Checksum-Filtered List Decoding Applied to H.264 and H.265 Video Error Correction

Firouzeh Golaghazadeh, Stephane Coulombe, François-Xavier Coudoux, Patrick Corlay

Authors' accepted manuscript. Article published in *IEEE Transactions on Circuits and Systems for Video Technology* 28(8), Aug. 2018.
https://doi.org/10.1109/TCSVT.2017.2686647

Checksum-Filtered List Decoding Applied to H.264 and H.265 Video Error Correction

Firouzeh Golaghazadeh, Stéphane Coulombe, *Senior Member, IEEE*
François-Xavier Coudoux, *Member, IEEE* and Patrick Corlay

***Abstract*—The latest video coding standards, H.264 and H.265, are highly vulnerable in error-prone networks. Reconstructed packets may exhibit significant degradation in terms of PSNR and visual quality. This paper presents a novel list decoding approach exploiting the receiver side user datagram protocol (UDP) checksum. The proposed method identifies the possible locations of errors by analyzing the pattern of the calculated UDP checksum. This permits to considerably reduce the number of candidate bitstreams in comparison to conventional list decoding approaches. When a packet composed of N bits contains a single-bit error, instead of considering N candidate bitstreams, as is the case in conventional list decoding approaches, the proposed approach considers $N/32$ candidate bitstreams, leading to a reduction of 97% of the number of candidates. For a two-bit error, the reduction increases to 99.6%. The method's performance is evaluated using H.264 and H.265 test model software. Our simulation results reveal that, on average, the error was corrected perfectly 80 to 90% of the time (the original bitstream was recovered). In addition, the proposed approach provides, on average, a 2.79 dB gain over frame copy (FC) error concealment using the Joint Model (JM) and a 3.57 dB gain over our implementation of FC error concealment in the HEVC Test Model (HM).**

***Index Terms*—Video Transmission, Video Error Correction, H.264, High Efficiency Video Coding (HEVC), H.265, List Decoding, Checksum.**

I. Introduction

In recent years, digital video communication, especially in the form of high quality content delivery, has attracted considerable attention in a wide variety of application environments, such as mobile video streaming, video conferencing, telepresence, etc. Restrictions related to data storage, processing power, transmission cost, and communication speed make compression a mandatory step in the efficient processing of video streams. However, the high compression performance of current video coding standards (e.g., H.264 [1], H.265 also known as high efficiency video coding (HEVC) [2]) makes the compressed video streams extremely vulnerable to channel impairments. Even a single-bit error in variable-length code (VLC) may cause the decoder to lose its synchronization with the corresponding encoder due to an incorrect parsing of codewords. Even worse, because of the motion compensated prediction technique employed in compression, an error can propagate from one frame to consecutive ones, and lead to severe visual artifacts [3].

This work was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant. Emails: firouzeh.golaghazadeh.1@ens.etsmtl.ca, stephane.coulombe@etsmtl.ca, {francois-xavier.coudoux, patrick.corlay}@univ-valenciennes.fr

Various error control mechanisms have been proposed to combat visual quality degradation caused by transmission errors [4]. Among them, retransmission is one of the basic mechanisms for providing reliable communication. However, it is rarely used in real-time conversational or broadcasting/multicasting applications due to the added delay or lack of feedback channel involved [1]. Error resilience injects redundancies during source coding to make the streams more robust against transmission errors, and as a result, the decoder can better deal with loss of information. It should be noted thought that all error resilience methods reduce coding efficiency or sacrifice bit-rate, and are inefficient, especially when there is no transmission error [5], [6]. Compared to the other mentioned approaches, error concealment (EC), as a post-processing mechanism at the decoder side, will not require any additional bandwidth, and will not introduce retransmission delays. EC methods estimate lost areas by exploiting the inherent correlation between adjacent pixels (spatial EC) [7], [8], [9] or neighboring frames (temporal EC) [10], [11], [12]. Spatiotemporal EC combines both approaches [13], [14], [15]. A state-of-the-art embodiment of this technique is proposed in [13]. Lost motion vectors (MVs) are recovered by a modified classic boundary matching algorithm referred to as the spatiotemporal boundary matching algorithm (STBMA), which minimizes a distortion function by considering both spatial and temporal smoothness properties of the neighboring macroblocks (MBs). Clearly, EC performance is reduced when lost areas have less correlation (spatial or temporal) with the surrounding areas, especially when the lost areas are large.

Most EC approaches treat a corrupted packet the same as a lost one, with corrupted packets ignored and the missing information concealed. In practice, network congestion results in packet loss, while wireless signal attenuation, fading, etc., result in corrupted packets. However, corrupted and lost packets must be handled differently. Partially damaged packets may contain valuable information that can be used to enhance the visual quality of the reconstructed video [16], [17]. This is the case when the error occurs at the end of the packet or when the residual bit error rate (after channel decoding) is low. The novel user datagram protocol (UDP) such as UDP-Lite has been developed to deliver partially damaged packets to the application layer [18].

Corrupted packet has been exploited using two distinct approaches: joint source channel decoding (JSCD) [19], [20], [21], [22] and list decoding [23], [24], [25]. In [19], sequential decoding and soft information provided by the channel decoder is used for the prediction of residual coefficients

coded with context-adaptive variable-length coding (CAVLC) in the H.264 Extended profile. The additional information from data partitioning, such as packet length in bits and number of MBs in the slice, is applied as the constraint to define a valid packet. A maximum a posteriori (MAP)-based JSCD approach is employed for the decoding of the MVs and CAVLC of H.264 in [20] and [21], respectively. In [22], JSCD combined with soft estimation techniques was adopted for correcting context-adaptive binary arithmetic coding (CABAC) bitstreams of H.264 sequences under the assumption that each packet carries an entire picture.

Generally, in list decoding approaches, multiple candidates of the damaged bitstream are generated by flipping bits in the corrupted received packet. Then, the candidates are ranked from the most likely to the least likely bitstream, based on the soft information or reliability parameters of each bit. Each candidate is then checked for semantic and syntactic errors. Finally, the winning candidate is the first one that passes the decoder semantic verification. In [23], [24], 300 likeliest candidates are generated based on the soft value of flipped bits. The slice candidate with the smallest sum of its soft values is identified as the most probable one. Moreover, in [24], a virtual checking step is proposed to accelerate the semantic verification process of each candidate by considering the information of previous failed candidates. Farrugia [25] adopted a list decoding strategy to derive the M most probable paths with the smallest Hamming distance during the decoding of each symbol, irrespective of their length. The bitstream that meets three constraints related to bitstream length, number of MBs in slice, and successful syntactic/semantic verification is identified as the likeliest one. However, all these approaches suffer from the major drawback of having a fairly large solution space for candidate packets, leading to a decoding process with extremely high computational complexity. Indeed, a packet containing N bits has 2^N possible candidates when any number of errors is considered (or N candidates when a single-bit error is considered). This issue alone restricts the use of these approaches in real-time applications. Recently, a significantly less complex list decoding approach has been proposed in [26], [27], [28], where a soft/hard output maximum likelihood decoding (SO/HO-MLD) method is applied at the syntax element level instead of at the whole slice level. The solution space is therefore limited to a set of valid codewords for each specific syntax element. Although the method performs well overall, any mistake in the decoding of a syntax element will propagate and reduce the performance. An important issue with most error correction methods is the access (or lack of access) to the soft information. Propagating soft information, i.e. a fixed or floating point log-likelihood ratio (LLR) value for each bit of the packet, throughout the protocol stack (from the physical up to the application layer), is complex to implement and deploy in practice. In our previous work [29], we studied the checksum bit pattern for the case of a single-bit error, and showed that it could be exploited to eliminate 97% of the candidate bitstreams considered by list decoding approaches. The proposed method was tested on H.264 CAVLC coded sequences. In this work, we extend our previous work to the case of several bit errors and propose a novel list decoding approach which exploits the receiver side UDP checksum to alleviate the large solution space problem of list decoding approaches. We first show that the checksum of corrupted packets exhibits specific bit patterns. We specifically categorize these patterns for the case of one-bit and two-bit errors. We study the probability of various bit error events (BEEs) based on observed checksum pattern types (CPTs). Observing these specific patterns allows the identification of the potential error locations in the corrupted packets. This information is used to remove non-valid candidate bitstreams in a novel checksum-filtered list decoding (CFLD) system capable of handling numerous bit errors. The proposed approach eliminates 97% and 99.6% of the non-valid candidates for the case of one-bit and two-bit errors, respectively, compared to traditional list decoding approaches. This considerably reduces the computational complexity. The proposed method has been validated on H.264 CAVLC and HEVC sequences. The experimental results reveal the superiority of the proposed approach over others in terms of PSNR [13], [28]. Furthermore, the proposed method repaired nearly 80% of H.264 sequences and 90% of HEVC sequences perfectly. The proposed method using hard information (conventional bit values), unlike methods using soft-information, is easy to deploy in existing communication systems as it requires few changes to the protocol stack enabling erroneous packets to be delivered to the application layer (similar to UDP-Lite).

This paper is organized as follows. A detailed introduction to the UDP checksum and its calculation is presented in section II. In section III, we explain how the checksum can be applied in error correction. We first define different bit error events and calculate their corresponding checksum values. Then, we show the most probable bit error event, considering the observed checksum values. The proposed system for CFLD is described in section IV. Simulation results are provided in section V, and concluding remarks are drawn in section VI.

II. Internet Checksum Calculation and Properties

Internet Checksum is used by different standard protocols (Internet protocol (IP), transmission control protocol (TCP), UDP) for error detection [30]. Internet checksum, which is a fixed-length tag added to a message at the transmission side, enables the receiver to verify the integrity of the delivered message by recomputing the tag and comparing it with the tag that was sent. In this section, we present how the Internet checksum is computed, along with its mathematical properties, which will be exploited in this paper. Although the following principles are applicable to other checksums (e.g., TCP), we will focus specifically on the UDP checksum.

A. Internet Checksum Definition and Mathematical Properties

The Internet checksum is a 16-bit field within the protocol header, and is defined as one's complement of the one's complement sum of all the 16-bit words in the computation data [30]. More specifically, the Internet checksum is calculated at the transmission side as follows:

- Divide the data into chunks of 16-bit words. If necessary, pad the data with one byte zero at the end to make it a multiple of 16 bits.
- Perform one's complement sum over all the words. If an overflow occurs during any sum, the ones' complement sum operation involves an "end-around carry". The end-around carry scheme routes the carry-out signal of the most significant bit (MSB) position c_n to the least significant bit (LSB) position, where it is used as a carry-in signal c_0 [31].
- Flip all the bits of the final sum (one's complement).

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

Mathematically, the set of 16-bit values, represented here in hexadecimal for the sake of convenience[1], $V = \{0000, 0001, \ldots, FFFF\}$ and the one's complement sum operation (denoted as +), together form an Abelian group (commutative group) which has the closure, associative, commutative, identity and inverse element properties [32]. Interestingly, in this Abelian Group, there are two identity elements, 0000 and FFFF, which correspond to the same zero (+0 and -0) value. In several references, it is mentioned that the identity element is unique. This is rather a consequence than a rule and since these identity elements correspond to the same value, the Abelian group's properties are still met. It is worth mentioning another property that can be deduced from the Abelian group and which we will use in the following sections: $\overline{\sum_i a_i} = \sum_i \overline{a}_i \quad ; \quad \forall a_i \in V$.

B. UDP Checksum Definition and Calculation

The UDP checksum is a 16-bit field in the UDP header, and is the one's complement of the one's complement sum of the pseudo UDP header, the UDP header and the application data message [33]. Fig. 1 shows the UDP datagram and its 12-byte prefix as a pseudo UDP header. The pseudo UDP header contains the source and destination IP addresses, the protocol, and the UDP length. This information initially comes from the IP header. The UDP checksum is calculated over all the segments shown in Fig. 1. Like the Internet checksum, the checksum field of the UDP header should also be initialized to zero before the calculation, and then set to the calculated value prior to transmission. Since the UDP checksum is optional, a zero transmitted checksum value means that it was disabled. If the computed checksum is zero, it should be transmitted as all ones (FFFF) [33]. Note that the calculated checksum for a real packet can never be FFFF (i.e., the sum prior to the final ones' complement can never be zero) unless all the words in the packet are zeros [31]. Let us assume that the UDP packet has a length of N bits (including padding), which is made

[1]Four-digit numbers in this paper represent hexadecimal numbers

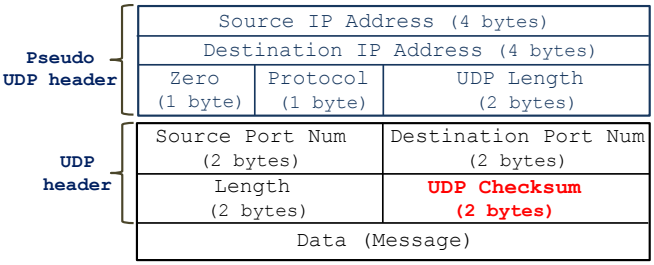

Fig. 1: UDP datagram and pseudo header.

up of m=N/16 16-bit words as $\{W_0, W_1, ..., W_{cs}, ..., W_{m-1}\}$ and W_{cs} is the checksum value in the checksum field. The i-th word and its inverse are respectively defined as:

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

where $\overline{w}_{i,c}$ represents the inverse of $w_{i,c}$, i.e., $\overline{w}_{i,c} = 1$ when $w_{i,c}$=0, and $\overline{w}_{i,c}$=0 otherwise. The transmission side's checksum (C_T) can be expressed as shown in Eq. (1):

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

The same process is performed at the receiver side to calculate the receiver side's checksum (C_R), except that instead of W_{cs}=0000, the value of the received checksum ($\widehat{W}_{cs}$=$\widehat{C_T}$) is used during the calculation of C_R, as shown in Eq. (2):

$$\begin{aligned} C_R &= \overline{\sum_{\substack{i=0 \\ i \neq cs}}^{m-1} \widehat{W}_i + \widehat{C_T}} = \overline{\sum_{\substack{i=0 \\ i \neq cs}}^{m-1} \widehat{W}_i + C_T} \\ &= \overline{\sum_{\substack{i=0 \\ i \neq cs}}^{m-1} \widehat{W}_i + \overline{\sum_{\substack{i=0 \\ i \neq cs}}^{m-1} W_i}} = \overline{\sum_{\substack{i=0 \\ i \neq cs}}^{m-1} (\widehat{W}_i + \overline{W}_i)} \end{aligned} \tag{2}$$

where the received versions of W and C_T are denoted as $\widehat{W}$ and $\widehat{C_T}$, respectively, and assuming that the 16-bit checksum word is intact ($\widehat{C_T} = C_T$). The receiver verifies the packet by re-calculating the checksum. It is obvious from Eq. (2) that if there is no error, which means $\widehat{W}_i = W_i$, the C_R value will be zero:

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

This is because the value of C_T, which is the inverse of the one's complement sum of all transmitted words, is included in the computation of C_R. Therefore, upon reception, when it is added to the one's complement sum of all words, the identity element FFFF is obtained.

C_R from Eq. (2) can be expanded to:

$$C_R = \overline{\sum_{\substack{i=0 \\ i \neq cs}}^{m-1} (\widehat{W}_i + \overline{W}_i)} = \overline{\sum_{\substack{i=0 \\ i \neq cs}}^{m-1} \sum_{c=0}^{15}(\widehat{w}_{i,c} + \overline{w}_{i,c}) \times 2^c} \tag{3}$$

An important property of the above one's complement sum with an end around carry expression is as follows (where "mod" means modulo):

$$(\widehat{w}_{i,c}+\overline{w}_{i,c})\times 2^c = \begin{cases} 2^c, & \text{if no error in bit } c \text{ of word } i \\ \overline{w}_{i,c}\times 2^{(c+1)mod(16)}, & \text{if error in bit } c \text{ of word } i \end{cases} \quad (4)$$

TABLE I: Values of $\widehat{w}_{i,c}+\overline{w}_{i,c}$ for various error scenarios. $0\rightarrow1$ represents a 0 flipped to 1 (and $1\rightarrow0$ the opposite).

$w_{i,c}$	$\widehat{w}_{i,c}$	Condition	$\widehat{w}_{i,c}+\overline{w}_{i,c}$	Carry
0	0	no error	1	0
0	1	error ($0\rightarrow1$)	0	1
1	0	error ($1\rightarrow0$)	0	0
1	1	no error	1	0

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

Fig. 2 contains an example of the checksum calculation at the transmission side and its validation procedure at the receiver side. In this example, the entire packet content is considered as three words (48-bit length) represented in hexadecimal. The checksum calculation steps were performed to establish the C_T. As can be seen, upon reception, the value of C_T is used in calculating the C_R. The zero value of the C_R in the first example validates the received packet. In the second example, a single bit 1 was flipped to 0 in column 24 of the packet, which corresponds to column 8 of the second word (24 modulo 16 is equal to 8), changing it from D1CB to D0CB (the column positions in the word are numbered from right to left). As can be seen, the C_R has a bit 1 in column 8 and 0 in the others. In other words, the C_R value provides important information related to the position of the error in the packet (i.e., that a bit in column 8 of a word was flipped). This simple example demonstrates our motivation to use the UDP checksum in an error correction approach.

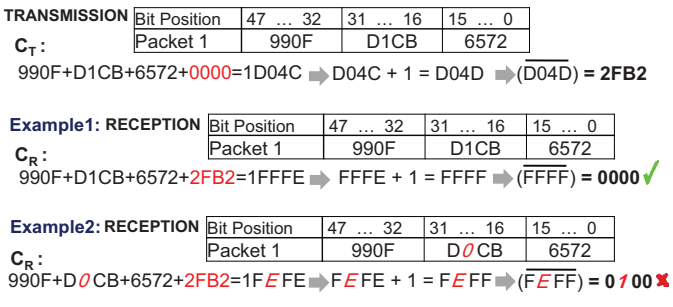

Fig. 2: UDP checksum calculation example.

III. Exploiting Checksum for Error Correction

As was shown in the example of Fig. 2 the C_R value can indicate the position of the flipped bits in modulo 16. The goal of this section is to study the C_R values in different error situations in order to show how they will change. This will help determine the potential error locations based on observed C_R values. In section III-A, we go through different BEEs and calculate their corresponding C_R values. The values are then grouped into different CPTs based on their similarity patterns. Finally, in section III-B, we examine the probability of each observed CPT to find the most probable BEE causing it.

A. Relationship Between C_R and Error Location

Different BEEs will create different bit patterns of C_R and now, we will study five different BEEs by considering one or two bits in error. This is reasonable since, in practice, the residual error after channel decoding should be low. Table II shows the definition of each BEE under study.

TABLE II: BEEs definitions for one and two bits in error.

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

In each BEE, two different bit modification cases are considered:

- $1_j\rightarrow0_j$, which means a bit 1 was flipped to 0 in column j of a word.
- $0_j\rightarrow1_j$, which means a bit 0 was flipped to 1 in column j of a word.

1) BEE=1: In this type of event, there is only one erroneous bit in the packet. If $1_j\rightarrow0_j$, where $w_{i,j}=1$ and $\widehat{w}_{i,j}=0$, as shown in Table I, $\widehat{w}_{i,j}+\overline{w}_{i,j}=0$ for column j, and for the other columns $c\neq j$, $\widehat{w}_{i,c}+\overline{w}_{i,c}=1$. Then $\overline{C_R}$ will have a bit 0 in column j and 1 for the others. By considering the final one's complement operation in Eq. (3), which flips all the bits, C_R will have a bit 1 in column j and 0 for the others. This is illustrated in the top part of Fig. 3.

For the case of $0_j\rightarrow1_j$ ($w_{i,j}=0$ and $\widehat{w}_{i,j}=1$), $\widehat{w}_{i,j}+\overline{w}_{i,j}=0$ for column j with an extra *carry* and $\widehat{w}_{i,c}+\overline{w}_{i,c}=1$ for the other columns ($c\neq j$). The extra carry generated in column j will affect the value of column $(j+1)mod(16)$ and change its value from 1 to 0 and also generate a carry which should be added to the next column $(j+2)mod(16)$. This carry propagation will continue and change all the bits 1 to 0, all the way, up to a column with a zero value. Since column j has a zero value, the carry propagation will finally stop there, and change its value from 0 to 1. That means there will be a 1 in column j of $\overline{C_R}$, while the other columns will have a 0. Therefore, C_R will have a 0 in column j and a 1 for all the other columns. This is illustrated in the bottom part of Fig. 3.

Fig. 3 summarizes all C_R values for these two cases. As can be seen, C_R values for these two cases are the inverse of each other. Depending on the error column, which can be one of the 16 columns in a word, C_R can have different patterns. All the 16 patterns of C_R for a $1_j\rightarrow0_j$ flip, as well as the 16-bit patterns of C_R for a $0_j\rightarrow1_j$ flip are grouped as CPT=1. CPT=1 is defined as the set of all C_R patterns that have fifteen bits 0 and a single bit 1, or vice versa. The error column in CPT=1 is indicated by the location of the bit, which is different from the others in the C_R value.

Let us revisit the second example of Fig. 2. The non-zero value of C_R there demonstrates that the received packet is

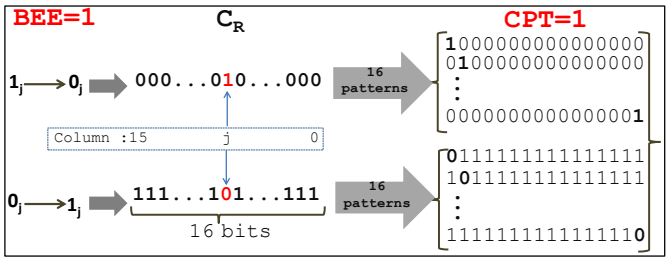

Fig. 3: BEE=1 and its corresponding 32 patterns of C_R forming CPT=1. Bold bits in CPT=1 indicate the error column.

corrupted. In addition, the location of bit 1 in the C_R pattern, column 8, signals that the potential error locations are the 8th bit of each word in the packet. So, in this example, the 8th, 24th and 40th bits of the packet are the three potential error locations. Moreover, the observed pattern of C_R (fifteen bit 0 and one bit 1 in CPT=1) indicates that a bit 1 was flipped to 0. Then, all the potential error locations having a bit value of 0 constitute the final set of potential error locations. In this case, only the 24th bit of the packet (8th bit of the second word, D0CB), has a value of 0, and is the final error location (in large packets, the list of candidates usually contains more elements).

In the case of two-bit error, four different BEEs are possible. Two erroneous bits can be in the same column or in different columns; as well, the two flipped bits can be the same (both 0 or both 1) or different (one 0 and the other, 1). All these BEEs and their corresponding generated C_Rs and CPTs are calculated and defined next:

2) BEE=2: In this type, two same bits in different columns are flipped. If $1_j \rightarrow 0_j$ and $1_k \rightarrow 0_k$, with $j \neq k$, $\widehat{w}_{i,c} + \overline{w}_{i,c} = 0$ for $c \in \{j,k\}$, and for the other columns, $\widehat{w}_{i,c} + \overline{w}_{i,c} = 1$. So, the corresponding C_R will have bits 1 in column j and k and bits 0 in other columns. For the case of $0_j \rightarrow 1_j$ and $0_k \rightarrow 1_k$, with $j \neq k$, $\widehat{w}_{i,c} + \overline{w}_{i,c} = 0$ for $c \in \{j,k\}$ plus two extra carries in columns k and j. As explained for BEE=1, an extra carry in column k will propagate and generate zeros all the way (from column $(k+1) mod(16)$ up to column $(j-1) mod(16)$), and will stop at column j by changing its value from 0 to 1. The extra carry in column j also propagates, and will stop in column k and change its value to 1. Finally, for $\overline{C_R}$, there should be two 1s in columns j and k and zeros for the others. In this case, C_R will have bits 0 in column j and k, and 1 in the other columns. Depending on which two columns of the words are hit by errors (2 out of 16 columns), the positions of the two bits 1 in the C_R pattern will change. We grouped all C_R patterns with fourteen bits 0 and two bits 1 (plus their inverse) as CPT=2, as shown in Fig. 4. The CPT=2 is divided into two sub-groups, CPT=2.1 and CPT=2.2, because CPT=2.1 is also observed in BEE=4.

3) BEE=3: In this type, two same bits in the same column are flipped. As shown in Fig. 5, this BEE generates the same pattern as BEE=1, which is CPT=1. As mentioned earlier, when there is no error $\widehat{w}_{i,c} + \overline{w}_{i,c} = 1$ for all 16 columns. When two $0_j \rightarrow 1_j$, then two extra 1s are obtained in column j, and this generates an additional carry. Then, column $(j+1) mod(16)$ will receive the extra carry, and its value will change to 0

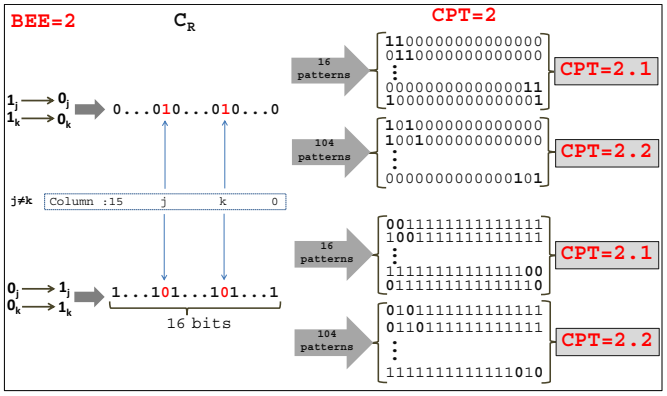

Fig. 4: BEE=2 and its corresponding 240 patterns of C_R forming CPT=2. The CPT=2 is divided further into two sub-groups as CPT=2.1 and CPT=2.2. All the patterns with two successive bit 1 (or 0) are grouded as CPT=2.1 and the rest are in CPT=2.2. Bold bits in CPT=2 indicate the error columns.

with an additional carry. In fact, such a carry will propagate and change all 1s, al1 the way up to a column with a 0 value. Since the value of column $(j+1) mod(16)$ is now 0, the carry propagation will stop there and change its value to 1. Therefore, for $\overline{C_R}$, all columns should be 0, except for column $(j+1) mod(16)$. In this case, C_R will have a 0 in column $(j+1) mod(16)$ and 1 in the other columns. In the other case, when two $1_j \rightarrow 0_j$, we are missing a carry which should have been generated by column j, and therefore, column $(j+1) mod(16)$ will contain a 0 instead of a 1. The C_R value will have a bit 1 in column $(j+1) mod(16)$ and 0 in the other columns. Like the other BEEs, the calculated C_R of the two cases are the inverse of each other (see Fig. 5). It is interesting to note that although this type leads to the same CPT as BEE=1, the location and type of errors in each case are quite different.

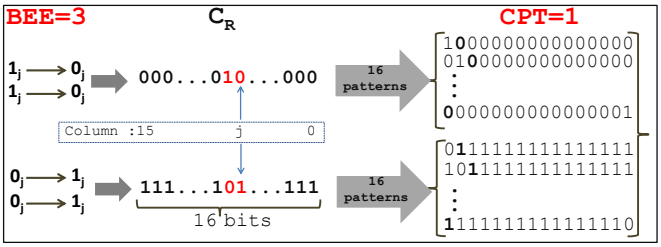

Fig. 5: BEE=3 and its corresponding 32 patterns of C_R forming CPT=1. Bold bits in CPT=1 indicate the error column.

4) BEE=4: In this type, two different bits in different columns are flipped. If $1_j \rightarrow 0_j$ and $0_k \rightarrow 1_k$, then $\widehat{w}_{i,j} + \overline{w}_{i,j} = 0$ and $\widehat{w}_{i,k} + \overline{w}_{i,k} = 0$, with a carry in column k, while for the other columns ($c \notin \{j,k\}$), $\widehat{w}_{i,c} + \overline{w}_{i,c} = 1$. The generated carry in column k will propagate and change all 1s to 0s, all the way up to the next 0 value, which is in column j, where it will stop by changing column j's value into 1. So, for $\overline{C_R}$, all the bits between columns k and j (moving circularly from right to left from k to j), excluding j, become 0, while the others remain 1. In this case, the C_R will have $|j-k| mod(16)$ bits 1 between column k and j (including k, but excluding j) and bits 0 in the others.

Depending on which two columns are hit, C_R can have different patterns. If the two columns are next to each other in modulo 16, i.e., $|j-k|mod(16)=1$, C_R has a single 1 and fifteen 0s (the same as CPT=1). But when $|j-k|mod(16)=2$, the generated pattern of C_R, which has two bits 1, is the same as CPT=2.1. In the other cases, when $3 \leqslant |j-k|mod(16) \leqslant 13$, where there are between three and thirteen bits 1 between column k and j, the C_Rs are grouped as CPT=3 (see Fig. 6).

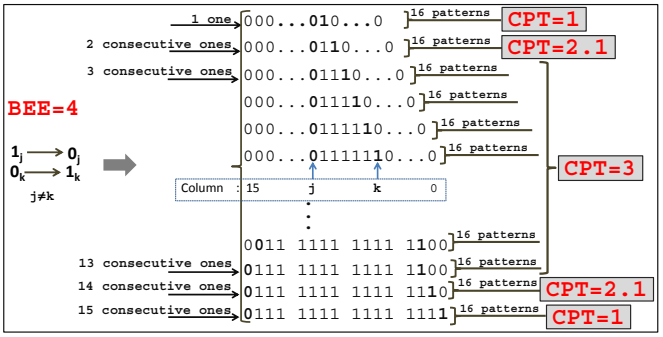

Fig. 6: BEE=4 and its corresponding 240 patterns of C_R forming CPT=1, CPT=2.1 and CPT=3. The column of bold bits 0 and 1 indicate the columns of bit 1→0 and 0→1, respectively.

5) BEE=5: In this type, two different bits in the same column are flipped. When $0_j \rightarrow 1_j$ and $1_j \rightarrow 0_j$, the first modification will add an extra 1 in column j, while the second one will remove a 1 in the same column. They will therefore cancel each other's effect and column j's value will not change. Therefore, $\widehat{w}_{i,j}+\overline{w}_{i,j}=1$ for all columns, and consequently, C_R will be zero, which is grouped as CPT=4 in Fig. 7. In this case, the observed pattern of C_R is exactly the same as the intact one. If information from the other layers shows that the received packet is corrupted, observing such a pattern indicates that BEE=5 has occurred. Only general information about the possible locations of the errors is available. We know that the two erroneous bits are in the same column, and that they are different bits.

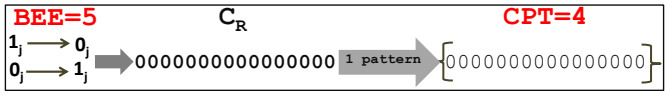

Fig. 7: BEE=5 and its only C_R pattern forming CPT=4.

The same process can be followed to describe the behavior of three bits in error. Some of the defined BEEs for three bits in error will map to existing CPTs, and additional CPTs will be observed. We ignore this case in the remainder of the paper because, as will be seen later in Eq. (6), the probability of having more than two bits in error is dramatically less than that of having a single-bit error in the applications of interest.

Table III summarizes the definition of each CPT for one and two bits in error. The computation of C_R for a received corrupted packet leads to one of the CPTs defined in Table III. Based on the CPT value, it is possible to determine the corresponding BEEs, as shown in Fig. 8. For each BEE, the C_R pattern will indicate the error columns and the type of modified bits (1→0 or 0→1). For instance, if the calculated C_R is "0000 0000 0010 0000", which has one bit 1 in column 5, it belongs to CPT=1, as defined in table III. This pattern can be generated by BEE=1, BEE=3 or BEE=4, as shown in Fig. 8. Based on each BEE, the C_R pattern will have different meanings. In the case of BEE=1, the C_R pattern indicates that there is a single-bit error, and it is $1_5 \rightarrow 0_5$. Then, all the bits 0 in column 5 are the potential error positions in this case. In the case of BEE=3, the pattern indicates that there are two bits in error, and both are $1_4 \rightarrow 0_4$, as presented in Fig. 5. In this case, the number of candidates is a 2-combination of the number of zeros in column 4. In the case of the BEE=4, the pattern indicates that there are two bits in error and $1_6 \rightarrow 0_6$; $0_5 \rightarrow 1_5$, as presented in Fig. 6. Therefore, it is possible to have more than one BEE for an observed CPT. The questions we must now answer are what the probability of occurrence of each of these possible BEEs is, and whether one is much more likely than the others to have occurred. We answer these questions in the following section.

TABLE III: Summary of CPT definitions.

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

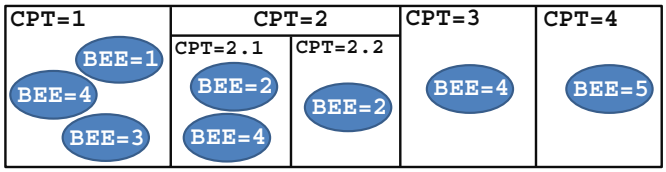

Fig. 8: Summary of observed CPTs and their corresponding BEEs for one and two bits in error.

B. Probability of BEEs Given Observed CPTs

As can be seen from Fig. 8, several BEEs can cause the same observed CPT. For instance, if the observed C_R value belongs to the patterns in CPT=1, then it could possibly be due to one of the three BEEs (BEE=1, 3 or 4). In this section, we show mathematically which one of these possible BEEs is more likely. The goal here is to find the probability associated to each BEE based on the observed CPT, which is defined as $\Pr(\text{BEE}=i|\text{CPT}=j)$. To compute this, we use the conditional probability and the law of total probability [34], as shown in Eq. (5).

$$\Pr(\text{BEE}=i|\text{CPT}=j)=\frac{\Pr(\text{BEE}=i,\text{CPT}=j)}{\Pr(\text{CPT}=j)}=\frac{1}{\Pr(\text{CPT}=j)} \times\Big\{\sum_{k=0}^{N}[\Pr(\text{BEE}=i,\text{CPT}=j|\text{nbErr}=k)\times\Pr(\text{nbErr}=k)\Big\} \tag{5}$$

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

$$\text{P}_k=\Pr(\text{nbErr}=k)=\rho^k\times(1-\rho)^{N-k} \tag{6}$$

Assuming that ρ is very small (e.g. $\rho \leq 10^{-5}$), then the probability of having more than two bits in error (P_k for $k>2$), even for large packet sizes, will be so small that the terms of

the summation for $k>2$ can be ignored. That is the reason we ignore considering more than two bits in error in the rest of the paper. Accordingly, Eq. (5) can be approximated with Eq. (7):

$$\Pr(\text{BEE}=i|\text{CPT}=j)\approx\frac{1}{\Pr(\text{CPT}=j)}\times\sum_{k=0}^{2}[\Pr(\text{BEE}=i,\text{CPT}=j|\text{nbErr}=k)\times\rho^k\times(1-\rho)^{N-k}] \quad (7)$$

By using the chain rule [34], the first probability in the previous equation can be expressed as:

$$\Pr(\text{BEE}=i,\text{CPT}=j|\text{nbErr}=k)=\Pr(\text{BEE}=i|\text{nbErr}=k)\times\Pr(\text{CPT}=j|\text{BEE}=i\cap\text{nbErr}=k) \quad (8)$$

The above two probabilities will be calculated as follows:

1) Probability of each BEE given the number of bits in error, Pr(BEE=i|nbErr=k): Assuming a packet with length of N bits, the packet is divided into words of sixteen bits, as shown in Fig. 9. For simplicity, the packet size is considered a multiple of 16 bits. Let nz_c and no_c represent the number of bits 0 and 1 in column c, respectively. In the following expressions, the probability value of $\Pr(\text{BEE}=i|\text{nbErr}=k)$ is calculated for the case of one and two bits in error ($k=1,2$). By definition of BEE=1:

$$\Pr(\text{BEE}=1|\text{nbErr}=1)=1 \text{ and } \Pr(\text{BEE}=1|\text{nbErr}=2)=0$$

By definition, all the BEEs from 2 to 5 are for two-bit error, and therefore, these BEEs cannot occur when the number of bits in error is one. However, they have values for a two-bit error. The probability value of each one can be calculated by the definition of each BEEs in Table II and considering the number of bits 0 and 1 in each column. The following equations reflect the number of possible combinations of taking two bits, same or different type, in the same or different columns. Assuming $\text{nz}_c=\text{no}_c$, which means the number of bits 0 and 1 in each column are the same, the expression can, however, be simplified as follows:

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

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

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

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

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

2) Probability of each CPT given the BEE and the number of bits in error, Pr(CPT=j|BEE=i∩nbErr=k): Here, the second probability of Eq. (8) will be examined. From the definition of BEE and CPT, it is clear that when there is a single-bit error, the following is obtained:

$$\Pr(\text{CPT}=j|\text{BEE}=1\cap\text{nbErr}=1)=\begin{cases}1 & j=1\\ 0 & j\in\{2.1,2.2,3,4\}\end{cases}$$

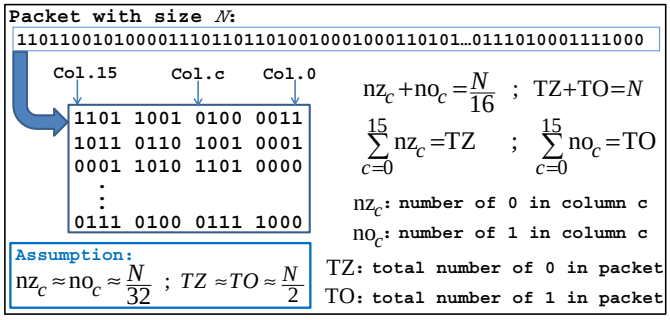

Fig. 9: Example of packet division into 16 bits.

and for the case of two bits in error, the following are obtained:

$$\Pr(\text{CPT}=j|\text{BEE}=2\cap\text{nbErr}=2)=\begin{cases}\frac{16}{120} & j=2.1\\ \frac{104}{120} & j=2.2\\ 0 & j\in\{1,3,4\}\end{cases}$$

$$\Pr(\text{CPT}=j|\text{BEE}=3\cap\text{nbErr}=2)=\begin{cases}1 & j=1\\ 0 & j\in\{2.1,2.2,3,4\}\end{cases}$$

$$\Pr(\text{CPT}=j|\text{BEE}=5\cap\text{nbErr}=2)=\begin{cases}1 & j=4\\ 0 & j\in\{1,2.1,2.2,3\}\end{cases}$$

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

BEE=4 comprises 240 different patterns, as shown in Fig. 6, 32 of which belong to CPT=1. Hence, the probability of having CPT=1 given BEE=4 is $32/240$. Similarly, the probability values for the other cases can be computed. By substituting the probability values of the two previous section III-B1 and section III-B2 into Eq. (8), we obtain the desired $\Pr(\text{BEE}=i,\text{CPT}=j|\text{nbErr}=k)$, as shown in Table IV. Note that P_1 and P_2 are computed from Eq. (6).

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

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

To verify the probability values, we conducted a simulation on different sequences with different packet sizes. In each bitstream, one or two bits were randomly flipped, and the simulation was repeated 10,000 times to estimate the empirical probability value of $\Pr(\text{BEE}=i,\text{CPT}=j|\text{nbErr}=k)$. Table V presents an example of the simulation results for the *crew* sequence. As can be observed, the simulation results are similar to the values in Table IV. These results demonstrate

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

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

that the assumption of having an equal number of bits 0 and 1 in each column is a reasonable assumption and, if they are the same on average, then the results will perfectly match the theoretical results. Similar results have been obtained on other sequences with different quantization parameters (QPs).

TABLE V: Empirical probability value of $\Pr(\text{BEE}=i, \text{CPT}=j|\text{nbErr}=k)$ for the "crew" sequence, QP=27, packet size=1432 bits. ρ=10^{-6}, so $P_1 \approx 10^{-6}$ and $P_2 \approx 10^{-12}$.

BEE	CPT=1	CPT=2.1	CPT=2.2	CPT=3	CPT=4	
1	1	-	-	-	-	$\times \mathbf{P_1}$
2	0	0.062	0.400	0	0	$\times \mathbf{P_2}$
3	0.028	0	0	0	0	
4	0.060	0.064	0	0.349	0	
5	0	0	0	0	0.036	

IV. Proposed CFLD video error correction approach

In this work, we propose to use the UDP header checksum value to decrease the number of possible candidates for list decoding approaches. The checksum value allows us to find the possible locations of the erroneous bits in the bitstream, based on the possible column(s) where they occurred and on the erroneous value at issue (a 0 or a 1). Fig. 10 shows the general schematic of the proposed method. When a packet is received, if it is intact (depending on the UDP checksum value), it will go directly to the video decoder, otherwise it will go through the error correction process. Since the UDP checksum is calculated over the pseudo header, the header and the payload, it is helpful to identify whether an error indicated by C_R, is from the headers or from video data. Therefore, the first step of the correction process is to fix the headers. Some fields of the UDP/real-time transport protocol (RTP) headers are static during the transmission (e.g., Source/Destination Port Num in UDP header), and some other parts are easily predictable (e.g., Sequence Number in RTP header) because of the redundant information in the headers [33], [35].

The next step after fixing all the headers is to decode the bitstream. Here, we consider two conditions which must be satisfied: 1) the sequence should be decodable, and 2) the number of blocks in the corrupted slices should be correct. This step helps save the sequences which had errors somewhere in the headers, but not in the video payload. Thus, they are not put through the correction process. It is assumed that the number of blocks in the packet is known. That is the case in several systems where the number of MBs or coding tree units (CTUs) in a packet is constant or can be deduced from the information within other packets (for instance, the *first MB in slice* syntax element in H.264). During the simulations, it was observed that because of the high compression properties of the encoding process, the coded bitstreams were very sensitive to errors and, in many cases, even a single-bit error can desynchronize the whole packet. This desynchronization creates non-valid syntax or semantic errors in the decoding process. This property is used to differentiate between decodable and non-decodable bitstreams. A decodable bitstream has syntactically/semantically valid codewords. Since it has been observed that decodable bitstreams can nevertheless still be fairly damaged, the constraint on the number of MBs, in the case of H.264 sequences, or CTUs, in the case of HEVC sequences, further eliminates corrupted candidates.

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

- Based on the observed CPT value of C_R, all the possible BEEs are determined and ordered from most likely to least likely, according to the results of Table IV.
- Starting with the most probable BEE, a candidate list is generated. This list includes the potential error locations, based on the observed CPT, which provides the potential error column(s) and the type of flipped bits at issue ($1 \rightarrow 0$ or $0 \rightarrow 1$). For each potential error location, a candidate bitstream is generated.
- Each candidate bitstream passes through the video decoder until one is found that satisfies the two conditions (the sequence is decodable; and the number of MBs, in the case of H.264 sequences, or the number of CTUs, in the case of HEVC, is correct), and from that the best candidate bitstream is determined.
- If none of the candidate bitstreams meets these two conditions, we restart the process of generating a candidate list of potential error locations with the next most probable BEE.

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

the decoder's conditions.

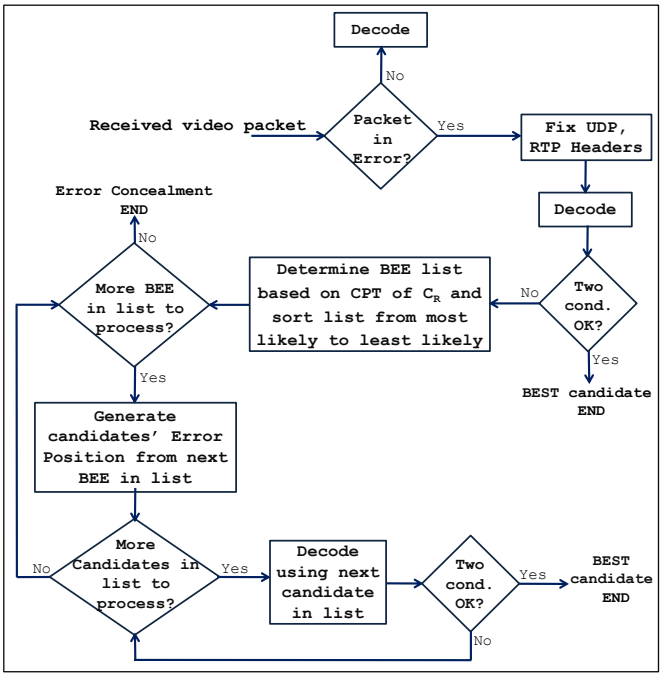

Fig. 10: Proposed CFLD system.

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

V. Experimental Results

In this section, we present the experimental results for the proposed approach. We only consider a single-bit error

TABLE VI: Average number of candidates for different observed packet lengths.

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

since for small values of ρ (e.g., 10^{-6}), the probability of having two or more bits in error is extremely low. We will show the performance of the proposed approach in comparison with other state-of-the-art approaches. In the simulations, we assume that the checksum is intact and the error is in the video payload. This is reasonable for 10,000-bit video packets since we will have 1 chance out of 625 (i.e., 10000/16) that the checksum is hit instead of the video payload. Furthermore, we first attempt to decode the packet (after making sure the headers are correct). Therefore, if the error is really in the checksum, it will not cause a problem in our algorithm.

A. Simulation Setup

We carry out the simulations using the H.264 Baseline profile, which is typically used in conversational services and mobile applications, and the HEVC Low Delay P Main profile. We use the Joint Model (JM) software, version 18.5 [36] for H.264 and the HEVC Test Model (HM) software, version 15 [37], for HEVC. The first 60 frames of NTSC (720×480) sequences (*Driving*, *Opening-ceremony*, *Whale-show*), 4CIF (704×576) sequences (*City*, *Crew*, *Ice*), CIF (352×288) sequence (*Foreman*) and PAL (720×576) sequence (*Walk*) are coded with JM18.5. The sequences are coded in IPPP... format (Intra refresh rate of 30 frames) at a 30 Hz frame rate. Each slice contains a single row of MBs, and is encapsulated into RTP packets.

We also carry out the simulation on HEVC sequences. The first 50 frames of five class B (1920×1080) sequences (*BasketballDrive, BQTerrace, Cactus, Kimono* and *ParkScene*) and four class C (832×480) sequences (*BasketballDrill*, *BQMall*, *PartyScene* and *RaceHorses*) are coded by HM. The slicing mode is chosen to fix number of CTUs in a slice. One row of 64×64 CTUs is considered per slice. All the sequences are encoded with different QP values, namely, 22, 27, 32, and 37.

For each QP, a single frame is randomly selected for error. Then, we apply a uniform error distribution on the bits of each packet with a ρ value varying between approximately 10^{-7} for small QPs to 10^{-6} for large QPs to obtain one bit in error. These residual bit error rates are much higher than those observed in some broadcasting systems, such as DVB-H and DVB-SH-A, in recommended operational conditions [38]. To simplify the simulations, we just consider the errors in the payload part. Also, the UDP checksum is only calculated on the UDP payload, which is an RTP packet. In our transmission simulations, the corrupted slices are identified prior to their decoding by verifying the checksum. The simulation is

repeated 100 times at each QP, to ensure that the location of the erroneous bits did not bias our conclusions.

In H.264 cases, four different approaches are then used to handle the corrupted sequences: (i) frame copy (FC) concealment by JM, (ii) state-of-the-art STBMA [13], error correction using HO-MLD [28], and the proposed CFLD approach. The first 30 frames are kept intact to allow the HO-MLD approach to gather video statistics. When CFLD approach falls back to EC, here we consider STBMA to be fair with other approaches (STBMA itself and HO-MLD which uses STBMA). However, our method never reached the point of calling EC during the simulations. In the case of HEVC sequences, the corrupted packets are handled by (i) implemented FC-EC in HM and (ii) the CFLD approach.

B. Simulation Results

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

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

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

For performance evaluation, we calculated the peak signal-to-noise ratio (PSNR) and structural similarity index measurement (SSIM) [39] of the corrupted frames after reconstruction, using various approaches in order to compare their visual quality. Table VIII and Table IX show the average PSNR values for different error handling approaches on H.264, HEVC class B and C sequences. The last column in the tables showing the percentage of times CFLD was able to correct the error such that the PSNR of the reconstructed bitstream would be exactly the same as the intact one. The simulation was repeated 100 times for each sequence for different QP values. The results for the H.264 sequences indicate that the proposed approach outperforms JM-FC, STBMA and HO-MLD in all cases.

Fig. 11 shows the average PSNR gains of each approach at different QP values. We observe that the proposed approach provides significant PSNR gains over JM-FC for all four QP values. For instance, it is more than 5 dB better than JM-FC at QP=22. On average, over all QPs, the CFLD approach was able to correct the bitstream 79% of the time compared to HO-MLD with only 6% in our simulation. Also, it offers a 2.79 dB gain over JM-FC and 1.19 dB and 1.41 dB gains over STBMA and HO-MLD, respectively. In the case of HEVC, the CFLD approach corrects the corrupted bitstream 91% of the time, and offers 2.35 dB and 4.97 dB gains over HM-FC in class B and C sequences, respectively.

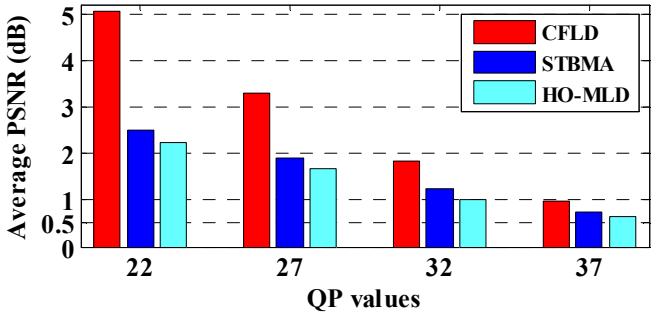

Fig. 11: Average PSNR gains of HO-MLD, STBMA and CFLD method over JM-FC for different QP values of H.264 sequences.

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

The difficulties of accessing soft information at the application layer in existing video communication systems make the approaches using only hard information very appealing to build robust video error correction systems. But ignoring the soft information in traditional list decoding approaches makes them highly inefficient (as the following simulations will show). Indeed, since all the bits then have the same probability of being flipped, as a result, all the candidate bitstreams have the same probability of being the best one. The best candidate would then be chosen through an exhaus-

TABLE VIII: Comparison of the average PSNR of reconstructed corrupted frames for different methods in H.264. The differences between each method and the JM-FC method appear in parentheses. The last column shows the percentage of packets that were fully corrected by the proposed approach.

Sequence	QP	Average PSNR of reconstructed corrupted frame					
		Intact	JM-FC	STBMA	HO-MLD	CFLD	
City (704×576)	22	40.88	36.47	40.32 (+3.58)	39.60 (+3.12)	40.77 (+4.3)	88%
	27	36.63	34.38	36.43 (+2.05)	35.81 (+1.43)	36.6 (+2.21)	84%
	32	33.08	32.06	32.99 (+0.93)	32.79 (+0.73)	33.06 (+1)	86%
	37	30.05	29.54	29.99 (+0.45)	29.95 (+0.41)	30.01 (+0.46)	78%
Crew (704×576)	22	41.76	39.21	40.69 (+1.48)	40.25 (+1.04)	41.76 (+2.55)	86%
	27	38.52	37.25	38.07 (+0.82)	37.69 (+0.72)	38.51 (+1.27)	84%
	32	35.7	34.91	35.38 (+0.47)	35.33 (+0.42)	35.64 (+0.73)	82%
	37	32.99	32.58	32.82 (+0.25)	32.83 (+0.25)	32.96 (+0.39)	80%
Ice (704×576)	22	43.73	38.58	41.83 (+3.26)	41.78 (+3.2)	43.49 (+4.91)	76%
	27	41.45	37.25	39.69 (+2.43)	39.75 (+2.5)	41.31 (+4.06)	90%
	32	39	36.38	38.11 (+1.72)	38.02 (+1.63)	38.93 (+2.55)	76%
	37	36.43	34.55	35.94 (+1.39)	35.91 (+1.36)	36.39 (+1.84)	78%
Foreman (352×288)	22	41.34	37.41	39.31 (+1.9)	39.21 (+1.8)	41.06 (+3.65)	64%
	27	37.83	35.65	36.8 (+1.14)	36.59 (+0.94)	37.66 (+2.01)	52%
	32	34.68	33.61	34.14 (+0.53)	34.14 (+0.53)	34.57 (+0.96)	74%
	37	31.96	31.47	31.61 (+0.14)	31.73 (+0.26)	31.94 (+0.47)	90%
Opening ceremony (720×480)	22	39.32	38.26	38.55 (+0.29)	38.8 (+0.54)	39.32 (+1.06)	88%
	27	35.39	35.03	35.08 (+0.05)	35.11 (+0.09)	35.34 (+0.31)	84%
	32	31.39	31.2	31.24 (+0.04)	31.26 (+0.06)	31.39 (+0.18)	94%
	37	27.76	27.66	27.71 (+0.04)	27.72 (+0.06)	27.76 (+0.1)	92%
Whale show (720×480)	22	41	35.69	36.58 (+0.89)	36.96 (+1.28)	40.93 (+5.24)	68%
	27	36.37	33.54	34.22 (+0.68)	34.42 (+0.88)	36.34 (+2.8)	72%
	32	32.08	30.86	31.2 (+0.34)	31.16 (+0.31)	32.06 (+1.2)	84%
	37	28.36	27.89	27.99 (+0.1)	27.97 (+0.08)	28.33 (+0.44)	76%
Driving (720×480)	22	41	34	38.02 (+4.02)	37.28 (+3.28)	40.45 (+6.45)	70%
	27	37.05	32.72	35.7 (+2.98)	34.72 (+2)	36.91 (+4.19)	64%
	32	33.3	31.03	32.76 (+1.72)	32.17 (+1.14)	33.19 (+2.16)	82%
	37	30.05	28.96	29.79 (+0.84)	29.59 (+0.63)	30 (+1.04)	82%
Walk (720×576)	22	43.29	30.27	34.66 (+4.39)	33.79 (+3.51)	42.48 (+12.21)	76%
	27	39.33	29.65	34.77 (+5.12)	33.34 (+4.7)	39.14 (+9.5)	72%
	32	35.56	29.33	33.52 (+4.19)	32.65 (+3.32)	35.23 (+5.9)	82%
	37	31.91	28.59	31.21 (+2.61)	30.72 (+2.12)	31.78 (+3.19)	78%
Average gain over JM-FC			**0**	**+1.6**	**+1.39**	**+2.79**	**79%**

TABLE IX: Comparison of the average PSNR of reconstructed corrupted frames for different methods in HEVC class B and C sequences. The differences between the CFLD and HM-FC methods appear in parentheses. The last column shows the percentage of packets that were fully corrected by the proposed approach.

Sequence	QP	Average PSNR of reconstructed corrupted frame			
		Intact	HM-FC	CFLD	
Class B Sequences					
BQ Terrace (1920×1080)	22	38.89	35.16	35.76 (+0.6)	58%
	27	36.3	34.32	35.68 (+1.36)	82%
	32	33.76	32.37	33.66 (+1.29)	92%
	37	31.17	30.26	31.15 (+0.89)	89%
Basketball Drive (1920×1080)	22	39.89	32.53	38.49 (+5.95)	84%
	27	38.23	32.28	37.67 (+5.39)	90%
	32	36.7	31.81	36.47 (+4.66)	96%
	37	34.8	31.51	34.8 (3.29)	100%
Cactus (1920×1080)	22	39.20	36.82	37.89 (+1.07)	76%
	27	36.74	34.59	36.25 (+1.66)	88%
	32	34.65	33.56	34.59 (+1.03)	98%
	37	32.31	31.55	32.03 (+0.48)	96%
Kimono (1920×1080)	22	42.15	36.69	41.62 (+4.93)	90%
	27	40.04	36.10	39.81 (+3.71)	96%
	32	38.20	34.78	38.07 (+3.29)	98%
	37	35.30	33.40	35.30 (+1.9)	98%
Park Scene (1920×1080)	22	40.11	37.39	39.63 (+2.24)	82%
	27	37.33	35.42	37.19 (+1.77)	96%
	32	34.83	33.86	34.74 (+0.88)	94%
	37	32.17	31.58	32.17 (+0.59)	100%
Average gain over HM-FC			**0**	**+2.35**	**91%**
Class C Sequences					
Basketball Drill (832×480)	22	40.44	31.9	39.91 (+8.01)	94%
	27	37.41	30.84	37.06 (+6.22)	94%
	32	34.66	30.07	34.56 (+4.49)	98%
	37	32.11	29.21	32 (+2.8)	98%
BQ Mall (832×480)	22	39.84	31.04	39.16 (+8.12)	92%
	27	36.91	30.03	36.23 (+6.2)	92%
	32	33.86	29.69	33.48 (+3.79)	94%
	37	30.68	27.83	30.5 (+2.67)	92%
Party Scene (832×480)	22	38.14	32.57	35 (+2.43)	72%
	27	34.66	31.32	33.52 (+2.2)	84%
	32	31.07	29.38	30.98 (+1.6)	96%
	37	27.76	26.94	27.47 (+0.53)	94%
Race Horses (832×480)	22	39.29	26.01	35.94 (+9.93)	70%
	27	36.21	25.48	35.16 (+9.68)	90%
	32	32.6	25.8	32.18 (+6.38)	92%
	37	29.44	24.98	29.32 (+4.34)	96%
Average gain over HM-FC			**0**	**+4.97**	**91%**

tive (brute force) search on all the candidates without any order preference. In the following simulations, we used the exhaustive search list decoding (ESLD) approach as another benchmark for comparison against the proposed CFLD to represent the performance of list decoding methods that would not have access to soft information to order their candidates. In this ESLD approach, all candidates will sequentially go through the video decoder and the first candidate that satisfies the decoder's two conditions is chosen as the best candidate. The candidates are generated by sequentially flipping the bits of the received packet from the first to the last one. We use this same order for CFLD but only considering the potential bit error locations.

Fig. 12 presents the PSNR and SSIM distributions (box plots) of two sequences having a low percentage of fully

corrected slices in Table VIII. As shown in the figures, for both sequences, the median value (red line in the middle of the box) of PSNR and SSIM for CFLD is exactly the same as the intact one and also the lower and higher bands of boxes (25-75 percentile of the data) confirm that in most cases the CFLD has the same or closest value to the intact one which is obviously higher than the other approaches. The detailed information of this simulation is presented in Table X. It is obvious that the CFLD search is much less complex than the ESLD search and it significantly reduces the number of candidates from N (for ESLD) to $N/32$. Thereby, it has more chances that the first candidate is the fully corrected packet. The ESLD perfectly corrects damaged H.264 packets 41% and 21% of the time for *Ice* and *Foreman* sequences, respectively, while the value for CFLD are 91% and 61%. In fact, if CFLD fails to fully correct the packet, for sure ESLD will fail. This is because ESLD will always retain a candidate that either comes before that of CFLD or the same one. Therefore, it is not possible for ESLD to select a fully corrected packet without CFLD also selecting it.

This has a huge impact on the visual quality of the reconstructed corrupted frame and, more importantly, prevents the propagation of errors to subsequent frames due to the predictive coding. In fact, a few decibels PSNR difference on the reconstructed corrupted frame increases to several dBs on subsequent frames due to this drift. Since in the simulations we choose the first satisfied candidate as the best one, there are some outliers (as shown with '+' red symbol) in the CFLD results. However, most cases which have very low PSNR, can be eliminated by adding an additional pixel-domain step (such as boundary matching or border checking) in our system. Indeed, instead of selecting the first candidate which satisfies the two conditions, we could rank all candidates satisfying the two conditions using a yet-to-be-defined pixel-domain likeliness measure or other likeliness measure based on the decoded information (e.g., motion vectors). For instance, for all the candidates satisfying the two conditions, we could use a pixel-domain metric such as the one based on the sum of distributed motion-compensated blockiness (SDMCB) proposed in [17] to rank them. We thus could select the candidate having the highest likeliness (e.g., lowest SDMCB value).

TABLE X: Detailed information of the box plot of Fig. 12: average PSNR and SSIM values, percentage of fully corrected packets.

Seq.		Intact	JM-FC	STBMA	HO-MLD	ESLD	CFLD
Ice QP=37	PSNR	36.49	35.07	35.96	35.91	35.91	36.48
	SSIM	0.9681	0.9654	0.967	0.9669	0.9679	0.9681
	fully corrected packets (%)				9%	41%	91%
Foreman QP=27	PSNR	37.77	36.39	37.17	37.11	37.36	37.65
	SSIM	0.9432	0.9383	0.9412	0.9414	0.9421	0.9428
	fully corrected packets (%)				1%	21%	61%

The gain in subjective quality is illustrated in Fig. 13. Comparing the reconstructed frame, it is clear that the CFLD method outperforms the other approaches and further confirms the robustness and superiority of the proposed method.

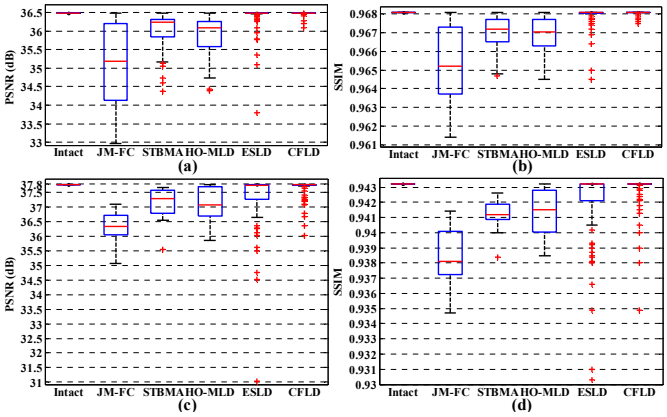

Fig. 12: PSNR and SSIM distributions of 100 runs on frame 45 of H.264 *Ice* at QP=37 (top) and *Foreman* at QP=27 (bottom) sequences.

Fig. 13: Visual comparison of a reconstructed frame with H.264 *Ice* sequence at QP=37 by different methods. One bit was flipped in frame 45, slice 22 and bit 381. The packet contains 472 bits. The proposed checksum provides 11 candidates. The first valid candidate which satisfied the two mentioned conditions is picked as CFLD output and the packet was perfectly corrected. The PSNR and SSIM values of each approach are as follows, respectively: Intact (36.49 dB, 0.9681), JM-FC (34.12 dB, 0.9649), STBMA (34.37 dB,0.9659), HO-MLD (34.39 dB, 0.966) and CFLD (36.49 dB, 0.9681).

From the results of all figures and tables, it can be inferred that the proposed approach can effectively remove non-valid candidates, and in nearly 80% of the cases in H.264, and 90% of the cases in HEVC, the sequence can be perfectly corrected. In contrast, HO-MLD perfectly corrects damaged H.264 packets only 6% of the time. So, as a result, the proposed CFLD provides a significantly higher PSNR value and better quality compared to other approaches. This is important not only for the corrupted frame, but for the following ones, as fewer visible drifting effects will result.

VI. Conclusion

In this paper, we proposed a new method that exploits the receiver side UDP checksum information to dramatically reduce the number of candidates that need to be considered by list decoding. For one bit in error, the method allows the removal of 97% of the candidates. For two bits in error, this reduction reaches 99.6%. Such a filtering of the candidates

as proposed, supplemented by checksum information dramatically reduces the complexity of the list decoding approach. Simulations results showed that the H.264 baseline, using CAVLC, is more robust to desynchronization due to errors, as compared to HEVC, using CABAC. This led to better error correction performance for H.265 as corrupted packets are more likely to cause desynchronization errors which invalidate the erroneous packets. Our simulation results revealed that 79% of the H.264 could be corrected perfectly, compared to 91% for HEVC. The proposed approach provides, on average, a 2.79 dB gain over FC-EC using JM, and a 3.57 dB gain over our implementation of FC-EC in HM. Although, current applications do not typically have access to soft information, the proposed CFLD approach can also be applied to that context, allowing it to perform even better by enabling it to exploit the soft information to rank the candidate bitstreams in each BEE. We also expect a further increase in performance by exploiting pixel domain information to select the best decodable candidate rather than selecting the first decodable candidate. This will be the subject of future research.

REFERENCES

[1] G. J. Sullivan and T. Wiegand, "Video compression-from concepts to the H.264/AVC standard," *Proceedings of the IEEE*, vol. 93, no. 1, pp. 18–31, 2005.

[2] ISO/IEC JTC 1/SC 29/WG 11, "High Efficiency Video Coding," 2013.

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

[4] Y. Wang, J. Ostermann, and Y. Q. Zhang, *Video processing and communications*, vol. 5. Prentice Hall Upper Saddle River, 2002.

[5] Y. Wang, S. Wenger, J. Wen, and A. K. Katsaggelos, "Error resilient video coding techniques," *IEEE Signal Process. Mag.*, vol. 17, no. 4, pp. 61–82, 2000.

[6] J. Xiao, T. Tillo, C. Lin, Y. Zhang, and Y. Zhao, "A real-time error resilient video streaming scheme exploiting the late-and early-arrival packets," *IEEE Trans. Broadcast.*, vol. 59, no. 3, pp. 432–444, 2013.

[7] H. Sun and W. Kwok, "Concealment of damaged block transform coded images using projections onto convex sets," *IEEE Trans. Image Process.*, vol. 4, no. 4, pp. 470–477, 1995.

[8] W. Zeng and B. Liu, "Geometric-structure-based error concealment with novel applications in block-based low-bit-rate coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, no. 4, pp. 648–665, 1999.

[9] J. Liu, G. Zhai, X. Yang, B. Yang, and L. Chen, "Spatial error concealment with an adaptive linear predictor," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 3, pp. 353–366, 2015.

[10] M. J. Chen, L. G. Chen, and R. M. Weng, "Error concealment of lost motion vectors with overlapped motion compensation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, no. 3, pp. 560–563, 1997.

[11] J. Wu, X. Liu, and K. Y. Yoo, "A temporal error concealment method for H.264/AVC using motion vector recovery," *IEEE Trans. Consum. Electron.*, vol. 54, no. 4, pp. 1880–1885, 2008.

[12] J. Zhou, B. Yan, and H. Gharavi, "Efficient motion vector interpolation for error concealment of H.264/AVC," *IEEE Trans. Broadcast.*, vol. 57, no. 1, pp. 75–80, 2011.

[13] 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," *IEEE Trans. Multimedia.*, vol. 10, no. 1, pp. 2–15, 2008.

[14] M. Ma, O. C. Au, S. G. Chan, and M. T. Sun, "Edge-directed error concealment," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 3, pp. 382–395, 2010.

[15] Z. Zhou, M. Dai, R. Zhao, B. Li, H. Zhong, and Y. Wen, "Video error concealment scheme based on tensor model," *Multimedia Tools and Applications*, pp. 1–17, 2016.

[16] L. Superiori, O. Nemethova, and M. Rupp, "Performance of a H.264/AVC error detection algorithm based on syntax analysis," in *Proc. MoMM*, pp. 49–58, 2006.

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

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

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

[20] Y. Wang and S. Yu, "Joint source-channel decoding for H.264 coded video stream," *IEEE Trans. Consum. Electron.*, vol. 51, no. 4, pp. 1273–1276, 2005.

[21] C. Bergeron and C. Lamy-Bergot, "Soft-input decoding of variable-length codes applied to the H.264 standard," in *Proc. IEEE 6th workshop Multimedia Signal Process.*, pp. 87–90, 2004.

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

[23] D. Levine, W. E. Lynch, and T. Le-Ngoc, "Iterative joint source-channel decoding of H.264 compressed video," in *Proc. IEEE Int. Symp. Circuits Syst.*, pp. 1517–1520, 2007.

[24] N. Q. Nguyen, W. E. Lynch, and T. Le-Ngoc, "Iterative joint source-channel decoding for H.264 video transmission using virtual checking method at source decoder," in *Proc. IEEE 23rd Can. Conf. Electr. Comput. Eng.*, pp. 1–4, 2010.

[25] R. A. Farrugia and C. J. Debono, "Robust decoder-based error control strategy for recovery of H.264/AVC video content," *IET Communications*, vol. 5, no. 13, pp. 1928–1938, 2011.

[26] F. Caron and S. Coulombe, "A maximum likelihood approach to video error correction applied to H.264 decoding," in *Proc. IEEE 6th Int. Conf. Next Gen. Mob. Appl., Serv., Technol.*, pp. 1–6, 2012.

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

[28] 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 Trans. Circuits Syst. Video Technol.*, vol. 25, no. 7, pp. 1161–1174, 2015.

[29] F. Golaghazadeh, S. Coulombe, F.-X. Coudoux, and P. Corlay, "Low complexity H.264 list decoder for enhanced quality real-time video over IP," in *submitted to the 2017 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, May 2017.

[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] K. R. Fall and W. R. Stevens, *TCP/IP illustrated, volume 1: The protocols*. Addison-Wesley, 2011.

[32] R. A. Dean, *Elements of abstract algebra*. John Wiley & Sons Inc, 1966.

[33] J. Postel, "User datagram protocol." IETF, RFC 768, Aug. 1980. [Online]. Available: https://www.rfc-editor.org/rfc/rfc768.txt.

[34] P. E. Pfeiffer, *Concepts of Probability Theory: Second Revised Edition*. Dover Publications Inc., 2013.

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

[36] "H.264/AVC JM Reference Software." version 18.5, [Online]. Available: http://iphome.hhi.de/suehring/tml/.

[37] "HEVC Test Model Software." version 15, [Online]. Available: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/branches/.

[38] L. Polák and T. Kratochvíl, "DVB-H and DVB-SH-A performance and evaluation of transmission in fading channels," in *IEEE 34th Int. Conf. Telecommunications and Signal Processing (TSP)*, pp. 549–553, 2011.

[39] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Trans. Image Process*, vol. 13, no. 4, pp. 600–612, 2004.

Firouzeh Golaghazadeh received the B.Sc. degree in electrical engineering from K. N. Toosi University of Technology, Tehran, Iran, in 2007 and the M.Sc. degree in information technology from Iran University of Science and Technology, Tehran, Iran, in 2010. She is currently pursuing the Ph.D. degree at École de technologie supérieure, Montreal Canada. Her research interests include multimedia signal processing, video coding, error correction and concealment for H.264/AVC and HEVC.

Stéphane Coulombe (S'90-M'98-SM'01) received a B.Eng. degree in Electrical Engineering from École Polytechnique de Montréal, Canada, in 1991, and a Ph.D. degree in Telecommunications (image processing) from INRS-Telecommunications, Montreal, in 1996. He is currently a Professor in the Software and IT Engineering Department, École de technologie supérieure (ÉTS is a constituent of the Université du Québec network). From 1997 to 1999, he was with Nortel Wireless Network Group in Montreal, and from 1999 to 2004, he worked with the Nokia Research Center, Dallas, TX, as Senior Research Engineer and as a Program Manager in the Audiovisual Systems Laboratory. He joined ÉTS in 2004, where he currently carries out research and development on video processing and systems, compression, and transcoding. Since 2009, he has held the Vantrix Industrial Research Chair in Video Optimization.

François-Xavier Coudoux (M'07) received the M.S. and Ph.D. degrees in electrical engineering from the University of Valenciennes, Valenciennes, France, in 1991 and 1994, respectively. Since 2004, he has been a Professor in the Department of Opto-Acousto-Electronics of the Institute of Electronics, Microelectronics, and Nanotechnologies, Valenciennes, France (UMR 8520). His research interests include telecommunications, multimedia delivery over wired and wireless networks, image quality, and adaptive video processing.

Patrick Corlay received the Ph.D. degree in 1994 from the University of Valenciennes, France. Since 2016, he has been a Professor at the Department of Opto-Acousto-Electronics of the Institute of Electronics, Microelectronics, and Nanotechnologies, France (UMR 8520). His current research interests are in telecommunications, multimedia delivery over wired and wireless networks, and optimal quality of service for video transmission.