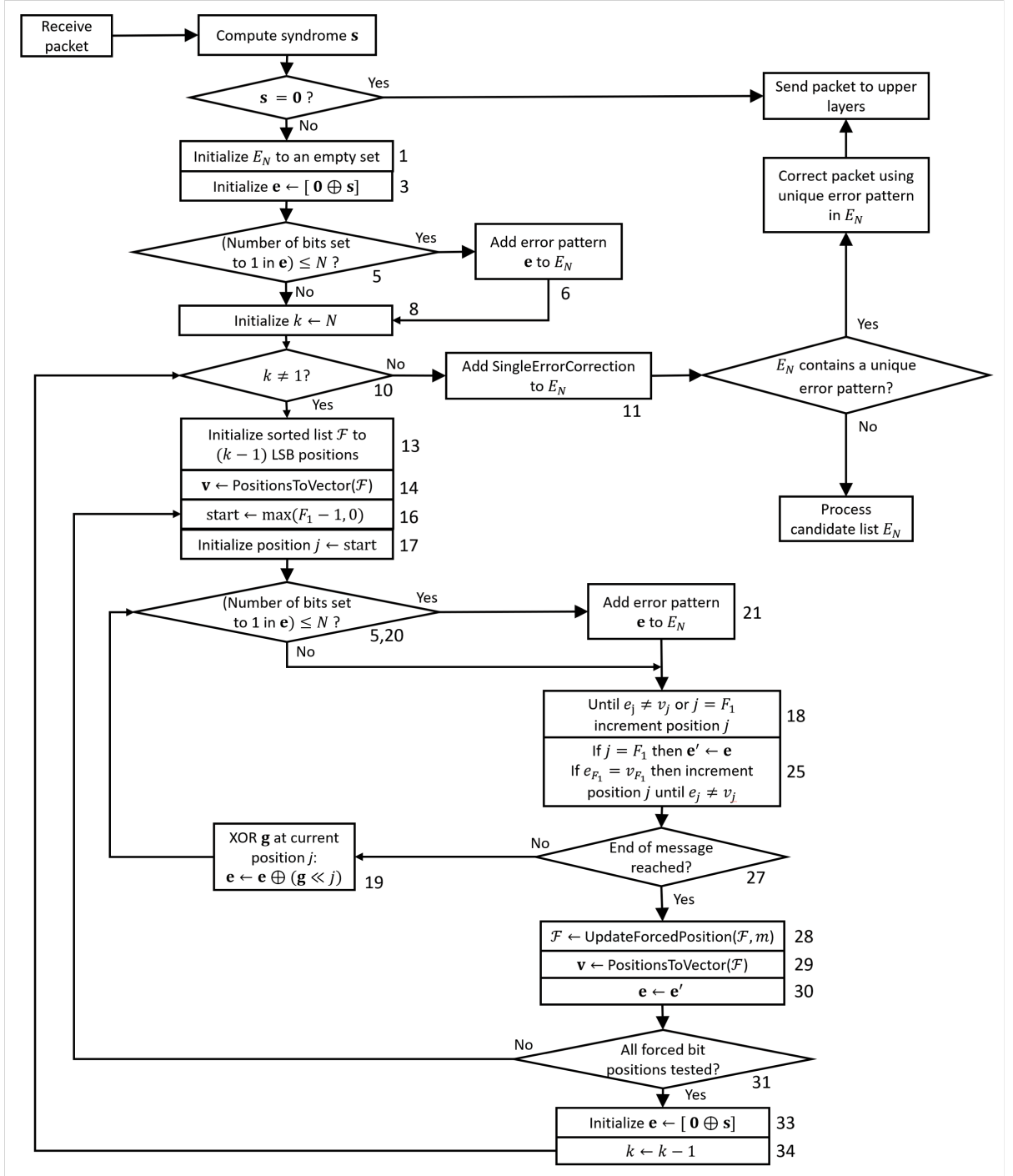


Figure 3.8 Flowchart of the proposed method algorithm for correcting multiple errors in the packet. Numbers show the corresponding steps in Algorithm 3.2

corresponds to a forced position. If v_j and e_j are both set to 0 or 1, it means that position j either must not be forced (to 1) and is already set to 0, or must be forced, but is already set to 1. In both cases, the algorithm simply jumps to the next element since what is required is already in place. However, when these two elements are set to different values, it corresponds to the cases where the position j has to be canceled and set to 1, or where the position j has to be forced to 1, but is set to 0. In both cases, an XOR operation with \mathbf{g} must be performed to maintain the equivalence relation and obtain what is required.

20-22: At each stage, the number of non-zero coefficients in the newly accumulated \mathbf{e} is observed, similarly to steps 5-6. Whenever \mathbf{e} contains N errors or less, a candidate is added to the list E_N .

25: We store the state of \mathbf{e} in a vector \mathbf{e}' to avoid re-canceling the same first LSBs at the next iteration.

28: At the end of each scan, the vector of forced positions is updated using the UpdateForced-Position function illustrated in Algorithm 3.3. In this algorithm, the $(k - 1)$ forced positions are successively updated to cover the entire message. At step 1, we check if the MSB forced position has reached its final position. If not, we increase its value by one. If it has reached its final position, we successively check forced positions from MSB to LSB at step 5. When a forced position can be increased, we reversely update the other positions, from LSB to MSB.

29: Each time the set of forced positions is updated, the binary vector \mathbf{v} is modified.

30: We recall the state \mathbf{e}' to start from it at the next iteration.

33: We recall the initial state (syndrome) when we update the number of errors to consider.

Fig. 3.7 shows a visual example of the algorithm applied to a CRC-8-CCITT, which has a generator polynomial $g(x) = x^8 + x^2 + x + 1$. This example illustrates a triple-error management, with \mathbf{v} having two non-zero values, represented as black boxes in Fig. 3.7. Four stages are represented here: the initial stage, where the forced error positions are $(F_1 = 0; F_2 = 1)$, two different stages that produce a valid candidate, namely $(F_1 = 1; F_2 = 6)$ and $(F_1 = 2; F_2 = 8)$, and the final step, with the last two forced positions $(F_1 = 10; F_2 = 11)$. By definition, these positions must remain set to 1 at the end of the scan. The other non-null values in \mathbf{e} must be canceled from LSB to MSB, using the single-error search method. At each step, we successively add

left-shifted versions of \mathbf{g} at these positions and if the sum of non-null values in \mathbf{e} is equal to 3 ($N = 3$), then \mathbf{e} is considered as a valid candidate. We can observe that this example produces three valid error patterns with error positions $(F_1=0; F_2=1; P_1=18)$, $(F_1=1; F_2=6; P_1=16)$ and $(F_1=2; F_2=8; P_1=18)$, shown in red font in Fig. 3.7. The design of Algorithm 3.2 yields a total complexity, measured in number of XOR operations, of $O(m^N)$. Testing the entire error pattern to determine which would match the received syndrome (i.e., brute force scheme) would have a complexity of $O(m^{N+1})$. We can note that the complexity increases significantly with the number of errors considered N . We thus recommend using the algorithm when N is low, depending on the processing time constraints of the targeted application. It is important to note that correcting a single error using algorithm 3.1 is not computationally more complex than performing a classic CRC check at the receiver.

3.4 Simulation and results

In this section, we present the theoretical and simulation performance of the proposed method, as compared to the lookup table approaches from the literature. Finally, we apply our method to Bluetooth Low Energy used in the IoT and compare its performance to state-of-the-art methods.

3.4.1 Correction rate

In this section, we evaluate the performance of the proposed error correction method. It is able to instantly correct the packet when there is only one candidate in the output candidate list. Hence, the performance can be expressed as the percentage of error cases that produce only one candidate in the list. We will refer to such a percentage as the SCR. The SCR is a function of three parameters: the generator polynomial used, the length of the protected data and the number of errors considered. Given a generator polynomial $g(x)$, we denote the N -error patterns for a packet length m leading to a single candidate as $\text{SinglePatterns}(m, N)$ and the total number of possible N -error patterns for the same length m as $\text{TotalPatterns}(m, N)$, and we can express the SCR as:

$$\text{SCR}(m, N) = \frac{\text{SinglePatterns}(m, N)}{\text{TotalPatterns}(m, N)} \quad (3.9)$$

where $\text{SinglePatterns}(m, N)$ was determined by running the algorithm over all the error cases and $\text{TotalPatterns}(m, N) = \binom{M}{N}$. SCRs are thus not estimated but computed over the entire set of possible error patterns. We verify that when the length and the number of errors considered increase, the SCR decreases rapidly. Moreover, the length of the generator polynomial, as well as the number of non-zero coefficients, modify the way the SCR decreases as the length of the protected data increases.

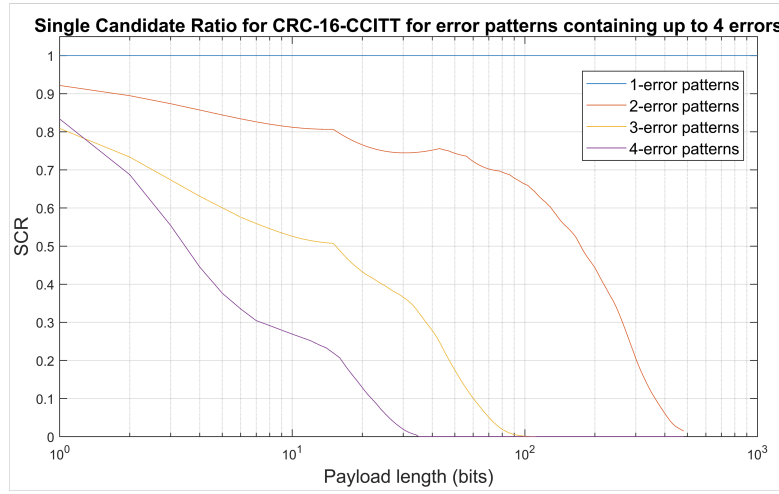


Figure 3.9 SCR for a payload length from 0 to 500 bits protected by a CRC-16-CCITT (Association, 2011) of generator polynomial $g(x) = x^{16} + x^{12} + x^5 + 1$, considering up to 4 errors

In Figs. 3.9 and 3.10, we show the evolution of the SCR for different generator polynomials and different numbers of errors considered. We observe in Fig. 3.10 that when a long generator polynomial is used and few errors are considered, the SCR stays at 100% up to a significant length. On the other hand, a short generator polynomial leads to a faster decrease of the SCR as the packet's length increases, as illustrated in Fig. 3.9. When the SCR is 100% up to a certain threshold length for N errors, it means that if N errors or less occur during the transmission of the packet, these errors can be identified with a certainty of 100% and without any possible ambiguity when the packet's length is lower than this threshold. From this threshold, the SCR does not fall to zero immediately. Depending on the generator polynomial chosen, it can still be at a high percentage level up to a significant packet size. If we take the example of CRC-24 used in the Bluetooth Low Energy protocol (SIG, 2013) (of generator polynomial

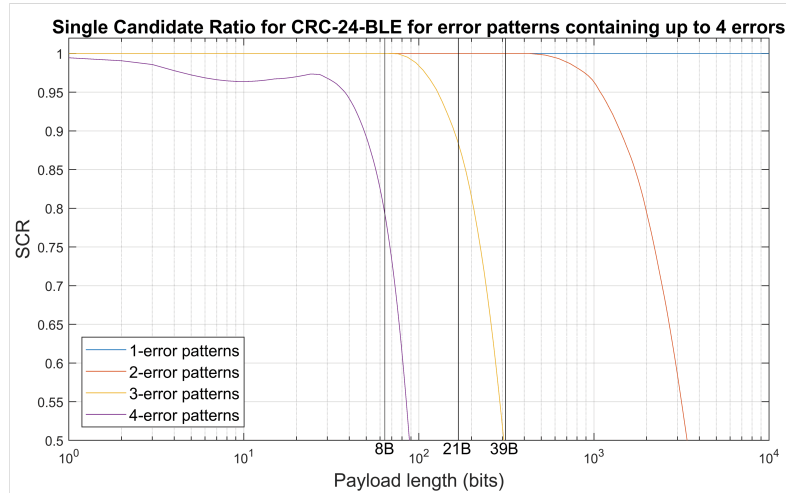


Figure 3.10 SCR for a payload length from 0 to 10 000 bits protected by a CRC-24-BLE (SIG, 2013) of generator polynomial $g(x) = x^{24} + x^{10} + x^9 + x^6 + x^4 + x^3 + x + 1$, considering up to 4 errors

$g(x) = x^{24} + x^{10} + x^9 + x^6 + x^4 + x^3 + x + 1$), the SCR is still above 80% for a payload of up to 2000 bits when considering that two errors occurred in the packet. For three and four errors, this number decreases greatly, but is still over 80% for up to 220 bits for three errors and up to 85 bits when considering four errors. The applications targeted by the CRC-24 used in the Bluetooth Low Energy standard concern the (IoT) (Zanella *et al.*, 2014; Al-Fuqaha *et al.*, 2015). In IoT environments, the average packet payload is often just a few bytes in size. Consequently, the proposed error correction method will be able to instantly correct most error patterns up to four errors, and even 100% of error patterns up to two errors, given a packet of 450 bits or less.

However, if the packet is highly corrupted, it may conceivably produce a syndrome the algorithm would recognize as the result of a low number of errors. In such a case, we would have a miscorrection. This corresponds, however, to very disadvantageous cases for all error correction methods. In fact, there is no error correction method that guarantees the validity of the reconstructed sequence. However, what we have is no more problematic than the case of highly corrupted packet yielding a CRC syndrome of zero, letting the receiver believe there is no error.

3.4.2 Memory requirements

The proposed method does not require storing any table. In contrast, the main drawback of lookup approaches is their memory requirements. The table must be stored in the receiver's memory, as shown in Fig. 3.11. On very small-sized packet lengths as considered in (Shukla & Bergmann, 2004) and (Babaie *et al.*, 2009), the lookup table represents a viable solution. When dealing with large packets, the required memory increases very rapidly. This rapid increase is also seen as the number of errors considered increases.

We evaluate the memory required when considering a specific number of errors and for each common syndrome length. Two different approaches are used to construct the lookup table. In both cases, the number of entries in the table corresponds to the number of possible error patterns. From this definition, it becomes clear that a lookup table that considers a packet length M and N errors would have $\binom{M}{N}$ rows. At each of these entries, the table must store the non-null syndrome for every possible error pattern. The syndrome is stored as a 1 to 4-byte number if we consider codes from CRC-8 to CRC-32 (used in Ethernet (Association, 2018), of generator polynomial $g(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$). To retrieve error positions, two strategies are used:

Syndrome	Positions		
0000 0000 0000 0000 0000 0111	0	1	2
0000 0000 0000 0000 0000 1011	0	1	3
0000 0000 0000 0000 0001 0011	0	1	4
0000 0000 0000 0000 0010 0011	0	1	5
...			
1111 1101 0010 1110 1011 0100	567	2504	8956
0110 0101 0001 0110 1111 0011	567	2504	8957
0101 0101 0110 0000 0010 0110	567	2504	8958
...			
1111 0010 0010 1101 0000 0101	11996	11997	11998
0011 0111 1101 1000 1111 1111	11996	11997	11999
0101 0101 0010 0010 0000 0010	11996	11998	11999
1110 0100 0101 1100 0101 0001	11997	11998	11999

Figure 3.11 Examples of the explicit design of a lookup table containing all triple-error patterns for packet length up to 12000 bits

- The first one is to explicitly store the error positions associated with the syndrome in the table, as numbers coded on 16 bits (2 bytes) for each entry. There are N such numbers per row. Using this lookup table design, the required memory, denoted B_{exp} , is expressed as:

$$B_{exp} = \binom{M}{N} \times [\text{length}(\mathbf{s}) + (2 \times N)] \quad (3.10)$$

where M is the total length of the packet, N is the number of errors considered, and $\text{length}(\mathbf{s})$ is the size in bytes of the syndrome associated with the CRC used. The expression $(2 \times N)$ is the representation of N 2-byte numbers per row, representing the positions of the N errors considered. This implementation allows finding directly the error patterns associated with the syndrome but at a significant memory cost.

- The second strategy uses an implicit error position. With this approach, the lookup table does not need to store N 2-byte numbers per entry, which reduces the total memory requirements by up to 9 times when considering a CRC-8 and four errors, as compared to the aforementioned strategy. The memory requirements, denoted B_{imp} , can now be expressed as:

$$B_{imp} = \binom{M}{N} \times \text{length}(\mathbf{s}) \quad (3.11)$$

However, such a strategy involves more calculations to update the error pattern corresponding to the syndrome as it navigates through the table.

Table 3.1 Memory requirements for storing the lookup tables considering a payload of 1500 bytes for several CRC lengths and number of errors considered with implicit and explicit error positions

N	CRC-8		CRC-16		CRC-24		CRC-32	
	Implicit	Explicit	Implicit	Explicit	Implicit	Explicit	Implicit	Explicit
1	12 kB	36 kB	24 kB	48 kB	36 kB	60 kB	48 kB	72 kB
2	72 MB	360 MB	144 MB	432 MB	216 MB	504 MB	288 MB	576 MB
3	288 GB	2.02 TB	576 GB	2.30 TB	864 GB	2.60 TB	1.16 TB	2.88 TB
4	864 TB	7.77 PB	1.73 PB	8.64 PB	2.59 PB	9.50 PB	3.45 PB	10.4 PB

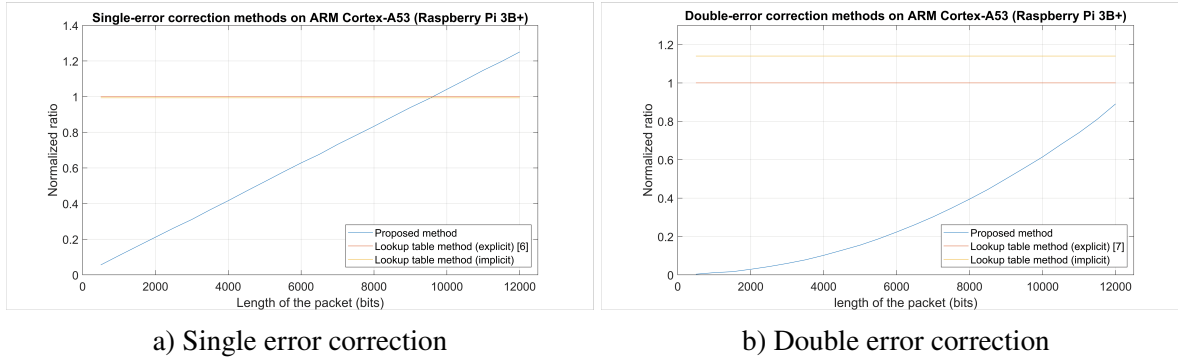


Figure 3.12 Evolution of the normalized time ratio to run the proposed method compared to lookup table-based approaches (explicit and implicit) for single and double error in the packet depending on the length of the packet. The normalized time ratio corresponds to $155\mu s$ in this case

We note that depending on the constraints present, one can choose among the two proposed designs to either save memory storage or save computations at the receiver side. Table 3.1 illustrates the memory requirements for both explicit and implicit implementations when considering large packets of 1500 bytes. We can see a significant increase in the memory requirements when considering each additional error. For such a packet length, considering three or four errors using a lookup table approach would be intractable.

3.4.3 Computational Time Comparison

In terms of processing time, we ran the C implementation of the proposed algorithm for a single- and a double-error correction on a Raspberry Pi model 3B+ (Foundation, 2018). For comparison purpose, we also implemented a table approach capable of considering every single-error position for packets up to 1500 bytes. We executed both algorithms for packets of different lengths, from a few bits to the maximum size available here, set to 1500 bytes. The Raspberry model 3B+ used to conduct the experiment is equipped with a System on a Chip (SoC) Broadcom BCM2837BO with an ARM Cortex-A53 quad-core processor at 1.4 GHz and 1 GB SDRAM LPDDR2.

Figs. 3.12a and 3.12b show the relative time for the error correction method on single- and double-error patterns, respectively. The proposed method's complexity is compared to both lookup table approaches (i.e., explicit and implicit) for different packet sizes. Lookup table-based approaches have a constant complexity since they must always check every entry of the table prior to conducting error correction to ensure that all candidates are identified. When considering a single error, both table-based approaches are of equal complexity, and the conversion from explicit to implicit is straightforward. For two errors, however, the implicit method is 10 to 15% slower due to the computations required to convert the table index into error positions. We note that for a large payload, the methods are similar in terms of computational complexity. The lookup table approach is still faster when considering single errors in large packets. However, as the packet becomes smaller with respect to the maximum allowed packet size, the proposed method surpasses the lookup approach due to its adaptability to the received packet size. The method can be more than 10 times faster than lookup methods for very small packets, and the gain in speed is even greater when double-error correction is considered. When used as part of a standalone error correction process, the algorithm performs at its maximum in terms of both correction rate and complexity for small packets or CRC-protected headers. In such cases, the SCR is significantly high or at a maximum for multiple-error correction.

Comparing the proposed algorithm to the lookup table approaches in the literature, we can verify that it provides improved capabilities in two main respects:

- **Flexibility.** Our method is more flexible than fixed-length lookup tables since it is not based on a specific packet size, but rather, is dynamically applied to protected data, and is thus adaptive to the data length. Consequently, the method will provide full coverage for any packet length. Furthermore, any generator polynomial, apart from the input parameter can be used with the proposed algorithm without modification. Lookup tables must be entirely recomputed when the generator polynomial considers changes. Alternatively, a table should be stored for each generator polynomial of interest, which significantly increases memory requirements.

- **Memory-free multiple-error correction.** The proposed method does not assume that only single errors are likely to occur. Even if this scenario can still be supported by setting the number of errors to 1 as the input parameter, we can also assume that up to $N > 1$ errors are possible and consider the whole set of possible candidates up to this number. A lookup table approach is able to list such error patterns but needs an intractable amount of memory storage to consider the whole set of N -error cases in large packets. In order to optimize the management of the number N , further work can be carried out to dynamically choose it by extracting information about channel conditions, such as the channel Signal-to-Noise Ratio (CSNR) estimation at the physical layer or the RSSI. The received RSSI level can be mapped to the crossover probability by measuring an average bit error rate (BER) for each RSSI level. If this BER estimation is low enough in terms of the length of the packet, we can set the parameter N to be 1.

3.4.4 Application to IoT

Considering the high performance of our method on small packets protected by strong generator polynomials, applying the proposed algorithm to the IoT domain can be highly desirable. A study of error distribution in a real environment of CRC-protected packets applied to the IoT (Al-Fuqaha *et al.*, 2015) domain is proposed in (Tsimbalo *et al.*, 2016). The authors consider both BLE packets protected by a CRC-24 and IEEE 802.15.4 (Association, 2011) packets protected by CRC-16-CCITT. Two packet sizes, 21 bytes and 39 bytes, are considered. The results of the experiments are represented in Table 3.2 which shows that over 50% of the erroneous packets contain fewer than three errors in any selected scenario. Moreover, more than 40% contain two errors or less, making it an ideal context to evaluate our proposed method's performance. As noted the authors of (Tsimbalo *et al.*, 2016), considering only slightly damaged packets can thus still enable a significant recovery rate. When soft information is unavailable, the authors of (Tsimbalo *et al.*, 2016) propose to use a received packet's RSSI to determine the BER.

Table 3.2 Error distribution in real environment for BLE and IEEE 801.15.4 and two packet sizes

Number of bit errors	BLE		IEEE 802.15.4	
	21B	39B	21B	39B
1	18%	16%	11%	10%
2	28%	27%	30%	27%
3	12%	11%	15%	16%
>3	42%	46%	44%	47%

In (Tsimbalo *et al.*, 2016), the authors present the average correction rate of their methods when a specific number of errors occur in the packet. The simulation results can be seen in Fig. 3.13, considering three payload sizes: 8 bytes, 21 bytes and 39 bytes. To compare our algorithm with these approaches, we tested an exhaustive set of error patterns for each size and each number of errors to get the average correction rate over all possible error cases. We applied the algorithm for each error case and checked the resulting list at the end of the process. The correction is considered successful only if the actual error pattern is the only candidate in the output list. If there are no or several candidates in the list, the packet is considered lost. For the ADMM and BP (Tsimbalo *et al.*, 2016), the simulation results in Fig. 3.13 show a maximum correction rate for single-error correction for all methods considered. For double-error correction, only the proposed method is able to achieve a 100% error correction. ADMM can correct an average of 80% for 8-byte payloads, which falls to less than 25% for 39-byte payloads. The results considering three errors are even more significant. The proposed method offers a 100% error correction rate for 8-byte payloads, whereas both ADMM and BP achieve 25%. When the payload length increases, the proposed method still can correct 86% and 47% for 21- and 39-byte payloads, respectively. Other methods can achieve a maximum of 5% error correction for such payloads.

These results can be retrieved in the SCR Fig. 3.10, where the three vertical bars correspond to the three payloads considered here. We can see that the correction rate for more than three

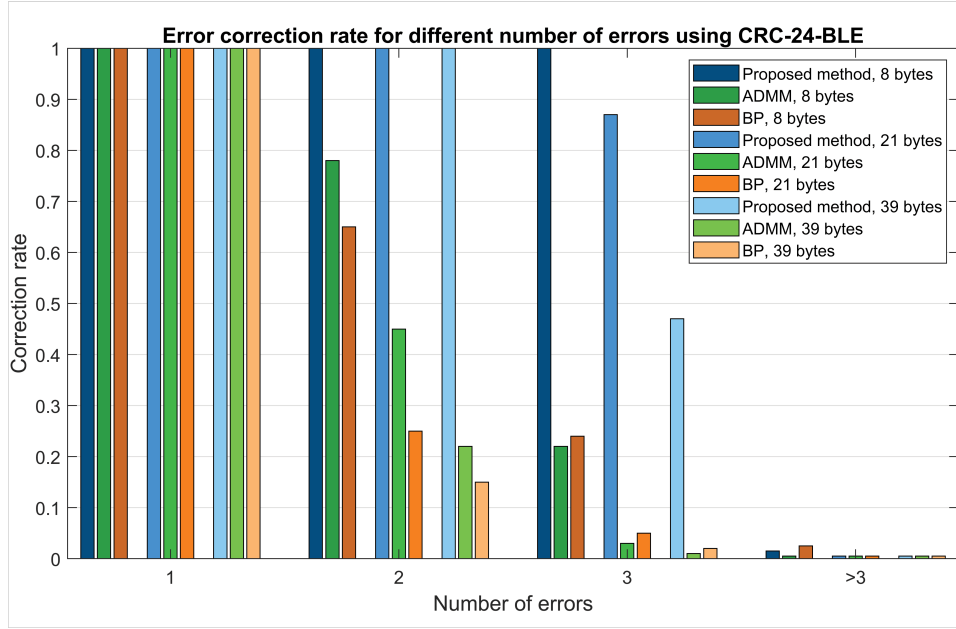


Figure 3.13 Error correction rate of the proposed method compared to two methods recently proposed in (Tsimbalo *et al.*, 2016) for different number of errors in the packet

errors is very low for all methods. In fact, it involves considering every error pattern containing more than three errors, which leads to a poor ratio since as the number of errors considered increases, the SCR decreases, becoming zero for large numbers of errors. However, we can note that we can still operate on four errors for small packets, as illustrated in Fig. 3.10 for 8-byte payloads, where the SCR is still 78%.

In (Tsimbalo *et al.*, 2016), the authors propose a configurable iterative decoding process, which means that its performance will depend on the number of iterations performed on the corrupted packet. The results provided here consider 1000 iterations at the decoder. The timing for this decoding applied to the fastest method (ADMM) takes an average of 85 ms for 21-byte packets on a desktop computer with an Intel i7 3.1 GHz CPU, 8 GB RAM and Microsoft Visual C++ 2010 Compiler. We tested our method on a desktop computer with an Intel i7 3.4 GHz CPU, 8 GB RAM and GCC compiler, and we noted that depending on the number of errors to consider, it takes an average time ranging from 2 μ s for single-error correction to 8 ms for three errors or less. Double-error correction takes an average of 150 μ s. Therefore, the proposed

method not only allows dramatically correcting more double- and triple-error cases, but it is also significantly faster than the state-of-the-art methods presented in (Tsimbalo *et al.*, 2016).

3.4.5 Future Work

In this paper, we have considered our algorithm as a standalone process that can only correct a packet when its output list contains a single element. In order to further increase the proposed method's error correction performance, it can be jointly used with other methods providing a list of potential error patterns as their output. For example, the work on UDP checksum proposed in (Golaghazadeh *et al.*, 2017a, 2018, 2017b) can be combined with our algorithm. Crosschecking both candidate lists would generate a matching list with a reduced number of entries. If our method is used in addition to the UDP checksum method, greater protected data lengths or a higher number of errors can be targeted for applications such as error correction on Ethernet frames, where a CRC covers the entire packet. Similarly, we could eliminate candidates leading to wrong values of known protocol fields, such as constant and predictable fields in the protocol's header (reserved and version fields are constant values during a communication, and some fields such as the sequence number in RTP are predictable since they are increased by 1 at each new packet throughout the communication). Some methods which consider a MAP approach have already proposed to use a CRC lookup table to validate their reconstruction, as described in (Niu & Chen, 2012) on Polar codes (Arikan, 2009). It could be beneficial to compute only the probability of valid candidates rather than considering the whole set of possible sequences, determining their probability of being sent, and finally checking their CRC compliance.

3.5 Conclusion

In this work, we have proposed a novel algorithm to correct transmission errors within data covered by a CRC, using the computed non-null syndrome at the receiver. This method is able to instantly correct single errors if the protected data length does not exceed the period of the generator polynomial. This method is also able to correct multiple errors in small-sized packets,

as used in the BLE standard. In such an environment, the proposed method achieves better error correction rates than the state-of-the-art methods considering up to three errors in the packet. The standalone error correction rate in BLE is at a maximum for single-, double- and some triple-error cases presented.

When instant correction is not possible, the algorithm still generates the list of all the possible error patterns that lead to the computed syndrome, according to a maximum number of errors considered. This list is usually small if we consider a reasonable number of errors. Further work to improve this method should use it in addition to existing methods that output a list of candidates. Crosschecking the lists of different methods would reduce the number of valid candidates, which would lead to fewer sequences to test or even to a reduction of the list size to a single candidate, allowing instant correction of damaged packets.

CHAPTER 4

ENHANCED CRC-BASED CORRECTION OF MULTIPLE ERRORS WITH CANDIDATE VALIDATION

Vivien Boussard^{1,2}, Stéphane Coulombe¹, François-Xavier Coudoux², Patrick Corlay²

¹ Département de génie logiciel et des technologies de l'information,
École de technologie supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3

² Univ. Polytechnique Hauts-de-France, CNRS, Univ. Lille, ISEN, Centrale Lille, UMR 8520 -
IEMN - Institut d'Électronique de Microélectronique et de Nanotechnologie, DOAE
Département d'Opto-Acousto-Électronique, F-59313 Valenciennes, France

Article submitted to Signal Processing: Image Communication in March 2021

4.1 Abstract

Cyclic redundancy checks (CRC) are widely used in transmission protocols to detect whether errors have altered a transmitted packet. It has been demonstrated in the literature that CRC can also be used to correct transmission errors. In this paper, we propose an improvement of the state-of-the-art CRC-based error correction method. The proposed approach is designed to significantly increase the error correction capabilities of the previous method, by handling a greater part of error cases through the management of candidate lists and using additional validations. Simulations and results for wireless video communications over 802.11p and Bluetooth Low Energy illustrate the Peak Signal-to-Noise Ratio (PSNR) and visual quality gains achieved with the proposed approach versus the state-of-the-art and traditional approaches. These gains range from 0.8 dB to 5.7 dB over Bluetooth Low Energy channels with Eb/No ratio of 10 dB and 8 dB, respectively.

4.2 Introduction

Transmission of compressed video content over unreliable channels often results in quality reduction at the viewer side. Even a slightly corrupted video sequence can suffer from severe

visual artifacts after video reconstruction of discarded regions, which decreases the viewing experience of the final users. In order to prevent quality deterioration, error concealment tools have been proposed in the literature. However, as the reconstructed area is often interpolated from spatially and/or temporally neighboring visual content, the quality of the results depend on the video content and the size of the lost regions. Other solutions to recover the originally transmitted packets include retransmission and error correction.

When transmitting video sequences in wireless environments such as vehicular and low energy networks, retransmission is not recommended and often not available to handle missing packets, due to delay constraints, and/or network congestion. As widely used protocols use a CRC (Sobolewski, 2003) to detect errors in the entire packet, it might be useful to use it for error correction purposes as well.

The CRC is first computed at the transmitter as the remainder of the long division of the protected data left shifted by n positions by the generator polynomial, as expressed as follows:

$$\frac{d_T(x) \ll n}{g(x)} = q(x) + \frac{r_T(x)}{g(x)} \quad (4.1)$$

where $r_T(x)$ is the remainder of the division, $d_T(x)$ is the protected data, $g(x)$ is the generator polynomial of degree n , and $+$ is the addition which corresponds to an *exclusive or* (XOR) between two binary polynomials¹. The remainder $r_T(x)$ is then appended to the protected data to produce the transmitted packet $p_T(x)$, such that $p_T(x) = (d_T(x) \ll n) + r_T(x)$. At the receiver side, the integrity of the received packet $p_R(x) = (d_R(x) \ll n) + r_R(x)$ is checked through another long division of the protected data appended by the remainder computed at the transmission by the same generator polynomial:

$$\frac{(d_R(x) \ll n) + r_R(x)}{g(x)} = q(x) + \frac{s(x)}{g(x)} \quad (4.2)$$

where $s(x)$ is the new remainder of the division, also known as the syndrome. If no error occurs during the transmission, such syndrome $s(x)$ is null. Otherwise, a non-null value of

¹ Indeed, binary packets of length m belong to the Galois Field (GF) $GF(2^m)$ where the addition is performed as the bitwise XOR (Luo *et al.*, 2012).

the syndrome indicates that one or several errors altered the transmitted packet. The standard handling of non-null syndrome is the discarding of the corrupted packet (UDP protocol) or the retransmission request of the corrupted packet (TCP protocol).

Several methods have been proposed in the literature to handle missing and corrupted packets. Some approaches are designed to reconstruct the missing video content by taking advantage of the correctly received video information. They can be classified as two main categories: spatial error concealment methods (Sun & Kwok, 1995; Koloda *et al.*, 2013; Liu *et al.*, 2015; Akbari *et al.*, 2017), that use the available information in the current frame to reconstruct the missing video areas, by interpolating and predicting the corrupted video content. Other methods propose to conceal the video chunks through temporal concealment (Wu *et al.*, 2008; Lam *et al.*, 1993; Chen *et al.*, 2003; Chung & Yim, 2020), which takes advantage of the temporally neighboring frames to recover video content. As temporal correlation exists in natural video contents, the motion vector from the missing part of the frame can be predicted from previous frames. Error concealment algorithms can also combine both spatial and temporal concealment to achieve more accurate results (Atzori *et al.*, 2001; Chen *et al.*, 2008). Traditional error concealment mainly use interpolation from neighboring content to reconstruct the video, while the most recent error concealment solutions use machine learning to recover large missing areas (Sankisa *et al.*, 2018; Kim *et al.*, 2019; Wang *et al.*, 2019). Such methods are able to handle missing packets as they systematically discard corrupted packets to perform error concealment. The drawback of such process is the loss of useful information. Indeed, a corrupted received packet may have been hit by only a few errors, and still contain valuable information that can help to reconstruct the missing video data.

Another category of methods from the literature propose to perform error correction on the received corrupted packets, based on the available information. Such approaches can search for the most probable sent bit sequence given reliability information on the packet and/or syntax knowledge of the transmitted packet (Moonseo Park & Miller, 2000; Caron & Coulombe, 2013). Some state-of-the-art solutions also propose to take advantage of error detection codes as error correction codes. Checksums have been investigated in such matter to identify bit error patterns from an erroneous checksum value (Golaghazadeh *et al.*, 2017a). CRC-based error correction

has also been proposed, through the use of lookup tables (LUT) (Shukla & Bergmann, 2004; Babaie *et al.*, 2009; Aiswarya & George, 2017) to identify the error position based on the storage of the syndrome corresponding to a list of error cases. In a previous work, we proposed a CRC-based error correction method using arithmetic operations (Boussard *et al.*, 2020b,a) that generates the list of possible error patterns containing no more than a predefined number of errors, given the computed syndrome at the receiver and the used generator polynomial. This paper is mainly based on the theory described and discussed in (Boussard *et al.*, 2020a) and the reader is expected to be familiar with this work.

In this paper, we propose a method to perform multiple error correction based on the CRC syndrome with candidates validation on Advanced Video Coding AVC and High Efficiency Video Coding HEVC encoded video content applied to widely used wireless communication applications. The contributions of this paper are:

- **Enhanced multiple error correction:** The method proposed in this paper offers the possibility to perform error correction on a corrupted packet when several valid candidates are considered as CRC-compliant (candidates with the same syndrome), while state-of-the-art method performed error concealment in such case.
- **Processing speed gains:** We propose to reduce the processing time of the CRC error correction method by detecting unnecessary loops based on the syndrome and generator polynomial parities.
- **Storage gains:** we propose a memory management of the error search to have a fixed low memory storage, independent from the packet length.
- **Demonstration of gains in wireless transmissions:** we prove the usefulness of the applicability CRC-based error correction method in practical applications by testing the method using realistic video communication scenarios over wireless networks. We applied the proposed method to wireless communication in 802.11p and Bluetooth Low Energy environments. In the latter case, the proposed approach offers significant gains in PSNR, up to 10 dB compared to error concealment methods.

This article is structured as follows. In section 4.3, we expose related works on CRC error correction found in the literature. In section 4.4, we propose a novel method to enhance the error correction capabilities of CRC codes with additional validation steps and optimization strategies. In section 4.5, we discuss the simulation results for different environments such as Wi-Fi 802.11p and Bluetooth Low Energy. In section 4.6, a conclusion is conducted and work perspectives are discussed.

4.3 Related works

4.3.1 CRC-based single error correction using tables

Error correction using the CRC syndrome has already been proposed in (Shukla & Bergmann, 2004; Babaie *et al.*, 2009; Aiswarya & George, 2017). In these methods, a lookup table is created prior to the communication, in which each possible non-null error syndrome is listed along with its associated error pattern. When receiving a corrupted packet, the syndrome list is scanned to find a match with the computed syndrome of the corrupted packet. If a match is found, the bits at the associated positions are flipped. If no match is found, the packet is discarded. Such method has several drawbacks: first, the lookup table must be recomputed if the maximum packet length changes. The method only works for a specific generator polynomial and the entire list is always scanned. Such method is still however considered to help the error correction of polar codes as demonstrated in (Niu & Chen, 2012; Liu *et al.*, 2018).

4.3.2 CRC-based single error correction using arithmetic operations

More efficient methods have been recently introduced to handle single error correction using the CRC syndrome. In (Boussard *et al.*, 2020b), we proposed a method consisting in using arithmetic operations on the CRC syndrome to correct damaged packets at the receiver. The method proposes to correct single error packets through CRC error correction. To perform error correction, this method uses the definition of CRC codes to identify all the possible single error positions given the computed syndrome at the receiver. The method lies on the definition of the

CRC computation as illustrated in (4.2). We can express the syndrome $s(x)$ at the receiver as:

$$s(x) = p_R(x) \bmod g(x) \quad (4.3)$$

where $p_R(x)$ is the received packet and is equal to the transmitted packet $p_T(x)$ potentially corrupted by errors: $p_R(x) = p_T(x) + e(x)$, where $e(x)$ corresponds to the error pattern that hit the packet. If no error occurred, we verify that $p_R(x) = p_T(x)$ and, as $p_T(x) \bmod g(x) = 0$ since the transmitted data is designed to produce a null syndrome, $s(x) = 0$. If an error occurred, we get $s(x) = e(x) \bmod g(x)$, that can also be expressed as:

$$e(x) = s(x) + q(x).g(x) \quad (4.4)$$

At each step, the number of remaining non-null position in the error vector is checked and if this number equals 1, a single error candidate is identified at that position. An example of such a single error method is provided in Fig. 4.1, applied on CRC-4-ITU with generator polynomial $g(x) = x^4 + x + 1$ over 10 data bits. In such example, the computed syndrome at the receiver is $s(x) = x^2 + 1$. We can observe that the first step is to initialize the error vector \mathbf{e} as zeros and set the LSBs to the value of the computed syndrome. From this stage and at each step, we count the number of non-null positions in the error vector as they represent the error positions. At the initialization step, we can observe that such value is equal to 2. As we are searching for error patterns containing only a single error, we do not consider such pattern as a valid single error pattern. Thus, we aim at canceling the LSB non-zero value by performing an XOR at such position. The resulting error vector contains 3 non-null positions and is still not a valid candidate for single error correction. By continuing such steps, we find a valid candidate at step 4, where there is a single non-null position in the error vector, at position x^8 . We ensure that there is no other candidate by scanning the whole packet length with such method. At the end of the process, we observe the candidates' list. In the case illustrated, the list contains a single entry, which is a single error at position x^8 . This means that if a single error occurred in the packet, it occurred at position x^8 . To correct the packet in such case, the bit at the corresponding position

in the received packet must be flipped. This method used alone has some limitations since it

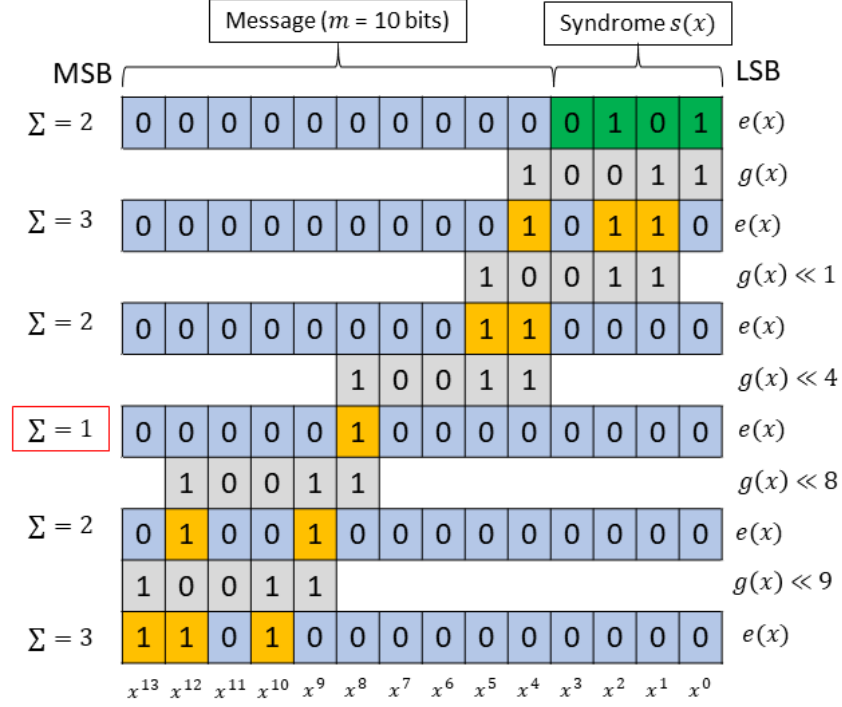


Figure 4.1 Example illustrating the single error correction method as described in (Boussard *et al.*, 2020b), using a CRC-4-ITU with generator polynomial $g(x) = x^4 + x + 1$ over 10 data bits, when the computed syndrome at the receiver is $s(x) = x^2 + 1$.

does not handle packets subject to several errors and does not validate the reconstructed packet. Thus, if a miscorrection occurs, which can happen if the received packet is highly corrupted, the packet can be unfortunately sent to the application even if it still contains errors. In compressed video transmissions, a single erroneous bit can lead the decoder to crash when receiving such corrupted packet, due to desynchronization.

4.3.3 Validation of CRC-based error correction

The work in (Golaghazadeh *et al.*, 2018) illustrates that there are bits in a compressed video stream that will cause desynchronization if hit by an error. A simple example of such so-called desynchronization bits is Exp-Golomb codes. In such codes, a prefix (i.e. several bits set to

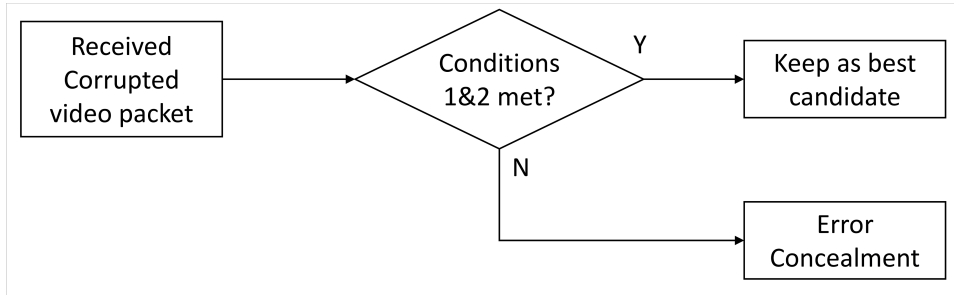


Figure 4.2 The system proposed in (Golaghazadeh *et al.*, 2018) allows a validation of the reconstructed bitstream based on a two step validation process.

0 followed by a single bit set to 1) indicates the length of the codeword to be read. It is clear that if an error occurs in such prefix, the bit length of the next codeword will be wrong and propagate as the packet is decoded. Such behavior is called desynchronization of the bitstream and makes the bit sequence undecodable in most cases. Arithmetic coding used in modern video compression standards AVC (Wiegand *et al.*, 2003), HEVC (Sullivan *et al.*, 2012) and Versatile Video Coding (VVC) (Bross *et al.*, 2021) is highly vulnerable to desynchronization.

As miscorrection can lead to desynchronized streams at the video decoder, the authors of (Golaghazadeh *et al.*, 2018) proposed to validate the reconstruction through a two-step process for H.264 encoded video sequences, as illustrated in Fig. 4.2. The two validation operations to validate compliance are:

- Check the number of macroblocks (MBs) in the packet (the authors configure the encoding of the video sequence to have a constant number of MBs in each packet).
- Check the decodability of the reconstructed video stream. Each candidate is thus tested and if it makes the video decoder crash, the next candidate is tested.

If the entire set of candidates is tested and no one met the two validation conditions, the error correction is aborted and an error concealment method is applied on the missing area. This method ensures that the decoding process can be completed even if there are still errors in the decoded video packet but it does not ensure that the reconstructed packet is the transmitted one.

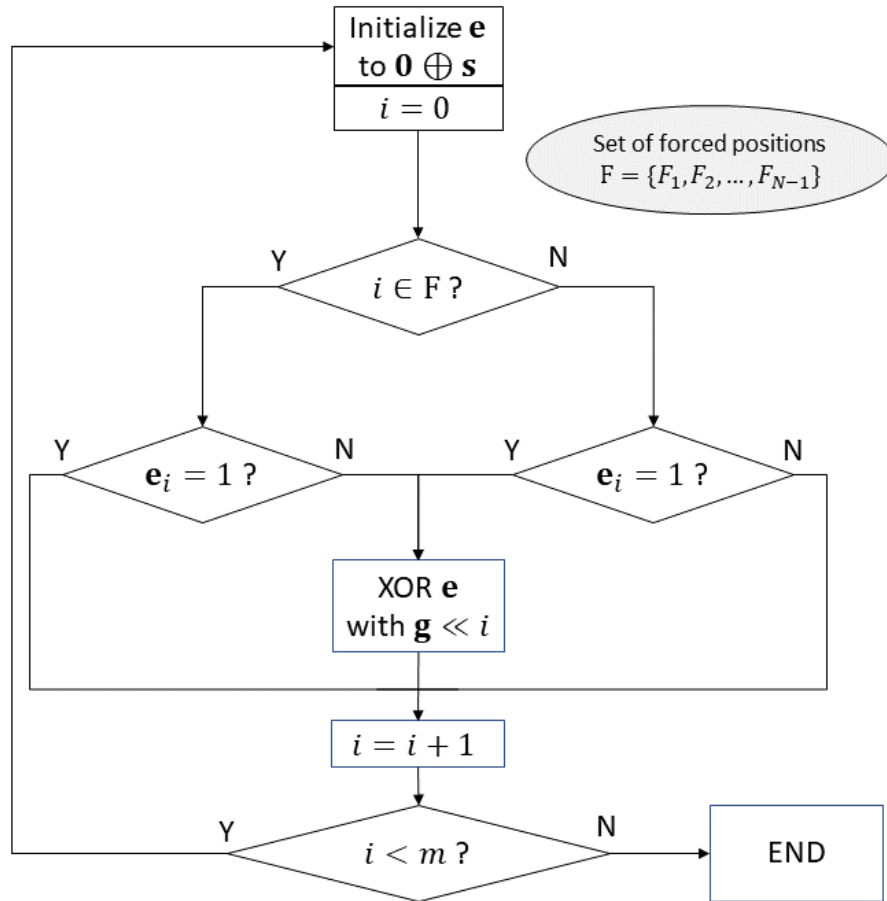


Figure 4.3 Illustration of each bit position management for multiple bit error correction using the CRC syndrome. The error vector \mathbf{e} is initialized to zero and its LSB are set as the syndrome \mathbf{s} . For each position, the current value is checked and the procedure depends on the elements in the set of forced position \mathcal{F} . The current position is flipped if it has to be forced and is currently 0 or has to be canceled and is currently set to 1.

4.3.4 CRC-based multiple error correction

It is possible to handle multiple error correction by simply extending the Look Up Table (LUT)-based methods as proposed in (Shukla & Bergmann, 2004). Such extension consists in storing, in a lookup table, the whole list of error patterns for the considered number of errors, denoted N , and their associated error syndrome. Such approach would lead to list all the error patterns containing N errors or less. However, the size of such tables, when the value of N increases, would rapidly become intractable. Depending on the number of errors N to consider,

m the payload length and $\text{length}(\mathbf{s})$ the byte length of each syndrome, the size in bytes of such table is expressed as follows:

$$\text{TableSize} = \binom{m}{N} \times [\text{length}(\mathbf{s}) + (2 \times N)]. \quad (4.5)$$

We consider here that the error positions are stored as 2 bytes numbers for each pattern. Such approach would thus require 2.6 TB of storage when using a CRC-24 used in BLE and a packet length of 1500 bytes to handle 3-error correction.

A more practical extension of the CRC error correction method to handle multiple error patterns was proposed in (Boussard *et al.*, 2020a), where the presented approach was designed to output the exhaustive list of CRC-compliant candidates that contains a defined number of errors N or less. The strategy in this approach is to force bit positions throughout the process. As the single error correction method cancels, at each step, the LSB non-null value, it is clear that it cannot consider error patterns that have multiple errors widely distributed throughout the whole bit sequence (i.e. multiple error patterns where all the errors are not within the error range). By forcing bit positions in an increasing order, the algorithm can handle such error cases by selecting positions of bits that will remain non-null during the process, by passing through or XORing \mathbf{g} at such position, depending on the initial value of the forced position. Hence, the algorithm aims at forcing $(N - 1)$ bit positions and performing a single error search on the remaining length of the packet. Such method can be illustrated as in Figure 4.3, where the set of forced positions is denoted as \mathcal{F} and contains the index of the $(N - 1)$ values to force to 1. At every step, the current position is checked and an XOR is performed when the position has to be forced but is currently to 0 or when the position must be canceled and is currently set to 1. The current bit position is then increased by 1. At every loop, the number of non-zero values remaining in the error vector is counted, and if this number is equal or less than N , a candidate is appended to the list, with error positions identified at the non-null positions of error vector \mathbf{e} . Once the entire packet length has been processed with the current set of forced positions, \mathcal{F} is updated, until all forced positions are tested.

By doing so, all the bit error patterns can be considered and the complexity remains lower than a

brute force scheme, which would test every possible error patterns and conduct to $\sum_{i=1}^N \binom{m+n}{i}$ operations, where $m + n$ is the total length of the packet.

This method significantly increases the error correction capability compared to the single error approach by handling more error cases and doesn't require a significant amount of memory. However, by increasing the number of errors to consider, the average candidates list's size increases as well. In (Boussard *et al.*, 2020a), the SCR was defined as the percentage of candidate lists containing a single entry as a function of the length of the packet and the number of errors to consider, as expressed in (4.6):

$$\text{SCR}(m, N) = \frac{\text{SinglePatterns}(m, N)}{\text{TotalPatterns}(m, N)} \quad (4.6)$$

where $\text{SinglePatterns}(m, N)$ is the number of error cases producing a single candidate as output, considering all N -error pattern cases for a message length of m , and $\text{TotalPatterns}(m, N)$ is the total number of individual N -error patterns for a message length of m bits.

When considering single error correction, it was shown that the SCR remained 100% for packets of length less than the *cycle* of the generator polynomial used. As the *cycle* length for CRC-24 and CRC-32 is greater than the maximum packet size tolerated in the communication protocol, it guarantees single error candidates as output of the single error method. Some examples of cycle lengths for common generator polynomials are illustrated in Table 4.1.

The authors observed that for commonly used 3 or 4 bytes generator polynomial, as for the CRC-24 used in BLE or the CRC-32 used in the Ethernet protocol, the ratio remains at 100% up to a reasonable packet length when considering single and double error cases. However, even strong CRCs will output candidate lists with many entries when considering that a high number of errors may have occurred in the packet. The choice for the maximum number of errors (i.e. parameter N) in the error patterns of the candidates list should thus be determined according to the expected channel conditions.

Anyway, these methods still suffer from some problems. Even if we set the number of errors to consider as a small value (less than or equal to 3 for instance), there will be a huge number of

Table 4.1 Cycle lengths for widely used CRCs: CRC-8-CCITT with $g(x) = x^8 + x^2 + x + 1$, CRC-16-CCITT with $g(x) = x^{16} + x^{12} + x^5 + 1$, CRC-24-BLE and CRC-32.

	CRC-8	CRC-16	CRC-24	CRC-32
Cycle length	$2^7 - 1$	$2^{15} - 1$	$2^{23} - 1$	$2^{32} - 1$

error cases that the previously proposed methods would not be able to handle. As soon as there are several candidates in the list, these methods will fail at performing error correction since they cannot determine which candidate is the correct (originally transmitted) one. Validation steps to help choose between candidates were proposed in the literature for checksum-generated error pattern lists (Golaghazadeh *et al.*, 2017a).

By definition of the proposed approach, all the candidates in the list pass the CRC check, which means that we cannot use it anymore to differentiate the candidates and/or select the best candidate. If we consider a random uniform error distribution, we might favor candidates with fewer errors as they are more likely to occur but this exposes us to miscorrections.

In what follows, we propose an enhanced approach that allows handling the candidate's list and remove bad candidates, in order to increase the error correction capability compared to previous methods while also ensuring the validity of the candidate.

4.4 Proposed method

State-of-the-art multiple bit error correction exhibits respectable error correction capabilities but suffers from several issues. First, increasing the number of error considered N greatly increases the computational complexity of the method, which can lead to significant processing time. Thus, proposing solutions to reduce such complexity would allow the method to handle multiple error in practical scenarios. Moreover, a high number of errors considered yields a higher number of candidates in the list. As the CRC is used to produce the candidates' list, it is obvious that it cannot be further used to identify bad candidates in the list as they are all CRC compliant (i.e. all the error patterns in the candidates' list correspond to the received computed

syndrome). In the proposed approach, we address such issue with additional validation steps on the candidates.

4.4.1 List handling: validation of candidates

In order to identify the actual error pattern among a list of candidates, the strategy is to spot bad candidates and remove them from the list, given that CRC can no longer be used to sort or invalidate candidates at this point.

However, a cross-layer approach using the UDP/TCP checksum (Braden *et al.*, 1989) can be used to drastically reduce the number of candidates in the list. By computing the checksum over all the candidate error patterns resulting of the CRC error correction method, we can identify the candidates that pass both the CRC and checksum tests to filter the candidates list, as illustrated in Fig. 4.4. At the reception of a corrupted packet, the multiple error correction method is performed and outputs the complete list of CRC-compliant error patterns (i.e. the error positions) up to N errors. If this list is empty, error concealment is applied. If not, we test each candidate by computing the checksum on the reconstructed bit sequence for each error pattern from the candidates list. The candidates that are both CRC and checksum compliant are kept in the list. If a candidate fails the CV, it is removed from the list as there are still errors in the packet. At the end of the process, if a single candidate remains in the list, it is considered as the winning candidate and the error correction is performed.

Computing the checksum will not require any additional resource and would not produce a large candidate list when combined with the proposed method, contrary to (Golaghazadeh *et al.*, 2017a). To illustrate the performance of the proposed cross-checking method, we take the example of a CRC-24 protecting a 250 bytes payload. In such case, when searching error patterns up to $N = 3$ errors, we can observe candidate lists containing at most between 150 and 200 candidates. These lists can contain multiple error patterns as well as single error patterns. In the proposed approach, we apply the same election process on multiple- and single-error patterns to ensure the validity of the reconstruction.

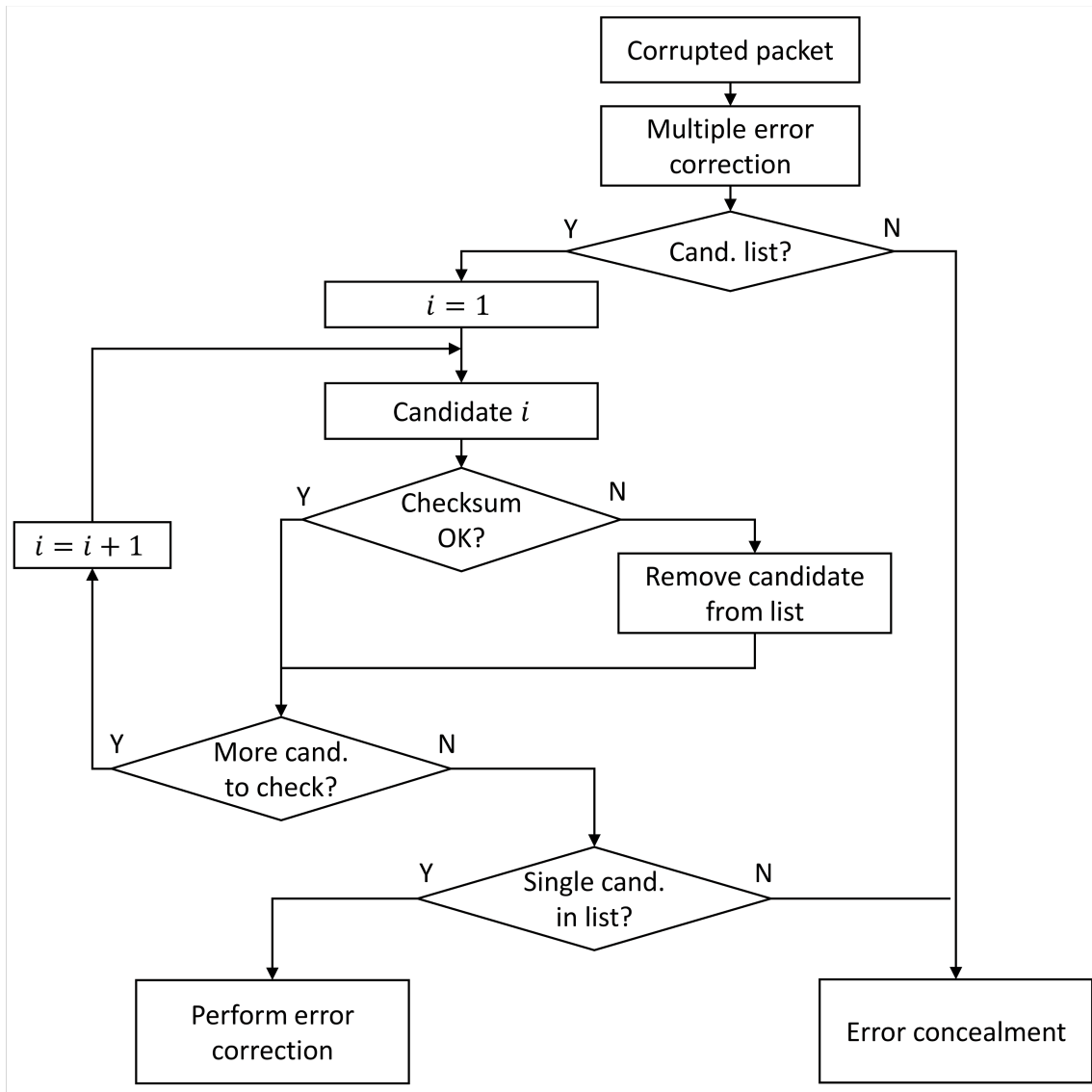


Figure 4.4 Flowchart illustrating the correction process using both CRC and checksum to reduce the number of candidates in the output list.

By simply computing the checksum on the candidate bit sequences, we observed that most of the lists were reduced to a single element, which removed the ambiguity of having several candidate error patterns. Although not all the triple error patterns resulted in a single candidate list after the crosscheck, we observed that the resulting number of candidates was significantly reduced, as shown in Table 4.2, where we compare the average candidates list size before and after CV in BLE environment, considering packets of 250 bytes at most and $N = 3$. In such case, the average

number of valid candidates goes from approximately 100 to less than 2 after the validation step. We observed at most 5 candidates in the final candidates list. For simplicity, in this paper, we only consider the checksum to demonstrate the benefits of adding validation steps to reduce the number of candidates. However, we can additionally validate the values of various application dependant fixed or predictable fields in the packet, such as the version in the IP protocol, source and destination ports as well as sequence numbers in the RTP protocol.

Whether there is a single or multiple candidates in the output list, we recommend, is in (Golaghazadeh *et al.*, 2018), to perform an additional step in the error correction process. Such step is the video decoding validation. To ensure the validity of the reconstructed video stream, the strategy is to try to decode the resulting candidate patterns. If the decoder crashes it means that the resulting sequence is still erroneous and moreover that the erroneous bits cause syntax error and/or desynchronization of the bit stream. In such case, the next candidate is tested. The first candidate to pass the video decoding test is considered as the winning candidate. Note that there is a probability that the packet still contains erroneous since at this point as we cannot detect errors in the video stream. However, as these errors will not cause desynchronization, we can let them pass as some study show that such errors produce less visual artefacts than concealing the whole lost packet (Trudeau *et al.*, 2011). The video validity check is performed even when there is only one candidate in the list as it is possible that many errors corrupted the packet. We can thus ensure that the correction is valid with this validation and reduce miscorrections. Such cross-layer design takes advantage from the error correction of the CRC at the link layer, the error detection capabilities of the checksum at the transport layer and finally the syntax check of the video decoder at the application layer.

Table 4.2 Average number of candidates before and after CV for different Bluetooth Low Energy channel qualities with $N = 3$.

	Eb/No value		
	10 dB	9 dB	8 dB
CRC \rightarrow CRC+CV	130.3 \rightarrow 1.62	98.04 \rightarrow 1.22	84.01 \rightarrow 1.07

4.4.2 Optimizing the method

Increasing the number of errors considered significantly increases the time required to perform error correction. It was shown that the computational complexity of the arithmetic method grows exponentially with N . Hence, the search for methods and strategies to reduce the time needed to perform the proposed method is important. Being able to know in advance the number of errors would help to save a lot of iterations and computational complexity.

4.4.2.1 Exploiting syndrome parity

The parity of both the generator polynomial and the syndrome can give valuable information on the number of errors in the video packet. Additional knowledge on the parity of the number of errors will help to skip most of the iterations that would not have produced any candidate.

In fact, the parity of widely used generator polynomials such as CRC-16-CCITT and CRC-24-BLE are even, which allows deducing information on the number of errors that occurred in the packet. To illustrate this, we can express the operation to update the error polynomial as:

$$\begin{aligned}
 e'(x) &= g(x) + e(x) \\
 &= \sum_{i=0}^n g_i \cdot x^i + \sum_{i=0}^{n-1} e_i \cdot x^i = g_n \cdot x^n + \sum_{i=0}^{n-1} (g_i \cdot e_i) \cdot x^i \\
 &\Rightarrow e'_n = g_n \text{ and } e'_i = (g_i \cdot e_i), \forall (n-1) \geq i \geq 0
 \end{aligned} \tag{4.7}$$

From this equation we can see that the updated value of $e(x)$, denoted $e'(x)$, will be affected by $g(x)$. In GF(2), the element 0 is neutral and will not affect the former value, e_i . However, when using addition +, the element 1 will always flip the former value as shows the addition truth table for GF(2) in Table 4.3. From such definition it is clear that k non-null coefficients in $g(x)$ will flip k values in the error polynomial. In such case, when k is odd, the parity changes from $e(x)$ to $e'(x)$ at every XOR performed, and when k is even, the parity of $e(x)$ and $e'(x)$ remains

the same for the whole process.

Let $\text{NbErr}(p)$ and $\text{Syn}(p)$ be the number of errors in a packet p of length $m+n$ and its computed syndrome of length n produced by $g(x)$, respectively. Considering the set of possible syndromes \mathcal{S} and the subsets \mathcal{S}_O and \mathcal{S}_E , containing the syndromes produced by an odd and an even number of errors in a packet, respectively, we can define:

$$\begin{aligned}\mathcal{S} &\triangleq \{s \in \mathbb{N} \mid 0 \leq s \leq (2^n - 1)\} \\ \mathcal{S}_O &\triangleq \{\text{Syn}(p) \mid 0 \leq p \leq 2^{m+n} \text{ and } \text{NbErr}(p) \text{ is odd}\} \\ \mathcal{S}_E &\triangleq \{\text{Syn}(p) \mid 0 \leq p \leq 2^{m+n} \text{ and } \text{NbErr}(p) \text{ is even}\}\end{aligned}\tag{4.8}$$

Clearly, $\mathcal{S}_O \subset \mathcal{S}$, $\mathcal{S}_E \subset \mathcal{S}$, $\mathcal{S}_O \cup \mathcal{S}_E = \mathcal{S}$ and

$$\mathcal{S}_O \cap \mathcal{S}_E = \begin{cases} \emptyset, & \forall g(x) \text{ with even parity} \\ \mathcal{S}, & \forall g(x) \text{ with odd parity} \end{cases}\tag{4.9}$$

Such knowledge is not sufficient to determine the exact number of errors in the packet but helps to reduce computational complexity by avoiding useless computation. Applied to triple error correction, the error correction method will have to test single and triple error candidates or only double error candidates, depending on the parity of the syndrome. The latter case saves a lot of computation as triple error correction requires the most computation time to be processed. We compared the processing time gains of considering the syndrome's parity applied to BLE

Table 4.3 Addition and multiplication truth tables for finite field $\text{GF}(2)$.

+	0	1
0	0	1
1	1	0

.	0	1
0	0	0
1	0	1

Table 4.4 Average processing time with (P) and without (NP) considering the syndrome's parity for different BLE channel qualities with $N = 3$.

	Eb/No value		
	10 dB	9 dB	8 dB
NP \rightarrow P	1.18 s \rightarrow 1.08 s	1.15 s \rightarrow 0.77 s	1.15 s \rightarrow 0.61 s

channels (SIG, 2013) in Table 4.4. We can see that the gains are increasing as the channel conditions decrease. Indeed, high quality channels will produce many error patterns containing a single error, where taking into account the parity will have little effect on the total processing time. When the average number of double-error patterns increases at channel Eb/No of 8 dB and 9 dB, the average processing time significantly decreases, down to half of the original processing time at Eb/No of 8 dB.

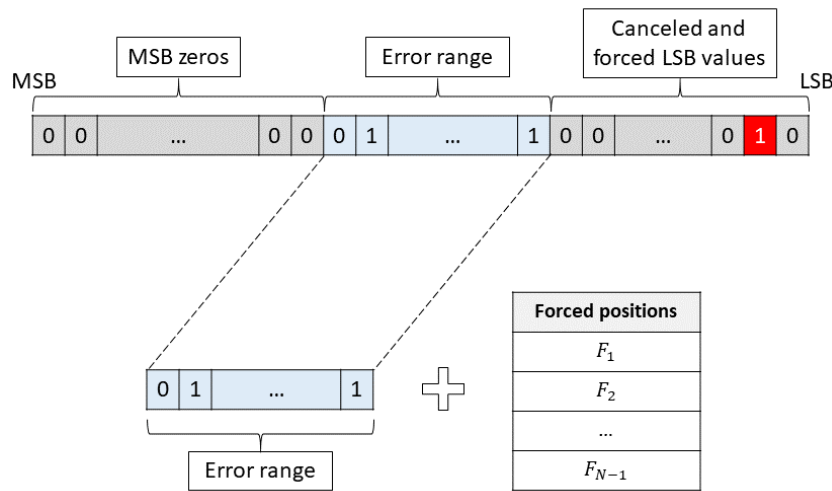


Figure 4.5 Illustration of the memory reduction induced by considering the error vector as only its error range. The set of forced positions must be stored in order to identify error positions if a candidate is found.

4.4.2.2 Reducing memory requirements

Another aspect that can be optimized is the memory required to store the error vector throughout the whole process. In (Boussard *et al.*, 2020a), we used an error vector with a size equal to the packet length to successively perform XORs and check the number of non-null coefficients at each step. We saw from (4.3) that the definition of the error polynomial is:

$$e(x) = s(x) + q(x).g(x) \quad (4.10)$$

which can be further expressed as:

$$\begin{aligned} e(x) &= \sum_{i=0}^{n-1} s_i.x^i + \sum_{j=0}^m q_j.x^j \cdot \sum_{i=0}^n g_i.x^i \\ &= \sum_{i=0}^{n-1} s_i.x^i + \sum_{j=0}^m \sum_{i=0}^n (q_j.g_i).x^{(i+j)} \end{aligned} \quad (4.11)$$

where m and n represent the lengths of the payload and syndrome, respectively. We remind here that the proposed method for searching error patterns is based on the building of $q(x)$, by adding $g(x)$, of degree n , to cancel LSB values of $e(x)$. From such definition, it is clear that for a given position j , (4.11) operates on the range from j to $j + n$. Moreover, as each operation cancels the bit at position j , the non-null values of the error polynomial $e(x)$ will range from position $j + 1$ to $j + n$. Out of this particular range, the values of $e(x)$ are either 0 due to initialization (for positions greater than $j + n$), or already canceled (for positions lower or equal to j), apart from the forced position (which are known at each step).

In (Boussard *et al.*, 2020a), we introduced the error range to explain the need to force bit positions throughout the process. It can be seen as a sliding window canceling every LSB non-null value. Without forcing bit positions, it was demonstrated that only patterns with multiple errors occurring within the error range can be identified, yielding the need for forcing positions to be able to identify every error case.

If we investigate further the error range, we can see that we can benefit from it to reduce the memory required to implement the method. By storing only the vector corresponding to the error

range, we are performing XORs on n -bit vectors instead of $(n + m)$ -bits, where m is the length of the protected data, generally far greater than n , the bit length of the generator polynomial. In order to avoid the problem raised by the error range, the forced bit positions must be known at any time, which were already the case since the forced bit positions were stored in the originally proposed multiple error correction algorithm. Instead of storing the entire vector with non-null LSBs only at forced bit positions, handling a vector with a bit length equal to the error range and storing the forced bit positions is more efficient for memory management. Such optimization is illustrated in Fig. 4.5 where we can see that the memory required to store the error vector is independent of the packet length and always corresponds to the length of the syndrome. The storage in bits needed for the state-of-the-art table approach, M_{LUT} , is:

$$M_{\text{LUT}} = \binom{m}{N} \times [\text{length}(\mathbf{s}) + 2N] \quad (4.12)$$

while the memory needed to perform arithmetic error correction, M_{Arith} , and for the proposed optimized method, M_{Opti} , can be expressed as:

$$\begin{aligned} M_{\text{Arith}} &= 2(m + \text{length}(\mathbf{s})) + 2(N - 1) \\ M_{\text{Opti}} &= 2(\text{length}(\mathbf{s})) + 2(N - 1) \end{aligned} \quad (4.13)$$

In (4.12) and (4.13), we consider that the position of each error is stored as a 2-byte integer, justifying the appearance of $2N$ in the equation, which actually represents the N -error pattern associated with the syndrome stored for the table approach. Table approaches need to store every error pattern syndrome of length n bits in the table along with the error pattern consisting of N integers. The arithmetic approach needs to store two vectors of size $(m + n)$ bits, corresponding to the two versions of the error vector used to process the algorithm. Such method also stores $(N - 1)$ integers as the forced bit positions. The optimized method also stores such forced positions but only needs to store the two versions of the error of a size of n bits (\mathbf{e}' and \mathbf{e}). For a search of double error pattern in a packet of length 1500 bytes, the storage needed for a lookup table approach is 432 MB, where only 6 bytes are needed for the proposed optimized approach. Moreover, when increasing the number of errors, the memory storage needed for the

table approach rapidly becomes intractable, with table sizes in Petabytes to handle 4 errors with CRC-16. The proposed approach also needs even less memory than state-of-the-art arithmetic method, which would require 3 kB to perform error correction. The memory required by the proposed approach is fixed for a given polynomial and does not depend on the packet length.

Algorithm 4.1 SingleErrorCorrection($\mathbf{s}, \mathbf{g}, n, m$)

Inputs:

\mathbf{s} : the syndrome vector
 \mathbf{g} : the vector associated with the generator polynomial
 used to compute the CRC
 n : the length of the syndrome vector
 m : the length of the payload vector

Output:

E_1 the list of valid error patterns for a single-bit error

```

1:  $E_1 \leftarrow \{\}$ 
2: Let  $\mathbf{e}$  be a vector of length  $n$ 
3:  $\mathbf{e} \leftarrow \mathbf{s}$ 
4: if  $\text{sum}(\mathbf{e}) = 1$  then
5:   Add  $\text{NZ}(\mathbf{e})$  to  $E_1$ 
6: end if
7: for  $j = 0$  to  $m - 1$  do
8:   if  $e_0 = 1$  then
9:      $\mathbf{e} \leftarrow (\mathbf{e} \oplus \mathbf{g}) \gg 1$ 
10:    if  $\text{sum}(\mathbf{e}) = 1$  then
11:      Add  $\text{NZ}(\mathbf{e}) + j$  to  $E_1$ 
12:    end if
13:  else
14:     $\mathbf{e} \leftarrow \mathbf{e} \gg 1$ 
15:  end if
16: end for
17: Return  $E_1$ 

```

4.4.3 Proposed approach implementation

To illustrate the implementation differences implied by the proposed approach, we describe, in this section, the updated algorithms to conduct multiple error correction. We encourage the

Algorithm 4.2 N -ErrorPatternGeneration($\mathbf{s}, \mathbf{g}, n, m, N, p_R$)**Inputs:**

\mathbf{s}, \mathbf{g} : the syndrome and generator polynomial binary vectors, respectively
 n, m : lengths of the syndrome and payload vectors
 N : the maximum number of bit errors considered
 p_R : the received corrupted packet

Output:

E_N the list of valid error patterns up to N bit errors

```

1: Let  $\mathbf{e}$  be a vector of length  $n$ 
2:  $E_N \leftarrow \{\}$  and  $\mathbf{e} \leftarrow \mathbf{s}$  and  $k \leftarrow N$ 
3: if  $\text{sum}(\mathbf{e}) \leq N$  then
4:   Add  $\text{NZ}(\mathbf{e})$  to  $E_N$ 
5: end if
6: while  $k \geq 1$  do
7:   if  $\text{mod}(\mathbf{g}, 2) = 1 \parallel (\text{mod}(\mathbf{s}, 2) = \text{mod}(k, 2))$  then
8:     if  $k = 1$  then
9:       Add  $\text{SingleErrorCorrection}(\mathbf{s}, \mathbf{g}, n, m)$  to  $E_N$ 
10:    else
11:      Let  $\mathcal{F} \leftarrow (0, \dots, k-2)$ 
12:      while  $\mathcal{F} \neq (m-(k-1), \dots, m-1)$  do
13:         $\text{start} \leftarrow \max(F_1-1, 0)$  and  $\text{nbForced} \leftarrow 0$ 
14:        for  $j = \text{start}$  to  $m-1$  do
15:          if  $j \in \mathcal{F}$  then
16:             $\text{nbForced} = \text{nbForced} + 1$ 
17:          end if
18:          if  $(e_0 = 0 \& j \in \mathcal{F}) \parallel (e_0 = 1 \& j \notin \mathcal{F})$  then
19:             $\mathbf{e} \leftarrow (\mathbf{e} \oplus \mathbf{g}) \gg 1$ 
20:            if  $\text{sum}(\mathbf{e}) + \text{nbForced} \leq N$  then
21:              Add  $(\text{NZ}(\mathbf{e}) + j)$  and  $F_i$  to  $E_N$ 
22:            end if
23:          else
24:             $\mathbf{e} \leftarrow \mathbf{e} \gg 1$ 
25:          end if
26:          if  $j = F_1$  then
27:             $\mathbf{e}' \leftarrow \mathbf{e}$ 
28:          end if
29:        end for
30:         $\mathcal{F} \leftarrow \text{UpdateForcedPositions}(\mathcal{F}, m)$  and  $\mathbf{e} \leftarrow \mathbf{e}'$ 
31:      end while
32:    end if
33:     $\mathbf{e} \leftarrow \mathbf{s}$  and  $k \leftarrow k-1$ 
34:  end if
35: end while
36: for  $i = 1$  to  $\text{size}(E_N)$  do
37:    $p_C \leftarrow p_T$  with flipped bit values at positions in  $E_N(i)$ 
38:   if  $\text{checksum}(p_C) \neq \text{OK} \parallel \text{decode}(p_C) \neq \text{OK}$  then
39:     Remove candidate  $E_N(i)$  from List
40:   end if
41: end for
42: Return  $E_N$ 

```

Algorithm 4.3 UpdateForcedPositions(\mathcal{F}, m)

Inputs:

\mathcal{F} : sorted list (F_1, \dots, F_{k-1}) of $(k - 1)$ bit positions
 forced to 1, such that $F_i < F_{i+1}, \forall i$
 m : the length of the payload vector

Note that $k = \text{len}(\mathcal{F}) + 1$, with $\text{len}(\mathcal{F})$ being the number of elements in the list \mathcal{F}

Output:

\mathcal{F}' : the updated sorted list of forced positions

```

1: if  $F_{k-1} < (m - 1)$  then
2:    $F_{k-1} \leftarrow F_{k-1} + 1$ 
3:   Return  $\mathcal{F}' \leftarrow (F_1, \dots, F_{k-1})$ 
4: else
5:   for  $i = k - 2$  to 1 do
6:     if  $F_i < F_{i+1} - 1$  then
7:        $F_i \leftarrow F_i + 1$ 
8:        $j \leftarrow i$ 
9:       while  $j < k - 1$  do
10:         $F_{j+1} \leftarrow F_j + 1$ 
11:         $j \leftarrow j + 1$ 
12:      end while
13:      Return  $\mathcal{F}' \leftarrow (F_1, \dots, F_{k-1})$ 
14:   end if
15: end for
16: end if

```

reader to compare them with the original algorithms presented in (Boussard *et al.*, 2020a). The main modifications are written as blue font in the proposed algorithms. Algorithm 4.1 illustrates the single error handling process, which takes as input the syndrome vector \mathbf{s} , the generator polynomial \mathbf{g} and the lengths expressed in bits of both the syndrome n and the payload m . In this updated algorithm, the main changes come from the memory optimization:

- **Step 2:** the error vector is of length n bits, the length of the syndrome, and is now independent of the payload length.
- **Step 3:** the error vector is initialized as the syndrome.

- **Step 4:** $\text{sum}(\mathbf{e})$ provides the number of bits set to 1 in vector \mathbf{e} . This sum is 1 when there is a single error.
- **Step 5:** instead of adding the whole error vector, we now only add the non-zero positions of \mathbf{e} to the list of single error candidates E_1 . The function NZ simply returns the degrees of the non-zero values in the input binary vector.
- **Step 8:** at each stage of the loop, we now only check the value of e_0 , i.e. the LSB value of the error vector \mathbf{e} . If such value is 1, the LSB must be canceled.
- **Step 9:** the cancellation process is performed through an XOR of the generator polynomial \mathbf{g} with the error vector \mathbf{e} . The result is then right shifted by one position to remain at n bits of length.
- **Step 11:** if there is only one non-null value in the error vector, its position is added to the candidates' list E_1 . As the position returned by $\text{NZ}(\mathbf{e})$ is relative, the number of shifted positions since the beginning of the process (i.e. j) must be added to that value.
- **Step 14:** when the LSB value of the error vector is 0, no cancellation is needed and the error vector is updated through a right shift by one position.

Algorithm 4.2 represents the multiple error handling process for a given syndrome \mathbf{s} , generator polynomial \mathbf{g} and number of errors considered N , including the parity check, memory optimization and CV. The modifications are represented in blue font in the algorithm. Note that in algorithms we denote the logical operations AND and OR as $\&$ and $||$, respectively. The inputs slightly differs from the original algorithm in (Boussard *et al.*, 2020a) as it includes the received corrupted packet p_R , which we will use in the CV step. The main changes are the following:

- **Steps 1 and 2:** the binary error vector \mathbf{e} is of length n and is initialized to the value of the computed syndrome.
- **Step 3:** we first check if the syndrome itself is a solution. If the number of non-null bits in the error vector is less or equal to the maximum number of errors considered, we add the corresponding error positions to the candidates' list E_N . The function $\text{NZ}(\mathbf{e})$ returns the degrees of the non-null positions in \mathbf{e} .

- **Step 7:** in order to avoid unnecessary computation, we check at the beginning of the main loop if the current case can lead to new candidates. We demonstrated that the search is unnecessary if the parity of the generator polynomial \mathbf{g} is even and if both the syndrome \mathbf{s} and the current numbers of error searched k have different parities.
- **Step 13:** we introduce a new variable, nbForced, which corresponds to the number of forced positions already set at the current time.
- **Step 16:** the variable nbForced is increased by one each time the current position j corresponds to a position to be forced (i.e. is included in the set \mathcal{F}).
- **Step 18:** to determine if the LSB value of the error vector e_0 has to be canceled or not, we must consider the value of the current position j . If j corresponds to a position to force and $e_0 = 0$ or if $e_0 = 1$ and the position must not be forced, we perform a XOR of the generator polynomial with the error vector, and then right shift the result by one position to keep a constant length of \mathbf{e} . If those conditions are not met, the error vector is simply right shifted by one position, as illustrated in step 24.
- **Step 21:** to consider a candidate as valid, the sum of the non-zero values in the error vector $\text{NZ}(\mathbf{e})$ added to the number of already forced bits must be less or equal to the number of considered errors N . If so, the candidate added to the list E_N comprises the non-zero values of the error vector added by the current position j , and the values of the currently forced positions: $F_i, \forall i \leq \text{nbForced}$. For example, if \mathcal{F} comprises a total of 3 forced positions and only 2 positions have been already forced at this stage of the process, the forced positions to be added to the error pattern are only positions F_1 and F_2 .
- **Step 33:** At each main loop iteration, the error vector is reinitialized to its original value \mathbf{s} .
- **Step 36:** While the state-of-the-art method stops the process at the end the main loop, we propose to add a validation step to reduce the size of the candidates' list. We test every candidate from the list.
- **Step 37:** for each candidate from the list we reconstruct a candidate packet, denoted p_C , which corresponds to the received packet p_R where the bits corresponding to the positions of the error pattern have been flipped.

- **Step 38:** We test the checksum value and the decodability of the candidate packet p_C . If one of those tests fails, the current candidate is removed from the candidates list at step 39. At the end of the process, the resulting list is returned.

4.5 Simulation and results

To illustrate the gains of the proposed method over the state-of-the-art ones in terms of error correction capability, we simulate transmission of compressed video over unreliable channels. Two compression standard profiles have been selected to encode the video sequences in our simulations: AVC Baseline 4.0 and HEVC Main. For both, we used an *IntraPeriod* of 30 frames. Two scenarios are considered: the first one addresses video communications in a vehicular environment, while the second is dedicated to IoT applications. For both scenarios, we conduct simulations to compare the reconstructed video quality from the following methods for AVC video content:

- JM-FC: standard AVC JM-FC reconstruction (Suhring, 2015),
- STBMA: spatio-temporal boundary matching algorithm error concealment method (Chen *et al.*, 2008) similar to the approaches used as comparison in a recent study (Kazemi *et al.*, 2020),
- CRC-ECA1: CRC-based single-bit error correction without any candidate validation (Bous-sard *et al.*, 2020b), and
- CRC-ECCV: the proposed CRC-based N -error correction with checksum and video validation to ensure the compliance of the reconstructed frame, with $N = 3$. When the proposed method is not able to perform error correction, we conceal the missing area using STBMA. As we discussed in Section 4.4.1, the CRC-based error correction for $N = 3$ without checksum validation would systematically produce several candidates in the output list. As no correction can be performed in such case, the performance would be the same as STBMA error concealment. Thus, we do not consider it in our forthcoming comparisons.

For HEVC, we compare CRC-ECA1, CRC-ECCV and the following method:

- Deblock+MVS: deblocking filter and iterative motion vector search error concealment techniques, available in the FFmpeg decoder (FFmpeg, 2019).

However, in CRC-ECCV for HEVC, we use Deblock+MVS when we cannot correct the packet. The *Intact* sequences correspond to error-free decoded sequences, using JM and HM software for AVC and HEVC encoded video sequences, respectively.

The video sequences used to conduct the simulations are *Ice*, *Crew* and *Mobcal* sequences. We chose such sequences as they show different visual characteristics:

- *Ice* sequence (4CIF 704×576) represents ice skaters moving laterally over a fixed ice rink background.
- *Crew* sequence (4CIF 704×576) represents crew members walking towards the camera, with no camera movement.
- *Mobcal* sequence (HD 1280×720) represents a toy train moving from the right to the left side of the frame, along with a camera down tilt movement.

As we will show, it is crucial to take into account the characteristics of the transmission channel when deciding to apply CRC-based error correction. Moreover, the proposed method performs logically better on channels for which the number of errors is relatively small (packets are mildly corrupted). It does not perform well on channels where the packets are either well received or severely corrupted. Note that the performance of the CRC-based multiple error correction method on realistic network scenarios was never demonstrated before.

4.5.1 Wi-Fi 802.11p

The Wi-Fi 802.11p standard (Association, 2016) was designed to operate and transmit data over vehicular networks, for vehicle to vehicle (V2V), vehicle to infrastructure (V2I) and vehicle to anything (V2X) scenarios. In (Rameau *et al.*, 2016), the authors show that video transmission from a vehicle to another can be useful in overtaking situations. They propose to transmit the video from the windshield camera from a vehicle to the vehicle right behind. By doing so, the second car can have access to the point of view of the car it wants to overtake, and can thus verify

that the overtaking can be done in a secure way. Video transmission can also ensure transport safety, such as in-vehicle surveillance in public transport that can share their video content live to authorities if a threat of vandalism is detected. V2I video transmission can also be used to monitor traffic conditions, by sending to a roadside unit (RSU) the video from external cameras of equipped vehicles in the area, to get feedback on the current traffic conditions.

V2V channels have been implemented in Matlab from real-world data collected by (Alexander *et al.*, 2011) for different scenarios, such as urban and rural scenario, considering that the vehicles are either on LOS or Non-Line-of-Sight (NLOS). The 802.11p transmission and vehicular channel models used in our simulations are available in the Matlab WLAN toolbox (MathWorks, 2018). We tested our method over several channel conditions described in each example.

Fig. 4.6 presents the comparison, in terms of reconstructed video quality, at the receiver between state-of-the-art methods and the proposed approach, for different channel SNRs, in a rural LOS scenario. We selected such channel SNR values as they range from near-optimal conditions to severely degraded reconstructions. The PSNR of the error-free (intact) reconstructed sequence is equal to 38.60 dB. The PSNR of the reconstructed sequence using 1) JM-FC, 2) STBMA, 3) CRC-ECA1, and finally 4) the proposed CRC-ECCV, are also given. We verify that the video quality is an increasing function of the channel SNR. The gains are expressed with the sequence PSNR, as recent works in (Kazemi *et al.*, 2020) conclude that to achieve better quality assessment, the whole video sequence should be considered to due to propagation of errors, and

Table 4.5 Average number of errors per corrupted packet for different channel SNR values in an 802.11p environment.

	SNR value				
	36 dB	32 dB	28 dB	24 dB	20 dB
1 error	1.3 %	1.9 %	2.5 %	1.4 %	0.9 %
2 errors	3.5 %	5.1 %	1.8 %	1.9 %	1.4 %
3 errors	16.9 %	11.1 %	7.8 %	5.5 %	2.8 %
> 3 errors	78.3 %	81.9 %	87.9 %	91.2 %	94.9 %

also demonstrate that in such case the sequence PSNR offers better results than more recent metrics. By performing a CRC error correction on 3-error patterns with a CV, the proposed approach is able to correct an average of 10% of the corrupted packets. This leads to PSNR gains from +0.2 dB at high channel SNRs (38 dB) to +0.6 dB for lower channel SNRs (24 dB) compared to STBMA error concealment method. The simulations conducted also show that increasing the number of errors to consider would not be worth the increase of computational complexity, as most of the packets contain a huge number of errors. As an example, in the simulation run for high channel SNR of 34 dB, we observed that almost 70% of the corrupted packets contain more than 10 erroneous bits.

Table 4.5 shows the experimental error distribution we obtain over 802.11p channel, for different channel qualities. We can observe that the average percentage of packets containing more than 3 errors tends to decrease as the channel quality increases, but still remains at less than 20% of the corrupted packets. When the channel conditions are degraded, the ratio of corrupted packets that the method can handle tend to decrease. We observe a maximum of 18.1% of corrupted packets containing less than 3 errors at a channel SNR of 32 dB to 5.1% at a channel SNR of 20 dB.

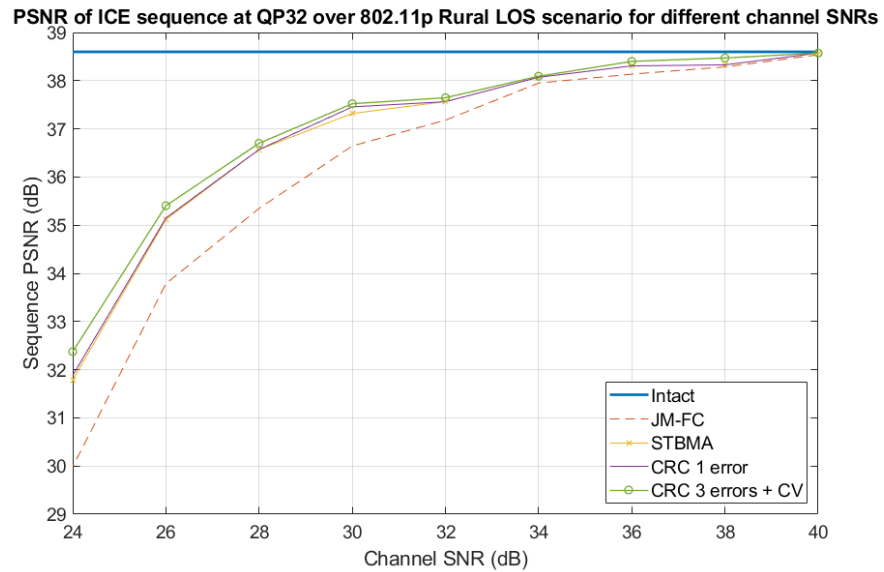


Figure 4.6 Comparison of sequence PSNRs for AVC encoded *Ice* sequence (4CIF 704x576) at QP 32 and different channel SNRs (vehicular Rural LOS scenario).

Correcting 10 to 20% of the corrupted packets can help to increase the PSNR up to 0.6 dB in this test case, but the visual impact is not that significant when many errors occurred in the packet. Of course, the average number of errors in the packet decreases when the channel quality increases, but still remains very high for most parts of the error cases. The proposed method is still able to correct most of the error patterns up to 3 errors but as the majority of the corrupted packets are highly corrupted, the gains are greatly affected by the high ratio of erroneous packets associated with the LOS rural scenario. Consequently, applying the proposed method to correct up to 3 errors in the 802.11p environment does not provide significant improvements. Indeed, it is clear that the CRC-based error correction is mainly efficient in communication environments where the corrupted packets are generally mildly corrupted. In the next section, we present another application which is more suitable for the proposed method, hence enabling better results and higher quality of the reconstructed video sequence.

Since the application of the proposed method on 802.11p does not provide spectacular improvements, we do not provide further results for this case.

4.5.2 Bluetooth Low Energy

The BLE (SIG, 2013) standard is widely used in IoT (Al-Fuqaha *et al.*, 2015) applications, to transmit sensors data in an energy-efficient way. In particular, BLE is used in Video Surveillance over Internet of Things (VS-IoT) devices to monitor security cameras when a movement is

Table 4.6 Average number of error per corrupted packet for different channel conditions.

	Eb/No value			
	10 dB	9 dB	8 dB	7 dB
1 error	76.5 %	53.3 %	31.3 %	17.3 %
2 errors	13.5 %	27.4 %	35.9 %	27.5 %
3 errors	4.8 %	13.0 %	20.4 %	20.9 %
> 3 errors	5.2 %	6.3 %	12.4 %	34.3 %

detected (Jyothi & Vardhan, 2016).

To conduct transmission simulations over BLE channels, we use the Bluetooth Low Energy simulation with radio frequency impairments proposed in the Matlab Communication toolbox (MathWorks, 2019). We simulate different channel conditions by varying the E_b/N_0 parameter. The E_b/N_0 corresponds to the SNR per bit of the communication. In the further examples, the E_b/N_0 typically ranges from 7 dB to 10.5 dB, corresponding to Bit Error Rates (BER) ranging from 10^{-4} to 10^{-7} . The BLE standard imposes a maximum packet length of 250 bytes. In what follows, videos were encoded with such packet size limit, to ensure that transmission is energy efficient and maintain a constant and low header overhead. Such encoding management will have an impact on error distribution within the video sequence, as an intra-coded frame requires much more packets to be delivered than an inter-predicted frame when the payload of each packet is constant, since it is less efficiently compressed.

Unlike Wi-Fi 802.11p, the error distribution over BLE channels is favorable to the CRC-based

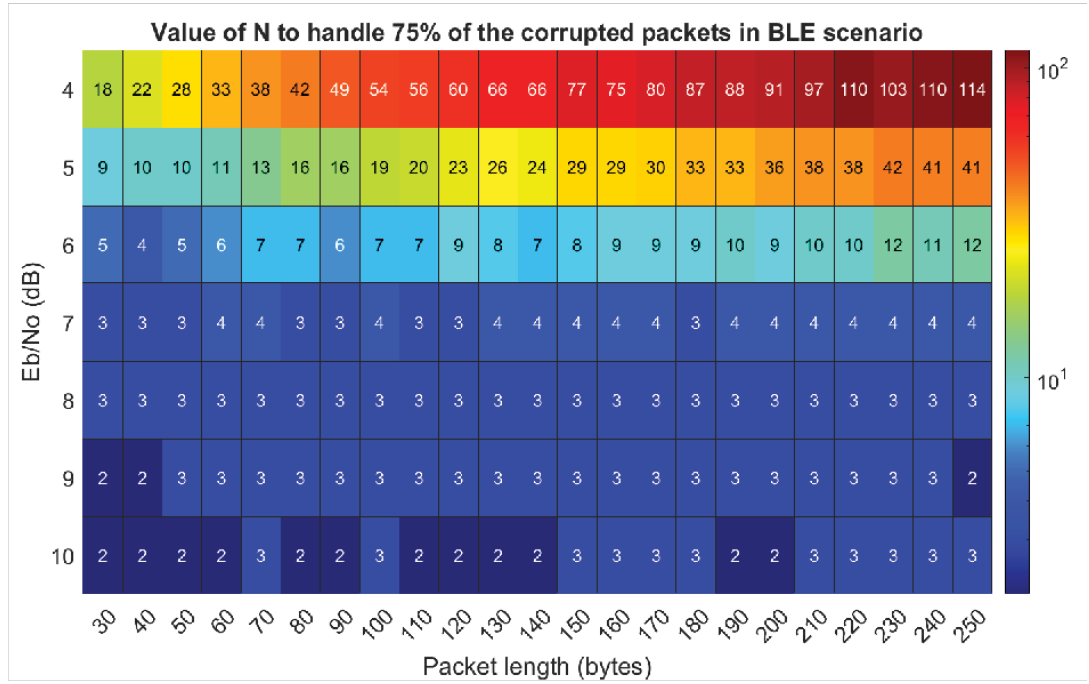


Figure 4.7 Heatmap representing the value of the number of errors to consider, N , to handle 75% of the error cases for a given channel quality and packet length over a BLE channel. If N is set to the value in the corresponding box for such parameters, then it can successfully correct 75% of the corrupted packets.

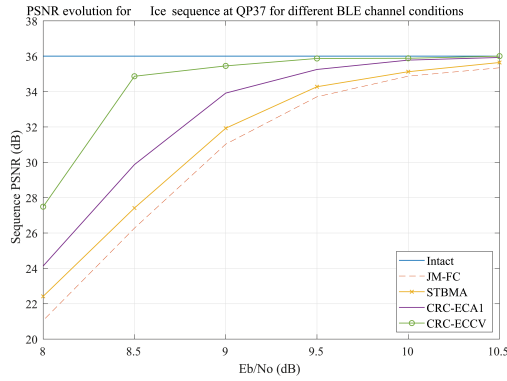
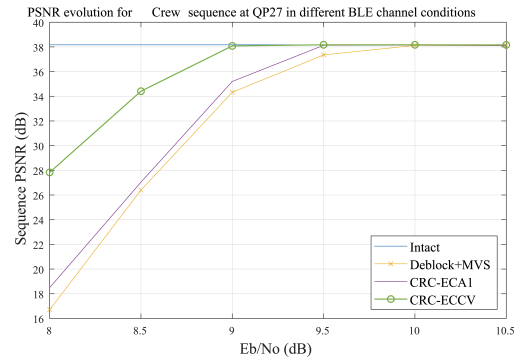
a) AVC *Ice* Sequence at QP 37b) HEVC *Crew* sequence at QP 27

Figure 4.8 Comparison of sequence PSNRs for AVC encoded *Ice* and HEVC encoded *Crew* sequences (4CIF 704×576) at different channel conditions in a Bluetooth Low Energy environment.

error correction design, as illustrated in Fig. 4.7. The heatmap shows the required value of N to handle 75% of the error cases. The two axes of the map are the E_b/N_0 ratio and the packet length considered, in bytes. In other words, it means that 75% of the error cases for the given packet length and channel quality contains the number of errors in the block or less. For example,

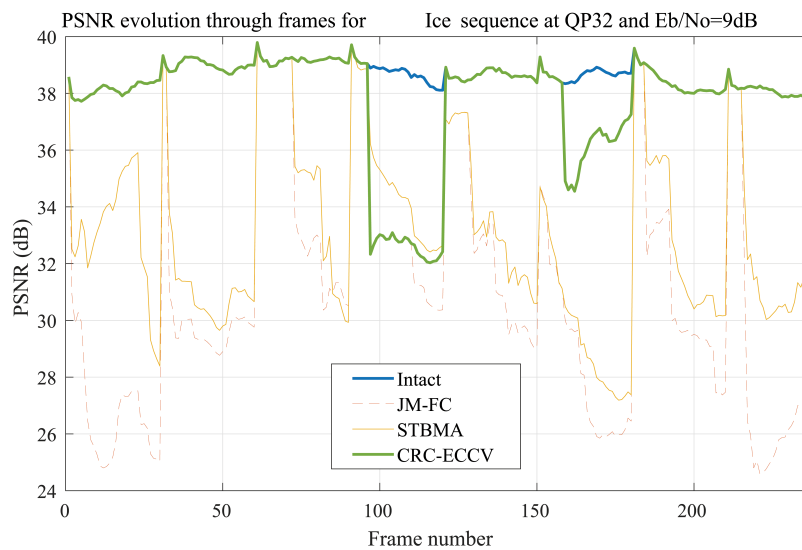


Figure 4.9 PSNR evolution through time for H.264 sequence *Ice* (4CIF) at QP32. The channel used has a E_b/N_0 of 9 dB.

for highest quality channels considered in this example, it means that 75% of the corrupted packets contains 3 errors or less. We can also observe here that the number of errors to consider increases rapidly when the channel quality decreases. As the proposed method yields better results in high quality channels, it is able to maintain a near-optimal video quality in high quality channels subject to mild corruption but does not offer as significant gains when the channel conditions are drastically decreasing. In Fig. 4.7, we can consider that the channel and packet length conditions where the method is able to operate efficiently correspond to values of N less or equal to 3, which are represented in dark blue in the heatmap and correspond to E_b/N_0 of 8 or higher up to the maximum packet length in the BLE standard (250 bytes).

Table 4.6 shows the average percentage of error cases encountered in corrupted packets for different channel conditions. We note that for high-quality channels, considering 3 errors instead of 1 can help to increase the correction rate from 76.5% to 94.8%. In this case, it is possible to reconstruct the video sequence with near-optimal visual quality. The gains for more severe channel conditions are even greater, as for an E_b/N_0 value of 8 dB, the correction rate jumps from 31.3% to 87.6%. However, such gains are less visible on the reconstructed video, as the reconstructed video content is basically of poor visual quality in such channel conditions. It is interesting to observe that for all channels considered, the percentage of packets hit by more than 3 errors increases as the channel conditions decrease. However, this percentage remains low. In all tested cases, more than 85% of the corrupted packets contained 3 errors or less. Our method is therefore well suited to operate on such transmissions and can achieve a significant correction rate compared to state-of-the-art methods.

In Fig. 4.8, we present the PSNR of two reconstructed video sequences after transmission using the BLE protocol as a function of the E_b/N_0 parameter value. In these simulations, we compare in Fig. 4.8a the JM-FC and STBMA error concealment methods to CRC-ECA1 and CRC-ECCV on the AVC encoded 4CIF *Ice* sequence, for a QP of 37. Fig. 4.8b compares the deblocking filter with motion vector search error correction to the same CRC-based error corrections, on the HEVC encoded 4CIF *Crew* sequence for a QP of 27. From these figures it is clear that the proposed approach offers significant gains over traditional methods when the channel conditions start to decrease. We can see that the proposed method maintains a high reconstruction quality

where other methods start to rapidly decrease. We can observe that once the channel conditions are too severe, the proposed approach's performance also decreases.

In order to illustrate the performance of the proposed method, we represent in Fig. 4.9 the evolution of the PSNR through time for a sequence transmitted through a Bluetooth Low Energy channel with $E_b/N_o=9$ dB. The tested sequence here is *Ice*, a 4CIF sequence of length



Figure 4.10 Visual comparison of the literature methods to the proposed approach on AVC encoded *Ice* sequence (4CIF) at QP32 for an E_b/N_o value of 8 dB.

240 frames, encoded at QP32. We compare the proposed CRC-ECCV approach to JM-FC, represented as red dashed line, and STBMA, represented in plain yellow. The proposed approach is the green curve and we represented the error-free version in blue. In this example, 47 packets have been corrupted throughout the transmission. We can observe that JM-FC and STBMA greatly suffer from these corrupted packets as their PSNR is significantly lower through the sequence. We can also see that each Intra frame, which comes every 30 frames, helps all methods to recover a high PSNR until a new packet is hit by one or several errors. The proposed method achieves a significantly better video quality during most of the video. However, we can see that even if the method is able to handle most parts of the corrupted packets, the PSNR greatly decreases at frames 97 and 159. It was verified that those packets were hit by more than 3 errors, thus the proposed approach was not able to perform error correction. Still, when considering the whole sequence, we can see that the global PSNR variability of the proposed approach is significantly lower than other approaches. Indeed, this is a crucial factor to consider for final users' perceived quality. It has been shown that human viewers are more affected by quality drops than average quality decrease (Yim & Bovik, 2011). We can see in Fig. 4.9 that the error concealment method yields great variability which would greatly affect the viewer's quality of experience.

A visual example of the gains obtained thanks to the proposed method over state-of-the-art methods is shown in Fig. 4.10 considering the worst BLE channel ($E_b/N_0=8$ dB). Each of the visual examples shows the reconstructed video for the different methods compared. We can observe that the video frame after error concealment still exhibits severe visual artifacts, while single error correction removed only a few of them. The proposed approach was able to correct most parts of the corrupted packets in such frame, resulting in a visually better video quality. However, we can also observe that not all the corrupted packets were corrected during the process. We can see that a packet containing more than 3 errors has been concealed near the center of the frame. As we encoded the video sequences under the packet length constraints of BLE, we can observe that the impact on visual quality of losing a packet is mild as it carries less visual information.

Table 4.7 Average PSNR comparison for different sequences and QPs over different BLE channel conditions for AVC encoded sequences. The tested decoding methods are the following: ①: intact sequence, ②: JM-FC concealed sequence, ③: STBMA concealed sequence, ④: CRC-ECA1 and ⑤: proposed CRC-ECCV.

Sequence	QP	Eb/No = 10 dB					Eb/No = 9 dB					Eb/No = 8 dB				
		①	②	③	④	⑤	①	②	③	④	⑤	①	②	③	④	⑤
Ice	22	43.53	35.14	38.05	42.84	43.50	43.53	27.72	31.65	35.37	41.95	43.53	18.37	20.84	22.70	31.80
	27	41.16	36.78	38.26	41.06	41.12	41.16	29.21	32.36	36.59	39.73	41.16	18.77	20.91	23.29	31.60
	32	38.60	35.68	36.53	37.88	38.15	38.60	31.66	33.33	35.94	37.55	38.60	21.52	23.37	24.99	29.83
	37	36.01	34.87	35.12	35.78	35.88	36.01	31.03	31.93	33.92	35.45	36.01	21.05	22.42	24.14	27.49
Crew	22	41.87	39.61	39.95	40.68	41.86	41.87	37.17	37.97	40.23	41.66	41.87	29.88	30.99	32.76	38.57
	27	38.80	38.03	38.20	38.78	38.79	38.80	34.45	34.99	36.37	38.03	38.80	27.09	28.05	29.65	36.18
	32	36.25	35.70	35.76	35.92	36.23	36.25	33.80	34.09	35.08	35.76	36.25	26.67	27.12	28.07	33.19
	37	33.74	33.33	33.36	33.49	33.70	33.74	31.53	31.57	32.37	33.67	33.74	26.53	26.71	28.42	31.92
Mobcal	32	34.77	33.72	34.27	34.56	34.70	34.77	27.98	29.71	30.92	34.07	34.77	20.14	22.09	23.72	29.11
	37	31.81	30.55	30.93	31.18	31.79	31.81	27.54	28.51	30.05	31.75	31.81	19.44	20.71	22.34	27.49
Δ_{PSNR}	22	-	0	1.63	4.38	5.30	-	0	2.36	5.35	9.36	-	0	1.79	3.60	11.06
	27	-	0	0.83	2.51	2.55	-	0	1.85	4.65	7.05	-	0	1.95	3.94	11.36
	32	-	0	0.50	1.09	1.33	-	0	1.23	2.83	4.64	-	0	1.42	2.82	7.93
	37	-	0	0.22	0.57	0.87	-	0	0.64	2.08	3.59	-	0	0.71	2.39	6.39

In Table 4.7, we compare the reconstructed sequences' PSNRs of several error concealment and error correction methods in a BLE environment, applied to several H264 encoded sequences. The results show that as the channel quality decreases, the gain of the proposed approach, compared to the literature, increases. For example, for a high E_b/N_0 of 10 dB, the proposed approach increases the sequence PSNR of 0.8 dB compared to STBMA and 0.2 dB compared to CRC single error correction. We can also observe that the gains greatly increase as the E_b/N_0 ratio decreases and at 8 dB, the average gains of the proposed method become 5.7 dB over STBMA and 4.2 dB over CRC single error correction. As these gains are better in less favorable channel conditions, we must observe the PSNR loss compared to the intact sequence. Even if we can gain up to 5 dB compared to other methods at E_b/N_0 of 8 dB, we also lose 5.5 dB on average compared to the error-free video sequence, where smaller gains for good channel conditions help to reconstruct a near-intact sequence, with an average PSNR difference of 0.15 dB for $E_b/N_0=10$ dB. As the channel conditions decrease, more packets containing a high number of errors are received. However, the method still can correct up to 80% of the corrupted packets. The proposed approach is thus designed to maintain a near-optimal visual quality in slightly disturbed channels where literature method would suffer from visual artifacts, and is still able to correct most of the packets as the channel quality decreases, but cannot ensure significant visual quality improvement in severe channel conditions.

4.6 Conclusion

In this paper, we proposed several improvements to the CRC-based error correction method to increase its error correction capability, by handling the list of candidates, and reducing its computational complexity.

This method achieves better results than the state-of-the-art CRC error correction method as it can handle most of the double and triple error cases that the original method would not have been able to. Validation of the reconstructed bit sequence is an essential additional step to ensure the validity of the corrected sequence and reducing the risk of wrong correction.

Simulations over different wireless environments including Wi-Fi 802.11p and BLE were

conducted. We demonstrated that the gains achieved by the proposed method strongly depend on the targeted application. The proposed method offers an average correction rate of 10 to 20% of the corrupted packet in a Wi-Fi 802.11p environment for channel SNRs from 24 to 36 dB in Rural LOS scenario. The gains are significantly better for Bluetooth Low Energy where the proposed method is able to correct 70% of the corrupted packets for severe channel conditions, at E_b/N_0 of 8 dB. The proposed approach offers PSNR gains from 0.8 dB to 5.7 dB over state-of-the-art error concealment methods, at E_b/N_0 of 10 dB and 8 dB, respectively.

Further works include conceiving a highly reliable method to select the best candidate when several candidates pass both the CRC and CV process and are decodable. Although exploiting fixed and predictable field values in protocols is expected to bring further gains, we also want to investigate pixel-domain approaches, using deep learning, to identify the most probable intact video among the reconstructed remaining candidates. The application of the proposed method can also be extended to the most recent video codecs such as VVC along with codec-specific validation steps based on syntax.

CHAPTER 5

CRC-BASED CORRECTION OF MULTIPLE ERRORS USING AN OPTIMIZED LOOKUP TABLE

Vivien Boussard^{1,2}, Stéphane Coulombe¹, François-Xavier Coudoux², Patrick Corlay²

¹ Département de génie logiciel et des technologies de l'information,
École de technologie supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3

² Univ. Polytechnique Hauts-de-France, CNRS, Univ. Lille, ISEN, Centrale Lille, UMR 8520 -
IEMN - Institut d'Électronique de Microélectronique et de Nanotechnologie, DOAE
Département d'Opto-Acousto-Électronique, F-59313 Valenciennes, France

Article submitted for publication in January 2021

5.1 Abstract

In this paper, we propose a new approach to performing multiple error correction in wireless communications over error-prone networks. It is based on the CRC syndrome, using an optimized lookup table that avoids performing arithmetic operations. This method is able to achieve the same correction performance as the state-of-the-art approaches while significantly reducing the computational complexity. The table is designed to allow multiple bit error correction simply by navigating within it. Its size is constant when considering more than two errors, which represents a tremendous advantage over earlier lookup table-based approaches. Simulation results of a C implementation performed on a Raspberry Pi 4 show that the proposed method is able to process single and double error corrections of large payloads in 100 ns and 642 μ s, respectively, while it would take 300 μ s and 1.5 s, respectively, to handle the same packet with the state-of-the-art CRC multiple error correction technique. This represents a speedup of nearly 3000 \times for single error and 2300 \times for double error correction, respectively. Compared to table-based approaches, the proposed method offers a speedup of nearly 1200 \times for single error and 2300 \times for double error correction under the same conditions. We also show that when multiple candidate error patterns occur are present, we can correct numerous errors by adding a checksum cross-validation step.

5.2 Introduction

In wireless communications, cyclic redundancy checks (CRCs) (Sobolewski, 2003) are widely adopted in order to enhance the communication reliability between a transmitter and a receiver. In fact, CRCs are widely used at different layers of the protocol stack of a data transmission as they detect transmission errors at the receiver. For instance, CRCs are present at the physical layer of widely deployed wireless protocols such as 802.11 (Association, 2016) to protect the header of the packet, and at the Medium Access Control (MAC) layer, to protect the entire packet. An example of the latter is the 802.3 Ethernet protocol (Association, 2018). In general, CRCs are only used to detect whether a transmitted packet has bit errors, in which case the erroneous packet is discarded.

CRCs are computed from two main components: the protected bitstream, i.e., the message or data to transmit, which we will call the payload, denoted $d_T(x)$ for transmitted data, and a generator polynomial, denoted $g(x)$. Both $d_T(x)$ and $g(x)$ are binary polynomials. The generator polynomial is a constant binary polynomial, which is defined according to the transmission standard used. There are various generator polynomials in use, differentiated by their length and their number of non-null coefficients. Larger generator polynomials will lead to stronger error detection capabilities, but at the cost of higher packet overheads.

The typical transmission and reception of a CRC-protected data packet follow three main steps (Sobolewski, 2003). Firstly, at the transmitter, the payload $d_T(x)$ is left-shifted by n positions, where n is the degree of the generator polynomial, to produce the transmitted packet $p_T(x) = d_T(x) \cdot x^n$. This result is then divided by the generator polynomial $g(x)$ to obtain the remainder of this division, denoted $r_T(x)$. Secondly, $r_T(x)$ is appended to the payload to be sent, and occupies the n rightmost positions of the transmitted message. The transmitted message is thus $d_T(x) \cdot x^n + r_T(x)$, where $+$ is the addition which corresponds to an *exclusive or* (XOR) between two binary polynomials¹, and has a total length of $m + n$ bits, where m is the payload length of the message. Thirdly, at the receiver, the received packet, denoted p_R , is divided by the generator polynomial $g(x)$ in order to check if an error occurred during the transmission. If

¹ Binary packets of length m belong to the Galois Field $\text{GF}(2^m)$, where the addition is performed as the bitwise XOR (Luo *et al.*, 2012).

the message is intact, the remainder of the division by $g(x)$, called the computed CRC syndrome $s(x)$, will produce a result equal to zero since $r_T(x)$ has been added to generate an entire multiple of $g(x)$. On the other hand, a non-zero CRC syndrome indicates an error in the transmitted message.

In currently deployed systems and methods, conventional CRC error management involves discarding a packet when an error is detected at the receiver, that is, when the remainder is not equal to zero at the receiver. These approaches do not allow for packets to be corrected. In such systems, a received corrupted packet is processed as a lost packet. For example, a packet having a single bit in error, and which would otherwise be a good packet, will be detected as an erroneous packet and will be entirely discarded. Discarding entire packets having only one or a few errors leads to a significant loss of valuable information. The literature contains methods used to complete the information missing from the packet, namely, error concealment and inpainting. These methods can perform spatial error concealment (Sun & Kwok, 1995; Koloda *et al.*, 2013; Liu *et al.*, 2015; Akbari *et al.*, 2017), in which the spatial correlation within a scene is exploited to generate the missing video part, or temporal error concealment (Wu *et al.*, 2008; Lam *et al.*, 1993; Chen *et al.*, 2003; Chung & Yim, 2020), which is based on the correlation between temporally adjacent frames, to conceal the lost video regions. Some methods also use both correlations to perform error concealment (Atzori *et al.*, 2001; Chen *et al.*, 2008). Video inpainting approaches aim to fill the missing video part by producing a natural-looking region. Early video inpainting methods proposed patch-based approaches, while today, video completion is performed with deep learning approaches (Sankisa *et al.*, 2018; Kim *et al.*, 2019; Wang *et al.*, 2019). Error concealment and video inpainting methods can achieve significant quality improvements, but their performance is a function of the naturalness and action of the scene. Furthermore, typically, corrupted packets are handled in such methods simply by discarding.

Extracting information from a lightly corrupted packet to enhance the quality of the received stream is definitely beneficial. A single corrupted packet can lead to severe visual degradation if it is not concealed perfectly (Kazemi *et al.*, 2020), as the slightest error will propagate through time due to temporal redundancy used in video compression standards. Reconstructing the

original packet when lightly corrupted should thus result in visual quality increases and prevent error propagation, as discussed in (Superiori *et al.*, 2007; Trudeau *et al.*, 2011).

Several methods have been designed to recover slightly corrupted packets, using an estimation of the most likely received bitstream based on Maximum A Posteriori (MAP) methods (Moonseo Park & Miller, 2000; Caron & Coulombe, 2013), or using error detection codes to correct errors (Golaghazadeh *et al.*, 2017a). Various existing methods, described in detail in section 5.3, already exploit the CRC to perform error correction in a packet (Shukla & Bergmann, 2004; Babaie *et al.*, 2009; Aiswarya & George, 2017; Liu *et al.*, 2018; Niu & Chen, 2012). One such approach in particular consists in storing the different syndromes produced by single error patterns (Aiswarya & George, 2017) in a lookup table. In this paper, we propose an alternative method for CRC-based multiple error correction using an optimized lookup table. In our method, most of the required operations are done offline and then store in a table. The table is designed to allow the results for single and double error patterns to be accessed by only navigating through the table, which results in significant processing time reductions over table-free methods. The proposed approach can also be applied to identify any number of errors, with significant speedups. The following are the benefits and contributions of the proposed method:

- **Significant speed gains:** The proposed approach is designed to allow most to all the arithmetic operations required in the state-of-the-art table-free error correction approach to be performed offline and stored in a table. This design contributes to greatly reduce the processing time of the proposed method.
- **Fixed-length tables:** State-of-the-art table-based approaches define distinct tables for each number of errors and maximum packet size considered. Furthermore, these tables grow in size exponentially with the number of errors considered, making them impractical when considering 3 or more errors. In contrast, the tables derived and used for the correction of multiple bits in the proposed approach are fixed for a given generator polynomial, and therefore possess a fixed length regardless of the number of errors or packet size under consideration. We also propose another version of the table for the special case of a single error correction, which requires less memory storage.

- **Analysis of the syndromes and of the table structure:** We provide equations to identify, for any generator polynomial, syndromes with special properties (e.g., syndromes for which there is no valid error pattern comprising a single erroneous bit). We also highlight the cyclic properties of the table-based process which produces syndrome elements looping on themselves.
- **Applicability to multiple error correction:** Because of its large speed gains and reasonable table sizes, the proposed method is well-positioned to correct multiple errors. However, in such situations, the candidate error patterns leading to the computed syndrome are numerous, making the identification of the true error pattern challenging. We show that we can significantly reduce the list of candidates, even to the point of having a single one and being able to correct the packet, by adding a validation step in the form of testing of the candidates with the checksum found in UDP and TCP protocols.

The rest of the paper is organized as follows. In section 5.3, we introduce related works on CRC error correction covering lookup table methods and state-of-the-art CRC multiple error correction. In section 5.4, we present the concepts and implementation of the proposed optimized table, as well as its use in multiple error correction systems. We also analyze the structure of the table and syndromes with special properties. In section 5.5, we evaluate the performance of the proposed approach in terms of processing speed and memory usage, as compared to existing methods. In section 5.6, we conclude and give an overview of future research works².

5.3 Related works

Several works have explored the error correction possibilities of error detection codes, such as CRC or checksums (Golaghazadeh *et al.*, 2017a). The first works conducted on table-based CRC error correction, and described in (Shukla & Bergmann, 2004; Babaie *et al.*, 2009), compute a lookup table (LUT) prior to the communication. In this scheme, each LUT entry contains the pre-computed syndrome corresponding to a specific single error position in the received packet.

² We assume that the reader is familiar with the notations and concepts described in our previous works (Boussard *et al.*, 2020a), especially those related to the Galois Field $GF(2)$ (Hachenberger & Jungnickel, 2020a) and its generalization to $GF(2^m)$.

An example of such a table for CRC-8-CCITT is given in Table 5.1. A search for the computed syndrome is performed in the table. If a match is found, the bit at the corresponding position is flipped. The number of entries in the lookup table is based on the size of the expected payload, and is constant. It has been applied to other generator polynomials by (Aiswarya & George, 2017). This method is fast when conducted on small packets, but suffers from some serious disadvantages. Firstly, the approach assumes that a single error occurred in the packet, yielding a miscorrection probability in severe channel conditions, since a highly corrupted packet can produce the same syndrome as a single error. Furthermore, to support the correction of multiple errors, methods based on this approach must store the syndromes of all combinations of error positions. Consequently, memory requirements grow exponentially with the number of errors considered, limiting their usefulness in practice.

Other methods exploiting the CRC syndrome do not rely on lookup tables to perform error correction. In (Boussard *et al.*, 2020a), we introduced a table-free multiple error correction. This approach generates an exhaustive list of error patterns corresponding to the computed syndrome up to a predetermined number of errors. The data packet includes a payload $d_T(x)$ and cyclic

Table 5.1 Example of lookup table for single error correction as proposed in literature (Shukla & Bergmann, 2004), applied to CRC8-CCITT of generator polynomial $g(x) = x^8 + x^2 + x + 1$ considering a 10-bit payload.

Error position	Associated syndrome	Error position	Associated syndrome
0	0000 0001	9	0011 1010
1	0000 0010	10	0111 0100
2	0000 0100	11	1110 1000
3	0000 1000	12	1100 1101
4	0001 0000	13	1000 0111
5	0010 0000	14	0001 0011
6	0100 0000	15	0010 0110
7	1000 000	16	0100 1100
8	0001 1101	17	1001 1000

redundancy check (CRC) information. The latter is calculated using a generator function or polynomial $g(x)$. The method generates an exhaustive list of valid error patterns containing N errors or less, by exploiting the definition of CRC computation. We can express the syndrome computed at the receiver as:

$$s(x) = (d_T(x) \cdot x^n + r_T(x) + e(x)) \bmod g(x) \quad (5.1)$$

where $e(x)$ represents the potential error pattern that hits the packet during the transmission, and where erroneous positions in $e(x)$ are identified by values of 1. From this definition, it is clear that the result of $(d_T(x) \cdot x^n + r_T(x)) \bmod g(x)$ is zero, as $r_T(x)$ is the remainder of the division of $d_T(x)$ by $g(x)$, and:

$$s(x) = e(x) \bmod g(x) \quad (5.2)$$

If we isolate the error vector, $e(x)$, we obtain:

$$e(x) = s(x) + q(x) \cdot g(x) \quad (5.3)$$

where $q(x)$ can be any binary polynomial of the highest degree $(m - 1)$, with m being the payload length. As the total number of possible values of $q(x)$ is too high (there are 2^m such polynomials), the method proposes focusing on lightly corrupted packets and then building $q(x)$ term by term. This is achieved by canceling the Least Significant Bit (LSB) term of the current $e(x)$, at each step of the process, through the addition of properly left shifted $g(x)$ (i.e., so that its LSB term is aligned with the term of $e(x)$ to cancel³). Adding $g(x)$ aligned at any position maintains the class equivalence of (5.3) and produces error pattern candidates having the same computed syndrome at each step (Boussard *et al.*, 2020a). The number of non-null coefficients in the polynomial $e(x)$ corresponds to the number of errors in the current candidate. The method thus only keeps candidates when the number of non-null coefficients in $e(x)$ is equal to or less than a predefined number N . To handle the correction of multiple bit errors, the method forces term values in $e(x)$ throughout the process (i.e., it sets such terms to 1 or turns them into a 1 by

³ In this paper, we assume that $g_n = g_0 = 1$ as observed in practice.

adding $g(x)$ aligned with their positions). Once $(N - 1)$ positions have been forced, single error management is performed on the remaining length of the packet to locate the last error. If this last error does not exist, it means that the forced term values did not correspond to a valid error pattern for the given syndrome (i.e., an error pattern leading to the computed syndrome).

Although this method requires very little memory space, one drawback with it, however, is that it is very complex when considering several errors in a packet. This complexity, measured in the number of additions involving $g(x)$, is $O(m^{N-1})$, where m represents the payload length and N the number of errors considered. It thus grows exponentially with N .

5.4 Proposed method

While the state-of-the-art method discussed in the last section aims at generating the list of valid error patterns with arithmetic operations, the proposed method does the same by exploiting tables, and accordingly avoids most arithmetic operations. Unlike previous methods (Shukla & Bergmann, 2004), where tables are sparse and contain error positions only for CRC syndromes compatible with a certain packet size (e.g., that associated with a standard), the proposed tables provide error patterns for every possible syndrome value. In addition to being usable for any packet size, these tables can be indexed directly by syndrome value instead of being searched, as was done in the previous case. Indeed, the process of searching for a pattern in a table, as in the previous approaches, is computationally intensive. This method thus provides enhanced speed performance, but at the cost of higher memory requirements. However, the proposed tables are solely dependent on the generator polynomial of interest, and are fixed for any number of errors N , while the previous methods lead to tables whose sizes grow exponentially with N .

In what follows, we will use the notation \mathbf{a} to represent the binary vector associated with a binary polynomial $a(x)$. The i -th element (LSB-wise) of this vector will be denoted a_i . The addition of polynomials $a(x)$ and $g(x)$, denoted $a(x) + g(x)$, will become the XOR of vectors \mathbf{a} and \mathbf{g} ,

denoted $\mathbf{a} \oplus \mathbf{g}$. The left shift of $g(x)$ by n positions, denoted $g(x) \cdot x^n$ will be denoted $\mathbf{g} \ll n$. The addition of two vectors should be interpreted as an XOR operation.

5.4.1 Single error correction

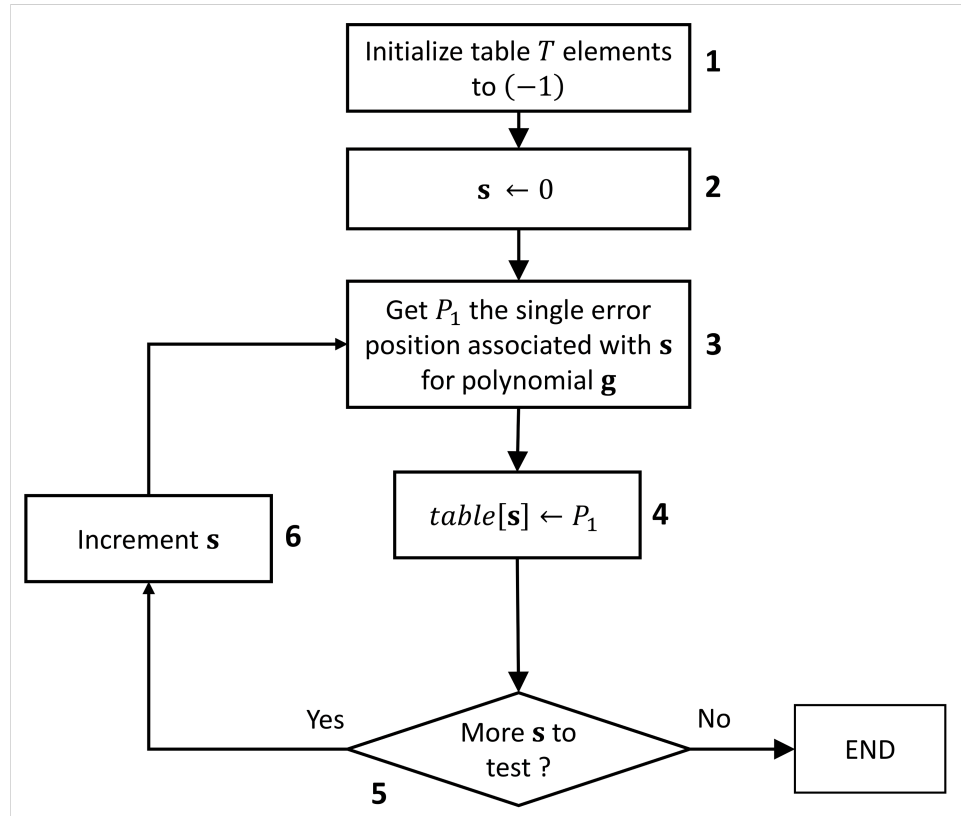


Figure 5.1 Flowchart representing the steps to build the table containing the whole list of syndromes along with their associated single error positions

The state-of-the-art method achieves CRC syndrome simplification through successive additions (XORs) of the current error pattern vector (initialized to the computed syndrome) with left-shifted versions of the generator polynomial, and an updating of the error vector \mathbf{e} with the result of the addition until a single bit error pattern appears in \mathbf{e} (i.e., a single-bit is set to 1). In the proposed method, we aim to avoid these computations by storing the location (relative distance) of the single error corresponding to each syndrome value in a table. Because the table is directly indexed by these syndrome values, the error location can be accessed directly by using the syndrome as an index in the table. The first step of the proposed method is thus to generate

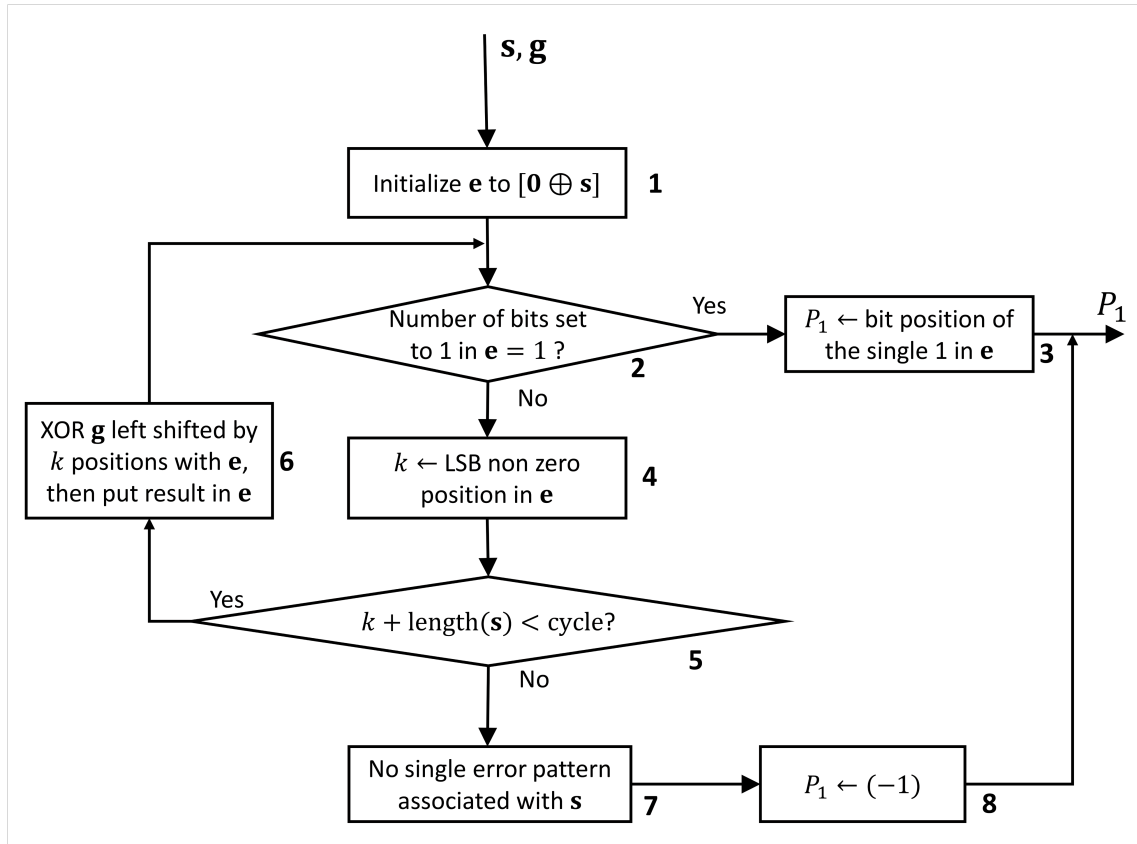


Figure 5.2 Flowchart representing the steps to obtain the single error position from a computed syndrome (step 3 in Fig 5.1). In step 1, $\mathbf{0}$ represents the null vector

the table for single error correction, following the steps illustrated in the flowchart given in Figure 5.1:

Step 1: The table is initialized to (-1) . We choose this initialization value as it cannot be a valid entry in the list, contrary to 0, which would correspond to a single error at position 0. The size of the table corresponds to the size of the possible set of syndromes, (i.e., $2^n - 1$, where n is the bit length of the syndrome s).

Step 2: The syndrome is then initialized to $\mathbf{0}$ although this value is not of interest as it would indicate that there is no error in the packet.

Step 3: Next, the single error position P_1 associated with the syndrome s for a given polynomial g is determined. The sub-steps to perform are illustrated in a separate flowchart in Figure 5.2, where it can be seen that the state-of-the-art single error algorithm is actually performed in steps

Table 5.2 Single error position table generated for a CRC-5 of generator polynomial
 $g(x) = x^5 + x^4 + x^2 + 1$

Index	P_1	Index	P_1
0	-1	16	4
1	0	17	-1
2	1	18	-1
3	-1	19	-1
4	2	20	-1
5	-1	21	5
6	-1	22	8
7	10	23	-1
8	3	24	-1
9	-1	25	9
10	-1	26	14
11	7	27	-1
12	-1	28	12
13	13	29	-1
14	11	30	-1
15	-1	31	6

1, 2, 4, 5 and 6. Figure 5.2 presents an error handling process similar to the CRC-based single error correction available in literature, where the packet length m is set to the largest packet size before single error periodicity (i.e., we denote this length as the cycle length of a generator polynomial). The first step is to initialize \mathbf{e} to m zeros and set its LSB value to \mathbf{s} . Then, for each syndrome, we thus count the number of non-null values in the error vector \mathbf{e} and check if this number equals 1. If that is the case, the corresponding error position is set in the corresponding entry of the table. If not, the LSB non-null value is canceled and the error vector is checked again, until we reach the cycle length. The cycle length for any generator polynomial is equal to or less than $2^n - 1$ and can be retrieved experimentally by determining the distance between two single error patterns having the same syndrome.

Step 4: The single error position provided by step 3 is stored in the table.

Step 5: If the whole set of possible syndromes has been tested (final value of \mathbf{s} reached), the

process stops. If not, the syndrome is updated by incrementing its value by 1.

At the end of the process, a table similar to Table 5.2 is obtained, and shows an example for the polynomial generator $g(x) = x^5 + x^4 + x^2 + 1$. The index column is actually implicit and do not need to be stored; in fact, it was added here simply for better readability. In this table, P_1 denotes the degree of the single error position (i.e., the single non-null position in \mathbf{e}). For instance, looking at the table, the error position is x^{10} when the computed syndrome is 7. It can be seen that half the indexes indicate a position P_1 of (-1) . This notation means that there is no single error position or solution associated with the corresponding syndrome value. Since the generator polynomial's parity is even, it mainly corresponds to syndromes with even parity. Of note, when the generator polynomial has even parity, syndrome values with even parity cannot lead to odd parity error patterns, and thus, they cannot lead to single error solutions. We will see later in the paper that there is also a specific syndrome for which no solution exists when the generator polynomial has even parity.

The steps for identifying candidate single error positions when receiving a corrupted packet are illustrated in Algorithm 5.1. The method proceeds by simply checking the value of P_1 in the table at the index corresponding to the computed syndrome s , as shown in step 2. If P_1 is different from (-1) and indicates a value less than the length of the packet, a candidate is appended to the list. A search for more valid candidates is conducted, considering the cycle length of the generator polynomial used. For example, in Table 5.2, the cycle length is $(2^4 - 1) = 15$. Thus, the position P_1 ranges from 0 to 14, producing syndromes 1 and 26 at these values (i.e., the corresponding Index value in the table). If an error occurs at position 15, as it corresponds to position $P_1 = (0 + cycle)$, it will produce a syndrome equal to 1. For a packet of length 50, when computing syndrome 1 (i.e., the same as position $P_1 = 0$), the error can thus be at positions 0, 15, 30 and 45. Since valid single error positions are cyclic, we test all possible values up to the length of the received packet. An error correction method may correct the packet if a single candidate is returned by Algorithm 5.1 or if one of them passes additional validations such as UDP or TCP checksum (Golaghazadeh *et al.*, 2018; Demirtas *et al.*, 2011).

Algorithm 5.1 SingleErrorCorrection($T[2^n], s, n, m, cycle$)

Inputs:

$T[2^n]$: indexed table containing P_1 for each syndrome
 s : the syndrome vector
 n : the length of the syndrome vector
 m : the length of the payload vector
 $cycle$: the cycle length of the generator polynomial

Output:

E_1 : the list of valid error positions for single-bit error

```

1:  $E_1 \leftarrow \{\}$ 
2:  $P_1 \leftarrow T[s]$ 
3: if  $P_1 \neq -1$  then
4:   while ( $P_1 < m + n$ ) do
5:     Add  $P_1$  to  $E_1$ 
6:      $P_1 \leftarrow P_1 + cycle$ 
7:   end while
8: end if
9: Return  $E_1$ 

```

The proposed single error correction approach differs from the state-of-the-art lookup table approaches as it considers the whole set of possible syndromes that can occur, regardless of the packet length. Moreover, current lookup table approaches are sorted in a bit error position-wise order, making it mandatory to scan the table to retrieve the error position. By sorting the table in a syndrome-wise order, we do not have to scan the table as it can be indexed with the computed syndrome value.

5.4.2 Double error correction

In order to handle double error correction, the strategy proposed in the state-of-the-art CRC-based multiple error correction method is to force a first error and then search for the second using the single error method. As first error is forced at a specific position F_1 by setting all lower positions to 0 and this position to 1 through successive conditional additions of \mathbf{g} at required positions, and the resulting syndrome is expected to correspond to a single error syndrome. Checking the

position of this error by applying the single error method will indicate whether the remaining error at position P_1 (as well as the other single error positions at positions separated by the cycle length for sufficiently large packets) occurs within the packet, which would lead to a new candidate error pattern with errors at positions $(P_1 + 1 + F_1)$ and F_1 . Thus, to consider the entire set of possibilities, the single error management must be called for each possible location of the first error. At the end of each verification, the LSB error is moved by one bit toward the most significant bit (MSB), from the LSB position where $F_1 = 0$ to the end of the payload at position $F_1 = m - 1$. This repositioning is conducted in two steps:

- The previous bit position tested (former F_1 , forced to 1) is set to 0 by XORing g at this position.
- The following bit toward the MSB is considered as the new first error (i.e., $F_1 \leftarrow F_1 + 1$) and is either kept at 1 if it was 1 or forced to 1 by XORing g at this position if it was 0.

The process is repeated until the forced bit position reaches the position m , corresponding to the length of the protected data (position $m - 1$ is the last position processed).

It can be seen that based on these steps, the succession of syndromes is always the same for

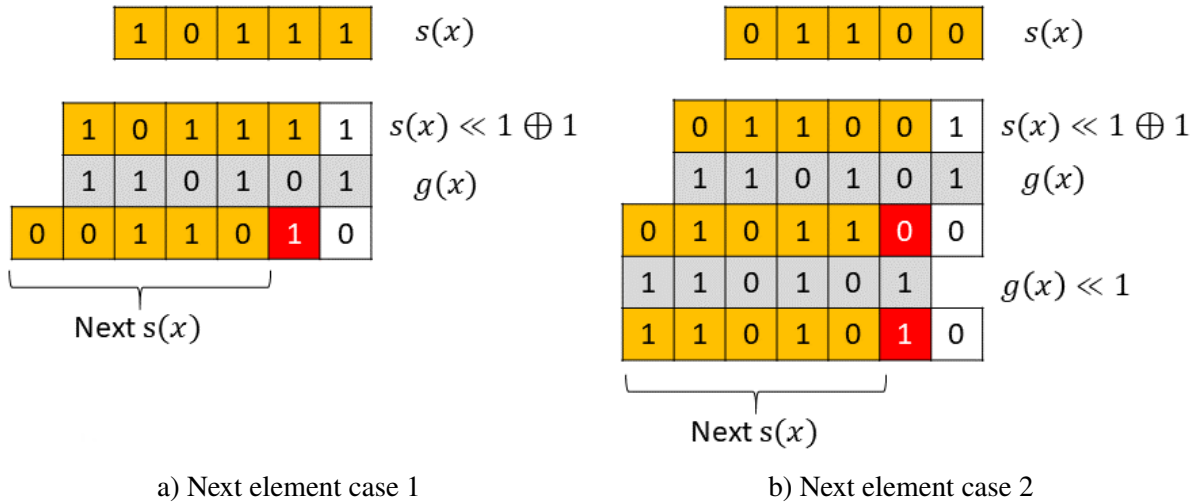


Figure 5.3 Illustration of the next elements generation for the two possible cases. In case 1, the newly forced value is already set to 1 after the first XOR of $g(x)$. In case 2, the newly forced position is 0 after the first XOR, which requires a second XOR of $g(x)$ to force this position to 1. The red boxes represent the new position to force.

a given generator polynomial and therefore we propose, in this method, to store the value of the next syndrome based on the current one. More specifically, the *next* element \mathbf{s}' of a given syndrome \mathbf{s} is:

$$next = \begin{cases} (\mathbf{s}' \oplus \mathbf{g}) \gg 1 & \text{if } s_0 = 0 \\ \mathbf{s}' \gg 1 & \text{if } s_0 = 1 \end{cases} \quad (5.4)$$

where \mathbf{s}' is defined as:

$$\mathbf{s}' = [((\mathbf{s} \ll 1) \oplus 1) \oplus \mathbf{g}] \gg 1 \quad (5.5)$$

The form of (5.5) actually corresponds to an updating of the forced error position by canceling the previously forced one and setting the new forced position to 1. These steps are illustrated in Figure 5.3, where both cases discussed are presented. In Figure 5.3a, the syndrome is first left shifted by one position and 1 is appended as the LSB, which represents the formerly forced position. We cancel this position by XORing the generator polynomial. As the new forced position (represented by a red box) is already set to 1, no further step is required and we simply consider the next syndrome. On the other hand, in Figure 5.3b, the new forced position is not set to 1 after the first XOR. In this case, therefore, another XOR with the generator polynomial is needed to produce the next syndrome. In the example of Figure 5.3a, it is clear that starting from the syndrome $\mathbf{s} = [10111] = 23$, we cancel the previously forced value by XORing the generator polynomial and leave the new position to force as 1, identified as a red box. We thus obtain the next element for the syndrome equal to 26, which is $\mathbf{s} = [00110] = 6$.

Linking a syndrome to the next will significantly reduce the complexity of the approach since the arithmetic operations will be pre-computed and stored in the reference table.

In (Boussard *et al.*, 2020a), it was shown that throughout the process of identifying error patterns with N or fewer errors, which operates from the LSB to the MSB, only a range of n bits can contain non-zero values since the lower positions are already eliminated and the higher positions, initialized to 0, have not yet been altered. The proposed method exploits this property by considering only values within this range (sliding window) and processing them as the syndromes of interest. In the case of a double error, the forced error position will alter the syndrome value on which our subsequent single error position determination is based. In

essence, we consider the effect of forcing the error position on the syndrome used for identifying the remaining error position. This latter position becomes relative to the forced position. It should be noted that the forced error position, represented by a red box in Figure 5.3a, is not part of the syndrome considered for single-bit error determination, but is implicitly present throughout the process described in (5.5).

The following are the steps for generating the table containing both the single error position and the next element, as illustrated in Figure 5.4:

Step 1: The whole table is initialized to (-1). The number of entries in the table is the same as the single error table previously described, with the difference lying in the number of columns. For each row, another column containing the next element of the current syndrome is appended.

Step 2: We initialize **a**, a variable representing the first syndrome of the loop, to 1.

Step 3: We now begin to complete the table. In order to avoid unnecessary computations, we first check that the current table position has yet not been processed and that the current values of **a** and **g** are able to produce a candidate.

Step 4: If that is the case, we initialize the next element to 0 and set a local syndrome **b** to the value of **a**.

Step 5: We first fill the single error position through to the steps described in the previous section (see Figure 5.2).

Step 6: The next element is identified for the current syndrome **b** and the generator polynomial **g**. The steps to determine the next element are described in a separate flowchart in Figure 5.5. To generate the next element, we cancel the previously forced bit position by XORing **g** and set the new forced position through a new XOR if necessary).

Step 7: We store the single error position and the next element in the table.

Step 8: The local syndrome **b** is then updated to the next position computed in step 6.

Step 9: If the next element has not yet been processed, we keep on computing its associated single error position and next element. If the next element has already been processed, the current cycle of *next* has looped, which means that we have to increment **a** and start a new loop.

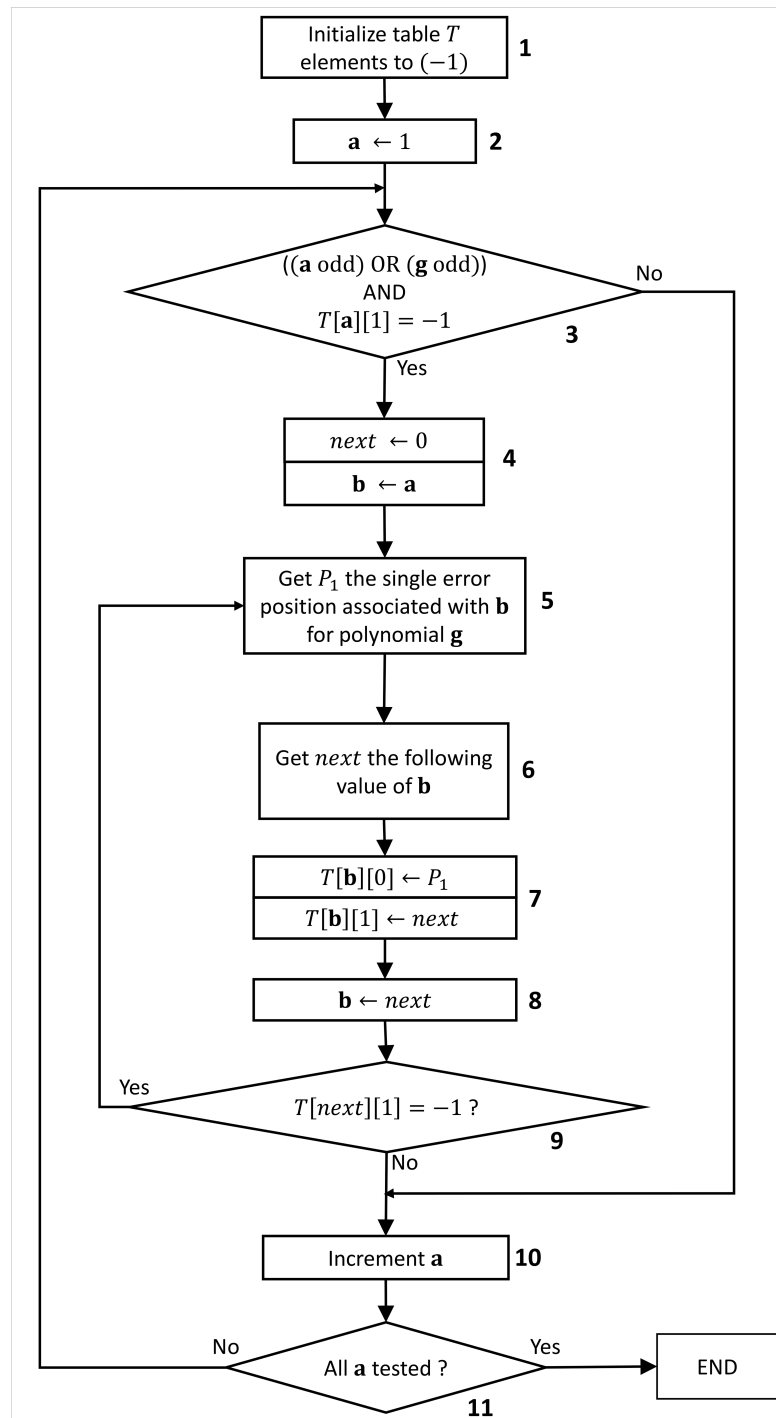


Figure 5.4 Flowchart representing the steps to generate the table T containing single error position and next element to handle double error correction

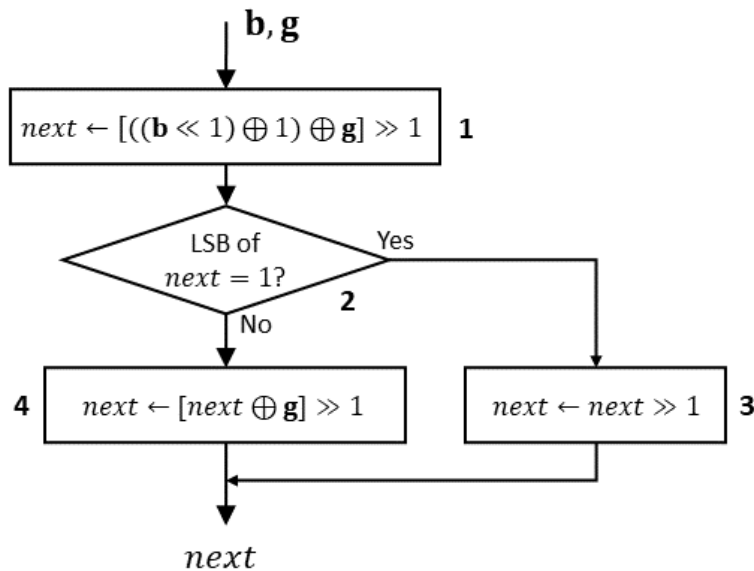


Figure 5.5 Flowchart representing the steps to follow to generate the next element from the computed syndrome and the generator polynomial (step 6 of Fig. 5.4, illustrated in Fig. 5.3)

Step 10: Once every value of \mathbf{a} has been tested, the process is ended as the table is complete.

Table 5.3 shows the complete table at the end of the process for the generator polynomial $g(x) = x^5 + x^4 + x^2 + 1$. The P_1 column corresponds to the relative distance of the remaining single error, and the $next$ column stores the next syndrome to be tested (i.e., the syndrome after the forced position is updated).

Algorithm 5.2 illustrates the different steps for performing the correction of double-bit errors when the table is generated:

Step 1: The candidate list containing the candidates with 2 errors is initialized as an empty set.

Step 2: A vector \mathbf{s}' of length n bits is created, and it will correspond to the updated version of the original syndrome \mathbf{s} .

Step 3-7: As each $next$ element from the table has been generated given that an error had already been forced, the first error must be added prior to navigation within the table. Thus, if the LSB of the syndrome \mathbf{s} is 0, an XOR with \mathbf{g} must be performed to force the first position. Since this forced bit is implicit and not part of the syndrome, a right shift is performed in both cases.

Algorithm 5.2 TwoErrorCorrection($T[2^n][2], s, n, m, cycle$)

Inputs:

$T[2^n][2]$: indexed table containing P_1 and *next* elements for each syndrome
 s : the syndrome vector
 n : the length of the syndrome vector
 m : the length of the payload vector
 $cycle$: the cycle length of the generator polynomial

Output:

E_2 : the list of valid error positions for double-bit error

```

1:  $E_2 \leftarrow \{\}$ 
2: Let  $s'$  be a vector of length  $n$ 
3: if  $s \wedge 1 = 0$  then
4:    $s' \leftarrow [s \oplus g] \gg 1$  // Can be stored in a table
5: else
6:    $s' \leftarrow s \gg 1$  // Can be stored in a table
7: end if
8: for  $F_1 = 0$  to  $m + n - 1$  do
9:    $P_1 \leftarrow T[s'][0]$ 
10:  if  $P_1 \neq -1$  then
11:    while ( $P_1 < m + n - F_1$ ) do
12:      Add ( $P_1 + 1 + F_1, F_1$ ) to  $E_2$ 
13:       $P_1 \leftarrow P_1 + cycle$ 
14:    end while
15:  end if
16:   $s' \leftarrow T[s'][1]$ 
17: end for
18: Return  $E_2$ 

```

Note that in practice, such operations can also be stored in the table. Adding two columns to store these results would double the memory storage needed but avoid any arithmetic operation.

Step 8: At this point, we consider the first error at position 0, and perform the loop over all possible forced positions, from 0 to $m + n - 1$.

Step 9-10: The relative single error position is accessed from table T . Two conditions must be met to consider the candidate as valid. First, the position P_1 must indicate a relative distance (i.e., the single error position for a syndrome value equal to the index), as tested in step 10, and

then the position of the error indicated by P_1 must be in the range of the packet.

Step 11: As the distance P_1 is relative to the current forced position F_1 , $P_1 < m + n$ is not enough to guarantee that the error pattern is possible. Thus, considering a relative packet size of $m + n - F_1$, we ensure that we identify only valid error patterns.

Step 12: When both conditions described in Step 10 and Step 11 are met, a candidate comprising the forced position F_1 and the remaining error position $P_1 + 1 + F_1$ is appended to the list.

Step 13: As the single error position is cyclic, we should test every possible error pattern. At each loop, we add the *cycle* length to the position P_1 and check if the resulting value is within the range of the packet. If not, the next forced position must be tested.

Step 16: Once a forced position has been processed, the updated syndrome, corresponding to the syndrome resulting from the next forced position is accessed from the table. It corresponds to the second column of the current syndrome index.

5.4.3 N-error correction

For N errors, the generalization of the strategy used for double error correction is performed, as illustrated in Algorithm 5.3. The concept is as follows. $(N - 2)$ bits must be forced to 1 at each step (initialized to the $(N - 2)$ LSBs of the syndrome). Then, the double-error method is performed on the remaining bits of the packet. The number of errors, N , is then decreased and when it reaches 1, the single error correction is performed on the computed syndrome.

For each forced bit error position (in the double-error approach after forcing the $(N - 2)$ bits), the *next* element (forced the next bit toward the MSB) can be accessed from the reference table to reduce computational complexity and avoid arithmetic operations. Therefore, we look at every possible combination of forcing $(N - 2)$ bits to 1 within the $m - 1$ first bits (LSB) of the packet. Let the forced bits be at positions F_1, F_2, \dots, F_{N-2} with $F_1 < F_2 < \dots < F_{N-2}$ (sorted by increasing the bit position). Forcing some bits to 1 means that the bits at positions F_1, F_2, \dots, F_{N-2} are set to 1 and the other bit positions below position F_{N-2} are set to 0. With this definition, it can be noted that all the bits with a position below F_{N-2} are actually forced (to

Table 5.3 Table generated for double-bit error correction for a CRC-5 of generator polynomial $g(x) = x^5 + x^4 + x^2 + 1$

Index	P_1	$next$	Index	P_1	$next$
0	-1	23	16	4	31
1	0	13	17	-1	5
2	1	22	18	-1	30
3	-1	12	19	-1	4
4	2	21	20	-1	29
5	-1	15	21	5	7
6	-1	20	22	8	28
7	10	14	23	-1	6
8	3	19	24	-1	27
9	-1	9	25	9	1
10	-1	18	26	14	26
11	7	8	27	-1	0
12	-1	17	28	12	25
13	13	11	29	-1	3
14	11	16	30	-1	24
15	-1	10	31	6	2

the value 0 or 1). Steps 4 to 12 of Algorithm 5.3 illustrate this forcing process.

Steps 1-2: First, we initialize both the candidate list to an empty set, and k , the local number of errors to consider, to its maximum value N .

Step 4-5: At step 4, the set of forced bits \mathcal{F} is initialized to the $(k - 1)$ LSB positions. Step 5 shows that the process continues until \mathcal{F} is set to the $(m + n)$ MSB positions. In step 8, $\&$ and $||$ represent the *logical and* and *logical or* operations, respectively.

In this approach, by “forced bits”, we mean the positions forced to 1, representing the $(N - 2)$ errors placed. However, it should be understood that when a bit within the range of forced bit positions is not forced to 1, then it is forced to 0. More generally, we focus on the positions forced to 1 because the algorithm strives to set the other positions to 0 during its elimination process.

Step 6: We start by initializing the binary vector \mathbf{s}' , representing the updated syndrome, to \mathbf{s} .

Algorithm 5.3 NErrorCorrection($T[2^n][2], s, n, m, cycle, N$)

Inputs:

$T[2^n][2]$: indexed table containing P_1 and *next* elements for each syndrome
 s : the syndrome vector
 n : the length of the syndrome vector
 m : the length of the payload vector
 $cycle$: the cycle length of the generator polynomial
 N : the maximum number of errors to consider

Output:

E_N : the list of valid error patterns up to N errors

```

1:  $E_N \leftarrow \{\}$ 
2:  $k \leftarrow N$ 
3: while  $k > 2$  do
4:    $\mathcal{F} \leftarrow (0, \dots, k-2)$ 
5:   while  $\mathcal{F} \neq (m+n-k+1, \dots, m+n-1)$  do
6:      $s' \leftarrow s$ 
7:     for  $i = 0$  to  $F_{k-2}$  do
8:       if  $(s'_1 = 0 \ \& \ i \in \mathcal{F}) \ || \ (s'_1 = 1 \ \& \ i \notin \mathcal{F})$  then
9:          $s' \leftarrow (s' \oplus g) \gg 1$  // Can be stored in a table
10:      else
11:         $s' \leftarrow (s' \gg 1)$  // Can be stored in a table
12:      end if
13:    end for
14:     $m' \leftarrow m - (F_{k-2} + 1)$ 
15:    Add TwoErrorCorrection( $T, s', n, m', cycle$ ),  $\mathcal{F}$  to  $E_N$ 
16:     $\mathcal{F} \leftarrow \text{UpdateForcedPosition}(\mathcal{F}, m)$ 
17:  end while
18:   $k \leftarrow k - 1$ 
19: end while
20: Add TwoErrorCorrection( $T, s, n, m, cycle$ ),  $\mathcal{F}$  to  $E_N$ 
21: Add SingleErrorCorrection( $T[][0], s, n, m, cycle$ ) to  $E_N$ 
22: Return  $E_N$ 

```

Steps 7-11: Forcing bit positions must be achieved through the successive addition of the generator polynomial vector and an accumulation in the updated syndrome s' in order to obtain the desired result.

The syndrome cannot be simply ignored and the bit values change at desired positions. Rather,

Algorithm 5.4 UpdateForcedPositions(\mathcal{F}, m)

Inputs:

\mathcal{F} : sorted list (F_1, \dots, F_{k-1}) of $(k - 1)$ bit positions
 forced to 1, such that $F_i < F_{i+1}, \forall i$
 m : the length of the payload vector

Note that $k = \text{len}(\mathcal{F}) + 1$, with $\text{len}(\mathcal{F})$ being the number
 of elements in the list \mathcal{F}

Output:

\mathcal{F}' : the updated sorted list of forced positions

```

1: if  $F_{k-1} < (m - 1)$  then
2:    $F_{k-1} \leftarrow F_{k-1} + 1$ 
3:   Return  $\mathcal{F}' \leftarrow (F_1, \dots, F_{k-1})$ 
4: else
5:   for  $i = k - 2$  to 1 do
6:     if  $F_i < F_{i+1} - 1$  then
7:        $F_i \leftarrow F_i + 1$ 
8:        $j \leftarrow i$ 
9:       while  $j < k - 1$  do
10:         $F_{j+1} \leftarrow F_j + 1$ 
11:         $j \leftarrow j + 1$ 
12:      end while
13:      Return  $\mathcal{F}' \leftarrow (F_1, \dots, F_{k-1})$ 
14:    end if
15:  end for
16: end if

```

the equivalence relationship with the original syndrome must be maintained, i.e., only shifted versions of \mathbf{g} may be added to it. This is done by starting at bit 0 of the syndrome and if its value is the desired value then nothing is done for that position. Otherwise, \mathbf{g} must be added at that position (i.e., XOR perform), which contains 1 at its LSB, to modify it. The decision as to whether or not to perform the XOR is illustrated in step 8 of Algorithm 5.3.

As the method successively processes the next positions from LSB (bit 0) to MSB (bit F_{N-2}) in a similar fashion, it is important that \mathbf{g} be added at each step at the current position (i.e., \mathbf{g} should be XORed at step 9 when processing position i) and the syndrome value when the conditions of

step 8 are met until position F_{N-2} is processed. Otherwise, the syndrome is right-shifted by one position at each step, as shown in step 11.

In practice, these operations can also be stored in a table when searching for N -error patterns. Thus, two additional columns would be added, doubling the required storage, but every computation would be stored in the table. Only the forced bit positions set \mathcal{F} and the current position i are needed to perform the full candidate list generation.

From there, since all the forced bit positions have been properly set, the process described for double-error handling is performed on the remaining length of the packet. The following are the remaining steps:

Step 14: We introduce m' , which corresponds to the remaining length of the packet. It is crucial to take into account the fact that the double-bit error correction must be performed with the current position taken into account to ensure the whole set of possibilities are considered.

Step 15: We perform a double-bit error correction on the updated syndrome s' , comprising the $(N - 2)$ LSB forced positions, for a packet length of m' . The candidate error patterns comprise the 2 positions returned by Algorithm 5.2 and the forced positions contained in \mathcal{F} .

Step 16: Once the double error correction has been performed, we must update the forced error position, as illustrated in Algorithm 5.4 to test all possible $(N - 2)$ forced positions.

Step 18: The main loop is performed for every number of errors k from N to 3 (i.e., when the error must be forced). When k reaches the value of two, the last steps to perform are a double-bit error correction, followed by a single-bit error correction on the original syndrome s and payload length m , as shown in steps 20 and 21. We return the completed list of candidates E_N at step 22.

5.4.4 Cycles of *next* elements

In this subsection, we study some properties of the generated lookup table. Although this knowledge does not impact the functionality of the proposed algorithms, it provides valuable insights into the nature of the solutions to expect, and that could be further exploited. Although more complex, the approach has similarities with periodic sequence and shift registers (linear

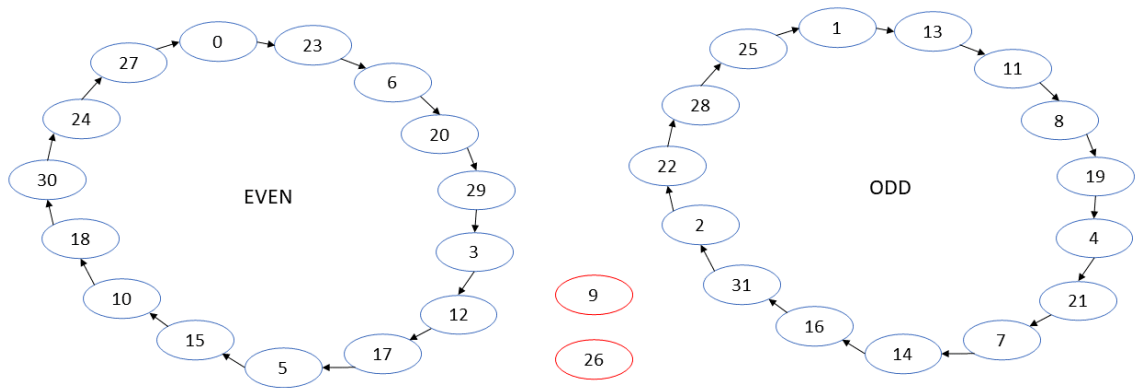


Figure 5.6 Example of cycles and exceptions when the generator polynomial is $g(x) = x^5 + x^4 + x^2 + 1$

and non-linear feedback shift registers) over Galois Fields (Hachenberger & Jungnickel, 2020b), which have been studied in the literature (Li *et al.*, 2020; Tripathi *et al.*, 2020; Jetzek, 2018). We can see in the flowchart of Figure 5.4 that there are two distinct variables for determining the syndromes, namely, **a** and **b**. In the previous subsection, **a** was described as the main loop syndrome and **b** as the local loop syndrome. While **a** was incremented by 1 at each main loop, the syndrome value of **b** was successively updated to its *next* element until the *next* element had been processed (i.e., all the syndromes in the current cycle had been added to the table).

Such cycles are observed for every generator polynomial. Figure 5.6 presents an example of this cycle. In the figure, we present the *next* element cycles for a generator polynomial $g(x) = x^5 + x^4 + x^2 + 1$. It can be seen that since the generator polynomial parity is even, the sets of syndromes resulting from an odd or an even number of errors are disjoint. The value of each syndrome is expressed in decimal value in Figure 5.6, where it can be seen that most parts of the table will be completed once these two cycles (i.e., two local loops on **b**) are performed. A straightforward design to generate the table would consider the local odd and even loops only. However, we observed two exceptions to these cycles. In fact, two syndrome values are out of the cycle loops and are represented by red circles in Figure 5.6. For the considered generator polynomial, these two syndrome values correspond to $s = 9$ and $s = 26$, respectively corresponding to $s = [01001]$ and $s = [11010]$ in binary vector representation, from MSB to LSB. We present the computation of the *next* element for both cases in Figure 5.7, where they

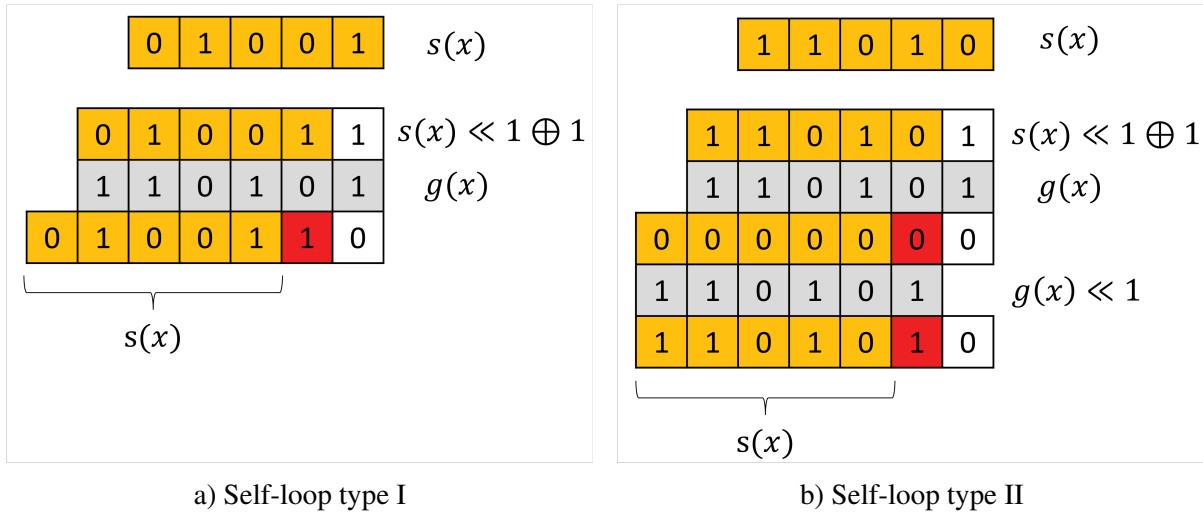


Figure 5.7 Representation of the self-loop *next* elements for the two syndromes exceptions in Figure 5.6

are referred to as self-loop syndromes types I and II. It can be seen that in both cases, the *next* element corresponds to the element itself. In order to understand why the *next* element is the syndrome itself and to make sure that there is no other case than these two, we performed an analysis of such syndromes.

For the case presented in Figure 5.7a, applied to even-parity polynomials, and based on the operations needed to obtain the *next* element, it can be seen that:

$$\begin{aligned}
 (\mathbf{s} \ll 2) \oplus (1 \ll 1) &= (\mathbf{s} \ll 1) \oplus 1 \oplus \mathbf{g} \\
 \implies (\mathbf{s} \ll 2) \oplus (\mathbf{s} \ll 1) &= \mathbf{g} \oplus (1 \ll 1) \oplus 1
 \end{aligned} \tag{5.6}$$

this equation can be expressed as:

$$s_{i-1} = g_{i+1} \oplus s_i \quad \forall (n-1) \geq i \geq 1 \tag{5.7}$$

with $s_{n-1} = 0$. Because $g(x)$ has even parity, it can be shown $s_0 \neq g_1$.

This equation can be applied to the generator polynomial to retrieve the even exception of this

$g(x)$:

$$\begin{aligned} s_4 &= 0, & s_3 &= g_5 \oplus s_4 = 1, & s_2 &= g_4 \oplus s_3 = 0 \\ s_1 &= g_3 \oplus s_2 = 0, & s_0 &= g_2 \oplus s_1 = 1 \end{aligned} \quad (5.8)$$

At the end of the process, the only exception of type I is $\mathbf{s} = [01001] = 9$, which corresponds to the one identified in Figure 5.6.

The other exception can be expressed in the same way, using the operations described in Figure 5.7b:

$$\begin{aligned} (\mathbf{s} \ll 2) \oplus (1 \ll 1) &= (\mathbf{s} \ll 1) \oplus 1 \oplus \mathbf{g} \oplus (\mathbf{g} \ll 1) \\ \implies (\mathbf{s} \ll 2) \oplus (\mathbf{s} \ll 1) &= (\mathbf{g} \ll 1) \oplus \mathbf{g} \oplus (1 \ll 1) \oplus 1 \end{aligned} \quad (5.9)$$

which can be expressed as:

$$s_{i-1} = g_{i+1} \oplus g_i \oplus s_i \quad \forall (n-1) \geq i \geq 1 \quad (5.10)$$

with $s_{n-1} = 1$. Because $g(x)$ has even parity, it can be shown that $s_0 = g_1$. By applying this equation to the considered generator polynomial, the following is obtained:

$$\begin{aligned} s_4 &= 1, & s_3 &= g_5 \oplus g_4 \oplus s_4 = 1, & s_2 &= g_4 \oplus g_3 \oplus s_3 = 0 \\ s_1 &= g_3 \oplus g_2 \oplus s_2 = 1, & s_0 &= g_2 \oplus g_1 \oplus s_1 = 0 \end{aligned} \quad (5.11)$$

At the end of the process, the only exception of type II is $\mathbf{s} = [11010] = 26$, which also corresponds to the one identified in Figure 5.6.

We will now study the case of odd-parity generator polynomials. Since the addition of such polynomials to a syndrome changes the parity, it can be seen that it is not possible that the next element of a syndrome $s(x)$ is itself in Figure 5.7a. However, since the generator polynomial is

added twice in 5.7b, it can be shown that the solution is:

$$s_i = g_{i+1} \quad \forall (n-1) \geq i \geq 0 \quad (5.12)$$

It can also be expressed as $\mathbf{s} = (\mathbf{g} \gg 1)$. Therefore, only type II self-loops exist for odd-parity generator polynomials.

Studying these elements is important because doing so provides insight into the nature of the solutions to be expected. The *next* elements have the potential to generate numerous candidates. Let us assume that $N - 1$ bit positions have been forced and that the syndrome is one of these two *next* elements. If the associated P_1 leads to the remaining bit error being at position k within the packet, then the same $N - 1$ forced bits along with any position $k + i < m + n$ with $i \geq 1$ also constitute a valid error pattern. For example, this *next* element in Table 5.3 is at index 26. For this syndrome value, the single error position P_1 is 14, and the *next* element itself is 26. If after forcing $(N - 1)$ errors, we obtain this syndrome and the remaining length is 50 bits, we get a first candidate error pattern, with errors at a forced position and at $(F_{N-1} + 1 + P_1)$. In the very next step, the update syndrome is 26 again. As the remaining length is now 49 bits, another candidate is found, with forced position F_{N-1} and position P_1 both increased by one. At each step, a new candidate is appended to the list until reaching the end of the message.

5.4.5 Syndromes with no solution for single error

We also observed an exception in the single error position search. Based on the parity of both the generator polynomial and the syndrome, we are able to determine whether the number of errors that occurred in the packet is odd or even. Based on this knowledge, a particular entry of the table represented in Table 5.3 is worthy of interest. The syndrome $\mathbf{s} = 19 = [10011]$ has an odd number of non-null coefficients, and is thus expected to have an associated single error position. However, Table 5.3 shows that there is no single error position for this syndrome, for any possible packet length. Figure 5.8 illustrates the single error correction applied to this

syndrome. It can be clearly seen that at each step, the resulting syndrome corresponds to the originally computed one, and therefore, there is no possibility of having only one non-null coefficient at any step of the process.

Similarly to the *next* element cycles described previously, it can be seen that for this exception to be realized, the following equality must be met for an even parity generator polynomial:

$$(\mathbf{s} \ll 1) = \mathbf{s} \oplus \mathbf{g} \implies (\mathbf{s} \ll 1) \oplus \mathbf{s} = \mathbf{g} \quad (5.13)$$

which can be expressed as:

$$s_{i-1} = g_i \oplus s_i \quad \forall (n-1) \geq i \geq 1 \quad (5.14)$$

with $s_{n-1} = 1$. It can easily be shown that $s_0 = 1$ for even parity generator polynomials, that the syndrome therefore never contains a single error pattern. Again, applying this equation to the generator polynomial used here yields:

$$\begin{aligned} s_4 = 1, \quad s_3 = g_4 \oplus s_4 = 0, \quad s_2 = g_3 \oplus s_3 = 0 \\ s_1 = g_2 \oplus s_2 = 1, \quad s_0 = g_1 \oplus s_1 = 1 \end{aligned} \quad (5.15)$$

At the end of the process, the syndrome is $\mathbf{s} = [10011] = 19$, which is the one identified in Table 5.3.

For odd parity generator polynomials, syndromes with no solution must belong to a cycle

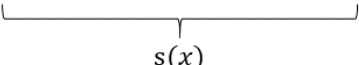
	1	0	0	1	1	$s(x)$
1	1	0	1	0	1	$g(x)$
1	0	0	1	1	0	
						

Figure 5.8 Single error correction method performed on syndrome $s(x) = x^4 + x + 1$ using a generator polynomial $g(x) = x^5 + x^4 + x^2 + 1$ (even parity)

comprising an even number of syndromes where an even parity syndrome leads to an odd parity syndrome after the addition of the generator polynomial, and vice versa. Let us now investigate the conditions for having a pair of syndromes with no solution (i.e., the shortest such cycles). Let \mathbf{s} and \mathbf{s}' be these two syndromes. They must meet the following expression:

$$\begin{aligned} (\mathbf{s}' \ll 1) \oplus \mathbf{s} = \mathbf{g} \text{ and } (\mathbf{s} \ll 1) \oplus \mathbf{s}' = \mathbf{g} \\ \implies (\mathbf{s}' \oplus \mathbf{s}) \ll 1 = \mathbf{s} \oplus \mathbf{s}' \end{aligned} \quad (5.16)$$

which can be expressed as:

$$s_{i-1} \oplus s'_{i-1} = s_i \oplus s'_i \quad \forall (n-1) \geq i \geq 1 \quad (5.17)$$

with $s_{n-1} = s'_{n-1}$. It follows that $s_i = s'_i, \forall i$, and therefore, there is no cycle of two elements, one leading to the other, for odd parity generator polynomials. Further investigation is required to conclude on the existence of longer cycles.

In Table 5.4, we present the decimal values of the syndrome exceptions for different generator polynomials. In the table, the syndrome whose *next* element is itself is denoted as "Self-loop". As has been demonstrated, there are two such types of elements. A syndrome that does not provide a single error candidate is denoted "No single error". Since CRC-32-Ethernet uses an odd parity generator polynomial, there is no such syndrome identified, and it exhibits a type II self-loop.

Having this knowledge on the structure of cycles and exceptions in CRC error correction can help save computations if such a syndrome is spotted as the receiver. By identifying an exception, we can avoid unnecessary computations while ensuring the exhaustive list of error patterns is obtained.

Table 5.4 Value of syndrome exceptions for commonly used generator polynomials (CRC-8-CCITT, CRC-16-CCITT, CRC-24-BLE and CRC-32-Ethernet)

	Self-loop (type I)	Self-loop (type II)	No single error
CRC-8	126	131	253
CRC-16	30 735	34 832	61 471
CRC-24	8 388 324	8 389 421	16 776 649
CRC-32	_____	2 187 366 107	_____

5.5 Performance and complexity

In this section, we compare different performance aspects of the proposed CRC-based error correction method using an optimized table (which we will refer to as **CRC-ECOT**) to several other CRC-based error correction methods, listed next:

- **Arithmetic operations (CRC-ECA)**: the method described in (Boussard *et al.*, 2020a), which generates the candidate list using logical operations on the fly, and does not require storing a table.
- **Explicit lookup table (CRC-ECEXP)**: the traditional lookup table approach (Shukla & Bergmann, 2004), which is based on storing the syndromes and their associated error positions. ECEXP (explicit) means that the error positions are explicitly inserted in the lookup table.
- **Implicit lookup table (CRC-ECIMP)**: a proposed design of a lookup table approach similar to CRC-ECEXP, but where the error positions are not added to the table. The values of the error positions correspond to an implicit index (specific order in which the error positions are scanned) of the associated syndrome, which reduces the memory needed to implement such tables.
- **Exhaustive search (CRC-ECES)**: this corresponds to the arithmetic brute force scheme. No table is required in this method, but all the possible combinations of N -error positions are successively tested to determine which ones lead to the computed syndrome.

Note that for our tests, we implemented the EC-ECOT version, as proposed in Algorithm 5.2, where the table size is smaller since steps 9 and 11 are not included.

5.5.1 Computational complexity

The methods compared have the same capability to correct multiple errors using the CRC syndrome. In terms of computational complexity, the CRC-ECOT method requires fewer operations as most of the computations are performed offline and then integrated into the table. We propose comparing the complexity of the method to that of the CRC-ECA and CRC-ECES approaches. Upon reception of a corrupted packet of m bits, the CRC-ECES method would test every error pattern up to N errors, i.e., flips the error positions of the pattern and then computes the syndrome over the reconstructed packet. If the syndrome is null, a valid error pattern is found.

The complexity in this section is expressed as the number of additions to perform with \mathbf{g} . In the case of CRC-ECES, m such operations are required for each long division. Thus, for the search of a single error, there are m long divisions to perform in order to test every possible error position, yielding m^2 operations. By extending this process to N errors, we obtain a global complexity of $O(m^{N+1})$.

The CRC-ECA method only performs one long division for a single error search, which can be performed through m additions of \mathbf{g} . Extending this method to search for several errors requires setting $(N - 1)$ forced positions, and a long division must be performed on the remaining length of the packet. A double-bit error search thus requires m^2 operations. Generalized to the search of N -error patterns, it yields a global complexity of $O(m^N)$, as demonstrated in (Boussard *et al.*, 2020a).

The complexity of the proposed CRC-ECOT method can be expressed through the complexity required to build the table and through the complexity required to perform the identification of the candidate error patterns. The former is highly complex, but is performed offline, prior to the communication. We will thus focus on the latter. For CRC-ECOT, the complexity up

to double-bit error correction is extremely low thanks to the *next* element column integrated into the table. Single error correction requires a single lookup, while double error correction requires m table lookups. When the number of errors is increased, forced positions must be set. Thus, when searching for N errors, $(N - 2)$ positions must be tested and the global complexity of the proposed approach will be $O(m^{N-2})$ additions of g times m table lookups. These gains are significant since the complexity increases significantly as N increases. Note that we could completely eliminate the arithmetic operations at the expense of a larger lookup table by storing $[s' \oplus g] \gg 1$ and $s' \gg 1$ for all syndrome values⁴ in steps 9 and 11 of Algorithm 5.3. This would allow to further increase the speed for $N > 2$ by a new global complexity of $O(m^{N-1})$ table lookups.

We tested a C implementation of the CRC-ECA, the CRC-ECOT and both the CRC-ECEXP and CRC-EXIMP approaches to compare their processing speeds on a Raspberry Pi model 4 (Foundation, 2019), with a Broadcom BCM2711 processor, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5 GHz and 8 GB RAM. The test results are available in Figure 5.9, with log-log scales.

First, it can be seen that when searching for single errors, as in Figure 5.9a, the processing time per symbol of the CRC-ECOT is independent of the length of the packet. In fact, as the table is syndrome-indexed, when a corrupted packet is received, we only need to read the content of the table at the computed syndrome's entry, regardless of the packet's length.

On the other hand, the CRC-ECEXP and CRC-ECIMP lookup table-based methods must scan the packet in order to search for potential single error positions, which leads to higher processing times as the maximum packet length increases. In our example, the maximum packet length considered is 2500 bytes. The processing time per syndrome on the tested CPU for the proposed CRC-ECOT method is 100 ns, on average, whereas the CRC-ECA method's processing time ranges from 1.7 μ s for the smallest payload (36 bits) to 300 μ s for the largest payload considered. Both lookup table-based methods offer a constant processing time since they must scan the whole table for any computed syndrome. This processing time is 120 μ s on the tested architecture. The

⁴ Note that depending on the architecture on which the algorithm is implemented and the generator polynomial of interest, it may be less complex to perform $s' \gg 1$ than to retrieve it from a table.

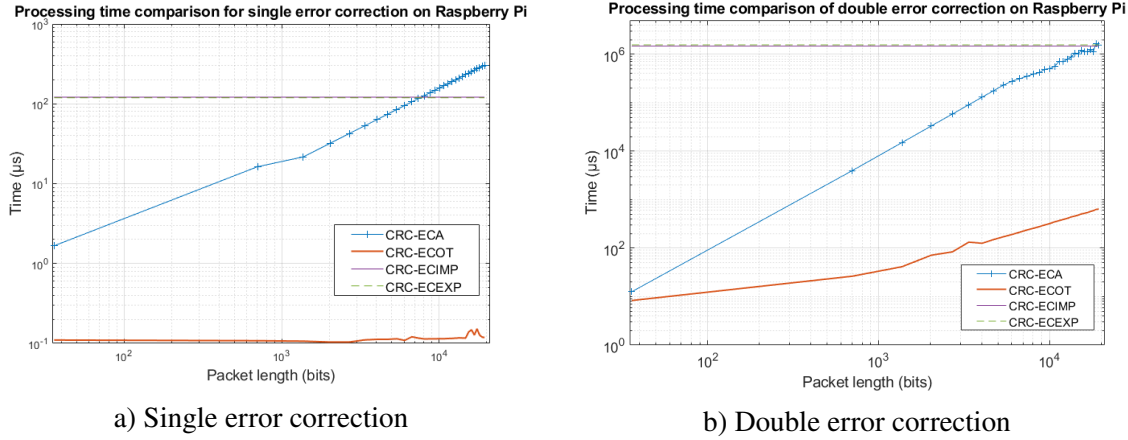


Figure 5.9 Comparison of the average processing time per syndrome for single and double error corrections for state-of-the-art CRC-based error correction (Boussard *et al.*, 2020a) (CRC-ECA), the proposed optimized table (CRC-ECOT), the implicit table (CRC-ECIMP) and explicit table (Shukla & Bergmann, 2004) (CRC-ECEXP) applied to CRC16-CCITT

proposed method's speedup for single error correction is thus 1 200 times, as compared to these table approaches, and ranges from 17 to 3000 times faster, as compared to CRC-ECA depending on the packet size.

The double error correction case illustrated in Figure 5.9b shows that again, CRC-ECOT processes each syndrome much faster than do all the other tested methods. Here, it can be seen that the processing time is not constant for CRC-ECOT as the packet length increases. For double error correction, we must consider the *next* element of each syndrome for the whole length of the packet. The processing time thus depends on the packet length. It is also interesting to note that as the packet length increases, the processing time gains also increase. When considering the smallest packet length (36bits), the average processing time for double error correction is 8.3 μs for the proposed method and 12.6 μs for the arithmetic method, which gives a time ratio of 1.5 between the two methods. This ratio increases with the packet length, and is ultimately very significant for the largest payloads. The proposed method requires 642 μs to process a syndrome while the arithmetic method needs 1.5 s, yielding a time ratio of about 2300. Of course, these gains are due to the design of the proposed method as most of the computation is performed offline, prior to the communication. CRC-ECIMP and CRC-ECEXP processing times

for double-bit error correction are still constant at 1.5 s. The proposed method's speedup goes from 174 000 for the smallest packets to 2300 for the largest ones, as compared to these table approaches, depending on the packet size. Note that these speedups can be further increased by storing the values of step 9 and 11 of Algorithm 5.3 in the table, but at the cost of increased memory requirements.

5.5.2 Memory requirements

We also compared the memory required to store the CRC-ECOT and CRC-ECEXP tables in Table 5.5, when using a packet of 1500 bytes, as it is the largest payload available in Ethernet (MTU). We compared these approaches for various generator polynomials:

- CRC-8-CCITT, where $g(x) = x^8 + x^2 + x + 1$.
- CRC-16-CCITT, used to protect the headers of 802.11 (Association, 2016) and in low consumption 802.15.4 (Association, 2011) communications, where $g(x) = x^{16} + x^{12} + x^5 + 1$.
- CRC-24-BLE, used to protect Bluetooth Low Energy (SIG, 2013) packets, where $g(x) = x^{24} + x^{10} + x^9 + x^6 + x^4 + x^3 + x + 1$.

Table 5.5 Comparison of the memory required for the tables in the CRC-ECEXP (Shukla & Bergmann, 2004) and the proposed CRC-ECOT approaches. The packet considered here has a length of 1500 bytes. Note that the arithmetic method is table-free, and thus, its memory requirements remain negligible for any case described in this table

N	CRC-8-CCITT (cycle = $2^7 - 1$)		CRC-16-CCITT (cycle = $2^{15} - 1$)		CRC-24-BLE (cycle = $2^{23} - 1$)		CRC-32-Ethernet (cycle = $2^{32} - 1$)	
	LUT	ECOT	LUT	ECOT	LUT	ECOT	LUT	ECOT
1	36 kB	256 B	48 kB	131 kB	60 kB	50.5 MB	72 kB	17.7 GB
2	360 MB	1 kB	432 MB	524 kB	504 MB	202 MB	576 MB	69.3 GB
3	2.02 TB	1 kB	2.30 TB	524 kB	2.60 TB	202 MB	2.88 TB	69.3 GB
4	7.77 PB	1 kB	8.64 PB	524 kB	9.50 PB	202 MB	10.4 PB	69.3 GB

- CRC-32-Ethernet, used to protect the entire packet in Ethernet (Association, 2018) protocol, where $g(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$.

The CRC-ECEXP approach calls for storage of a syndrome along with the corresponding error positions for every possible error case, yielding the following memory requirement:

$$M_{\text{CRC-ECEXP}} = \binom{m}{N} \times [\text{length}(\mathbf{s}) + (2 \times N)] \text{ bytes} \quad (5.18)$$

where m is the bit length of the payload, $\text{length}(\mathbf{s})$ corresponds to the byte length of the syndrome, and N is the maximum number of errors. We considered positions to be integer variables of 2 bytes, since the maximum single position is 12 000 in this implementation.

We also propose a comparison with the CRC-ECIMP method, which is an implementation requiring less memory than CRC-ECEXP. The implicit table approach consists in indexing a syndrome with its associated error position, which reduces the size of the table:

$$M_{\text{CRC-ECIMP}} = \binom{m}{N} \times \text{length}(\mathbf{s}) \text{ bytes} \quad (5.19)$$

This strategy calls for knowledge of error position management when considering several errors (e.g., syndrome at index 12001 corresponds to a double error at positions (0, 1)).

The proposed CRC-ECOT approach requires listing, for all the syndromes, the *next* element, which is always of size $\text{length}(\mathbf{s})$ bytes, as well as the error position P_1 . Note that considering that the table is initialized to (-1) , a negative element, the number of bits required to store P_1 is $\log_2(\text{cycle} + 1) + 1$. As most *cycle* lengths are $(2^{n-1} - 1)$ as shown in Table 5.5, it would take n bits to store P_1 . It can be seen that the *cycle* length for CRC-32 is $2^n - 1$, yielding the need of an additional bit, for a total of $(n + 1)$ bits to store P_1 (i.e., 33 bits⁵ for CRC-32). Except for such cases, the memory required can thus be expressed as:

$$M_{\text{CRC-ECOT}} = \begin{cases} 2^n \times \text{length}(\mathbf{s}) \text{ bytes,} & \text{if } N = 1 \\ 2^n \times 4 \times \text{length}(\mathbf{s}) \text{ bytes,} & \text{if } N > 1 \end{cases} \quad (5.20)$$

⁵ Note that some architectures will not support 33-bit numbers efficiently, and will consider 64 bits when exceeding $2^{32} - 1$, which would double the memory required.

It can first be seen in Table 5.5 that, unlike in CRC-ECEXP, where the table size is a function of the number of errors considered, the proposed approach has a fixed length for ($N > 1$). In fact, for single error correction, the column comprising the *next* elements is not needed, and with double error correction, the columns can store both the *next* element and the results of step 9 and 11 of Algorithm 5.3, which multiply the storage needed by 4. The table size for the proposed approach is less than for CRC-ECEXP, for all generator polynomial lengths n when ($N > 2$). The proposed table is also smaller for $N = 2$ up to $n = 24$, and up to $n = 8$ for $N = 1$, due to how cycles are handled in our method. It should be recalled that the arithmetic method does not need to store a table at all, but this is at the cost of much higher computational complexity. The best compromise will depend on the application and hardware on which the method is implemented. For instance, the proposed method is very appealing for small CRCs or for correcting a high number of errors, while the arithmetic method may be more appealing for single error correction when long CRCs (e.g., CRC-32) are used.

5.5.3 Application to the correction of multiple errors

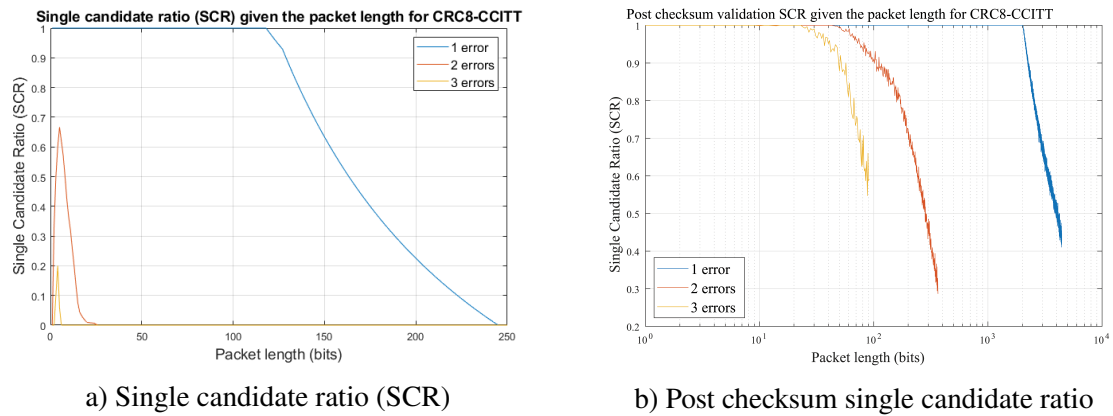


Figure 5.10 Evolution of the SCR and post checksum validation SCR for CRC-8-CCITT as a function of the packet length

The method proposed in this paper is able to output an exhaustive list of candidate error patterns having up to N errors. In practice, we assume that this method is applied in the context of relatively reliable communication channels, where corrupted packets tend to be mildly damaged.

The authors in (Tsimbalo *et al.*, 2016) demonstrated a realistic experimental scenario in which most damaged packets in a Bluetooth low energy (BLE) environment contained 3 errors or less. As our method is able to generate the list with a lower complexity, its use can be considered to increase the value of N to handle more error cases. However, the number of candidates increases significantly with N for a given packet length. In (Boussard *et al.*, 2020a), we investigated the ratio of error patterns that would output a candidate list with a single entry, thus allowing the correction of the packet, over all possible error patterns for a given N , and termed this ratio the Single Candidate Ratio (SCR). We show that CRCs with low-degree generator polynomials never reach an SCR of 100% when considering multiple bit errors, and rapidly decreases as the packet length increases.

As an example, we propose to analyze the impact of a checksum cross-validation on the SCR for the generator polynomial used in CRC-8-CCITT. The SCR of this CRC is illustrated in Figure 5.10a. It can be seen that when considering a single error, up to 127 bits (i.e., $2^7 - 1$), the SCR is 100%, which illustrates the importance of the *cycle* length, as there is no list with multiple single error candidates for packets smaller than this *cycle* value. The SCR then rapidly decreases and reaches 0% for packets greater than 245 bits. When considering double- and triple-bit error patterns, the SCR is very low, even for the smallest packet values considered. It reaches 0% for lengths of 26 and 6 bits, for double- and triple-error patterns, respectively.

Implementing a checksum validation step, as in UDP and TCP protocols, helps to significantly increase such ratios and achieve decent error correction rates on small packets, even when using CRC-8-CCITT. In Figure 5.10b, it can be seen that the SCR is noticeably higher for all the numbers of errors considered. The SCR remains at 100% when dealing packet lengths of up to 41 bits and 11 bits for double- and triple-error patterns. Moreover, it can be seen that the SCR is still over 50% for double error patterns, up to a packet length of 270 bits. The single error correction, whose associated SCR reaches 0% at 245 bits in Figure 5.10a, becomes 100% up to a significant length of 2000 bits with checksum validation, which allows reconstruction of much more error cases. The method should perform even better with an actual BLE system with small packets (up to 39 bytes and now increased to a maximum of 255 bytes) protected by a strong CRC-24.

Thus, when increasing the number of errors, validation steps such as a checksum cross-validation will help maintain a high correction rate when the candidate list size increases significantly. This allows to take advantage of the processing speed gains of the proposed method and to consider larger values of N . Increasing N brings a lot of changes and challenges in literature methods as they are designed for a specific number of errors. We demonstrated that table-based CRC-ECEXP and CRC-ECIMP produces intractable table sizes from $N = 3$. The complexity of CRC-ECA methods is greatly affected whenever N is increased. The proposed CRC-ECOT replaces the arithmetic operations with successive table lookups, thus reducing the global complexity. As well, the table size is easily managed and constant for $N \geq 2$, which makes it very appealing for multiple error correction.

5.6 Conclusion and perspectives

In this paper, we propose an optimized table-based method for performing multiple error correction based on the CRC syndrome. The approach offers a low complexity alternative to the state-of-the-art error correction method as it generates a table that contains precomputed operations required to perform error pattern searches, avoiding most to all arithmetic operations. Thanks to offline table generation, the proposed approach achieves the same error correction performance as the state-of-the-art approaches while providing computations savings and thus improving the processing speeds. We show through simulations that the proposed method achieves significant speed gains over the table-free arithmetic method, and is between $2300\times$ and $3000\times$ faster when generating the list of double and single error patterns, respectively.

Thus, reducing the complexity offers the possibility of increasing the number of errors to consider while keeping the same processing time. Since this greatly increases the number of candidates in the output error pattern list, we present a validation step that increases the correction rate for lists containing many candidates. Future work will look at integrating the proposed CRC-based error correction solution into a complete cross-layer receiver architecture in order to benefit from other validation mechanisms available in the protocol stack. The objective is to further reduce the list

size or being able to determine the best candidate out of several in order to reconstruct the best signal quality at the receiver side (e.g., visual quality, in the case of video content transmission).

CONCLUSION AND RECOMMENDATIONS

In this thesis, we presented the CRC-based methods for multiple error correction that we developed throughout our research. These approaches are based on the generation of a candidate list containing the exhaustive set of error patterns containing a determined number of errors or less that would produce the computed syndrome at the receiver. We presented two different approaches to generate such list:

- A table-free approach, based on arithmetic operations on the syndrome that is able to output the candidate list of error patterns for any generator polynomial and any packet length. Moreover, the maximum number of errors per pattern to consider can be set to perform multiple error correction. However, increasing such value also significantly increases the computational complexity of the method, as well as the average size of the resulting candidate list. Thus, this method is designed to operate on channels and environments where the average number of errors in a corrupted packet is small. Used alone, such method can only perform error correction when the list contains a single candidate. In order to avoid such limitations, additional validation steps were added, by cross-validating the reconstruction with the checksum and ensuring the decodability of the bitstream in the case of video communications. By applying such error correction method to transmission simulation of H.264 and HEVC encoded video over wireless channels, significant gains over state-of-the-art method are observed. Depending on the targeted environment, the proposed method offers PSNR gains up to 5.7 dB in BLE environments.
- The second approach is based on generating optimized table that share the same error pattern generation process as the table-free method. Instead of performing arithmetic operations on-the-fly, this approach stores precomputed operations in a table. This table is organized such that single error correction can be performed in a single table call, and double error correction can be performed in m table navigation, where m is the payload length of the protected packet. Building such table reduces significantly the time required to process error

correction on a packet compared to table-free approach. In our tests performed on Raspberry Pi 4 model B, the proposed approach was able to perform double error correction on packets of 1250 bytes in approximately 300 μ s while it takes 600 ms to table-free approach. In fact, this optimized table approach requires memory storage contrary to the table-free method. However, it has been shown that the size of the proposed table does not grow exponentially with the number of errors. Whether considering two errors or more, the table size remains the same, contrary to literature lookup table-based approaches.

Further works and perspectives on this aspect will look at reducing further the number of candidates in the output list. As for now, it can happen that several candidates pass the whole validation process. In such case, we do not perform error correction, even if all the candidates are decodable streams, as it can produce visual artifacts on the video in case of miscorrection. A research field of interest would be the candidate selection using machine learning-based classification. By training or using a neural network able to estimate the visual quality of several candidates, we would be able to increase the percentage of error cases the method can handle and thus achieve a better visual quality for the end user.

BIBLIOGRAPHY

- Aiswarya, A. & George, A. (2017). Fixed latency serial transceiver with single bit error correction on FPGA. *2017 International Conference on Trends in Electronics and Informatics (ICEI)*, pp. 902–907.
- Akbari, A., Trocan, M. & Granado, B. (2017). Sparse recovery-based error concealment. *IEEE Transactions on Multimedia*, 19(6), 1339–1350.
- Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M. & Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE communications surveys & tutorials*, 17(4), 2347–2376.
- Al-Mualla, M., Canagarajah, N. & Bull, D. (1999). Temporal error concealment using motion field interpolation. *Electronics Letters*, 35(3), 215–217.
- Alexander, P., Haley, D. & Grant, A. (2011). Cooperative intelligent transport systems: 5.9-GHz field trials. *Proceedings of the IEEE*, 99(7), 1213–1235.
- Alkachouh, Z. & Bellanger, M. G. (2000). Fast DCT-based spatial domain interpolation of blocks in images. *IEEE Transactions on Image Processing*, 9(4), 729–732.
- Anderson, J. & Lin, C. (1986). M-algorithm decoding of channel convolutional codes. *Conf. Proc., 20th Annu. Conf. Inform. Sci. Syst.*
- Arikan, E. (2009). Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on information Theory*, 55(7), 3051–3073.
- Arndt, J. (2011). Binary Polynomials. In *Matters Computational* (pp. 822–863). Springer.
- Association, I. S. (2010). IEEE 1609.4-2010 - IEEE Standard for Wireless Access in Vehicular Environments (WAVE) - Multi-channel operations.
- Association, I. S. (2011). IEEE 802.15.4-2011 - IEEE Standard for Local and metropolitan area networks—Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs).
- Association, I. S. (2016). IEEE 802.11-2016, IEEE Standard for Local and Metropolitan Area Networks—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.
- Association, I. S. (2018). IEEE 802.3-2018 - IEEE Standard for Ethernet.

- Atzori, L., De Natale, F. G. B. & 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.
- Babaie, S., Zadeh, A. K., Es-hagi, S. H. & Navimipour, N. J. (2009). Double bits error correction using CRC method. *2009 Fifth International Conference on Semantics, Knowledge and Grid*, pp. 254–257.
- Bhatnagar, N. (2018). *Mathematical Principles of the Internet, Volume 1: Engineering*. CRC Press.
- Boussard, V. & Coulombe, S. (2020). Methods and systems for bit error determination and correction field. PCT/CA2020/050724, 2020-05-27, [Available] <https://patentscope.wipo.int/search/en/detail.jsf?docId=WO2020237377>.
- Boussard, V., Coulombe, S., Coudoux, F.-X. & Corlay, P. (2020a). Table-Free Multiple Bit-Error Correction Using the CRC Syndrome. *IEEE Access*, 8, 102357–102372.
- Boussard, V., Golaghazadeh, F., Coulombe, S., Coudoux, F.-X. & Corlay, P. (2020b). Robust H.264 Video Decoding Using CRC-Based Single Error Correction And Non-Desynchronizing Bits Validation. *2020 IEEE International Conference on Image Processing (ICIP)*, pp. 1098–1102.
- Boussard, V., Coulombe, S., Coudoux, F.-X. & Corlay, P. (2021a). *Enhanced CRC-Based Correction of Multiple Errors with Candidate Validation*. Submitted to Signal Processing: Image Communications.
- Boussard, V., Coulombe, S., Coudoux, F.-X. & Corlay, P. (2021b). *CRC-Based Correction of Multiple Errors Using an Optimized Lookup Table*. Submitted for publication.
- Boyd, S. & Vandenberghe, L. (2018). *Introduction to applied linear algebra: vectors, matrices, and least squares*. Cambridge university press.
- Boyd, S., Parikh, N. & Chu, E. (2011). *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc.
- Braden, R., Borman, D. & Partridge, C. (1989). Computing the internet checksum. *ACM SIGCOMM Computer Communication Review*, 19(2), 86–94.
- Bross, B., Chen, J., Ohm, J.-R., Sullivan, G. J. & Wang, Y.-K. (2021). Developments in International Video Coding Standardization After AVC, With an Overview of Versatile Video Coding (VVC). *Proceedings of the IEEE*, 1-31.

- Byongsu, H., Jonghyon, J. & Cholsu, R. (2019). An improved multi-directional interpolation for spatial error concealment. *Multimedia Tools and Applications*, 78(2), 2587–2598.
- Caron, F. & Coulombe, S. (2013). Video error correction using soft-output and hard-output maximum likelihood decoding applied to an H.264 baseline profile. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(7), 1161–1174.
- Chen, T., Zhang, X. & Shi, Y. . (2003). Error concealment using refined boundary matching algorithm. *International Conference on Information Technology: Research and Education, 2003. Proceedings. ITRE2003.*, pp. 55-59.
- 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 Transactions on Multimedia*, 10(1), 2-15.
- Chung, B. & Yim, C. (2020). Bi-Sequential Video Error Concealment Method Using Adaptive Homography-Based Registration. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(6), 1535-1549.
- Chung, B. & Yim, C. (2014). Hybrid error concealment method combining exemplar-based image inpainting and spatial interpolation. *Signal Processing: Image Communication*, 29(10), 1121 - 1137.
- Collotta, M., Pau, G., Talty, T. & Tonguz, O. K. (2018). Bluetooth 5: A concrete step forward toward the IoT. *IEEE Communications Magazine*, 56(7), 125–131.
- Dahlman, E., Parkvall, S. & Skold, J. (2013). *4G: LTE/LTE-advanced for mobile broadband*. Academic press.
- Demirtas, A. M., Reibman, A. R. & Jafarkhani, H. (2011). Performance of H.264 with isolated bit error: Packet decode or discard? *2011 18th IEEE International Conference on Image Processing*, pp. 949–952.
- Dholakia, A. (2012). *Introduction to convolutional codes with applications*. Springer Science & Business Media.
- Duhamel, P. & Kieffer, M. (2009). *Joint source-channel decoding: A cross-layer perspective with applications in video broadcasting*. Academic Press.
- ETSI, T. (2013). Intelligent Transport Systems (ITS); Access layer specification for Intelligent Transport Systems operating in the 5 GHz frequency band. EN 302 663 V1.2.1.

- ETSI, T. (2017). LTE - Evolved Universal Terrestrial Radio Access (E-UTRA) Multiplexing and channel coding. TS 136 212 V14.2.0.
- 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.
- Festag, A. (2015). Standards for vehicular communication—from IEEE 802.11p to 5G. *e & i Elektrotechnik und Informationstechnik*, 132(7), 409–416.
- FFmpeg. (2019). FFmpeg codec documentation. [Available] <https://ffmpeg.org/ffmpeg-codecs.html#Video-Decoders>.
- Forney, G. D. (1973). The viterbi algorithm. *Proceedings of the IEEE*, 61(3), 268–278.
- Foundation, T. R. P. (2018). Raspberry PI 3 Model B+. [Available] <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/>.
- Foundation, T. R. P. (2019). Raspberry PI 4 Model B. [Available] <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/>.
- Gallager, R. (1962). Low-density parity-check codes. *IRE Transactions on information theory*, 8(1), 21–28.
- Golaghazadeh, F., Coulombe, S., Coudoux, F.-X. & Corlay, P. (2017a). 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.
- Golaghazadeh, F., Coulombe, S., Coudoux, F.-X. & Corlay, P. (2017b). Low complexity H.264 list decoder for enhanced quality real-time video over IP. *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 1–6.
- Golaghazadeh, F., Coulombe, S., Coudoux, F.-X. & Corlay, P. (2018). The impact of H.264 non-desynchronizing bits on visual quality and its application to robust video decoding. *2018 12th International Conference on Signal Processing and Communication Systems (ICSPCS)*, pp. 1–7.
- Gomes, P., Olaverri-Monreal, C. & Ferreira, M. (2012). Making Vehicles Transparent Through V2V Video Streaming. *IEEE Transactions on Intelligent Transportation Systems*, 13(2), 930–938.
- Gomez, C., Oller, J. & Paradells, J. (2012). Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors*, 12(9), 11734–11753.

- Gupta, A. & Jha, R. K. (2015). A survey of 5G network: Architecture and emerging technologies. *IEEE access*, 3, 1206–1232.
- Hachenberger, D. & Jungnickel, D. (2020a). Basis Representations and Arithmetics. In *Topics in Galois Fields* (pp. 355–425). Springer.
- Hachenberger, D. & Jungnickel, D. (2020b). Shift Register Sequences. In *Topics in Galois Fields* (pp. 427–487). Springer.
- Jelinek, F. (1969). Fast sequential decoding algorithm using a stack. *IBM journal of research and development*, 13(6), 675–685.
- Jetzek, U. (2018). *Galois Fields, Linear Feedback Shift Registers and their Applications*. Carl Hanser Verlag GmbH Co KG.
- Jinghong Zheng & Lap-Pui Chau. (2003). A motion vector recovery algorithm for digital video using Lagrange interpolation. *IEEE Transactions on Broadcasting*, 49(4), 383–389.
- Jyothi, S. N. & Vardhan, K. V. (2016). Design and implementation of real time security surveillance system using IoT. *2016 International Conference on Communication and Electronics Systems (ICCES)*, pp. 1–5.
- Kazemi, M., Ghanbari, M. & Shirmohammadi, S. (2020). The Performance of Quality Metrics in Assessing Error-Concealed Video Quality. *IEEE Transactions on Image Processing*, 29, 5937–5952.
- Kenney, J. B. (2011). Dedicated short-range communications (DSRC) standards in the United States. *Proceedings of the IEEE*, 99(7), 1162–1182.
- Kim, D., Woo, S., Lee, J.-Y. & Kweon, I. S. (2019). Deep video inpainting. *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5792–5801.
- Koloda, J., Østergaard, J., Jensen, S. H., Sánchez, V. & Peinado, A. M. (2013). Sequential Error Concealment for Video/Images by Sparse Linear Prediction. *IEEE Transactions on Multimedia*, 15(4), 957–969.
- Kung, W. ., Kim, C. . & Kuo, C. . J. (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 Transactions on Consumer Electronics*, 39(3), 455–460.

- Lakovic, K., Villaseñor, J. & Wesel, R. (1999). Robust joint Huffman and convolutional decoding. *Gateway to 21st Century Communications Village. VTC 1999-Fall. IEEE VTS 50th Vehicular Technology Conference (Cat. No.99CH36324)*, 5, 2551-2555.
- Lam, W. M., Reibman, A. R. & Liu, B. (1993). Recovery of lost or erroneously received motion vectors. *1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*, 5, 417-420.
- Levine, D., Lynch, W. E. & Le-Ngoc, T. (2010). Iterative joint source–channel decoding of H.264 compressed video. *Signal Processing: Image Communication*, 25(2), 75–87.
- Li, Y., Yang, Z., Li, K. & Qu, L. (2020). A new algorithm on the minimal rational fraction representation of feedback with carry shift registers. *Designs, Codes and Cryptography*, 88(3), 533–552.
- Li, Z.-N., Drew, M. S. & Liu, J. (2004). *Fundamentals of multimedia*. Springer.
- Liu, C., Ma, R. & Zhang, Z. (2012). Error concealment for whole frame loss in HEVC. In *Advances on Digital Television and Wireless Multimedia Communications* (pp. 271–277). Springer.
- Liu, J., Zhai, G., Yang, X., Yang, B. & Chen, L. (2015). Spatial Error Concealment With an Adaptive Linear Predictor. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(3), 353-366.
- Liu, X., Wu, S., Xu, X., Jiao, J. & Zhang, Q. (2018). Improved polar SCL decoding by exploiting the error correction capability of CRC. *IEEE Access*, 7, 7032–7040.
- Luo, J., Bowers, K. D., Oprea, A. & Xu, L. (2012). Efficient software implementations of large finite fields GF (2ⁿ) for secure storage applications. *ACM Transactions on Storage (TOS)*, 8(1), 1–27.
- Marin, C. (2009). *Vers une solution réaliste de décodage source-canal conjoint de contenus multimédia*. (PhD Thesis, Université Paris Sud - Paris XI).
- Marin, C., Leprovost, Y., Kieffer, M. & Duhamel, P. (2010). Robust mac-lite and soft header recovery for packetized multimedia transmission. *IEEE Transactions on Communications*, 58(3), 775–784.
- MathWorks. (2018). 802.11p Packet Error Rate Simulation for a Vehicular Channel.
- MathWorks. (2019). End-to-End Bluetooth Low Energy PHY Simulation with RF Impairments and Corrections.

- Ming Jia, Jiangtao Wen, Shaozhi Ye & Xing Li. (2005). Error restricted fast MAP decoding of VLC. *IEEE Communications Letters*, 9(10), 909-911.
- Moonseo Park & Miller, D. J. (2000). Joint source-channel decoding for variable-length encoded data by exact and approximate MAP sequence estimation. *IEEE Transactions on Communications*, 48(1), 1-6.
- Mumtaz, S., Huq, K. M. S., Ashraf, M. I., Rodriguez, J., Monteiro, V. & Politis, C. (2015). Cognitive vehicular communication for 5G. *IEEE Communications Magazine*, 53(7), 109–117.
- Nafaa, A., Taleb, T. & Murphy, L. (2008). Forward error correction strategies for media streaming over wireless networks. *IEEE Communications Magazine*, 46(1), 72-79.
- Nguyen, H. & Duhamel, P. (2004). Iterative joint source-channel decoding of variable length encoded video sequences exploiting source semantics. *2004 International Conference on Image Processing, 2004. ICIP'04.*, 5, 3221–3224.
- Nguyen, H. & Duhamel, P. (2009). Iterative joint source-channel decoding of VLC exploiting source semantics over realistic radio-mobile channels. *IEEE transactions on communications*, 57(6), 1701–1711.
- 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. *CCECE 2010*, pp. 1–4.
- Niu, K. & Chen, K. (2012). CRC-aided decoding of polar codes. *IEEE communications letters*, 16(10), 1668–1671.
- Nshimiyimana, A., Agrawal, D. & Arif, W. (2016). Comprehensive survey of V2V communication for 4G mobile and wireless technology. *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, pp. 1722–1726.
- Panwar, N., Sharma, S. & Singh, A. K. (2016). A survey on 5G: The next generation of mobile communication. *Physical Communication*, 18, 64–84.
- Postel, J. et al. (1980a). *Transmission control protocol* (Report n°793). RFC Editor.
- Postel, J. et al. (1980b). *User datagram protocol* (Report n°768). RFC Editor.
- Qiang Peng, Tianwu Yang & Changqian Zhu. (2002). Block-based temporal error concealment for video packet using motion vector extrapolation. *IEEE 2002 International Conference on Communications, Circuits and Systems and West Sino Expositions*, 1, 10-14.

- Rameau, F., Ha, H., Joo, K., Choi, J., Park, K. & Kweon, I. S. (2016). A real-time augmented reality system to see-through cars. *IEEE transactions on visualization and computer graphics*, 22(11), 2395–2404.
- Sabeva, G., Jamaa, S. B., Kieffer, M. & Duhamel, P. (2006). Robust decoding of H.264 encoded video transmitted over wireless channels. *2006 IEEE Workshop on Multimedia Signal Processing*, pp. 9–13.
- Sankisa, A., Punjabi, A. & Katsaggelos, A. K. (2018). Video Error Concealment Using Deep Neural Networks. *2018 25th IEEE International Conference on Image Processing (ICIP)*, pp. 380–384.
- Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V. et al. (1996). *RTP: A transport protocol for real-time applications* (Report n°1889). RFC Editor.
- Shah, S. A. A., Ahmed, E., Imran, M. & Zeadally, S. (2018). 5G for vehicular communications. *IEEE Communications Magazine*, 56(1), 111–117.
- Shannon, C. E. (2001). A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, 5(1), 3–55.
- Shirani, S., Kossentini, F. & Ward, R. (1999). Error concealment methods, a comparative study. *Engineering Solutions for the Next Millennium. 1999 IEEE Canadian Conference on Electrical and Computer Engineering (Cat. No.99TH8411)*, 2, 835–840.
- Shukla, S. & Bergmann, N. W. (2004). Single bit error correction implementation in CRC-16 on FPGA. *Proceedings. 2004 IEEE International Conference on Field-Programmable Technology (IEEE Cat. No. 04EX921)*, pp. 319–322.
- SIG, B. (2013). Specification of the Bluetooth system. Core Version 4.1, Bluetooth SIG.
- Sjoberg, K., Andres, P., Buburuzan, T. & Brakemeier, A. (2017). Cooperative intelligent transport systems in Europe: Current deployment status and outlook. *IEEE Vehicular Technology Magazine*, 12(2), 89–97.
- Sobolewski, J. S. (2003). Cyclic redundancy check. In *Encyclopedia of Computer Science* (pp. 476–479).
- Stockhammer, T., Hannuksela, M. M. & Wiegand, T. (2003). H.264/AVC in wireless environments. *IEEE transactions on circuits and systems for video technology*, 13(7), 657–673.

- Suhring, K. (2015). H.264/AVC JM Reference Software 19.0. [Available] <http://iphone.hhi.de/suehring/tml/>.
- Sullivan, G. J. & Wiegand, T. (2005). Video Compression - From Concepts to the H.264/AVC Standard. *Proceedings of the IEEE*, 93(1), 18-31.
- Sullivan, G. J., Ohm, J., Han, W. & Wiegand, T. (2012). Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12), 1649-1668.
- Sun, H. & Kwok, W. (1995). Concealment of damaged block transform coded images using projections onto convex sets. *IEEE Transactions on Image Processing*, 4(4), 470-477.
- Superiori, L., Nemethova, O. & Rupp, M. (2007). Performance of a H.264/AVC Error Detection Algorithm Based on Syntax Analysis. *J. Mobile Multimedia*, 3(4), 314-330.
- Tripathi, S. K., Gupta, B. & Pandian, K. K. S. (2020). The Shortest Register With Non-Linear Update for Generating a Given Finite or Periodic Sequence. *IEEE Communications Letters*, 24(6), 1173-1177.
- Trudeau, L., Coulombe, S. & Pigeon, S. (2011). Pixel domain referenceless visual degradation detection and error concealment for mobile video. *2011 18th IEEE International Conference on Image Processing*, pp. 2229-2232.
- Tsimbalo, E., Fafoutis, X. & Piechocki, R. (2015). Fix it, don't bin it!-CRC error correction in Bluetooth Low Energy. *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pp. 286-290.
- Tsimbalo, E., Fafoutis, X. & Piechocki, R. J. (2016). CRC error correction in IoT applications. *IEEE Transactions on Industrial Informatics*, 13(1), 361-369.
- Wang, C., Huang, H., Han, X. & Wang, J. (2019). Video inpainting by jointly learning temporal structure and spatial details. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33, 5232-5239.
- Wen, H., Ho, P.-H. & Gong, G. (2009). A novel framework for message authentication in vehicular communication networks. *GLOBECOM 2009-2009 IEEE Global Telecommunications Conference*, pp. 1-6.
- Wenger, S. (2003). H.264/AVC over IP. *IEEE transactions on circuits and systems for video technology*, 13(7), 645-656.

- Whaiduzzaman, M., Hossain, M. R., Shovon, A. R., Roy, S., Laszka, A., Buyya, R. & Barros, A. (2020). A Privacy-Preserving Mobile and Fog Computing Framework to Trace and Prevent COVID-19 Community Transmission. *IEEE Journal of Biomedical and Health Informatics*, 24(12), 3564-3575.
- Wicker, S. B. & Bhargava, V. K. (1999). *Reed-Solomon codes and their applications*. John Wiley & Sons.
- Wiegand, T., Sullivan, G. J., Bjontegaard, G. & Luthra, A. (2003). Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7), 560-576.
- Wu, J., Liu, X. & Yoo, K. (2008). A temporal error concealment method for H.264/AVC using motion vector recovery. *IEEE Transactions on Consumer Electronics*, 54(4), 1880-1885.
- Xin Li & Orchard, M. T. (2002). Novel sequential error-concealment techniques using orientation adaptive interpolation. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(10), 857-864.
- Yao Wang & Qin-Fan Zhu. (1998). Error control and concealment for video communication: a review. *Proceedings of the IEEE*, 86(5), 974-997.
- Yim, C. & Bovik, A. C. (2011). Evaluation of temporal variation of video quality in packet loss networks. *Signal Processing: Image Communication*, 26(1), 24–38.
- Yue Wang & Songyu Yu. (2005). Joint source-channel decoding for H.264 coded video stream. *IEEE Transactions on Consumer Electronics*, 51(4), 1273-1276.
- Zanella, A., Bui, N., Castellani, A., Vangelista, L. & Zorzi, M. (2014). Internet of things for smart cities. *IEEE Internet of Things journal*, 1(1), 22–32.
- Zhou, J., Yan, B. & Gharavi, H. (2011). Efficient Motion Vector Interpolation for Error Concealment of H.264/AVC. *IEEE Transactions on Broadcasting*, 57(1), 75-80.