Contemporary Engineering Sciences, Vol. 4, 2011, no. 1, 1 – 12

Error Free Transport of Transmitter Payload over Ad Hoc Wireless Network Using Osculating Polynomial Cross Products

Siddesh G.K.

Department of Electronics and Communication
SBM Jain College of Engineering, Bangalore, India
professorsiddeshgk@gmail.com

K. N. Muralidahara

Department of Electronics and Communication
PES College of Engineering, Mandya, India

Keywords: Pay Load, Osculating, Cyclic Redundancy Codes, Ad Hoc Networks, Polynomials, RS Codes

Abstract

In any communication scenario, is it a wired network or an ad hoc wireless network or digital cordless telephony, transport of payload in an error-free manner is of paramount importance. It is matter of common knowledge that transport of data (whether modulated or un-modulated) over a medium is always subject to external noise, introducing bit errors. Received data containing errors is practically of no use in most of the situations. Traditionally, the most common solution is to employ Cyclic Redundancy Checksums which are generated using primitive modulo-2 polynomials and appended to the payload. The receiver computes the CRC of the

payload locally and compares with the received CRC. If a match doesn't occur, it is assumed that the received data is in error and usually the implemented protocol requests for a retransmission. This implies that whenever there is error in the data stream, the protocol transfer time is doubled or trebled due to retransmissions. Such a situation is not acceptable in scenarios where the transfer needs to take place in real time without much latency. This is typical of Ad Hoc Wireless Networks deployed for disaster management where time plays a crucial role.

It would be ideal if there is a method whereby errors can somehow be detected at the receiver and corrected on the fly without the need of retransmission. Technologies like Digital TV operating in real time have used Reed Solomon Error Correcting Codes to correct bit errors at the receiver end. Correction of bit errors involves just flipping the bits to the opposite state. In this paper we use Osculating Polynomials which have the desirable property that their cross product changes with change in bit patterns and given the original cross product, the original bit patterns of the data can be easily recovered by the inverse process of de-convolution. This paper, therefore, seeks to discuss the properties of osculating polynomials and their application in forward error correction on digital streams within the framework of Ad Hoc Networks.

Introduction

Environmental interference and physical defects in the communication medium can cause random bit errors during data transmission. Error coding is a method of detecting and correcting these errors to ensure information is transferred intact from its source to its destination. Error coding is used for fault tolerant computing in computer memory, magnetic and optical data storage media, satellite and deep space communications, network communications, cellular telephone networks, and almost any other form of digital data communication. Error coding uses mathematical formulas to encode data bits at the source into longer bit words for transmission. The "code word" can then be decoded at the destination to retrieve the information. The extra bits in the code word provide redundancy that, according to the coding scheme used, will allow the destination to use the decoding process to determine if the communication medium introduced errors and in some cases correct them so that the data need not be retransmitted. Different error coding schemes are chosen depending on the types of errors expected, the communication medium's expected error rate, and whether or not data retransmission is possible. Faster processors and better communications technology make more complex coding schemes, with better error detecting and correcting capabilities, possible for smaller embedded systems, allowing for more robust communications. However, tradeoffs between bandwidth and coding overhead, coding complexity and allowable coding delay between transmissions must be considered for each application.

Error coding is a method of providing reliable digital data transmission and storage when the communication medium used has an unacceptable bit error rate (BER) and a low signal-to-noise ratio (SNR). Error coding is used in many digital applications like computer memory, magnetic and optical data storage media, satellite and deep space communications, network communications, and cellular telephone networks. Rather than transmitting digital data in a raw bit for bit form, the data is encoded with extra bits at the source. The longer "code word" is then transmitted, and the receiver can decode it to retrieve the desired information. The extra bits transform the data into a valid code word in the coding scheme. The space of valid code words is smaller than the space of possible bit strings of that length and therefore the destination can recognize invalid code words.

If errors are introduced during transmission, they will likely be detected during the decoding process at the destination because the code word would be transformed into an invalid bit string. Given a data string to be transmitted that is k bits long, there are 2^k possible bit strings that the data can be. Error coding assumes the worst case scenario that the information to be encoded can be any of these bit strings. Therefore there will be 2^k valid code words. The code words will be n bits long, where $n > k$. So just having extra bits in the data transmission eliminates many of the possible 2^n bit strings as valid code words.

Perhaps the simplest example of error coding is adding a parity check bit. A bit string to be transmitted has a single bit concatenated to it to make a code word for transmission. The bit is a 1 or a 0 depending on the parity. If odd parity is being used, the parity bit will be added such that the sum of 1's in the code word is odd. If even parity is being used, the sum of 1's in the code word must be even.

A number of error correcting mechanisms are in vogue in the domain of communication each of which has its own merits and demerits. Most notable of these are Reed-Solomon Codes, BCH, Turbo Codes, Cyclic Redundancy Checksums etc. to name a few. Although these methods are capable of correction of bit errors, they are not by any way foolproof. Multiple bit errors are quite difficult to handle. However, a brief mention of Reed-Solomon Code will be appropriate in the context of exposition on error correction.

Reed-Solomon Codes

In coding theory, **Reed–Solomon (RS) codes** are non-binary cyclic error correcting codes invented by Irving S. Reed and Gustave Solomon. They described a systematic way of building codes that could detect and correct multiple random symbol errors.

By adding t check symbols to the data, an RS code can detect any combination of up to t erroneous symbols, and correct up to ⌊t/2⌋ symbols. As an erasure code, it can correct up to t known erasures, or it can detect and correct combinations of errors and erasures. Furthermore, RS codes are suitable as multiple-burst bit-error correcting codes, since a sequence of b+1 consecutive bit errors can affect at most two symbols of size b. The choice of t is up to the designer of the code, and may be selected within wide limits.

In Reed-Solomon coding, source symbols are viewed as coefficients of a polynomial p(x) over a finite field. The original idea was to create n code symbols from k source symbols by oversampling p(x) at n > k distinct points, transmit the sampled points, and use interpolation techniques at the receiver to recover the original message. That is not how RS codes are used today. Instead, RS codes are viewed as cyclic BCH codes, where encoding symbols are derived from the coefficients of a polynomial constructed by multiplying p(x) with a cyclic generator polynomial. This gives rise to an efficient decoding algorithm, which was discovered by Elwyn Berlekamp and James Massey, and is known as the Berlekamp-Massey decoding algorithm.

Reed-Solomon codes have since found important applications from deep-space communication to consumer electronics. They are prominently used in consumer electronics such as CDs, DVDs, Blu-ray Discs, in data transmission technologies such as DSL & WiMAX, in broadcast systems such as DVB and ATSC, and in computer applications such as RAID 6 systems. In 1977, RS codes were notably implemented in the Voyager program in the form of concatenated codes. The first commercial application in mass-produced consumer products appeared in 1982 with the compact disc, where two interleaved RS codes are used. Today, RS codes are widely implemented in digital storage devices and digital communication standards, though they are being slowly replaced by more modern low-density parity-check (LDPC) codes or turbo codes. For example, RS codes are used in the digital video broadcasting (DVB) standard DVB-S, but LDPC codes are used in its successor DVB-S2.

The original concept of Reed-Solomon coding (Reed & Solomon 1960) describes encoding of k message symbols by viewing them as coefficients of a polynomial p(x) of maximum degree k-1 over a finite field of order N, and evaluating the polynomial at n>k distinct input points. Sampling a polynomial of degree k-1 at more than k points creates an over-determined system, and allows recovery of the polynomial at the receiver given any k out of n sample points using (Lagrange) interpolation. The sequence of distinct points is created by a generator of the finite field's multiplicative group, and includes 0, thus permitting any value of n up to N. Using a mathematical formulation, let $(x_1, x_2, ..., x_n)$ be the input sequence of n distinct values over the finite field F; then the codebook **C** created from the tuplets of values obtained by evaluating every polynomial (over F) of degree less than k at each x_i is

$$C = \{(f(x_1), (f(x_2), (f(x_3) \dots (f(x_n)) \mid f \in F[x], \deg(f) < k$$

where F[x] isthe polynomial ring over F, and k and n are chosen such that $1 \leq k \leq n \leq N$. As described above, an input sequence $(x_1, x_2, ..., x_n)$ of n=N values is created as

$$(0, \alpha^0, \alpha^1, \dots, \alpha^{N-2}),$$

where α is a primitive root of F. When omitting 0 from the sequence, and since $\alpha^{N-1} = 1$, it follows that for every polynomial p(x) the function p(αx) is also a polynomial of the same degree, and its codeword is a cyclic left-shift of the codeword derived from p(x); thus, a Reed–Solomon code can be viewed as a cyclic code. This is pursued in the classic view of RS codes, described subsequently.

Error Correction based on Osculating Polynomials

Consider a case in which we desire an interpolating polynomial that not only agrees with the function values at a discrete number of points, but whose derivative(s) also agree(s) with the derivative(s) of the function at these same points. These needs can be fulfilled by so-called osculating polynomials. Theory of Osculating Polynomials is to be found extensively in various literatures in mathematics. Also known as Hermite Polynomials, they have the most desirable property of detecting sequences of bit errors.

In this paper, we seek to exploit this property in developing a systematic procedure for error detection and correction on the fly by computing the so-called Osculating Polynomial Cross Product. Henceforth, throughout this discussion, we will refer to our error correction scheme as **OPCP** for short.

Introduction

OPCP codes are some of the sturdiest error correcting codes. Error correcting codes are very useful in sending information over long distances or through channels where errors might occur in the message. They have become more prevalent as telecommunications have expanded and developed a use for codes that can self-correct.

Here below, we discuss vector operations and describe how to form a vector from a polynomial. Subsequently, we introduce OPCP codes and show how to form encoding matrices and encode messages. Thereafter, we give a method for decoding OPCP encoded messages. Finally, we give an example of decoding an encoded message in the concluding pages.

Definition of Terms and Operations

The vector spaces used in this paper consist of strings of length 2^m, where m is a positive integer, of numbers in $F2 = \{0,1\}$. The code words of a OPCP code form a subspace of such a space. Vectors can be manipulated by three main operations: addition, multiplication, and the dot product.

For two vectors $X = (x_1, x_2, x_3, \dots, x_n)$ and $Y = (y_1, y_2, y_3, \dots, y_n)$, addition is defined by:

$X + Y = (x_1 + y_1, x_2 + y_2, x_3 + y_3, \dots, x_n + y_n);$

where each x_i or y_i is either 1 or 0, and 1 + 1 = 0, 0 + 1 = 1, 1 + 0 = 1, 0 + 0 = 0:

For example, if x and y are defined as x = (10011110) and y = (11100001), then the sum of X and Y is $X + Y = (10011110) + (11100001) = (01111111);$

The addition of a scalar $a \in F2$ to vector X is defined by

$a + X = (a + x_1, a + x_2, \dots, a + x_n).$

The complement $\bar{X}$ of a vector X is the vector equal to $1 + X$. An example of the addition of a constant to a vector is 1 + (000111) = (111000). Multiplication is defined by the formula,

$X * Y = (x_1 * y_1, x_2 * y_2, x_3 * y_3, \dots, x_n * y_n)$

where each x_i and y_i is either 1 or 0 and 1 * 1 = 1, 0 * 1 = 0, 1 * 0 = 0, 0 * 0 = 0.

For example, using the same X and Y above, the product of $X \text{ and } Y$ is:

$X * Y = (10011110) * (11100001) = (10000000).$

The multiplication of a constant $a \in F2$ to vector Y is defined by:

$a * X = (a * x_1, a * x_2, a * x_3, \dots a * x_n)$

An example is 0 * (111001) = (000000). The dot product of X and Y is defined by:

$X.Y = (x_1 . y_1, x_2 . y_2, x_3 . y_3, \dots, x_n . y_n)$

For example, using X and Y from above:

$X.Y = (10011110).(11100001) = 1 + 0 + 0 + 0 + 0 + 0 + 0 + 0 = 1.$

All three of these operations require vectors with the same number of coordinates. Vectors can be associated with Boolean polynomials. A Boolean polynomial is a linear combination of Boolean monomials with coefficients in F2. A Boolean monomial p in the variables:

$x_1, x_2, x_3, \dots, x_n$

is an expression of the form,

$p = x_1^{r_1} x_2^{r_2} x_3^{r_3} \dots x_n^{r_n}$

where $r_i \in \{0, 1, 2, \dots\} \text{and } 1 \leq i \leq m.$

The reduced form $p \ and \ p'$ is obtained by applying the rules:
$x_i x_j = x_j x_i \ and \ x_i^2 = x_i$
until the factors are distinct. The degree of p is the ordinary degree of p', which is the number of variables in p'. A Boolean polynomial is in reduced form if each monomial is in reduced form. The degree of a Boolean polynomial q is the ordinary degree of its reduced form q'.
An example of a Boolean polynomial in reduced form with degree three is:
$q = x_1 + x_2 + x_1x_2 + x_2x_3x_4$

We can now describe how to associate a Boolean monomial in **m** variables to a vector with 2^m entries. The degree-zero monomial is 1, and the degree-one monomials are $(x_1, x_2, \dots, x_m)$.
First we define the vectors associated with these monomials. The vector associated with the monomial 1 is simply a vector of length 2^m, where every entry of the vector is 1. So, in a space of size 2^3, the vector associated with 1 is (11111111). The vector associated with the monomial x_1 is 2^{m-1} ones, followed by 2^{m-1} zeros. The vector associated with the monomial x_2 is 2^{m-2} ones, followed by 2^{m-2} zeros, then another 2^{m-2} ones, followed by another 2^{m-2} zeros. In general, the vector associated with a monomial x_i is a pattern of 2^{m-i} ones followed by 2^{m-i} zeros, repeated until 2^m values have been defined. For example, in a space of size 2^4, the vector associated with x_4 is (1010101010101010).
To form the vector for a monomial $x_1^{r_1} x_2^{r_2} x_3^{r_3} \dots$, first place the monomial in reduced form. Then multiply the vectors associated with each monomial x_i in the reduced form. For example, in a space with m = 3, the vector associated with the monomial $x_1x_2x_3$ can be found by multiplying (11110000) * (11001100) * (10101010) which gives (10000000).
To form the vector for a polynomial, simply reduce all of the monomials in the polynomial, and find the vectors associated with each of the monomials. Then, add all the vectors associated with each of these monomials together to form the vector associated with the polynomial. This gives us a bijection between reduced polynomials and vectors. From now on, we will treat the reduced polynomial and the vector associated with that polynomial interchangeably.

OPCP Code and Encoding Matrices

An r^{th} order OPCP code $R(r, m)$ is the set of all binary strings (vectors) of length equal to $n = 2^m$ associated with the Boolean polynomials $P(x_1, x_2, x_3, \dots, x_m)$ of degree at most r. The 0^{th} order OPCP code $R(0, m)$ consists of the binary strings associated with the constant polynomials 0 and 1, that is, $R(0, m) = \{0,1\} Rep(2^m)$.

Thus, $R(0,m)$ is just a repetition of either zeros or ones of length 2^m. At the other extreme, the m^{th} order OPCP code $R(m,m)$ consists of all binary strings of length 2^m. To define the encoding matrix of $R(r,m)$, let the first row of the encoding matrix be 1, the vector length 2^m with all entries equal to 1. If r is equal to 0, then this row is the only one in the encoding matrix. If r is equal to 1, then add m rows corresponding to the vectors $x_1, x_2, x_3, \dots, x_m$ to the $R(0,m)$ encoding matrix. To form a $R(r,m)$ encoding matrix where r is greater than 1, add $\binom{m}{r}$ rows to the $R(r-1,m)$ encoding matrix. These added rows consist of all the possible reduced degree r monomials that can be formed using the rows $x_1, x_2, x_3, \dots, x_m$. For example, when m = 3 we have:

$$\left[\begin{array}{ccccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ x_1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ x_2 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ x_3 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{array}\right] \quad \Re(1,3).$$

The rows x_1x_2 = 11000000, x_1x_3 = 10100000, and x_2x_3 = 10001000 are added to form:

$$\left[\begin{array}{ccccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ x_1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ x_2 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ x_3 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ x_1x_2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_2x_3 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right] \quad \Re(2,3).$$

Finally, the row $x_1x_2x_3$ = 10000000 is added to form:

$$\left[\begin{array}{ccccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ x_1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ x_2 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ x_3 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ x_1x_2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_1x_3 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ x_2x_3 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ x_1x_2x_3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \quad \Re(3,3).$$

Another example of a OPCP encoding matrix is:

$$\left[\begin{array}{l|cccccccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
x_1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
x_2 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
x_3 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
x_4 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
x_1x_2 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
x_1x_3 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
x_1x_4 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
x_2x_3 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
x_2x_4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
x_3x_4 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{array}\right] \quad \Re(2,4).$$

Encoding a message using OPCP code $R(r,m)$ is straightforward. Take the code we are using to be $R(r,m)$. Its dimension is:

$$k = 1 + \binom{m}{1} + \binom{m}{2} + \binom{m}{3} + \dots + \binom{m}{r}$$

In other words, the encoding matrix has k rows. We send messages in blocks of length k. Let $m = (m_1, m_2, m_3, \dots, m_k)$, the encoded message M_c is: $M_c = \sum_{i=1}^{k} m_i R_i$, where R_i is a row of the encoding matrix of $R(r,m)$.

For example, using $R(1,3)$ to encode m = (0110) gives:
0 * (11111111) + 1 * (11110000) + 1 * (11001100) + 0 * (10101010) = (00111100).
Similarly, using $R(2,4)$ to encode m = (10101110010) gives (0011100100000101).

Decoding OPCP:
Decoding OPCP encoded messages is more complex than encoding them. The theory behind encoding and decoding is based on the distance between vectors. The distance between any two vectors is the number of places in the two vectors that have different values. The distance between any two code words in $R(r,m)$ code is 2^{m-r}. The basis for OPCP encoding is the assumption that the closest codeword in $R(r,m)$ to the received message is the original encoded message. Thus for **e** errors to be corrected in the received message, the distance between any two of the code words in $R(r,m)$ must be greater than **2e**.
The decoding method used is not very efficient, but is straightforward to implement. It checks each row of the encoding matrix and uses majority logic to determine whether that row was used in forming the encoding message. Thus, it is possible to determine what the error-less encoded message was and what the original message was. This method of decoding is given by the following algorithm: Apply Steps 1

and 2 below, to each row of the matrix, starting from the bottom and working upwards.

Step 1.

Choose a row in the $R(r,m)$ encoding matrix. Find 2^{m-r} characteristic vectors (this process is described below) for that row, and then take the dot product of each of those rows with the encoded message.

Step 2.

Take the majority of the values of the dot products and assign that value to the coefficient of the row.

Step 3.

After doing Steps 1 and 2 for each row except the top row from the bottom of the matrix up, multiply each coefficient by its corresponding row and add the resulting vectors to form M_y. Add this result to the received encoded message. If the resulting vector has more ones than zeros, then the top row's coefficient is 1, otherwise it is 0. Adding the top row, multiplied by its coefficient to M_y gives the original encoded message. Thus, we can identify the errors. The vector formed by the sequence of coefficients starting from the top row of the encoding matrix and ending with the bottom row is the original message.

To find the characteristic vectors of any row of the matrix, take the monomial r associated with the row of the encoding matrix. Then, take **E** to be the set of all x_i that are not in the monomial r, but are in the encoding matrix. The characteristic vectors are the vectors corresponding to the monomials in x_i and $\overline{x_i}$, such that exactly one of x_i or $\overline{x_i}$ is in each monomial for all x_i in **E**. For example, the last row of the encoding matrix $R(2,4)$ is associated with x_3x_4 so the characteristic vectors correspond to the following combinations of $x_1 x_2 \overline{x_1}$, and

$\overline{x_2}$: x_1x_2, $x_1\overline{x_2}$, $\overline{x_1}\ x_2$, $\overline{x_1}\ \overline{x_2}$

These characteristic vectors have the property that the dot product is zero with all the rows in $R(r,m)$ except the row to which the characteristic vectors correspond.

Example:

If the original message is m = (0110) using $R(1,3)$, then the encoded message is $M_c = (00111100)$. Because the distance in $R(1,3)$ is 2^{3-1} = 4, this code can correct one error. Let the encoded message after the error be M_e = (10111100): The characteristic vectors of the last row x_3 = (10101010) are x_1x_2, $x_1\overline{x_2}$, $\overline{x_1}x_2$ and $\overline{x_1}\ \overline{x_2}$. The vector associated with x_1 is (11110000), so $\overline{x_1}$ = (00001111). The vector associated with x_2 is (11001100), so $\overline{x_2}$ = (00110011). Therefore, we have x_1x_2 = (11000000), $x_1\overline{x_2}$ = (00110000), $\overline{x_1}x_2$ = (00001100), and $\overline{x_1}\ \overline{x_2}$ = (00000011). Taking the dot products of these vectors with M_e, we get

(11000000) . (10111100) = 1; (00110000) . (10111100) = 0;

(00001100) . (10111100) = 0; (00000011) . (10111100) = 0:

We can conclude that the coefficient of x_3 is 0.
Doing the same for the second to last row of the matrix, x_2 = (11001100), we get the characteristic vectors x_1x_3, $x_1\overline{x_3}$, $\overline{x_1}x_3$ and $\overline{x_1}\ \overline{x_3}$. These vectors are (10100000), (01010000), (00001010), and (00000101), respectively. Taking the dot products of these vectors with M_e, we get:

(10100000) . (10111100) = 0; (01010000) . (10111100) = 1;
(00001010) . (10111100) = 1; (00000101) . (10111100) = 1:

So, we can conclude that the coefficient of x_2 is 1. Doing the same for the second row of the matrix x_1 = (11110000), we get:

(10001000) . (10111100) = 0; (00100010) . (10111100) = 1;
(01000100) . (10111100) = 1; (00010001) . (10111100) = 1:

We can conclude that the coefficient for x_1 is also 1.
If we add 0 * (10101010) and 1 * (11001100) and 1 * (11110000) we get M_y, which is equal to (00111100). Then we see that the sum of M_y and M_e is equal to (00111100) + (10111100) = (10000000).
This message has more zeros than ones, so the coefficient of the first row of the encoding matrix is zero. Thus we can put together coefficients for the four rows of the matrix, 0,1,1, and 0, and see that the original message was (0110). We can also see that the error was in the first place of the error-free message M_c = (00111100).

Conclusion:

It is no exaggeration that communication protocols in Wireless Ad Hoc Networks should be secure and error free, considering the fact that such networks are deployed in emergency situations requiring real time response. Most of the protocols implemented hitherto in Wireless Ad Hoc Networks are quite unsatisfactory in terms of error-free communication (DARPA Report: 26754 – November 2009), especially considering the fact that the nodes are likely to be highly mobile and that a noise-free environment is in no way guaranteed. A compromise has to be reached whereby a balance between clean reception and speed of communication is achieved without impairing the quality and real-time nature.

A majority of the protocols specifically Reed-Solomon, Hamming Codes and Turbo Codes are not robust enough in achieving the required goal. While isolated bit errors are corrected well, long-duration burst errors are almost impossible to correct. Our Osculating Polynomial based correction scheme achieves the desired goal to a large extent. Simulations on burst errors of varying duration give very satisfactory results. Very long burst errors are in any case impossible to correct – this is a state where no strategy, however sophisticated, will succeed. Practical implementations of

Osculating Polynomial Error Correction will have to look into FPGA implementations to mitigate the delays encountered in software implementations.

References

[1] Arazi, Benjamin, A Commonsense Approach to the Theory of Error Correcting Codes,_MIT Press, 1988.

[2] Hoffman, D. G. et al. Coding Theory - the Essentials, Marcel Dekker Inc., 1991.

[3]Mann,Henry B., Error Correcting Codes, John Wiley and Sons,1968.

[4] Purser, Michael, _Introduction to Error-Correcting Codes, Artech House Inc., Norwood, MA, 1995.

[5] Irving S., A Class of Multiple- Error-Correcting Codes and Decoding Scheme, MIT Lincoln Laboratory, 1953.

[6] Roman, Steven, Coding and Information Theory, Springer-Verlag, 1992.

[7] Van Lindt, J. H., Introduction to Coding Theory, Springer-Verlag, 1982.

[8] The Art of Error Correcting & Coding – Todd Moon

[9] C.Barrett et al., "Characterizing the Interaction Between Routing and MAC Protocols inAd-hoc Networks," Proc. MobiHoc 2002 , pp. 92-103

[10] J.Broch etal.,"A Performance Comparison of Multi-Hop Wireless AdHocNetworkRouting Protocols," Proc. Mobicom '98.

Received: August, 2010