Accepted in the 2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE 2017), 2017
Windsor, ON, Canada, 30 April-3 May 2017 https://doi.org/10.1109/CCECE.2017.7946732

Low complexity H.264 list decoder for enhanced quality real-time video over IP

Firouzeh Golaghazadeh, Stéphane Coulombe
Department of Software and IT Engineering
École de technologie supérieure
1100 Notre-Dame Street West, Montreal, Canada
Firouzeh.Golaghazadeh.1@ens.etsmtl.ca
Stephane.Coulombe@etsmtl.ca

François-Xavier Coudoux, Patrick Corlay
Univ. Valenciennes, CNRS, Univ. Lille, ISEN,
Centrale Lille, UMR 8520 - IEMN,
DOAE, F-59313 Valenciennes, France
Francois-Xavier.Coudoux@univ-valenciennes.fr
Patrick.Corlay@univ-valenciennes.fr

***Abstract*—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 in the packet by analyzing the calculated UDP checksum value at the receiver side. This makes it possible 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 in 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 97% reduction in the number of candidates. Our simulation results on H.264 compressed sequences reveal that, on average, the error is corrected perfectly 80% of the time, and thus, the original bitstream is fully recovered when the first valid candidate is considered as the best candidate. In addition, the proposed approach provides, on average, a 2.78 dB gain over the error concealment approach used by the H.264 reference software, as well as 1.31 dB and 1.51 dB gains over the state-of-the-art error concealment and HO-MLD approaches, respectively.**

***Index Terms*—Video Transmission over IP, Video Error Correction, H.264, List Decoding, UDP Checksum.**

I. Introduction

H.264 is the most widely used video coding standard, with significant penetration in the optical disc, broadcast, and streaming video markets. In 2014, about half the bits transmitted on communication networks worldwide were for coded video using MPEG advanced video coding (AVC) [1]. Due to the high compression efficiency of H.264/AVC, the compressed video data is more vulnerable in error-prone networks. Various error control mechanisms have been proposed to combat visual quality degradation caused by transmission errors [2]. Retransmission is one of the basic mechanisms used to provide reliable communication, but it is rarely used in real-time conversational or broadcasting/multicasting applications due to the added delay it involves or to the lack of a feedback channel [3]. Error resilience injects redundancies during source coding to make the streams more robust against transmission errors, thereby allowing the decoder to better deal with the loss of information. However, every error resilience method reduces coding efficiency or involves bit rate sacrifices, and is inefficient, especially when there is no transmission error [4]. Compared to the other approaches mentioned, error concealment, as a post-processing mechanism at the decoder side, will not require any additional bit rate, and will not introduce retransmission delays. Error concealment methods estimate lost areas by exploiting the inherent correlation between adjacent pixels (spatial error concealment) [5], [6] or neighboring frames (temporal error concealment) [7], [8] or both (spatiotemporal error concealment) [9], [10].

Partially damaged packets may contain valuable information that can be used to enhance the visual quality of reconstructed videos [11]. The goal of joint source channel decoding (JSCD) [12], [13], [14], [15] and list decoding [16], [17], [18] is to use these corrupted packets to recover the originally sent packet content, rather than assuming that the whole packet is lost. In [12], sequential decoding and soft information provided by the channel decoder is used in predicting the residual coefficients coded with context-adaptive variable-length coding (CAVLC) in the H.264 Extended profile. The additional information, such as packet length in bits and number of macroblocks (MBs) in the slice, is used as the constraint to define a valid packet. A maximum a posteriori-based JSCD approach is employed for the decoding of motion vectors and CAVLC of H.264 in [13] and [14], respectively. In [15], 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. The candidates are then 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 [16], [17], 300 likeliest candidates are generated based on the soft value of flipped bits. The slice candidate with the smallest sum of soft values is identified as the most probable one. Moreover, in [17], a virtual checking step is proposed to accelerate the semantic verification process of each candidate by considering the information of previous failed candidates.

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

Farrugia [18] 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 the 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 [19], [20], [21], 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 code words 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.

In this work, we propose a novel list decoding approach which exploits the receiver side user datagram protocol (UDP) checksum to alleviate the large solution space problem of the list decoding approaches. We show that the UDP checksum of corrupted packets exhibits specific patterns. Observing these specific patterns allows us to identify the potential error locations in the bitstream by assuming that there is one erroneous bit in the packet. This information is used to remove non-valid candidate bitstreams at the first step of a list decoding approach. When there is one bit in error in the packet, the checksum verification will eliminate 97% of the non-valid candidates. That way, the proposed approach considerably reduces the number of candidate bitstreams as compared to general list decoding approaches, and accordingly reduces the computational complexity. The experimental results for H.264 reveal the superiority of the proposed approach as compared to other state-of-the-art methods [9], [21]. Finally, a detailed study of multiple bits in error with new experimental results for both H.264 and HEVC is the object of a submitted publication [22].

This paper is organized as follows. A detailed introduction of the UDP checksum is presented in Section II. In Section III, we present how the UDP checksum can be applied in error correction, especially for the case of one bit in error. In Section IV, we explain how the checksum can be employed to reduce the number of candidates in general list decoding approaches. The performance of the proposed checksum-based approach for error correction, compared to other state-of-the-art methods is provided in Section V. Concluding remarks are drawn in Section VI.

II. UDP Checksum Definition

The UDP checksum is a 16-bit field in the UDP header, and is one's complement of the one's complement sum of a pseudo UDP header, UDP header and application data message [23].

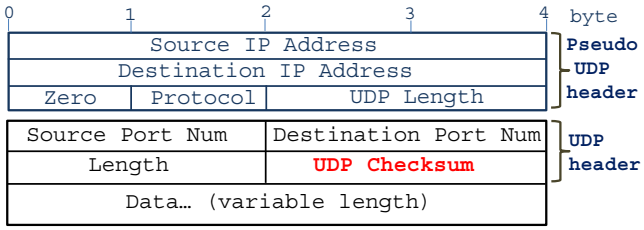

Fig. 1. UDP datagram and pseudo header.

Fig. 1 shows the UDP datagram format and its 12-byte prefix as a pseudo UDP header. The pseudo UDP header contains the source and destination internet protocol (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, as shown in Fig. 1, and it is computed as follows:

- Divide the packet into chunks of 16-bit words. If necessary, pad the data with a one-byte zero at the end to render it into a multiple of 16 bits.
- Compute one's complement sum over all the words. If there is an overflow, the one's complement sum operation involves an "end-around carry". The end-around carry scheme routes carry-out signals 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 [24].
- Flip all the bits of the final sum (one's complement).

It should be noted that during the calculation of the checksum at the transmission side, the two-byte checksum field in the UDP header is set to zero, and after the calculation, it is replaced by the computed one 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, as is mentioned in the standard, it should be transmitted as all ones (FFFF). The calculated checksum for a real packet can never be FFFF unless all the words in the packet are zeros [24]. 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 UDP checksum, rather than zeros. Received data is valid if the recomputed checksum at the receiver side is zero; otherwise, it is corrupted.

Fig. 2 presents an example of the UDP checksum computation. In this example, the entire packet content is considered as having 32-bit length. The above-mentioned steps were performed to establish the transmission side's checksum (C_T). As can be seen, at the reception side, the value of C_T is used during the calculation of the receiver side's checksum (C_R). The zero value of C_R validates the received packet.

III. Exploiting Checksum for Error Correction

Let us assume that the UDP packet has a length of N bits (including padding), made up of $m{=}N/16$ 16-bit words, as $\{W_0, W_2, ..., W_{cs}, ..., W_{m-1}\}$ and W_{cs} is the 16-bit word in the checksum field. Mathematically, the set of 16-bit words, represented here as $W{=}\{0000, 0001, ..., FFFF\}$, and the one's complement sum operation, denoted as +, together form an

TRANSMISSION

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

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

RECEPTION

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

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

Fig. 2. UDP checksum calculation example on a packet data at the transmission side and its validation process at the reception side (transmission of the checksum is not shown).

Abelian group [25]. Then $\overline{W}$ is defined as the inverse element of W, which is obtained by performing a one's complement operation on element W (flipping all its bits). 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. Mathematically, C_T can be expressed as shown in (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 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 (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 version 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 (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 one's complement sum of all transmitted words, is included in the computation of C_R. Therefore, at the reception side, when it is added to the one's complement sum of all words, the identity element FFFF is obtained.

The C_R from (2) can be expanded as:

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

From (3), it follows that when there is no error in column c of word i ($w_{i,c}=\widehat{w}_{i,c}$), $\widehat{w}_{i,c}+\overline{w}_{i,c}=1$. Let us assume that there is only one erroneous bit in the packet. Then, two different bit modification cases should be considered:

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

If $1_j \rightarrow 0_j$, where $w_{i,j}=1$ and $\widehat{w}_{i,j}=0$, then $\widehat{w}_{i,j}+\overline{w}_{i,j}=0$ for column j, and $\widehat{w}_{i,c}+\overline{w}_{i,c}=1$ for the other columns $c\neq j$. 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 (3), which flips all the bits, C_R will have a bit 1 in column j while the rest of the bits will be 0. Fig. 3 contains an example of this case on the same packet of Fig. 2. A single bit 1 was flipped to 0 in column 26 of the packet, which corresponds to column 10 of the second word (26 modulo 16 equals 10). As can be seen, C_R has a bit 1 in column 10 and 0 in the others.

For the case of $0_j \rightarrow 1_j$ ($w_{i,j}=0$ and $\widehat{w}_{i,j}=1$), $\widehat{w}_{i,j}+\overline{w}_{i,j}=0$ for column j with an extra *carry* and $\widehat{w}_{i,c}+\overline{w}_{i,c}=1$ for the other columns ($c\neq j$). That 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 along 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 be 0. Therefore, C_R will have a 0 in column j and a 1 for all the other columns.

Fig. 4 summarizes all the 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 values. All

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

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

Fig. 3. UDP checksum validation procedure example at the reception side on a received packet data having one bit error at bit 26.

the 16 different values of C_R in the case of flipping $1_j \rightarrow 0_j$ have a similar pattern of fifteen bits 0 and a single bit 1. The column location of the bit 1 in the pattern signals the potential error locations in the words. On the other hand, all the 16 different values of C_R in the case of flipping $0_j \rightarrow 1_j$ have a similar pattern of fifteen bits 1 and a single bit 0. In this case, the column location of the bit 0 in the pattern indicates the potential error locations in the words.

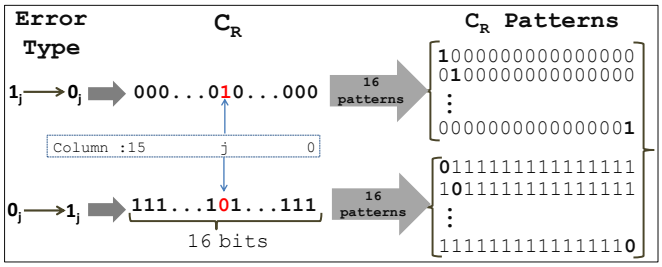

Fig. 4. One bit in error and its corresponding 32 patterns of C_R value. Bold bits in C_R patterns indicate the error column.

When there is a single error in the payload of a packet, the calculated C_R at the receiver side will have one of the patterns of Fig. 4. The error column is indicated by the location of the bit which is different from the others in C_R value. Furthermore, different patterns of C_R when $1_j \rightarrow 0_j$ (fifteen bits 0) and when $0_j \rightarrow 1_j$ (fifteen bits 1), indicate if a bit 1 or a bit 0 was flipped in column j, thereby further reducing the number of candidate error locations in the packet.

Let us revisit the example of Fig. 3. The non-zero value of C_R demonstrates that the received packet is corrupted. In addition, the location of bit 1 in the C_R pattern, column 10, signals that the potential error locations are the 10th bit of each of the words in the packet. So, in this example, the 10th bit and the 26th bit of the packet are the two potential error locations. Moreover, the observed pattern of C_R (fifteen bits 0 and a single bit 1) indicates that a bit 1 was flipped to 0. Then, all the potential error locations having a bit value of 0 will constitute the final set of potential error locations. In this case, only the 26th bit of the packet has a value of 0, and is the final error location (in large packets, the list of candidates usually contains more elements).

IV. Reduction of Candidate List

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 helps determine the potential error column in the words as well as 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 containing N bits, there are N possible candidate bitstreams for the case of one bit in error, whereas the proposed checksum validation process 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 flipped bit. Since the packet is divided into words of 16 bits, 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 then be $N/32$. This means that in case of one bit in error, there is about a 97% reduction in the number of candidate bitstream, and only about 3% of the candidate bitstreams should be considered, as compared to other list decoding approaches. Table I presents the average number of candidates for different packet lengths in the case of one bit in error by using the proposed checksum verification.

TABLE I
AVERAGE NB OF CANDIDATES FOR VARIOUS OBSERVED PACKET LENGTHS.

Packet length (bits)	Average number of candidates	Eliminated candidates
272	9	97%
768	24	97%
1120	35	97%
5280	165	97%

V. Simulation Results

To evaluate the gains provided by the proposed checksum validation process in list decoding, we coded the first 60 frames of an NTSC (720×480) sequence (*Opening-ceremony*), a 4CIF (704×576) sequence (*Crew*), a CIF (352×288) sequence (*Foreman*), and a PAL (720×576) sequence (*Walk*), using the H.264 Baseline profile of the Joint Model (JM) software, version 18.5 [26]. The sequences are coded in IPPP... format (Intra refresh rate of 30 frames) at a frame rate of 30 Hz. Each slice contains a single row of MBs, which are encapsulated into real-time transport protocol (RTP) packets. All the sequences are encoded with different quantization

parameters (QPs), namely, 22, 27, 32, and 37. For each QP, a single frame (between frames 31 and 60) is randomly selected for error. Then, we apply a uniform error distribution on the bits of each packet with a channel residual bit error rate (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 [27]. Moreover, for this range of bit error rates, having more than one bit in error is extremely infrequent. To simplify the simulations, we just consider the errors in the payload part. Also, the UDP checksum is only calculated on the UDP payload, which is an RTP packet. In our transmission simulations, the corrupted slices are identified prior to their decoding by verifying the checksum. The simulation is repeated 100 times at each QP, to ensure that the location of the erroneous bits did not bias our conclusions. Four different approaches are then used to handle the corrupted sequences: (i) frame copy (FC) concealment by JM, (ii) a state-of-the-art concealment approach using the spatiotemporal boundary matching algorithm (STBMA) [9], (iii) error correction using HO-MLD [21], and (iv) the proposed approach. The first 30 frames are kept intact to allow the HO-MLD approach to gather video statistics.

Like all other list decoding approaches, the generated candidate bitstreams should go through additional constraints. First, all the candidate bitstreams should be checked for syntax or semantic errors. We perform this by decoding each candidate bitstream to see whether or not it is decodable. Secondly, we check whether or not the number of MBs in the decoded bitstream is valid [17], [18]. Since we coded the sequence with one row of MBs in each slice, we already know about the second condition. Moreover, the number of MBs in the slice can be deduced from the information within other slices (for instance, the *first MB in slice* syntax element in H.264 coded sequence). The first bitstream that satisfies these two mentioned constraints is considered as the best candidate.

For performance evaluation, we calculated the peak signal-to-noise ratio (PSNR) of the corrupted frames after reconstruction, using various approaches in order to compare their objective video quality. The structural similarity index measurement (SSIM) [28] was also evaluated as it is well known that SSIM is better correlated to human visual judgment than PSNR. Table II refers to the average PSNR values for different error handling approaches on H.264 sequences. The last column in the table shows the percentage of times the proposed approach was able to perfectly correct the packet, consequently leaving the PSNR of the reconstructed bitstream exactly the same as the intact one. The simulation was repeated 100 times for each sequence for different QP values. The results indicate that the proposed approach outperforms JM-FC, STBMA and HO-MLD in all cases. On average, the proposed approach brings about a 2.78 dB gain over JM-FC, a 1.31 dB gain over STBMA, and a 1.51 dB gain over HO-MLD approaches. Furthermore, over all QPs, the proposed approach was able to correct the bitstream 80% of the time, while for HO-MLD, it was achieved only 6% of the time.

TABLE II
COMPARISON OF THE AVERAGE PSNR OF RECONSTRUCTED CORRUPTED FRAMES FOR DIFFERENT METHODS IN H.264 SEQUENCES. THE DIFFERENCES BETWEEN EACH METHOD AND THE JM-FC APPEAR IN PARENTHESES. THE LAST COLUMN SHOWS THE PERCENTAGE OF PACKETS THAT WERE FULLY CORRECTED BY THE PROPOSED APPROACH.

Sequence	**QP**	**Avg. PSNR of reconstructed corrupted frames**					
		Intact	JM-FC	STBMA	HO-MLD	Proposed	
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%
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%
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**	**+1.47**	**+1.27**	**+2.78**	**79%**

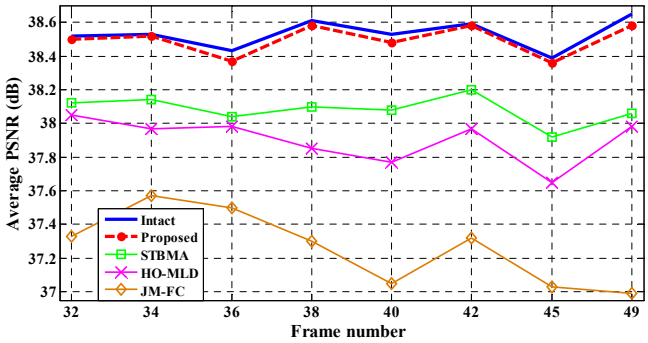

Fig. 5. Average PSNR for *Crew* sequence at QP=27 on different frames.

Fig. 5 presents the average PSNR value for the *Crew* sequence at QP=27 on different frames. Each point on the figure has an average PSNR of 100 random errors. We can clearly see that the average PSNR of the proposed approach (no matter which frame was hit by an error) is the closest PSNR value to the intact one. The results show that a significantly higher PSNR is obtained with the proposed method.

As mentioned earlier, the decoding process ends by selecting the first candidate bitstream that satisfies the two

constraints. However, it is obvious that this candidate bitstream is not always necessarily the best one, i.e., the one with a corrected bitstream. Some first valid candidate bitstreams have very low PSNR values, which negatively impacts the average PSNR values shown in Table II. Fig. 6 shows PSNR and SSIM results for the *Walk* sequence with box-and-whisker plots (showing the quartiles and variability outside the upper and lower quartiles). It is clear that most often, the proposed approach can correct the bitstream perfectly and, as a result, the PSNR and SSIM values of the reconstructed frame are exactly the same as the intact one. 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, the PSNR difference of a few decibels on the reconstructed corrupted frame increases to several dBs on subsequent frames due to this drift.

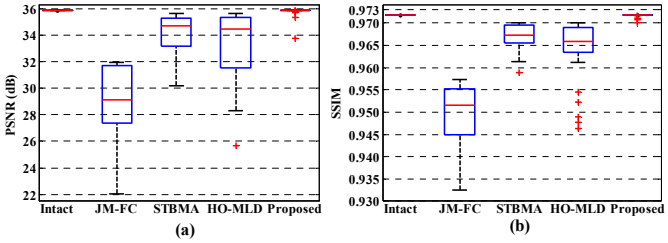

Fig. 6. PSNR and SSIM distributions of 100 runs on frame 45 of *Walk* sequence at QP 32. 80% of the time, the packet was perfectly corrected by the proposed approach.

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, as compared to existing list decoding approaches. Filtering of the candidates such as the one proposed, dramatically reduces the complexity of the list decoding approach. This approach can be applied to other list decoding methods to reduce the generated candidate list. Our simulation results revealed that the proposed method can perfectly repair the majority of damaged H.264 packets and provide, on average, a 2.78 dB gain over FC concealment using JM. The proposed approach can also be applied on HEVC coded sequences, and it can be extended to the case of more than one bit in error.

References

[1] V. Sze, M. Budagavi, and G. J. Sullivan, "High efficiency video coding (HEVC)," in *Integrated Circuit and Systems, Algorithms and Architectures*. Springer, 2014, pp. 1–375.

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] W. M. Lam, A. R. Reibman, and B. Liu, "Recovery of lost or erroneously received motion vectors," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, vol. 5, 1993, pp. 417–420.

[8] 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.

[9] 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.

[10] J. Koloda, J. Ostergaard, S. H. Jensen, V. Sanchez, and A. M. Peinado, "Sequential error concealment for video/images by sparse linear prediction," *IEEE Trans. Multimedia.*, vol. 15, no. 4, pp. 957–969, 2013.

[11] 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.*, 2011, pp. 2229–2232.

[12] 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.*, 2004, pp. 299–302.

[13] 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.

[14] 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.*, 2004, pp. 87–90.

[15] 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.*, 2006, pp. 9–13.

[16] 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.*, 2007, pp. 1517–1520.

[17] 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.*, 2010, pp. 1–4.

[18] 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.

[19] 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.*, 2012, pp. 1–6.

[20] ——, "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.*, 2013, pp. 1870–1874.

[21] ——, "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.

[22] F. Golaghazadeh, S. Coulombe, F. X. Coudoux, and P. Corlay, "Checksum-filtered list decoding applied to H.264 and H.265 video error correction," *IEEE Trans. Circuits Syst. Video Technol.(in press)*, March 2017. DOI 10.1109/TCSVT.2017.2686647.

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

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

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

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

[27] 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)*, 2011, pp. 549–553.

[28] 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, Apr. 2004.