Hindawi Publishing Corporation
EURASIP Journal on Wireless Communications and Networking
Volume 2010, Article ID 263210, 12 pages
doi:10.1155/2010/263210

Research Article

Design of Packet-Based Block Codes with Shift Operators

Ali Al-Shaikhi[1] and Jacek Ilow[2]

[1] *Department of Electrical Engineering, King Fahd University of Petroleum and Minerals, P.O. Box 1203, Dhahran 31261, Saudi Arabia*
[2] *Department of Electrical and Computer Engineering, Dalhousie University, 1360 Barrington St., P.O. Box 1000 Halifax, NS, Canada B3J-2X4*

Correspondence should be addressed to Ali Al-Shaikhi, shaikhi@kfupm.edu.sa

Received 22 October 2009; Accepted 31 December 2009

Academic Editor: Nicholas Kolokotronis

Copyright © 2010 A. Al-Shaikhi and J. Ilow. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This paper introduces packet-oriented block codes for the recovery of lost packets and the correction of an erroneous single packet. Specifically, a family of systematic codes is proposed, based on a Vandermonde matrix applied to a group of k information packets to construct r redundant packets, where the elements of the Vandermonde matrix are bit-level right arithmetic shift operators. The code design is applicable to packets of any size, provided that the packets within a block of k information packets are of uniform length. In order to decrease the overhead associated with packet padding using shift operators, non-Vandermonde matrices are also proposed for designing packet-oriented block codes. An efficient matrix inversion procedure for the off-line design of the decoding algorithm is presented to recover lost packets. The error correction capability of the design is investigated as well. The decoding algorithm, based on syndrome decoding, to correct a single erroneous packet in a group of $n = k + r$ received packets is presented. The paper is equipped with examples of codes using different parameters. The code designs and their performance are tested using Monte Carlo simulations; the results obtained exhibit good agreement with the corresponding theoretical results.

1. Introduction

Real-time applications are delay sensitive and, in the Internet, are primarily based on user datagram protocol (UDP). Packet-level forward error correction (FEC) is a packet loss recovery technique which does not require retransmissions and allows packet delivery with bounded delay and controllable reliability [1]. In order to protect k information packets, $r = n - k$ additional redundancy packets are also sent. The term "packet" is loosely applied in this context, as in many proposals, packet-level FEC is used at the data-link layer. Packet-level FEC aims at recovering some of the lost packets, where the lost packets originate from erroneous bit transmissions and packet discarding at the lower protocol layers, especially for multihop networks, as well as from congestions in the network and buffer overflows. When packet level FEC is deployed alone to recover from lost packets, the packet loss rate (PLR) is reduced compared to the PLR in the network. However, there is no guarantee that all packets will be recovered at the destination. This is acceptable in some applications like video and audio streaming or multicasting protocols [2].

The parity packets in the existing packet-level FEC schemes are constructed in a similar fashion as parity bits/symbols in the linear block codes used in digital transmission systems, except the bits used in the encoding process are from different packets. The receiver is able to recover up to a certain number of lost/erroneous packets in a block of n transmitted packets governed by the minimum distance of the code [1, 3].

There are a number of powerful and efficient FEC schemes to recover from erasures and/or errors [4], such as low density parity check (LDPC) codes and tornado codes, which use bipartite graphs [2, 5]. Also, Reed Solomon (R-S) codes are used in many applications. However, the codewords in these codes are rather short, and this dictates the construction of the parity packets, which are usually visualized as arranging information packets row-wise and running the FEC code column-wise [6, 7]. An alternative to this is to use the turbo codes to construct the turbo code frame, and then split the frame into packets. This is only feasible in turbo codes because of the large size of the codewords [8].

In practical applications, in addition to erasure/error recovery capability of the code, another important aspect to consider is the complexity of the encoding and decoding processes [9]. This aspect motivates the investigations in this paper, where the only operations permitted on packets are arithmetic packet shifts and binary additions. Particularly, this paper focuses on systematic codes with coefficient matrices based on the Vandermonde or NonVandermonde structures. Both designs, by incorporating packet shifts, facilitate fast matrix-vector multiplication and efficient inversion of submatrices involved in the erasure or erroneous packet recovery processes. In contrast to other systematic erasure codes based on Vandermonde matrices, the proposed codes are not using Vandermonde matrices to manipulate elements (packet fragments) from the Galois field (GF), but rather use them to operate on whole packets by working with their shifts. The benefit of this approach is the lower encoding and decoding complexities of the designs presented in this paper. The proposed codes are maximum distance separable (MDS) and, while maintaining comparable performance as in the more conventional ones, are also quite flexible in the choice of the code rate.

2. Linear Block Codes in Packet-Level FEC

In conventional applications of systematic codes to packet-level FEC, a unit of information, either symbol or bit, m_i, $i = 1, \ldots, k$, is taken from each of the k information packets. These k symbols are used to construct r parity symbols with the help of the coefficient matrix $\mathbb{P}$. The parity symbols are then transmitted in r redundancy packets. The coded/transmitted symbols on $n = r + k$ packets, represented by the column vector $\mathbf{p}$, are calculated at the transmitter (encoder) using the following linear system of equations:

$$\mathbf{G} \cdot \mathbf{m} = \mathbf{p}, \tag{1}$$

where $\mathbf{m}$ is the column vector of k information symbols from k packets, $\mathbf{G} = \begin{bmatrix} \mathbf{I} \\ \cdots \\ \mathbb{P} \end{bmatrix}$ is the $n \times k$ generator matrix, and $\mathbf{I}$ is the $k \times k$ identity matrix. For R-S codes, the matrix multiplication in (1) uses GF arithmetics. In general, if all information packets are of the same length $\mathcal{L}$, (1) is used $\lfloor \mathcal{L}/b \rfloor + 1$ times to construct the coded packets, where b is the number of bits represented in each symbol and $\lfloor \cdot \rfloor$ is the floor operator. The assumption used in this paper is that all packets in the coded group are of the same length. This imposes some limitations which can be overcome by padding the packets to the same length. For brevity of notation, for all symbols in the group of packets, we rewrite (1) using the packet version of this relationship as:

$$\mathbf{G} \cdot \mathbf{M} = \mathbf{P}, \tag{2}$$

where $\mathbf{M}$ is a matrix of information packets arranged in rows, each with $\lfloor \mathcal{L}/b \rfloor + 1$ symbol elements, and $\mathbf{P}$ is the corresponding matrix of coded packets.

At the receiver side, for MDS codes, if there are lost information packets, we have to solve the following system of equations for $\mathbf{M}$:

$$\mathbf{G}^k \cdot \mathbf{M} = \mathbf{P}^k \tag{3}$$

or equivalently, determine

$$\mathbf{M} = \left(\mathbf{G}^k\right)^{-1} \cdot \mathbf{P}^k, \tag{4}$$

where $\mathbf{P}^k$ is the vector of any k received packets, $\mathbf{G}^k$ is a $k \times k$ submatrix of $\mathbf{G}$ with rows corresponding to the k received packets as determined by the received packets sequence numbers, and $(\cdot)^{-1}$ represents the inverse of a matrix. There are many matrix inversion techniques that could be used, such as those based on Cramer's rule procedure, Gaussian elimination, or Gaussian Jordan elimination methods [10–12]. We will present later a suitable technique for our designs for finding the matrix inverse efficiently for the proposed codes.

The main challenge in the design of an erasure code is to determine $\mathbf{G}$, or in the case of systematic codes, the corresponding coefficient matrix $\mathbb{P}$. For MDS codes, the matrix $\mathbf{G}$ should be designed in such a way that any $k \times k$ submatrix, $\mathbf{G}^k$, has to be invertible (full rank). The total number of such submatrices is $\binom{n}{k}$, where $\binom{\cdot}{\cdot}$ represents the n-choose-k operator. The simplest two designs of $\mathbf{G}$ are the repetition code and the single parity check code. In the former case, the code is $(n, 1, n)$ where $\mathbb{P}$ is the column vector of all 1's, while the latter is the code $(k + 1, k, 2)$ which adds one more packet consisting of the parity check of all information packets; that is, $\mathbb{P}$ is the row vector of all 1's. These two codes, though simple to use, are not the best, since the former has a low rate while the latter recovers at most only one missing packet. In the case of the systematic codes considered in this paper, there are other matrices that could be used as $\mathbb{P}$ resulting in the desired properties of $\mathbf{G}^k$ being invertible, such as the Cauchy and Vandermonde matrices. We will discuss next the Vandermonde matrix which is used as the coefficient matrix $\mathbb{P}$ in the R-S erasure codes. The Vandermonde matrix is also utilized in the design of the proposed code in this paper, but with a different building element than in the case of R-S codes.

The Vandermonde matrix $\mathbf{V}$, with $r \times k$ elements, is given by the following [13]:

$$\mathbf{V} = \begin{bmatrix} 1 & \alpha_0 & \alpha_0^2 & \cdots & \alpha_0^{k-1} \\ 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^{k-1} \\ 1 & \alpha_2 & \alpha_2^2 & \cdots & \alpha_2^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_{r-1} & \alpha_{r-1}^2 & \cdots & \alpha_{r-1}^{k-1} \end{bmatrix}. \tag{5}$$

This matrix is proven to be nonsingular if the parameters α_z, for $z = 0, \ldots, r-1$, are distinct. For R-S erasure codes with $\mathbb{P} = \mathbf{V}$, the α_z elements are taken from the extended GF, GF(p^b), where p is a prime number ($p = 2$ is used most of the time) and b is any integer ($b = 8$ is

used for highest efficiency to represent a byte). Therefore, multiplication and addition operations in (2) and in (4) must be done on that extended GF. As a result, the following problems are encountered when working with R-S codes for packet-level FEC: (i) the code rates (parameters) are limited and (ii) the encoding and decoding processes are computationally intensive. Moreover, it has been shown that the Vandermonde matrix, based on the elements taken from the finite GF, is not always nonsingular [13–15].

In the rest of the paper we deal with the processing of packets; however, some of the concepts involved are very close to symbol processing in FEC codes. Therefore, we will follow commonly accepted symbols and terms to denote corresponding operations.

3. Vandermonde Matrix-Based Binary Erasure Code Design

In this section, we present a code design using a coefficient matrix $\mathbb{P}$, based on the Vandermonde matrix, which (i) results in computationally efficient processing of long packets, and (ii) possesses the desired properties when performing the decoding procedure in the proposed MDS code using (4). We choose the α_z elements in the Vandermonde matrix as in (5) to be x^z, for $z = 0, \ldots, r-1$, where $x^z \cdot M_i$ stands for a right arithmetic shift operator by z bits applied to the row information packet M_i, $i = 1, \ldots, k$ represented in bits from now on. Therefore, the coefficient matrix, $\mathbb{P}$, in our code is given by:

$$\mathbf{V} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & x^1 & x^2 & \cdots & x^{k-1} \\ 1 & x^2 & x^4 & \cdots & x^{2(k-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x^{r-1} & x^{2(r-1)} & \cdots & x^{(r-1)(k-1)} \end{bmatrix}. \tag{6}$$

To ensure that this Vandermonde matrix, consisting of the arithmetic shifts operators x^z, when applied to the information packets, results in packets that can be recovered, the packet size has to be increased by at least $(r-1)(k-1)$ over the original information packet size $\mathcal{L}$ by zero padding these packets to the size of $ps = \mathcal{L} + (r-1)(k-1)$ [4]. This packet size ensures that the arithmetic shifts implement delay (not cyclic shifts). Therefore, the overall effective rate for $(n, k, d_{\min})$ code is $(k \cdot \mathcal{L})/(n \cdot ps)$. For example, when applying the proposed design to Ethernet frames with transmission units of 1500 bytes and using the code $(10, 5, 6)$, the effective code rate is $(5 \cdot 1500)/(10 \cdot 1502) = 0.4993$ which is close to the conventional rate 1/2 of this code. With the proposed coefficient matrix as in (6), based on (2) and (4), the encoding process and packet loss recovery process in the proposed codes are described in Sections 3.1 and 3.2, respectively.

3.1. Encoding Process. The Vandermonde-based matrix that is augmented with the identity matrix (systematic code) comprise the $n \times k$ generator matrix of the code which is given by:

$$\mathbf{G} = \begin{bmatrix} \mathbf{I}_{k\times k} \\ \cdots \\ \mathbf{V}_{r\times k} \end{bmatrix}, \tag{7}$$

where $\mathbf{V}$ is the designed matrix. For the proposed $(n, k, n-k+1)$ systematic MDS code, the encoder uses (2) and (7) to get:

$$\begin{bmatrix} & \mathbf{I}_{k\times k} & \\ 1 & 1 & \cdots & 1 \\ 1 & x^1 & \cdots & x^{k-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x^{r-1} & \cdots & x^{(r-1)(k-1)} \end{bmatrix} \cdot \mathbf{M} = \mathbf{P}, \tag{8}$$

where P_i $(i = 0, \ldots, n-1)$ are the n coded packets comprising $\mathbf{P}$, where the first k of them are the original information packets. The remaining $n-k$ packets are generated by modulo-2 addition of all the k information packets after a proper shift of each information packet. Because of similarities with the construction of codewords in cyclic codes [6], it is natural to interpret (8) as a system of polynomial equations. In this system of algebraic equations, with M_i being represented as a polynomial $M_i(x)$, the product $x \cdot M_i(x)$ is a right shift, where $x^i \cdot M_i(x)$ is the right shift by i bits. The addition of polynomials with binary coefficients corresponding to bits in the packets is a modulo-2 addition. For brevity of notation, we will use M_i to describe the polynomial representation of a packet.

As compared to conventional encoding processes based on (1) and (2), the encoding process in (8) is fast and efficient because it uses just shifting and modulo-2 addition operations of packets. This may benefit hardware implementation of the proposed packet coding or, in software implementation, reduce the number of memory accesses. Because of the invertibility properties of the Vandermonde matrix and its submatrices concatenated with the identity matrix, when we get any k coded packets out of n transmitted ones, all the original k information packets can be recovered, at the receiver side [13]. Therefore, the minimum distance of such code is $d_{\min} = n - k + 1$, which is an MDS code with a small overhead because each packet is padded with $(r-1)(k-1)$ zeros. These codes can correct for $t = \lfloor (d_{\min} - 1)/2 \rfloor = \lfloor (n-k)/2 \rfloor$ errors or $e = d_{\min} - 1 = n - k$ erasures [1].

3.2. Decoding Process. Initially we assume that the packet is either received correctly or lost. Assume that out of the received packets, some are information packets P_{i_p} indexed by $i_p \in [0, k-1]$ and some are parity packets P_{p_p} indexed by $p_p \in [k, n-1]$. If the total number of packets received is greater than or equal to k, k of these

packets, including all the received information packets, are used to construct the vector $\mathbf{P}^k$ in (4). The submatrix $\mathbf{G}^k$ is obtained from the generator matrix $\mathbf{G}$ by knocking off the rows of $\mathbf{G}$ corresponding to packets not used or lost during transmission. Moreover, since some of these received k packets are information packets, their corresponding rows and columns in $\mathbf{G}^k$ can be removed so that one would be calculating only the missing information packets $\mathbf{M}^L$. As opposed to (4), this can be accomplished using the reduced system of equations given by:

$$\mathbf{M}^L = \left(\mathbf{G}^L\right)^{-1} \cdot \left(\hat{\mathbf{P}}\right)^L, \tag{9}$$

where $(L \leq k)$ and $\hat{\mathbf{P}}^L$ is the received parity packet after substituting properly for the received information packets excluded from the recovery. Essentially, $\mathbf{G}^L$ can be any square submatrix of $\mathbf{V}$. When $\mathbf{V}$ is a square matrix, the number of individual $L \times L$ submatrices is calculated using the following:

$$y = \binom{k}{L}. \tag{10}$$

The subsystem of equations in (4) or (9) always has a unique solution, or simply $(\mathbf{G}^k)^{-1}$ is invertible, because of the geometric progression nature in each row of the Vandermonde matrix when the elements are the shift operators. Even though this is not a formal proof for the invertibility of $(\mathbf{G}^k)^{-1}$, we verified this, by simulation, for a large set of parameters n and k [4].

4. Efficient Implementations of the Design

In this section, we present a two-step efficient implementation for recovering the lost packets $\mathbf{M}^L$ or equivalently finding $(\mathbf{G}^L)^{-1}$ in (9). At the receiver side, from (9), we get:

$$\frac{1}{|\mathbf{G}^L|} \cdot \left(\text{adj}\mathbf{G}^L\right) \cdot \hat{\mathbf{P}}^L = \mathbf{M}^L, \tag{11}$$

where $\text{adj}\mathbf{G}^L$ is the adjoint matrix of $\mathbf{G}^L$, which is the transpose of the minors of $\mathbf{G}^L$ since the cofactor matrix equals the minors matrix in the binary field. To find $\mathbf{M}^L$, the $\text{adj}\mathbf{G}^L$ must be found first, multiplied by $\hat{\mathbf{P}}^L$, and then the result is divided by $|\mathbf{G}^L|$. A basis for the proposed two-step procedure to find $\mathbf{M}^L$ is an equivalent representation of (11) written as:

$$\left(\text{adj}\mathbf{G}^L\right) \cdot \hat{P}^L = \left|\mathbf{G}^L\right| \cdot \mathbf{M}^L \tag{12}$$

Therefore, the first step in finding $\mathbf{M}^L$ is to solve the LHS of (12) which is essentially finding the $\text{adj}\mathbf{G}^L$ efficiently. Since the elements of the adjoint matrix are polynomials (shifts) with coefficients from the binary field, it is not that complex to calculate the LHS of (12). The RHS of (12) dictates that the result obtained from the LHS is $\mathbf{M}^L$ multiplied by $|\mathbf{G}^L|$. Therefore, the second step in finding $\mathbf{M}^L$ is to extract $\mathbf{M}^L$ efficiently from the result in the first step.

Next, we will show how to find the $\text{adj}\mathbf{G}^L$ and $|\mathbf{G}^L|$ which comprise the matrix inverse and then how to extract $\mathbf{M}^L$.

4.1. Efficient Calculation of Matrix Inverse. The most expensive operation in the recovery of $\mathbf{M}^L$ is to find the determinant and the adjoint matrix of $\mathbf{G}^L$ which is any submatrix of $\mathbf{V}$. As explained in Section 2, the elements of the adjoint matrix can be found using the determinants of the submatrices of $\mathbf{G}^L$. Therefore, finding a way to calculate the determinants results in computing the inverse. Although the original Vandermonde matrix has a known formula to find its determinant and its adjoint matrix, most of the submatrices are no longer Vandermonde matrices and such formulas donot apply to them. We present an efficient way to find the inverse of any submatrix of $\mathbf{V}$. This method arises because our elements in the Vandermonde matrix are monomials (single term polynomials) representing shifts. Therefore, we are actually interested in the powers of the elements in the designed matrix and submatrices. We demonstrate the principle for calculating underlying matrices inverse in the case of $\mathbf{V}$. By applying the logarithmic operator to each element of $\mathbf{V}$ and removing the common factor $\log(x)$, we get the required shifts (monomials order representation) as follows:

$$\begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 2 & \cdots & k-1 \\ 0 & 2 & 4 & \cdots & 2(k-1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & r-1 & 2(r-1) & \cdots & (r-1)(k-1) \end{bmatrix}. \tag{13}$$

To find the required determinants, we apply the permutation technique [16]: write down all permutations of $(1,\ldots,l)$, denoted by $\{P_r(l)\}$ of cardinality $l!$, where ! represents factorial operator and take each permutation as the subscripts of the letters $a, b, \ldots$ which are the rows of the matrix and sum with signs determined by $(-1)^{y(p_r(l))}$, where $y(p_r(l))$ is the number of permutation inversions in $P_r(l)$. However, we donot need the permutation inversions since in the binary field the digits 1 or -1 are both 1. For example, with $l = 3$, the permutations and the number of inversions they contain are 123(0), 132(1), 213(1), 231(2), 312(2), and 321(3), so the determinant of $\begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix}$ is $(a_1b_2c_3 - a_1b_3c_2 - a_2b_1c_3 + a_2b_3c_1 + a_3b_1c_2 - a_3b_2c_1)$ which is equivalent to $(a_1b_2c_3 + a_1b_3c_2 + a_2b_1c_3 + a_2b_3c_1 + a_3b_1c_2 + a_3b_2c_1)$ in the binary field. When working with matrices of monomial elements, after applying the logarithmic operator, the determinant of the resultant powers can be represented:

$$\begin{aligned} &a_{1'} + b_{2'} + c_{3'}, \quad a_{1'} + b_{3'} + c_{2'}, \quad a_{2'} + b_{1'} + c_{3'}, \\ &a_{2'} + b_{3'} + c_{1'}, \quad a_{3'} + b_{1'} + c_{2'}, \quad a_{3'} + b_{2'} + c_{1'}, \end{aligned} \tag{14}$$

where $(\cdot)'$ represents the logarithmic value of $(\cdot)$. Therefore, the determinant of (13) can be calculated in a simplified way as in (14). This simplified procedure can be used to calculate the determinant of any square matrix. The routine in Algorithm 1 illustrates how to find the adjoint matrix using the determinant technique in (14) for any Vandermonde submatrix.

Given
  $\mathbf{G}^L$*; The* $L \times L$ *matrix of interest*
Find
  $(\mathbf{G}^L)'$*; The submatrix transpose*
  P_r*; The permutations of* $(1, \ldots, L-1)$ *for* $(\mathbf{G}^L)'$
Then
  For the adjoint element in location (i, j) *of* $(\mathbf{G}^L)'$
    Construct the $(L-1) \times (L-1)$ *matrix by deleting row i and column j*
    Find the determinant using then technique that uses P_r
    The numbers with even multiples are canceled out
    The numbers with odd multiple stay
End

Algorithm 1: A routine to find the adjoint matrix using the permutation technique.

Given
  $\mathbf{G}^L$*; The* $L \times L$ *matrix of interest*
  $\mathbf{M}^S$*;* $\mathbf{M}^S = |\mathbf{G}^L| \cdot \mathbf{M}^L$ *where* $\mathbf{M}^L$ *is unknown*
  $\mathbf{M}_i^S$*; The packet at row in* $\mathbf{M}^S$
  $\mathbf{M}_i$*; The packet at row in* $\mathbf{M}^L$
  p_s*; The packet size after padding with* g_{max}
Find
  g*; The vector of the powers of* $|\mathbf{G}^L|$ *in ascending order*
Then
  For m=1: ps
    For k=2 : length (g)
      Factor=g(k)−g(1)
      J1=modulus(m−1, ps)
      J2=modulus(Factor+m −1, ps)
      $\mathbf{M}_i^S(1, J2+1) = x$ *or* $(\mathbf{M}_i^S(1, J1+1), \mathbf{M}_i^S(1, J2+1))$
    End
  End
$\mathbf{M}_i$*=Cyclically shift to the left* $\mathbf{M}_i^S$ *with* $g(1)$

Algorithm 2: A routine to extract $\mathbf{M}^L$ from the product of $\mathbf{M}^L$ with $|\mathbf{G}^L|$.

4.2. Extracting the Information from the Information Modified by the Determinant. After obtaining adj$\mathbf{G}^L$ and $|\mathbf{G}^L|$, we can calculate the LHS of (12). By doing that, we will obtain the matrix with its rows dependent only on the corresponding rows of $\mathbf{M}^L$. This dependence implies shifts and additions determined by the $|\mathbf{G}^L|$. To be able to find the $\mathbf{M}^L$, we modify the condition that we pad the information packets with $(r-1)(k-1)$ zeros into padding with the maximum degree of the determinant in the set of all Vandermonde submatrices, denoted by $g_{\max}$. This padding is actually greater than the original padding since $g_{\max} > (r-1)(k-1)$. However, it will not further decrease the effective rate of the code since this extra padding can be done at the receiver side. Each unknown packet in $\mathbf{M}^L$ is found by sequential bit by bit recovery within this packet by removing the effect of the previously found bit as shown in Algorithm 2 [4].

The examples in Section 5.1 will clarify further the proposed techniques in Sections 4.1 and 4.2.

5. Erasure Code Designs and Performance

In this section, we discuss some specific cases for the design of erasure codes proposed in this paper. We demonstrate the PLR reduction capability of these codes through analytical calculations and simulation results for different code parameters.

5.1. Erasure Code Design. At the receiver side, we need to find the inverse of any $L \times L$ submatrix of $\mathbf{P}$ where $L \leq k$. When we receive only the parity packets, we need to find $\mathbf{V}^{-1}$ which is given by:

$$\mathbf{V}^{-1} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & x & x^2 \\ 1 & x^2 & x^4 \end{bmatrix}^{-1} = \frac{1}{x^5 + x} \begin{bmatrix} x^5 + x^4 & x^4 + x^2 & x^2 + x \\ x^4 + x^2 & x^4 + 1 & x^2 + 1 \\ x^2 + x & x^2 + 1 & x + 1 \end{bmatrix}. \tag{15}$$

The elements of the adjoint matrix as given in the RHS of (15) can be found by running the algorithm described previously. In this algorithm, in Step I, we apply the logarithmic operator of the matrix and remove the common factor $\log x$, while in Step II we calculate the determinants with polynomial powers represented in parenthesis $(\cdot, \cdot)$ as follows:

$$\begin{aligned} \operatorname{adj} \begin{bmatrix} 1 & 1 & 1 \\ 1 & x & x^2 \\ 1 & x^2 & x^4 \end{bmatrix} &\overset{\text{Step I}}{\Longrightarrow} \operatorname{adj} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 2 & 4 \end{bmatrix} \\ &\overset{\text{Step II}}{\Longrightarrow} \begin{bmatrix} (5,4) & (4,2) & (2,1) \\ (4,2) & (4,0) & (2,0) \\ (2,1) & (2,0) & (1,0) \end{bmatrix}. \end{aligned} \tag{16}$$

For example, the element $(5, 4)$ is found by calculating the determinant after crossing the first row and the first column of $\mathbf{V}$ and then applying (14) but for the resulting 2×2 matrix. It is $(4 + 1 = 5, 2 + 2 = 4)$. To find the determinant which is the denominator polynomial in (15), we gather the elements of any row or any column of the adjoint matrix after

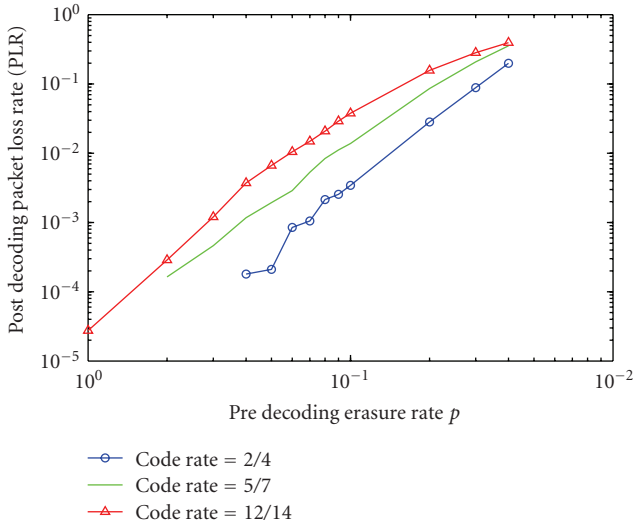

Figure 1: Packet loss rate for systematic codes with $d_{\min} = 3$.

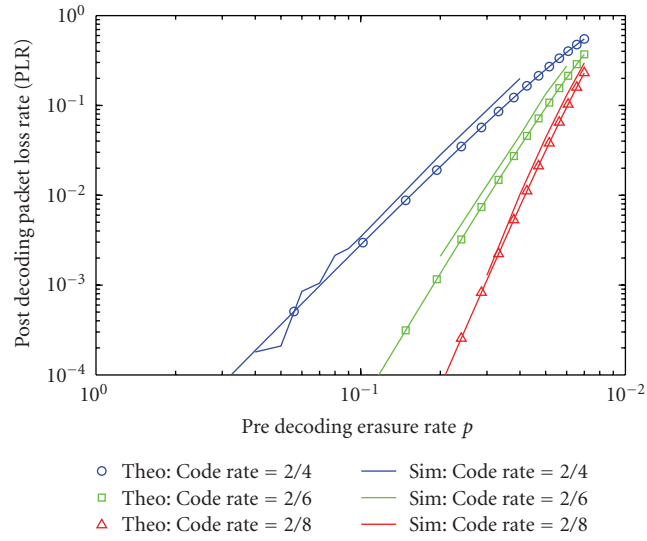

Figure 2: Packet loss rate for systematic codes with $k = 2$ but different $d_{\min}$.

adding to them their corresponding original powers (shifts) which appear in the middle term in (16). Then, any number (including zero) with even multiples is canceled out and any number (including zero) with odd multiples stays because we operate in the binary field. For example, by considering the second row in the RHS of (16), we add to the elements their corresponding powers which are 0, 1, and 2, respectively, and then we gather them to be $(4, 2, 5, 1, 4, 2)$. Then, 4 and 2 are canceled out because they have even multiples, and 5 and 1 stay because they have odd multiples. This results in $(5, 1)$ which is the power of the determinant shown in the denominator polynomial in (15). By applying this principle, any submatrix can be inverted by simple additions only.

The choice for the parameters of the proposed code is quite flexible. For example, we can design the code $(5, 2, 4)$ having a rate of 2/5. By receiving any two packets, we can recover the remaining three packets. Another example is the code $(5, 3, 3)$ having a rate of 3/5. By receiving any three packets, we can recover the remaining two packets.

5.2. Simulation Results. This section shows post decoding packet loss recovery performance, $\mathrm{PLR}_{\mathrm{post}}$, of the proposed codes for a wide range of raw PLR in the network. As discussed earlier, we assume that we either receive the bits/packets correctly or they are missing (in doubt).

The presented results are obtained through the simulations of the actual encoding, network packet loss and decoding processes. The PLR recovery simulation results are plotted using continuous unmarked lines in all figures in this section. For comparison purposes, we also plot the theoretical curves using continuous marked lines based on the following formula [6]:

$$\mathrm{PLR}_{\mathrm{post}} \approx \frac{1}{n} \sum_{i=e+1}^{n} i \binom{n}{i} p^i (1-p)^{n-i}, \tag{17}$$

where p represents the raw PLR. The PLR performance is independent of the actual packet length; however the latter determines the percentage of overhead related to padding the packets to the desired length determined by $g_{\max}$. Also, since the performance of the designed codes is characterized by the minimum distance, it is not necessary to compare it with the performance of other codes.

Figure 1 shows the PLR performance for three codes with the same minimum distance of three and the same packet size of 1000 bits but different code rates. The three codes have the parameters $(4, 2, 3)$, $(7, 5, 3)$, and $(14, 12, 3)$. The rates of these codes are 2/4, 5/7, and 12/14, respectively. We can observe that the performance improves as the code rate decreases because the codes can recover two packets in a group of n coded packets where $n = 4, 7,$ and 14.

Figure 2 shows the PLR performance for three codes with different minimum distances and different rates. The three codes have the parameters $(4, 2, 3)$, $(6, 2, 5)$, and $(8, 2, 7)$. The rates of these codes are 2/4, 2/6, and 2/8, while the minimum distances are 3, 5, and 7, respectively. We can observe that the performance improves as the code rate decreases because they can recover 2, 4, and 6 packets, respectively. We can observe also that the theoretical PLR performances as given by (17) agree with the simulation results.

6. Modified Erasure Designs

In this section, two modifications are introduced in order to lower the amount of zero padding needed. In the first modification, the shift elements are chosen and positioned in the parity matrix $\mathbb{P}$ such that the determinant of the new matrix $\mathbf{U}$, replacing $\mathbf{V}$, has a lower degree. A lower degree determinant implies less zero padding for the packets and hence a reduced overall overhead. The new parity matrix $\mathbf{U}$ is such that all its submatrices are invertible. We show some of the matrix designs for a number of different sizes. We prove

that the new designed matrices and their submatrices are invertible by finding the inverses using simulations. Also, the maximum degree determinant is calculated for **U**. A comparison to the same size Vandermonde-based designs is shown.

6.1. Various Sizes Matrix Designs. The best design found that satisfy the invertibility condition using exhaustive search for the 3×3 matrix is

$$\mathbf{U} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & x & x^2 \\ 1 & x^2 & x \end{bmatrix}. \tag{18}$$

The matrix in (18) is a nonVandermonde matrix. The invertibility of this matrix and its submatrices is proven by using brute force simulations. This is done by finding all the submatrices of (18) and calculating the determinants of these submatrices. For this matrix, we found inverses for one 3×3 matrix, nine 2×2 submatrices, and nine 1×1 submatrices. The number of submatrices that have inverses complies with the maximum number in (10), meaning that the design is invertible for any submatrix. This design has a maximum degree determinant of four compared to its corresponding Vandermonde matrix design which has a maximum degree determinant of five.

The two designs

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & x & x^2 & x^3 \\ 1 & x^2 & x^4 & x^6 \\ 1 & x^3 & x & x^4 \end{bmatrix}, \quad \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & x & x^2 & x^3 & x^4 \\ 1 & x^2 & x^4 & x^6 & x^8 \\ 1 & x^3 & x^6 & x^2 & x^7 \\ 1 & x^4 & x^8 & x^5 & x \end{bmatrix} \tag{19}$$

are good candidates for the parity coefficient matrices of sizes 4×4 and 5×5, respectively. These two matrices are nonVandermonde matrices and invertible. The 4×4 design has a maximum degree determinant of 11 compared to its corresponding Vandermonde matrix design, which has a maximum degree determinant of 14. The 5×5 design has a maximum degree determinant of 21 compared to its corresponding Vandermonde matrix design, which has a maximum degree determinant of 30.

For the 6×6 matrix, the best design found is given by

$$\mathbf{U} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & x & x^2 & x^3 & x^4 & x^5 \\ 1 & x^2 & x & x^6 & x^8 & x^{10} \\ 1 & x^3 & x^6 & x & x^5 & x^8 \\ 1 & x^4 & x^8 & x^5 & x^3 & x \\ 1 & x^5 & x^{10} & x^8 & x & x^6 \end{bmatrix}. \tag{20}$$

The matrix in (20) is also a nonVandermonde matrix. For each square size matrix, Table 1 shows the number of submatrices, the maximum degree determinant among them, and a comparison with the corresponding same size Vandermonde design. The number of submatrices having inverses complies with the maximum number in (10). This design has a maximum degree determinant of 33 compared to its corresponding Vandermonde matrix design, which has a maximum degree determinant of 55.

Table 1: The Maximum Degree Determinant in 6×6 Matrix Design.

	Vandermonde	nonVandermonde	# of Submatrices
1×1	25	10	36
2×2	41	20	225
3×3	50	26	400
4×4	54	32	225
5×5	55	33	36
6×6	55	33	1

Higher dimension matrices can also be designed and found in the same manner by generating the elements of the required size matrix and then testing the invertibility of each submatrix using brute force simulation.

The second modification that also will reduce the amount of zero padding is to zero pad with the maximum shift in the designed matrix, not with the maximum degree of the determinant. At the encoder side, each packet will be padded with the maximum shift in the matrix. Then at the receiver side, before starting decoding, the received packets are extra padded with zeros to make the total number of zero padding equal to the maximum degree determinant. This reduces the amount of overhead in the transmitted packets. This modification applies for any design (Vandermonde or nonVandermonde), and the advantages benefit equally both modifications. For example, for a 6×6 Vandermonde matrix, 55 zeros are needed originally, while only 25 zeros are needed if we adopt the second modification, since the maximum shift in the Vandermonde matrix design is 25. For a 6×6 nonVandermonde matrix, 33 zeros are needed originally, while only 10 zeros are needed if we adopt the second modification, since the maximum shift in the nonVandermonde matrix design in (20) is 10.

6.2. Simulation Results. This section shows post decoding packet loss recovery performance, PLR_{post}, of the modified nonVandermonde codes for a wide range of raw PLR in the network. For comparison purposes, the performance of the corresponding Vandermonde based designs are also plotted.

Figure 3 shows the PLR performance for five codes with different minimum distances, but the same code rate of 1/2 and the same packet size of 1000 bits. The five codes have the parameters $(4, 2, 3)$, $(8, 4, 5)$ Vandermonde-based, $(8, 4, 5)$ nonVandermonde based (Modified), $(12, 6, 7)$ Vandermonde-based, and $(12, 6, 7)$ nonVandermonde based (Modified). The minimum distances are 3, 5, and 7, respectively. By receiving any k packets, each code can recover the remaining $r = k$ packets. As the channel condition improves, the code with higher parameters outperforms the others since it can recover more packets. This is the reason

that the performance improves as the minimum distance of the code increases. Also, we observe that the modified designs (nonVandermonde) and the original designs (Vandermonde) have identical performances.

7. Error Correction Capability and Performance

In this section, the general error correction capability and the decoding process using the designed codes are presented. An error decoding technique capable of correcting a single erroneous packet irrespective of the number of errors in this packet is presented. We demonstrate the packet error rate (PER) reduction capability of these codes based on the proposed error decoding technique through analytical calculations and simulation results for different code parameters.

We assume here that there is no packet loss. Therefore, at the receiver, all the coded packets **P** are received. From the received packets arranged row-wise in a matrix **R**, we have to infer first which packet(s) is(are) in error, and then, within this(these) packet(s), where the error locations are and their values. For binary codes considered in this paper, the error values are not required, since by knowing their positions, one just flips them. There are many procedures that could correct for errors by observing **R**. A typical way is to use syndrome decoding which proceeds by finding the parity check matrix **H**. The parity check matrix, **H**, of **G** in (7) is the $n \times r$ matrix given by

$$\mathbf{H} = \begin{bmatrix} \mathbf{V}'_{k\times r} \\ \cdots \\ \mathbf{I}_{r\times r} \end{bmatrix}, \tag{21}$$

where $\mathbf{V}'$ is the transpose of **V**. Accordingly, the multiplication of the two partitioned matrices, $\mathbf{H}'$ and **G** gives the zero matrix.

Now assume that the n coded packets arranged in **P** are transmitted and they are corrupted by errors. The received packets can be viewed as the coded packet corrupted (modulo-2 added) with packets having 1's in the error locations. These packets are referred to as error packets and are arranged row-wise in **E**, where the latter is a column vector consisting of the elements E_i, $i = 0, \ldots, n-1$. The received packets arranged in the matrix **R** are given by

$$\mathbf{R} = \mathbf{P} + \mathbf{E} = \begin{bmatrix} P_0 \\ \vdots \\ P_{k-1} \\ \vdots \\ P_{n-1} \end{bmatrix} + \begin{bmatrix} E_0 \\ \vdots \\ E_{k-1} \\ \vdots \\ E_{n-1} \end{bmatrix}, \tag{22}$$

where **R** is a column vector consisting of the received packet R_i $(i = 0, \ldots, n-1)$. By pre-multiplying (22) by $\mathbf{H}'$, one gets the packet syndrome denoted by **S** as follows:

$$\mathbf{S} = \mathbf{H}' \cdot \mathbf{R} = \mathbf{H}' \cdot \mathbf{E}. \tag{23}$$

If $\mathbf{S} \neq \mathbf{0}$, we have the indication that there were errors during the transmission. It is observed that this syndrome decoding technique in (23) depends only on the error patterns, **E**, but not on the transmitted coded packets, **P**. The syndrome decoding technique enables the code to correct for t packet(s) irrespective of the number of bits in error inside the packet(s). A packet is considered in error if at least one bit of the packet is in error.

We show next how the syndrome decoding is utilized to correct a single erroneous packet in a group of n received packets. Extending the technique to correct for more erroneous packets needs further study and is beyond the scope of this paper.

For the MDS codes capable of correcting single erroneous packet out of n packets, $d_{\min} = n - k + 1 = 3$; that is, $r = 2$ and thus $n = k + 2$. Now we pre-multiply the resultant syndrome equation in (23) by the matrix **Q**, which is the column-wise reverse of **H** [4]. We call the resultant matrix, the error locator matrix **W**, which is given as follows:

$$\mathbf{W} = \mathbf{Q} \cdot \mathbf{H}' \cdot \mathbf{E}, \tag{24}$$

where

$$\mathbf{Q} = (\mathbf{H})^{\mathrm{cr}} = \begin{bmatrix} ((\mathbf{V}')^{\mathrm{cr}})_{k\times r} \\ \cdots \\ ((\mathbf{I})^{\mathrm{cr}})_{r\times r} \end{bmatrix} \tag{25}$$

and $(\cdot)^{\mathrm{cr}}$ represents column-reversed matrix. By substituting (25) in (24), one gets (26):

$$\begin{aligned} \mathbf{W} &= \begin{bmatrix} ((\mathbf{V}')^{\mathrm{cr}})_{k\times r} \\ \cdots \\ ((\mathbf{I})^{\mathrm{cr}})_{r\times r} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{V}_{r\times k} \ \vdots \ \mathbf{I}_{r\times r} \end{bmatrix} \cdot \mathbf{E} \\ &= \begin{bmatrix} ((\mathbf{V}')^{\mathrm{cr}})_{k\times r} \cdot \mathbf{V}_{r\times k} & \vdots & ((\mathbf{V}')^{\mathrm{cr}})_{k\times r} \cdot \mathbf{I}_{r\times r} \\ \cdots & \vdots & \cdots \\ ((\mathbf{I})^{\mathrm{cr}})_{r\times r} \cdot \mathbf{V}_{r\times k} & \vdots & ((\mathbf{V})^{\mathrm{cr}})_{r\times r} \cdot \mathbf{I}_{r\times r} \end{bmatrix} \cdot \mathbf{E} \\ &= \begin{bmatrix} ((\mathbf{V}')^{\mathrm{cr}})_{k\times 2} \cdot \mathbf{V}_{2\times k} & \vdots & ((\mathbf{V}')^{\mathrm{cr}})_{k\times 2} \\ \cdots & \vdots & \cdots \\ ((\mathbf{V})^{\mathrm{rr}})_{2\times k} & \vdots & ((\mathbf{I})^{\mathrm{cr}})_{2\times 2} \end{bmatrix} \cdot \mathbf{E} \end{aligned} \tag{26}$$

where $(\cdot)^{\mathrm{rr}}$ represents row-reversed matrix. By carrying out the calculation in (26), one gets (27)

$$\mathbf{W} = \begin{bmatrix} 0 & 1+x & \cdots & \cdots & 1+x^{k-1} & \vdots & 1 & 1 \\ x+1 & 0 & x+x^{k-2} & \cdots & x+x^{k-1} & \vdots & x & 1 \\ \vdots & \cdots & \ddots & \cdots & \vdots & \vdots & x^2 & 1 \\ x^{k-2}+1 & x^{k-2}+x & \cdots & 0 & x^{k-2}+x^{k-1} & \vdots & \vdots & \vdots \\ x^{k-1}+1 & \cdots & \cdots & x^{k-1}+x^{k-2} & 0 & \vdots & x^{k-1} & 1 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \vdots & \cdots & \cdots \\ 1 & x & x^2 & \cdots & x^{k-1} & \vdots & 0 & 1 \\ 1 & 1 & 1 & \cdots & 1 & \vdots & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} E_0 \\ E_1 \\ E_2 \\ \vdots \\ E_{k+1} \end{bmatrix} \tag{27}$$

From (27), we observe that the diagonal elements of $\mathbf{Q} \cdot \mathbf{H}'$ are zeros. This means that if all the error packets are zeros except one error packet E_i, the only all-zero row in $\mathbf{W}$ will be the row W_i, where $i \in [0, n-1]$. Also, we notice that E_i corresponds to the last row in the error locator matrix $\mathbf{W}$. This is true except for the last one $E_i = E_{n-1} = E_{k+1}$, in which case any row in the error locator matrix is the error packet.

We would like to mention that the above technique needs more careful processing to handle the scenario that the error packet and a shifted version of it produce the same packet such as the all-ones error packet. Although the occurrence of such scenario is extremely small, especially for long packets, it can be handled by padding the packets resulting from the syndrome equation with $(r-1)\cdot(k-1)$ zeros and then discarding these zeros when finding the error packet. These zeros are not counted as an overhead since they are padded at the receiver side.

7.1. Error Correction Designs. In this section, we discuss some specific cases for the design of packet-level error correction codes proposed in this paper. First, we consider the $(4, 2, 3)$ systematic code and present the decoding process without using $\mathbf{W}$, and then we demonstrate the benefits of error locator matrix in this example. This code is capable of correcting one packet in error out of the received four packets. The generator and the parity check matrices are $\mathbf{G} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & x \end{bmatrix}$ and $\mathbf{H} = \begin{bmatrix} 1 & 1 \\ 1 & x \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$, respectively. At the receiver side, the decoding process starts by applying the syndrome decoding in (23) to get the following:

$$\mathbf{S} = \mathbf{H}' \cdot \mathbf{E} = \begin{bmatrix} E_0 + E_1 + E_2 \\ E_0 + xE_1 + E_3 \end{bmatrix}. \tag{28}$$

From (28), if the error occurs in the first received packet R_0, the only packet that is not all-zero is E_0 while $E_1 = E_2 = E_3 = \mathbf{0}$. Therefore, the two packets comprising the syndrome matrix $\mathbf{S}$, in (28), are identical and are the error packet, E_0, itself. To correct the erroneous packet R_0, one adds to it one of the packets obtained from the syndrome calculation. If the error occurs in the second received packet R_1, the second packet of the syndrome is a shifted version by one of the first packet in the syndrome. To correct the erroneous packet R_1, add it to the first packet E_1 of the syndrome matrix. If the error occurs in the third received packet R_2, the second packet of the syndrome is the all-zero packet, while the first packet in the syndrome is the error packet E_2. To correct the erroneous packet R_2, add to it the first packet of the syndrome matrix. If the error occurs in the fourth received packet R_3, the second packet of the syndrome is the error packet E_3 while the first packet is the all-zero packet. To correct the erroneous packet R_3, add it to the second packet of the syndrome matrix.

The above correction can be done more efficiently by finding the error locator matrix $\mathbf{W}$ using (24). By pre-multiplying the resultant syndrome equation in (28) by $\mathbf{Q}$, $\mathbf{W}$ is found to be

$$\begin{aligned} \mathbf{W} &= \mathbf{Q} \cdot \begin{bmatrix} E_0 + E_1 + E_2 \\ E_0 + xE_1 + E_3 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 1 \\ x & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} E_0 + E_1 + E_2 \\ E_0 + xE_1 + E_3 \end{bmatrix} = \begin{bmatrix} E_1 + xE_1 + E_2 + E_3 \\ xE_0 + E_0 + xE_2 + E_3 \\ E_0 + xE_1 + E_3 \\ E_0 + E_1 + E_2 \end{bmatrix}. \end{aligned} \tag{29}$$

The above error locator matrix reduces to the first, second, third, or fourth column in Table 2 when the packet in error is the first, second, third, or fourth one, respectively. For example, if the packet in error is the fourth received packet R_3, E_3 will be nonzero packet while $E_0 = E_1 = E_2 = 0$. Therefore, based on (29), $\mathbf{W}$ reduces to the fourth column in Table 2. This means that if we get $\mathbf{W}$ with the last row comprised of all-zeros, we decide that the erroneous packet is the fourth one. In this case, E_3 can be taken either as the first, second, or third packet in the fourth column of $\mathbf{W}$. When the packet in error is the first, second, or third one, we notice as

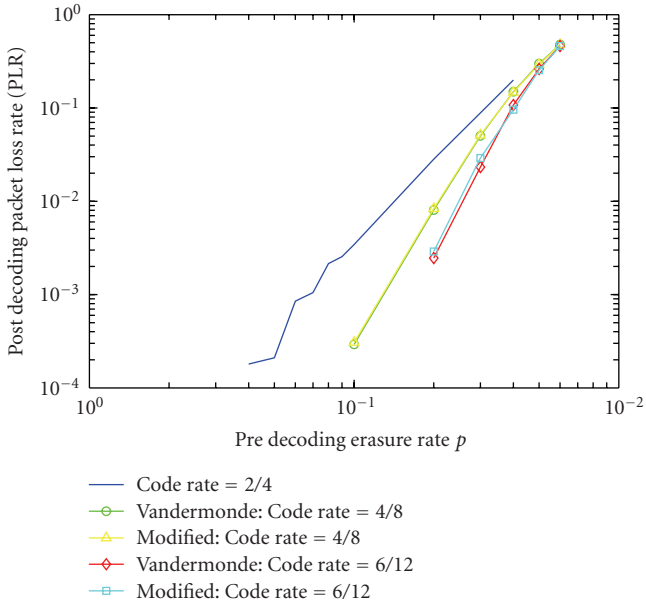

Figure 3: Packet loss rate for systematic codes with the same code rate of 1/2 but different $d_{\min}$.

Table 2: The error locator matrix, **W**, for a single packet in error for the $(4, 2, 3)$ systematic code.

R_0 in error	R_1 in error	R_2 in error	R_3 in error
W	**W**	**W**	**W**
0	$E_1 + xE_1$	E_2	E_3
$xE_0 + E_0$	0	xE_2	E_3
E_0	xE_1	0	E_3
E_0	E_1	E_2	0

Table 3: The error locator matrix, **W**, for a single packet in error for the $(5, 3, 3)$ systematic code.

R_0 in error	R_1 in error	R_2 in error	R_3 in error	R_4 in error
W	**W**	**W**	**W**	**W**
0	$E_1 + xE_1$	$E_2 + x^2E_2$	E_3	E_4
$xE_0 + E_0$	0	$xE_2 + x^2E_2$	xE_3	E_4
$x^2E_0 + E_0$	$x^2E_1 + xE_1$	0	x^2E_3	E_4
E_0	xE_1	x^2E_2	0	E_4
E_0	E_1	E_2	E_3	0

before that the zero in the error locator matrix indicates the location of the packet in error and the error packet can be taken as the last packet in the error locator matrix in Table 2.

The $(5, 3, 3)$ systematic code is capable of correcting one packet in error out of the received five packets. By following the procedure from (26), the error locator matrix **W** for this code is as follows:

$$\begin{aligned}\mathbf{W} &= \mathbf{Q} \cdot \begin{bmatrix} E_0 + E_1 + E_2 + E_3 \\ E_0 + xE_1 + x^2E_2 + E_4 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 1 \\ x & 1 \\ x^2 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} E_0 + E_1 + E_2 + E_3 \\ E_0 + xE_1 + x^2E_2 + E_4 \end{bmatrix}, \end{aligned} \tag{30}$$

$$\mathbf{W} = \begin{bmatrix} E_1 + xE_1 + E_2 + x^2E_2 + E_3 + E_4 \\ xE_0 + E_0 + xE_2 + x^2E_2 + xE_3 + E_4 \\ x^2E_0 + E_0 + x^2E_1 + xE_1 + x^2E_3 + E_4 \\ E_0 + xE_1 + x^2E_2 + E_4 \\ E_0 + E_1 + E_2 + E_3 \end{bmatrix}.$$

The above error locator matrix reduces to the first, second, third, fourth, or fifth column in Table 3 when the packet in error is the first, second, third, fourth, or fifth one, respectively, in a block of 5 received packets. We notice as before that the zero in the error locator matrix indicates the location of the packet in error. Also, the error packet that should be added to correct the erroneous packet is the last row in the error locator matrix. This is true except when the last received packet is in error, in which case any row in the error locator matrix is the error packet.

We discussed two codes $(4, 2, 3)$ and $(5, 3, 3)$ which are both single error correcting code like the $(7, 4, 3)$ Hamming code. However, the rates of these three codes are 35/70, 42/70, and 40/70, respectively. The decoding process for the first code is simple, but the code has a rate of 0.5. The decoding process of the second code is a little bit more involved compared to the first one, but the code has a higher rate of 0.6. Higher rate single error correcting codes can be designed, but the decoding complexity increases slightly as the code rate increases.

We presented an efficient decoding algorithm to correct a single erroneous packet in a family of codes having a minimum distance of three. Therefore, this family is capable of correcting all bits in error within a single erroneous packet, irrespective of the size of the packet. The family has the parameters $(k + 2, k, 3)$, where $k \geq 2$. The rate of this family is $k/(k + 2)$. This designed family has more flexible code parameters when compared to the family of Hamming codes having the parameters $(2^m - 1, 2^m - m - 1, 3)$, where $m \geq 3$. The next Hamming code after the $(7, 4, 3)$ is the $(15, 11, 3)$ which has a code rate of $11/15 = 0.733$. A comparable code performance in our design is when taking $k = 6$ to construct the code $(8, 6, 3)$ having a code rate of $6/8 = 0.75$. However, the latter is less complex since it has a code length of 8, which is almost half of the Hamming code of length 15. As a result, the delay in constructing the encoder and decoder matrices is greatly reduced, especially as the code rate increases.

7.2. Simulation Results. This section shows postdecoding packet error rate performance, PER_{post}, of the proposed codes for a wide range of raw PER in the network.

The presented results are obtained through the simulations of the actual encoding, network packet error and

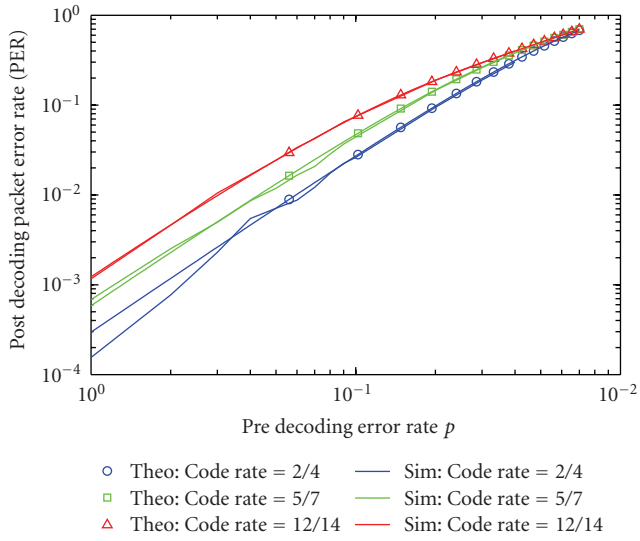

FIGURE 4: Packet error rate for systematic codes with $d_{\min} = 3$.

decoding processes. The PER recovery simulation results are plotted using continuous unmarked lines. For comparison purposes, we also plot the theoretical curves using continuous marked lines based on (17) but taking the summation from $t+1$ with $t = 1$ instead of $e+1$ and p representing here the raw PER.

Figure 4 shows the PER performance for three codes with the same minimum distance of three but different code rates. The three codes have the parameters $(4, 2, 3)$, $(7, 5, 3)$, and $(14, 12, 3)$. The rates of these codes are 2/4, 5/7, and 12/14, respectively. We can observe that the performance improves as the code rate decreases because the codes can correct for one packet in a group of n coded packets where $n = 4, 7,$ and 14. Also, it can be noted that the theoretical PER performances agree with the simulation results.

8. Conclusion

We summarize now the advantages of working with the proposed code design for packet-level FEC in which the elements of the Vandermonde matrix are the shift operator. The code design is applicable to recover from lost packets up to $n - k$ out of the n coded packets, or correct one erroneous packet out of the n received packets. This design is simple to implement since all our arithmetic operations are done in the binary field using only simple shifts and modulo-2 additions.

The only disadvantage is the overhead associated with the need to zero pad each packet with the maximum degree, $g_{\max}$, of the determinants among the set of all determinants of square submatrices of the designed Vandermonde matrix. To reduce the overhead considerably, however, we proposed modified nonVandermonde matrix designs which were found by exhaustive search. We believe that finding such designs in more structured way is still a challenging problem especially as the matrix size increases. To even further reduce this overhead in both designs, we can only zero pad with the maximum shift in the matrix which is much less than $g_{\max}$.

The overhead reduces the efficiency of the design (overall code rate) especially when designing for large code parameters. However, as the packets size increases, the efficiency improves. Therefore, the design is applicable to packets of any size provided that they are not very small. For moderate code parameters, packets of few hundred bits (all network standards requires even more than this) are good enough that will not affect the efficiency of the code very much. For large code parameters, the efficiency can be improved by increasing the packet size and/or by utilizing the mentioned ways of reducing the overhead.

For erasure recovery, we showed how to find the inverse of a matrix using a simple algorithm by exploiting the logarithmic operator of the elements of the Vandermonde matrix and converting the operations to simple modulo-2 additions. For error correction, we presented a syndrome decoding algorithm that corrects for a single erroneous packet using a specialized error locator matrix. The design is suitable for real-time applications and multicasting, where conventional ARQ protocols employing retransmission are inadequate, due to the introduction of delay and jitter. Also, the design can be exploited in cross-layer protocols design to recover from both erasures and errors simultaneously.

Acknowledgment

The authors acknowledge the support of King Fahd University of Petroleum and Minerals (KFUPM).

References

[1] D. J. Costello, J. Hagenauer, H. Imai, and S. B. Wicker, "Applications of error-control coding," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2531–2560, 1998.

[2] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Efficient erasure correcting codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 569–584, 2001.

[3] S. Karande and H. Radha, "Partial Reed Solomon codes for erasure channels," in *Proceedings of the IEEE Information Theory Workshop (ITW '03)*, pp. 82–85, April 2003.

[4] A. Al-Shaikhi, *Innovative designs and deplyments of erasure codes in communication systems*, Ph.D. dissertation, Dalhousie University, Nova Scotia, Canada, 2007.

[5] R. L. Collins and J. S. Plank, "Assessing the performance of erasure codes in the wide-area," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN '05)*, pp. 182–187, Yokohama, Japan, June 2005.

[6] S. Lin and D. J. Costello, *Error Control Coding*, Prentice-Hall, Upper Saddle River, NJ, USA, 2004.

[7] S. S. Karande and H. Radha, "The utility of hybrid error-erasure LDPC (HEEL) codes for wireless multimedia," in *Proceedings of the IEEE International Conference on Communications (ICC '05)*, vol. 2, pp. 1209–1213, Seoul, South Korea, May 2005.

[8] A. Al-Shaikhi, J. Ilow, and X. Liao, "An adaptive FEC-based packet loss recovery scheme using RZ turbo codes," in *Proceedings of the 5th Annual Conference on Communication Networks and Services Research (CNSR '07)*, pp. 263–267, Fredericton, Canada, May 2007.

[9] A. A. Al-Shaikhi and J. Ilow, "Packet loss recovery codes based on Vandermonde matrices and shift operators," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT '08)*, pp. 1058–1062, Toronto, Canada, July 2008.

[10] F. J. Ayres, *Schaum's Outline of Theory and Problems of Matrices*, Schaum, New York, NY, USA, 1962.

[11] A. Ben-Israel and T. N. Greville, *Generalized Inverses: Theory and Applications*, Wiley Interscience, New York, NY, USA, 1977.

[12] D. S. Dummit and R. M. Foote, *Abstract Algebra*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1998.

[13] J. Lacan and J. Fimes, "Systematic MDS erasure codes based on Vandermonde matrices," *IEEE Communications Letters*, vol. 8, no. 9, pp. 570–572, 2004.

[14] J. Fimes, J. Lacan, et al., "Estimation of the number of singular square submatrices of Vandermonde matrices defined over a finite field," Tech. Rep. RE-2003-01, ENSICA, January 2003.

[15] F. MacWilliams and N. Sloane, *The Theory of Error-Correcting Codes*, North Holland, Amsterdam, The Netherlands, 1978.

[16] T. Muir, *Treatise on the Theory of Determinants*, Dover Phoenix Editions, New York, NY, USA, 1960.