Error Correcting Codes for Wireless Communications

Final Report

EE6390: Introduction to Wireless Communications

University of Texas at Dallas

6 December, 1999

James Agniel

Abstract—This paper investigates the theory and implementation of error correction coding for wireless communication systems. The purpose of this research is to familiarize the uninitiated with several methods of coding data for integrity through transmission in noisy channels typical of radio communication. A previous report [1] introduced basic ideas and summarized key papers [2][3][4] in the early development of coding theory. The present research builds on this information by detailing construction and decoding of common codes and comparing performance results. The two main types of error correcting codes—block codes and convolutional codes—are reviewed. A recent coding advancement utilizing novel methods from both block and convolutional codes called turbo coding is also reviewed. Finally, some case studies are presented as examples of real-world ECC implementations. Because of the complex nature of the subject, this research is focused on gaining a complete understanding of the existing literature and techniques rather than on generating and testing new theories.

I. Introduction

Channel coding for error detection and correction helps the communication system designer mitigate the effects of a noisy transmission channel. Error control coding has been widely used in all types of wireline and wireless communication systems for many years, and has been the subject of intense study since the 1940s. In 1948 Shannon [5] published his probabilistic, fundamental theorem for the binary symmetric channel (BSC), $C = \log_2(1+\frac{S}{N})$, where C is the channel capacity in bits/sec/Hz, and $\frac{S}{N}$ is the channel signal-to-noise ratio [6]. Shannon proposed that every BSC with capacity C can be encoded with arbitrary reliability and an information rate $R \le C$, but arbitrarily close to C. In other words, there exist codes such that $P_e \to 0$ and $R \to C$, as the code length $n \to ¥$. These codes may be implemented probabilistically or algebraically, theoretically to the same end [7].

The probabilistic definition of communications defines the need for error coding. Let $\mathbf{v}$ represent a transmitted vector of information symbols, and let $\mathbf{r}$ be the received vector. For a discrete, memoryless channel (DMC), we wish to maximize $P(\mathbf{v}\,|\,\mathbf{r}) = \dfrac{P(\mathbf{r}\,|\,\mathbf{v})P(\mathbf{v})}{P(\mathbf{r})}$ which implies maximizing $P(\mathbf{r}\,|\,\mathbf{v}) = \prod_i P(r_i\,|\,v_i)$ since, for a DMC, each received symbol depends only on its transmitted symbol. Maximum likelihood (ML) receivers for a DMC choose the codeword $\hat{\mathbf{v}}$ that maximizes the log-likelihood summation $\log P(\mathbf{r}\,|\,\mathbf{v}) = \sum_i \log P(r_i\,|\,v_i)$ [8].

For a binary symmetric, additive white Gaussian-noisy (AWGN) channel (BSC), the error probabilities are characterized as $P(r_i\,|\,v_j) = \begin{cases} p, & i \neq j \\ q = 1-p, & i = j \end{cases}$. If the distance between the length-n codeword $\mathbf{v}$ and the received vector $\mathbf{r}$ is $d(\mathbf{r},\mathbf{v})$, then ML receivers for a BSC choose $\hat{\mathbf{v}}$ for $\mathbf{v}$ to minimize the distance $d(\mathbf{r},\mathbf{v})$ between the chosen vector and the received vector. In terms of the log-likelihood, $\log P(\mathbf{r}\,|\,\mathbf{v}) = d(\mathbf{r},\mathbf{v})\log\frac{p}{q} + n\log q$ [8].

II. Linear Block Codes

In any error control code, data from a source or source encoder enters the channel coder, where redundant data is added to the message to add protection against random or burst errors. For k bits of binary information, there are $M=2^k$ valid codewords. If the encoder adds $n-k$ check bits to the message to form a codeword, the receiver may receive any of 2^n codewords, where 2^n-2^k could be in error. The code rate R is defined $R \equiv \frac{k}{n}$.

If we denote the ith bit of a codeword K by x_i, then all 2^k valid codewords represented by K satisfy $n-k$ linear equations in x_i for $i=1,2,\ldots,n$. That is, a code is considered linear if it can be defined by a homogeneous system of linear equations. Since there are $n-k$ equations in n unknowns, there are $n-(n-k)=k$ independent variables and, hence, 2^k solutions to the system [7].

The weight of a codeword is the number of its positions that are distinct from zero. The Hamming distance between two codewords is the number of positions in which the two codewords differ. The minimum distance d_{min} of a code is the smallest Hamming distance over all pairs of distinct codewords. A code's minimum distance is an important measure of its ability to detect or correct errors: A code will detect t errors if and only if $d_{min}>t$, and it will correct t errors if and only if $d_{min}>2t$.

The problem of finding codes with a given error correcting capability and with minimal length is a major one. For small codes, though, there are some well-known results for the upper bound $B(n,d_{min})$ on the number of message bits able to be protected by a code [7]. For example, for a code with $d_{min}=3$ (for an error correcting capability of one bit) and a message length $k=4$, $M = 2^k = 16 \le B(n,3) = \frac{2^n}{n+1} \Rightarrow n = 7$. This code consists of four message bits followed by three parity-check bits, for a codeword length $n=7$.

Codes are either systematic or nonsystematic; systematic codes have the check bits appended to the message bits, while the codewords for nonsystematic codes are formed by linear combinations resulting in unique codewords. Systematic codes are preferable for typical systems because of the ease of implementation as logic circuits.

Define **b** as the *n-k*-column vector of check bits, **m** as the *k*-column vector of message bits, **c** as the *n*-column codeword vector, **P** as the (*n-k*) x *k* coefficient matrix, **G** as the *n* x *k* generator matrix, and **H** as the (*n-k*) x *n* check matrix. Then the parity check bits are generated by $\mathbf{b} = \mathbf{Pm}$, and $\mathbf{G} = \left[\frac{\mathbf{I}_k}{\mathbf{P}}\right]$ where $\mathbf{I}_k$ is the *k* x *k* identity matrix. The block codeword is generated from the generator matrix and the message vector as follows: $\mathbf{c} = \mathbf{Gm} = \left[\frac{\mathbf{I}_k}{\mathbf{P}}\right]\mathbf{m} = \left[\frac{\mathbf{m}}{\mathbf{Pm}}\right] = \left[\frac{\mathbf{m}}{\mathbf{b}}\right]$. The parity check matrix is closely related to the generator matrix **G**, simply the coefficient matrix appended by the identity matrix $\mathbf{H} = [\mathbf{P} \mid \mathbf{I}_{n-k}]$, but it bears the important property that

$\mathbf{Hc} = [\mathbf{P} \mid \mathbf{I}_{n-k}]\left[\frac{\mathbf{m}}{\mathbf{b}}\right] = \mathbf{Pm} + \mathbf{b} = \mathbf{b} + \mathbf{b} = \mathbf{0}$ (for modulo-2 addition) [7].

As with most codes, decoding of block codes for error correction is a more involved process than the encoding. The process most often used is *syndrome decoding*. If the received vector **r** is corrupted by noise, we can represent it as $\mathbf{r} = \mathbf{c} + \mathbf{e}$, where **e** is the error pattern corrupting the transmitted codeword. The product of the received vector and the parity check matrix defines the syndrome s: $\mathbf{Hr} = \mathbf{Hc} + \mathbf{He} = \mathbf{0} + \mathbf{He} = \mathbf{s}$. Therefore the syndrome depends only on the received error pattern. In fact, **s** comprises the sum of the columns of **H** corresponding to the bit locations of the errors, so there are multiple error patterns that can have the same syndrome. If error detection is the decoder's goal, a nonzero syndrome indicates the presence of an error. The set of these possible error patterns is known as the *coset*; for ML decoding, we choose from the coset the error pattern with the smallest weight as the *coset leader*, or the most likely error pattern, and add it to **r** to correct the erred codeword. If there is more than one candidate coset leader, there are more erred bits than the code can correct. Typically, for short codes (*n-k*<10), the coset leader for each of the possible 2^{n-k} syndromes is stored in a lookup table. More efficient, elegant schemes for hard-decision decoding have been devised for more complex codes [8].

III. Cyclic Codes

Cyclic codes are a subset of the linear block codes that satisfy the additional

condition that any cyclic shift of the elements of a codeword yields a word that is also a codeword. This property gives cyclic codes a structure that can be exploited in designing more powerful codes. Where the previous codes were constructed by summing bits to create a parity checksum, cyclic codes use polynomial division to increase a code's complexity.

Let the code polynomial be defined as $C(x) = \sum_{i=0}^{n-1} c_i x^i$, the message polynomial as $M(x) = \sum_{i=0}^{k-1} m_i x^i$, and the parity check polynomial as $B(x) = \sum_{i=0}^{n-k-1} b_i x^i$. Then, for a systematic code, we have $c_0 c_1 \mathrm{K}\ c_{n-k-2} c_{n-k-1} c_{n-k} c_{n-k+1} \mathrm{K}\ c_{n-2} c_{n-1} \Rightarrow b_0 b_1 \mathrm{K}\ b_{n-k-2} b_{n-k-1} m_0 m_1 \mathrm{K}\ m_{k-2} m_{k-1}$ where c_0 represents the MSB. Thus, $C(x) = B(x) + x^{n-k} M(x)$. It can be shown [9] that $x^j C(x) = Q(x)(x^n + 1) + C^{(j)}(x)$, where $Q(x)$ is a quotient, and $C^{(j)}(x)$ is simply the $C(x)$ coefficients right-shifted by j places. So the shifted version of a codeword is the remainder after dividing $x^j C(x)$ by x^n+1: $C^{(j)}(x) = x^j C(x) \bmod (x^n + 1)$.

The generator matrix **G** has the polynomial equivalent $G(x) = 1 + \sum_{i=1}^{n-k-1} g_i x^i + x^{n-k}$. We choose $G(x)$ to be a factor of $x^n + 1$, and form systematic codewords according to $C(x) = A(x)G(x) = B(x) + x^{n-k} M(x)$. Thus the parity check polynomial $B(x)$ is the remainder left by dividing $x^{n-k}M(x)$ by the generator polynomial $G(x)$, $\frac{x^{n-k} M(x)}{G(x)} = A(x) + \frac{B(x)}{G(x)}$. The codeword is then formed by the coefficients of $C(x)$ [9].

Syndrome decoding of cyclic codes is analogous to the previous description. The received vector **r** is represented in its polynomial form $R(x) = C(x) + E(x) = A(x)G(x) + E(x)$. The syndrome is obtained by the remainder left by dividing the received polynomial by the generator function: $\frac{R(x)}{G(x)} = Q(x)G(x) + S(x) = A(x)G(x) + E(x) \Rightarrow E(x) = [Q(x) + A(x)]G(x) + S(x)$. If $G(x)$ divides $R(x)$ exactly, then $S(x)$=0, no error has occurred and $R(x)$ represents the correct codeword [9].

Figure 1 shows a linear shift register implementation of an (*n-k*)-stage systematic cyclic block coder and decoder. Encoding is accomplished by clearing the registers, setting Enable, clearing Decode in, and shifting the message through Encode in. Once all message bits have been clocked through, Enable is cleared and the check bits ($B(x)$) are clocked out after the message. The generator polynomial is pre-loaded into the coder. The decode operation is similar; after clocking through the received bits, the register state holds the syndrome.

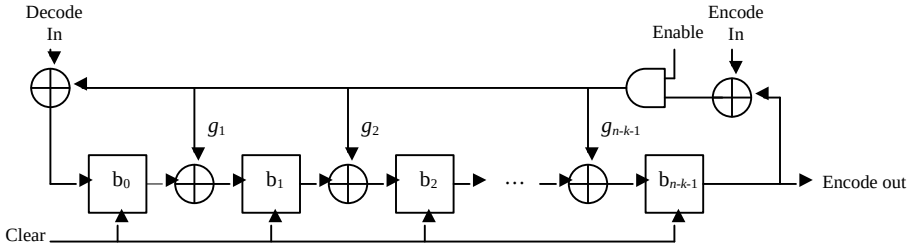

Figure 1. Linear shift register codec for cyclic codes [7]

The most important class of linear block codes is the group of BCH codes [10][11]. BCH codes conform to the following: for positive integers m ($m \geq 3$) and t ($t < 2^m - 1$), codes will have $n = 2^m - 1$, $n - k \leq mt$, and $d_{\min} \geq 2t + 1$. The generator polynomial is specified in terms of its roots in the Galois field GF(2^m) [8]. For binary BCH codes, m=1. A most important class of nonbinary BCH codes is the Reed-Solomon (RS) codes [4]. RS codes are typically designed with the intent of protecting against burst errors. This protection is accomplished by mapping RS code symbols to m-bit bytes on GF(2^m). The block length becomes $n = m(2^m - 1)$, and the check symbols $n - k = 2mt$. Whereas binary BCH codes are *t-bit correcting codes*, RS codes are *t-byte-correcting codes*, since symbols are typically designed to accommodate one byte of information. In this manner, RS codes will correct burst errors in any combination of $l = \dfrac{t}{1 + \lfloor (l + m - 2)/m \rfloor}$ or fewer bursts of length l while still correcting t or fewer random errors [8].

IV. Convolutional Codes

Convolutional codes differ from block codes in several ways, most notably in that the encoder contains memory, causing the encoder outputs to depend on previous outputs. Codes are typically described as (*n,k,m*) codes, where *n* is the number of outputs to be multiplexed, *k* is the number of inputs to demultiplex, and *m* is the length of the

encoder's memory. The code rate R is defined identically to that of the block codes, $R \equiv \frac{k}{n}$. A code's constraint length is defined $n_A \equiv n(m+1)$. The constraint length is the maximum number of encoder outputs that can be affected by a single input bit [8]. Figure 2 shows an example of a (2,1,3) encoder.

Since the convolutional encoder can be considered a sequential circuit, its operation is usually described by a state diagram, and analyzed with the help of a trellis diagram, as in Figure 2.

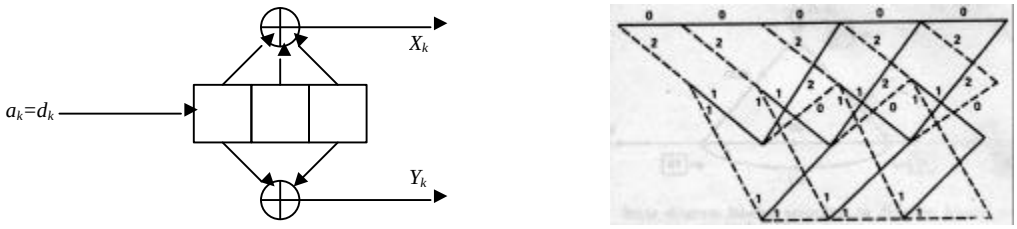

Figure 2. A (2,1,3) rate ½ nonsystematic convolutional coder and its trellis diagram showing branch weights [12]

Convolutional codes are defined by generator sequences $\mathbf{g}^{(i)}$, i=1,…,*n*, which are equivalent to the responses of the system through each of its signal paths to the impulse sequence **u**=(1 0 0 …). For *n*=2 and input sequence **u**, the encoding equations can be written $\begin{matrix}\mathbf{v}^{(1)} = \mathbf{u} * \mathbf{g}^{(1)} \\ \mathbf{v}^{(2)} = \mathbf{u} * \mathbf{g}^{(2)}\end{matrix}$, where * denotes the discrete convolution [8]. The encoding generators can be found by inspection of the sequential circuit: For the coder in Figure 2, $\mathbf{g}^{(1)}$=[0 1 1 1] and $\mathbf{g}^{(1)}$=[0 1 0 1].

A code's transfer function *T*(*D*) is calculated from the state diagram by solving its state equations. *T*(*D*) yields a function of its distance properties. The *minimum free distance* of the code is measured as the minimum Hamming distance of the sequence of output bits corresponding to each branch from the sequence of output bits defining the all-zeros branch [9].

The code in Figure 2 is a nonsystematic convolutional (NSC) code. It has been shown [13] that the bit error rate (BER) for NSC codes is typically better than that of the NSC dual, recursive systematic convolutional (RSC) codes, at large SNR. An RSC code can be obtained from its counterpart by applying the same generators to the systematic architecture, as in Figure 3.

Viterbi [12] devised the most popular method for decoding convolutional codes of

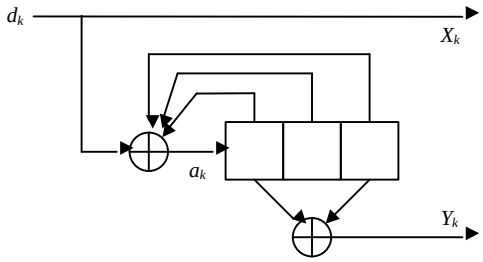

Figure 3. The recursive systematic form of the code in Figure 2 [13]

short constraint lengths. The Viterbi algorithm is an optimum, maximum likelihood decoding method. Due to its complexity, it is left out of this brief review. Proakis [9] and Lin and Costello [8] offer thorough treatments of Viterbi decoding.

Convolutional as well as block decoding can take on one of two meanings: Hard-decision decoding is suited for the BSC, while soft-decision or unquantized decoding is optimum for the AWGN, discrete memoryless channel. Coding gains of approximately 2 dB over hard decision decoding have been demonstrated using soft-decision decoding [9].

V. Turbo Codes

The class of turbo codes introduced in 1993 [13] promises to reduce BER levels to previously unattainable levels for a given code complexity. Turbo codes use parallel-concatenated RSC coders to encode a message. Interleavers that guarantee each encoder a distinct version of the same message separate the coders. The interleavers perform the function $\tilde{l} = a(l)$, where l is the input bit position and $\tilde{l}$ is the output bit position. $a(\cdot)$ is chosen such that $l \bmod p = a(l) \bmod p, \quad \forall l$, where $p \le 2^m$-1 is the period of the impulse response of each constituent RSC encoder [14]. The parity bits Y_k (as in Figure 3) are multiplexed on to the channel according to some puncturing rule. If the same number of bits is not taken from each encoder, the system can be seen as the parallel concatenation of convolutional codes of different rates. Berrou alludes [13] that performance gains can be found by decoding the lower rate codes first, similar to using a lower rate inner code in serial concatenation schemes.

A Typical turbo decoder is shown in Figure 4. The decoding operation gives turbo codes their name: soft-decision output is fed from DEC1 to the input of DEC2. DEC2 can then either output hard-decision data or feedback more extrinsic data to the input of DEC1. In this manner the decoder works like a turbocharger of an engine.

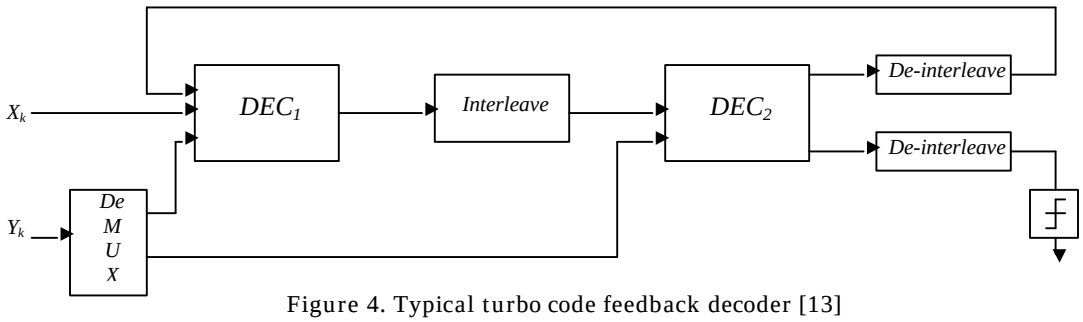

Figure 4. Typical turbo code feedback decoder [13]

Unfortunately, for most codes there is no closed-form method for predicting error correcting performance. The best we can do is to apply union or loose upper bounds to the probability of word error [15]. For example, the probability of word error for a block code that encodes k bit information sequences is bounded by $P_c \le \sum_{m=2}^{2^k} P_2(m)$, where $P_2(m)$ is the pairwise error probability between word 1 (the all-zero codeword) and word m [14]. For BPSK signaling and soft-decision decoding, $P_2(m) = Q\left(\sqrt{\frac{2e_b}{N_0} R w_m}\right)$ where w_m is the Hamming weight of word m. This metric, however, is almost irrelevant with respect to codes designed to minimize the probability of bit errors rather than word errors [13]. Codes of interest must be designed and implemented in software in order to test or compare their performance. However, Berrou [13] reports that for turbo codes with a block size of 2^{16}, and for at least 18 iterations of the decoding procedure, a BER of 10^{-5} at e_b/N_o of 0.7 dB are attainable.

VI. Conclusion

In this paper, a detailed overview of the major concepts in error correction coding was presented. Several examples were given to illustrate important concepts. It would not have been possible for a newcomer to the field to develop useful simulations or even to perform a thorough survey of techniques currently in use without this first, important research step. With more time, or as a future project, such a survey would be completed to determine the state of the art for certain broadband wireless applications. A simulation would then be devised to determine realizable coding schemes that outperform current methods for use in emerging fixed broadband technologies.

References

[1] J. Agniel, “A Review of the Origins of Error Correcting Codes,” prepared for EE6390: Introduction to Wireless Communications, Oct., 1999.

[2] R. W. Hamming, “Error detecting and error correcting codes,” *Bell Syst. Tech. J.*, vol. 29, pp. 147-160, Apr., 1950.

[3] M. J. E. Golay, “Notes on digital coding,” *Proc. IRE*, vol 37, p. 657, June 1949.

[4] S. Reed and G. Solomon, “Polynomial codes over certain finite fields,” *J. Soc. Ind. Appl. Math.*, vol. 8, pp. 739-741, 1960.

[5] C. E. Shannon, “A Mathematical Theory of Communication,” *Bell Syst. Tech. J.*, vol. 27, pp. 379-423, July, 1948.

[6] T. S. Rappaport, *Wireless Communications: Principles and Practice*, New Jersey: Prentice-Hall, 1996.

[7] R. Togneri, *Lecture notes from ITC314*, The University of Western Australia, May, 1999. http://www.ee.uwa.edu.au/~roberto/units/itc314/handouts/lectures/,

[8] S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications.* New Jersey: Prentice-Hall, 1983.

[9] J. G. Proakis, *Digital Communications*. New York: McGraw-Hill, 1995.

[10] R. C. Bose and D. K. Ray-Chaudhuri, “On a Class of Error Correcting Binary Group Codes,” *Inf. Control*, vol. 3, pp. 68-79, March, 1960.

[11] W. W. Peterson, “Encoding and Error-Correction Procedures for the Bose-Chaudhuri Codes,” *IRE Trans. Inf. Theory*, vol. IT-6, pp. 459-470, Sept. 1960.

[12] A. J. Viterbi, “Convolutional Codes and Their Performance in Communication Systems,” *IEEE Trans. Comm. Tech.*, vol. COM-19, pp. 751-772, Oct. 1971.

[13] C. Berrou, A. Glavieux, and P. Thitimasjshima, “Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes,” in *Proc., IEEE Int. Conf. On Communications*, (Geneva, Switzerland, May 1993), pp. 1064-1070.

[14] M. C. Valenti, “An Introduction to Turbo Codes”, unpublished, Virginia Polytechnic Inst. & S.U., May 1996, http://www.cs.wvu.edu/~mvalenti/pubs.html.

[15] T. M. Duman and M. Salehi, “New Performance Bounds for Turbo Codes,” in *Proc., GLOBECOM ’97*, pp. 634-638, 1997.