Reed-Solomon Forward Error Correction (FEC)

Jérôme Lacan, Vincent Roca, Sami Peltotalo, Jani Peltotalo

Reliable Multicast Transport
Internet-Draft
Expires: August 27, 2006

J. Lacan
ENSICA/LAAS-CNRS
V. Roca
INRIA
J. Peltotalo
S. Peltotalo
Tampere University of Technology

February 23, 2006

Reed-Solomon Forward Error Correction (FEC)
draft-ietf-rmt-bb-fec-rs-00.txt

Status of this Memo

Copyright Notice

Abstract

This document describes a Fully-Specified FEC scheme for the Reed-Solomon forward error correction code and its application to reliable delivery of data objects on the packet erasure channel.

The Reed-Solomon codes belong to the class of Maximum Distance Separable (MDS) codes, i.e, they enable a receiver to recover the k source symbols from any set of k received symbols.

The implementation described here is compatible with the IPR-free implementation from Luigi Rizzo.

Table of Contents

1. Introduction

The use of Forward Error Correction (FEC) codes is a classical solution to improve the reliability of multicast and broadcast transmissions. The [RFC3452] and [draft-ietf-rmt-fec-bb-revised-03] documents describe a general framework to use FEC in Content Delivery Protocols (CDP). The companion document [RFC3453] describes some applications of FEC codes for content delivery.

Recent FEC schemes like [draft-ietf-rmt-bb-fec-raptor-object-03] and [draft-ietf-rmt-bb-fec-ldpc-01] proposed erasure codes based on sparse graphs/matrices. These codes are efficient in terms of CPU but not optimal in terms of correction capabilities, at least for small objects.

The FEC scheme presented in this document belongs to the class of Maximum-Distance Separable codes, i.e., it is optimal in terms of erasure correction capability. In others words, it enables the receiver to recover the k source symbols from any set of k encoding symbols. Even if the encoding/decoding complexity is larger than that of [draft-ietf-rmt-bb-fec-raptor-object-03] or [draft-ietf-rmt-bb-fec-ldpc-01], this family of codes is very useful for applications sending small objects (e.g., for video and audio streaming).

Indeed many applications dealing with content transmission or content storage already rely on packet-based Reed-Solomon codes. In particular, many of them are derived from the implementation of Luigi Rizzo [RS-Rizzo]. This latter is compatible with the Reed-Solomon codes specification of the present document.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [rfc2119].

3. Definitions Notations and Abbreviations

3.1. Definitions

This document uses the same terms and definitions as those specified in [draft-ietf-rmt-fec-bb-revised-03]. Additionally, it uses the following definitions:

Source symbol: unit of data used during the encoding process.

Encoding symbol: unit of data generated by the encoding process.

Repair symbol: encoding symbols that are not source symbols.

Systematic code: a code in which the source symbols are part of the encoding symbols

Source block: a block of k source symbols that are considered together for the encoding.

Encoding Symbol Group: a group of encoding symbols that are sent together, within the same packet, and whose relationships to the source object can be derived from a single Encoding Symbol ID.

Source Packet a data packet containing only source symbols.

Repair Packet a data packet containing only repair symbols.

3.2. Notations

This document uses the following notations:

L denotes the object transfer length in bytes

k denotes the number of source symbols in a source block

n_r denotes the number of repair symbols generated for a source block

n denotes the encoding block length, i.e., the number of encoding symbols generated for a source block. Then n = k+ n_r

max_n Maximum Number of Encoding Symbols generated for any source block

B denotes the maximum source block length in symbols, i.e., the maximum number of source symbols per source block

N denotes the number of source blocks into which the object shall be partitioned

E denotes the encoding symbol length in bytes

sz denotes the symbol size in units of m bit elements

m defines the number of elements in the finite field, namely q = 2^^m.

G denotes the number of encoding symbols per group, i.e., the number of symbols sent in the same packet

rate denotes the so-called "code rate", i.e. the k/n ratio

a^^b denotes a raised to the power b

a^^-1 denotes the inverse of a

I_k denotes the k*k identity matrix

3.3. Abbreviations

This document uses the following abbreviations:

ESI Encoding Symbol ID

RS Reed-Solomon

MDS Maximum Distance Separable code

GF(q) finite field (A.K.A. Galois Field) with q elements

4. Formats and Codes

4.1. FEC Payload IDs

The FEC Payload ID is composed of the Source Block Number and the Encoding Symbol ID:

- The Source Block Number (16 bit field) identifies from which source block of the object the encoding symbol(s) in the payload is (are) generated. There is a maximum of 2^^16 blocks per object.

- The Encoding Symbol ID (16 bit field) identifies which specific encoding symbol generated from the source block is carried in the packet payload. There is a maximum of 2^^16 encoding symbols per block. The first k values (0 to k-1) identify source symbols, the remaining n-k values identify repair symbols.

There MUST be exactly one FEC Payload ID per packet. In case of an Encoding Symbol Group, when multiple encoding symbols are sent in the same packet, the FEC Payload ID refers to the first symbol of the packet. The other symbols can be deduced from the ESI of the first symbol by incrementing sequentially the ESI.

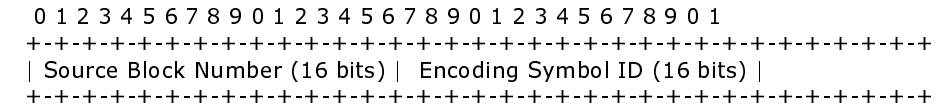

Figure 1: FEC Payload ID encoding format for FEC Encoding ID XX

4.2. FEC Object Transmission Information

4.2.1. Mandatory Elements

- FEC Encoding ID: the Fully-Specified FEC Scheme described in this document use the FEC Encoding ID XX.

4.2.2. Common Elements

The following elements MUST be defined with the present FEC Scheme:

- Transfer-Length (L): a non-negative integer indicating the length of the object in bytes. There are some restrictions on the maximum Transfer-Length that can be supported:

max_transfer_length = 2^^16 * B * E

For instance, if B = 2^^8-1 (because the codec operates on a finite field with 2^^8 elements), and if E = 1024 bytes, then the maximum transfer length is 2^^34 bytes (i.e., a bit more than 17 Giga Bytes). For larger objects, it is expected that other FEC codes (e.g., LDPC codes) or another Reed-Solomon FEC Scheme with a larger Source Block Number field in the FEC Payload ID be used.

- o Encoding-Symbol-Length (E): a non-negative integer indicating the length of each encoding symbol in bytes.

- o Maximum-Source-Block-Length (B): a non-negative integer indicating the maximum number of source symbols in a source block.

- o Max-Number-of-Encoding-Symbols (max_n): a non-negative integer indicating the maximum number of encoding symbols generated for any source block.

Section 5 explains how to derive the values of each of these elements.

4.2.3. Scheme-Specific Elements

The following element MUST be defined with the present FEC Scheme. It contains two distinct pieces of information:

- o G: a non-negative integer indicating the number of encoding symbols per group used for the object. The default value is 1, meaning that each packet contains exactly one symbol. When no G parameter is communicated to the decoder, then this latter MUST assume that G = 1.

- o Finite Field size parameter, m: The m parameter defines the finite field size equal to q = p^^m elements. The default value is m = 8. When no finite field size parameter is communicated to the decoder, then this latter MUST assume that m = 8.

4.2.4. Encoding Format

This section shows two possible encoding formats of the above FEC OTI. The present document does not specify when or how these encoding formats should be used.

4.2.4.1. Using the General EXT_FTI Format

The FEC OTI binary format is the following, when the EXT_FTI mechanism is used.

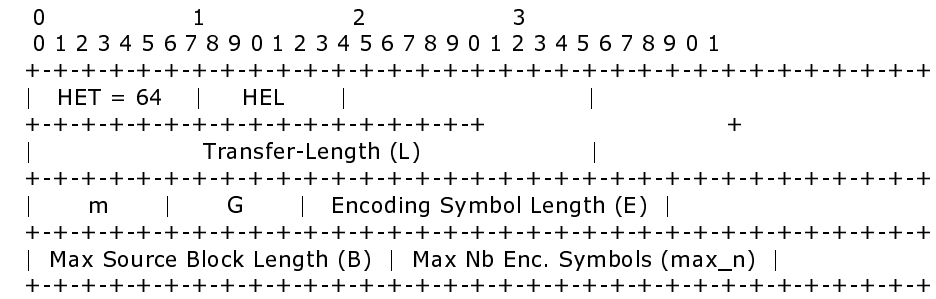

Figure 2: EXT_FTI Header Format

4.2.4.2. Using the FDT Instance (FLUTE specific)

When it is desired that the FEC OTI be carried in the FDT Instance of a FLUTE session, the following XML elements must be described for the associated object:

- FEC-OTI-Transfer-length
- FEC-OTI-Encoding-Symbol-Length
- FEC-OTI-Maximum-Source-Block-Length
- FEC-OTI-Max-Number-of-Encoding-Symbols
- FEC-OTI-Number-Encoding-Symbols-per-Group (optional)
- FEC-OTI-Finite-Field-Size-Parameter (optional)

When no finite field size parameter is to be carried in the FEC OTI, the sender simply omits the FEC-OTI-Finite-Field-Size-Parameter element.

5. Procedures

This section defines procedures for FEC Encoding ID XX.

5.1. Determining the Maximum Source Block Length (B)

The finite field size parameter, m, defines the number of non zero elements in this field, q = 2^^m-1. Note that q is also the theoretical maximum number of encoding symbols that can be produced for a source block. For instance, when m = 8 (default):

max1_B = 2^^8-1

Additionally, a codec MAY impose other limitations on the maximum block size. Yet it is not expected that such limits exist when using m = 8 (default). This decision SHOULD be clarified at implementation time, when the target use case is known. This results in a max2_B limitation.

Then, B is given by:

B = min(max1_B, max2_B)

Note that this calculation is only required at the coder, since the B parameter is communicated to the decoder through the FEC OTI.

5.2. Determining the Number of Encoding Symbols of a Block

The following algorithm, also called "n-algorithm", explains how to determine the actual number of encoding symbols for a given block.

AT A SENDER:

Input:

B: Maximum source block length, for any source block. Section 5.1 explains how to determine its value.

k: Current source block length. This parameter is given by the source blocking algorithm.

rate: FEC code rate, which is given by the user (e.g., when starting a FLUTE sending application) for a given use case. It is expressed as a floating point value.

Output:

max_n: Maximum number of encoding symbols generated for any source block

n: Number of encoding symbols generated for this source block

Algorithm:

```
max_n = floor(B / rate);

if (max_n >= 2^^m) then return an error ("invalid code rate");

n = floor(k * max_n / B);
```

AT A RECEIVER:

Input:

B Extracted from the received FEC OTI

max_n Extracted from the received FEC OTI

k Given by the source blocking algorithm

Output:

n

Algorithm:

```
n = floor(k * max_n / B);
```

Note that a Reed-Solomon decoder does not need to know the n value. Therefore the receiver part of the "n-algorithm" is not necessary from the Reed-Solomon decoder point of view. Yet a receiving application using the Reed-Solomon FEC scheme will sometimes need to know the value of n used by the sender, for instance for memory management optimizations. To that purpose, all the needed information is carried in the FEC OTI.

6. Reed-Solomon Codes

Reed-Solomon (RS) codes form a special class of linear block codes, which offer maximum erasure correction capability. A [n,k]-RS code encodes a sequence of k source elements defined over a finite field GF(q) into a sequence of n encoding elements, where n is upperbounded by q-1. The implementation described in this document is based on a generator matrix built from a Vandermonde matrix put into systematic form.

6.1. Finite field

A finite field GF(q) is defined as a finite set of q elements which have a structure of field. It contains necessarily q = p^^m elements, where p is a prime number. With packet erasure channels, p is always set to 2. The elements of the field GF(2^^m) can be represented by polynomials with binary coefficients (i.e., over GF(2)) of degree less than m. The polynomials can be associated to binary vectors of length m. For example, the vector (11001) represents the polynomial 1 + x + x^^4. This representation is often called polynomial representation. The addition between two elements is defined as the addition of binary polynomials in GF(2) and the multiplication is the multiplication modulo a given irreducible (i.e., non-factorizable) polynomial of degree m with coefficients in GF(2).

Since a finite field GF(2^^m) is completely characterized by the irreducible polynomial, we propose the following polynomials to represent the field GF(2^^m), for m varying from 2 to 16:

m = 2, "111" (1+x+x^^2)

m = 3, "1101", (1+x+x^^3)

m = 4, "11001", (1+x+x^^4)

m = 5, "101001", (1+x^^2+x^^5)

m = 6, "1100001", (1+x+x^^6)

m = 7, "10010001", (1+x^^3+x^^7)

m = 8, "101110001", (1+x^^2+x^^3+x^^4+x^^8)

m = 9, "1000100001", (1+x^^4+x^^9)

m = 10, "10010000001", (1+x^^3+x^^10)

m = 11, "101000000001", (1+x^^2+x^^11)

m = 12, "1100101000001", (1+x+x^^4+x^^6+x^^12)

m = 13, "11011000000001", (1+x+x^^3+x^^4+x^^13)

m = 14, "110000100010001", (1+x+x^^6+x^^10+x^^14)

m = 15, "1100000000000001", (1+x+x^^15)

m = 16, "11010000000010001", (1+x+x^^3+x^^12+x^^16)

For implementation issues, these polynomials are also primitive elements of GF(2^^m), i.e., any element of GF(2^^m) can be expressed as a power of a root of this polynomial. These polynomials also contain the minimum number of monomials.

6.2. Reed-Solomon Encoding Algorithm

The encoding algorithm produces a vector of n encoding elements e=(e_0, ..., e_(n-1)) over GF(2^^m) from a source vector of k elements s=(s_0, ..., s_(k-1)) over GF(2^^m).

The linear codes can be encoded by multiplying the source vector by a generator matrix Gm of k rows and n columns over GF(2^^m). Thus: e = s * Gm. The definition of the generator matrix completely characterizes the code.

Let us consider that: n = 2^^m - 1 and: 0 < k <= n. Let us denote alpha a primitive element of GF(2^^m) (i.e., any element of GF(2^^m) can be expressed as a power of alpha).

The generator matrix is build from a k*n-Vandermonde matrix denoted by V_{k,n}. The entries of V_{k,n} are v_{i,j} = alpha^^(i*j), where 0 <= i <= k - 1 and 0 <= j <= n - 1. This matrix generates a MDS code. However, it is not systematic as required by most of network applications. To obtain a systematic matrix, the simplest solution is to consider the matrix V_{k,k} formed by the first k columns of V_{k,n} then to invert it and to multiply this inverse by V_{k,n}. Clearly, the product V_{k,k}^^-1 * V_{k,n} contains the identity matrix I_k on its first k columns and generates a MDS code.

The product V_{k,k}^^-1 * V_{k,n} is denoted by Gm and is the generator matrix of the code considered in this document.

Note that, for practical applications, the length of the code can be shortened to k <= n' < n by considering the sub-matrix formed by the n' first columns of Gm.

6.2.1. Encoding Complexity

The encoding process can be done by first pre-computing G and by multiplying the source vector by Gm. The complexity is one multiplication s*Gm, where Gm is a k*(n-k) matrix. The complexity of the vector-matrix multiplication is then k*(n-k) (i.e., k operations per repair element).

The encoding can also be processed by first computing the product s* V_{k,k}^^-1 and then by multiplying the result by V_{k,n}. The multiplication by the inverse of a square Vandermonde matrix is known as the interpolation problem and its complexity is O(k log^^2 (k)). The multiplication by a Vandermonde matrix, known as the multipoint evaluation problem, requires O((n-k) log(k)) by using Fast Fourier Transform, as explained in [fastMatrix-vectorMultiplication]. The total complexity of this encoding algorithm is then O(k/(n-k) log^^2 (k)+ log(k)) operations per repair symbol.

6.3. Reed-Solomon Decoding Algorithm for the Erasure Channel

The Reed-Solomon decoding algorithm for the erasure channel allows the recovery of the k source elements from any set of k received elements. It is based on the fundamental property of the generator matrix which is such that any k*k-submatrix is invertible (see [MWS]). The first step of the decoding consists in extracting the k*k submatrix of the generator matrix obtained by considering the columns corresponding to the received symbols. Indeed, since any encoding element is obtained by multiplying the source vector by one column of the generator matrix, the received vector of k encoding symbols can be considered as the result of the multiplication of the source vector by a k*k submatrix of the generator matrix. Since this submatrix is invertible, the second step of the algorithm is to invert this matrix and to multiply the received vector by the obtained matrix to recover the source vector.

6.3.1. Decoding Complexity

The decoding algorithm described previously includes the matrix inversion and the vector-matrix multiplication. With the classical Gauss-Jordan algorithm, the matrix inversion requires O(k^^3) operations and the vector-matrix multiplication is performed in O(k^^2) operations.

This complexity can be improved by considering that the received submatrix of Gm is the product between the inverse of a Vandermonde matrix (V_(k,k)^^-1) and another Vandermonde matrix (denoted by V' which is a submatrix of V_(k,n)). The decoding can be done by multiplying the received vector by V'^^-1 (interpolation problem with

complexity O(k log^^2(k))) then by V_{k,k} (multipoint evaluation with complexity O(k log(k))). The global decoding complexity is then O(log^^2(k)) operations per source symbol.

6.4. Implementation

6.4.1. Implementation for the Packet Erasure Channel

In a packet erasure channel, each packet is either received correctly or erased. The location of the erased packets in the sequence of packets must be known. The following specification describes the use of Reed-Solomon codes for generating redundant packets from k source packets and to recover the source packets from k received packets.

The k source symbols of a source block are assumed to be composed of sz m-bit elements. Each m-bit element is associated to an element of the finite field GF(2^^m) through the polynomial representation (Section 6.1). If some of the source symbols contain less than sz elements, they are virtually padded with zero elements (it can be the case for the last symbol of the last block of the object).

The encoding processing produces n-k repair symbols of sz elements by encoding each of the sz encoding vectors from the sz source vectors (Figure 3). The j-th source vector is composed of the j-th element of each of the source symbols. Similarly, the j-th encoding vector is composed of the j-th element of each encoding symbol.

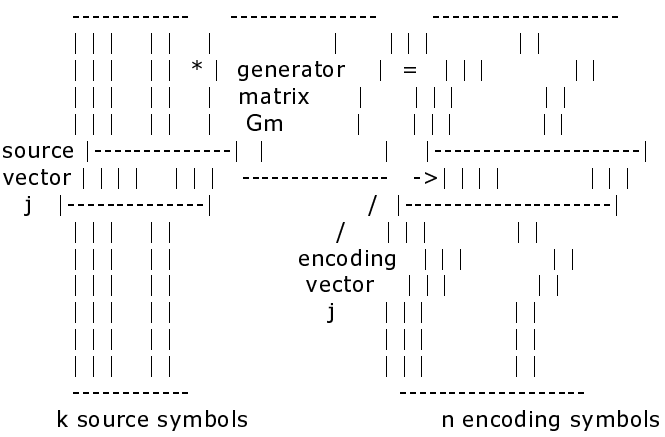

Figure 3: Packet encoding scheme

An asset of this scheme is that the loss of some of encoding symbols produce the same erasure pattern for each of the sz encoding vectors.

It follows that the matrix inversion must be done only once and will be used by all the sz encoding vectors. For large sz, this complexity cost of the inversion becomes negligible compared to the sz matrix-vector multiplications.

Another asset is that repair symbols can be produced on demand, e.g., depending on the observed erasures on the channel. The only constraint is the finite field size (see Section 6.1)

7. Security Considerations

The security considerations for this document are the same as that of [RFC3452].

8. Intellectual Property

To the best of our knowledge, there is no patent or patent application identified as being used in the Reed-Solomon FEC scheme. Yet other flavors of Reed-Solomon codes and associated techniques MAY be covered by Intellectual Property Rights.

9. IANA Considerations

Values of FEC Encoding IDs and FEC Instance IDs are subject to IANA registration. For general guidelines on IANA considerations as they apply to this document, see [draft-ietf-rmt-fec-bb-revised-03]. This document assigns the Fully-Specified FEC Encoding ID XX under the ietf:rmt:fec:encoding name-space to "Reed-Solomon Codes".

10. Acknowledgments

11. References

11.1. Normative References

[RFC3452] Luby, M., "Forward Error Correction (FEC) Building Block", RFC 3452, December 2002.

[RFC3453] Luby, M., "The Use of Forward Error Correction (FEC) in Reliable Multicast", RFC 3453, December 2002.

[draft-ietf-rmt-fec-bb-revised-03]
Watson, M., Luby, M., and L. Vicisano, "Forward Error Correction (FEC) Building Block", draft-ietf-rmt-fec-bb-revised-03.txt (work in progress), January 2006.

[rfc2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119.

11.2. Informative References

[MWS] Mac Williams, F. and N. Sloane, "The Theory of Error Correcting Codes", North Holland, 1977 .

[RS-Rizzo]
Rizzo, L., "New version of the FEC code (revised 2 july 98), available at http://info.iet.unipi.it/~luigi/vdm98/vdm980702.tgz", July 1998.

[draft-ietf-rmt-bb-fec-ldpc-01]
Roca, V., Neumann, C., and D. Furodet, "Low Density Parity Check (LDPC) Forward Error Correction", draft-ietf-rmt-bb-fec-ldpc-01.txt (work in progress), March 2006.

[draft-ietf-rmt-bb-fec-raptor-object-03]
Luby, M., "Raptor Forward Error Correction Scheme", Internet Draft (draft-ietf-rmt-bb-fec-raptor-object-03 : work in progress), October 2005.

[fastMatrix-vectorMultiplication]
Gohberg, I. and V. Olshevsky, "Fast algorithms with preprocessing for matrix-vector multiplication problems", Journal of Complexity, pp. 411-427, vol. 10, 1994 .

Authors' Addresses

Jerome Lacan
ENSICA/LAAS-CNRS
1, place Emile Blouin
Toulouse 31056
France

Email: jerome.lacan@ensica.fr
URI:

Vincent Roca
INRIA
655, av. de l'Europe
Zirst; Montbonnot
ST ISMIER cedex 38334
France

Email: vincent.roca@inrialpes.fr
URI: http://planete.inrialpes.fr/~roca/

Jani Peltotalo
Tampere University of Technology
P.O. Box 553 (Korkeakoulunkatu 1)
Tampere FIN-33101
Finland

Email: jani.peltotalo@tut.fi
URI:

Sami Peltotalo
Tampere University of Technology
P.O. Box 553 (Korkeakoulunkatu 1)
Tampere FIN-33101
Finland

Email: sami.peltotalo@tut.fi
URI:

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.