Video Error Correction Using Soft-Output and Hard-Output Maximum Likelihood Decoding Applied to H.264 Baseline Profile

François Caron and Stéphane Coulombe

***Abstract*—Error concealment has long been identified as the last line of defense against transmission errors. Since error handling is outside the scope of video coding standards, decoders may choose to simply ignore corrupted packets or attempt to decode their content. In this paper, we present a novel joint source-channel decoding approach that can be applied to received video packets containing transmission errors. Soft-output information is combined with our novel syntax-element-level maximum likelihood decoding framework to effectively extract valid macroblocks from corrupted H.264 slices. Simulation results show that our video error correction strategy provides an average PSNR improvement neer 2 dB compared to the error concealment approach used by the H.264 reference software, as well as an average PSNR improvement of 0.8 dB compared to state-of-the-art error concealment. The proposed method is also applicable when only hard-information is available, in which case it performs better than state-of-the-art error concealment especially in high error conditions. Finally, in our simulations, the proposed method increased the decoder computational complexity by only 5% to 20%, making it applicable for real-time applications.**

***Index Terms*—video error correction, joint source-channel decoding, maximum likelihood decoding, H.264, real time video applications**

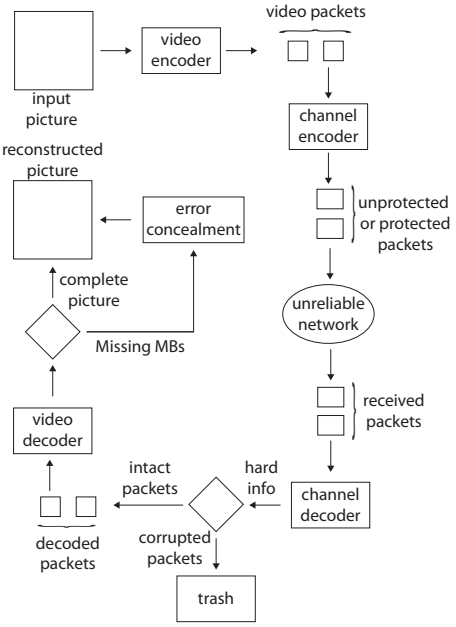


Fig. 1. Widespread architecture for video communication systems

I. Introduction

REAL-time video communication systems frequently use unreliable means to exchange information as their timing requirements exclude the deployment of some well-known error-recovery techniques [1]. For instance, video conferencing applications use UDP [2], as retransmission delays threaten their interactive nature. Broadcasting applications prevent the use of retransmission algorithms completely due to network flooding considerations [3].

Regardless of the reason, the use of unreliable delivery means transfers the error handling responsibilities to the application layer [4], [5]. The introduction of new error robustness tools in Rec. H.264 [6] clearly shows that integrating these responsibilities in the system's design is being taken seriously [4], [5], [7]. However, error handling still remains outside the scope of video coding standards [8].

Fig. 1 depicts a video communication system's architecture where it is assumed that corrupted packets are not decoded but discarded [8]. Here error resiliency/robustness is used to minimize the impact of packet loss [9], [10], and video error concealment [11], [12], [13], [14], [15], [16] to deal with missing macroblocks (MB). The fact that random bit errors can desynchronize the coded information when variable length codewords are used [1] explains its widespread adherence.

The fact that a video decoder's reaction to corrupted packets is not standardized [3], coupled with the fact that real-time video communication would benefit from exploiting damaged packets rather than discarding them [17], has given rise to a different family of decoders that attempt to decode corrupted video packets. Taking advantage of the fact that the human eyes can tolerate a certain degree of distortion in image and video signals [1], Chu [18] and Shyu [19] successfully applied error detection techniques exploiting constraints imposed on compressed images to the H.261 and MPEG-2 standards respectively. The idea was to retrieve intact information from the corrupted packets prior to applying error concealment in order to reconstruct higher quality images. More recently, Superiori [20], Farrugia [21], and Trudeau [22] have demonstrated that similar error detection approaches also work for the H.264 standard.

Joint source-channel decoding (JSCD) approaches [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35] have also shown that involving the video decoder in the correction process yields better correction capabilities.

Financial support was provided by the Natural Sciences and Engineering Research Council of Canada (Ph.D. scholarship) and the Fonds de recherche du Québec - Nature et technologies (grant #141698).

Ultimately, this leads to better visual fidelity, as JSCD reduces the number of MBs needing to be concealed when the entire corrupted slice cannot be salvaged [21].

The published JSCD approaches can be split into two families. Both show significant PSNR improvement over the traditional approach, where only intact packets reach the video decoder. How they use the channel information and the source semantics to correct transmission errors separates them.

The first family [23], [24], [25], [26], [27], [33] relies on the soft-output information provided by the channel decoder to generate and sort candidate bitstreams. Such approaches are also referred to as *list decoding*. Candidates are generated by flipping one or more bits in the received corrupted packet, without the knowledge of source semantics. Using the soft-output information (real values between $-\infty$ and ∞, where the lower bound refers to a certainty a 1 was received, the higher bound, a certainty that a 0 was received, and 0 indicates the impossibility of differentiating a 0 from a 1), a flipping cost is assigned to each bit. The total cost for flipping these bits is used to sort the candidates. Without the soft-output information, generating the solution space becomes impractical since each bit would have the same flipping cost, and therefore all the candidates having the same number of flipped bits would have the same cost. An iterative approach is then typically used to pass each candidate slice, from lowest to highest cost, in the syntax checker until a valid bitstream is generated. The first syntactically valid candidate slice becomes the winning slice, and is fed to the video decoder. If no candidate slice is syntactically valid, a possible situation if the solution space is restricted for performance considerations, the common approach is to use the candidate slice where the syntax error is detected farthest from the start.

There are three problems with such approaches. First, the fact that source semantics are ignored during the candidate generation step increases the computational complexity of the approach as syntactically invalid candidates will only be discovered by the syntax checker. Second, the definition of a *valid bitstream* requires additional constraints. The simple fact that a candidate only contains syntactically valid codewords is insufficient, as the number of extracted MBs may differ from the one sent (the number of MBs contained in a packet may be variable). Again, this increases the computational complexity of the solution, as the syntax checker will have to deal with syntactically valid candidates that do not meet the additional constraints. Finally, the first syntactically valid slice is not necessarily the one providing the best visual quality as the process only takes into account the cost of flipping bits, ignoring the resulting slice's content besides validity.

The second family [28], [29], [30], [31], [32] combines the soft-output information and the video semantics in a single step process, joint source-channel decoding (JSCD), rather than having two independent processes interacting with each other. However, most solutions focus on a single aspect of the bitstream. Nguyen [29], Weidmann [31], and Bergeron [32] propose solutions to correct transmission errors in the syntax element used to transmit the residual information (DCT coefficients). Wang [28] tackles the specific case of repairing corrupted motion vectors transmitted using Data Partitioning.

In this paper, we present a novel soft-output maximum likelihood decoding (SO-MLD) approach. Unlike *list decoding* approaches, SO-MLD does not have to decode streams from a large solution space. SO-MLD considers both the cost of flipping bits, and the likelihood of the resulting slice. Furthermore, it does not impose non-standard constraints, nor does it require specific coding tools to find a solution. Like the JSCD family of approaches, soft-output information and source semantics are combined to correct transmission errors. Finally, it targets the entire bitstream instead of focusing only on a specific aspect of the stream.

The rest of the paper is organized as follows. In Section II, video error correction is posed as an optimization problem where our slice-level solution is presented and compared to the published approaches used in [23], [28], [24], [25], [26], [27]. Our proposed solution is then derived at the syntax-element-level in Section III. Simple probability models for different H.264 Baseline profile syntax elements are then presented in Section IV. Our experimental setup is described in Section V to properly test our proposed SO-MLD approach, as well as a hard-output approach (HO-MLD) when soft-output information is not available. Our observations are then presented and discussed. Concluding remarks, as well as future work, are presented in Section VI.

II. Slice-Level Error Correction

As depicted in Fig. 1, when transmission errors are detected in received packets (here we assume that the communication protocol used to deliver the packet uses an error detection mechanism similar to UDP's checksum [17]), they are discarded. However, we believe that sending those packets to the video decoder, along with additional information such as soft-output information, is a better strategy. Although the number of errors is unknown, corrupted packets often contain information that can be exploited as shown in [18], [19], [20], [21], [22]. Fig. 2 shows the proposed system's architecture. It is similar to Fig. 1 except that corrupted packets are processed instead of being discarded. This process includes a channel decoder with the capability of providing soft-output information and a maximum likelihood video error correction system, which we describe in this paper. The proposed system also works, with a relatively small penalty in video correction performance, when only hard-output information is available (i.e. only the decoded bits are provided) as in the case of currently deployed communication systems.

As such, video error correction can be expressed as an optimization problem. Given that we know the received packet is corrupted, the objective is to modify the received information in such a way that we obtain the best balance between the cost of flipping bits (the more bits we flip, the less likely it becomes) and the credibility of the resulting content (based on the context, some syntax elements are more likely than others or impossible to receive).

Applied to the H.264 video standard [6], the optimization problem can be described as selecting the most likely candidate network abstraction layer unit (NALU) given the received corrupted one. Note that what follows applies to

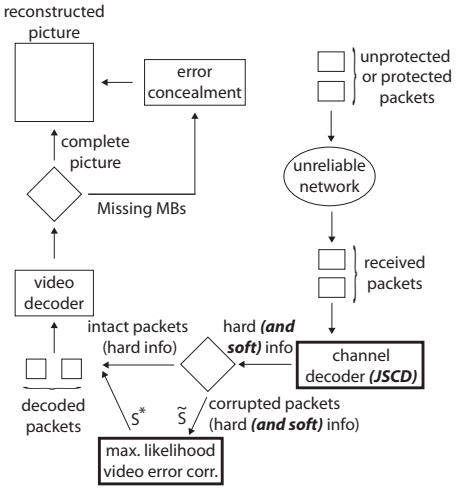


Fig. 2. Proposed architecture for video communication systems

any application that uses a sequence of codewords taken from a defined syntax. JPEG, MPEG-2, H.263, MPEG-4, and HEVC [36] all fall into this category. The problem can be posed as:

$$S^* = \arg\max_{\hat{S}\in H} \left\{ P(\mathcal{T}=\hat{S}|\mathcal{R}=\tilde{S}) \right\} \quad (1)$$

where $\mathcal{R}$ is a discrete random vector of bits representing the received NALU, $\tilde{S}$ is a realization of $\mathcal{R}$ (the actually received NALU), $\mathcal{T}$ is a discrete random vector of bits representing the transmitted NALUs, $\hat{S}$ is one possible realization of $\mathcal{T}$, H is the discrete random vector's support containing the set of all possible realizations of $\mathcal{T}$ (i.e., all possible transmitted NALUs), S^* is the likeliest realization of $\mathcal{T}$ given the received NALU $\tilde{S}$, and $P(\mathcal{T}=\hat{S}|\mathcal{R}=\tilde{S})$ is the likelihood function (probability) that $\hat{S}$ was sent, given that $\tilde{S}$ was received. Although we know that the received slice $\tilde{S}$ comes from a corrupted video packet, we do not wish to explicitly exclude it from H since the transmission error(s) may be in the packet's header (e.g. checksum), making $\tilde{S}$ a valid solution.

In this context, it seems natural that both discrete random vectors $\mathcal{R}$ and $\mathcal{T}$, and their realizations $\tilde{S}$ and $\hat{S}$, contain the same number of bits since the number of bits is known, although some are corrupted. The use of discrete random vectors conveniently integrates this consideration, as any realization has the same finite set of dimensions, defined by the number of bits present in the received NALU $\tilde{S}$. As it happens, this consideration is consistent with the list decoding and JSCD approaches previously published. In fact, H, the discrete random vector's support, contains 2^N realizations, the exact size of the solution space (here N is the number of bits in the received NALU/the number of dimensions of the discrete random vectors and their realizations) explored by list decoding approaches.

The computational complexity required to solve (1) can be reduced if we consider that H only holds valid NALUs. However, generating such a solution space is a very difficult task to tackle [31], one that requires knowledge held by the source decoder, as a valid NALU holds meaning at the application level.

Intuitively, we are aiming to find the NALU $\hat{S}$, composed of the likeliest codewords that holds the closest resemblance to the received NALU $\tilde{S}$. In other words, we are looking to make as few alterations as possible to the received bits, while improving the likeliness of the codewords. As such, the conditional distribution $P(\mathcal{T}=\hat{S}|\mathcal{R}=\tilde{S})$ in (1) can be interpreted as a compromise between the likeliness of the codewords present in $\hat{S}$, and its Hamming distance with $\tilde{S}$.

It is more convenient to rewrite (1) using Bayes' theorem. Its expanded form better illustrates our objective:

$$S^* = \arg\max_{\hat{S}\in H} \left\{ \frac{P(\mathcal{R}=\tilde{S}|\mathcal{T}=\hat{S}) \times P(\mathcal{T}=\hat{S})}{P(\mathcal{R}=\tilde{S})} \right\} \quad (2)$$

where $P(\mathcal{R}=\tilde{S}|\mathcal{T}=\hat{S})$ expresses the likelihood that the realization $\tilde{S}$ was received given that the realization $\hat{S}$ was sent, $P(\mathcal{T}=\hat{S})$ represents the probability that the realization $\hat{S}$ was sent, and $P(\mathcal{R}=\tilde{S})$, the probability that the realization $\tilde{S}$ was received. The latter probability is a constant in the maximization process, and therefore, it can be ignored. We can rewrite (2) as a maximum a posteriori (MAP) problem

$$S^* = \arg\max_{\hat{S}\in H} \left\{ P(\mathcal{R}=\tilde{S}|\mathcal{T}=\hat{S}) \times P(\mathcal{T}=\hat{S}) \right\} \quad (3)$$

As mentioned previously, our objective is to minimize the alterations to the received NALU $\tilde{S}$, while maximizing the effect they have in terms of likeliness of the codewords. In its current form, (3) can be solved using a JSCD approach. The likelihood $P(\mathcal{R}=\tilde{S}|\mathcal{T}=\hat{S})$ can be evaluated using the soft-output information provided by the channel decoder. The probability $P(\mathcal{T}=\hat{S})$ can be evaluated at the application layer by exploiting source semantics. In what follows, we first tackle the computation of the likelihood $P(\mathcal{R}=\tilde{S}|\mathcal{T}=\hat{S})$. Then, we show how to compute the probability $P(\mathcal{T}=\hat{S})$.

For the channel decoder, a NALU is nothing more than a sequence of bits. For convenience, let $\mathcal{R}_n$ and $\mathcal{T}_n$ be n-th bits of discrete random vectors $\mathcal{R}$ and $\mathcal{T}$ respectively, and let $\tilde{S}_n$ and $\hat{S}_n$ be n-th bits of the realizations $\tilde{S}$ and $\hat{S}$ respectively.

Thus, the Hamming distance can be used to evaluate $P(\mathcal{R}=\tilde{S}|\mathcal{T}=\hat{S})$, as we are only dealing with substitutions (erroneous bits). The distance represents the number of differing components between the realizations $\hat{S}$ and $\tilde{S}$. More importantly, it can be interpreted as the number of successes in N independent Bernoulli trials with independent success rates.

$$P(\mathcal{R}=\tilde{S}|\mathcal{T}=\hat{S}) = \prod_{n=1}^{N} P(\mathcal{R}_n=\tilde{S}_n|\mathcal{T}_n=\hat{S}_n) \quad (4)$$

The conditional distribution $P(\mathcal{R}_n=\tilde{S}_n|\mathcal{T}_n=\hat{S}_n)$ in (4) represents the cost associated with either keeping (bits $\tilde{S}_n$ and $\hat{S}_n$ match) or flipping (bits $\tilde{S}_n$ and $\hat{S}_n$ differ) the n-th received bit. This cost combines both the possibility of a transmission error and the individual flipping cost of the n-th bit. The latter is more apparent if we rewrite (4) using Kolmogorov's definition of conditional events.

$$P(\mathcal{R}=\tilde{S}|\mathcal{T}=\hat{S}) = \prod_{n=1}^{N} \frac{P(\mathcal{R}_n=\tilde{S}_n \cap \mathcal{T}_n=\hat{S}_n)}{P(\mathcal{T}_n=\hat{S}_n)} \quad (5)$$

where the numerator represents the probability of transmitting $\hat{S}_n$, and receiving $\tilde{S}_n$, and the denominator represents the probability that $\hat{S}_n$ was transmitted.

There are four outcomes for the numerator in (5). Two of them indicate a corrupted bit, and the two others, an intact bit. Assuming 0s and 1s are equally likely to be affected by transmission errors, let us assume that:

$$P(\mathcal{R}_n=\tilde{S}_n \cap \mathcal{T}_n=\hat{S}_n) = \begin{cases} \frac{\rho}{2} & ,\hat{S}_n \neq \tilde{S}_n \\ \frac{1-\rho}{2} & ,\hat{S}_n = \tilde{S}_n \end{cases} \quad (6)$$

where ρ is the estimated bit error rate (BER).

To evaluate the denominator in (5), the channel decoder can use the log-likelihood ratios (LLR). LLRs represent the natural logarithm of the ratio of the probability that a 1 was sent over the probability a 0 was sent, given a random source of noise [37, Annex B].

$$LLR_n = \ln\left(\frac{P(\mathcal{T}_n=1|y)}{P(\mathcal{T}_n=0|y)}\right) \quad (7)$$

where y is a random noise signal which represents channel noise (i.e., the source of bit errors), and LLR_n is the log-likelihood ratio associated with $\hat{S}_n$. For simplicity, y is assumed to be additive Gaussian noise in our simulations. From the computed LLR_n, the probability that $\hat{S}_n$ was transmitted is:

$$P(\mathcal{T}_n=\hat{S}_n|y) = \begin{cases} \frac{1}{exp(LLR_n)+1} & ,\hat{S}_n = 0 \\ \frac{exp(LLR_n)}{exp(LLR_n)+1} & ,\hat{S}_n = 1 \end{cases} \quad (8)$$

When soft-output information is not available, i.e. in the case of HO-MLD, LLR_n simply becomes a constant value. There is no need to define the value, as it can be ignored from the maximization problem at hand, and we would be left with $P(\mathcal{R}=\tilde{S}|\mathcal{T}=\hat{S}) = \prod_{n=1}^{N} P(\mathcal{R}_n=\tilde{S}_n \cap \mathcal{T}_n=\hat{S}_n)$, a simplified version of (5).

Now, we are interested in the probability $P(\mathcal{T}_n=\hat{S}_n)$, rather than in (8). Fortunately, the random noise present in (7) and (8) is a constraint in our solution. Our goal is to find the likeliest NALU, not to find the likeliest random noise that explains the received corrupted NALU. Therefore, let us assume that:

$$P(\mathcal{T}_n=\hat{S}_n) \approx P(\mathcal{T}_n=\hat{S}_n|y) \quad (9)$$

Integrating (6) and (9) into (5), we now have the means to evaluate the conversion cost associated with a candidate slice.

At the application layer, a NALU is more meaningful if it is seen as a realization of a discrete random vector of codewords (i.e., $\hat{S} = [\hat{c}_1 \quad \hat{c}_2 \quad \dots \quad \hat{c}_M]$). We can partition $\mathcal{T}$ and $\mathcal{R}$ into codewords. Let $\mathcal{T}_{c,i}$ and $\mathcal{R}_{c,i}$ be the i-th codewords of $\mathcal{T}$ and $\mathcal{R}$, respectively, and $\hat{c}_i$ be the i-th codeword of $\hat{S}$.

Inherently, with these definitions, $P(\mathcal{T}=\hat{S})$ represents the joint probability mass function of each codeword in the vector. This natural representation allows each realization $\hat{S}$ to hold a differing number of dimensions, as variable-length codewords (VLC) are widely used at the application layer. However, this is not actually a problem since we are not considering other realizations. Thus, the realizations do not have to share a common number of dimensions at this stage.

$$P(\mathcal{T}=\hat{S}) = P(\cap_{i=1}^{L_C(\hat{S})} (\mathcal{T}_{c,i}=\hat{c}_i)) \quad (10)$$

where $L_C(\cdot)$ is a function returning the number of codewords in a NALU.

Using the Chain rule, (10) can be computed with greater ease. Additionally, the sequential dependencies between the codewords are better expressed in this way.

$$P(\mathcal{T}=\hat{S}) = \prod_{i=1}^{L_C(\hat{S})} P(\mathcal{T}_{c,i}=\hat{c}_i|\Psi_i \cap_{k=1}^{i-1} (\mathcal{T}_{c,k}=\hat{c}_k)) \quad (11)$$

where Ψ_i is the set of decoded codewords in past slices, as well as the decoding variables, from past slices, and up to the i-th codeword in the current slice, such as the current MB's address. For the sake of compactness, let $\Omega_{c,i} = \Psi_i \cap_{k=1}^{i-1} (\mathcal{T}_{c,k}=\hat{c}_k)$. $\Omega_{c,i}$ can be interpreted as the set of variables which hold the context necessary to decode the i-th syntax element. Then, (11) becomes:

$$P(\mathcal{T}=\hat{S}) = \prod_{i=1}^{L_C(\hat{S})} P(\mathcal{T}_{c,i}=\hat{c}_i|\Omega_{c,i}) \quad (12)$$

Integrating (5) and (12) into (3), we now have:

$$S^* = \arg\max_{\hat{S}\in H} \left\{ \begin{array}{r} \prod_{n=1}^{N} \frac{P(\mathcal{R}_n=\tilde{S}_n \cap \mathcal{T}_n=\hat{S}_n)}{P(\mathcal{T}_n=\hat{S}_n)} \times \\ \prod_{i=1}^{L_C(\hat{S})} P(\mathcal{T}_{c,i}=\hat{c}_i|\Omega_{c,i}) \end{array} \right\} \quad (13)$$

We can further refine the notations by acknowledging that each codeword is a vector of bits and write $\hat{c}_i = \begin{bmatrix}\hat{c}_{i,1} & \hat{c}_{i,2} & \dots & \hat{c}_{i,L_B(\hat{c}_i)}\end{bmatrix}$, where $L_B(\cdot)$ is a function returning dimension of the realization $\hat{c}_i$, and $\hat{c}_{i,n}$ is its n-th vector component (i.e., n-th bit). Let $\mathcal{T}_{c,i,n}$ be the n-th bit of the i-th codeword of $\mathcal{T}$.

Having split our original discrete random vector into smaller discrete random vectors, it would seem consistent to rewrite $P(\mathcal{R}=\tilde{S}|\mathcal{T}=\hat{S})$ in (3) as a product of distances at the codeword level to take advantage of this natural representation.

$$S^* = \arg\max_{\hat{S}\in H} \left\{ \prod_{i=1}^{L_C(\hat{S})} \begin{array}{r} P(\mathcal{R}=\tilde{S}|\mathcal{T}_{c,i}=\hat{c}_i) \times \\ P(\mathcal{T}_{c,i}=\hat{c}_i|\Omega_{c,i}) \end{array} \right\} \quad (14)$$

where we have:

$$P(\mathcal{R}=\tilde{S}|\mathcal{T}_{c,i}=\hat{c}_i) = \prod_{n=1}^{L_B(\hat{c}_i)} \frac{P(\mathcal{R}_{\Delta(i)+n}=\tilde{S}_{\Delta(i)+n} \cap \mathcal{T}_{c,i,n}=\hat{c}_{i,n})}{P(\mathcal{T}_{c,i,n}=\hat{c}_{i,n})} \quad (15)$$

where $\Delta(i) = \sum_{k=1}^{i-1} L_B(\hat{c}_k)$. The sum expresses the position of the last bit belonging to the realization $\hat{c}_{i-1}$. When dealing with the first realization $\hat{c}_1$, $\Delta(1) = 0$.

With the current form of (14), we still have to evaluate every candidate slice in the solution space H, a colossal task which is far too complex in practice.

The limitations of the published *list decoding* approaches are still present here, as flipping a bit at the channel layer, creating a different (yet valid) realization in H, does not guarantee that the realization $\hat{S}$ will be valid at the application level.

III. Syntax-Element-Level Error Correction

We propose to solve the problem at the syntax-element-level. This is a more intuitive approach, which would also aid the decoding process. Codewords are sequentially decoded, and the valid realizations $\hat{c}_i$ at each step of the decoding process are known (making error detection possible). Refining the problem at the syntax-element-level solves two important problems. Firstly, we no longer need to generate H, only the proper set of valid syntax elements at each stage of the decoding process. Secondly, we no longer need to deal with realizations $\hat{S}$ that are valid at the channel level, but invalid at the application layer.

Fortunately, (14) can be modified to meet our goal. At each step of the decoding process, we want to select the optimal codeword as a function of previously decoded codewords:

$$c_i^* = \underset{\hat{c}_i \in C_i}{\arg\max} \left\{ P(\mathcal{R}{=}\tilde{S}|\mathcal{T}_{c,i} = \hat{c}_i) \times P(\mathcal{T}_{c,i} = \hat{c}_i|\Omega_{c,i}) \right\} \tag{16}$$

where c_i^* is the likeliest realization in the solution space C_i (the set of valid codewords for i-th decoded syntax-element). We start decoding with $i = 1$ and increment i as we decode the next codeword. Note that the likeliest sequence of codewords (14) (optimal solution), and the sequence of likeliest codewords (16) (greedy approach) are not necessarily the same. Therefore, it is expected that this syntax-element-level approach will find a different solution. Moreover, this approach accepts the fact that not all of the corrupted NALU's content may be recovered, as the decisions taken may lead to a step where the solution space C_i is empty. If such a case were to occur, the missing MBs would have to be concealed.

A slight modification to (16) is required to address VLCs. The initial optimization problem deals with discrete random vectors of equal lengths. Thus, $P(\mathcal{R} = \tilde{S}|\mathcal{T} = \hat{S})$ always considers the same number of independent Bernoulli trials. This is no longer the case, as C_i does not represent a discrete random vector's support. The realizations it contains do not have to share a common dimension.

As a result, a weighting factor needs to be added to even out the number of independent trials considered for each candidate. Without this factor, the maximization process would favor shorter realizations because they have smaller Hamming distances. This in fact penalizes longer codewords, codewords that are already expected to be less frequently used than shorter ones. Weidmann [31] uses the so-called random tail assumption to align sequences of different lengths. We will do the same here (looking at (15)).

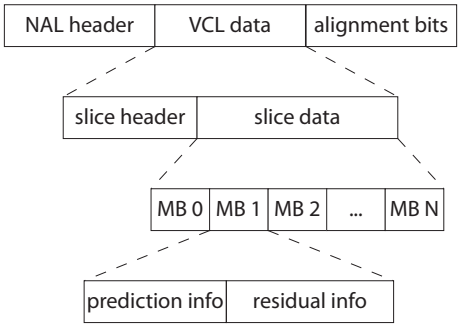


Fig. 3. Hierarchical structure of VCL units

$$c_i^* = \underset{\hat{c}_i \in C_i}{\arg\max} \left\{ \begin{array}{l} P(\mathcal{R}{=}\tilde{S}|\mathcal{T}_{c,i} = \hat{c}_i){\times} \\ \quad \frac{1}{2}^{\beta_i - L_B(\hat{c}_i)} \times P(\mathcal{T}_{c,i} = \hat{c}_i|\Omega_{c,i}) \end{array} \right\} \tag{17}$$

where β_i is the length of longest codeword in C_i.

With this optimization problem, we can now select the likeliest codeword in C_i at each step in the decoding process, starting with c_1^*, and moving towards c_n^*. The likeliest codewords selected will affect the video decoder the same way a decoded codeword from an intact NALU would have, as our proposed solution does not affect the rest of the decoding process.

Exploiting the source semantics, a solution can now be tailored for each syntax element. It is also worth mentioning that the approach is independent of the video standard. We work with NALUs, introduced in the H.264 standard, but any term representing a sequence of codewords could have been used.

IV. Modeling syntax elements

The mathematical framework derived in the previous section explains how the likeliest codeword will be selected. The task at hand is to identify which information to use to evaluate $P(\mathcal{T}_{c,i} = \hat{c}_i|\Omega_{c,i})$ in (17) for each syntax element in the bitstream. In this section, we focus on H.264 Baseline Profile syntax elements, but the approach would also work for other profiles, previous standards (H.263, MPEG-2, etc.), as well as the new HEVC standard. The models presented here adequately fit our observations, but more precise models could improve the correction performances of the proposed approach. However, such models are beyond the scope of this paper. Moreover, all the variables used in the models are dynamically acquired during the decoding process using the previously processed slices, both intact and repaired.

Figure 3 illustrates the hierarchy of a NALU carrying video coding layer (VCL) information. Models for the slice header's syntax elements have already been proposed in [34]. We quickly present them here. Then, we concentrate on modeling the syntax elements *mb_type*, *mb_skip_run*, *prev_intra4x4_pred_mode_flag*, *rem_intra4x4_pred_mode*, *intra_chroma_pred_mode*, *sub_mb_type*, *mvd_l0*, and *coded_block_pattern*. These syntax elements are used to describe the prediction scheme used to reduce spatio-temporal redundancies, and indicate the presence or absence of residual

information. The syntax elements used to communicate the residual information are left as future work.

In this discussion, we assume that non-VCL NAL units (i.e., Sequence Parameter Sets, Picture Parameter Sets, etc.) are sent using reliable means, and that fragmentation units [38] are not present. We also assume that the corrupted slices were coded using the Baseline Profile, that the four constraint flags in the active Sequence Parameter Set are set to 0, and that the QP is constant for the entire frame. These constraints are imposed as a way to limit the scope of the paper, not our method. Moreover, the proposed models are purposely kept rudimentary to show the potential of our new framework. More sophisticated models could be proposed to achieve even better results.

A. Slice Header

We propose models for the first five fields of the slice header. We assume that the remaining fields use constant values throughout the sequence, simplifying the correction process. The fields are presented following their order in a coded slice.

1) first_mb_in_slice*:* The *first_mb_in_slice* syntax element represents the raster scan index of the first coded MB carried in the NALU. Under our current assumptions, the number of MBs carried in a NALU can be expressed as the difference between the values used in consecutive NALUs associated with the same picture. Since we know that transmission errors can affect the number of MBs extracted from a corrupted NALU, let us assume that the number of MBs carried in a NALU follows a Normal distribution.

$$P(FMIS = \hat{c}_i|\Omega_{c,i}) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{(\hat{c}_i-(fmis_{Prev}+\mu))^2}{2\sigma^2}\right) \tag{18}$$

where $FMIS$ represents a discrete random variable representing the starting address of a NALU, $fmis_{Prev}$ is the previous observed value of *first_mb_in_slice*, μ is the observed average number of MBs carried in a NALU, and σ, the observed standard deviation. Both μ and σ should consider the coding type of the previously received NALU, as intra- and inter-coding are not expected to fit the same number of MBs in one NALU. Both μ and σ are obtained by studying the differences between the *first_mb_in_slice* values in consecutive slices.

It is worth mentioning that the last NALU associated with a picture should not be used to evaluate μ and σ. Limiting the maximum transmission unit (MTU) size of a packet introduces the possibility that the last NALU of a picture contains significantly fewer MBs than the other ones since MBs associated with different pictures cannot be transported together.

The codebook C_i should only contain the remaining undecoded addresses in the picture, starting with the last received value of *first_mb_in_slice* plus 1. This follows the logic that each MB is coded once. However, by limiting C_i, the probabilities will not sum to 1. Imposing the address 0 in C_i, the only valid address for the next picture, solves this problem elegantly. Its presence acknowledges the possibility that the next NALU starts a new picture. $P(FMIS = 0|\Omega_{c,i})$ represents the complement of the summed probabilities. As NALUs are decoded, the probability that the next NALU starts a new picture increases, which is exactly what happens with the above definition.

A special case exists if the previously received NALU was intact. Assuming that all NALUs are received in the correct order, the address following the last one used is expected. That value can be used as if a constant was expected instead of using (18).

2) slice_type*:* The *slice_type* syntax element indicates the coding type employed in the current NALU. The Baseline Profile restricts the outcomes to inter-coding (0 or 5), and intra-coding (2 and 7). The value used here has an influence on the allowed coding types the MBs can use. Additionally, values above 4, corresponding to the higher range, are used to indicate that all the slices associated with the current picture share the same coding type. Thus, the value of the *first_mb_in_slice* must be considered, since the effect of using a *slice_type* value above 4 is limited by the picture boundaries. We assume that the encoder does not mix values from the lower and higher ranges within a picture, because if it does, the only NALU using information in the higher range could be lost during transmission.

We can model the *slice_type* syntax as two pairwise independent Bernoulli trials. The first experiment checks for the range used, where a value in the range above 4 indicates success. The second experiment checks for the coding type, where the use of intra-coding indicates success. This can be expressed as:

$$P(ST = \hat{c}_i|\Omega_{c,i}) = \begin{cases} \alpha[\delta(\hat{c}_i-5)(1-\beta)+\delta(\hat{c}_i-7)\beta]+ & \\ (1-\alpha)[\delta(\hat{c}_i)(1-\beta)+\delta(\hat{c}_i-2)\beta] & , fmis = 0 \\ \delta(\hat{c}_i)(1-\beta)+\delta(\hat{c}_i-2)\beta & , fmis \neq 0, st_{Prev} \leq 4 \\ \delta(\hat{c}_i - st_{Prev}) & , fmis \neq 0, st_{Prev} > 4 \end{cases} \tag{19}$$

where ST is a discrete random variable representing the coding scheme used in a NALU, $fmis$ is the latest decoded value of *first_mb_in_slice*, st_{Prev} is the *slice_type* value decoded in the previous NALU, $\delta(\cdot)$ is the discrete Dirac function, α represents the probability that a *slice_type* value above 4 is used, and β represents the probability that the *slice_type* uses intra-coding. Both probabilities are estimated from the previously reconstructed *slice_type* values.

3) pic_parameter_set_id*:* The *pic_parameter_set_id* syntax element indicates which PPS is to be used during the decoding process of the current NALU. We assume that the coded streams always refer to the same PPS, and thus the value is a constant.

4) frame_num *and* pic_order_cnt_lsb*:* The *frame_num* and *pic_order_cnt_lsb* syntax elements are used to identify pictures. They both represent the least significant bits of monotonically increasing sequences, where the use of a new value is triggered when the value of *first_mb_in_slice* equals 0 (the start of a new picture). Subclause 7.4.3 [6] specifically

indicates that all NALUs belonging to the same picture shall use the same values of *frame_num* and *pic_order_cnt_lsb*. The only difference is that *pic_order_cnt_lsb* typically uses an increment of 2 instead of 1. Assuming that slices may be damaged, but never lost, this behavior indicates that we only need to consider two outcomes: either the same values present in the previous NALU are used, or the least significant bits of the next value in the monotonically increasing sequence are used. The syntax elements cannot be modeled as a single field because they are not necessarily contiguous.

$$P(FN = \hat{c}_i|\Omega_{c,i}) = \begin{cases} \delta(\hat{c}_i - fn_{Prev}) & , fmis \neq 0 \\ \delta(\hat{c}_i - lsb(fn_{Prev}+1)) & , fmis = 0 \end{cases} \quad (20)$$

where FN is a discrete random variable representing the *frame_num* value used by a NALU, fn_{Prev} is the *frame_num* value decoded in the previous NALU, and $lsb(\cdot)$ is the modulo operator using the SPS value *log2_max_frame_num_minus4*.

$$P(POCL = \hat{c}_i|\Omega_{c,i}) = \begin{cases} \delta(\hat{c}_i - pocl_{Prev}) & , fmis \neq 0 \\ \delta(\hat{c}_i - lsb(pocl_{Prev}+2)) & , fmis = 0 \end{cases} \quad (21)$$

where $POCL$ is a discrete random variable representing the *pic_order_cnt_lsb* value used by a NALU, $pocl_{Prev}$ is the *pic_order_cnt_lsb* value decoded in the previous NALU. In this case, the modulo operator uses the SPS value *log2_max_-pic_order_cnt_lsb_minus4*.

B. Slice Data

The syntax elements used to communicate the prediction mechanisms used have all been modeled, with the exception of the *mb_qp_delta* syntax element. Our assumption that the QP is fixed for the entire sequence makes this value constant. However, we do not take advantage of this assumption to use the syntax element as a synchronization marker (to show that the proposed method works despite not using this information).

1) mb_type*:* There are 26 intra-, and six inter-coding schemes available. 31 of those 32 schemes are signaled with the *mb_type* syntax element. The P_Skip coding scheme is inferred by the use of the *mb_skip_run* syntax element, when it is present.

Inter-coding schemes can only be used in NALUs whose *slice_type* syntax element uses inter-coding. Intra-coding schemes can be used in both intra and inter NALUs. Moreover, the MBs's location in the current NALU has an influence on the available intra 16x16 coding schemes. These coding schemes use predetermined luminance prediction modes (see Table 7 - 11 in Rec. H.264 [6]) that may be unavailable given the MB's location and the current slice's boundaries.

In addition to the addresses of the NALU *slice_type* and of the MB, we can also consider the coding schemes used by the neighboring spatio-temporal MBs. To keep the model simple, we consider the coding scheme used by the colocated MB in the previous frame.

$$P(CS = \hat{c}_i|\Omega_{c,i}) = P(CS = \hat{c}_i|st) \times P(CS = \hat{c}_i|addr) \times P(CS = \hat{c}_i|cs_{Co}) \quad (22)$$

where CS is a discrete random variable representing the coding scheme used, st is the latest decoded value of *slice_type*, st_{prev} has the same meaning as in (19), $addr$ indicates the address of the current MB, and cs_{Co} indicates the coding scheme used by the colocated MB in the previous frame.

Note that the coding scheme is used rather than the *mb_type* syntax element. This is because the value of *mb_type* for the colocated MB may not be available if *mb_skip_run* was used. This general definition will still lead to the selection of a valid *mb_type* value, as the codebook C_i will be populated as such.

2) mb_skip_run*:* The *mb_skip_run* syntax element is only present when the current slice uses inter prediction. It indicates the number of skipped MBs before the next coded one, or the end of the slice/frame. Since a MB is either coded or skipped, let us use these as the two outcomes of a Bernoulli trial. Then, the syntax element can be modeled as a Geometric distribution, where a success is defined as a coded MB, or the end of the slice/frame. As was the case with the *mb_type* syntax element, the probability of success is expected to vary with the MB's location in the frame. This is expressed through independent success rates.

$$P(\text{mb\_skip\_run} = \hat{c}_i|\Omega_{c,i}) = \prod_{n=0}^{\hat{c}_i-1} P(CS = \text{P\_Skip}|\Omega_{c,i} \cap ADDR = addr + n) \times (1 - P(CS = \text{P\_Skip}|\Omega_{c,i} \cap ADDR = addr + \hat{c}_i)) \quad (23)$$

where each term in the product, as well as the final term (the one outside the product) is evaluated using (22). However, the address at which the conditional probability will be evaluated is specified by the discrete random variable $ADDR$. Here, $\hat{c}_i$ is the number of skipped MBs before the next coded MB, or the end of the slice. The codebook C_i should be populated only with the values leading to the last valid address. This also requires that the probability of using any coding scheme at an address outside the frame equals 0. This last consideration ensures numerical stability. When the last entry in C_i is evaluated, the final term in (23) refers to the MB following the last valid one. That probability would thus be 1, keeping only the product of the probability that all the remaining MBs are skipped.

3) Intra4x4PredMode: Intra-coded MBs, with the exception of I_PCM MBs, use spatial prediction on both luminance and chrominance values to remove redundancies. While the luma prediction modes are built into the intra 16x16 coding schemes, intra 4x4 MBs use one of nine prediction modes for each of the 4x4 blocks it contains.

The mode selection is signaled using two syntax elements: *prev_intra4x4_pred_mode_flag* and *rem_intra4x4_pred_mode*. The former indicates whether the mode the decoder derived matches or differs from the one used. The latter, present when

prev_intra4x4_pred_mode_flag equals 0, indicates which of the eight remaining modes to use.

The MB's current location plays the same role here as it did with the *mb_type* syntax element, limiting the available modes when the required neighboring information is unavailable.

Modeling both fixed-length syntax elements as a VLC makes the correction process more accurate. First, it integrates *rem_intra4x4_pred_mode*'s conditional presence into the correction process. Second, it takes advantage of the derivation process built into the decoder [6, Section 8.3.1.1]. This results in a richer syntax element. Furthermore, the probability distribution is better tailored to our needs. We will use Intra4x4PredMode (IPM), the standard variable used to explain the derivation process, to represent the joint distribution of the syntax elements.

$$P(IPM = \hat{c}_i|\Omega_{c,i}) = P(IPM = \hat{c}_i|addr)\times \\ P(IPM = \hat{c}_i|ipm_W) \times P(IPM = \hat{c}_i|ipm_N) \quad (24)$$

where ipm_W and ipm_N are the left and above Intra4x4PredMode values used in the derivation process for the current 4x4 block. The conditional distributions used in (24) are in fact pmfs built from the observed prediction modes in the previously decoded intact slices; one tracking the correlation with the MB's left neighbor, and the other tracking the correlation with the MB's top neighbor.

4) intra_chroma_pred_mode*:* The *intra_chroma_pred_mode* syntax element indicates which of the four prediction modes is to be used. The MB's location also affects the available types.

The syntax element can be corrected using the product of two conditional distributions; one tracking the correlation with the MB's left neighbor, and the other tracking the correlation with the MB's top neighbor.

$$P(ICPM = \hat{c}_i|\Omega_{c,i}) = P(ICPM = \hat{c}_i|addr)\times \\ P(ICPM = \hat{c}_i|icpm_W) \times P(ICPM = \hat{c}_i|icpm_N) \quad (25)$$

where $ICPM$ is a discrete random variable representing the *intra_chroma_pred_mode* used, and $icpm_W$ and $icpm_N$ are the chrominance prediction modes used by the west and north MBs, respectively. The conditional distributions used in (25) are also custom built mass functions from the observed decoded values in previous intact NALUs.

5) sub_mb_type*:* Motion compensation, used to reduce temporal redundancies, is described using motion vectors. To improve coding efficiency, the H.264 standard introduced a more sophisticated motion compensation scheme compared to earlier standards. The latest value of *mb_type* indicates how many motion vectors are associated with the current MB. Additionally, when the coding scheme is either P_8x8 or P_8x8ref0, the syntax element *sub_mb_type* will be present to further segment the MB's displacement.

sub_mb_type elements always come in groups of four consecutive values. This can be exploited to build a more meaningful syntax element. Each value of *sub_mb_type* tells the decoder how many motion vectors to extract. The total number of motion vectors provides a convenient means to evaluate the probability of the joint distribution; convenient because we can observe the number of motion vectors used at any MB for any coding scheme.

In spite of the fact counting motion vectors is computationally light, the joint distribution is less accurate with respect to the vector count of each 8x8 block. To address this consideration, we assume that all possible outcomes leading to the same number of motion vectors follow a Uniform distribution. The conversion cost evaluated by the channel decoder will help decide which codeword is the likeliest.

$$P(MVC = \hat{c}_i|\Omega_{c,i}) = \frac{1}{N_i} \times P(MVC = \hat{c}_i|mvc_{Co}) \quad (26)$$

where MVC is a discrete random variable used to represent the number of motion vectors associated with an MB, N_i is the number of combinations associated with a given number of motion vectors per MB (for instance, there are 8 combinations where $MVC = 5$, and only 1 combination where $MVC = 16$), and mvc_{Co} represents the number of motion vectors used by the colocated MB in the previous frame. The conditional distribution is built using the observed values decoded in intact NALUs.

6) mvd_l0*:* The *mvd_l0* syntax element represents the difference, both horizontal and vertical, between the selected motion vector during motion estimation, and that predicted from the available neighboring ones. The correction process can take advantage of the fact they come in pairs, and model them as a single codeword.

The correction process can also exploit the derivation process [6, Section 8.4.1] to build a more precise model. The *mvd_l0* values are added to the predicted values to obtain a final motion vector. To ensure that each slice is independently decodable, the prediction process is limited by the slice boundaries. At the start of each slice, the prediction motion vector will be (0,0), as no information will be available. In high motion sequences, we expect that this will lead to high prediction errors. As the decoding of the slice progresses, the derivation process should yield better predictors. Thus, the slice boundaries make it difficult to model the residual displacement.

Unlike the derivation process, the proposed correction process can use the available spatio-temporal motion vectors to build a motion vector field (MVF). The correction process can then use the MVF to build conditional distributions with the available spatio-temporal vectors. The optimal number of distributions used depends on the sequence and the coding parameters. We propose to use the product of five distributions, as it works well in most cases.

$$P(MV = (\hat{c}_i, \hat{c}_{i+1})|\Omega_{c,i}) = \\ \prod_{n\in\{W,N,C,E,S\}} P(MV = (\hat{c}_i, \hat{c}_{i+1})|mv_n) \quad (27)$$

where mv_W and mv_N refer to the west and north motion vectors in the current frame, and mv_C, mv_E and mv_S, to the colocated, east and south motion vectors in the previous frame, all with respect to the current location.

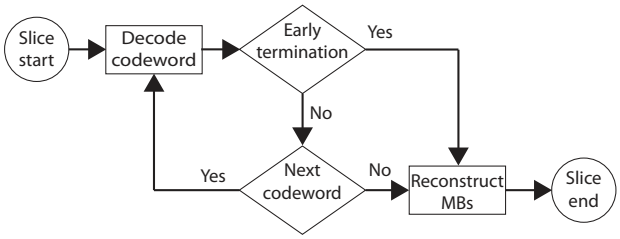


Fig. 4. Proposed procedure to decode a corrupted slice

The conditional distributions can all be evaluated using a joint Normal distribution centered at mv_n with $n \in \{W, N, C, E, S\}$. The standard deviation can be tracked using the previously decoded motion vectors in intact slices.

7) coded_block_pattern*:* The *coded_block_pattern* syntax element communicates the presence or absence of residual coefficients, and is used by both inter and intra 4x4 MBs. The value is also built into the intra 16x16 coding schemes, but they use less precise masks. The syntax element is a reordered exponential Golomb code. The ordering differs for inter and intra 4x4 MBs, but the mapped values hold the same meaning.

There are 48 values, meant to be interpreted as a 6-bit binary mask. This makes it difficult to track occurrences alone, as a value of 8 and a value of 4 both mean that a single 8x8 block contains residual information.

We can exploit the CAVLC decoding process here to help us build a model. The VLC table is selected using the *coeff_token* values of the left and top 4x4 blocks. This tells us that there is a strong correlation between neighboring blocks. We can mimic this behavior and use two conditional distributions, one tracking the above occurrences, and the other tracking the western occurrences.

$$P(CBP = \hat{c}_i|\Omega_{c,i}) = P(CBP = \hat{c}_i|cs)\times \\ P(CBP = \hat{c}_i|cbp_W) \times P(CBP = \hat{c}_i|cbp_N) \quad (28)$$

where CBP is a discrete random variable representing the *coded_block_pattern* value of an MB, cs is the latest decoded value of *mb_type*, and cbp_W and cbp_N refer to the *coded_block_pattern* values west and north of the current MB, when they are available.

C. Early Termination

Errors in the residual information leading to a syntactically valid sequence of codewords are problematic, since the prediction information is interlaced with the residual information as illustrated in Fig. 3. Upon desynchronization, the correction process will force the use of a valid codeword in any modeled syntax element, no matter how unlikely. To address this problem, we use a threshold to stop the correction process when the selected codeword seems too unlikely. A flowchart presenting our proposed decoding approach is given in Fig. 4.

Exploiting the fact that most codewords are short, it becomes highly improbable that multiple bits inside the same codeword are erroneous, even at very high bit error rates, as shown in Fig. 5. Thus, we propose to stop the correction process when the Hamming distance between the likeliest codeword and the received bits is greater than 1, as there is less than 1% chance of there being multiple errors in codewords that are shorter than 15 bits at a BER of 10^{-2}.

Furthermore, our implementation uses a greedy approach to find the series of likeliest codewords. A single codeword is retained at each step, and the remaining outcomes are discarded. Type I errors (i.e., changing intact bits) and type II errors (i.e., keeping corrupted bits) may desynchronize the bitstream, leading to an unlikely path requiring that multiple bits be flipped. To avoid following such a path, we stop the correction process when the ratio of flipped bits over interpreted bits exceeds 10^{-2}. This value has been arbitrarily set. Increasing it would make the decoder more tolerant to errors introduced by the correction process.

Once the decoding process stops, either because the slice was entirely decoded, an error was detected, or a threshold was reached, the extracted MBs are reconstructed and the missing ones, if any, are concealed.

V. Experimental Results

Several simulations were conducted under various coding and channel conditions to demonstrate the performance of our SO-MLD approach with H.264. As a comparison basis, the same corrupted sequences were also decoded using our decoder equipped with the default error concealment algorithm [8] used by the reference software JM 16.0 (the error concealment mechanism is broken in more recent versions [41]), using our decoder equipped with state-of-the-art error concealment STBMA+PDE [12] which, to our knowledge, is the best error concealment mechanism. Moreover, to show that our proposed approach performs well even when soft-outputs are not available, the corrupted streams were also decoded using HO-MLD.

The first 120 frames of the NTSC (720x480) sequences *driving*, *opening-ceremony*, *whale-show*, 4CIF (704x576) sequences *city*, *crew*, *harbour*, *ice*, *soccer*, and PAL (720x576) sequence *walk* were coded using the JM 18.2 [39].

The Baseline profile was used, and the Intra refresh rate was set at 30 frames. The MBs were coded following the raster scan order, and arbitrary slice ordering was not used. Following the recommendations in [40], the maximum transmission unit (MTU) size was set to 200 bytes. Each sequence was coded with a QP value of 22, 27, 32, and 37.

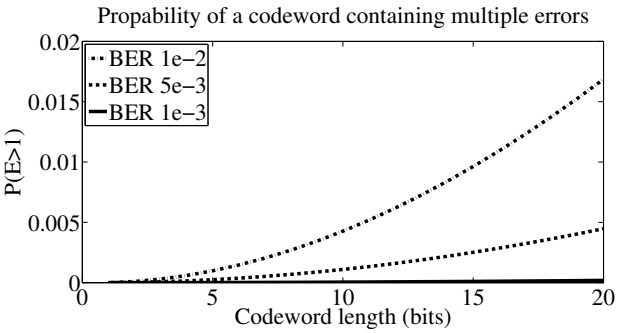


Fig. 5. Probability of having multiple errors in a single codeword

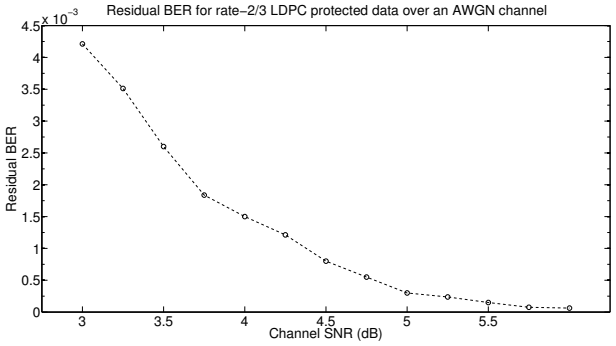


Fig. 6. Residual BER for packets protected with a rate-2/3 LDPC code and submitted to an AWGN channel.

As an experiment, a single frame between the 61^{st} and the 110^{th} was randomly selected using a Uniform distribution. The remaining frames all contain intact slices. This is to observe the performance of our method on a single frame rather than successive corrupted frames. The first 60 frames were purposely kept intact so that our decoder can gather information such as the frequency of I and P slices, the average number of MBs carried in each slice, the average difference between the predicted motion vector and the actual motion vector, etc. The transmission of the slices belonging to the randomly selected frame was simulated using a rate-2/3 low-density parity-check (LDPC) code, binary phase shift keying (BPSK) modulation, and an additive white Gaussian noise (AWGN) channel. The latter two were used, as was the case for [25], [26], [27], [28], [30], for their simplicity. The transmission simulation was repeated 30 times for each sequence, to ensure that the location of the erroneous bits did not bias our conclusions.

The actual choice of the coding rate is outside the scope of the experiment. Our interest lay in the bit error rate (BER) after the packet was decoded by the channel decoder (the LDPC codes are used to correct as many transmission errors as possible). Fig. 6 shows a plot mapping the channel conditions to the BERs under our experimental setup. The values in the graph indicate the residual error rate (i.e., remaining errors after the LDPC codes have been used to correct transmission errors). The experiment used channel SNRs of 4 dB (10^{-3} BER), and 5 dB (10^{-6} BER) to simulate both harsh and moderate transmission conditions. With the selected MTU, this translates to packet loss rates (PLR) of 80% for harsh conditions, and 5% under moderate conditions.

The experiment was conducted 10 times for each sequence at each channel SNR. In our transmission simulations, we assume that the communication protocol used is equiped with error detection. Thus, our experiment assumes that the corrupted slices are identified prior to their decoding. Alternatively, error detection methods such as the ones proposed in [20] or [21] could be used. Using this information, we created complementary streams, in which the corrupted slices were discarded, since this is what the reference software expects [8]. These slices were used to obtain the results with the JM 16.0 as well as with STBMA+PDE.

The streams containing corrupted slices were first decoded using our H.264 decoder. SO-MLD and HO-MLD were applied to the corrupted slices. Then, the complementary streams carrying only intact slices were decoded with the missing MBs concealed with the JM 16.0 and STBMA+PDE. The missing MBs, following the application of SO-MLD and HO-MLD were also concealed using STBMA+PDE.

The average PSNR of the frames affected by transmission errors are presented in Tables I, II, III, and IV. The results indicate that SO-MLD outperforms JM 16.0 in all cases, with the exception of the *opening-ceremony* sequence. Moreover, the results also show that on average, SO-MLD and HO-MLD perform better than STBMA+PDE for the majority of the sequences studied. On average, STBMA+PDE's PSNR gain is 1.15 dB over JM 16.0, while SO-MLD's gain is 1.82 dB, and H0-MLD's, 1.43 dB at a channel SNR of 4 dB. The 0.67 dB increase is an indication that exploiting corrupted slices yields better results. The results also indicate that the gap between STBMA+PDE and SO-MLD/HO-MLD closes with improved channel conditions. On average, SO-MLD yields 0.33 dB better PSNR than STMA+PDE at a channel SNR or 5 dB, while HO-MLD performs similarly. This was to be expected, as a reduction in the number of corrupted slices decreases the amount of intact MBs that can be extracted to provide better PSNR than concealing them. Inversely, as the channel conditions worsen, error correction actually helps error concealment perform better as it reduces the size of the concealment region and provides error concealment with better/more information which result in better quality concealed MBs.

A visual inspection of the results obtained further confirms that SO-MLD should be integrated into real-time video communication systems. Fig. 7 shows the original 99-th frame of the *ice* sequence coded at 1.2 Mbps, and transmitted with a channel SNR of 5 dB. The luminance differences between the frames produced with the JM 16.0, STBMA+PDE, and SO-MLD approaches are also presented to highlight the problem areas. Fig. 8 shows another example in which SO-MLD performs better than JM 16.0 and STBMA+PDE. The 65-th frame of the *walk* sequence is presented, along with the luminance differences between the intact frame and the reconstructed frames. In both cases, fewer differences are introduced when using SO-MLD. This is important not only for the current frame, but for the subsequent ones, as fewer visible drifting effects will be seen.

The simulations were conducted on an iMac equipped with a

TABLE V
AVERAGE DECODING TIME PER STANDARD DEFINITION FRAME MEASURED IN MILLISECONDS USING THE FOUR DECODING APPROACHES ON ALL TEST SEQUENCES

Channel SNR	QP	JM 16.0	STBMA+PDE	SO-MLD	HO-MLD
4 dB	22	27	28	33	32
	27	21	22	24	23
	32	17	18	22	20
	37	13	14	16	15
5 dB	22	27	28	29	28
	27	21	21	22	22
	32	17	18	18	18
	37	14	15	15	15

TABLE I
COMPARISON OF THE AVERAGE PSNR (DB) OBSERVED WITH THE FOUR DECODING APPROACHES FOR STREAMS CODED WITH A FIXED QP OF 37

Sequences	Channel SNR								
		4 dB				5 dB			
	Intact	JM16.0	STBMA + PDE	SO-MLD	HO-MLD	JM16.0	STBMA + PDE	SO-MLD	HO-MLD
driving	30.15	20.44	21.69 (+1.25)	22.80 (+2.36)	22.06 (+1.63)	23.60	26.44 (+2.84)	26.52 (+2.93)	25.59 (+1.99)
opening-ceremony	27.76	26.34	26.18 (-0.16)	26.14 (-0.20)	26.14 (-0.20)	27.12	27.02 (-0.10)	26.96 (-0.17)	26.91 (-0.21)
whale-show	28.22	23.20	23.24 (+0.03)	23.97 (+0.76)	23.64 (+0.43)	25.39	25.50 (+0.11)	26.07 (+0.68)	25.77 (+0.38)
city	29.89	24.96	25.43 (+0.45)	26.44 (+1.46)	26.41 (+1.43)	27.38	28.68 (+1.29)	28.61 (+1.22)	28.60 (+1.22)
crew	33.16	25.42	25.52 (+0.10)	26.40 (+0.98)	26.01 (+0.59)	28.52	29.25 (+0.73)	29.76 (+1.23)	29.42 (+0.90)
harbour	30.09	24.60	25.27 (+0.67)	25.47 (+0.87)	25.47 (+0.86)	26.25	27.63 (+1.38)	27.33 (+1.08)	27.29 (+1.04)
ice	36.17	24.77	24.94 (+0.17)	25.81 (+1.04)	25.32 (+0.55)	28.47	29.39 (+0.92)	30.16 (+1.69)	29.80 (+1.33)
soccer	31.30	21.23	21.84 (+0.61)	22.84 (+1.61)	22.40 (+1.17)	24.73	25.64 (+0.91)	26.47 (+1.74)	25.94 (+1.20)
walk	31.79	18.51	20.64 (+2.13)	21.43 (+2.91)	20.06 (+1.55)	20.79	23.33 (+2.54)	24.77 (+3.98)	24.01 (+3.22)
average	30.94	23.28	23.82 (+0.53)	24.59 (+1.30)	24.17 (+0.88)	25.81	26.99 (+1.18)	27.41 (+1.6)	27.04 (+1.23)

TABLE II
COMPARISON OF THE AVERAGE PSNR (DB) OBSERVED WITH THE FOUR DECODING APPROACHES FOR STREAMS CODED WITH A FIXED QP OF 32

Sequences	Channel SNR								
		4 dB				5 dB			
	Intact	JM16.0	STBMA + PDE	SO-MLD	HO-MLD	JM16.0	STBMA + PDE	SO-MLD	HO-MLD
driving	33.43	20.47	21.95 (+1.47)	22.77 (+2.30)	22.21 (+1.73)	23.89	26.32 (+2.43)	27.21 (+3.31)	26.39 (+2.50)
opening-ceremony	31.46	28.50	28.28 (-0.23)	28.56 (+0.06)	28.18 (-0.32)	29.93	30.13 (+0.20)	30.27 (+0.33)	30.17 (+0.24)
whale-show	31.98	23.69	23.82 (+0.13)	24.65 (+0.95)	24.39 (+0.70)	26.75	27.01 (+0.26)	27.80 (+1.05)	27.52 (+0.77)
city	32.99	25.14	26.79 (+1.65)	27.37 (+2.23)	27.41 (+2.27)	27.73	30.56 (+2.83)	30.00 (+2.27)	30.18 (+2.45)
crew	35.80	26.17	26.59 (+0.42)	27.95 (+1.78)	27.28 (+1.11)	29.10	30.45 (+1.35)	30.96 (+1.85)	30.67 (+1.57)
harbour	33.58	24.02	25.96 (+1.95)	25.41 (+1.39)	25.05 (+1.13)	27.12	29.53 (+2.42)	29.17 (+2.05)	28.51 (+1.39)
ice	38.76	25.02	25.51 (+0.49)	26.77 (+1.75)	26.27 (+1.25)	28.57	30.49 (+1.92)	31.33 (+2.77)	31.01 (+2.44)
soccer	34.27	19.87	21.68 (+1.81)	21.71 (+1.84)	21.47 (+1.60)	25.50	27.63 (+2.13)	28.40 (+2.90)	27.92 (+2.42)
walk	35.21	18.27	20.12 (+1.85)	21.18 (+2.91)	19.76 (+1.49)	22.13	25.57 (+3.44)	26.09 (+3.96)	25.61 (+3.48)
average	34.16	23.46	24.52 (+1.06)	25.15 (+1.69)	24.67 (+1.21)	26.75	28.63 (+1.89)	29.02 (+2.28)	28.66 (+1.92)

TABLE III
COMPARISON OF THE AVERAGE PSNR (DB) OBSERVED WITH THE FOUR DECODING APPROACHES FOR STREAMS CODED WITH A FIXED QP OF 27

Sequences	Channel SNR								
		4 dB				5 dB			
	Intact	JM16.0	STBMA + PDE	SO-MLD	HO-MLD	JM16.0	STBMA + PDE	SO-MLD	HO-MLD
driving	37.12	20.66	22.51 (+1.85)	22.95 (+2.30)	22.45 (+1.79)	23.92	28.84 (+4.92)	28.65 (+4.73)	27.78 (+3.86)
opening-ceremony	35.39	29.86	29.55 (-0.32)	30.03 (+0.17)	29.65 (-0.21)	32.26	32.93 (+0.67)	32.99 (+0.73)	33.04 (+0.78)
whale-show	36.30	23.94	24.16 (+0.22)	24.86 (+0.92)	24.67 (+0.73)	26.51	27.63 (+1.13)	28.57 (+2.06)	28.23 (+1.72)
city	36.64	24.71	28.34 (+3.64)	28.86 (+4.15)	28.39 (+3.68)	27.73	33.84 (+6.11)	33.87 (+6.14)	33.27 (+5.53)
crew	38.59	26.04	26.67 (+0.63)	28.00 (+1.95)	27.42 (+1.38)	28.75	30.57 (+1.83)	31.50 (+2.75)	30.86 (+2.12)
harbour	37.32	23.15	26.08 (+2.93)	24.53 (+1.38)	24.31 (+1.16)	28.55	30.68 (+2.13)	30.08 (+1.54)	29.87 (+1.32)
ice	41.30	25.14	26.05 (+0.92)	27.53 (+2.40)	26.94 (+1.80)	29.26	32.18 (+2.91)	32.98 (+3.71)	32.54 (+3.27)
soccer	37.88	22.05	23.06 (+1.01)	23.96 (+1.91)	23.49 (+1.43)	24.99	28.35 (+3.36)	28.34 (+3.45)	28.15 (+3.16)
walk	38.97	17.45	19.71 (+2.26)	20.68 (+3.23)	20.47 (+3.02)	21.67	26.36 (+4.70)	26.34 (+4.68)	26.21 (+4.54)
average	37.72	23.67	25.13 (+1.46)	25.71 (+2.04)	25.31 (+1.64)	27.07	30.15 (+3.08)	30.37 (+3.30)	29.99 (+2.92)

TABLE IV
COMPARISON OF THE AVERAGE PSNR (DB) OBSERVED WITH THE FOUR DECODING APPROACHES FOR STREAMS CODED WITH A FIXED QP OF 22

Sequences	Channel SNR								
		4 dB				5 dB			
	Intact	JM16.0	STBMA + PDE	SO-MLD	HO-MLD	JM16.0	STBMA + PDE	SO-MLD	HO-MLD
driving	41.03	20.51	22.29 (+1.79)	23.12 (+2.62)	22.59 (+2.09)	24.15	29.41 (+5.25)	29.27 (+5.12)	29.34 (+5.18)
opening-ceremony	39.37	32.02	32.33 (+0.31)	32.69 (+0.67)	33.15 (+1.13)	30.62	30.42 (-0.20)	29.86 (-0.75)	29.39 (-1.23)
whale-show	40.92	24.29	24.45 (+0.17)	25.08 (+0.79)	24.92 (+0.64)	27.30	28.27 (+0.97)	30.19 (+2.89)	29.75 (+2.45)
city	40.91	24.74	28.92 (+4.17)	28.92 (+4.17)	28.48 (+3.73)	28.27	34.67 (+6.41)	34.27 (+6.00)	33.97 (+5.70)
crew	41.87	25.94	26.48 (+0.54)	27.90 (+1.96)	27.15 (+1.21)	28.96	30.96 (+2.00)	32.07 (+3.11)	31.64 (+2.68)
harbour	41.18	24.25	26.41 (+2.16)	25.70 (+1.45)	25.46 (+1.22)	27.45	31.06 (+3.61)	30.23 (+2.78)	29.89 (+2.44)
ice	43.61	24.56	25.46 (+0.89)	27.30 (+2.73)	26.78 (+2.21)	28.32	31.98 (+3.66)	32.73 (+4.41)	32.10 (+3.79)
soccer	41.93	21.21	23.03 (+1.83)	23.51 (+2.30)	23.38 (+2.17)	24.90	29.05 (+4.15)	29.53 (+4.63)	29.32 (+4.41)
walk	42.98	17.83	19.89 (+2.06)	21.39 (+3.56)	21.48 (+3.65)	22.23	27.16 (+4.93)	27.48 (+5.26)	27.35 (+5.13)
average	41.53	23.93	25.47 (+1.55)	26.18 (+2.25)	25.93 (+2.00)	26.91	30.33 (+3.42)	30.63 (+3.71)	30.31 (+3.39)

(a) (b)

(c) (d)

Fig. 7. Luminance differences of the 99-th frame of the *ice* sequence coded with a QP of 27, and submitted to a channel SNR of 5 dB: (a) The original sequence (41.02 dB); (b) JM 16.0 (26.42 dB); (c) STBMA+PDE (31.24 dB); and (d) SO-MLD (34.66 dB).

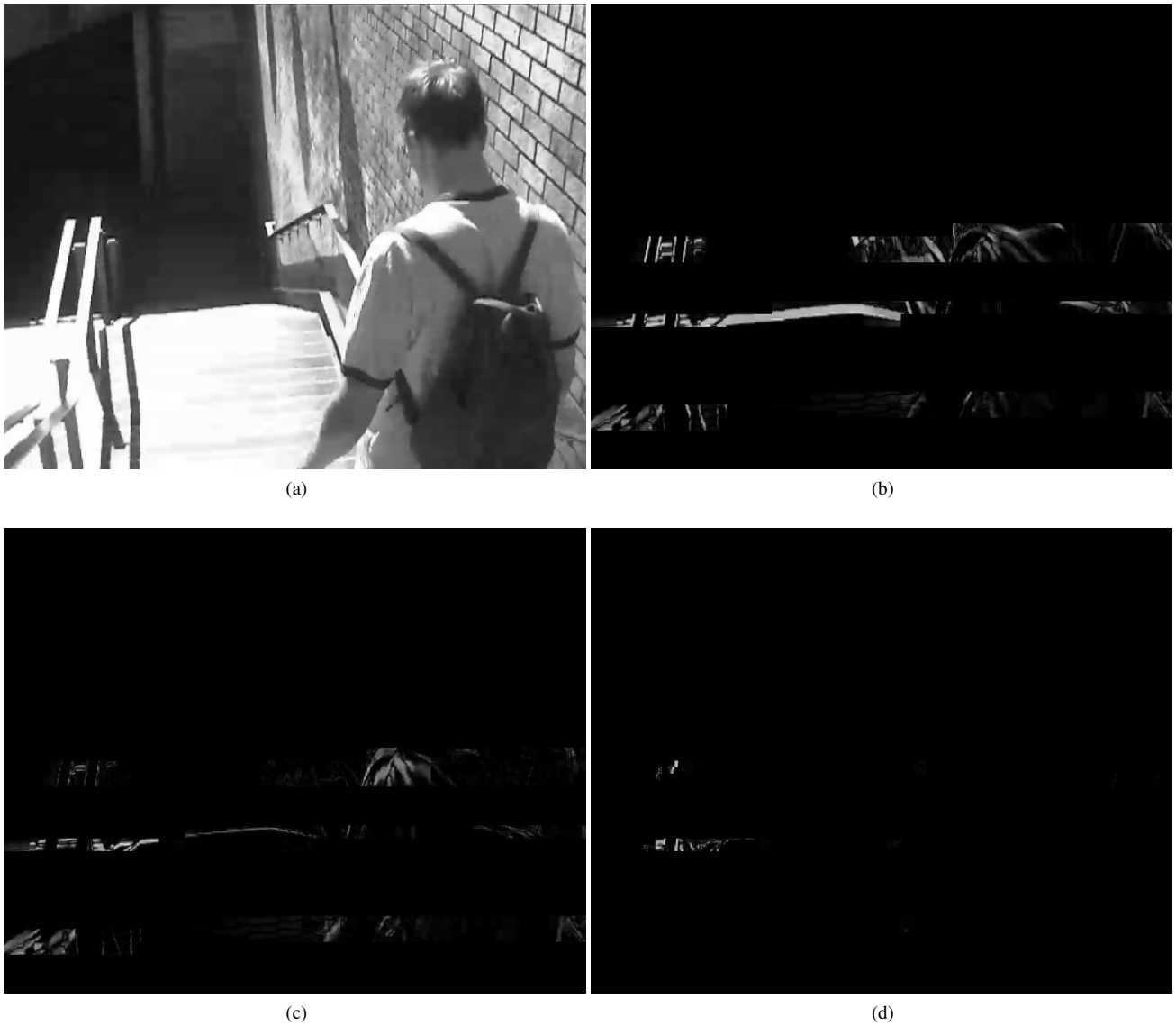
3.4 GHz Intel i7 processor, 8 Gb of 1333 DDR3 RAM running OS X 10.7. Our decoder was implemented using the C++ programming language, and compiled favoring execution speed over program size. Our implementation of the STBMA+PDE algorithm uses a maximum of 30 iterations in the PDE step. We collected the decoding time of all four approaches for all the simulations we ran as a way to measure the complexity of the different approaches. Table V summarizes the average decoding time per frame for all 9 sequences since the tests sequences share similar resolutions. The decoding times are in milliseconds per frame, and they are separated by channel SNR and QP since the amount of errors and the stream quality both affect the decoding and concealment time.

The results indicate that both SO-MLD and HO-MLD have slightly longer decoding times than STBMA+PDE. This can be explained by the fact that the concealment only approaches decode (far) less packets. Accounting for the fact that our approach deals with all packets (corrupted in addition to intact), the increase in complexity is acceptable, even for real-time communication systems. Note that neither the decoder nor the error correction method have been optimized yet.

VI. Conclusion

In this paper, we have presented our novel video error correction framework. The framework presents a general solution, where erroneous packets are exploited at the decoder side. Using JSCD, soft-output information from the physical layer is combined with the source semantics to sequentially select the most likely candidate at each step. We have also demonstrated that our solution also provides good results when soft-output information is unavailable at the application layer.

The solution was applied to H.264 Baseline profile decoding. The syntax elements used to communicate the MB prediction information were added to the models we had



(a) (b)

(c) (d)

Fig. 8. Luminance differences of the 83-rd frame of the *walk* sequence with a QP of 37, and submitted to a channel SNR of 5 dB: (a) The original frame (29.88 dB); (b) JM 16.0 (21.61 dB); (c) STBMA+PDE (25.75 dB); and (d) SO-MLD (28.97 dB)

previously proposed. The models were then compared against both standard and state-of-the-art error concealments.

The experimental results demonstrate that under the various test conditions, SO-MLD is expected to perform better. Average PSNR gains greater than 1 dB over JM 16.0 are seen in more than 70% of the scenarios we studied (42% of the scenarios show an average gain over 2 dB). The results also show that the proposed method performs better than state-of-the-art error concealment techniques. This fact is worth mentioning, as Baseline profile syntax elements still have to be modeled. Referring to Fig. 3, the syntax elements of the NAL header, the slice header, and the prediction info have been modeled. We expect the method to perform even better once the syntax elements associated to the residual info box have been modeled.

Our future work will involve applying the method to the remaining H.264 profiles as well as the HEVC standard.

ACKNOWLEDGMENT

The authors thank Dr. Yan Chen for validating our spatio-temporal cost function implementation of [12].

REFERENCES

[1] Y. Wang, and Q. F. Zhu. Error control and concealment for video communication: a review. *Proceedings of the IEEE*, 86(5):974–997, May 1998.

[2] J. Postel. User Datagram Protocol. http://www.ietf.org/rfc/rfc768.txt, August 1980.

[3] Y. Wang, S. Wenger, J. Wen, and A. K. Katsaggelos. Error resilient video coding techniques. *IEEE Signal Processing Magazine*, 17(4):61–82, July 2000.

[4] S. Wenger. H.264/AVC Over IP. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):645–656, July 2003.

[5] T. Stockhammer, M. M. Hannuksela, and T. Wiegand. H.264/AVC in Wireless Environments. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):657–673, July 2003.

[6] International Telecommunications Union. ITU-T Recommendation H.264 : Advanced video coding for generic audiovisual services. http://www.itu.int/rec/T-REC-H.264/en, November 2007.

[7] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H.264/AVC Video Coding Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, July 2003.

[8] Y. K. Wang, M. M. Hannuksela, V. Varsa, and A. Hourunranta. The error concealment feature in the H.26L test model. In *2002 International Conference on Image Processing*, 2:II-729–II-732, 2002.

[9] S. Kumar, M. Mandal L. Xu, and S. Panchanathan. Error Resiliency Schemes in H.264/AVC Standard. *Journal of Visual Communication and Image Representation*, 17(2):425–450, April 2006.

[10] P. Lambert, W. De Neve, Y. Dhondt, and R. Van de Walle. Flexible Macroblock Ordering in H.264/AVC. *Journal of Visual Communication and Image Representation*, 17(2):358–375, April 2006.

[11] Y. Xu and Y. Zhou. H.264 Video Communication Based Refined Error Concealment Schemes. *IEEE Transactions on Consumer Electronics*, 50(4):1135–1141, November 2004.

[12] Y. Chen, Y. Hu, O. S. Au, H. Li, and C. W. Chen. Video Error Concealment Using Spatio-Temporal Boundary Matching and Partial Differential Equation. *IEEE Transactions on Multimedia*, 10(1):2–15, January 2008.

[13] J. Wu, X. Liu, and K. Y. Yoo. A Temporal Error Concealment Method for H.264/AVC Using Motion Vector Recovery. *IEEE Transactions on Consumer Electronics*, 54(4):1880–1885, November 2008.

[14] S. C. Huang and S. Kuo. Optimization of Hybridized Error Concealment for H.264. *IEEE Transactions on Broadcasting*, 54(3):499–516, September 2008.

[15] D. Persson and T. Eriksson. Mixture Model- and Least Squares-Based Packet Video Error Concealment. *IEEE Transactions on Image Processing*, 18(5):1048–1054, May 2009.

[16] B. Yan and H. Gharavi. A Hybrid Frame Concealment Algorithm for H.264/AVC. *IEEE Transactions on Image Processing*, 19(1):98–107, January 2010.

[17] L. Larzon, M. Degermark, and M. Pink, S. Degermark. UDP Lite for Real Time Multimedia Applications. Technical report, In Proceedings of the QoS mini-conference of IEEE International Conference of Communications, 1999.

[18] W. J. Chu, and J. J. Leou. Detection and concealment of transmission errors in H.261 images. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(1):74–84, February 1998.

[19] H. C. Shyu, and J. J. Leou. Detection and concealment of transmission errors in MPEG-2 images-a genetic algorithm approach. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(6):937–948, September 1999.

[20] L. Superiori, O. Nemethova, and M. Rupp. Performance of a H.264/AVC Error Detection Algorithm Based on Syntax Analysis. In *International Conference on Advances in Mobile Computing and Multimedia*, pages 49–58, 2006.

[21] R. A. Farrugia and C. J. Debono. A Robust Error Detection Mechanism for H.264/AVC Coded Video Sequences Based on Support Vector Machines. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(12):1766–1770, 2009.

[22] L. Trudeau, S. Coulombe, and S. Pigeon. Pixel Domain Referenceless Visual Degradation Detection and Error Concealment for Mobile Video. In *18th International Conference on Image Processing*, pages 2229–2232. IEEE, September 2011.

[23] X. Ma and W. E. Lynch. Iterative Joint Source-Channel Decoding Using Turbo Codes for MPEG-4 Video Transmission. In *International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages 657–660. IEEE, May 2004.

[24] G. Sabeva, S. Ben Jamaa, M. Kieffer, and P. Duhamel. Robust Decoding of H.264 Encoded Video Transmitted over Wireless Channels. In *8th Workshop on Multimedia Signal Processing*, pages 9–13. IEEE, October 2006.

[25] D. Levine, W. E. Lynch, and T. Le-Ngoc. Iterative Joint Source-Channel Decoding of H.264 Compressed Video. In *International Symposium on Circuits and Systems*, pages 1517–1520. IEEE, May 2007.

[26] N. 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 *23rd Canadian Conference on Electrical and Computer Engineering*, pages 1–4. IEEE, May 2010.

[27] R.A. Farrugia and C.J. Debono. Robust decoder-based error control strategy for recovery of H.264/AVC video content. *IET Communications*, 5(11):1928–1938, 2011.

[28] Y. Wang and S. Yu. Joint Source-Channel Decoding for H.264 Coded Video Stream. *IEEE Transactions on Consumer Electronics*, 51(4):1273–1276, November 2005.

[29] H. Nguyen and P. Duhamel. Iterative joint source-channel decoding of variable length encoded sequences exploiting source semantics. In *International Conference on Image Processing*, volume 5, pages 3221–3224. IEEE, October 2004.

[30] W.-T. Lee, H.-J. Chen, Y.-S. Hwang, and J.-J. Chen. Joint Source-Channel Decoder for H.264 Coded Video Employing Fuzzy Adaptive Method. In *International Conference on Multimedia and Expo*, pages 755–758. IEEE, July 2007.

[31] C. Weidmann, P. Kadlec, O. Nemethova, and A. A. Moghrabi. Combined Sequential Decoding and Error Concealment of H.264 Video. In *6th Workshop on Multimedia Signal Processing*, pages 299–302. IEEE, October 2004.

[32] C. Bergeron and C. Lamy-Bergot. Soft-input decoding of variable-length codes applied to the H.264 standard. In *6th Workshop on Multimedia Signal Processing*, pages 87–90. IEEE, October 2004.

[33] R.A. Farrugia and C.J. Debono. A Hybrid Error Control and Artifact Detection Mechanism for Robust Decoding of H.264/AVC Video Sequences. *IEEE Transactions on Circuits and Systems for Video Technology*, 20(5):756–762, 2010.

[34] F. Caron and S. Coulombe. A Maximum Likelihood Approach to Video Error Correction Applied to H.264 Decoding. In *6th International Conference on Next Generation Mobile Applications, Services, and Technologies*, pages 1–6. IEEE, September 2012.

[35] F. Caron and S. Coulombe. Joint Source-Channel Decoding for H.264 Coded Video Using a Maximum Likelihood Approach to Correct Transmission Errors. Accepted in the *20th International Conference on Image Processing*, September 2013.

[36] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1649–1668, December 2012.

[37] Bernard Sklar. *Digital Communications Fundamentals and Applications*. Prentice Hall, second edition, 2001.

[38] S. Wenger, M.M. Hannuksela, T. Stockhammer, M. Westerlund, and D. Singer. RTP Payload Format for H.264 Video. http://www.ietf.org/rfc/rfc3984.txt, February 2005.

[39] Joint Video Team. H.264/AVC JM reference software. http://iphome.hhi.de/suehring/tml/, November 2011.

[40] A. T. Connie, P. Nasiopoulos, V. C. M. Leung, and Y. P. Fallah. Video Packetization Techniques for Enhancing H.264 Video Transmission over 3G Networks. In *5th Consumer Communications and Networking Conference*, pages 800–804. IEEE, January 2008.

[41] Joint Video Team. 0000277: using RTP mode, when droping some packets with rtp_loss.exe decoder can not decode. https://ipbt.hhi.fraunhofer.de/mantis/view.php?id=277, August 2011.

François Caron received a B.Eng. in IT Engineering from the École de technologie supérieure, Canada in 2008, and a Ph.D in 2013. Since April 2013, he has joined Vantrix Corporation, as a software developer working on H.265/HEVC.

Stéphane Coulombe (S'90-M'98-SM'01) received a B.Eng. in Electrical Engineering from the École Polytechnique de Montréal, Canada, in 1991, and a Ph.D. from INRS-Telecommunications, Montréal, in 1996. He is a Professor at the Software and IT Engineering Department, École de technologie supérieure (ÉTS is a part 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 Engineer and as 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, media adaptation, and transcoding. Since 2009, he has held the Vantrix Industrial Research Chair in Video Optimization.