

# Continuous Error Detection (CED) for Reliable Communication

Raghavan Anand, Kannan Ramchandran, and Igor V. Kozintsev, *Member, IEEE*

**Abstract**—Block Cyclic Redundancy Check (CRC) codes represent a popular and powerful class of error detection techniques used almost exclusively in modern data communication systems. Though efficient, CRCs can detect errors only after an entire block of data has been received and processed. In this work, we exploit the "continuous" nature of error detection that results from using arithmetic codes for error detection, which provides a novel tradeoff between the amount of added redundancy and the amount of time needed to detect an error once it occurs. We demonstrate how this continuous error detection framework improves the overall performance of communication systems, and show how considerable performance gains can be attained. We focus on several important scenarios: 1) automatic repeat request (ARQ) based transmission; 2) forward error correction (FEC) frameworks based on (serially) concatenated coding systems involving an inner error-correction code and an outer error-detection code; and 3) reduced state sequence estimation (RSSE) for channels with memory. We demonstrate that the proposed CED framework improves the throughput of ARQ systems by up to 15% and reduces the computational/storage complexity of FEC and RSSE by a factor of two in the comparisons that we made against state-of-the-art systems.

**Index Terms**—Arithmetic codes, automatic repeat request, decision feedback equalizers, error detection coding, forward error correction.

## I. INTRODUCTION

CONVENTIONAL communication systems use block Cyclic Redundancy Checks (CRC) for error detection. Since CRCs operate on blocks of data, they can detect errors only after an entire block of data has been processed. An error detection scheme that is "continuous" can detect errors while the block is being processed. Thus, it can enhance the performance of communication systems, through increased throughput of data transmission systems, as well as reduced complexity of sequence estimation—based receivers. In this work, we describe a new class of "continuous" error detection techniques and show its utility in a variety of common communication scenarios such as:

- 1) automatic repeat request (ARQ)-based data transmission;
- 2) (serially) concatenated coding systems employing an inner error-correction code and an outer error-detection code;
- 3) reduced state sequence estimation for channels with memory.

Our proposed approach is based on arithmetic coding, which is a popular source (entropy) coding technique [1]. During our research, we became aware of the fact that the idea of continuous error detection based on arithmetic coding had been proposed earlier by Boyd *et al.* in [2]. We undertake a more rigorous analysis of this approach, quantifying the underlying tradeoffs involved in the process and also establish the impressive gains in system performance attainable through integration of this novel method into different communication systems.

The basic idea is simple: add to the list of data symbols to be arithmetically encoded an extra "forbidden" symbol that is never actually transmitted. However, a controlled amount of probability space is reserved for it, nonetheless. (Note that this increases the redundancy of the coded symbol stream.) By increasing the amount of coding space that the forbidden symbol occupies, it is possible to make statistical guarantees about *where the errors may have occurred*. That is, it is possible to isolate the location of the error in a statistical sense, to any desired confidence level, to the previous  $m$  bits, where  $m$  depends on the amount of invested excess redundancy and the desired confidence level.

It is interesting to note that the error detection performance is independent of whether the errors are random or bursty and depends only on the amount of redundancy introduced. Thus CED can be used for communication over additive white Gaussian noise (AWGN) channels, or fading channels, or in conjunction with communication systems with error propagation at the receiver, like the decision feedback equalizer with CED as an outer error detecting code.

CED can be of great use in several communication systems where there is state space explosion, such as in sequence estimation-based receivers. In this work, we would like to present CED as a new approach to be incorporated into communication systems which can improve the throughput in some scenarios as well as reduce the complexity in some others. CED can be approached in other ways as well without using arithmetic coding. For example, a high rate convolutional code that introduces redundancy in a periodic fashion in a bitstream can also detect errors based on illegal state transitions. Another example would be a cyclic redundancy check code used periodically. In these examples of traditional error control codes, however, error detection can only occur at instances corresponding to parity check

Paper approved by R. D. Wesel, the Editor for Coding and Communication Theory of the IEEE Communications Society. Manuscript received July 16, 1999; revised August 28, 2000, and November 29, 2000. This paper was presented in part at the IEEE Data Compression Conference, Snowbird, UT, March 1998, at the IEEE International Conference on Communications, New Orleans, LA, June 2000, and at the Asilomar Annual Conference on Signals, Systems and Computers, Pacific Grove, CA, November 1997.

R. Anand and K. Ramchandran are with the Electrical Engineering and Computer Science Department, University of California at Berkeley, Berkeley, CA 94720 USA (e-mail: anand@eecs.berkeley.edu; kannanr@eecs.berkeley.edu).

I. V. Kozintsev is with the Microcomputer Research Labs, Intel Corporation, Santa Clara, CA 95052 USA (e-mail: igor.v.kozintsev@intel.com).

Publisher Item Identifier S 0090-6778(01)08176-4.

symbols<sup>1</sup> and in this sense is not really continuous. In the proposed arithmetic coding-based CED approach, encoded bits are both data and parity bits and thus allow error detection to occur virtually at any instance. Even at its best, convolutional or cyclic codes cannot provide the continuously tunable redundancy that comes with arithmetic coding, where the redundancy can be any real number. Therefore, the focus of this paper is to present performance improvements in different communication systems that use CED, over conventional CRC-based systems. We will use arithmetic coding as a vehicle to demonstrate the power of continuous error detection, though the CED framework is general enough to encompass other schemes such as the one based on convolutional coding as discussed above.

In an ARQ system, the utility of CED comes from the ability to make statistical guarantees pertaining to the time of error occurrence, once the error is detected. This results in potential savings in the number of bits that need to be retransmitted when an error is detected. In this setting, we optimize the tradeoff between added redundancy and error-detection time to attain throughput gains of up to 15% compared to a fully optimized conventional ARQ system.

In a serially concatenated coding system using inner convolutional codes, CED can likewise be useful in eliminating invalid trellis paths, leading to potential performance gains. The ability to dynamically prune inadmissible subpaths in a “list” Viterbi algorithm can be exploited to increase the likelihood of retaining the correct path in the final list. Our approach reduces the number of paths which have to be processed and stored in the list Viterbi algorithm by a factor of *two* compared to the number of paths needed with block CRC detection for the same overall performance.

In reduced state sequence estimation schemes, there is a limit on the number of paths that can be maintained. CED allows for the checking of the validity of paths on the fly in the reduced state trellis and pruning the incorrect candidates earlier. This increases the likelihood of the correct path being present at the end. As in the concatenated coding scheme mentioned above, CED allows for a reduction in the complexity/storage requirements by a factor of two in the state-of-the-art system that we considered.

The rest of the paper is organized as follows. Arithmetic coding is introduced in Section II and details of how it can be used for CED are discussed. Section III presents an application of CED for ARQ transmission where it provides significant throughput gains over conventional CRC-based schemes. In concatenated coding systems, using CED as an outer error detection code can significantly reduce the complexity of the system. This is discussed in Section IV. In Section V, we show how CED can reduce the complexity of reduced state sequence estimation by a factor of two over a conventional CRC-based scheme. We conclude in Section VI.

## II. ARITHMETIC CODING AND ITS APPLICATION FOR CED

Arithmetic coding [1], with a suitable source model, has been widely accepted as the optimum method for data compression

<sup>1</sup>It is always possible to partition the coded bits into information content bits and parity check bits.

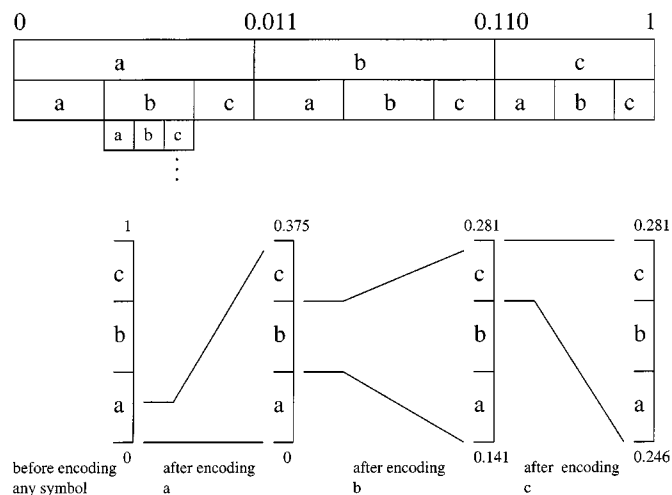


Fig. 1. An example of arithmetic coding: The source symbols are  $a, b, c$  with  $p(a) = 0.011, p(b) = 0.011, p(c) = 0.010$ .

for achieving the entropy of the source. Here data strings are mapped onto code strings that represent the probabilities of the corresponding data strings. To realize this mapping, some model has to be assumed for the source, which could be changed adaptively in practical situations. Based on this model for symbol probabilities, the coding algorithm progresses in a symbol-wise recursive fashion. On each recursion, it partitions an interval of the number line  $0 < a \leq b < 1$  and retains one of the partitions as the new interval. Thus, the data string gets mapped onto a fractional number between zero and one.

This can be illustrated better with an example. Consider a source alphabet with three symbols  $a, b$ , and  $c$  with  $p(a) = 0.011, p(b) = 0.011$ , and  $p(c) = 0.010$ , where the probabilities are all in binary representation (Fig. 1). Let us consider encoding a sequence  $abc \dots$ . After encoding the first symbol  $a$ , the transmit sequence would lie between 0 and 0.375 (0.011 in binary). At this point, the encoder can release the first encoded bit: “0.” The second symbol confines the transmit sequence to the range  $((0.375)(0.375), (0.375)(0.375) + (0.375)(0.375))$ , i.e.,  $(0.1406, 0.2812)$ . Each symbol encoded reduces the interval in which the transmit sequence would lie. As the number of symbols increases, we get a fractional number between zero and one that represents the probability of occurrence of that particular sequence of symbols.

The probabilities of symbols are allowed to vary in the process of encoding with the single restriction that they can only depend on past data to ensure that the decoder can also track them. It is actually the model adaptation which contributes the most to the complexity of the practical arithmetic coders and makes the term “arithmetic coding” synonymous with “computationally expensive.” However, if the probabilities of symbols are fixed through the encoding process, then the computational burden is quite insignificant. Normally only two registers are required for a data string with a few real operations per symbol [3]. This complexity is for the updates for the size and the beginning pointer for the current interval, which require two multiplications. It has been shown in [4] that each multiplication can be closely approximated by a shift and an add, thereby greatly reducing the complexity of each

update to two shifts and adds instead of two multiplications. These results are used in this paper to justify the claims about complexity/storage requirements of our proposed systems since they need no modification of probability models based on the previous data during encoding. We restate the fact here that the arithmetic coder is not being used as a source coder, and so we do not have symbol probabilities that are changing, to which the coder has to adapt. We do not consider using the arithmetic coder as a joint source-channel coder in this work, though that is a straightforward extension of the concepts presented here. Some work in this direction was presented in [5].

#### A. Incorporating Error Detection Into Arithmetic Coding

The arithmetic decoder is able to perfectly reconstruct the transmitted data by reversing the encoding operations. However, an error in an arithmetically coded stream often causes a loss of synchronization, and all subsequently decoded symbols become invalid. It is this loss of synchronization that we wish to exploit to give us error detection capabilities. The idea (first proposed in [2]) is to introduce a forbidden symbol that is never encoded by the arithmetic coder, but is nonetheless assigned a nonzero probability. Upon decoding, if an error occurs, this forbidden symbol is likely to be eventually decoded with very high probability. This is because loss of synchronization corrupts the subsequent decoded stream so that it can lie anywhere in the interval  $(0, 1]$ , which means that the forbidden symbol, that has a finite probability assigned to it, can be decoded with some nonzero probability. Investigation of the conditions under which the detection of errors is guaranteed is not undertaken in this paper. What is exploited in this work is the fact that, if the forbidden symbol is decoded, this guarantees that an error has surely occurred. The amount of time needed to decode the forbidden symbol after the occurrence of an error is inversely related to the amount of added redundancy due to the introduction of the forbidden symbol.

To understand this better, consider the case where there are only two symbols (see Fig. 2): a “useful” symbol  $a$  and the forbidden symbol  $x$ . Suppose that  $x$  is assigned probability  $\epsilon$  ( $0 < \epsilon < 1$ ) and  $a$  is assigned probability  $(1 - \epsilon)$ . Each time a data symbol  $a$  is encoded, the subinterval corresponding to  $a$  (with current uncertainty interval  $\alpha$ ) will be partitioned as follows:

- $(1 - \epsilon)\alpha$  of the subinterval is allocated to  $a$ ;
- $\epsilon\alpha$  of the subinterval is allocated to  $x$ .

Because the forbidden symbol is never encoded, the subintervals corresponding to  $x$  are never partitioned. Then, after encoding  $n$  symbols, the length of the interval containing valid codewords is decreased by the factor  $(1 - \epsilon)^n$ . If an error is introduced in the encoded stream, the decoded codeword will change its position on the interval. This change in position is well modeled as being uniformly distributed on the interval  $[0, 1)$ . This is because the errors would be totally random for any good compressor, and so any error is equally probable, as all correlations in the source stream would have been removed by the arithmetic encoder. If we then define a random variable  $Y$  that represents the number of bits it takes to detect an error after it occurs, then  $Y$  can be modeled as having a geometric distribution

$$P_y(k) = (1 - \epsilon)^{k-1}\epsilon, \quad k = 1, 2, \dots, \infty. \quad (1)$$

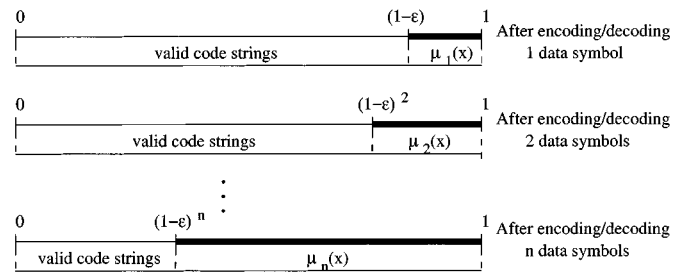


Fig. 2. The proper divisions of the unit interval  $[0, 1)$  when encoding the symbol  $a$  and adding a symbol  $x$  which is never encoded. The probability of  $x$  is  $\epsilon$  and the probability of  $a$  is  $(1 - \epsilon)$ . The subintervals representing data symbols that contain an  $x$  are denoted as  $\mu_n(x)$ .

When an error occurs, the bitstream is no longer restricted to be in the subintervals that belong to the data symbols, but will now have an  $\epsilon$  probability of falling in the subinterval corresponding to  $x$  (i.e., decoding the symbol  $x$ ) and a  $(1 - \epsilon)$  probability of falling in the valid subinterval corresponding to  $a$ .<sup>2</sup> From this, we see that

$$P[Y > n] = (1 - \epsilon)^n. \quad (2)$$

Letting  $\delta = P[Y > n]$ , taking the logarithm of both sides of (2) and solving for  $n$ , we obtain

$$n = \frac{\log_2(\delta)}{\log_2(1 - \epsilon)}. \quad (3)$$

What (3) says is that, given that an error happened,  $\delta$  denotes the confidence measure corresponding to the fact that the error happened in the previous  $n$  bits, for a given  $\epsilon$ . There is still a small probability that, given that an error occurred, the error pattern will result in the original code word being mapped to another valid code word, so that the error is never detected. This is a rare event with insignificant probability, which never occurred in the different simulations that we conducted.

At this point, some observations regarding burst error detection properties of the CED scheme have to be mentioned. This is of particular interest when considering error detection times for fading channels which cause bursty errors in the transmitted data, or when CED is used as an outer error detection code with an inner decoding scheme that could have error propagation, like the decision feedback equalizer. Analysis of detection times for burst errors is not very straightforward as the effects of loss of synchronization depend on the particular symbol sequence and the particular burst of errors, and a general characterization would be difficult to make. However, it has been observed from simulations that the position of detection of error is independent of the burst length. The histogram for error detection position was similar for burst lengths ranging from 1 to 50 bits for a frame of 2000 bits that we ran our simulations for.

This can be intuitively understood as follows. Once an error happens in a stream encoded using arithmetic coding, the received codeword can lie anywhere in the  $[0, 1)$  interval with equal probability [6]. The only way in which an error burst could

<sup>2</sup>Simulations using various sizes of  $\epsilon$  confirm that the geometric distribution is a valid model.

affect this is if it could cause a resynchronization of the erroneous codeword into another valid codeword. This is a very rare event, and the probability of its happening within a finite frame size is quite small. Hence the position of detection of the error depends almost entirely on where the first error happened. This helps us detect burst errors of different lengths with the same chances of success or failure as detecting a single error.

It is also worth mentioning that a popular block-based Huffman (prefix) coding for data compression can be viewed as a special case of arithmetic coding. Thus, our results may also be extended to include Huffman coding, though the continuous nature of error detection will, in general, be lost. (The extension is obvious: if a Huffman decoder receives a prefix that does not correspond to any codeword from the alphabet, an error has surely occurred during the transmission.)

### B. Redundancy Versus Error Detection Time

Having established the error-detecting capabilities of introducing a forbidden symbol upon encoding, the next task to consider is the price (in terms of extra bits) that is paid for introducing this forbidden symbol. To determine the coding inefficiency resulting from the presence of the forbidden symbol, we note that typically  $-\log_2(\gamma)$  bits are needed to represent a subinterval of width  $\gamma$  on the unit interval  $[0, 1)$ . By introducing the forbidden symbol  $x$  that occupies a subinterval of width  $\epsilon$ , the rest of the data symbols will be confined to a subinterval of width  $1 - \epsilon$ . As a result, the amount of redundancy  $R_x$  due to adding the forbidden symbol  $x$  is

$$R_x = -\log_2(1 - \epsilon) \quad (4)$$

bits per symbol encoded. From (4) and (3), we obtain

$$n = -\frac{\log_2(\delta)}{R_x}. \quad (5)$$

Also, we see that the amount of redundancy added for a given “confidence level” (captured by  $\delta$ ) is inversely related to the amount of time it takes to detect an error. This is useful for ARQ where there are conditions in which fast detection is more important than the amount of redundancy added (noisy channel conditions) and there are conditions in which the opposite is true (calm channel conditions). From (5), we have direct *continuous* control over the amount of redundancy added versus the expected amount of time it takes to detect an error.

## III. ARQ WITH CED

In this section, we briefly mention the performance gains achieved by the application of CED for ARQ transmission. A more elaborate description can be found in [7].

To incorporate the new error detection scheme into ARQ, a few minor modifications need to be made to the conventional ARQ framework. The most critical change is that error detection is done continuously; as soon as the forbidden symbol is decoded at the receiver, a retransmission of  $n$  bits is requested, where  $n$  corresponds to a confidence level of  $(1 - \delta)$  as in (3). In our scheme, the packet size  $N$  is only significant in contributing to the amount of overhead  $\log_2(N)$  bits and may thus be chosen

independently of the channel conditions. This is not the case, however, for conventional ARQ, where  $N$  must be carefully chosen in accordance to the bit error probability of the channel in order to optimize throughput. To maximize throughput in our case,  $n$  (i.e., the number of bits to retransmit) must be optimally matched to the channel conditions.

### A. Throughput Analysis

Assuming a noiseless return channel and an infinite receive buffer, throughput analysis can be done for ARQ transmission over a binary symmetric channel (BSC), where the throughput is defined as the average amount of bits transmitted per channel bit. This can be written as [7]

$$T = \beta T_c. \quad (6)$$

The ratio of the expected number of bits successfully received to the expected number of bits transmitted on the channel,  $T_c$ , can be expressed as

$$T_c = \frac{p_b^2(1 - \epsilon)^{n+1} - \epsilon^2(1 - p_b)^{n+1}}{p_b^2 - \epsilon^2 + R_n \epsilon p_b (p_b - \epsilon)(1 - p_b)} \quad (7)$$

where  $p_b$  is the bit error probability of the BSC and  $R_n$  is the total overhead bits.

The fraction of useful data contained within each bit after parity  $\beta$  can be expressed as [7]

$$\beta = \frac{H(p_s)}{H(p_s) - \log_2(1 - \epsilon)} \quad (8)$$

where  $H(p_s)$  represents the entropy of the source with source probabilities  $p_s$ .

Simulations confirm that the throughput (6) models the actual throughput very well, as will be seen next.

### B. Results for ARQ With CED

Simulations were run using a BSC model at various bit-error probabilities with multisymbol data alphabets. One thousand packets 10 kbits in size were sent at each bit-error probability, and the resulting throughput was calculated. As a measure of performance, we compared our method of ARQ to conventional methods of ARQ. In an attempt to make the comparison between our method and the conventional ARQ methods fair, in our simulations the optimal packet size<sup>3</sup> was used at each of the bit-error probabilities tested using conventional ARQ; the optimal  $\epsilon$  was also found and used for each of the bit-error probabilities tested using our new method of ARQ. The resulting throughputs are shown in Fig. 3.

From the figure, we see that the new method of ARQ outperforms conventional ARQ methods at all bit-error probabilities.<sup>4</sup> Furthermore, our new method of ARQ is well suited for time-varying channels, because  $\epsilon$  can be *continuously adapted as a function of the channel conditions*.

<sup>3</sup>The optimal packet size for conventional ARQ methods is found by solving for the packet size which maximizes the throughput [8].

<sup>4</sup>Throughputs for bit-error probabilities greater than  $p_b = 10^{-3}$  are not shown in the figure, because ARQ is no longer practical at such high bit-error probabilities.

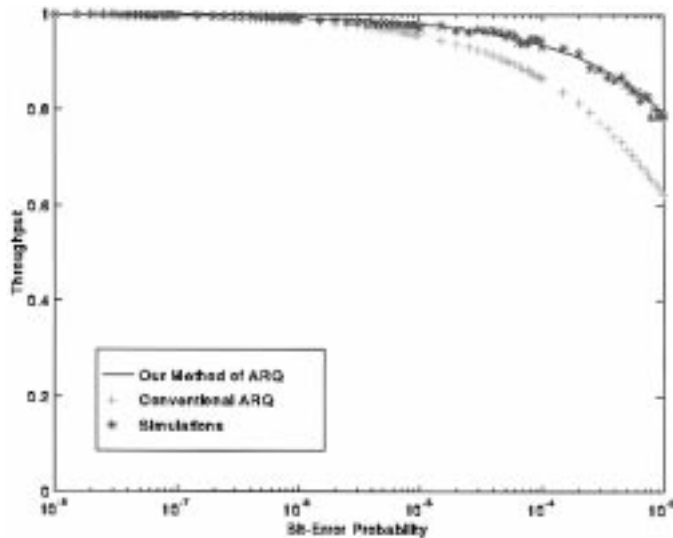


Fig. 3. Throughput curves of our new method of ARQ versus conventional ARQ at various bit-error probabilities.

#### IV. CED FOR CONCATENATED CODING SYSTEMS

Concatenated coding schemes for forward error control have been extremely popular due to their high performance and the existence of efficient decoding algorithms. In this work, we consider serially concatenated coding schemes with an inner convolutional coder and an outer error detection coder and show how CED can be applied to significantly reduce the complexity of such systems.

Suppose that a data sequence  $\{u_k\}$  is encoded using a convolutional encoder to generate the transmitted vector sequence  $\{y_k\}$ . This goes over a memoryless AWGN channel and is received as  $\{z_k\}$  at the receiver. The Viterbi Algorithm (VA), originally proposed in [9], finds the maximum likelihood data sequence  $\{\hat{u}_k\}$  from the received vector  $\{z_k\}$ . Further details on the VA may be found in [9].

Despite the fact that the VA in its original form produces a maximum likelihood estimate of the transmitted codeword, this estimate may be incorrect. By using an additional error-detecting channel code (typically CRC), it is possible to tell with a very high confidence level whether the survivor path is error-free at the price of an insignificant increase in the overall code rate. By making appropriate modifications, the Viterbi algorithm can be changed to output an ordered list of the “ $N$ -most likely” transmitted codewords (for a list- $N$  VA), i.e., instead of finding the best path we require a list of the  $N$  best paths, resulting in considerable improvements over the original VA. An example of such modification is called the “parallel” list VA [10]. In the parallel list VA, at every step of the algorithm, the list of  $N$  best paths coming into each state is maintained rather than a single path as in the conventional VA. Those  $N$  survivor paths are checked (in the order determined by their accumulated metrics) at the end of the data block using an error detection code until the first valid path is found.

The performance of the parallel list Viterbi decoding procedure can be shown to have a worst-case asymptotic gain of 3

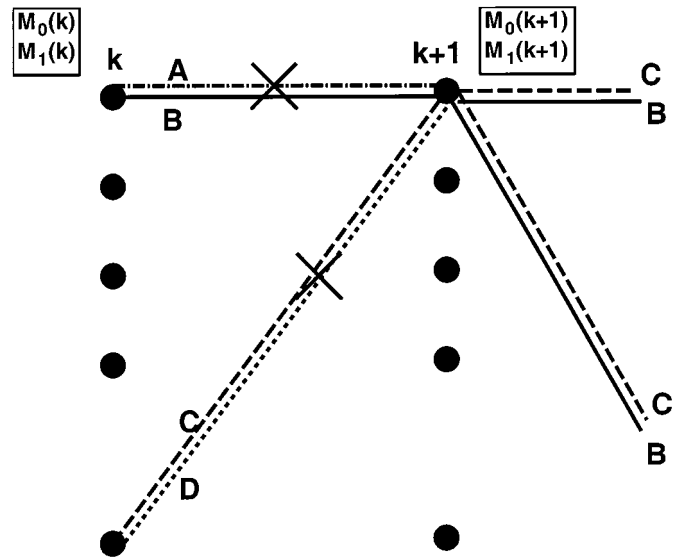


Fig. 4. Illustration of the parallel list VA for  $N = 2$  paths in the list. At time  $k + 1$  for each state,  $N$  survivor paths are chosen based on their accumulated metrics  $M_i(k + 1)$ . In the example, paths  $B$  and  $C$  are chosen and paths  $A$  and  $D$  are pruned.

dB over the original (i.e.,  $N = 1$ ) VA [10]. Note that the complexity of the decoder and its performance are proportional to  $N$ , and can be chosen depending on the tradeoff between complexity and performance. The parallel list VA is illustrated in Fig. 4 for  $N = 2$ . A more efficient “serial” implementation is also possible which stores all decisions and associated metrics along the trellis and allows for the request of additional path estimates only when needed. To summarize, the list VA algorithms allow a tradeoff of the complexity of the decoder for its performance. It is also true that most of the improvement in performance can be achieved by maintaining a list of three to five paths instead of one [10]. Further increases in the size of the list correspond to rather insignificant gains.

Our proposed scheme uses the same idea of producing a list of best path estimates as in [10]. However, the outer block error detection code (CRC) is replaced by a continuous error detection code. We assume that the total number of paths in the list ( $N$ ) is fixed, to address computational complexity/delay constraints, and we consider a parallel version (which is well suited for hardware implementations) of the list VA<sup>5</sup> as shown in Fig. 4. The motivation for using CED is that if channel errors occur at the beginning of the block, there is a good chance that this will render all the paths in the list- $N$  VA (for small  $N$ ) incorrect, resulting in an undesirable “no correct path” situation. The proposed CED scheme allows some incorrect paths to be pruned before the end of the block is reached, hence, increasing the probability that the correct path will survive. We illustrate our proposed scheme in Fig. 5. Note the presence of a feedback path from the arithmetic decoder to the list VA decoder.

The system operates as follows (see Fig. 4). At each state of the trellis,  $N$  best paths are chosen from among the incoming  $2N$  paths (we assume a binary alphabet as shown in Fig. 4).

<sup>5</sup>For fixed  $N$ , both parallel and serial implementations have the same performance.

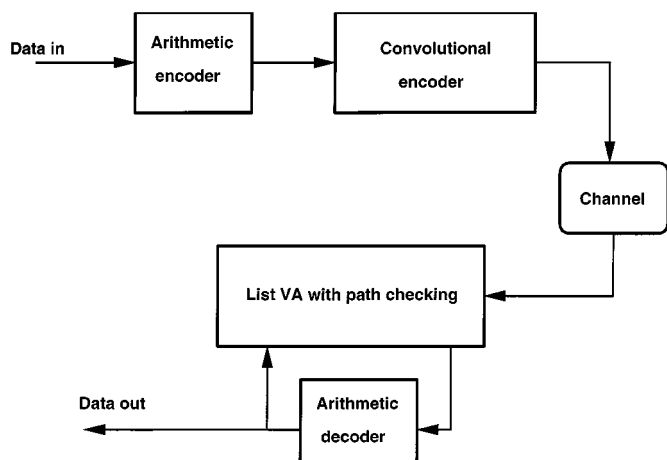


Fig. 5. The list VA with CED. All paths in the decoder are checked periodically (feedback from the arithmetic decoder to the list VA decoder) and if an incorrect path is found it is replaced by the valid path in the list.

The decision about the survivor paths is made based on both the accumulated metric and *the validity of the path*. In this case, if the arithmetic decoder flags a path down as wrong, the path is pruned and replaced by a “valid” path having the next best accumulated metric.

#### A. Results for List VA With CED

To assess the performance of the proposed modified list VA decoder, we compare it to the list VA scheme of [10] in terms of the complexity/storage requirements for target performance and signal-to-noise rate (SNR) gains. In all of our experiments, we set  $\epsilon = 10^{-2}$ , which resulted in approximately a 1.5% increase in the data rate: this is accounted for by appropriately reducing the SNR in the comparisons. We also chose to keep the same packet structure as used with CRCs for simulation purposes.

1) *Complexity/Storage and SNR Performance:* We compare the list VA with block CRC against the list VA with CED to see what savings in complexity can be achieved for the same target performance (frame error rate). The channel code used is a rate 1/3, memory 4, Rate Compatible Punctured Convolutional (RCPC) code [11]. The results are presented in Fig. 6. Note that the proposed system requires only about *half the list size* to achieve the same performance.

The storage requirements are significantly reduced in the proposed scheme since only a single register is needed to store the state of the arithmetic coder/decoder for each path, while the number of maintained paths is significantly lower. Computational complexity added by the arithmetic codec does not depend on the size of the data and can be addressed by efficient implementations, where each multiplication operation can be replaced by a shift and an add. This was mentioned in Section II.

The proposed scheme also offers gains in channel SNR for a target bit error rate (BER) when compared to the list VA with block error detection as illustrated in Fig. 7. The gains are up to 0.25 dB in channel SNR, nonnegligible considering that we target only error detection (with only 1.5% rate increase) and not error correction. For comparison purposes, we also show the performance of the original VA (single path in the list).

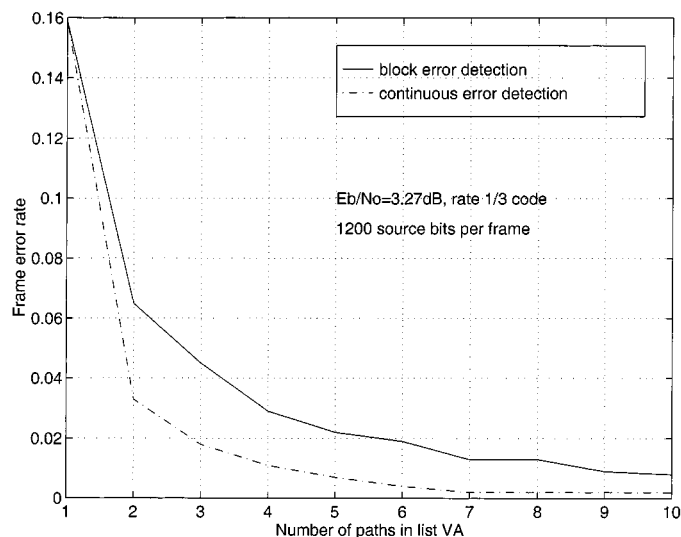


Fig. 6. Comparison of a frame error rate versus the number of paths for the reference list VA and the list VA with CED.

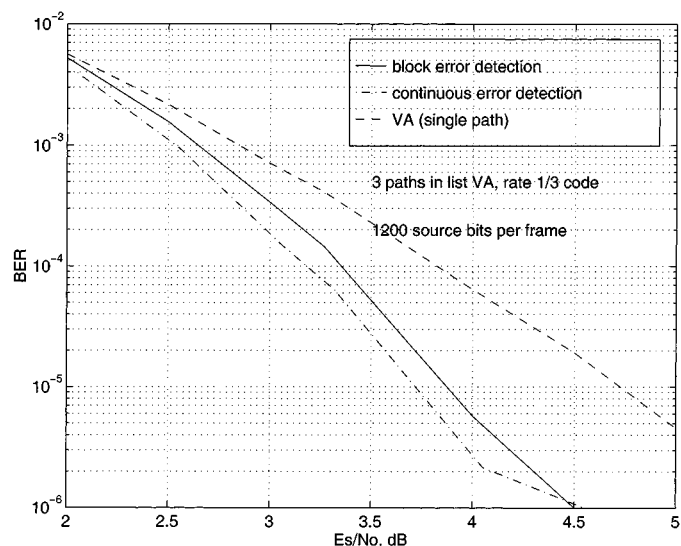


Fig. 7. Comparison of BER versus SNR for rate 1/3 memory 4 code for the reference list VA and the list VA with CED (using  $N = 3$ ).

#### V. APPLICATION OF CED FOR JOINT EQUALIZATION AND CODING FOR ISI CHANNELS

Combating intersymbol interference (ISI) is one of the main challenges in digital communication over band-limited channels. Several equalization techniques can be used to cancel out the resulting ISI.

While methods like decision feedback equalization (DFE) have the problem of error propagation, maximum likelihood sequence estimation (MLSE) has exponential increase in complexity with channel memory and so is difficult to implement. Reduced state MLSE schemes seek to reduce this complexity by choosing not to grow the full trellis based on some set of rules.

In [12], a novel scheme was presented that combined DFE with high rate error detection coding. Here, a list of possible paths are maintained by branching out from the conventional DFE output using estimated values of channel noise. At the end

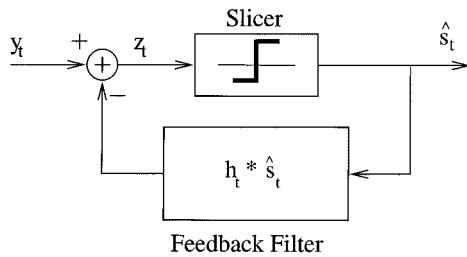


Fig. 8. The DFE model.

of a frame, a CRC is used to select the correct path from the list of paths. Our scheme integrates CED into that framework and shows how, by providing an effective way of continuously pruning erroneous paths, one can reduce the complexity of the scheme by a factor of two.

#### A. CRC-Based Approach

In this section, some details of the algorithm discussed in [12] that are relevant to our proposed scheme are presented.

The source data stream has taken  $k$  symbols at a time and encoded using a cyclic code to  $n$  symbols. Since the code is cyclic, it can detect all bursts of error of length  $n - k$  or less. Let  $s_1, s_2, \dots$  be the transmitted sequence,  $n_1, n_2, \dots$  be the channel noise samples, and  $h_0, h_1, \dots, h_M$  be the channel impulse response. The channel impulse response is assumed to be known, or reliably estimated, at the receiver. We can assume that  $h_0 = 1$ . (If it is not, then it can be normalized to unity.) Transmit symbols are drawn from a modulation alphabet of size  $|S| = q$ .

In Fig. 8,  $z_t$  is the signal at the slicer input, and  $\hat{s}_t$  denotes the estimated symbol. The output of the DFE, i.e.,  $\hat{s}_1, \hat{s}_2, \dots$ , will be referred to as the standard path. The modulation alphabet considered is pulse amplitude modulation (PAM) with equiprobable symbols taking on values in  $A = \{\pm 1, \pm 3, \dots, \pm(q-1)\}$ , where  $q = |A|$  is a positive even integer. A suitable estimate of the channel noise for this modulation scheme, required for the path allocation mechanism, can be computed as

$$\hat{n}_t = \begin{cases} z_t - \hat{s}_t, & \text{if } |z_t| < (q-1) \\ 0, & \text{otherwise} \end{cases} \quad (9)$$

A new path is opened in the trellis whenever this estimated value of the channel noise exceeds a certain threshold  $\tau$ . Paths already formed in the trellis also branch using the same threshold. Thus,  $\tau$  provides a tradeoff between performance and complexity. High values for  $\tau$  would mean that we would have very few paths in the trellis, and so the performance would approach that of the single zero-forcing DFE. On the other hand, lower values would mean that we are approaching the full MLSE trellis and we would have increasing complexity.

Errors can happen due to two different reasons. A *channel error* is said to have occurred at position  $t$ , if  $|n_t| \geq 1$  and  $|s_t + n_t| \leq q - 1$ ; and if  $\hat{s}_t \neq s_t$  we say that a *decision error* has occurred. Typically a channel error is followed by several decision errors, due to error propagation.

A path diverging from the standard path is called a *branch* and a path that diverges from a branch is called a *sprout*. Going beyond a depth of two in the DFE tree would give only small

performance gains at the expense of a lot of complexity [12]. So this algorithm does not look for paths beyond a depth of two in the DFE tree.

The key rules for branching are summarized below.

- 1) Branches are allocated at  $\mathcal{B}$  positions where the estimated noise amplitude has the greatest absolute value. We place a constraint that a maximum of  $\mathcal{S}$  sprouts can be allocated from branches at positions with largest noise amplitude.
- 2) Along  $\mathcal{B}^*$  branches with maximum noise amplitudes ( $\mathcal{B}^* < \mathcal{B}$ ),  $\mathcal{S}^* = \mathcal{S}/\mathcal{B}^*$ , sprouts are assigned to positions where estimated branch noise amplitudes have maximum absolute value. The other  $\mathcal{B} - \mathcal{B}^*$  branches are not assigned any sprouts.

If a path goes through  $b$  branches and  $s$  sprouts, the path is said to have  $b + s$  breakpoints. For a DFE tree with a depth of two, there can be a maximum of  $2\mathcal{B}$  breakpoints. A path with  $\delta = b + s$  breakpoints is the correct path only if  $\delta$  channel errors occur during the transmission. So, we see that a path with  $\delta + 1$  breakpoints is less likely to be the correct path than one with  $\delta$  breakpoints. The important rules for path selection at the end of the frame are as follows.

- 1) The paths in the DFE tree are processed in a fixed order to verify if they belong to the cyclic code, and the first one that happens to be in the code is selected as the estimated output sequence.
- 2) If a path belonging to the code is not found among the paths processed, the standard path is declared as the estimated sequence.
- 3) Only those paths that have up to  $\Delta$  breakpoints are processed (in the order of increasing  $\delta$ ).

#### B. Application of CED to the RSSE Framework

In this section, we present how CED can enhance the performance of the RSSE scheme described in the previous section. We refer to the original system as the CRC-based system, and the proposed system that uses CED to enhance the performance of the CRC-based system shall be referred to as the CED-based system. In this system, CED is used for path pruning, and CRCs are used at the end of the frame for selecting the best path.

The block diagram for the CED-based system is presented in Fig. 9. The DFE outputs estimates of the symbols at each instant. New paths are created on the trellis, based on the path generation mechanism described in Section V-A. Whenever all resources for branches and sprouts are used up and a valid candidate comes up, all the paths are checked against the arithmetic decoder and the flagged paths are removed, and if all paths through a branch or sprout are removed, then the resource is reallocated to the next deserving candidate.

How well can the CED-based scheme perform compared to the CRC-based scheme? Is it lower-bounded by the same curve as the CRC-based scheme? The lower bound discussed in [12] consists of the case where the standard path is in error and it goes undetected by the path selection scheme. This can happen under two different scenarios: there are one or two channel errors and the CRC does not detect the error, and the case where there are more than two channel errors. Since we do not keep paths with more than two breakpoints, our scheme will not be

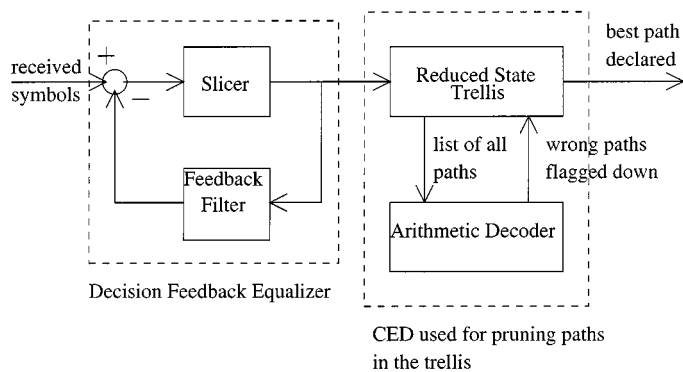


Fig. 9. System block diagram for the CED-based scheme.

able to capture the cases where there are more than two channel errors. This results in negligible loss of performance, but there is significant reduction in complexity [12].

A significant part of the errors are because of the fact that they would have required more than two breakpoints to capture them. Our modification by using CED only helps in reducing the errors because of error bursts that go undetected by the outer CRC code. Therefore, we are also essentially lower bounded by the same curve as the CRC-based scheme. The improvement is in terms of fewer branches (sprouts) required to achieve the same performance. The derivation of the improvement in the other two terms in the lower bound is interesting, though straightforward and is omitted for conciseness.

### C. Simulation Results and Discussion

Simulations were carried out for a block length of  $n = 256$  symbols. A four-level PAM alphabet  $A = \{-3, -1, +1, +3\}$  was employed with equal *a priori* probabilities of transmission. The number of break points ( $\Delta$ ) was chosen as two, as discussed in Section V-A. The SNR mentioned in the simulation results refers to the ratio of the average signal power to the variance of the Gaussian noise, as seen at the input to the slicer. The channel is a 2-kft-AWG26 channel, encountered in HDSL applications, with the following channel coefficients:  $h_1 = -0.6, h_2 = -0.15, h_3 = -0.12, h_4 = -0.05, h_5 = 0.00$ , and  $h_6 = 0.05$  [12].

We carried out two different kinds of comparisons. If we use an 8-bit CRC and 16 bits redundancy for the arithmetic coder, then we would effectively be using a code that uses 8 bits more per frame than the CRC-based scheme and would be of rate 0.954. With this code, we required only four branches and one sprout to achieve the same performance as the CRC-based scheme, and if we were to use the same number of branches as the CRC-based scheme, there was a modest performance gain of 0.5 dB. These results are shown in Fig. 10. The error rate shown is the block error rate, i.e., the probability that a frame was decoded incorrectly. Performance curves for  $\mathcal{B} = 10$  and  $\mathcal{S} = 4$  are shown for the CRC and CED-based schemes, as well as for  $\mathcal{B} = 4$  and  $\mathcal{S} = 1$  for the CED-based scheme, which matches the CRC-based scheme. The lower and upper bounds for the CRC-based scheme are also shown for comparison. The performance gains seem to increase as SNR increases. One reason for this could be the following. At low SNRs, the predominant

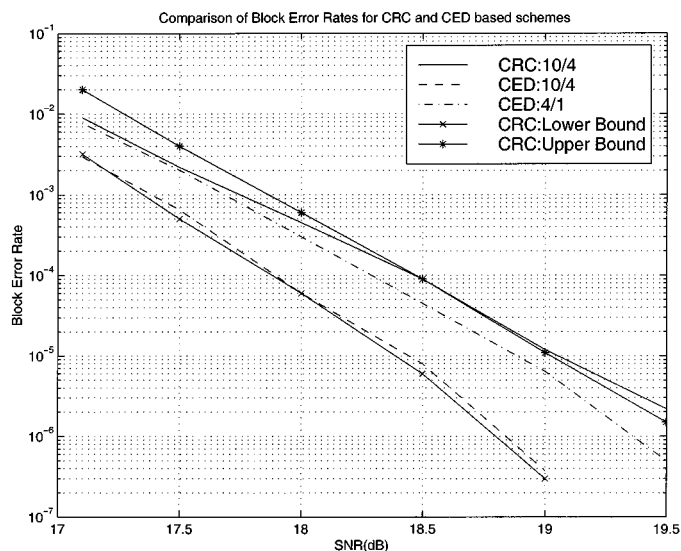


Fig. 10. Plot of block error rate versus SNR using code of rate 0.954.

factor in the block error rate would be the cases where two breakpoints were not sufficient to capture all the channel errors. Since we also use only two break points, the errors in the CED-based scheme are also mainly due to this term. But as the SNR goes up, quite a few of the errors in the CRC-based scheme would be because the correct branch was replaced by some erroneous branch with a higher noise amplitude. The CED-based scheme is able to remove most of those errors by being able to detect those paths to be in error, on the fly. Thus, it achieves more gains over the CRC-based algorithm at higher SNRs than at lower SNRs.

Since we use some redundancy for the arithmetic coder and some for the CRC at the end of the frame for path selection, comparison was also done with the CRC-based scheme assuming the same power per frame. In our case, the excess rate due to redundancy for the arithmetic coder was offset by lower energy per symbol so that the overall energy per frame would be the same as the CRC-based scheme. Simulations were carried out with ten branches and four sprouts as in [12] to see the gains, and also with four branches and one sprout to see the reduction in the resources required. These are shown in Fig. 11.

Here we should also mention the fact that an effective SNR-based comparison would be a pessimistic metric as far as performance of the CED-based scheme is concerned. More bits for a CRC would only make path selection more reliable. The same bits when used for CED can prune out wrong branches and increase the probability of finding the correct branch among the list of branches at the end of a frame. So, the bits serve different purposes and cannot be compared in a straightforward manner.

We observe that even with a pessimistic comparison, the CED-based scheme achieves the performance of a purely CRC-based scheme that uses ten branches and four sprouts, with about six branches and one sprout, which is still a complexity reduction by a factor of two (see Fig. 11).

1) *Complexity Versus Performance Gains*: An important issue to be discussed is the tradeoff of complexity versus performance gains. The implementation of the algorithm described in Section V-A requires that all the paths in the trellis

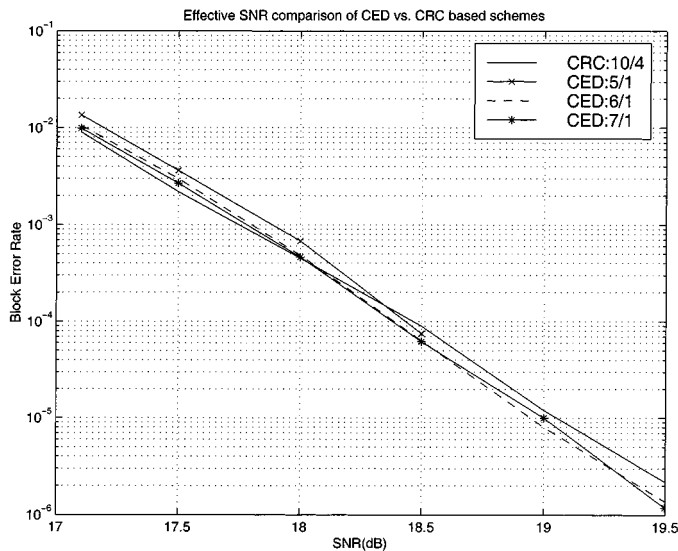


Fig. 11. Plot of block error rate versus SNR using code of rate 0.968.

be saved so that it is not required to search through the trellis and locate all paths when we perform error detection using the arithmetic decoder. Since many paths are flagged down by the CED-based scheme, the memory requirement for this is not very large. Typically, we need to store about 20 paths at worst and an average of 10 paths. This would have to be compared against the computational complexity required for a path search in a block-based scheme at the end of a frame, where the correct path might be found only after several paths have been invalidated by the path selection procedure as in Section V-A. On the other hand, at the end of the frame in the CED-based scheme, the correct path would be one of the first few in the list as most of the wrong paths would have been removed by then. Memory requirements for the CRC-based scheme would include keeping track of the nodes where branches (sprouts) diverge and remerge from the standard path (parent branch). It has already been mentioned in Section II that the complexity incurred by having to perform the arithmetic encoding would be two shifts and adds per symbol and a register to store the state of the arithmetic encoder.

Another question to be considered is the frequency with which we have to check the paths out against the arithmetic decoder. This is important when implementing a CED-based system in hardware, as it gives us an idea of how many arithmetic decoder sections to implement in parallel. It has been observed that, on an average, every path visits the decoder once in every three or four instants. Since we know the maximum number of paths, we can include these many decoders in the hardware implementation.

## VI. CONCLUSION

In this work, we have considered a new error detection scheme that performs well in different communication scenarios like ARQ-based communication systems, concatenated coding systems, and reduced state sequence estimation for channels with memory. It has the novelty of lending itself to

be physically combined with the source entropy encoder in a single device (i.e., the arithmetic coder). Significant gains of the proposed CED detection scheme over conventional block codes are demonstrated for both ARQ and forward error correction frameworks. These gains result from the ability of the new method to detect errors earlier than the end of the block and thus to be more responsive than block CRC-based schemes. In the reduced state sequence estimation framework, the new method gives us a means to allocate scarce resources in a better way by letting us know, on-the-fly, which of the paths are wrong, thereby enabling reallocation of resources to more deserving candidates.

We also mention here that CED can be put to good use to improve throughput performance of transport protocols like TCP over heterogeneous networks, where early detection of an error can result in a potentially greater number of retransmits, thereby increasing the probability of successful reception over a fading channel. This is currently being verified. The scenarios presented here are in no way exhaustive and CED could find applications in other frameworks also, like multiuser detection where there is a state space explosion. The goal of this work is to present the benefits that communication systems can derive from using CED for complexity reduction and/or throughput enhancement.

## ACKNOWLEDGMENT

The authors would like to thank J. Chou for all the encouraging discussions and for providing the results quoted in Section III.

## REFERENCES

- [1] G. Langdon, "An introduction to arithmetic coding," *IBM J. Res. Develop.*, vol. 28, pp. 135–149, Mar. 1984.
- [2] C. Boyd, J. Cleary, S. Irvine, I. Rinsma-Melchert, and I. Witten, "Integrating error detection into arithmetic coding," *IEEE Trans. Commun.*, vol. 45, pp. 1–3, Jan. 1997.
- [3] W. Pennebaker, J. Mitchell, G. Langdon, and R. Arps, "An overview of the basic principles of the Q-coder adaptive binary arithmetic coder," *IBM J. Res. Develop.*, vol. 32, pp. 717–726, Nov. 1988.
- [4] S.-M. Lei, "Efficient multiplication-free arithmetic codes," *IEEE Trans. Commun.*, vol. 43, pp. 2950–2958, Dec. 1995.
- [5] B. D. Pettijohn, K. Sayood, and M. W. Hoffman, "Joint source/channel coding using arithmetic codes," in *Proc. IEEE Data Compression Conf.*, Snowbird, Utah, 2000, pp. 73–82.
- [6] I. Kozintsev, J. Chou, and K. Ramchandran, "Image transmission using arithmetic coding based continuous error detection," in *Proc. IEEE Data Compression Conf.*, 1998, pp. 339–348.
- [7] J. Chou and K. Ramchandran, "Arithmetic coding based continuous error detection for efficient arq-based image transmission," *IEEE J. Select. Areas Commun.*, vol. 18, pp. 861–867, June 2000.
- [8] M. Schwartz, *Telecommunication Networks: Modeling and Analysis*. Reading, MA: Addison-Wesley, 1987.
- [9] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimal decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 260–269, 1967.
- [10] N. Seshadri and C.-E. W. Sundberg, "List Viterbi decoding algorithms with applications," *IEEE Trans. Commun.*, vol. 42, pp. 313–323, 1994.
- [11] J. Hagenauer, "Rate compatible punctured convolutional codes (RCPCC-codes) and their applications," *IEEE Trans. Commun.*, vol. 36, pp. 389–400, Apr. 1988.
- [12] D. Yellin, A. Vardy, and O. Amrani, "Joint equalization and coding for intersymbol interference channels," *IEEE Trans. Inform. Theory*, vol. 43, pp. 409–425, Mar. 1997.



**Raghavan Anand** received the B.Tech. degree from the Indian Institute of Technology, Madras, in 1997 and the M.S. degree from the University of Illinois, Urbana-Champaign, in 1999, both in electrical engineering. He is currently pursuing the Ph.D. degree at the Department of Electrical Engineering and Computer Science, University of California at Berkeley.

His research interests include video coding/transmission, multimedia networking, and transport over heterogeneous networks.



**Kannan Ramchandran** received the M.S. and Ph.D. degrees in electrical engineering from Columbia University, New York, NY, in 1984 and 1993, respectively.

From 1984 to 1990, he was a Member of the Technical Staff at AT&T Bell Labs in the telecommunications R&D area. From 1990 to 1993, he was a Graduate Research Assistant at the Center for Telecommunications Research at Columbia University. From 1993 to 1999, he was a Research Assistant Professor in the Electrical and Computer Engineering Department

at the University of Illinois at Urbana-Champaign, and a Research Assistant Professor at the Beckman Institute and the Coordinated Science Laboratory. Since Fall 1999, he has been an Associate Professor in the Electrical Engineering and Computer Science Department at the University of California at Berkeley. His research interests include image and video compression and communication, wavelet theory and multirate signal processing, multimedia networking, distributed signal processing, and unified algorithms for signal processing, communications, and networking.

Dr. Ramchandran was the recipient of the 1993 Elaihu I. Jury Award at Columbia University for the best doctoral thesis in the area of systems, signal processing, or communications. He received the NSF Research Initiation Award in 1994, the Army Research Office Young Investigator Award in 1996, the NSF CAREER Award in 1997, and the Office of Naval Research Young Investigator Award in 1997, and the Okawa Foundation Award in 2000. In 1998, he was selected as a Henry Magnusky Scholar by the ECE Department at the University of Illinois chosen to recognize excellence among junior faculty. He is the co-recipient of two Best Paper awards from the IEEE Signal Processing Society: the 1996 Senior Paper Award and the 1998 Signal Processing Magazine Paper Award. He is a member of the technical committees of the IEEE Image and Multidimensional Signal Processing Committee and the IEEE Multimedia Signal Processing Committee, and serves as an Associate Editor for the IEEE TRANSACTIONS ON IMAGE PROCESSING.

**Igor V. Kozintsev** received the Diploma with honors in electrical engineering from the Moscow State Technical University n.a. Bauman Moscow, Russia, in 1994 and the M.S. and Ph.D. degrees from the University of Illinois at Urbana-Champaign (UIUC), in 1997 and 2000, respectively, all in electrical engineering.

Since 1996, he has been a Research Assistant at the Image Formation and Processing Laboratory at the Beckman Institute for Advanced Science and Technology at UIUC. In May 2000, he joined the Microprocessor Research Labs at Intel Corporation, Santa Clara, CA, where he currently holds the position of Senior Software Engineer. His research interests include multimedia processing, wireless communications, and networking.