Authors accepted manuscript
Article published in *Signal Processing: Image Communication*. Vol. 99, Nov. 2021.
The final published version available at https://doi.org/10.1016/j.image.2021.116475

Enhanced CRC-Based Correction of Multiple Errors with Candidate Validation

Vivien Boussard, Stéphane Coulombe, François-Xavier Coudoux, Patrick Corlay

***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 on average from 1.6 dB to 7.3 dB over Bluetooth Low Energy channels with Eb/No ratio of 10 dB and 8 dB, respectively.**

***Index Terms*—Cyclic Redundancy Check, Error Correction, Wireless Communication, IEEE 802.11, Bluetooth Low Energy.**

I. Introduction

The 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 severe visual artifacts after video reconstruction of discarded regions, which decreases the viewing experience for end 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 results depends on the video content and on the size of the lost regions. Other solutions for recovering 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 is often not available to handle missing packets, due to delay constraints and/or network congestion. Cyclic Redundancy Checks (CRC) [1] are used in protocols commonly employed to detect errors in the entire packet, and may therefore be useful in error correction 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, expressed as follows:

$$\frac{d_T(x) \ll n}{g(x)} = q(x) + \frac{r_T(x)}{g(x)} \tag{1}$$

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 corresponding to an *exclusive or* (XOR) between two binary polynomials[1]. 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:

$$\frac{(d_R(x) \ll n) + r_R(x)}{g(x)} = q(x) + \frac{s(x)}{g(x)} \tag{2}$$

where $s(x)$ is the new remainder of the division, also known as the syndrome. If there is no error during transmission, the syndrome $s(x)$ is null. Otherwise, it has a non-null value indicating that one or several errors altered the transmitted packet. Usually, the non-null syndrome is handled by discarding the corrupted packet (User Datagram Protocol (UDP)) or by retransmitting the corrupted packet (Transmission Control Protocol (TCP)).

The literature contains several methods proposed 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. These can be classified under two main categories:

- spatial error concealment methods [3]–[6], which use the available information in the current frame to reconstruct the missing video areas by interpolating and predicting the corrupted video content, and
- temporal concealment methods [7]–[10], which propose a concealment of the video chunks, and leverage temporally neighboring frames to recover video content. As there is a temporal correlation in natural video content, 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 [11], [12]. Traditional error concealment methods mainly use interpolations from neighboring content to reconstruct the video, while the most recent error concealment solutions

This work was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant.

V. Boussard and S. Coulombe are with the Department of Software Engineering and Information Technology, École de technologie supérieure, Montreal, Canada. Emails: vivien.boussard.1@etsmtl.net, stephane.coulombe@etsmtl.ca. F-X Coudoux and P. Corlay are with UPHF, CNRS, Univ. Lille, ISEN, Centrale Lille, UMR 8520 - IEMN, DOAE, Valenciennes, France. Emails: {francois-xavier.coudoux, patrick.corlay}@uphf.fr.

[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 [2].

use machine learning to recover large missing areas [13]–[15]. Such methods are able to handle missing packets as they systematically discard corrupted packets when performing error concealment. The drawback with such a process is the associated loss of useful information. A corrupted received packet may have only a few errors but still contain valuable information that can help to reconstruct the missing video data, but with this process, such packets are discarded.

The literature also contains other methods that propose error correction on received corrupted packets based on available information. Such approaches can search for the most probable sent bit sequence as a function of the reliability information on the packet and/or syntax knowledge of the transmitted packet [16], [17]. Some state-of-the-art solutions also propose to use error detection codes as error correction codes. Checksums are investigated in these cases to identify bit error patterns from an erroneous checksum value [18]. CRC-based error correction has also been proposed, through the use of lookup tables (LUT) [19]–[21] 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 [22], [23], and which 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 generator polynomial used. This paper is mainly based on the theory described and discussed therein [23], with which the reader is expected to be familiar.

The present work proposes a method to perform multiple error correction based on the CRC syndrome, with candidate validation on Advanced Video Coding (AVC) [24] and High Efficiency Video Coding (HEVC) [25] encoded video content applied to widely used wireless communication applications. Its contributions are as follows:

- **Enhanced multiple error correction**: The proposed method offers the possibility of 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 methods perform error concealment in such cases.
- **Processing speed gains**: We propose reducing 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 memory management of the error search, allowing a fixed low memory storage, independent of the packet length.
- **Demonstration of gains in wireless transmissions**: We prove the usefulness of a CRC-based error correction method in practical applications by testing the method using realistic video communication scenarios over wireless networks. We apply the proposed method to wireless communication in 802.11p and Bluetooth Low Energy environments. In the latter case, the proposed approach offers significant gains of up to 10 dB in PSNR relative to error concealment methods.

This article is structured as follows. In section II, we present related works on CRC error correction found in the literature. In section III, we propose a novel method to enhance the error correction capabilities of CRC codes with additional validation steps and optimization strategies. In section IV, we discuss simulation results for two different environments, namely, Wi-Fi 802.11p and Bluetooth Low Energy. Section V concludes the work and future perspectives are discussed.

II. Related works

A. CRC-based single error correction using tables

Error correction using the CRC syndrome has already been proposed in [19]–[21]. Here, a lookup table is created prior to communication, with each possible non-null error syndrome 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. Some of the drawbacks of such a method include the fact that the lookup table must be recomputed if the maximum packet length changes, and that the method only works for a specific generator polynomial, with the entire list always scanned. However, as shown in [26], [27], this method is useful in the error correction of polar codes.

B. CRC-based single error correction using arithmetic operations

More efficient methods have recently been introduced to handle single error correction using the CRC syndrome. In [22], we proposed a method consisting in using arithmetic operations on the CRC syndrome to correct damaged packets at the receiver. The method proposes correcting single error packets through CRC error correction. For error correction, the method uses the definition of CRC codes to identify all possible single error positions given the computed syndrome at the receiver. The method is based on the definition of the CRC computation, as illustrated in (2). We can express the syndrome $s(x)$ at the receiver as:

$$s(x) = p_R(x) \mod g(x) \tag{3}$$

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 affecting 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 $s(x) = e(x) \bmod g(x)$, which can also be expressed as:

$$e(x) = s(x) + q(x).g(x) \tag{4}$$

At each step, the number of remaining non-null positions in the error vector is checked and if is equal to 1, a single error candidate is identified at that position. An example of such a single error method is provided in Fig. 1, applied on CRC-4-ITU, with generator polynomial $g(x) = x^4 + x + 1$ over 10 data bits. In this example, the computed syndrome at the

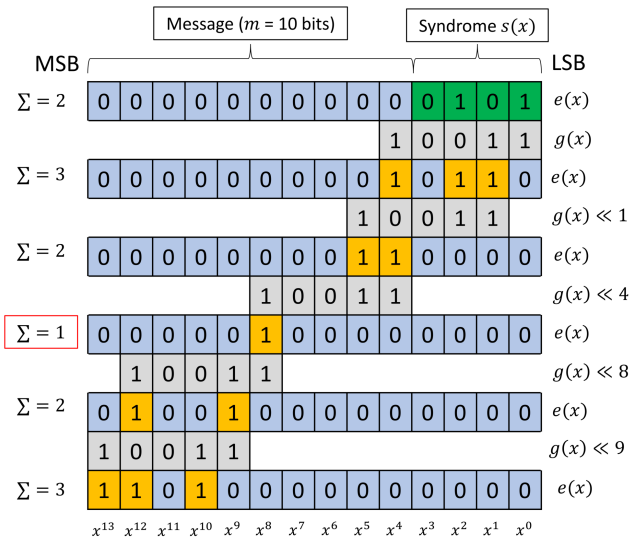


Fig. 1: Example illustrating the single error correction method as described in [22], using a CRC-4-ITU with generator polynomial $g(x) = x^4 + x + 1$ over 10 data bits, when the computed syndrome at the receiver is $s(x) = x^2 + 1$

receiver is $s(x) = x^2 + 1$. We can observe that the first step is to initialize the error vector **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 this value is equal to 2. As we are searching for error patterns containing only a single error, we do not consider such a pattern as a valid single error pattern. Thus, our goal is to cancel the LSB non-zero value by performing an XOR at that position. The resulting error vector contains 3 non-null positions and is still not a valid candidate for single error correction. As we proceed, we find a valid candidate at step 4, where there is a single non-null position in the error vector, at x^8. We ensure that there is no other candidate by scanning the whole packet length with this method. At the end of the process, we observe the list of candidates. 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 here, the bit at the corresponding position in the received packet must be flipped. Used alone, this method has some limitations since it 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 unfortunately be 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 a corrupted packet, due to desynchronization.

C. Validation of CRC-based error correction

The work in [28] illustrates that some bits in a compressed video stream will cause desynchronization if an error occurs. Exp-Golomb codes are a simple example of such so-called

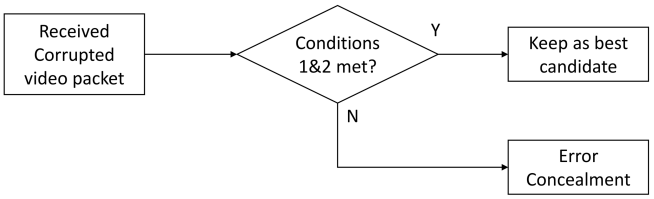


Fig. 2: System proposed in [28] allowing a validation of the reconstructed bitstream based on a two-step validation process

desynchronization bits. In such codes, a prefix (i.e., several bits set to 0, followed by a single bit set to 1) indicates the length of the codeword to be read. Clearly, if an error occurs in this prefix, the bit length of the next codeword will be wrong, and will propagate as the packet is decoded. This behavior is known as the desynchronization of the bitstream, and makes the bit sequence undecodable in most cases. Arithmetic coding used in modern video compression standards AVC [24], HEVC [25] and Versatile Video Coding (VVC) [29] is highly vulnerable to desynchronization.

As miscorrection can lead to desynchronized streams in the video decoder, the authors of [28] proposed the validation of the reconstruction through a two-step process for AVC encoded video sequences, as illustrated in Fig. 2. The two compliance validation operations are:

1) 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).
2) Check the decodability of the reconstructed video stream. Each candidate is thus tested and if it crashes the video decoder, the next candidate is tested.

After the entire set of candidates is tested, if none 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 is completed even where errors still exist in the decoded video packet, but it does not guarantee that the reconstructed packet is the transmitted one.

D. CRC-based multiple error correction

Multiple error correction can be performed by simply extending LUT-based methods, as proposed in [19]. This 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. This approach would lead to a listing of all the error patterns containing N errors or less. However, when the value of N increases, the size of such tables rapidly becomes 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 this table is expressed as follows:

$$\text{TableSize} = \binom{m}{N} \times [\text{length}(\mathbf{s}) + (2 \times N)]. \quad (5)$$

Here, we consider that the error positions are stored as 2-byte numbers for each pattern. This approach would thus require 2.6 TB of storage when using the CRC-24 used in BLE and a packet length of 1500 bytes for 3-error correction.

A more practical extension of the CRC error correction method to handle multiple error patterns was proposed in [23], where the approach was designed to output an exhaustive list of CRC-compliant candidates containing a defined number of errors N or less. In this approach, bit positions are forced throughout the process. Because the single error correction method cancels the LSB non-null value at each step, it is clear that it cannot consider error patterns having 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 increasing order, the algorithm can handle such error cases by selecting the positions of bits that will remain non-null during the process, by passing through or XORing **g** at these positions, 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. This method can be illustrated as shown in Fig. 3, where the set of forced positions is denoted $\mathcal{F}$ and contains the index of the $(N-1)$ values to force to 1. At each step, the current position is checked and an XOR is performed when the position has to be forced but is currently set 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 each loop, the number of non-zero values remaining in the error vector is counted, and if this number is equal to or less than N, a candidate is appended to the list, with error positions identified at the non-null positions of error vector **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.

Thus, all the bit error patterns can be considered and the complexity remains lower than in a brute force scheme, which tests every possible error patterns and conduct up 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, as compared to the single error approach, by handling more error cases, and does not require a lot of memory. However, by increasing the number of errors to consider, the average candidate list size increases as well. In [23], the Single Candidate Ratio (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 (6):

$$\mathrm{SCR}(m,N) = \frac{\mathrm{SinglePatterns}(m,N)}{\mathrm{TotalPatterns}(m,N)} \tag{6}$$

TABLE I: 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	2^8-1	$2^{15}-1$	$2^{23}-1$	$2^{32}-1$

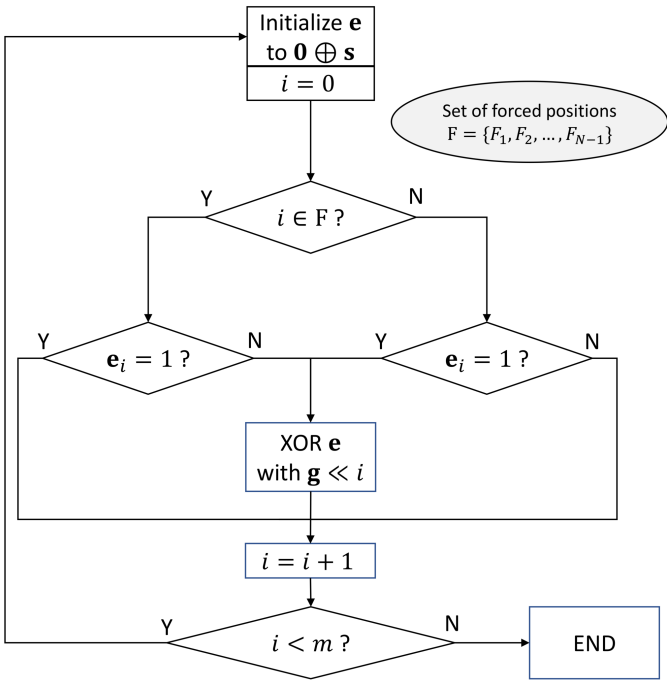


Fig. 3: Illustration of management of each bit position for multiple bit error correction using the CRC syndrome. The error vector **e** is initialized to zero and its LSB are set as the syndrome **s**. For each position, the current value is checked and the procedure depends on the elements in the set of forced position $\mathcal{F}$. Current position is flipped if it has to be forced and is currently 0 or has to be canceled and is currently 1

where $\mathrm{SinglePatterns}(m,N)$ 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 $\mathrm{TotalPatterns}(m,N)$ is the total number of individual N-error patterns for a message length of m bits.

When considering single error correction, it has been shown that the SCR remained at 100% for packets with lengths 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 the output of the single error method. Some examples of cycle lengths for common generator polynomials are illustrated in Table I.

The authors observed that for commonly used 3- or 4-byte generator polynomials, with the CRC-24 used in Bluetooth Low Energy (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, considering that a high number of errors may have occurred in the packet. The choice of the maximum number of errors (i.e., parameter N) in the error patterns of the candidate list should thus be determined according to the expected channel conditions.

Nevertheless, these methods still suffer from certain serious problems. Even if we set the number of errors to be considered to a small value (e.g., equal to or less than 3), there will be

a huge number of error cases that the previously proposed methods would not be able to handle. Whenever several candidates are present in the list, these methods will fail in performing error correction since they cannot determine the correct (originally transmitted) candidate. Validation steps are proposed in the literature to help choose between candidates for checksum-generated error pattern lists [18].

By definition, all the candidates in the list in the proposed approach pass the CRC check, which means that we can no longer use it 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 list and removing bad candidates, in order to increase the error correction capability as compared to previous methods, while also ensuring the validity of the candidate.

III. Proposed method

State-of-the-art multiple bit error correction methods exhibit respectable error correction capabilities, but suffer from several issues. First, increasing the number of errors considered N greatly increases the computational complexity of the method, which can lead to very long processing times. Having solutions to reduce this complexity would allow the method to handle multiple errors in practical scenarios. Moreover, considering a high number of errors yields a higher number of candidates in the list. Since the CRC is used to produce the candidate list, it can obviously not be used as well to identify bad candidates in the list as these are all CRC compliant (i.e., all the error patterns in the candidate list correspond to the received computed syndrome). In the proposed approach, we address this issue with additional validation steps applied to the candidates.

A. List handling: validation of candidates

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

However, a cross-layer approach using the UDP/TCP checksum [30] can be used to drastically reduce the number of candidates in the list. By computing the checksum over all the candidate error patterns resulting from the CRC error correction method, we can identify the candidates that pass both the CRC and checksum tests in order to filter the candidate list, as illustrated in Fig. 4. Upon 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 candidate list. Candidates that are both CRC- and checksum-compliant are kept in the list. If a candidate fails the checksum validation, it is removed from the list as there are still errors in the packet. At the end of the process, if there is just one candidate remaining in the list, it is considered as the winning candidate, and error correction is performed.

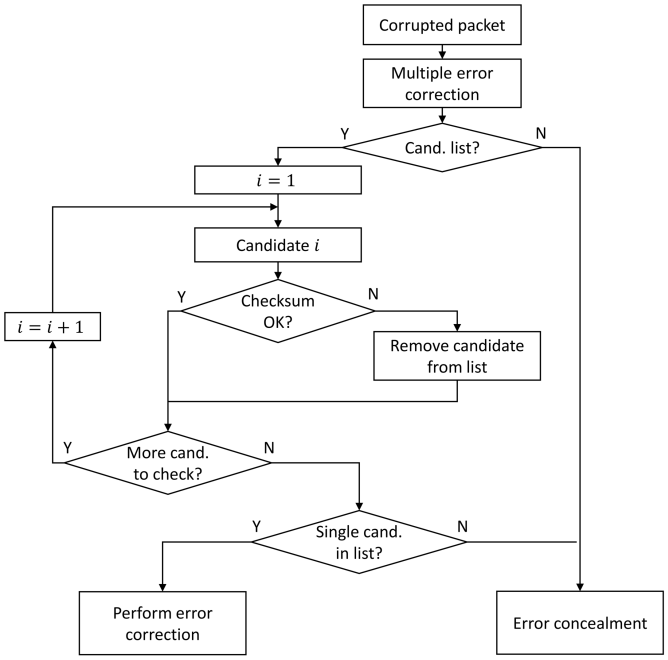


Fig. 4: Flowchart illustrating the correction process using both CRC and checksum to reduce the number of candidates in the output list

Computing the checksum does not require any additional resource and should not produce a large candidate list when combined with the proposed method, contrary to what is obtained in [18]. To illustrate the performance of the proposed cross-checking method, we take the example of a CRC-24 protecting a 250-byte payload. In this case, when searching for 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 selection process on multiple- and single-error patterns to ensure the validity of the reconstruction.

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 cross-check, we observed that the resulting number of candidates was significantly reduced, as shown in Table II. Here, we compare the average candidate list size before and after checksum validation in a Bluetooth Low Energy environment, considering packets of at most 250 bytes and $N = 3$. In this case, the average number of valid candidates goes from approximately 100 to less than 2 after the validation step. We observed a maximum of 5 candidates in the final candidate list. For simplicity, in this paper, we only consider the checksum to demonstrate the benefits of adding validation steps in order to reduce the number of candidates. However, we can further

validate the values of various application-dependent 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.

We recommend, as in [28], performing an additional step in the error correction process, irrespective of whether there is a single or multiple candidates in the output list. This is the video decoding validation step. 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 furthermore, that the erroneous bits are causing syntax errors and/or a desynchronization of the bit stream. In that case, the next candidate is tested, and the first candidate to pass the video decoding test is considered the winning candidate. Note that at this point, there is a probability that the packet still contains errors since errors cannot be detected in the video stream. However, because the errors will not cause any desynchronization, they can be overlooked as some studies have shown that such errors produce fewer visual artifacts than is the case when concealing the whole lost packet [31]. Even when the list contains only one candidate, a video validity check is still performed because there is the possibility that the packet could have been corrupted by many errors. The check thus ensures that the correction is valid and reduces miscorrections. Such cross-layer designed 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.

B. Optimizing the method

Increasing the number of errors considered also significantly increases the time required to perform error correction. It has been shown that the computational complexity of the arithmetic method grows exponentially with N, which is why it is crucial to seek out for methods and strategies to reduce the time needed to perform the proposed method. Knowing the number of errors in advance should help cut down on the iterations and reduce the computational complexity.

1) Exploiting syndrome parity: The parity of both the generator polynomial and the syndrome can provide valuable information on the number of errors in the video packet. Additional knowledge on the parity of the number of errors should help skip most of the iterations that would not have produced any candidate.

The parity of widely used generator polynomials such as CRC16-CCITT and CRC24-BLE is even, which allows deducing information on the number of errors that occurred in the packet. To illustrate this, we can express the operation for updating the error polynomial as:

$$\begin{aligned} e'(x) &= g(x) + e(x) \\ &= \sum_{i=0}^{n} g_i.x^i + \sum_{i=0}^{n-1} e_i.x^i = g_n.x^n + \sum_{i=0}^{n-1} (g_i.e_i).x^i \quad (7) \\ \Rightarrow\ & e'_n = g_n \text{ and } e'_i = (g_i.e_i), \forall\ (n-1) \geq i \geq 0 \end{aligned}$$

From this equation, it can be seen that the updated value of $e(x)$, denoted $e'(x)$, will be affected by $g(x)$. In $\mathrm{GF}(2)$, 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 shown in the addition truth table for $\mathrm{GF}(2)$ in Table III. From this definition, it is clear that k non-null coefficients in $g(x)$ will flip k values in the error polynomial. As such, 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 $\mathrm{NbErr}(p)$ and $\mathrm{Syn}(p)$ 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:

$$\begin{aligned} \mathcal{S} &\triangleq \{s \in \mathbb{N} \mid 0 \leq s \leq (2^n - 1)\} \\ S_O &\triangleq \{\mathrm{Syn}(p) \mid 0 \leq p \leq 2^{m+n} \text{ and } \mathrm{NbErr}(p) \text{ is odd}\} \\ S_E &\triangleq \{\mathrm{Syn}(p) \mid 0 \leq p \leq 2^{m+n} \text{ and } \mathrm{NbErr}(p) \text{ is even}\} \end{aligned} \quad (8)$$

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

$$\mathcal{S}_O \cap \mathcal{S}_E = \begin{cases} \emptyset, & \forall\ g(x) \text{ with even parity} \\ \mathcal{S}, & \forall\ g(x) \text{ with odd parity} \end{cases} \quad (9)$$

By itself, this knowledge does not allow determining the exact number of errors in the packet, but it helps reduce the computational complexity by avoiding unnecessary computation. When applied to triple error correction, the error correction method must 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 processing since triple error correction is computationally more intensive. We compared the processing time gains provided by considering the syndrome's parity applied to Bluetooth Low Energy channels [38] in Table IV. It can be seen that the gains increase as the channel conditions decrease. Indeed, high quality channels produce many error patterns containing a single error, where considering the parity has little effect on the total processing time. When the average number of double

TABLE II: Average number of candidates before and after checksum validation for different Bluetooth Low Energy channel qualities with $N = 3$

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

TABLE III: Addition and multiplication truth tables for finite field $\mathrm{GF}(2)$

+	0	1
0	0	1
1	1	0

.	0	1
0	0	0
1	0	1

TABLE IV: 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 → P	$1.18\,s \rightarrow 1.08\,s$	$1.15\,s \rightarrow 0.77\,s$	$1.15\,s \rightarrow 0.61\,s$

error patterns increases for channel Eb/No ratios of 8 dB and 9 dB, the average processing time significantly decreases, going down to half of the original processing time at Eb/No of 8 dB.

2) Reducing memory requirements: Another aspect that can be optimized is the memory required to store the error vector throughout the whole process. In [23], 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 (3) that the definition of the error polynomial is:

$$e(x) = s(x) + q(x).g(x) \tag{10}$$

which can be further expressed as:

$$\begin{aligned} e(x) &= \sum_{i=0}^{n-1} s_i.x^i + \sum_{j=0}^{m} q_j.x^j . \sum_{i=0}^{n} g_i.x^i \\ &= \sum_{i=0}^{n-1} s_i.x^i + \sum_{j=0}^{m}\sum_{i=0}^{n} (q_j.g_i).x^{(i+j)} \end{aligned} \tag{11}$$

where m and n represent the lengths of the payload and syndrome, respectively. It should be recalled that the proposed method for searching for 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 this definition, it is clear that for a given position j, (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 $j+n$. Outside 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 equal to or lower than j), apart from the forced positions (which are known at each step). In [23], 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, hence the need to force positions in order to be able to identify every error case.

Investigating the error range further, we can see that it can be leveraged to reduce the memory required in implementing 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 was 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. This optimization is illustrated in Fig. 5, where it can be seen 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, $\mathrm{M_{LUT}}$, is:

$$\mathrm{M_{LUT}} = \binom{m}{N} \times [\mathrm{length}(\mathbf{s}) + 2N] \tag{12}$$

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

$$\begin{aligned} \mathrm{M_{Arith}} &= 2(m + \mathrm{length}(\mathbf{s})) + 2(N-1) \\ \mathrm{M_{Opti}} &= 2(\mathrm{length}(\mathbf{s})) + 2(N-1) \end{aligned} \tag{13}$$

In (12) and (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 must store every error pattern syndrome of length n bits in the table along with the error pattern consisting of N integers. For its part, 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 size n bits ($\mathbf{e}'$ and $\mathbf{e}$). For a double error pattern search in a packet of length 1500 bytes, the storage needed for a lookup table approach is 432 MB, whereas only 6 bytes are needed for the proposed optimized approach. Moreover, when the number of errors increases, the memory storage needed for the table approach rapidly becomes intractable, with table sizes in Petabytes needed to handle 4 errors with CRC-16. The proposed approach also needs even less memory than the state-of-the-art arithmetic method, which requires 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.

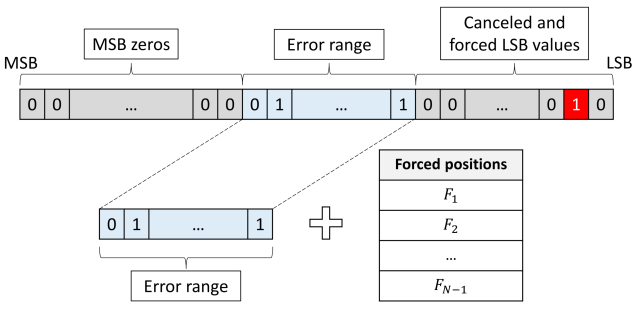


Fig. 5: Illustration of the memory reduction induced by considering only the error range of the error vector. The set of forced positions must be stored in order to identify error positions if a candidate is found

C. Proposed approach implementation

Algorithm 1 SingleErrorCorrection(**s**,**g**,n,m)

Inputs:
s: the syndrome vector
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

$E_1 \leftarrow \{\}$
Let **e** be a vector of length n
e $\leftarrow$ **s**
if sum(**e**) $= 1$ **then**
  Add NZ(**e**) to E_1
end if
for $j = 0$ to $m - 1$ **do**
  if $e_0 = 1$ **then**
    e $\leftarrow$ (**e** $\oplus$ **g**) $\gg 1$
    if sum(**e**) $= 1$ **then**
      Add NZ(**e**) $+ j$ to E_1
    end if
  else
    e $\leftarrow$ **e** $\gg 1$
  end if
end for
Return E_1

To illustrate the implementation differences between by the literature and the proposed approach, in this section, we describe the updated algorithms used to conduct multiple error correction. We encourage the reader to compare them with the original algorithms presented in [23]. The main changes are shown in blue font in the proposed algorithms. Algorithm 1 illustrates the single error handling process, which takes as input the syndrome vector **s**, the generator polynomial **g** and the lengths of both the syndrome n and the payload m, expressed in bits. In this updated algorithm, the main changes relate to 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: Sum(**e**) provides the number of bits set to 1 in vector **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 **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 **e**. If this value is 1, the LSB must be canceled.
Step 9: The cancellation process is performed through an XOR of the generator polynomial **g** with the error vector **e**. The result is then right-shifted by one position to remain at bits of

Algorithm 2 N-ErrorPatternGeneration(**s**,**g**,n,m,N,p_R)

Inputs:
s: the syndrome binary vector
g: the generator polynomial binary vector
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

Let **e** be a vector of length n
$E_N \leftarrow \{\}$ and **e** $\leftarrow$ **s**
if sum(**e**) $\leq N$ **then**
  Add NZ(**e**) to E_N
end if
$k \leftarrow N$
while $k \geq 1$ **do**
  if mod(**g**, 2) $= 1$ || (mod(**s**, 2) $=$ mod(k, 2)) **then**
    if $k = 1$ **then**
      Add SingleErrorCorrection(**s**,**g**,n,m) to E_N
    else
      Let $\mathcal{F} \leftarrow (0, \ldots, k-2)$
      while $\mathcal{F} \neq (m-(k-1), \ldots, m-1)$ **do**
        start $\leftarrow \max(F_1 - 1, 0)$
        nbForced $\leftarrow 0$
        for $j =$ start to $m - 1$ **do**
          if $j \in \mathcal{F}$ **then**
            nbForced $=$ nbForced $+ 1$
          end if
          if $(e_0 = 0 \,\&\, j \in \mathcal{F}) \,||\, (e_0 = 1 \,\&\, j \notin \mathcal{F})$ **then**
            e $\leftarrow$ (**e** $\oplus$ **g**) $\gg 1$
            if sum(**e**) $+$ nbForced $\leq N$ **then**
              Add (NZ(**e**) $+ j$) and F_i to E_N
            end if
          else
            e $\leftarrow$ **e** $\gg 1$
          end if
          if $j = F_1$ **then**
            e$'$ $\leftarrow$ **e**
          end if
        end for
        $\mathcal{F} \leftarrow$ UpdateForcedPositions($\mathcal{F}$,m)
        e $\leftarrow$ **e**$'$
      end while
    end if
    e $\leftarrow$ **s** and $k \leftarrow k - 1$
  end if
end while
for $i = 1$ to size(E_N) **do**
  $p_C \leftarrow p_T$ with flipped bit values at positions in $E_N(i)$
  if checksum(p_C) $\neq$ OK || decode(p_C) $\neq$ OK **then**
    Remove candidate $E_N(i)$ from List
  end if
end for
Return E_N

Algorithm 3 UpdateForcedPositions($\mathcal{F}$,m)

Inputs:

$\mathcal{F}$: sorted list $(F_1, \ldots, F_{k-1})$ 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 $\mathcal{F}' \leftarrow (F_1, \ldots, F_{k-1})$
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 $\mathcal{F}' \leftarrow (F_1, \ldots, F_{k-1})$
14: **end if**
15: **end for**
16: **end if**

length n.

Step 11: If there is only one non-null value in the error vector, its position is added to the candidate 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 2 represents the multiple error handling process for a given syndrome **s**, generator polynomial **g** and number of errors considered N, including the parity check, memory optimization and checksum validation. The changes are shown in blue font in the algorithm. Note that in the algorithms, we denote the logical operations AND and OR as & and ||, respectively. The input differs slightly from the original algorithm in [23] as it includes the received corrupted packet p_R, which we will use in the checksum validation step. The following are the main changes:

Steps 1 and 2: The binary error vector **e** is of length n and is initialized to the value of the computed syndrome.

Step 4: We check if the syndrome itself is a solution. If the number of non-null bits in **e** is equal to or less than the maximum number of errors considered, we add the corresponding error positions to the candidate list E_N. The function $\text{NZ}(\mathbf{e})$ returns the degrees of the non-null positions in **e**.

Step 8: In order to avoid unnecessary computation, at the beginning of the main loop, we check whether the current case can lead to new candidates. We demonstrated that the search is unnecessary if the parity of the generator polynomial **g** is even and if both the syndrome **s** and the current numbers of errors searched k have different parities.

Step 15: We introduce a new variable, nbForced, which corresponds to the number of forced positions already set at the current time.

Step 17: 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 20: To determine whether or not the LSB value of the error vector e_0 should be canceled, 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 an XOR of the generator polynomial with the error vector, and then right-shift the result by one position to keep a constant length of **e**. If these conditions are not met, **e** is simply right-shifted by one position, as illustrated in step 26.

Step 23: 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 equal to or less than 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: $F_i,\ \forall i \leq$ 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, only positions F_1 and F_2 will be the forced positions to be added to the error pattern.

Step 36: At each main loop iteration, the error vector is reinitialized to its original value **s**.

Step 39: While the state-of-the-art method stops the process at the end of the main loop, we propose adding a validation step to reduce the size of the candidate list. We test every candidate from the list.

Step 40: 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 41: We test the checksum value and the decodability of the candidate packet p_C. If one of these tests fails, the current candidate is removed from the candidate list at step 42. At the end of the process, the resulting list is returned.

IV. Simulation and results

To illustrate the gains provided by the proposed method versus the state-of-the-art ones in terms of error correction capability, we simulated the transmission of compressed video over unreliable channels. Two compression standard profiles were 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 were considered: the first one addresses video communications in a vehicular environment, while the second is dedicated to IoT applications. For both scenarios, we conducted simulations to compare the reconstructed video quality from the following methods for AVC video content:

- JM-FC: Standard AVC Joint Model frame copy reconstruction [32].
- STBMA: Spatio-temporal boundary matching algorithm error concealment method [12] similar to the approaches used for comparison in a recent study [37].

- CRC-ECA1: CRC-based single-bit error correction without any candidate validation [22], 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 III-A, 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 compared CRC-ECA1, CRC-ECCV and the following techniques:

- Deblock+MVS: deblocking filter and iterative motion vector search error concealment techniques, available in the FFmpeg decoder [33].

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 *Ice*, *Crew*, *City*, *Mobcal*, *Ducks* and *ParkRun* video sequences were used to conduct the simulations. We chose these sequences because they have 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.
- *City* sequence (4CIF 704×576) represents a city filmed from an helicopter, with horizontal travelling movements.
- *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.
- *Ducks* sequence (HD 1280×720) represents ducks taking off from a water pond, with no camera movement.
- *ParkRun* sequence (HD 1280×720) represents a person running. Horizontal travelling follows the runner's movements.

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

A. Wi-Fi 802.11p

The Wi-Fi 802.11p standard [34] 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 [35], the authors show that video transmission from a vehicle to another can be useful in overtaking situations. They propose transmitting the video from the windshield camera of a vehicle to the vehicle right behind it. As such, 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 safely. Video transmission can also ensure transport safety, for example in the form of in-vehicle surveillance in public transport, where video content can be shared live to authorities if a threat of vandalism is detected. V2I video transmission can also be used to monitor traffic conditions, by sending video from vehicles equipped with external cameras to a roadside unit (RSU), thus providing information on the traffic conditions.

V2V channels have been implemented in Matlab from real-world data collected by [42] for different scenarios, such as urban and rural scenarios, considering that the vehicles are either in line-of-sight (LOS) or non-line-of-sight (NLOS) situations. The 802.11p transmission and vehicular channel models used in our simulations are available in the Matlab WLAN toolbox [36]. We tested our method over several channel conditions, which are described in each example.
Fig. 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 the stated 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 verified that the video quality was an increasing function of the channel SNR. The gains are expressed with the sequence PSNR, as recent works in [37] conclude that achieving better quality assessment requires considering the whole video sequence due to errors propagating, they also demonstrate that in such cases the sequence PSNR offers better results than more recent metrics. By performing a CRC error correction on 3-error patterns with a checksum validation,

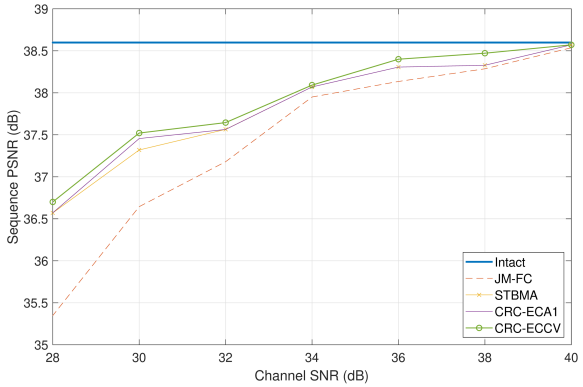


Fig. 6: Comparison of sequence PSNRs for AVC encoded *Ice* sequence (4CIF 704x576) at QP32 and different channel SNRs (vehicular Rural LOS scenario)

TABLE V: Average number of errors per corrupted packet for different channel SNR values in Rural LOS 802.11p environment

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

the proposed approach is able to correct an average of 10% of the corrupted packets. This leads to PSNR gains ranging from +0.2 dB at high channel SNRs (38 dB) to +0.6 dB for lower channel SNRs (24 dB) as compared to the STBMA error concealment method. The simulations conducted also show that increasing the number of errors to consider would not be worth the accompanying increased computational complexity, as most of the packets contain a very large number of errors. As an example, in the simulation run for a high channel SNR of 34 dB, we observed that almost 70% of the corrupted packets contained more than 10 erroneous bits.

Table V shows the experimental error distribution we obtained over the Rural LOS 802.11p channel, for different channel qualities. It can be seen 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 total corrupted packets. When the channel conditions are degraded, the ratio of corrupted packets the method can handle tends to decrease. We observe a maximum of 18.1% of corrupted packets containing less than 3 errors at a channel SNR of 32 dB and of 5.1% at a channel SNR of 20 dB. Correcting 10 to 20% of corrupted packets can help increase the PSNR by up to 0.6 dB in this test case, but the visual impact is not that significant when the packet contains many errors. Of course, the average number of errors in the packet decreases when the channel quality increases, but still remains very high in most cases. The proposed method is still able to correct most of the error patterns up to 3 errors, but because most of the corrupted packets are highly so, the gains are greatly affected by the high ratio of erroneous packets associated with the LOS rural scenario.

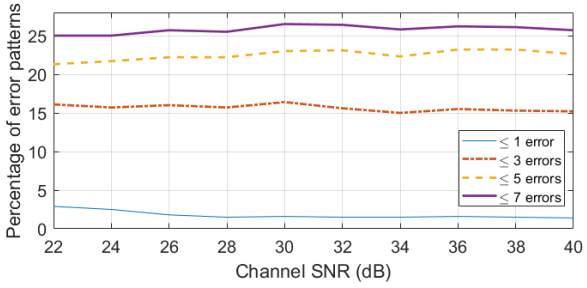


Fig. 7: Percentage of error patterns containing less than a determined number of error in Wifi 802.11p Urban NLOS environment for different channel SNR values

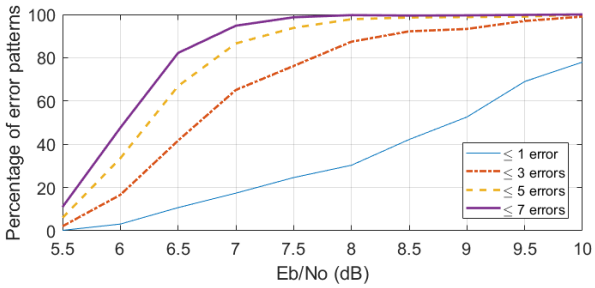


Fig. 8: Percentage of error patterns containing less than a determined number of error in Bluetooth Low Energy environment for different Eb/No values

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 Fig. 7, we show the percentage of error patterns containing less than a target number of errors in 802.11p environment for the Urban NLOS scenario, for channel SNRs varying from 22 dB to 40 dB. We observe that as the targeted number of errors increases, the percentage gains increase less and less, going from 1.5% for 1 error to 15% for 3 errors, then increasing to 22% for 5 errors and 25% for 7 errors. Even when considering up to 7 errors, 75% of the corrupted packets cannot be handled. With such error distributions, the visual gains of the proposed method applied to H.264 encoded sequences cannot be significant, as for each corrected packet there is still 2 erroneous packets that will affect the visual quality. For example, we performed transmission simulations of H.264 encoded video sequences over an 802.11p channel. The different methods' PSNRs for the *Ice* 4CIF sequence at QP=32, having a PSNR of 38.75 dB when intact, are 29.56 dB for the frame copy concealed video and 30.05 dB for the CRC error correction searching for 5 errors or fewer. The CRC error correction thus provides only a small improvement over frame copy, yielding a PSNR loss of 8.7 dB compared to the intact compressed sequence.

The error distribution of 802.11p channels is thus not the most adapted to our method. In fact, our error correction is based on the assumption that received corrupted packets are mildly corrupted. Thus, more significant gains can be achieved for applications that show a low number of errors per corrupted packet. In Fig. 8, we show the percentage of error patterns containing fewer than a target number of errors in BLE environment, for channel Eb/No varying from 5.5 dB to 10 dB. We can observe that such channel is more suitable for the proposed method, as the error distributions show that the vast majority of corrupted packets contain a very low number of errors, especially when the channel quality is high. For example, at Eb/No of 9 dB, 52.6% of corrupted packets contain a single error, 93.3% contain 3 errors or fewer, 98.8% contain 5 errors or fewer and 99.2% contain 7 errors or fewer. In the next section, we focus on the BLE application, which provides better results and a higher quality

of reconstructed video sequences. Since the application of the proposed method on 802.11p environments does not provide significant improvements, we will not present any further results for this case.

B. Bluetooth Low Energy

The Bluetooth Low Energy (BLE) [38] standard is widely used in IoT [39] applications to transmit sensor 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 detected [40].

To conduct transmission simulations over BLE channels, we use the Bluetooth Low Energy simulation with radio frequency impairments proposed in the Matlab Communication toolbox [41]. We simulate different channel conditions by varying the Eb/No parameter. The Eb/No corresponds to the SNR per bit of the communication. In examples, the Eb/No typically ranges from 7 dB to 10.5 dB, corresponding to Bit Error Rates (BER) of 10^{-4} to 10^{-7}. The BLE standard imposes a maximum packet length of 250 bytes. In what follows, videos were encoded with this packet size limit in order to ensure that transmission was energy efficient and to maintain a constant and low header overhead. This encoding management will have an impact on error distribution within the video sequence, since an intra-coded frame requires many more packets to be delivered than an inter-predicted frame when the payload of each packet is constant, since it is less efficiently compressed in the former case.

Unlike Wi-Fi 802.11p, the error distribution over BLE channels is favorable to the CRC-based error correction design, as illustrated in Fig. 9. The heatmap shows the value of N needed to handle 75% of the error cases. The two axes of the map are the Eb/No ratio and the packet length considered, in bytes. In other words, 75% of the error cases for the given packet length and channel quality contain the number of errors in the block or less. For example, for the highest quality channels considered in this example, it means that 75% of the corrupted packets contain 3 or fewer errors. We can also observe here that the number of errors to consider increases rapidly as the channel quality decreases. Since the proposed method yields better results in high quality channels, it is able to maintain a near-optimal video quality in them, subject to mild corruption; however, the gains it provides are not as significant when the channel conditions decrease drastically. In Fig. 9, we can consider that the channel and packet length conditions in which the method is able to operate efficiently correspond to values of N equal to or less than 3, represented in dark blue in the heatmap, i.e., Eb/No of 8dB or higher (up to the maximum packet length in the BLE standard, 250 bytes).

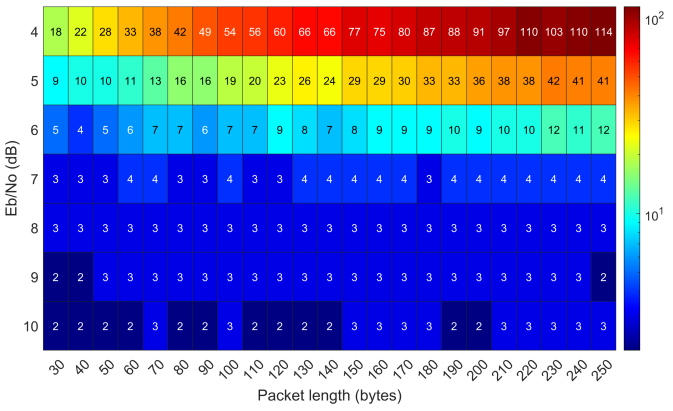


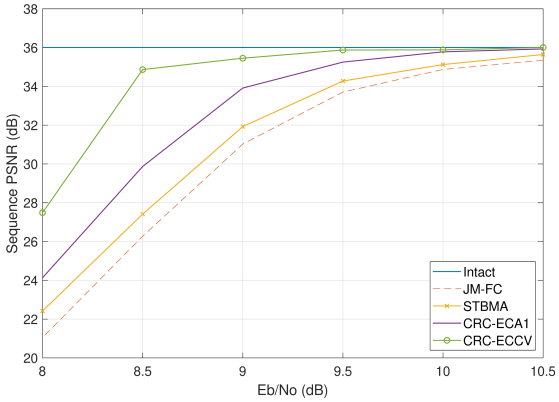
Fig. 9: Heatmap representing the value of the number of errors to consider, N, to handle 75% of the error cases for a given channel quality and packet length over a BLE channel. If N is set to the value in the corresponding box for such parameters, then it can successfully correct 75% of the corrupted packets

Table VI 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 increase the correction rate from 76.5% to 94.8%. In this case, the video sequence can be reconstructed with near-optimal visual quality. The gains for more severe channel conditions are even greater, since for an Eb/No 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 its content is basically of poor visual quality in such channel conditions. It is interesting to note that for all channels considered, the percentage of packets with 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 transmissions, and can achieve a significant correction rate versus state-of-the-art methods.

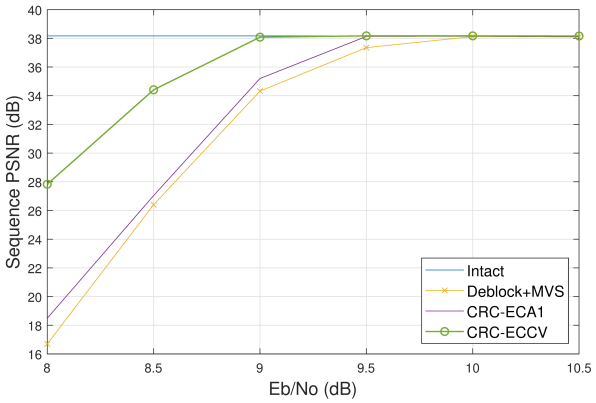
In Fig. 10, we present the PSNR of two reconstructed video sequences after transmission using the BLE protocol as a function of the Eb/No parameter value. In these simulations, we compare, in Fig. 10a, the JM-FC and STBMA error concealment methods to CRC-ECA1 and CRC-ECCV on the AVC encoded 4CIF *Ice* sequence, for a QP of 37. Fig. 10b 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 whereas in the other methods, the quality starts to rapidly decrease. It can also be seen that when the channel conditions are too severe, the performance of the proposed approach also decreases.

TABLE VI: Average number of errors 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 %



(a) AVC *Ice* Sequence at QP 37



(b) HEVC *Crew* sequence at QP 27

Fig. 10: 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.

In order to illustrate the performance of the proposed method, in Fig. 11, we present the evolution of the PSNR through time for a sequence transmitted through a Bluetooth Low Energy channel with Eb/No=9d dB. The tested sequence here is *Ice*, a 4CIF sequence of length 240 frames, encoded at QP32. We compare the proposed CRC-ECCV approach to JM-FC, represented by a red dashed line, and STBMA, presented in plain yellow. The proposed approach is shown by the green curve, and we present the error-free version in blue. In this example, 47 packets have been corrupted throughout the transmission. It can be seen that JM-FC and STBMA suffer greatly from these corrupted packets as their PSNR is significantly lower through the sequence. It can also be seen that each Intra frame, which are located every 30 frames, helps all methods 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, it can be seen that even if the method is able to handle most of the corrupted packets, the PSNR greatly decreases at frames 97 and 159. It was verified that these packets were hit by more than 3 errors, and as a result, the proposed approach was not able to perform error correction on them. Nevertheless, when considering the whole sequence, it can be seen that the global PSNR variability of the proposed approach is significantly lower than with other approaches. Indeed, this is a crucial factor to consider when it comes to the quality perceived by end users. It has been shown that human viewers are more affected by quality drops than by average quality decrease [43]. Fig. 11 shows that the error concealment method yields great variability, which would greatly affect the viewer's quality of experience.

A visual example of the gains provided by the proposed method versus state-of-the-art methods is shown in Fig. 12, considering the worst BLE channel (Eb/No=8 dB). Each of the visual examples shows the reconstructed video for the different methods compared. It can be seen that the video frame after

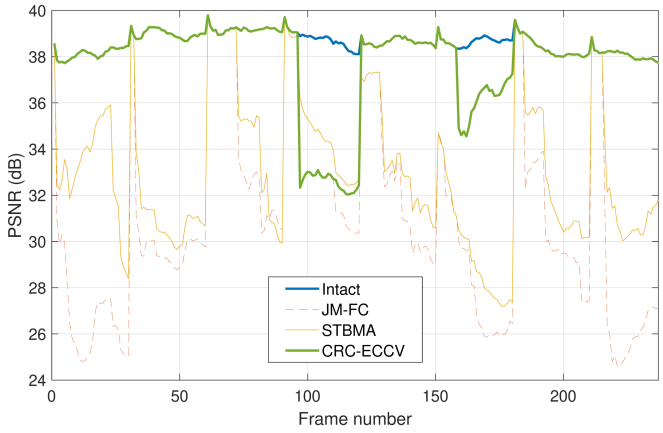
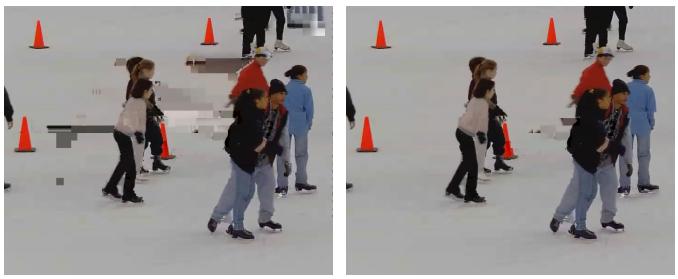


Fig. 11: PSNR evolution through time for AVC sequence *Ice* (4CIF) at QP32. The channel used has a Eb/No of 9 dB

(a) JM-FC (b) STBMA



(c) CRC-ECA1 (d) CRC-ECCV

Fig. 12: Visual comparison of the literature methods to the proposed approach on AVC encoded *Ice* sequence (4CIF) at QP32 for an Eb/No value of 8 dB

TABLE VII: Average PSNR comparison for different sequences and QPs over different BLE channel conditions for AVC encoded sequences. The following are the tested decoding methods: 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
City	22	40.90	35.36	37.72	39.93	40.57	40.90	29.52	33.18	36.09	39.28	40.90	17.24	19.80	21.56	33.35
	27	36.64	34.17	35.43	36.57	36.63	36.64	29.27	32.19	33.74	35.79	36.64	19.62	22.41	23.96	30.02
	32	33.01	32.24	32.72	32.90	33.00	33.01	29.14	31.11	31.92	32.75	33.01	20.82	23.06	24.56	29.85
	37	29.93	29.16	29.42	29.91	29.92	29.93	27.49	28.07	29.07	29.56	29.93	20.79	22.34	23.77	27.78
Mobcal	22	40.76	37.47	38.87	39.71	40.11	40.76	34.28	35.33	38.33	39.97	40.76	26.74	28.89	30.85	35.22
	27	37.90	33.85	35.20	37.57	37.70	37.90	30.43	32.73	34.68	36.17	37.90	23.46	24.95	27.18	33.02
	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	21.59	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
Ducks	22	40.42	36.56	36.79	39.78	40.35	40.42	33.00	33.58	35.87	37.85	40.42	25.07	26.79	31.02	36.50
	27	36.58	34.14	34.46	35.97	36.52	36.58	31.07	31.43	33.52	35.97	36.58	24.21	25.13	30.67	33.34
	32	33.07	31.90	31.94	32.86	33.06	33.07	30.21	30.47	31.64	33.02	33.07	22.28	23.02	25.32	28.73
	37	29.40	29.13	29.16	29.38	29.39	29.40	27.38	27.47	28.32	29.33	29.40	21.73	22.17	23.59	26.88
ParkRun	22	40.40	33.30	35.69	39.49	40.38	40.40	31.11	33.88	36.34	38.86	40.40	23.91	25.42	30.83	34.98
	27	35.33	31.35	32.08	34.96	35.26	35.33	28.39	29.61	32.30	34.24	35.33	23.11	24.37	27.22	30.31
	32	31.02	28.67	29.98	30.77	31.01	31.02	24.57	27.81	29.36	30.72	31.02	19.38	21.79	24.12	28.11
	37	27.42	26.51	27.05	27.25	27.41	27.42	23.98	26.01	26.71	27.31	27.42	17.78	21.28	21.97	25.49
$\Delta_{\mathbf{PSNR}}$	22	-	0	1.60	4.16	4.89	-	0	2.13	4.91	7.79	-	0	1.92	4.75	11.54
	27	-	0	0.89	2.77	2.95	-	0	1.75	3.90	6.19	-	0	1.59	4.29	9.70
	32	-	0	0.55	1.16	1.37	-	0	1.53	2.92	4.42	-	0	1.52	3.33	8.00
	37	-	0	0.26	0.58	0.77	-	0	0.76	1.80	3.02	-	0	1.39	2.82	6.62

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 of the corrupted packets in this frame, resulting in a visually better video quality. However, we can also note that not all the corrupted packets were corrected during the process, and a packet containing more than 3 errors was concealed near the center of the frame. As we encoded the video sequences under the packet length constraints of BLE, we could see that losing a packet has only a mild impact on visual quality because it carries less visual information.

In Table VII, we compare the reconstructed sequence PSNRs of several error concealment and error correction methods in a BLE environment, applied to several H264 encoded sequences. In this table, $\Delta_{\mathbf{PSNR}}$ corresponds to the average PSNR difference between each concealed or corrected sequence compared to the JM-FC concealed video, for each QP and each Eb/No value. The results show that as the channel quality decreases, the gains provided by the proposed approach increases, as compared to the literature. For example, for a high Eb/No of 10 dB, the proposed approach increases the sequence PSNR by 1.6 dB as compared to STBMA and by 0.3 dB as compared to CRC single error correction. We can also observe that the gains increase significantly as the Eb/No ratio decreases, and that at 8 dB, the average gains of the proposed method are 7.3 dB over STBMA and 5.1 dB over CRC single error correction. Since these gains see greater jumps as the channel conditions become less favorable, we must note the PSNR loss as compared to the intact sequence. While gains of up to 5 dB are observed as compared to other methods at Eb/No of 8 dB, on average, there is also a 4.9 dB loss as compared to the error-free video sequence. In the latter case, there are smaller gains for good channel conditions, and these help reconstruct a near-intact sequence, with an average PSNR difference of 0.10 dB for Eb/No=10 dB. As the channel conditions decrease, more packets containing a high number of errors are received. However, the method can still 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 the methods in the literature suffer from visual artifacts; further, it is still able to correct most packets as the channel quality decreases, but cannot ensure significant visual quality improvement in severe channel conditions.

V. Conclusion

In this paper, we propose 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. The validation of the reconstructed bit sequence is an essential additional step to ensure the validity of the corrected sequence and to reduce the risk of wrong corrections.

Simulations over different wireless environments, namely, Wi-Fi 802.11p and Bluetooth Low Energy, were conducted. We demonstrated that the gains achieved by the proposed method depend strongly 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 ranging from 24 to 36 dB in a Rural LOS scenario. The gains are significantly greater for Bluetooth Low Energy, where the proposed method is able to correct 70% of the corrupted packets for severe channel conditions, at Eb/No of 8 dB. The proposed approach offers, on average, PSNR gains of 1.6 dB to 7.3 dB versus state-of-the-art error concealment methods, at Eb/No 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 checksum validation processes and are decodable. Although exploiting fixed and predictable field values in protocols is expected to bring further gains, we would also like to investigate pixel-domain approaches, using deep learning, to identify the most probable intact video among the remaining reconstructed 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.

REFERENCES

[1] J. Sobolewski, "Cyclic Redundancy Check," in *Encyclopedia of Computer Science*, John Wiley and Sons Ltd, 2003.

[2] J. Luo, K. D. Bowers, A. Oprea, and L. Xu, "Efficient software implementations of large finite fields GF(2n) for secure storage applications". in ACM Transactions on Storage vol. 8, no. 1, 27 pages, Feb. 2012.

[3] H. Sun and W. Kwok, "Concealment of damaged block transform coded images using projections onto convex sets" IEEE Transactions on Image Processing, vol. 4, No. 4, pp. 470–477, Apr. 1995.

[4] J. Koloda, J. Østergaard, S. H. Jensen, V. Sánchez, and A. M. Peinado, "Sequential error concealment for video/images by sparse linear prediction" IEEE Transactions on Multimedia, vol. 15, no. 4, pp. 957–969, Jun. 2013.

[5] J. Liu, G. Zhai, X. Yang, and B. Yang and L. Chen "Spatial error concealment with an adaptive linear predictor" in IEEE Transactions on Circuits and Systems for Video Technology, Vol. 25, No. 3, pp. 353-366, Mar. 2015.

[6] A. Akbari, M. Trocan, and B. Granado, "Sparse recovery-based error concealment" IEEE Transactions on Multimedia, vol. 19, no. 6, pp. 1339–1350, Jun. 2017.

[7] J. Wu, X. Liu and K. Y. Yoo, "A Temporal Error Concealment Method for H.264/AVC Using Motion Vector Recovery IEEE Transactions on Consumer Electronics, Vol 54, No 4, pp. 1880-1885, Nov. 2008.

[8] W. M. Lam, A. R. Reibman and B. Liu "Recovery of lost or erroneously received motion vectors" in Proceedings of the IEEE International Conference on Acoustics, Speech ans Signal Processing ICASSP1993, vol. 5, pp. 417-420, Apr. 1993.

[9] T. Chen, X. Zhang and Y. Q. Shi, "Error Concealment Using Refined Boundary Matching Algorithm" in IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, pp.560-576, Jul. 2003.

[10] B. Chung, C. Yim, "Bi-Sequential Video Error Concealment Method Using Adaptive Homography-Based Registration" IEEE Transactions on Circuits and Systems for Video Technology, Vol. 30, No. 6, pp. 1535-1549, Jun. 2020.

[11] L. Atzori, F. G. B. De Natale, and C. Perra, "A spatio-temporal concealment technique using boundary matching algorithm and mesh-based warping (BMA-MBW)" IEEE Transactions on Multimedia, vol. 3, No. 3, pp. 326–338, Sep. 2001.

[12] Y. Chen, Y. Hu, O. C. Au, H. Li, and C. W. Chen "Video Error Concealment using Spatio-Temporal Boundary Matching and Partial Differential Equation" in IEEE Transactions on Multimedia, vol. 10, No. 1, pp. 2-15, Jan. 2008.

[13] A. Sankisa, A. Punjabi and A. K. Katsaggelos, "Video Error Concealment Using Deep Neural Networks," 2018 25th IEEE International Conference on Image Processing (ICIP), Athens, 2018, pp. 380-384.

[14] Dahun Kim, Sanghyun Woo, Joon-Young Lee and In So Kweon "Deep Video Inpainting" in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 5792-5801.

[15] Wang, C., Huang, H., Han, X. and Wang, J. "Video Inpainting by Jointly Learning Temporal Structure and Spatial Details", Proceedings of the AAAI Conference on Artificial Intelligence, 2019, vol. 33, no. 1, pp. 5232-5239.

[16] M. Park and D. J. Miller "Joint Source-Channel Decoding for Variable-Length Encoded Data by Exact and Approximative MAP Sequence Estimation" in IEEE Transactions on Communications, vol. 48, no. 1, pp. 1-6, Jan. 2000.

[17] F. Caron and S. Coulombe, "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, vol. 25, no. 7, pp. 1161–1174, 2015.

[18] F. Golaghazadeh, S. Coulombe, F.-X. Coudoux, P. Corlay, "Checksum Filtered List Decoding Applied to H.264 and H.265 Video Error Correction," in *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 8, pp. 1993-2006, Aug. 2018.

[19] S. Shukla, N. W. Bergmann, "Single Bit Error Correction Implementation in CRC-16 on FPGA," in *IEEE International Conference on Field-Programmable Technology*, Brisbane, Australia, pp. 319-322, 6-8 Dec. 2004.

[20] S. Babaie, A. K. Zadeh, S. H. Es-Hagi and N. j. Navimpour, "Double Bits Error Correction using CRC Method," in Fifth International Conference on Semantics, Knowledge and Grid, pp. 254-257, 12-14 Oct. 2009.

[21] A. S. Aiswarya and G. Anu, "Fixed Latency Serial Transceiver with Single Bit Error Correction on FPGA," *2017 International Conference on trends in Electronics and Informatics (ICEI)*, 11-12 May 2017.

[22] V. Boussard, F. Golaghazadeh, S. Coulombe, F.-X. Coudoux and P. Corlay, "Robust H.264 Video Decoding Using CRC-Based Single Error Correction And Non-Desynchronizing Bits Validation," *2020 IEEE International Conference on Image Processing (ICIP)*, Abu Dhabi, United Arab Emirates, 2020, pp. 1098-1102.

[23] V. Boussard, S. Coulombe, F.-X. Coudoux and P. Corlay, "Table-Free Multiple Bit-Error Correction Using the CRC Syndrome," in *IEEE Access*, vol. 8, pp. 102357-102372, 2020.

[24] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," IEEE Transactions on Circuits and systems for Video Technology, vol. 13, no. 7, pp. 560–576, Jul. 2003.

[25] G. J. Sullivan, J-R. Ohm, W-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1648–1667, Dec. 2012.

[26] K. Niu, K. Chen, "CRC-Aided Decoding of Polar Codes," in *IEEE Communications Letters*, vol. 16, no. 10, pp. 1668-1671, Oct. 2012.

[27] X. Liu, S. Wu, X. Xu, J. Jiao and Q. Zhang, "Improved Polar SCL Decoding by Exploiting the Error Correction Capability of CRC," in *IEEE Access*, vol. 7, pp. 7032–7040, Dec. 2018.

[28] F. Golaghazadeh, S. Coulombe, F.-X. Coudoux and P. Corlay, "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), Cairns, Australia, 2018, pp. 1-7.

[29] B. Bross, J. Chen, J. -R. Ohm, G. J. Sullivan and Y. -K. Wang, "Developments in International Video Coding Standardization After AVC, With an Overview of Versatile Video Coding (VVC)," in Proceedings of the IEEE, pp. 1-31, 2021.

[30] R. T. Braden, D. A. Borman, and C. Partridge, "Computing the internet checksum," IETF, RFC 1071, Sep. 1988. [Online]. Available: https://www.rfc-editor.org/rfc/rfc1071.txt.

[31] L. Trudeau, S. Coulombe and S. Pigeon, "Pixel domain referenceless visual degradation detection and error concealment for mobile video,"

2011 18th IEEE International Conference on Image Processing (ICIP), Brussels, 2011, pp. 2229-2232.
[32] "H.264/AVC JM reference software," [Online]. Available: http://iphome.hhi.de/suehring/tml/, version 18.5.
[33] FFmpeg codec documention. [Online] Available: https://ffmpeg.org/ffmpeg-codecs.html#Video-Decoders.
[34] IEEE 802.11: Part 11: "Wireless LAN medium access control (MAC) and physical layer (PHY) specifications", Dec. 2016.
[35] F. Rameau, H. Ha, K. Joo, J. Choi, K. Park and I. S. Kweon, "A Real-Time Augmented Reality System to See-Through Cars," in IEEE Transactions on Visualization and Computer Graphics, vol. 22, no. 11, pp. 2395-2404, Nov. 2016.
[36] 802.11p Packet Error Rate Simulation for a Vehicular Channel, [https://fr.mathworks.com/help/wlan/ug/802-11p-packet-error-rate-simulation-for-a-vehicular-channel.html], Accessed on Dec. 2020.
[37] M. Kazemi, M. Ghanbari and S. Shirmohammadi, "The Performance of Quality Metrics in Assessing Error Concealed Video Quality", in *IEEE Transactions on Image Processing*, vol. 29, pp. 5937-5952, 2020.
[38] Specification of the Bluetooth system. Core Version 4.1, Bluetooth SIG, 2013. [Online]. Available: http://www.bluetooth.com.
[39] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," in *IEEE Communication Surveys and Tutorials*, vol. 17, no. 4, Jun. 2015.
[40] S. N. Jyothi and K. V. Vardhan, "Design and implementation of real time security surveillance system using IoT," in 2016 International Conference on Communication and Electronics Systems (ICCES), Coimbatore, 2016, pp. 1-5.
[41] End-to-End Bluetooth Low Energy PHY Simulation with RF Impairments and Corrections [https://fr.mathworks.com/help/comm/ug/end-to-end-bluetooth-low-energy-phy-simulation-with-rf-impairments-and-corrections.html], Accessed on Dec. 2020.
[42] P. Alexander, D. Haley and A. Grant, "Cooperative Intelligent Transport Systems: 5.9-GHz Field Trials," in Proceedings of the IEEE, vol. 99, no. 7, pp. 1213-1235, July 2011.
[43] C. Yim and A. C. Bovik. "Evaluation of temporal variation of video quality in packet loss networks" in Signal Processing: Image Communication, Volume 26, Issue 1, pp. 24-38, 2011.