Multi-resolution Adaptation of the SPIHT Algorithm for Multiple Description

Nedeljko Varnica   Michael Fleming   Michelle Effros[1]

Abstract

Multiple description codes are data compression algorithms designed with the goal of minimizing the distortion caused by data loss in packet-based or diversity communications systems. Recently, techniques that achieve multiple description coding by combining embedded source codes with unequal error protection channel codes have become popular in the literature. These codes allow for data reconstruction with any subset of the transmitted packets and achieve progressively better source reconstructions as more and more packets are decoded. The given methods may be applied to any embedded source description. While applicability to all embedded source codes provides great flexibility, this separation approach begs the question of whether better performance could be achieved by taking advantage of the internal structure of a particular embedded code. In this paper, we investigate an extremely simple method for using an embedded source code's internal state information in the construction of a multiple description code. In particular, we protect an embedded SPIHT bitstream by adding to that bitstream periodic descriptions of state information from the encoder, and we demonstrate how the state information can be used to recover lost bits. For low probabilities of network packet loss, the proposed algorithm achieves performance within 0.35 dB of the performance of a more sophisticated channel coding algorithm when both algorithms are applied to same SPIHT embedded source code.

1. Introduction

In packet-based communication systems, packet losses are inevitable, due, for example, to buffer overflows. The most common strategy to protect the compressed data from packet loss is to request retransmission of lost packets. This ensures that all data will be received by the decoder. However, retransmission requires the network to employ a reverse channel (so that the receiver can notify the transmitter of the losses), and retransmission often involves significant delays. Moreover, since packet loss often results from buffer overflows in times of high network traffic, retransmitted packets will often experience the same lossy network conditions that caused the initial packet losses.

[1]N. Varnica is with the School of Electrical Engineering, University of Belgrade, Serbia, Yugoslavia, Ph:+381-11-310-2825. M. Fleming and M. Effros are with the Department of Electrical Engineering, MC 136-93, California Institute of Technology, Pasadena, CA 91125, Ph: 626-395-2219. This material is based upon work supported by NSF Award No. CCR-9909026, the Pickering Fellowship, the F.W.W. Rhodes Memorial Scholarship, a Redshaw Award, and the Intel Technology for Education 2000 program.

Applications with high delay sensitivity or no reverse channel may require data reconstruction without missing packets. For such applications, multiple description (MD) source coding provides the most appropriate approach to dealing with the problem of packet loss. An MD code is a code that generates multiple independent representations of the data in such a way as to provide better and better reconstructions as the number of received data descriptions increases. (In a packet-based network, each packet is considered as a description). The aim of MD code design is to yield good rate-distortion performance under a wide variety of information loss scenarios.

The MD codes of [1, 2] combine forward error correction (FEC) techniques with embedded or multi-resolution image descriptions. In an embedded image description, low rate (high distortion) data descriptions are embedded within higher rate (lower distortion) descriptions of the same data, i.e. the description information can be considered to be arranged in order of decreasing importance. An embedded description is generally very sensitive to erasures or errors; a single bit error or erasure can cause loss of synchronization in the decoder, resulting in sharp degradation in performance. Indeed, in many cases it is better to stop decoding at the point the error occurred than to continue decoding the corrupted bitstream. Thus, to be used in a multiple description code, an embedded code must be protected from data loss. The schemes of [1, 2] use FEC to protect an embedded bitstream against erasures, employing unequal loss protection to protect the most important information (at the start of the embedded description) most strongly, and apply progressively weaker protection to subsequent bits in the embedded description. Unlike more direct FEC-based approaches, the MD codes of [1, 2] preserve the progressive reconstruction property of their embedded code. That is, decoding of the entire image can begin once the first packet is received, and each subsequent packet allows the image reconstruction to be refined.

The FEC-based unequal loss protection algorithms of [1, 2] are designed to work with *any* embedded source code. Separating the source and channel codes in this way allows for independent optimization of the source and channel codes. However, the constraint imposed by separation implies that such an algorithm cannot take advantage of any information internal to the source coder. A code that takes advantage of this information, and hence violates the separation principle, can, in theory, always achieve performance at least as good as these existing FEC algorithms. However, it is unclear how to build such codes in practice.

This paper considers the questions of how state information from the source code could be used for erasure protection if it were available and what gains might be achieved through its use. To investigate this question, we propose an extremely simple MD code based on the SPIHT embedded source coding algorithm [3]. Like the unequal loss protection methods mentioned earlier, the new algorithm preserves the embedded or multi-resolution property of SPIHT. The proposed MD code protects SPIHT's encoded bitstream through periodic descriptions of partial state information extracted from the encoder rather than by more traditional error correction methods. The corrupted bitstream is corrected using an iterative algorithm based only on the state information – which may itself be corrupted by erasures due to packet losses.

The proposed approach has some different characteristics from those of multiple description codes based on FEC protection. For example, with FEC methods, less and less of the original description is decoded as more and more packets are lost. In the proposed algorithm, the entire source data stream is always decoded, but as that bitstream gets corrupted by more and more erasures, the quality of the reconstruction based on that full data stream degrades.[2]

The paper is organized as follows. Section 2 presents background information about the SPIHT algorithm and unequal loss protection schemes. In Section 3 we present an algorithm that periodically adds state information to an unprotected source bitstream and we describe how this state information can be used to reconstruct lost source data bits. Section 4 presents performance results for the new algorithm, and conclusions are drawn in Section 5.

2. Background

In this section we present background information on the SPIHT algorithm and on unequal loss protection schemes.

The SPIHT Algorithm

The SPIHT algorithm [3] is extremely popular because it produces an embedded image description with high efficiency and low computational complexity. A key feature of this algorithm is that it stores and updates state information as it encodes an image. This state information is in the form of lists - the list of insignificant pixels (LIP), the list of significant pixels (LSP), and the list of insignificant sets (LIS). These lists keep a record of the significance of each wavelet coefficient at each step of the algorithm. Coefficients or sets of coefficients are said to be significant with respect to a given threshold if they exceed that threshold, and insignificant with respect to that threshold otherwise. All significant coefficients are listed as the elements of the LSP. All insignificant coefficients accessed during the previous passes of the algorithm are listed in the LIP. The sets of coefficients that need to be checked for significance are kept as LIS entries.

According to their significance, the wavelet coefficients are added to and removed from the lists during the four steps of the SPIHT algorithm. These steps are: initialization, sorting, refinement, and updating of the quantization-step. The sorting pass can be further divided into two parts: one that deals with the LIP entries, and one that deals with the LIS entries. We call these parts the LIP pass and the LIS pass respectively.

The bits produced during the refinement pass are unpredictable and do not change the state of the encoder. Thus, it is impossible to use state information to correct these bits. In the discussion that follows, these bits are treated separately from rest of the source data and protected via a forward error correcting code.

The internal state of the SPIHT encoder is updated when any non-refinement

[2]Here and subsequently, when we refer to source data or the source bitstream, we are referring to the binary description created by the embedded source coder.

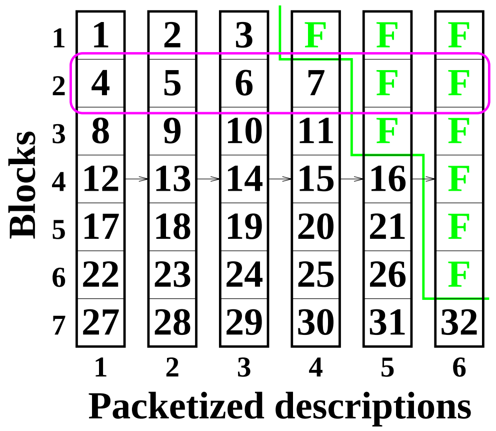

Figure 1: Each of the rows is a block and each of the columns is a packet. A packet consists of one byte from each block. Here, a message of 32 bytes of data (numbered 1-32) and ten bytes of FEC (F) forms seven blocks which are encoded into six packets.

source data bit is produced. Perfect knowledge of the state of the SPIHT encoder at every point in the bitstream would allow us to correct an erasure of any such bit. Describing the state of the SPIHT encoder at every point would take as many bits as describing the source data itself. However, lost bits can be deduced with high probability from only partial knowledge of the state information; a method to do this is presented in Section 3.

Unequal Loss Protection Schemes

Unequal loss protection (ULP) schemes are FEC based schemes that protect the source bitstream without using state information. The source bitstream is divided sequentially into blocks, the size of each block being determined by the importance of its data. The first block, formed from the data at the beginning of the source description, is protected most strongly, since this is the most important part of an embedded image description – a lost bit in the first block would render almost the entire description undecodable. The error protection on subsequent blocks is slowly reduced, since the data becomes less and less important. Figure 1, reproduced with permission of the author, shows the approach adopted in [1]. The amount of data in each block is chosen so that the number of data and protection bytes combined is constant for each block. Packets are then formed using corresponding bytes from each block.

3. Algorithm

In this section we present a method for using state information to recover lost bits from a source data stream. We incorporate this into a simple algorithm to demonstrate a way in which this idea could be used to build an MD code.

Encoder

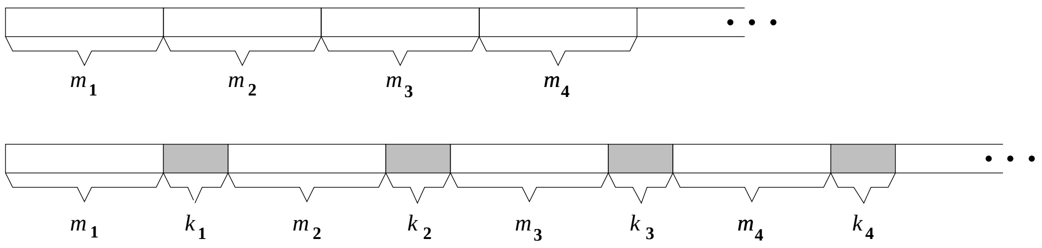

Figure 2: Redundancy is added to the original SPIHT bitstream.

In order to protect the output bits of the SPIHT encoder, redundancy is added. The redundancy consists of explicit descriptions of changes in the state information comprising the encoder and is added to the SPIHT bitstream as shown in Figure 2. For example, at the end of the first block of m_1 bits, a block of k_1 protection bits is added. This block contains information about state changes during the encoding of the previous m_1 bits. The protection block of k_2 bits corresponds to the previous block of m_2 bits, and so on. The numbers m_i and k_i can take arbitrary values, depending on the amount of state information to be described (or, equivalently, the amount of protection required). Appropriate values for m_i and k_i to be used when encoding an image are found by optimizing these values on a training set of similar images. The state information used for the redundancy is the number of list elements added (to the LSP in both LIP and LIS passes, or to the LIP in the LIS pass) or the number of elements removed (from the LIP in the LIP pass) or both numbers (for LIS). After protection, the bitstream is packetized, as discussed in the following section.

Packetization

The packetization method produces an MD code while preserving the multi-resolution property of SPIHT. We first deinterleave the SPIHT bitstream to separate the bitstreams associated with different wavelet coefficient trees (zerotrees), since each of these bitstreams can be encoded independently. The number of trees is equal to the number of the coefficients in the highest level of decomposition of the wavelet transform, since the zerotree structure used is that of [4]. Next, these bitstreams are further divided into $N+1$ substreams, each corresponding to one bitplane of the binary description of the wavelet coefficients in a single zerotree. Here, N represents the index of the most significant bitplane. Each of the newly formed substreams corresponds to a different bitplane, with index $n = N, N-1, N-2, \ldots, 0$. Bitstreams from different trees that correspond to the same n are aligned horizontally, and packetized vertically, as shown in Figure 3: the first packet is formed from the first bit from each bitstream, the second packet is formed from the second bit of each bitstream, and so on. Note, however, that the bitstreams are not of the same lengths. So, once the last bit from the shortest bitstream is put in the corresponding packet, this method must be modified. To deal with different packet lengths, we first calculate the average number of bits of the substreams that correspond to the same bitplane. We then cut the tails of those substreams that have longer than average lengths and append those tails to the shorter ones. This equalizes the lengths of the substreams for transmission.

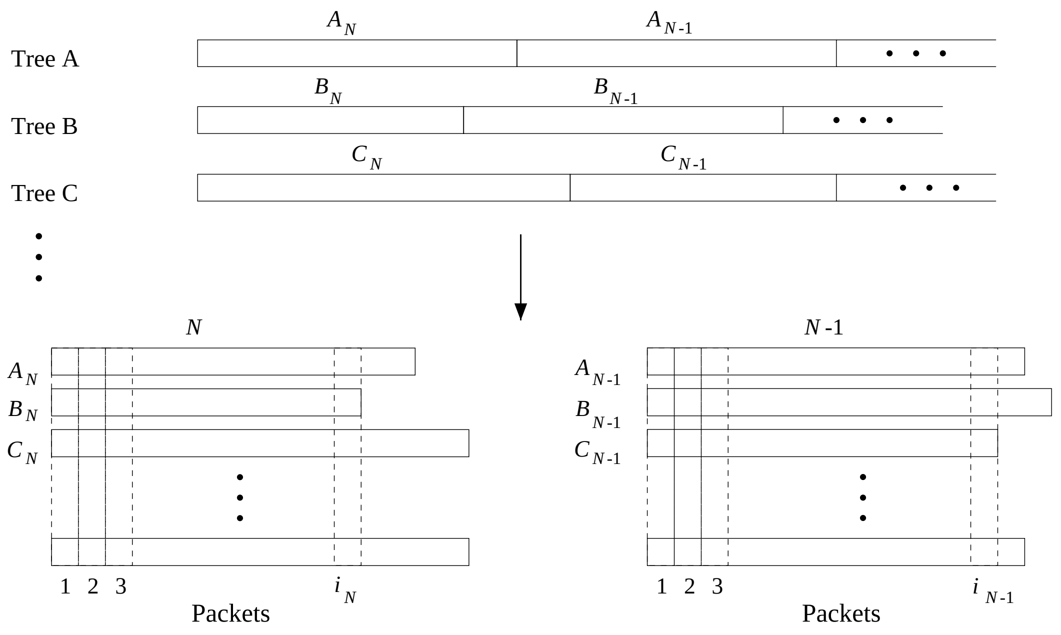

Figure 3: Packetization of bitstreams from different trees for the two highest values of n. Packets 1 through i_n (i_n is the length of the shortest bitstream) are formed in a straightforward fashion. Bitstreams are rearranged in order to form the rest of the packets.

Given this approach, the length of each substream needs to be known for the packetization process to be reversed once the packets reach the receiver. To this end, we add bits to the start of each substream to specify that substream's length. Since the loss of this information would leave some information undecodable, we aggressively protect it through FEC coding. The length information can be protected heavily since the overhead involved is a very small percentage of the total bitstream.

This approach provides us with the packets sorted in an embedded fashion, since the groups of packets are arranged and transmitted bitplane by bitplane, from most significant to least.

In the decoder, the bitstreams are reconstructed using the received packets. The depacketization method is shown in Figure 4; it is simply the reverse of the procedure used for packetization. The tails of the bitstreams are first rearranged into their original positions; this is possible since the exact lengths of the bitstreams are known to the decoder due to the aggressively protected length information. The packets are then aligned vertically (lost packets are colored gray) and the ith bitstream is made of the ith bit from each of the packets. Lost packets are transformed into lost bits in the bitstreams. Except for these lost bits, the depacketized bitstreams are identical to the original ones.

Decoder

The SPIHT decoder is adapted in order to recover lost bits using the state information added during the encoding process. The state information describes the lengths of the LIP, LSP, or LIS at the point of the encoding where they were added into the

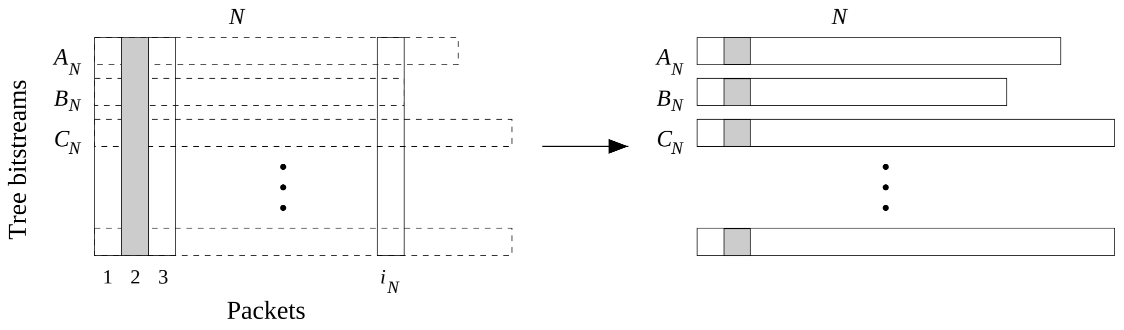

Figure 4: Depacketization of the bitstreams. The lost packets (the shaded regions on the left) are transformed into the lost bits (the shaded regions on the right)

original SPIHT bitstream. The encoder and the decoder should have identical lists at each step of the algorithm. When lost bits are encountered in a block, they are replaced with values that ensure that the decoder lists are consistent with the state information at the end of the block. (If the decoder lists are not consistent with the state information at the end of a block, then the decoder must be in a different state than the encoder was at that position. This would imply that at least one of the lost bits has been replaced incorrectly.) The approach is summarized in the following algorithm:

Start at block $i = 0$. For each block i do:

1. Beginning with the all-zero combination, try each possible combination for the lost bits. Replace the lost bits with the first combination of bits that is found to satisfy the state information check.

2. After replacement, increment i.

3. If there is no bit combination that satisfies the state information check, decrement i and continue searching for valid bit combinations, starting this time from the combination directly after the one that was previously accepted.

The algorithm is illustrated in the example shown in Figure 5. In this example, three bits in block i have been lost during transmission. The lost bits are colored in light gray, and the redundant bits in dark gray. We start checking bit combinations from ’000’, and continue until we find the first one that satisfies the state information check; the lost bits are then replaced with this combination. In the example, this bit combination is ’010’. It is important to note that this combination is not necessarily the correct one, it is just the first that was found to satisfy the state information check. Nevertheless, after this replacement we increment i and move to the next block. If the combination ’010’ was not the right one, then it is very likely that in the next block (as in this example) or in one of the subsequent blocks, there will be no combination of bits that satisfies the state information check for that block. When this happens, we know that an incorrect choice has been made in one of the previous blocks. The decoder goes back to the previous block (i is decremented) and continues to search for a different valid bit combination, starting this time from the first bit

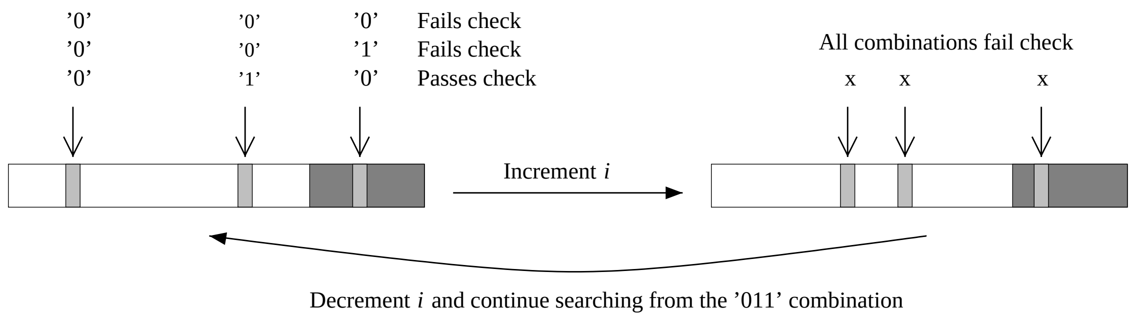
'0' '0' '0' Fails check
'0' '0' '1' Fails check
'0' '1' '0' Passes check
All combinations fail check
x x x
Increment i
Decrement i and continue searching from the '011' combination

Figure 5: Example of decoding and recovering bits. The lost bits are first replaced with the '010' combination, but since no combination is satisfactory for the next block, the decoder backtracks and continues checking with the '011' combination.

combination that has not yet been checked; in this example, the '011' combination. Once again, as soon as the next valid bit combination is found, the lost bits are replaced, i is incremented, and so forth. It is possible that a decoding error occurred in a block prior even to this one, yet passed all the intervening information checks; in such a case the decoder may have to backtrack multiple times.

One of the important features of the decoding process is that the redundant information for a given block can allow correction of an error from any earlier block. This differs from forward error correction, in which the redundant information for a specific block can be used to correct errors in that block only.

The algorithm is guaranteed to find a combination for all of the lost bits that will satisfy the state information check for every block; we know that such a combination exists because the original bitstream contained such a combination. However, if this combination is not unique, then the decoder may produce an incorrect reconstruction. Both the average and worst-case complexity of the algorithm depend on the level of detail contained in the state information. If the information allows a unique determination of the lost bits for each block, then the complexity is linear in the number of blocks, but if the information allows multiple combinations of bits to be accepted for every block, then the complexity will grow exponentially with the number of blocks. Increasing the number of lost bits is likely to increase the number of combinations that are accepted for each block, hence increasing complexity, but the exact relationship will depend on the particular image being described. Thus the complexity is higher than that of the algorithm in [2], which takes time linear in the number of packets used. However, the running time is still typically less than 1 second on a Pentium 200.

4. Results

The following results were obtained using a 5-level pyramid wavelet transform constructed with the 9/7-tap filters of [5] on monochrome, 8 bits per pixel, 512×512 standard images, at a total rate (including redundancy) of 0.201 bits per pixel. The algorithm was trained on the Lena image and tested on the Goldhill image. The training determined the block length and amount of state information to be added

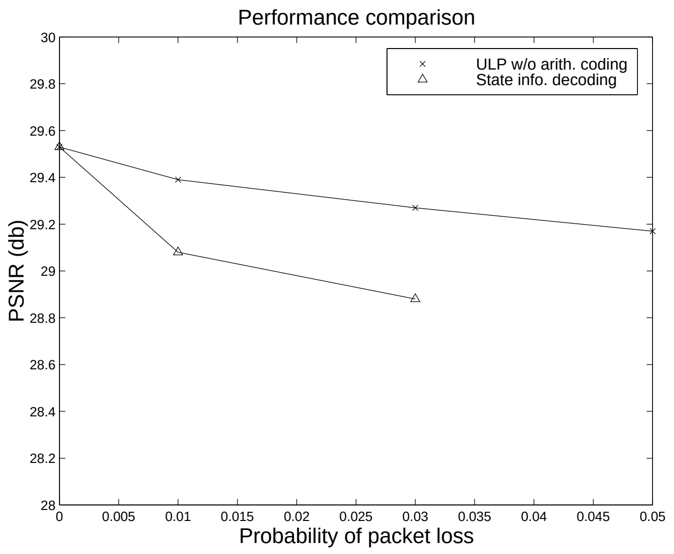

Figure 6: The PSNR (in dB) achieved by the two methods at different packet loss probabilities.

by the encoder, and these values were fixed at both the encoder and the decoder.

Results are presented at packet losses ranging from 0 to 3%. For simplicity, we assume that each packet has the same probability p of being lost.

The performance of the new method is compared with that obtained by the ULP scheme of [1] applied to the output of the SPIHT algorithm. Note that the performance of the ULP scheme could be increased by approximately 0.3 dB if the output of the SPIHT algorithm is arithmetically encoded before protection; this was not done here so that the efficiency of correcting lost bits via state information could be compared directly to that of correcting them via direct forward error protection.

Figure 6 shows the experimental performance in PSNR. The PSNR is calculated as $\mathrm{PSNR} = 10\log_{10}(255^2/\mathrm{MSE})\,\mathrm{dB}$, where MSE denotes the mean squared-error between the original and the reconstructed images. We see that the the simple state information based algorithm achieves performance close to that of the more sophisticated ULP scheme, although the performance gap increases as the probability of packet loss increases. At 3% loss, the performance gap is 0.35 dB.

5. Summary

We consider the question of how to incorporate state information from an embedded source coder together with the source data to create a multiple description code. The motivation for this consideration is that, in theory, a code that does not obey the separation principle could achieve better performance than one that does. Thus, the use of state information may allow the development of a code with performance surpassing that of existing separation-based schemes. However, the optimal way to use state information in such a code is not obvious. As a first step towards solving this problem, we develop a method by which source data bits lost in transmission can be recovered through the use of partial state information. As an illustration, we develop a simple multiple description code in which correction of lost source data bits is performed using state information. A comparison of this method to that of directly protecting the source bits using FEC techniques, as in unequal loss protection schemes, shows that the simple code achieves within 0.35 dB of the more sophisticated FEC-based scheme for low probabilities of packet loss. This suggests that more sophisticated approaches using state information, perhaps employing some of the ideas behind unequal loss protection algorithms, might yield very efficient multiple description codes. More work is required to discover the most effective use of state information.

6. Acknowledgments

The authors would like to thank Alex Mohr for providing experimental results using his unequal loss protection simulator, his discussion of the results, and his permission to reproduce an illustration from one of his published works.

References

[1] A. E. Mohr, E. A. Riskin, and R. E. Ladner. Graceful degradation over packet erasure channels through forward error correction. In *Proceedings of the Data Compression Conference*, pages 92–101. IEEE, March 1999.

[2] R. Puri and K. Ramchandran. Multiple description source coding using forward error correction. In *Conference Record, Thirty-Third Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, October 1999.

[3] A. Said and W. A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3):243–250, June 1996.

[4] J. M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41:3445–3462, December 1993.

[5] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies. Image coding using wavelet transform. *IEEE Transactions on Image Processing*, 1:205–220, April 1992.