8-2004

Packet-loss resilient coding scheme with only XOR operations

Gui Liang FENG
University of Louisiana at Lafayette

Robert H. DENG
Singapore Management University, robertdeng@smu.edu.sg

Feng BAO
Institute for Infocomm Research

DOI: https://doi.org/10.1049/ip-com:20040423

A Novel Packet-Loss Resilient Coding Scheme With Only XOR Operations

Gui-Liang Feng
Center for Advanced Computer Studies
University of Louisiana at Lafayette
Lafayette, LA 70504, USA

Robert H. Deng
Institute for Infocomm Research
21 Heng Mui Keng Terrace
Singapore 119597

Feng Bao
Institute for Infocomm Research
21 Heng Mui Keng Terrace
Singapore 119597

1 Introduction

Error-control codes have been studied for a long time and have been widely used in the lower protocol layers of computer networks to achieve reliable communications [1]. With the advent of various new communications and multimedia technologies and standards, there has been a growing trend in applying error-control coding in the higher layers of communication networks or directly in applications themselves. This trend is especially evident in multicast and multimedia communications. For example, the development of IP multicast and the Internet multicast backbone has led to many scalable audio/video conferencing applications based on the lightweight sessions model [2] which provides efficient multi-way communications and scales from two to several thousand participants. Multicast and multimedia applications are normally very sensitive to transmission errors and error-control coding can play an important part in improving system performance [3, 4].

The nature of error control in the higher layers are fundamentally different from that in the lower layers of communication networks. In the lower layers, transmission channels are described in terms of bit error rates which are mainly caused by low signal-to-noise ratios. Therefore, in the lower layers, error control codes have been long employed to deal with bit errors within a packet and encoders and decoders are implemented in hardware. In the higher layers, however, a communication connection is no longer link-by-link but end-to-end (i. e., concatenation of multiple links) in nature. Such connections are characterized by packet loss due to buffer overflows at intermediate nodes and uncorrectable bit errors at the link layer. By monitoring a large multicast session on MBone over several days, it was reported in [5] that most receivers experience packet loss in the range of 2-5 percent where the overwhelming cause of packet loss was due to congestion at the routers. Consequently, error-control codes in the higher layers or in applications must be adopted to recover lost packets and encoders and decoders are implemented in software.

Both ARQ (Automatic Repeat reQuest) and FEC (Forward Error Correction) can be used to recover missing or erroneous packets [1]. ARQ is based on retransmission of lost packets, which is trigged by explicit or implicit ACK/NACKs from the receivers. ARQ is simple to implement since no complicated processing of received packets need to be carried out at the receivers. However, it requires a feedback channel and introduces nonuniform delays because of the packet retransmissions. This has been a serious drawback for time sensitive applications. Another shortcoming of ARQ is that it scales badly in multicast protocols as the number of receivers becomes large, because random packet losses at different receivers might require retransmission of the majority of packets [6, 7].

Different from ARQ, FEC anticipates the amount of packet losses, and obviates by sending redundant data which allows the receiver to reconstruct up to a certain number of missing packets. The communication process thus includes an encoding phase at the sender,where redundant or parity check data are constructed from the source information data, and a decoding phase at the receivers, where lost source data are recovered. Since no retransmission is needed in FEC, it is suitable for real-time communications. For multicast communications, independent losses do not combine so badly anymore, because the same redundant data allows different receivers to reconstruct distinct missing packets. Therefore, FEC-based multicast protocols scale much better than ARQ-based protocols. FEC has been used for bit error correction in data link layers for a long time but its use for packet loss recovery in higher layers is relatively new. FEC is used for ATM cell recovery in [8, 9], for packet reconstruction in transport protocols in [10], for realizing reliable multicast data distribution protocol in

[11, 12] and for packet loss recovery in video streaming [13, 14].

Most of the above proposals for packet loss recovery with FEC use symbol-oriented Reed-Solomon codes operating in symbol erasure-correction mode. An (n, k) Reed-Solomon code encoder over $GF(2^m)$ accepts an information sequence of k symbols and outputs a n symbol codeword, where a symbol is an element in $GF(2^m)$ and the size of a symbol is typically 8 bits. The two most important properties of erasure-correction codes are the amount of redundance needed to recover the information sequence and the decoding speed. Reed-Solomon code is optimal in the sense that it is maximal distance separable (MDS). An (n, k) erasure-correction code is MDS if it can recover the k symbol information sequence given a received codeword with $n - k$ missing symbols. Since decoding of Reed-Solomon code involves operations over $GF(2^m)$ using look-up table [15], it is slow, especially when m is large since the size of the look-up table increases exponentially as m increases.

In [16], Blomer, et. al. proposed an XOR-based erasure coding scheme. The idea is to transform the parity-check matrix of a version of Reed-Solomon code to binary form by combining *Cauchy* matrices with the matrix expression of finite field elements over $GF(2^m)$. This allows them to replace arithmetic operations on field elements by XORs of computer words, and in practice XORs of computer words are much more efficient than multiplications of finite fields. However, in the implementation of the code, they avoid polynomial arithmetic by transforming multiplications and divisions into additions and substraction of exponents by using a look-up table of discrete logarithms. Thus, in this scheme, only calculation of syndromes uses XOR operations while recovery of lost packets still cannot avoid field operations by using look-up tables.

More recently, Luby, et. al. developed a probabilistic packet-loss resilient code called *Tornado* code [17, 18]. Tornado code is attractive since it only requires XOR operation and its decoding time is $O(E)$, where E is the number of packets to be recovered. However, Tornado code has a number of shortcomings. First, the code is not MDS since the number of redundant packets is larger than the number of packets which can be recovered. The code is efficient asymptotically, i. e., when many packets are lost and the code length is very large. Second, it recovers lost packets with a certain probability, albeit close to 1. This means that even if only one packet is lost, it has a probability of not being recovered. Finally, the design of the code is very difficult. There is no explicit form for parity-check matrix or code generating matrix.

In this paper, we propose an algebraic packet-loss resilient coding scheme with only XOR operations. This scheme maps the parity check matrix of an (n, k) Reed-Solomon code over $GF(2^m)$ into a binary parity-check matrix of our $(n, k)/(m, l)$ packet-loss resilient code. This code encodes k information packets into a codeword of $n = 2^m - 1$ packets, where each packet consists of m l-bit tuples with l being an arbitrary positive integer. In practice l is a multiple of the size of the computer words. This code achieves the capacity of the erasure channel [19] and is MDS, i. e., if the number of lost packets in a received codeword is not more than $n - k$, then the k information packets can be recovered. The code is very efficient in encoding and decoding operations since they only require XOR operations.

In the next section, we first describe the binary vector and matrix expressions of finite field elements and review their properties. We then define a so called "bit-vector multiplication" and based on this definition, we define the new $(n, k)/(m, l)$ packet-loss resilient code. In section 3, we describe the general encoding and decoding procedures for the new code. To speed up the encoding and decoding procedures, we present a recursive algorithm for solving a system of linear equations with the Vandermonde matrices in Section 4 and a fast algorithm for calculating syndromes in Section 5. Finally, In Section 6 we present our computer simulation results.

2 Definition of the New Packet-Loss Resilient Code

In this section, we briefly review binary vector and matrix expressions for elements in $GF(2^m)$ followed by the definition of the new packet-loss resilient code. Throughout the paper, we will use lower case letters, e. g., a, b, to denote scalar variables and capital letters, e. g., A, B, to denote vector and matrix variables.

2.1 Binary Vector and Matrix Expressions for Finite Field Elements

Let α be a primitive element in $GF(2^m)$. We have

$$GF(2^m) = \{0, 1, \alpha, \alpha^2, \cdots, \alpha^{2^m-2}\}$$

and an element $\alpha^i \in GF(2^m)$, $i = 0, 1, \cdots, 2^m - 2$, can be written uniquely as

$$\alpha^i = (i)_0 + (i)_1\alpha + \cdots + (i)_{m-1}\alpha^{m-1}. \qquad (2.1)$$

The vector

$$V_{\alpha^i} = ((i)_0\ (i)_1\ \cdots\ (i)_{m-1}) \qquad (2.2)$$

and the matrix

$$M_{\alpha^i} = [V_{\alpha^i}{}^T \; V_{\alpha^{i+1}}{}^T \; \cdots \; V_{\alpha^{i+m-1}}{}^T]_{m\times m}, \qquad (2.3)$$

are called the vector expression and the matrix expression, respectively, of α^i under the basis $\{1, \alpha, \cdots, \alpha^{m-1}\}$.

Theorem 1 *Let α be a primitive element in $GF(2^m)$. Then the set of binary matrices $\{M_0, M_1, M_\alpha, M_{\alpha^2}, \cdots, M_{\alpha^{2^m-2}}\}$ with multiplication and addition over $GF(2)$ forms a finite field isomorphic to $GF(2^m)$.*

Next, we introduce the concept of power matrix which is very useful for calculating conjugate syndromes in the decoding process. Given α^i as in (2.1), we have

$$\begin{aligned} \alpha^{2i} &= (i)_0 + (i)_1\alpha^2 + \cdots + (i)_{m-1}\alpha^{2(m-1)} \\ &= (2i)_0 + (2i)_1\alpha + \cdots + (2i)_{m-1}\alpha^{m-1} \end{aligned}$$

Thus, it follows that

$$\begin{bmatrix} (2i)_0 \\ (2i)_1 \\ \vdots \\ (2i)_{m-1} \end{bmatrix} = M \begin{bmatrix} (i)_0 \\ (i)_1 \\ \vdots \\ (i)_{m-1} \end{bmatrix}, \qquad (2.4)$$

where

$$M = \begin{bmatrix} V_{\alpha^0}{}^T \; V_{\alpha^2}{}^T \; \cdots \; V_{\alpha^{2(m-1)}}{}^T \end{bmatrix}_{m\times m}. \qquad (2.5)$$

We remark that the vector and matrix expressions of finite field elements can also be defined under normal basis. The concept of power matrix under normal basis leads to significant reduction in computations of syndrome valuables. However, to keep the paper compact, we will not further discuss it here.

2.2 The New Packet-Loss Resilient Code

We start with a *"bit-vector multiplication"* definition which will be used extensively in the encoding and decoding of our new packet-loss resilient code. Let v be a binary variable in $GF(2)$ and A be a l-dimensional binary vector in $GF(2)^l$. The bit-vector multiplication is defined by

$$v \times A = v \cdot A = \begin{cases} 0 & v = 0 \\ A & v = 1 \end{cases}. \qquad (2.6)$$

Note that $v \times A \in GF(2)^l$. Let $V = (v_0 \; v_1 \; \cdots \; v_{n-1})$ and $B = (B_0 \; B_1 \; \cdots \; B_{n-1})$, where $v_i \in GF(2)$ and $B_i \in GF(2)^l$. We further define

$$V \cdot B^T = v_0 \cdot B_0 + v_1 \cdot B_1 + \cdots + v_{n-1} \cdot B_{n-1}. \qquad (2.7)$$

where "$\cdot$" is defined in (2.6) and "$+$" is the XOR operation. Obviously, $V \cdot B^T \in GF(2)^l$. The definitions (2.6) and (2.7) can be generalized to the case where the vectors A and B are elements from any Abelian group.

Let α be a primitive element in $GF(2^m)$. The party check matrix of an (n, k) Reed-solomon code over $GF(2^m)$ is given by $H_{RS} =$

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^{(n-2)} \\ 1 & \alpha^2 & \alpha^4 & \cdots & \alpha^{2(n-2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{r-1} & \alpha^{2(r-1)} & \cdots & \alpha^{(n-2)(r-1)} \end{bmatrix} \qquad (2.8)$$

where $n = 2^m - 1$ is the length of the Reed-Solomon codewords, $r = n - k$ is the number of party check components in a codewords and k is the number of information components which can be encoded into an n-component codeword. Note that a component in the Reed-Solomon code is an element over $GF(2^m)$. This code has minimum Hamming distance $r + 1$ and therefore is able to recover r or fewer missing code components in a received codeword [1, 15].

From (2.2) and (2.3) we know that an element $\alpha^i \in GF(2^m)$ can be mapped onto a $m \times m$ binary matrix M^i_α. Replacing the elements in H_{RS} with their corresponding $m\times m$ matrices, we obtain the following $rm \times nm$ binary matrix:

$$H = \begin{bmatrix} M_1 & M_1 & \cdots & M_1 \\ M_1 & M_\alpha & \cdots & M_{\alpha^{n-1}} \\ M_1 & M_{\alpha^2} & \cdots & M_{\alpha^{2(n-1)}} \\ \vdots & \vdots & \vdots & \vdots \\ M_1 & M_{\alpha^{(r-1)}} & \cdots & M_{\alpha^{(r-1)(n-1)}} \end{bmatrix}. \qquad (2.9)$$

Let

$$I \triangleq (I[0] \; I[1] \; \cdots \; I[n-1]) \qquad (2.10.1)$$

be a sequence of n packets, with the ith packet given by

$$I[i] \triangleq \begin{bmatrix} I[i][0] \\ I[i][1] \\ \vdots \\ I[i][m-1] \end{bmatrix} \qquad (2.10.2)$$

where $I[i][j]$ is a l-bit binary vector over $GF(2)^l$. Our $(n, k)/(m, l)$ packet-loss resilient code is defined as

$$C = \{I \| HI^T = 0^T\}, \qquad (2.11)$$

where the product HI^T is carried out in terms of the "bit-vector multiplication" defined in (2.6) and (2.7),

I is a codeword of length $n = 2^m - 1$ and H is the parity check matrix. Note that, a component in a Reed- Solomon code is an element in $GF(2^m)$ while a code component in our packet-loss resilient code is a packet (see (2.10.2)) which $I[i]$ consists of m l-bit binary vectors over $GF(2)^l$. For efficiency reasons, l should be a multiple of the length of a computer word (e. g., $l = 32$ bits in Pentium processors).

From Theorem 2.1 and the fact that the (n, k) Reed-Solomon code defined by (2.8) has a minimum Hamming distance $r + 1$, it is easy to show that our $(m, k)/(m, l)$ code also has a minimum Hamming distance $r+1$. Therefore, it is able to recover any t missing packets $I[i_1], I[i_2], \cdots, I[i_t]$ in a codeword provided that $t \leq r$.

3 Encoding and Decoding of the Packet-Loss Resilient Code

In this section, we describe the general encoding and decoding operations of the $(n, k)/(m, l)$ packet-loss resilient code C as defined in (2.11). Methods for speeding up encoding and decoding operations when implemented in software will be presented in Sections 4 and 5. To make our description simpler and without loss of generality, we assume that l equal the bit length of a computer word. Therefore, we will speak that a codeword I consists of n packets and a packet in turn comprises m l-bit computer words.

3.1 Encoding Procedure

Given k information packets $I[i]$, $i = r, r+1, \cdots, n-1$, the encoding procedure determines the parity-check packets $I[i]$, $i = 0, 1, \cdots, r-1$, such that (2.11) is satisfied. To this purpose, we first compute

$$B[\mu] = \begin{bmatrix} B[\mu][0] \\ B[\mu][1] \\ \vdots \\ B[\mu][m-1] \end{bmatrix} \triangleq \sum_{i=r}^{n-1} M_{\alpha^{\mu i}} I[i], \qquad (3.1)$$

for $\mu = 0, 1, \cdots, r-1$, and then calculate the parity-check packets $I[0], I[1], \cdots, I[r-1]$ from

$$\begin{bmatrix} I[0] \\ I[1] \\ \vdots \\ I[m-1] \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & M_\alpha & \cdots & M_{\alpha^{r-1}} \\ \vdots & \vdots & & \vdots \\ 1 & M_{\alpha^{r-1}} & \cdots & M_{\alpha^{(r-1)(r-1)}} \end{bmatrix}^{-1} \begin{bmatrix} B[0] \\ B[1] \\ \vdots \\ B[r-1] \end{bmatrix}. \qquad (3.2)$$

It is straightforward to verify that the codeword $I = (I[0]\ I[1]\ \cdots\ I[r-1]\ \cdots\ I[n-1])$ satisfies our code definition equation (2.11).

3.2 Decoding Procedure

Assume that a codeword $I = (I[0]\ I[1])\ \cdots\ I[n-1])$ is transmitted and that t packets, say $I[\mu_1], I[\mu_2], \cdots, I[\mu_t]$ are lost. Then the received codeword is given by

$$J = (J[0]\ J[1]\ \cdots\ J[n-1]), \qquad (3.3.1)$$

where

$$J[i] = \begin{cases} I[i] & i \notin \{\mu_1, \mu_2, \cdots, \mu_t\} \\ 0 & i \in \{\mu_1, \mu_2, \cdots, \mu_t\} \end{cases} \qquad (3.3.2)$$

We define the syndromes of the received codeword J as

$$\begin{bmatrix} S[0] \\ S[1] \\ \vdots \\ S[r-1] \end{bmatrix} \triangleq H \cdot J^T. \qquad (3.4)$$

Let

$$L = (L[0], L[1], \cdots, L[n-1]),$$

and

$$L[i] = \begin{cases} 0 & i \notin \{\mu_1, \mu_2, \cdots, \mu_t\} \\ I[i] & i \in \{\mu_1, \mu_2, \cdots, \mu_t\} \end{cases}$$

It follows that

$$I = J + L.$$

From (2.11), (3.4) and $I = J + L$, we have

$$H \cdot L^T = \begin{bmatrix} S[0] \\ S[1] \\ \vdots \\ S[r-1] \end{bmatrix},$$

or equivalently

$$\begin{bmatrix} M_1 & M_1 & \cdots & M_1 \\ M_{\alpha^{\mu_1}} & M_{\alpha^{\mu_2}} & \cdots & M_{\alpha^{\mu_t}} \\ \vdots & \vdots & & \vdots \\ M_{\alpha^{(t-1)\mu_1}} & M_{\alpha^{(t-1)\mu_2}} & \cdots & M_{\alpha^{(t-1)\mu_t}} \end{bmatrix} \begin{bmatrix} I[\mu_1] \\ I[\mu_2] \\ \vdots \\ I[\mu_t] \end{bmatrix}$$

$$= \begin{bmatrix} S[0] \\ S[1] \\ \vdots \\ S[t-1] \end{bmatrix}. \qquad (3.5)$$

Note that the $t \times t$ matrix in (3.5) is a Vandermonde matrix.

The decoding process can be briefly summarized as follows. Given a received codeword J and the locations of lost packets $\mu_1, \mu_2, \cdots, \mu_t$, we first compute syndromes from (3.4) and then determine the values of the lost packets $I[\mu_1], I[\mu_2], \cdots, I[\mu_t]$ by solving the set of t linear equations in (3.5). In Section 4, we show an efficient recursive algorithm for solving the system of linear equations with Vandermonde matrix and in Section 5 we present a high-speed approach to calculating syndromes.

4 Recursive Algorithm for Solving Linear Equations with Vandermonde Matrix

Assuming that the syndromes $S[0], S[1], \cdots, S[t-1]$ have been computed according to (3.4), we now derive a recursive algorithm for solving the linear equations with Vandermonde matrix (3.5) in order to recover the missing packets $I[\mu_1], I[\mu_2], \cdots, I[\mu_t]$. We begin our derivation with the following simple example.

Example 1
Consider

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ x_1^2 & x_2^2 & x_3^2 & x_4^2 \\ x_1^3 & x_2^3 & x_3^3 & x_4^3 \end{bmatrix} \begin{bmatrix} I_0 \\ I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} S_0^{(0)} \\ S_1^{(0)} \\ S_2^{(0)} \\ S_3^{(0)} \end{bmatrix},$$

or equivalently

$$\begin{bmatrix} I_0 & + & I_1 & + & I_2 & + & I_3 \\ x_1 I_0 & + & x_2 I_1 & + & x_3 I_2 & + & x_4 I_3 \\ x_1^2 I_0 & + & x_2^2 I_1 & + & x_3^2 I_2 & + & x_4^2 I_3 \\ x_1^3 I_0 & + & x_2^3 I_1 & + & x_3^3 I_2 & + & x_4^3 I_3 \end{bmatrix} = \begin{bmatrix} S_0^{(0)} \\ S_1^{(0)} \\ S_2^{(0)} \\ S_3^{(0)} \end{bmatrix}.$$

We solve the above set of 4 linear equations for the unknowns I_0, I_1, I_2 and I_3 through 3 forward steps and 3 backward steps.

First, consider the forward steps. Left multiplying both sides of the above equation by the matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ x_1 & 1 & 0 & 0 \\ 0 & x_1 & 1 & 0 \\ 0 & 0 & x_1 & 1 \end{bmatrix},$$

we have

$$\begin{bmatrix} I_0 + I_1 + I_2 + I_3 \\ (x_2 + x_1)I_1 + (x_3 + x_1)I_2 + (x_4 + x_1)I_3 \\ x_2(x_2 + x_1)I_1 + x_3(x_3 + x_1)I_2 + x_4(x_4 + x_1)I_3 \\ x_2^2(x_2 + x_1)I_1 + x_3^2(x_3 + x_1)I_2 + x_4^2(x_4 + x_1)I_3 \end{bmatrix}$$

$$= \begin{bmatrix} S_0^{(0)} \\ S_1^{(0)} + x_1 S_0^{(0)} \\ S_2^{(0)} + x_1 S_1^{(0)} \\ S_3^{(0)} + x_1 S_2^{(0)} \end{bmatrix} \triangleq \begin{bmatrix} S_0^{(0)} \\ S_1^{(1)} \\ S_2^{(1)} \\ S_3^{(1)} \end{bmatrix}. \quad (4.1)$$

Left multiplying both sides of (4.1) by

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & x_2 & 1 & 0 \\ 0 & 0 & x_2 & 1 \end{bmatrix},$$

and then by

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & x_3 & 1 \end{bmatrix},$$

We obtain

$$\begin{bmatrix} I_0 + I_1 + I_2 + I_3 \\ (x_2 + x_1)I_1 + (x_3 + x_1)I_2 + (x_4 + x_1)I_3 \\ (x_3 + x_2)(x_3 + x_1)I_2 + (x_4 + x_2)(x_4 + x_1)I_3 \\ x_3(x_3 + x_2)(x_3 + x_1)I_2 + x_4(x_4 + x_2)(x_4 + x_1)I_3 \end{bmatrix}$$

$$= \begin{bmatrix} S_0^{(0)} \\ S_1^{(1)} \\ S_2^{(1)} + x_2 S_1^{(1)} \\ S_3^{(1)} + x_2 S_2^{(1)} \end{bmatrix} \triangleq \begin{bmatrix} S_0^{(0)} \\ S_1^{(1)} \\ S_2^{(2)} \\ S_3^{(2)} \end{bmatrix} \quad (4.2)$$

and

$$\begin{bmatrix} I_0 + I_1 + I_2 + I_3 \\ (x_2 + x_1)I_1 + (x_3 + x_1)I_2 + x_4 + x_1)I_3 \\ (x_3 + x_2)(x_3 + x_1)I_2 + (x_4 + x_2)(x_4 + x_1)I_3 \\ (x_4 + x_3)(x_4 + x_2)(x_4 + x_1)I_3 \end{bmatrix}$$

$$= \begin{bmatrix} S_0^{(0)} \\ S_1^{(1)} \\ S_2^{(2)} \\ S_3^{(2)} + x_3 S_2^{(2)} \end{bmatrix} \triangleq \begin{bmatrix} S_0^{(0)} \\ S_1^{(1)} \\ S_2^{(2)} \\ S_3^{(3)} \end{bmatrix}, \quad (4.3)$$

respectively. Now we have rearranged the original set of equations into a set of triangular equations in the unknowns. Observing the right hand sides of (4.1)-(4.3), where we carried out (3+2+1)=6 $(p + ap')$-type operations, and the $(p + ap')$-type operation contains one multiplication and one addition.

Next, we perform the 3 backward steps. Left multiplying both sides of (4.3) by

$$\begin{bmatrix} 1 & 0 & 0 & (x_4 + x_3)^{-1}(x_4 + x_2)^{-1}(x_4 + x_1)^{-1} \\ 0 & 1 & 0 & (x_4 + x_3)^{-1}(x_4 + x_2)^{-1} \\ 0 & 0 & 1 & (x_4 + x_3)^{-1} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

we have

$$\begin{bmatrix} I_0+I_1+I_2 \\ (x_2+x_1)I_1+(x_3+x_1)I_2 \\ (x_3+x_2)(x_3+x_1)I_2 \\ (x_4+x_3)(x_4+x_2)(x_4+x_1)I_3 \end{bmatrix} =$$

$$\begin{bmatrix} S_0^{(0)}+(x_4+x_3)^{-1}(x_4+x_2)^{-1}(x_4+x_1)^{-1}S_3^{(3)} \\ S_1^{(1)}+(x_4+x_3)^{-1}(x_4+x_2)^{-1}S_3^{(3)} \\ S_2^{(2)}+(x_4+x_3)^{-1}S_3^{(3)} \\ S_3^{(3)} \end{bmatrix}$$

$$\triangleq \begin{bmatrix} S_0^{(4)} \\ S_1^{(4)} \\ S_2^{(4)} \\ S_3^{(3)} \end{bmatrix}, \tag{4.4}$$

which can be rewritten as

$$\begin{bmatrix} I_0+I_1+I_2 \\ (x_2+x_1)I_1+(x_3+x_1)I_2 \\ (x_3+x_2)(x_3+x_1)I_2 \\ I_3 \end{bmatrix}$$

$$= \begin{bmatrix} S_0^{(4)} \\ S_1^{(4)} \\ S_2^{(4)} \\ S_0^{(4)}+S_0^{(0)} \end{bmatrix} \triangleq \begin{bmatrix} S_0^{(4)} \\ S_1^{(4)} \\ S_2^{(4)} \\ S_3^{(4)} \end{bmatrix}. \tag{4.5}$$

Following the same approach, we have

$$\begin{bmatrix} I_0+I_1 \\ (x_2+x_1)I_1 \\ I_2 \\ I_3 \end{bmatrix} =$$

$$\begin{bmatrix} S_0^{(4)}+(x_3+x_2)^{-1}(x_3+x_1)^{-1}S_2^{(4)} \\ S_1^{(4)}+(x_3+x_2)^{-1}S_2^{(4)} \\ S_0^{(4)}+S_0^{(5)} \\ S_3^{(4)} \end{bmatrix}$$

$$\triangleq \begin{bmatrix} S_0^{(5)} \\ S_1^{(5)} \\ S_2^{(5)} \\ S_3^{(4)} \end{bmatrix}, \tag{4.6}$$

and finally

$$\begin{bmatrix} I_0 \\ I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} S_0^{(5)}+(x_2+x_1)^{-1}S_1^{(5)} \\ S_0^{(5)}+S_0^{(6)} \\ S_2^{(5)} \\ S_3^{(4)} \end{bmatrix} \triangleq \begin{bmatrix} S_0^{(6)} \\ S_1^{(6)} \\ S_2^{(5)} \\ S_3^{(4)} \end{bmatrix}. \tag{4.7}$$

From the right hand sides of (4.4)-(4.7), we see that we needed $(3+2+1)=6$ $(p+ap^{'})$-type operations and 3 additions to get solutions $I_0=S_0^{(6)}, I_1=S_1^{(6)}, I_2=S_2^{(5)}$ and $I_3=S_3^{(4)}$.

□

In general, to solve

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_t \\ \vdots & \vdots & & \vdots \\ x_1^{t-1} & x_2^{t-1} & \cdots & x_t^{t-1} \end{bmatrix} \begin{bmatrix} I_0 \\ I_1 \\ \vdots \\ I_{t-1} \end{bmatrix} = \begin{bmatrix} S_0^{(0)} \\ S_1^{(0)} \\ \vdots \\ S_{t-1}^{(0)} \end{bmatrix}, \tag{4.8}$$

for the solutions $I_0, I_1, \cdots, I_{t-1}$, we perform $t-1$ forward steps followed by $t-1$ backward steps. In the forward steps, we calculate

$$S_i^{(j)} \triangleq S_i^{(j-1)} + x_j S_{i-1}^{(j-1)},$$

$$\text{for} \quad j=1,2,\cdots,t \quad \text{and} \quad j \le i \le t-1 \tag{4.9}$$

to obtain

$$\begin{bmatrix} S_0^{(0)} & & & \\ S_1^{(0)} & S_1^{(1)} & & \\ \vdots & \vdots & \ddots & \\ S_{t-1}^{(0)} & S_{t-1}^{(1)} & \cdots & S_{t-1}^{(t-1)} \end{bmatrix} \tag{4.10}$$

with $\frac{t(t-1)}{2}$ $(p+ap^{'})$-type operations. In the $t-1$ backward steps, we calculate

$$\begin{cases} S_j^{(t+i)} = S_j^{(t+i-1)} + \theta_{i,j} S_{t-i-1}^{(t+i-1)} \\ S_{t-i-1}^{(t+i)} = S_0^{(t+i)} + S_0^{(t+i-1)} \end{cases} \quad \text{for}$$

$$i=0,1,2,\cdots,t-2, j=0,1,2,\cdots,t-i-2, \tag{4.11}$$

where $\theta_{i,j}=(x_{t-i}+x_{t-i-1})^{-1}\cdots(x_{t-i}+x_{j+1})^{-1}$, to get

$$\begin{bmatrix} S_0^{(t-1)} & S_0^{(t)} & S_0^{(t+1)} & \cdots & S_0^{(2t-2)} \\ S_1^{(t-1)} & S_1^{(t)} & S_1^{(t+1)} & \cdots & S_1^{(2t-2)} \\ \vdots & \vdots & \vdots & & \\ S_{t-2}^{(t-1)} & S_{t-2}^{(t)} & S_{t-2}^{(t+1)} & & \\ S_{t-1}^{(t-1)} & S_{t-1}^{(t)} & & & \end{bmatrix} \tag{4.12}$$

with $\frac{t(t-1)}{2}$ multiplication and addition operations. The solution we seek is given by

$$\begin{cases} I_i = S_i^{(2t-i-1)} \text{ for } i=1,2,\cdots,t-1. \\ I_0 = S_0^{(2t-2)} \end{cases} \tag{4.13}$$

Note the exact correspondence between (3.5) and (4.8). Therefore, (4.9)-(4.13) can be used for solving (3.5) to obtain values of lost packets $I[\mu_1], I[\mu_2], \cdots, I[\mu_t]$.

5 Fast Algorithm for Calculating the Syndromes

Most of the computational load in the decoding of our code is due to the syndrome calculation. In this section, we introduce an high speed algorithm for computing the syndromes.

5.1 Definitions and Notations

Assuming that there are t missing packets in a received codeword J as given in (3.3.1) and (3.3.2). From (3.4), the syndrome value $S[\mu]$ corresponding to α^μ, is given by

$$S[\mu] \triangleq \begin{bmatrix} S[\mu][0] \\ S[\mu][1] \\ \vdots \\ S[\mu][m-1] \end{bmatrix} = \sum_{i=0}^{n-1} M_{\alpha^{\mu i}} J[i], \qquad (5.1)$$

for $\mu = 0, 1, \cdots, t-1$. Note the similarity of (5.1) and (3.1). Therefore, our fast algorithm for computing syndromes also improves the speed of encoding operation.

Our experiments have shown that at least in the Pentium family of processors, the speed of transfer (i.e., value assignment) operation is roughly the same as that of the XOR operation of two computer words. Hence, we will use the numbers of transfer operations and XOR operations to measure the efficiency of our algorithm. One of the advantages of the algorithm is that once $S[\mu]$ is calculated, $S[2^\xi \mu]$ $(\xi \geq 1)$ can be obtained easily.

Using the vector expression (2.1) of an element $\alpha^i \in GF(2^m)$ and Theorem 2.1, we have

$$M_{\alpha^i} = (i)_0 M_1 + (i)_1 M_\alpha + \cdots + (i)_{m-1} M_{\alpha^{m-1}} = \sum_{\lambda=0}^{m-1} (i)_\lambda M_{\alpha^\lambda}. \qquad (5.2)$$

Substituting (5.2) into (5.1) we obtain

$$S[\mu] = \begin{bmatrix} S[\mu][0] \\ S[\mu][1] \\ \vdots \\ S[\mu][m-1] \end{bmatrix} = \sum_{i=0}^{n-1} M_{\alpha^{\mu i}} J[i] =$$

$$\sum_{i=0}^{n-1} \sum_{\lambda=0}^{m-1} M_{\alpha^\lambda} (\mu i)_\lambda J[i] = \sum_{\lambda=0}^{m-1} M_{\alpha^\lambda} \sum_{i=0}^{n-1} (\mu i)_\lambda J[i]$$

$$= \sum_{\lambda=0}^{m-1} M_{\alpha^\lambda} \begin{bmatrix} \sum_{i=0}^{n-1} (\mu i)_\lambda J[i][0] \\ \sum_{i=0}^{n-1} (\mu i)_\lambda J[i][1] \\ \vdots \\ \sum_{i=0}^{n-1} (\mu i)_\lambda J[i][m-1] \end{bmatrix}$$

$$\triangleq \sum_{\lambda=0}^{m-1} M_{\alpha^\lambda} \begin{bmatrix} Q_\mu[\lambda][0] \\ Q_\mu[\lambda][1] \\ \vdots \\ Q_\mu[\lambda][m-1] \end{bmatrix}$$

$$\triangleq \sum_{\lambda=0}^{m-1} M_{\alpha^\lambda} Q_\mu[\lambda], \qquad (5.3.1)$$

where

$$Q_\mu[\lambda] = \begin{bmatrix} Q_\mu[\lambda][0] \\ Q_\mu[\lambda][1] \\ \vdots \\ Q_\mu[\lambda][m-1] \end{bmatrix}, \lambda = 0, 1, \cdots, m-1. \qquad (5.3.2)$$

and

$$Q_\mu[\lambda][j] = \sum_{i=0}^{n-1} (\mu i)_\lambda J[i][j], \lambda, j = 0, 1, \cdots, m-1. \qquad (5.3.3)$$

Thus, to compute $S[\mu]$ we first calculate $Q_\mu[\lambda][j]$ from (5.3.3) and then $S[\mu]$ from (5.3.1).

5.2 Calculate $Q_\mu[\lambda][j]$

Eqn. (5.3.3) can be rewritten as

$$\begin{bmatrix} Q_\mu[0][j] \\ Q_\mu[1][j] \\ \vdots \\ Q_\mu[m-1][j] \end{bmatrix} = M \begin{bmatrix} J[0][j] \\ J[1][j] \\ \vdots \\ J[n-1][j] \end{bmatrix},$$

$$j = 0, 1, \cdots, m-1; \mu = 0, 1, \cdots, t-1, \qquad (5.4.1)$$

where

$$M = \begin{bmatrix} (0)_0 & (\mu)_0 & \cdots & (\mu(n-1))_0 \\ (0)_1 & (\mu)_1 & \cdots & (\mu(n-1))_1 \\ \vdots & \vdots & \cdots & \vdots \\ (0)_{m-1} & (\mu)_{m-1} & \cdots & (\mu(n-1))_{m-1} \end{bmatrix}_{m \times n}. \qquad (5.4.2)$$

To simplify our notations in this subsection, we denote $Q_\mu[i][j] \triangleq Q_i$ and $J[k][j] \triangleq J_k$. Then (5.4.1) can be written as

$$\begin{bmatrix} Q_0 \\ Q_1 \\ \vdots \\ Q_{m-1} \end{bmatrix} = M \begin{bmatrix} J_0 \\ J_1 \\ \vdots \\ J_{n-1} \end{bmatrix},$$

$$j = 0, 1, \cdots, m-1; \mu = 0, 1, \cdots, t-1. \quad (5.5)$$

Note that if μ is not a factor of $2^m - 1$, all the columns in M are distinct from each other and none of them are the all-zero column. Since each row of M has exactly 2^{m-1} 1's, direct computation of $Q_\mu[i][j]$, for $i = 1, 2, \cdots, m$, from (5.5)(or equivalently from (5.4.1)) requires $(2^{m-1} - 1)m$ XORs and m transfer operations. The total complexity is $O(m2^m)$. Instead of carrying out direct multiplication in (5.5), we divide M into a $\lceil \frac{m}{2} \rceil \times n$ binary sub-matrix M_u and a $\lfloor \frac{m}{2} \rfloor \times n$ binary sub-matrix M_d. First, let's focus on M_u. Let $(v_0 v_1, \cdots v_{\lceil \frac{m}{2} \rceil - 1})^T$ be a column vector of M_u. Its weighted value is defined as

$$v = v_0 + v_1 \times 2 + \cdots + v_{\lceil \frac{m}{2} \rceil - 1} \times 2^{\lceil \frac{m}{2} \rceil - 1}.$$

We perform a permutation σ on the n columns of M_u to rearrange them into $2^{\lceil \frac{m}{2} \rceil}$ groups, $X_0, X_1, X_2, \cdots, X_{2^{\lceil \frac{m}{2} \rceil} - 1}$, where a group is made up of identical columns (i.e., columns with the same weighted value). We also perform the same permutation on the n vectors $J_0, J_1, \cdots, J_{n-1}$, to form $2^{\lceil \frac{m}{2} \rceil}$ groups, $Y_0, Y_1, Y_2, \cdots, Y_{2^{\lceil \frac{m}{2} \rceil} - 1}$. Note the one-to-one correspondence between X_v and Y_v.

We then calculate

$$P_v = \sum \{J_i | i \in Y_v\}, \ v = 1, 2, \cdots, 2^{\lceil \frac{m}{2} \rceil} - 1 \quad (5.6)$$

and

$$Q_j = \sum \{P_v | \text{all } \ v = v_0 + v_1 \times 2 + \cdots$$

$$+ v_{\lceil \frac{m}{2} \rceil - 1} \times 2^{\lceil \frac{m}{2} \rceil - 1}, v_j \neq 0\}, \quad (5.7)$$

$j = 0, 1, \cdots, \lceil \frac{m}{2} \rceil - 1$. By the same approach we can compute $Q_{\lceil \frac{m}{2} \rceil}, Q_{\lceil \frac{m}{2} \rceil + 1}, \cdots, Q_{m-1}$ from M_d.

The above process of dividing a matrix into a upper submatrix and a lower submatrix can be done recursively. That is, M_u and M_d can each be further divided into two submatrices and so on. Let $F(m)$ and $G(m)$ be the number of XORs and transfer operations respectively in calculating (5.6) and (5.7), then we have the following recursive relations:

$$F(m) = 2(2^{\lceil \frac{m}{2} \rceil} - 1)(2^{\lfloor \frac{m}{2} \rfloor} - 1) + F(\lceil \frac{m}{2} \rceil) + F(\lfloor \frac{m}{2} \rfloor)$$

with

$$F(4) = 22, F(3) = 8, F(2) = 2. \quad (5.8)$$

and

$$G(m) = (2^{\lceil \frac{m}{2} \rceil} + 2^{\lfloor \frac{m}{2} \rfloor} - 2) + G(\lceil \frac{m}{2} \rceil) + G(\lfloor \frac{m}{2} \rfloor)$$

$$G(4) = 10, G(3) = 4, G(2) = 2. \quad (5.9)$$

The total complexity of $F(m)+G(m)$ is $O(2^m)$. When $m > 4$, the new algorithm is much better than the original algorithm.

To illustrated the above concept, consider the following example.

Example 2

Let $M =$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \end{bmatrix},$$

and

$$J = (J_0, J_1, \cdots, J_{14}),$$

Direct multiplication of M and J^T yields

$$Q_1 = J_0 + J_4 + J_7 + J_8 + J_{10} + J_{12} + J_{13} + J_{14},$$
$$Q_2 = J_1 + J_4 + J_5 + J_7 + J_9 + J_{10} + J_{11} + J_{12},$$
$$Q_3 = J_2 + J_5 + J_6 + J_8 + J_{10} + J_{11} + J_{12} + J_{13},$$
$$Q_4 = J_3 + J_6 + J_7 + J_9 + J_{11} + J_{12} + J_{13} + J_{14}.$$

It needs 28 XORs and 4 transfer operations. Using our algorithm, we divide M into two sub-matrices,

$M_u =$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix},$$

and

$M_d =$

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

We rearrange the columns of M_u into 4 groups such that the first, second, third and the fourth groups consist of

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

respectively. Thus, we have

$M'_u =$

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

and

$$Y_1 = (J_1, J_5, J_9, J_{11}),$$
$$Y_2 = (J_0, J_8, J_{13}, J_{14}),$$
$$Y_3 = (J_4, J_7, J_{10}, J_{12}).$$

Using (5.6) and (5.7) we have

$$P_1 = J_1 + J_5 + J_9 + J_{11},$$

$$P_2 = J_0 + J_8 + J_{13} + J_{14},$$

$$P_3 = J_4 + J_7 + J_{10} + J_{12},$$

and

$$Q_1 = P_2 + P_3,$$

$$Q_2 = P_1 + P_3.$$

Hence, to calculate Q_1 and Q_2 using our method needs 11 XORs and 5 transfer operations. Q_3 and Q_4 can be computed using M_d in a similar manner.

□

5.3 Calculate $S[\mu]$ from $Q_\mu[\lambda]$

Given $Q_\mu[\lambda][0], Q_\mu[\lambda][1], \cdots, Q_\mu[\lambda][m-1]$, from (5.3.1), the syndromes can be computed from

$$S[\mu] = \begin{bmatrix} S[\mu][0] \\ S[\mu][1] \\ \vdots \\ S[\mu][m-1] \end{bmatrix} = \sum_{\lambda=0}^{m-1} M_{\alpha^\lambda} \begin{bmatrix} Q_\mu[\lambda][0] \\ Q_\mu[\lambda][1] \\ \vdots \\ Q_\mu[\lambda][m-1] \end{bmatrix}.$$

Using the Horner algorithm, we have

$$\begin{aligned} S[\mu][j] = \; & M_\alpha(\cdots M_\alpha(M_\alpha Q[m-1][j] + \\ & Q[m-2][j]) + Q[m-3][j])\cdots) + Q[0][j] \end{aligned}$$

which needs $(m-1)(m+\delta-2)$ XOR operations and $(m-1)m$ transfer operations, where δ is the number of terms in the primitive polynomial of $GF(2^m)$. For example, the primitive polynomial of $GF(2^{10})$ is $x^{10}+x+1$, then $\delta = 3$. The summation of the number of XORs and transfer operations is $2m^2 - m - 1 + (m-1)(\delta-3)$.

In the following, we introduce an more efficient approach which needs $(m-1)^2 + 2m$ XORs and $2(m-2)+m$ transfer operations. Instead of introducing more notations and describing the general formulas, we illustrate our approach with a specific example.

Example 3
Let $m = 10$ and we want to compute

$$\begin{bmatrix} S[\mu][0] \\ S[\mu][1] \\ \vdots \\ S[\mu][9] \end{bmatrix} = \sum_{\lambda=0}^{9} M_{\alpha\lambda} \begin{bmatrix} Q[\lambda][0] \\ Q[\lambda][1] \\ \vdots \\ Q[\lambda][9] \end{bmatrix}.$$

Consider the 10×19 binary matrix:

$$\widetilde{M} = \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0 \\ 0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1 \\ 0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0&1&0&0&1&0&0&0&1&0 \\ 0&0&0&0&1&0&0&0&0&0&0&1&0&0&1&0&0&0&1 \\ 0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&1&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&1&0 \\ 0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&1 \\ 0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0 \end{bmatrix}$$

where the jth column is the vector expression of α^{j-1} for $j = 1, 2, \cdots, 19$. Also consider the 10×190 binary matrix

$$M \triangleq [M_1, M_\alpha, M_{\alpha^2}, \cdots, M_{\alpha^{18}}]_{10\times 190},$$

where M_{α^j} is the matrix expression of α^j. From (2.4) we know that M_{α^j} consists of the $(j-1)th, jth, \cdots, (j+8)th$ columns of $\widetilde{M}$. Let

$$N = [1,2,3,4,5,6,7,8,9,10,9,8,7,6,5,4,3,2,1],$$

where the jth component is the number of occurrence of the jth column of $\widetilde{M}$ in M. Let

$$\begin{aligned} r[0] = \; & Q[0][0]; \\ r[1] = \; & Q[0][1] + Q[1][0]; \\ \vdots \; & \quad \vdots \\ r[j] = \; & Q[0][j] + Q[1][j-1] + \cdots + Q[j][0]; \\ & \text{for } 0 \le j \le 8 \\ \vdots \; & \quad \vdots \\ r[9] = \; & Q[0][9] + Q[1][8] + \cdots + Q[9][0]; \\ \vdots \; & \quad \vdots \\ r[j] = \; & Q[j-9][9] + Q[j-8][8] + \cdots + Q[9][j-9]; \\ & \text{for } 10 \le j \le 18 \end{aligned}$$

Thus, to calculate all $r[j]$ we need $2(1+2+\cdots+8)+9 = 81$ XORs. From the matrix $\widetilde{M}$, we have

$$\begin{aligned} S[\mu][0] &= r[0] + r[10] + r[17]; \\ S[\mu][1] &= r[1] + r[11] + r[18]; \\ S[\mu][2] &= r[2] + r[12]; \\ S[\mu][3] &= r[3] + r[10] + r[13] + r[17]; \\ S[\mu][4] &= r[4] + r[11] + r[14] + r[18]; \\ S[\mu][5] &= r[5] + r[12] + r[15]; \\ S[\mu][6] &= r[6] + r[13] + r[16]; \\ S[\mu][7] &= r[7] + r[14] + r[17]; \\ S[\mu][8] &= r[8] + r[15] + r[18]; \\ S[\mu][9] &= r[9] + r[16]; \end{aligned}$$

where there are $2+2+1+3+3+2+2+2+2+1 = 20$ XORs. Hence, to calculate $S[\mu][j]$ from $Q_\lambda[j]$ we need 101 XORs and 28 transfer operations.

□

5.4 Calculate $S[2\mu]$ from $S_[\mu]$

From (5.2) we have

$$S[2\mu] = \sum_{\lambda=0}^{m-1} M_{\alpha^\lambda} \begin{bmatrix} Q_{2\mu}[\lambda][0] \\ Q_{2\mu}[\lambda][1] \\ \vdots \\ Q_{2\mu}[\lambda][m-1] \end{bmatrix}, \qquad (5.10)$$

where

$$Q_{2\mu}[\lambda][j] = \sum_{i=0}^{n-1} (2\mu i)_\lambda J[i][j], \quad for \quad j = 0, 1, \cdots, m-1.$$

Using (2.4), $Q_{2\mu}[\lambda][j]$ can be easily derived from $Q_\mu[\lambda][j]$ and

$$\begin{bmatrix} Q_{2\mu}[0][j] \\ Q_{2\mu}[1][j] \\ \vdots \\ Q_{2\mu}[m-1][j] \end{bmatrix} = M \begin{bmatrix} Q_{\mu}[0][j] \\ Q_{\mu}[1][j] \\ \vdots \\ Q_{\mu}[m-1][j] \end{bmatrix}, \qquad (5.11)$$

where M is defined by (2.5).

Thus, to calculate $S[2\mu]$, we first compute $Q_{2\mu}[\lambda][j]$ from $Q_\mu[\lambda][j]$ using (5.11) and then calculate $S[2\mu]$ using (5.10).

5.5 Calculate S_μ When $\mu|(2^m - 1)$

Let $\Delta = \frac{2^m-1}{\mu}$, then we have $\{i|i = 0, 1, \cdots, 2^m - 2\} = \{i + \xi\Delta | i = 0, 1, \cdots, \Delta - 1, \xi = 0, 1, \cdots, \mu - 1\}$. It follows from (5.1) that

$$S[\mu] = \sum_{i=0}^{n-1} M_{\alpha^{\mu i}} J[i][j]$$

$$= \sum_{i=0}^{\Delta-1} \sum_{\xi=0}^{\mu=0} M_{\alpha^\mu}(i + \xi\Delta) J[i + \xi\Delta][j].$$

Since, $\alpha^{\mu\Delta} = \alpha^{2^m-1} = 1$, i. e., $\alpha^{\mu(i+\xi\Delta)} = \alpha^{\mu i}$, then

$$S[\mu] = \sum_{i=0}^{\Delta-1} M_{\alpha^{\mu i}} \sum_{\xi=0}^{\mu-1} J[i + \xi\Delta][j] = \sum_{i=0}^{\Delta-1} M_{\alpha^{\mu i}} J^{'}[i][j], \qquad (5.12)$$

where

$$J^{'}[i][j] = \sum_{\xi=0}^{\mu-1} J[i + \xi\Delta][j]. \qquad (5.13)$$

Using (5.12) and (5.13) to calculate $S[\mu]$, the number of XORs required is

$$(\mu-1)m\Delta + 2m\Delta + m(m-8)2^{\frac{m}{2}} - 1 - m(m-2). \qquad (5.14)$$

6 Simulation Results and Discussion

In this paper we proposed an $(n, k)/(m, l)$ packet-loss resilient code based on an (n, k) Reed Solomon code over $GF(2^m)$. The code accepts k-packet information sequences and encodes them into n-packet codewords, where each packet consists of m l-bit tuples with l an arbitrary positive integer. We also presented procedures for fast encoding and decoding of the code with our description slanted towards decoding operation. By letting l be a multiple of the size of the computer words, almost all of the decoding operations are XOR's of computer words.

To evaluate the performance of our $(n, k)/(m, l)$ packet-loss resilient code and compare it to related work, we simulated the encoding and decoding operations of our code as well as Rizzo's algorithm proposed in [11, 12] on a Pentium II 233 PC with 128MB memory.

In our $(n, k)/(m, l)$ code, the encoder accepts k-packet sequences and outputs n-packet codewords, where $n = 2^m - 1$ and m a positive integer. This code is able to recover any $n - k$ or less lost packets. Each packet consists of m l-bit tuples. For efficiency reason, l should be a multiple of the size of the CPU's computer word which is 32 bits in Pentium II processors. In our simulations, we selected the scenario by letting $l = 32$ bits. Let t be the execution time for encoding/decoding μ codewords in a simulation run. The encoding/decoding speed in bytes per second can be calculated from

$$\frac{n \times m \times l \times \mu}{t \times 8} MB/s.$$

We also simulated Rizzo's scheme which uses a (n, k) Reed-Solomon code over $GF(2^m)$ to recover lost packets. In this scheme, k data packets each of ml bits are encoded into l Reed-Solomon codewords of length n bits. This scheme is able to recover any $n - k$ or less lost packets by performing erasure decoding of the l Reed-Solomon codewords. Again let t be the execution time for encoding/decoding μ codewords in a simulation run. The encoding/decoding speed in bytes per second can be calculated from

$$\frac{n \times m \times \mu}{t \times 8} MB/s.$$

Note that the encoding and decoding speed of the Rizzo's scheme is independent of the parameter l.

The results of the simulations for the cases of $n = 63$ $(m = 6)$, $n = 255$ $(m = 8)$and $n = 1023$ $(m = 10)$ are shown in the table below. In all the cases, the packet loss recovery capability or equivalently the code redundance is kept at $r = n - k = 11$.

In each simulation run, the number of codewords being encoded/decoded is on the order of 10^5. From this table we observe that the decoding speed of our code is 10 to 30 times than Rizzo's scheme. It is interesting to note that the encoding and decoding speed for $n = 255$ is faster than those of $n = 63$ and 1023. This is because 255 is dividable by both 3 and 5, which results in faster syndrome calculations according to our algorithm presented in Section 5.

Table 1: $n = 63$

	Ours	Rizzo's
Encoding	23.6	3.0
Decoding		
1 error	378	11.8
2 errors	126	3.4
3 errors	108	3.0
4 errors	54.0	2.4
5 errors	44.5	1.8
6 errors	30.2	1.5
7 errors	25.2	1.2
8 errors	18.4	1.0
9 errors	15.4	0.9
10 errors	12.6	0.7
11 errors	11.0	0.6

Table 2: $n = 255$

	Ours	Rizzo's
Encoding	32.6	3.3
Decoding		
1 error	272	42.5
2 errors	204	11.6
3 errors	136	8.0
4 errors	74.2	6.1
5 errors	62.8	4.7
6 errors	45.3	3.9
7 errors	38.7	3.3
8 errors	32.6	2.8
9 errors	29.1	2.4
10 errors	26.3	2.1
11 errors	24.7	1.8

Table 3: $n = 1023$

	Ours	Rizzo's
Encoding	16.1	0.55
Decoding		
1 error	127.9	32.0
2 errors	62.0	4.57
3 errors	62.0	2.46
4 errors	42.6	1.72
5 errors	40.9	1.30
6 errors	24.1	1.05
7 errors	23.5	0.88
8 errors	16.6	0.76
9 errors	16.5	0.67
10 errors	15.9	0.59
11 errors	15.6	0.53

References

[1] S. Lin and D. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Englewood Cliffs, NJ, Prentice-Hall, 1983.

[2] S. Floyd, V. Jacbson, C. Liu, S. McCanne, L. Zhang, "A Reliable Multicast Framework for Lightweight Sessions and Application Level Framing, Scalable Reliable Multicast (SRM)", *ACM SIGCOMM 95*.

[3] R. Deng,"Hybrid ARQ Schemes for Point-to-Multipoint Communications over Nonstationary Broadcast Channels," *IEEE Transactions on Communications*, Vol. 41, No. 9, pp. 1379-1387, Sept. 1993

[4] J. Nonnenmacher, E. Biersack and D. Towsley, "Parity- Based Loss Recovery for Reliable Multicast Transmission", *IEEE/ACM Transactions on Networking*, Vol. 6, No. 4, pp. 349-361, Aug. 1998.

[5] M. Handley, "An Examination of Mbone Performance", *USC/ISI Research Report: ISI/RR-97-450*, April 1997.

[6] I. Gopal and J. Jaffe, "Point-to-Multipoint Communication Over Broadcast Links", *IEEE Transactions on Communications*, Vol 32, pp. 1034-1044, 1984.

[7] M. Yajnik, J. Kurose, D. Towsley, "Packet Loss Correlation in the MBone multicast Network", *IEEE Global Communications Conf.*, pp. 94-99, London, Nov. 1996.

[8] E. Biersack, "Performance Evaluation of Forward Error Correction in ATM Networks", *Proceedings of SIGCOMM'92*, pp.248-257, Baltimore, Aug. 1992.

[9] G. Carle, "Towards scaleable error control for reliable multipoint services in ATM networks", 12th Int. Conf. on Computer Communication, ICCC'95, Seoul, Koera, Aug. 1995.

[10] A. McAuley, "Reliable Broadband Communication Using a Burst Erasure Correcting Code", *ACM SIGCOMM'90*, pp. 297-306, Sep. 1990.

[11] L. Rizzo, "Effective Erasure Codes for Reliable Computer Communication Protocols", *ACM Computer Communication Review*, Vol. 27, n.2, Apr. 97, pp.24-36.

[12] L. Rizzo, L. Vicisano, "RMDP: an FEC-Based Reliable Multicast Protocol for Wireless Environments", *ACM Mobile Computing and Communications Review*, Vol. 2, No. 2, April 1998.

[13] T.Nguyen and A.Zakhor, "Distributed Video Streaming with Forward Error Correction", *International Packet Video Workshop 2002*, Apr. 24-26, 2002, Pittsburgh, PA.

[14] A.E.Mohr, E.A.Riskin, and R.E.Ladner, "Unequal Loss Protection: Graceful Degradation of Image Quality over Packet Erasure Channels through Forward Error Corrections", *IEEE Journal of Selected Areas in Communications*, vol. 18, No.6, pp.819-828, June 2000.

[15] R.E.Blahut, *Theory and Practice of Error Control Codes*, Addison Wesley, MA, 1984

[16] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby and D. Zuckerman, "An XOR-based Erasure-Resilient Coding Scheme", *ICSI Technical Report*, No. TR-95-048, Aug 1995.

[17] Michael G. Luby, Michael Mitzenmacher, M. Amin Shokrollahi, Daniel A. Spielman, Volker Stemann, "Practical Loss-Resilient Codes", *STOC'97*, El Paso, Texas, May 1997.

[18] Michael Luby, Michael Mitzenmacher, Amin Shokrollahi,"Analysis of Random Processes via And-Or Tree Evaluation", ICSI TR-97-042

[19] P.Elias, "Coding for Two Noisy Channels", *Information Theory*, Third London Symposium, pp.61-76, September 1955,

The foolowing are used for discissions on data storage and steaming authentication in the last chapter.

[20] R.Yavatkar, J.Griffioen, M.Sudan, "A Reliable Dissemination Protocol for Interactive Collaborative Applications", *ACM Multimedia 95*.

[21] J.M.Park, E.K. S.Chong, and H.J.Siegal, "Efficient Multicast Packet Authentication Using Signature Amortization", *2002 IEEE Symposium on Security and Privacy*

[22] D.Dean, M.Franklin, and A.Stubbdefied, "An Algebraic Approach to IP Traceback", *ACM Transcation on Information and System Security*, Vol. 5, No.2, May 2002.