Slice Error Detection and Correction Decoder for H.264/AVC Video Sequences

Kai Yen and Sumei Sun
Institute for Infocomm Research, A*STAR
1 Fusionopolis Way, #21-01 Connexis (South Tower), Singapore 138632
Email: {yenkai,sunsm}@i2r.a-star.edu.sg

Abstract—In this paper, we propose a slice error detection and correction decoder (SEDCD) for the H.264/AVC encoded video sequence. Error detection is accomplished by making use of the CRC checksum as well as syntax analysis. Once errors are detected in the slice, a tree-search-based algorithm is invoked in order to identify and correct the errors. A unique feature of our proposed SEDCD is that it exploits the knowledge about the syntax failure to aid in its search for the errors. Through simulations, we show that the SEDCD can improve the PSNR by a maximum of 3.5 dB as compared to one without any error correction.

Keywords—H.264/AVC, Syntax analysis, error correction decoding

I. INTRODUCTION

H.264/AVC is the latest video coding standard for future multimedia applications and services [1], [2]. One of the main attributes of H.264/AVC is its excellent compression efficiency. However, this also makes the H.264/AVC-coded video highly susceptible to transmission errors, which can result in significant degradation in the perceptual quality of the decompressed video sequence. For applications that are not critically restricted by end-to-end delay constraints, automatic repeat request (ARQ) mechanisms can be invoked in order to ensure that the received video sequence is free of errors. On the other hand, applications with very low latency tolerance cannot afford the luxury of having such feedback and retransmission protocols. Hence, in order to maintain a high perceptual video quality, the transmission of such applications is usually operated at a very low error rate. Nevertheless, bit errors can still occur occasionally. In packetized transmission, erroneous packets are usually detected with the aid of an embedded CRC checksum and subsequently discarded. Hence the reconstructed video quality will be severely degraded, especially if error propagation arises.

A common approach to minimizing the visual artifacts in the video due to errors is by error concealment [3]. In general, erroneous macroblocks (MB) can be concealed using either intra-frame or inter-frame interpolation by exploiting correctly decoded information from other parts of the video. The advantages of error concealment methods are that the overall coding efficiency remains the same and that they can generally be applied to any video standards. On the other hand, they performed rather poorly under high channel error rate. Because of this, they are usually used in conjunction with error resiliency techniques [4]. In general, the purpose of error resiliency techniques is to minimize the spread of distortion in the video that are caused by transmission errors. Error resilience techniques generally lower the overall coding efficiency because redundancies are being introduced into the video sequence. Slice structuring and flexible macroblock ordering (FMO) are some of the more popular error resilient techniques.

Recently, Farrugia and Debono proposed a novel error control strategy based on list decoding, which enables erroneous bits in the video slice to be corrected [5]–[7]. They showed that their method, aided by a slice structuring error resilience technique and zero motion vector error concealment, is capable of achieving a peak signal-to-noise ratio (PSNR) gain of about 2 dB as compared to one without any error correction [7]. Further gain can be obtained by incorporating their method with more sophisticated error resilience and error concealment techniques. The list decoder is defined by a parameter \( M \), which dictates the number of surviving branches at each stage. Analogous to the breadth-first tree search, for a given value of \( M \), the performance of the list decoder is heavily dependent on the length of the decoded sequence as well as the number of possible paths at each stage. The results in [5]–[7] were based on a modest size of 100 bytes per slice. Furthermore, some syntax (stages) have a high number of possible codewords (paths). Hence for a given value of \( M \), the improvement achieved by the \( M \)-based list decoder may be diminished for longer slice length. Nevertheless, having the capability of error correction will definitely lead to an improved reconstructed video quality.

In this paper, we proposed an alternative method to the \( M \)-based list decoder. We shall refer to our proposed decoder simply as Slice Error Detection and Correction Decoder (SEDCD). Instead of the parameter \( M \), the SEDCD is based on another parameter, which we denoted as \( \text{HD}_{\text{max}} \). This parameter defines the maximum Hamming distance we postulated the erroneous slice would be from the error-free slice. Another difference between our proposed SEDCD and the list decoder of [5]–[7] is that the SEDCD also exploits the information about the syntax failure to aid its decoding. From the simulation results, we observed a consistent improvement in the PSNR independent of the slice length. The gain in the PSNR can be as high as 3.5 dB when compared to one without any error correction. The SEDCD is most suitable for low BER conditions whereby conventionally, one or two bit errors will
render the entire slice unusable. It should be noted that the distinguishing difference between the SEDCD and the similar joint-source-channel decoder (JSCD) [8] is that the SEDCD does not need to be applied strictly at the physical layer; it can be used in the upper layers.

This paper is organized as follows. Following this introduction, we will highlight the main components of the H.264/AVC that are relevant for this paper in Section II. The SEDCD is discussed in Section III. Simulation results will be presented in Section IV. Finally, conclusions are given in Section V.

II. H.264/AVC VIDEO CODEC

The basic framework of the H.264/AVC video coding is well documented in the standards [9] as well as many references [1], [2]. Hence we will only highlight the components that are relevant to our discourse in this paper. Here we are only interested in the Video Coding Layer Network Abstraction Layer Unit (VCL NALU), which composes of a slice header and one or more encoded macroblocks of the video. The video can be coded such that a single slice will constitute an NALU and each NALU is assumed to fit into a packet that is framed by the system transport protocol for packetized transmission. Hence depending on the slice length, a single video frame may comprised of severalNALUs.

Two levels of error detection are being invoked at the receiver in our proposal. Firstly, if the received packet contains errors, it will be detected at the transport layer via the CRC checksum. Conventionally, the erroneous packet will be discarded and the slice contained in the packet will be subsequently concealed. However, in our case, as well as in [5]–[7], [11], erroneous packets will not be discarded at this stage but are instead retained for a second level of error detection. The packet discarding/slice concealment (SLC) approach will serve as the benchmark for comparison against our proposed method in Section IV.

For the second level of error detection, the slice in an erroneous packet is checked for any violation in its syntax/semantic rules. A slice corrupted by errors will likely result in a violation of the syntax/semantic rules, which can be exploited for the purpose of error detection as exemplified by syntax analysis-based error correction methods in [5]–[8], [11]. Detailed descriptions on the different types of errors that can be detected by syntax analysis were outlined in [11]. Similar to the other methods, our proposed SEDCD will be invoked only if the syntax analysis detected a violation in the slice.

Note that certain errors in a slice will not violate the syntax/semantic rules [11], [12], and thus cannot be detected by syntax analysis. Hence such errors are termed as undetectable errors and they cannot be corrected by syntax analysis-based error correction methods, such as the SEDCD. However, since the slice was clearly flagged as erroneous at the transport layer, it will be discarded. Hence in our study, all erroneous slices that passed the syntax analysis are discarded and concealed. The SEDCD will not be activated. Thus in this paper, we are only interested in those errors that caused syntax failures. Henceforth, any errors mentioned will denote these errors, unless otherwise stated.

Fig. 1: Illustration of a syntax error caused by one bit error in a slice.

Fig. 2: Illustration of syntax errors caused by two bit errors in a slice.

Two different entropy coding methods are available in H.264/AVC, namely; Context-Adaptive Variable Length Coding (CAVLC) and Context-Adaptive Binary Arithmetic Coding (CABAC). CAVLC will be considered in our work although the underlying principle can also be applied to CABAC. Furthermore, we assumed that the slice header as well as headers associated with non-VCL NALUs are free of errors. Readers may refer to [13] for an error correction scheme for the non-VCL headers.

III. PROPOSED DECODER

As the name implies, the SEDCD comprises of error detection and error correction. The error detection is based on the CRC checksum as well as syntax analysis, as mentioned previously. While the syntax analysis is largely similar to those highlighted in [5]–[7], [11], a unique feature in our proposed SEDCD is that the information gleaned from the syntax analysis is being exploited in the decoding.

A. Error Detection with Syntax Analysis

Let us first examine the event of a syntax error caused by a single bit error in the slice. A similar description was also given in [11]. Suppose a bit error occurs at position $E_A$ in the slice, which corresponds to macroblock $MB_{E_A}$, as shown in Fig. 1. Due to error propagation, the actual syntax error is only detected at position $S_A$ corresponding to $MB_{S_A}$. Note that the bit in $S_A$ may not necessarily be erroneous and that it is possible that $MB_{E_A} = MB_{S_A}$ even though it was shown differently in the figure. Moreover, it is also possible that $E_A = S_A$, i.e. the bit error causes a syntax error at the exactly the same position with no error propagation.

When a syntax error is registered at $S_A$, it can be deduced that at least one bit error has occurred between position 0 and $S_A$. If there is only one bit error, as illustrated in Fig. 1 in this case, then correcting this error at $E_A$ will eventually render the slice a success in the syntax analysis. The brute-force approach is a simple, yet tedious, way of finding the bit error whereby each bit from position 0 up to $S_A$ is perturbed systematically and the validity of the slice is checked by syntax analysis. The SEDCD offers a less tedious way of finding this bit error, as we shall highlight later.

Now what happens if there are two bit errors in the slice, as shown in Fig. 2? Here, another bit error has occurred at $E_B$ corresponding to $MB_{E_B}$ and this error causes a syntax failure at $S_B$. The figure shows that $E_B$ occurs after $S_A$ but...
this is not necessarily the case as $E_B$ may occur before $S_A$. Nevertheless, regardless of the positions of $E_B$ and $S_A$, let us first assume that the syntax error $S_B$ occurs after $S_A$. During the first round of syntax analysis, as before, a syntax error has been discovered at $S_A$. The syntax error at $S_B$ will not be discovered at this point in time since the analysis will terminate once a syntax error is discovered. The SEDCD will try to detect and correct the bit error that is causing the syntax error at $S_A$. If it succeeds in correcting the error at $E_A$, then the syntax error at $S_A$ will vanished. However, the slice will still failed the syntax analysis but this time the syntax error appears at $S_B$. Again, it can be deduced that a bit error has occurred between position 0 to $S_B$ and the SEDCD will be invoked to detect and correct the bit error that is causing this syntax error.

Now what if $S_B$ occurs before $S_A$ but $E_B$ comes after $E_A$? In this case, the syntax error at $S_B$ will be detected first and the SEDCD will try to detect and correct the bit error at $E_B$. Assuming that the correction is successful, the next syntax error will now occur at $S_A$. Hence in the second round of correction, the SEDCD will detect and correct the bit error at $E_A$. Because of this scenario, it is preferable to start the second iteration of correction from position 0 instead of $E_B$ as it is possible that $E_A$ occurs before $E_B$ even though the syntax error occurs after it.

This iterative approach of correction is very tedious if there are many errors in the slice. Hence we have set a limit to the number of iterations for the SEDCD, which we have denoted as $H_{\text{dmax}}$. Since one bit error can be corrected for each iteration of correction, $H_{\text{dmax}}$ can be considered as the number of errors we postulated in the slice. If the syntax analysis still fails after $H_{\text{dmax}}$ number of iterations, the slice will be discarded and concealed.

Furthermore, it is also possible that two or more bit errors can jointly caused a single syntax error. For instance, a codeword corresponding to a syntax may contain two or more errors, which will only cause a single syntax error. In this case, these erroneous bits have to be corrected jointly and the brute-force approach will be impractical. Our proposed SEDCD is able to solve this without too much complexity.

In general, the essential element during the syntax analysis is to track the position in the slice where the syntax failure occurs. This knowledge will be exploited during the error correction.

### B. Error Correction

Having detected the presence of bit error/s through syntax analysis, the SEDCD will attempt to correctly identify the bit/s that are causing the syntax failure. The SEDCD will first assume that there is only one bit error that is causing the syntax failure. As mentioned previously, the SEDCD will exploit the knowledge about the syntax failure to aid in its search for the erroneous bit. More specifically, the SEDCD will track the position of the syntax failure in order to determine if the bit error has been corrected.

Similar to the list decoder proposed by Farrugia and Debono, the SEDCD adopts a tree-search algorithm, whereby the tree follows the semantics of the H.264/AVC standard and each stage represents a syntax. The number of branches emanating from each stage depends on the number of possible codewords associated with the syntax.

We shall use Figs. 1 and 2 as examples in our discourse. Let us supposed that there is only one bit error, which causes a syntax failure at $S_A$ as illustrated in Fig. 1. The search will commence from MB0. Note that MB0 denotes the first MB in the slice, but not necessarily the first MB of the frame. For the first iteration of correction, we set a parameter $H_D = 1$ and the erroneous slice will be used as the basis slice. According to the standard, the first syntax, assuming the slice corresponds to an I-slice, is $\text{mb}_0_{\text{type}}$. This syntax hash 25 possible codewords.

Since we have assumed only a single bit error, those codewords with more than one bit difference from the basis slice are not considered. For those codewords that are exactly one bit different from the slice, they are concatenated with the remaining length of the slice. For example, if the length of the codeword is $k$, then the slice from $k$ to $N - 1$ will be concatenated to it. Bits from 0 to $k - 1$ will correspond to the codeword. In this way, the Hamming distance of the generated slice relative to the basis slice is one. If the bit error at $E_A$ does not belong to this syntax, then the Hamming distance of the generated slice relative to the original transmitted slice is two, since an additional bit error is generated, together with the one at position $E_A$. The generated slice is then subjected to a syntax analysis test. Several scenarios will happen here, which is listed below:

1. The syntax failure remains at $S_A$. This could mean that the new bit error is an undetectable bit error, or it causes a syntax failure after $S_A$.
2. The syntax failure happens before $S_A$. Obviously, this is caused by the new bit error.
3. The syntax failure happens after $S_A$. This may mean that the bit error at $E_A$ has been resolved and another bit error in the slice is causing this new syntax failure.
4. The syntax analysis registers a success.

For the first two cases, the generated slice is discarded since it is certain that the bit error has not been resolved. For the third case, the generated slice is placed in a so-called $\text{Likely}(HD)$ bin. For the fourth case, the valid generated slice is placed in a so-called $\text{Success}(HD)$ bin. Then the tree-search proceeds on to the next syntax and the procedure is repeated until the bit position at $S_A$ is reached. At this point, if there are at least one valid slice in the $\text{Success}(HD)$ bin, the algorithm is stopped.

There may exists several valid slices in the bin, each having a Hamming distance equals to HD from the received slice. Following the method used in [5]–[7], the CRC checksum is re-computed for each and every valid slice to determine the correct slice. This slice is then forwarded to the video decoder. If none of the slices satisfies the checksum, which is likely due to the presence of undetectable bit errors in the slice, the slice is declared as erroneous.

On the other hand, if the $\text{Success}(HD)$ bin is empty, the parameter HD is incremented by 1. If HD <= $H_{\text{dmax}}$, the

---

1 Exp-Golomb variable-length codewords
algorithm will proceed with the next iteration. Otherwise, the algorithm is stopped and the slice is declared as erroneous and discarded.

For the next iteration, if the Likely(HD) bin is not empty, a new basis slice will be selected from the bin for the decoding. Usually, there will be several slices in this bin. To select the best basis slice, we choose the one having the farthest placed syntax error. Let’s supposed that this position is $S_B$, as shown in Fig. 2. Then the same error correction procedure will commence from the first syntax as before and repeated until the bit position at $S_B$. The same stopping criteria and decision making are then applied, as highlighted in the previous paragraphs.

On the other hand, if the Likely(HD) bin is empty, then the same basis slice from the previous iteration will be used. However, the decoding will be altered slightly. Now, instead of considering only those codewords having 1 bit difference from the basis slice at each stage as in the previous iteration, those codewords with 2 bit differences are also considered in this iteration. Branches having 2 bit differences relative from the basis slice are concatenated with the remaining length of the basis slice and the generated slices are tested by the syntax analysis. Branches having a 1 bit difference relative to the basis slice are kept for the subsequent syntax, analogous to surviving branches. This course of action will resolve those special situations whereby two bit errors jointly caused a syntax failure. This approach can also be applied for more than 2 bit errors.

It is obvious that the SEDCD is more discerning than the brute-force approach as the search is based on the H.264/AVC semantics. Furthermore, the SEDCD relies heavily on the knowledge of the position of the syntax error, which makes our algorithm unique as compared to Farrugia and Debono’s method [5]–[7]. In [11], this knowledge was also exploited but only for the purpose of an efficient error concealment but not error correction.

C. Complexity

Our method invokes the syntax analysis frequently. However, the analysis is based on checking the validity of the slice. Hence no arithmetic operation is involved and the analysis can be performed very rapidly and it will stop once a syntax failure is detected. Furthermore, no metric calculation is involved in our method. The SEDCD only keeps track of the Hamming distance of the slice in relation to the basis slice as well as the point of syntax failure. Hence there is not a lot of calculations involved.

It was shown in [11] that the error propagation delay, i.e. the point of the bit error to the point of syntax failure, usually ranges within 2 MBs for I-slices to 15 MBs for P-slices. Hence, instead of starting the decoding from the first MB of the slice, the decoding can commence from $MB_{syn} = X$, where $MB_{syn}$ is the MB in which the syntax failure occurs and $X$ is a predetermined value depending on the type of slice. This approach will reduce the latency and complexity of the decoding but it may also reduced the achievable gain as compared to starting the decoding from the first MB. In all our simulations, the decoding starts from the first MB in the slice.

In order to reduce latency further, slices in the Success(HD) bin can be validated against the CRC checksum concurrently. The algorithm can be stopped immediately once a slice that agrees with the checksum is found. Hence there is no need to proceed with the decoding until the point of syntax error.

IV. SIMULATION RESULTS

The Foreman QCIF sequence is used in our simulations. There are a total of 400 frames in the sequence and the GOP size is set to 10. The quantization parameter (QP) is set to 28. The performances for two cases are simulated. In the first case (Case 1), the entire frame constitutes a slice. In this case, the slice length will be fairly long, especially for I-slices ($\approx 25K$ bits long). In the second case (Case 2), the error resilient slice structuring technique is invoked whereby each slice is limited to a maximum of 700 bytes. In both cases, if the slice still fails the CRC checksum after the error correction, the slice is discarded and is concealed by means of a zero motion temporal error concealment. As mentioned previously, the SLC-only method (without any error correction) will be used as the benchmark. All the results are plotted against the average input bit error rate (BER), whereby the bits in the slices are randomly perturbed according to the desired BER.

The PSNR of the luminance component (Y-PSNR) results are shown in Fig. 3 for both cases with $HD_{max} = 1$ and 2. We observed that the SEDCD is capable of improving the PSNR by a maximum of about 3.5 dB relative to SLC for both cases. Hence there is a consistent improvement regardless of the slice length. We also noticed that there is only a meagre improvement for $HD_{max} = 2$ over $HD_{max} = 1$. The difference between $HD_{max} = 2$ and $HD_{max} = 1$ is that the former is capable of correcting two errors instead of just one. However, this would inadvertently imply an increased in the probability of having undetectable bit errors in the slice, which would render the slice as erroneous, regardless of whether the SEDCD has succeeded in correcting the detectable errors. Thus, it is more feasible to set $HD_{max}$ not greater than 2.

The average output BER performance of the SEDCD is shown in Fig. 4. The SLC method will have the same average input and output BER for both cases. We observed an
improvement in the average output BER when the SEDCD is invoked. Since the BER is independent of the slice length, we observed that both cases exhibit approximately the same BER performance. There is only a slight gain for \( \text{HD}_{\text{max}} = 2 \) over \( \text{HD}_{\text{max}} = 1 \) in the higher average input BER region. Hence, the SEDCD is ideal in low BER conditions whereby one or two bit errors can be corrected, which would otherwise render the entire slice unusable. Also, the improved BER runs parallel with the original BER at the low BER region. The constant gap between the two BERs indicates the maximum gain that can be gleaned from the SEDCD. This limitation is due to the undetectable errors, which cannot be corrected by the SEDCD. Thus, the BER cannot be improved further unless a solution to correct the undetectable errors is developed.

On the other hand, the slice error rate (SER) is dependent on the slice length. This is shown in Fig. 5. Clearly, a shorter slice length will have a lower SER, for a given input BER. We observed that the SEDCD is able to improve the SER performance for both cases, resulting in a better PSNR as shown previously in Fig. 3.

V. Conclusions

In this paper, we have proposed a slice error detection and correction decoder (SEDCD). The SEDCD make use of the CRC checksum and syntax analysis for the purpose of error detection. If errors are detected, the SEDCD will attempt to identify and correct the errors based on a tree-search algorithm, in which the knowledge about the syntax failure is being exploited in order to aid its search. Simulation results showed that the SEDCD is capable of improving the PSNR by a maximum of 3.5 dB relative to one without any error correction, regardless of the slice length. The corresponding BER and SER are also improved.

Acknowledgement

The authors would like to thank Dr. Jinghong Zheng for her technical guidance on the H.264/AVC codec.

References


