           Reed-Solomon Forward Error Correction (FEC) Schemes

Status of This Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements. Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol. Distribution of this memo is unlimited.

Abstract

   This document describes a Fully-Specified Forward Error Correction
   (FEC) Scheme for the Reed-Solomon FEC codes over GF(2^^m), where m is
   in {2..16}, and its application to the reliable delivery of data
   objects on the packet erasure channel (i.e., a communication path
   where packets are either received without any corruption or discarded
   during transmission). This document also describes a Fully-Specified
   FEC Scheme for the special case of Reed-Solomon codes over GF(2^^8)
   when there is no encoding symbol group. Finally, in the context of
   the Under-Specified Small Block Systematic FEC Scheme (FEC Encoding
   ID 129), this document assigns an FEC Instance ID to the special case
   of Reed-Solomon codes over GF(2^^8).

   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 schemes described
   here are compatible with the 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 [RFC5052] document describes a general framework
   to use FEC in Content Delivery Protocols (CDPs). The companion
   document [RFC3453] describes some applications of FEC codes for
   content delivery.

   Recent FEC schemes like [RFC5053] and [RFC5170] proposed erasure
   codes based on sparse graphs/matrices. These codes are efficient in
   terms of processing but not optimal in terms of correction
   capabilities when dealing with "small" objects.

   The FEC schemes described in this document belongs to the class of
   Maximum Distance Separable codes that are optimal in terms of erasure
   correction capability. In others words, it enables a receiver to
   recover the k source symbols from any set of exactly k encoding
   symbols. They are also systematic codes, which means that the k
   source symbols are part of the encoding symbols. Even if the
   encoding/decoding complexity is larger than that of [RFC5053] or
   [RFC5170], this family of codes is very useful.

   Many applications dealing with content transmission or content
   storage already rely on packet-based Reed-Solomon codes. In
   particular, many of them use the Reed-Solomon codec of Luigi Rizzo
   [RS-codec] [Rizzo97]. The goal of the present document is to specify
   an implementation of Reed-Solomon codes that is compatible with this
   codec.

   The present document:

   o introduces the Fully-Specified FEC Scheme with FEC Encoding ID 2,
      which specifies the use of Reed-Solomon codes over GF(2^^m), where
      m is in {2..16},

   o introduces the Fully-Specified FEC Scheme with FEC Encoding ID 5,
      which focuses on the special case of Reed-Solomon codes over
      GF(2^^8) and no encoding symbol group (i.e., exactly one symbol
      per packet), and

   o in the context of the Under-Specified Small Block Systematic FEC
      Scheme (FEC Encoding ID 129) [RFC5445], assigns the FEC Instance
      ID 0 to the special case of Reed-Solomon codes over GF(2^^8) and
      no encoding symbol group.

   For a definition of the terms Fully-Specified and Under-Specified FEC
   Schemes, see [RFC5052], Section 4.

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 [RFC5052]. 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 symbol that is not a source symbol.

      Code rate: the k/n ratio, i.e., the ratio between the number of
         source symbols and the number of encoding symbols. By
         definition, the code rate is such that: 0 < code rate <= 1. A
         code rate close to 1 indicates that a small number of repair
         symbols have been produced during the encoding process.

      Systematic code: FEC 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 block 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.

      Packet Erasure Channel: a communication path where packets are
         either dropped (e.g., by a congested router, or because the
         number of transmission errors exceeds the correction
         capabilities of the physical layer codes) or received. When a
         packet is received, it is assumed that this packet is not
         corrupted.

3.2. Notations

   This document uses the following notations:

       L   the object transfer length in bytes.

       k   the number of source symbols in a source block.

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

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

       max_n   the maximum number of encoding symbols generated for any
               source block.

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

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

       E   the encoding symbol length in bytes.

       S   the symbol size in units of m-bit elements. When m = 8,
               then S and E are equal.

       m   the length of the elements in the finite field, in bits.
               In this document, m belongs to $\{2..16\}$.

       q   the number of elements in the finite field. We have: q =
               2^m in this specification.

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

       GM   the Generator Matrix of a Reed-Solomon code.

       CR   the "code rate", i.e., the k/n ratio.

       a^^b   a raised to the power b.

       a^^-1   the inverse of a.

       I_k   the k*k identity matrix.

3.3. Abbreviations

   This document uses the following abbreviations:

      ESI   Encoding Symbol ID.

      FEC OTI   FEC Object Transmission Information.

      RS   Reed-Solomon.

      MDS   Maximum Distance Separable code.

      GF(q)   a finite field (also known as Galois Field) with q
                elements. We assume that $q = 2^m$ in this document.

4. Formats and Codes with FEC Encoding ID 2

   This section introduces the formats and codes associated with the
   Fully-Specified FEC Scheme with FEC Encoding ID 2, which specifies
   the use of Reed-Solomon codes over $GF(2^m)$.

4.1. FEC Payload ID

   The FEC Payload ID is composed of the Source Block Number and the
   Encoding Symbol ID. The lengths of these two fields depend on the
   parameter m (which is transmitted in the FEC OTI) as follows:

   o The Source Block Number (field of size 32-m bits) identifies from
      which source block of the object the encoding symbol(s) in the
      payload are generated. There is a maximum of $2^{(32-m)}$ blocks per
      object.

   o The Encoding Symbol ID (field of size m bits) identifies which
      specific encoding symbol(s) generated from the source block are
      carried in the packet payload. There is a maximum of 2^m
      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 source or repair 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.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Source Block Number (32-8=24 bits)      | Enc. Symb. ID |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
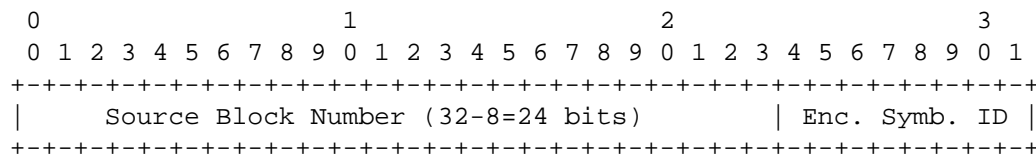
        Figure 1: FEC Payload ID Encoding Format for m = 8 (Default)

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Src Block Nb (32-16=16 bits) |   Enc. Symbol ID (m=16 bits)   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
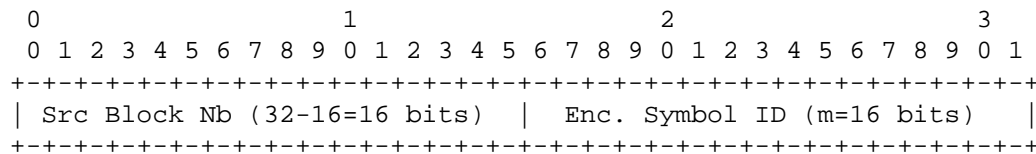
            Figure 2: FEC Payload ID Encoding Format for m = 16

   The formats of the FEC Payload ID for m = 8 and m = 16 are
   illustrated in Figure 1 and Figure 2, respectively.

4.2. FEC Object Transmission Information

4.2.1. Mandatory Elements

   o FEC Encoding ID: the Fully-Specified FEC Scheme described in this
      section uses FEC Encoding ID 2.

4.2.2. Common Elements

   The following elements MUST be defined with the present FEC scheme.

   o 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:

         $max\_transfer\_length = 2^{(32-m)} * B * E$

   For instance, for m = 8, for $B = 2^8 - 1$ (because the codec
   operates on a finite field with 2^8 elements), and if E = 1024
   bytes, then the maximum transfer length is approximately equal to
   2^{42} bytes (i.e., 4 terabytes). Similarly, for m = 16, for $B =
   2^{16} - 1$, and if E = 1024 bytes, then the maximum transfer length
   is also approximately equal to 2^{42} bytes. For larger objects,
   another FEC scheme, with a larger Source Block Number field in the
   FEC Payload ID, could be defined. Another solution consists in
   fragmenting large objects into smaller objects, each of them
   complying with the above limits.

   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 6 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 the latter MUST
      assume that G = 1.

   o m: The m parameter is the length of the finite field elements, in
      bits. It also characterizes the number of elements in the finite
      field: $q = 2^m$ elements. The default value is m = 8. When no
      finite field size parameter is communicated to the decoder, then
      the latter MUST assume that m = 8.

4.2.4. Encoding Format

   This section shows the two possible encoding formats of the above FEC
   OTI. The present document does not specify when one encoding format
   or the other 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 (e.g., within the ALC [ALC] or NORM [NORM]
   protocols).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    HET = 64   |    HEL = 4    |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                       Transfer Length (L)                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       m       |       G       |    Encoding Symbol Length (E) |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Max Source Block Length (B)  |  Max Nb Enc. Symbols (max_n)  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
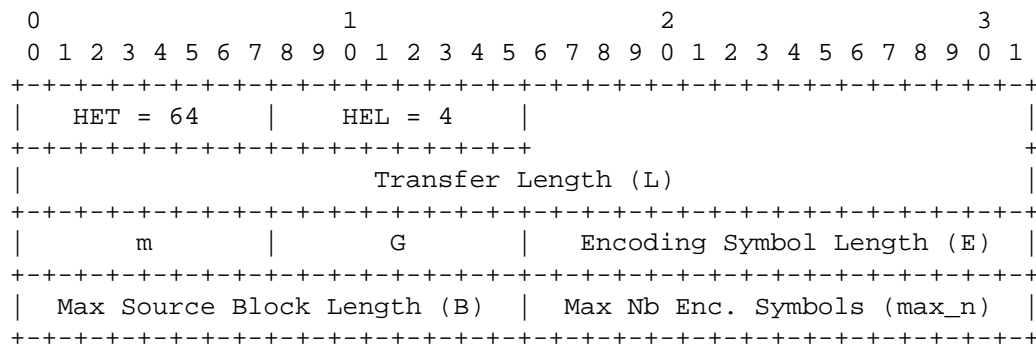
                    Figure 3: EXT_FTI Header Format

4.2.4.2. Using the FDT Instance (FLUTE specific)

   When it is desired that the FEC OTI be carried in the FDT (File
   Delivery Table) Instance of a FLUTE session [FLUTE], the following
   XML attributes must be described for the associated object:

   o FEC-OTI-FEC-Encoding-ID

   o FEC-OTI-Transfer-Length (L)

   o FEC-OTI-Encoding-Symbol-Length (E)

   o FEC-OTI-Maximum-Source-Block-Length (B)

   o FEC-OTI-Max-Number-of-Encoding-Symbols (max_n)

   o FEC-OTI-Scheme-Specific-Info

   The FEC-OTI-Scheme-Specific-Info contains the string resulting from
   the Base64 encoding (in the XML Schema xs:base64Binary sense) of the
   following value:

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       m       |       G       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

    Figure 4: FEC OTI Scheme Specific Information To Be Included in the
                            FDT Instance

   When no m parameter is to be carried in the FEC OTI, the m field is
   set to 0 (which is not a valid seed value). Otherwise, the m field
   contains a valid value as explained in Section 4.2.3. Similarly,

when no G parameter is to be carried in the FEC OTI, the G field is
set to 0 (which is not a valid seed value). Otherwise, the G field
contains a valid value as explained in Section 4.2.3. When neither m
nor G are to be carried in the FEC OTI, then the sender simply omits
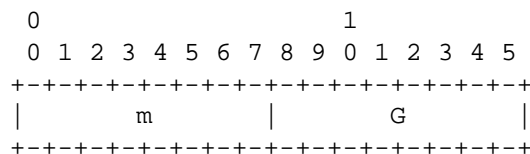the FEC-OTI-Scheme-Specific-Info attribute.

During Base64 encoding, the 2 bytes of the FEC OTI Scheme-Specific
Information are transformed into a string of 4 printable characters
(in the 64-character alphabet) that is added to the FEC-OTI-Scheme-
Specific-Info attribute.

5. Formats and Codes with FEC Encoding ID 5

This section introduces the formats and codes associated with the
Fully-Specified FEC Scheme with FEC Encoding ID 5, which focuses on
the special case of Reed-Solomon codes over $GF(2^8)$ and no encoding
symbol group.

5.1. FEC Payload ID

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

o The Source Block Number (24-bit field) identifies from which
   source block of the object the encoding symbol in the payload is
   generated. There is a maximum of 2^{24} blocks per object.

o The Encoding Symbol ID (8-bit field) identifies which specific
   encoding symbol generated from the source block is carried in the
   packet payload. There is a maximum of 2^8 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 source or repair packet.
This FEC Payload ID refers to the one and only symbol of the packet.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Source Block Number (24 bits)         | Enc. Symb. ID |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
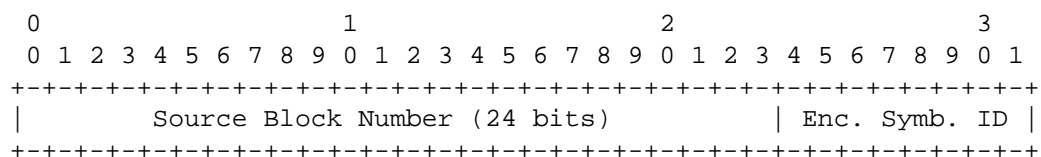
Figure 5: FEC Payload ID Encoding Format with FEC Encoding ID 5

5.2. FEC Object Transmission Information

5.2.1. Mandatory Elements

   o FEC Encoding ID: the Fully-Specified FEC Scheme described in this
      section uses FEC Encoding ID 5.

5.2.2. Common Elements

   The Common elements are the same as those specified in Section 4.2.2
   when m = 8 and G = 1.

5.2.3. Scheme-Specific Elements

   No Scheme-Specific elements are defined by this FEC scheme.

5.2.4. Encoding Format

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

5.2.4.1. Using the General EXT_FTI Format

   The FEC OTI binary format is the following, when the EXT_FTI
   mechanism is used (e.g., within the ALC [ALC] or NORM [NORM]
   protocols).

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |   HET = 64    |    HEL = 3    |                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
   |                       Transfer Length (L)                    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |   Encoding Symbol Length (E)  | MaxBlkLen (B) |     max_n     |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
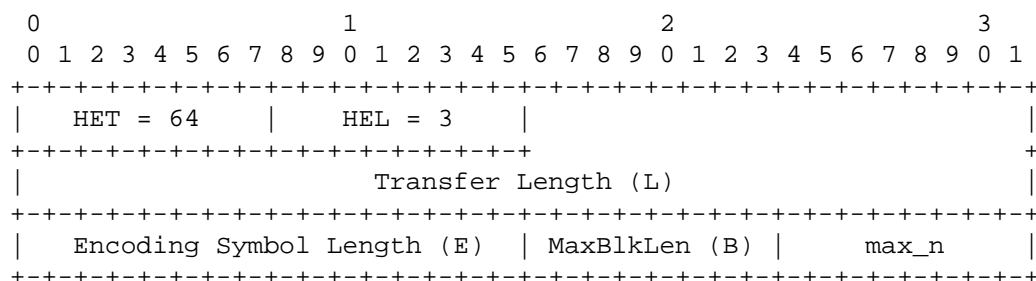
          Figure 6: EXT_FTI Header Format with FEC Encoding ID 5

5.2.4.2. Using the FDT Instance (FLUTE specific)

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

   o FEC-OTI-FEC-Encoding-ID

   o FEC-OTI-Transfer-Length (L)

   o FEC-OTI-Encoding-Symbol-Length (E)

   o FEC-OTI-Maximum-Source-Block-Length (B)

   o FEC-OTI-Max-Number-of-Encoding-Symbols (max_n)

6. Procedures with FEC Encoding IDs 2 and 5

   This section defines procedures that are common to FEC Encoding IDs 2
   and 5. In case of FEC Encoding ID 5, m = 8 and G = 1. The block
   partitioning algorithm that is defined in Section 9.1 of [RFC5052]
   MUST be used with FEC Encoding IDs 2 and 5.

6.1. Determining the Maximum Source Block Length (B)

   The finite field size parameter, m, defines the number of non-zero
   elements in this field, which is equal to: $q - 1 = 2^m - 1$. Note
   that $q - 1$ is also the theoretical maximum number of encoding symbols
   that can be produced for a source block. For instance, when m = 8
   (default) there is a maximum of $2^8 - 1 = 255$ encoding symbols.

   Given the target FEC code rate (e.g., provided by the user when
   starting a FLUTE sending application), the sender calculates:

      max1_B = floor(($2^m - 1$) * CR)

   This max1_B value leaves enough room for the sender to produce the
   desired number of parity symbols.

   Additionally, a codec MAY impose other limitations on the maximum
   block size. Yet it is not expected that such limits exist when using
   the default m = 8 value. This decision MUST 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.

6.2. Determining the Number of Encoding Symbols of a Block

   The following algorithm, also called "n-algorithm", explains how to
   determine the maximum number of encoding symbols generated for any
   source block (max_n) and the number of encoding symbols for a given
   block (n) as a function of the target code rate.

   AT A SENDER:

   Input:

      B: Maximum source block length, for any source block. Section 6.1
      explains how to determine its value.

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

      CR: FEC code rate, which is given by the user (e.g., when starting
      a FLUTE sending application). 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 = ceil(B / CR);

      if (max_n > 2^^m - 1), 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 block partitioning algorithm.

Output:

    n

Algorithm:

    n = floor(k * max_n / B);

It is RECOMMENDED that the "n-algorithm" be used by a sender, but
other algorithms remain possible to determine max_n and/or n.

At a receiver, the max_n value is extracted from the received FEC
OTI. Since the Reed-Solomon decoder does not need to know the actual
n value, using the receiver part of the "n-algorithm" is not
necessary from a decoding point of view.

However, a receiver may want to have an estimate of n for other
reasons (e.g., for memory management purposes). In that case, a
receiver knows that the number of encoding symbols of a block cannot
exceed max_n. Additionally, if a receiver believes that a sender
uses the "n-algorithm", this receiver MAY use the receiver part of
the "n-algorithm" to get a better estimate of n. When this is the
case, a receiver MUST be prepared to handle symbols with an Encoding
Symbol ID superior or equal to the computed n value (e.g., it can
choose to simply drop them).

7. Small Block Systematic FEC Scheme (FEC Encoding ID 129) and Reed-
    Solomon Codes over GF(2^^8)

In the context of the Under-Specified Small Block Systematic FEC
Scheme (FEC Encoding ID 129) [RFC5445], this document assigns the FEC
Instance ID 0 to the special case of Reed-Solomon codes over GF(2^^8)
and no encoding symbol group.

The FEC Instance ID 0 uses the Formats and Codes specified in
[RFC5445].

The FEC scheme with FEC Instance ID 0 MAY use the block partitioning
algorithm defined in Section 9.1 of [RFC5052] to partition the object
into source blocks. This FEC scheme MAY also use another algorithm.
For instance, the CDP sender may change the length of each source
block dynamically, depending on some external criteria (e.g., to
adjust the FEC coding rate to the current loss rate experienced by
NORM receivers) and inform the CDP receivers of the current block
length by means of the EXT_FTI mechanism. This choice is out of the
scope of the current document.

8. Reed-Solomon Codes Specification for the Erasure Channel

   Reed-Solomon (RS) codes are linear block codes. They also belong to
   the class of MDS codes. 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 upper bounded by q - 1. The
   implementation described in this document is based on a generator
   matrix built from a Vandermonde matrix put into systematic form.

   Sections 8.1 to 8.3 specify the [n,k]-RS codes when applied to m-bit
   elements, and Section 8.4 specifies the use of [n,k]-RS codes when
   applied to symbols composed of several m-bit elements. The use
   described in Section 8.4 is the crux of this specification.

   A reader who wants to understand the underlying theory is invited to
   refer to references [Rizzo97] and [MWS77].

8.1. Finite Field

   A finite field GF(q) is defined as a finite set of q elements that
   has a structure of field. It contains necessarily $q = p^{\wedge\wedge}m$ elements,
   where p is a prime number. With packet erasure channels, p is always
   set to 2. The elements of the field $GF(2^{\wedge\wedge}m)$ can be represented by
   polynomials with binary coefficients (i.e., over GF(2)) of degree
   lower or equal to m-1. The polynomials can be associated with binary
   vectors of length m. For example, the vector (11001) represents the
   polynomial $1 + x + x^{\wedge\wedge}4$. 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
   polynomial over GF(2) of degree m. Note that all the roots of this
   polynomial are in $GF(2^{\wedge\wedge}m)$ but not in GF(2).

   The chosen polynomial representation of the finite field $GF(2^{\wedge\wedge}m)$ is
   completely characterized by the irreducible polynomial. The
   following polynomials are chosen to represent the field $GF(2^{\wedge\wedge}m)$, for
   m varying from 2 to 16:

      m = 2, "111" $(1+x+x^{\wedge\wedge}2)$

      m = 3, "1101", $(1+x+x^{\wedge\wedge}3)$

      m = 4, "11001", $(1+x+x^{\wedge\wedge}4)$

      m = 5, "101001", $(1+x^{\wedge\wedge}2+x^{\wedge\wedge}5)$

      m = 6, "1100001", $(1+x+x^{\wedge\wedge}6)$

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)

In order to facilitate the implementation, these polynomials are also
primitive. This means that any element of GF(2^^m) can be expressed
as a power of a given root of this polynomial. These polynomials are
also chosen so that they contain the minimum number of monomials.

8.2. Reed-Solomon Encoding Algorithm

8.2.1. Encoding Principles

Let s = (s_0, ..., s_{k-1}) be a source vector of k elements over
GF(2^^m). Let e = (e_0, ..., e_{n-1}) be the corresponding encoding
vector of n elements over GF(2^^m). Being a linear code, encoding is
performed 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
RS code.

Let us consider that n = 2^^m - 1 and that 0 < k <= n. Let us denote
by alpha the root of the primitive polynomial of degree m chosen in
the list of Section 8.1 for the corresponding value of m. Let us
consider a Vandermonde matrix of k rows and n columns, denoted by
V_{k,n}, and built as follows: the {i, j} entry of V_{k,n} is v_{i,j}
= alpha^^(i*j), where 0 <= i <= k - 1 and 0 <= j <= n - 1. This
matrix generates a MDS code. However, this MDS code is not
systematic, which is a problem for many networking applications. To

obtain a systematic matrix (and code), the simplest solution consists
in considering 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, meaning that the first k encoding
elements are equal to source elements. Besides, the associated code
keeps the MDS property.

Therefore, the generator matrix of the code considered in this
document is:

   $$GM = (V_{k,k}^{-1}) * V_{k,n}$$

Note that, in practice, the [n,k]-RS code can be shortened to a
[n',k]-RS code, where k <= n' < n, by considering the sub-matrix
formed by the n' first columns of GM.

8.2.2. Encoding Complexity

Encoding can be performed by first pre-computing GM and by
multiplying the source vector (k elements) by GM (k rows and n
columns). The complexity of the pre-computation of the generator
matrix can be estimated as the complexity of the multiplication of
the inverse of a Vandermonde matrix by n-k vectors (i.e., the last
n-k columns of $V_{k,n}$). Since the complexity of the inverse of a
k*k-Vandermonde matrix by a vector is $O(k * (log(k))^2)$, the
generator matrix can be computed in $O((n-k)* k * (log(k))^2)$
operations. When the generator matrix is pre-computed, the encoding
needs k operations per repair element (vector-matrix multiplication).

Encoding can also be performed by first computing the product s *
$V_{k,k}^{-1}$ and then by multiplying the result with $V_{k,n}$. The
multiplication by the inverse of a square Vandermonde matrix is known
as the interpolation problem and its complexity is $O(k *
(log(k))^2)$. 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 [GO94]. The total
complexity of this encoding algorithm is then $O((k/(n-k)) *
(log(k))^2 + log(k))$ operations per repair element.

8.3. Reed-Solomon Decoding Algorithm

8.3.1. Decoding Principles

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

[MWS77]). 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 elements. 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
elements 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.

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

8.4. Implementation for the Packet Erasure Channel

In a packet erasure channel, each packet (including its symbol(s),
since packets contain G >= 1 symbols) is either correctly received or
erased. The location of the erased symbols in the sequence of
symbols MUST be known. The following specification describes the use
of Reed-Solomon codes for generating redundant symbols from the k
source symbols and for recovering the source symbols from any set of
k received symbols.

The k source symbols of a source block are assumed to be composed of
S m-bit elements. Each m-bit element corresponds to an element of
the finite field $GF(2^m)$ through the polynomial representation
(Section 8.1). If some of the source symbols contain less than S
elements, they MUST be virtually padded with zero elements (this can
be the case for the last symbol of the last block of the object).
However, this padding does not need to be actually sent with the data
to the receivers.

The encoding process produces n encoding symbols of size S m-bit
elements, of which k are source symbols (this is a systematic code)
and n-k are repair symbols (Figure 7). The m-bit elements of the
repair symbols are calculated using the corresponding m-bit elements
of the source symbol set. A logical u-th source vector, comprised of
the u-th elements from the set of source symbols, is used to
calculate a u-th encoding vector. This u-th encoding vector then
provides the u-th elements for the set encoding symbols calculated
for the block. As a systematic code, the first k encoding symbols
are the same as the k source symbols, and the last n-k repair symbols
are the result of the Reed-Solomon encoding.

```
         Input:  k source symbols

    0              u                              S-1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |           |X|                                 | source symbol 0
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |           |X|                                 | source symbol 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                 . . .
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |           |X|                                 | source symbol k-1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

               *

        +--------------------+
        |  generator matrix  |
        |         GM         |
        |       (k x n)      |
        +--------------------+

               |
               V

        Output: n encoding symbols (source + repair)

    0              u                               S-1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |           |X|                                 | enc. symbol 0
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |           |X|                                 | enc. symbol 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
               . . .
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |           |Y|                                 | enc. symbol n-1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
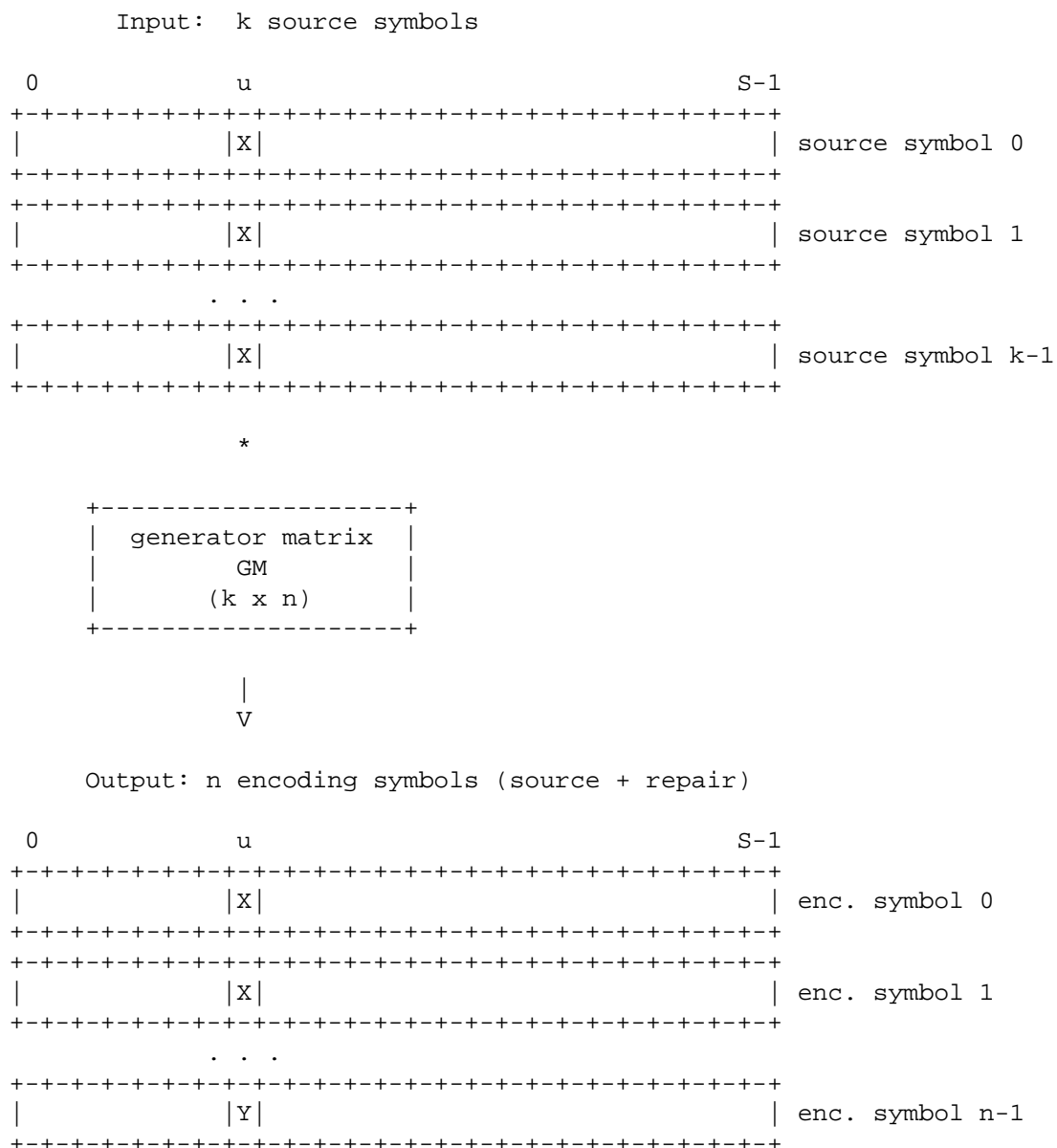
                    Figure 7: Packet Encoding Scheme

   An asset of this scheme is that the loss of some encoding symbols
   produces the same erasure pattern for each of the S encoding vectors.
   It follows that the matrix inversion must be done only once and will
   be used by all the S encoding vectors. For large S, this matrix
   inversion cost becomes negligible in front of the S vector-matrix
   multiplications.

   Another asset is that the n-k repair symbols can be produced on
   demand. For instance, a sender can start by producing a limited
   number of repair symbols and later on, depending on the observed
   erasures on the channel, decide to produce additional repair symbols,
   up to the n-k upper limit. Indeed, to produce the repair symbol e_j,
   where k <= j < n, it is sufficient to multiply the S source vectors
   with column j of GM.

9. Security Considerations

9.1. Problem Statement

   A content delivery system is potentially subject to many attacks:
   some of them target the network (e.g., to compromise the routing
   infrastructure, by compromising the congestion control component),
   others target the Content Delivery Protocol (CDP) (e.g., to
   compromise its normal behavior), and finally some attacks target the
   content itself. Since this document focuses on a FEC building block
   independently of any particular CDP (even if ALC and NORM are two
   natural candidates), this section only discusses the additional
   threats that an arbitrary CDP may be exposed to when using this
   building block.

   More specifically, several kinds of attacks exist:

   o those that are meant to give access to confidential content (e.g.,
      in case of non-free content),

   o those that try to corrupt the object being transmitted (e.g., to
      inject malicious code within an object or to prevent a receiver
      from using an object),

   o and those that try to compromise the receiver's behavior (e.g., by
      making the decoding of an object computationally expensive).

   These attacks can be launched either against the data flow itself
   (e.g., by sending forged symbols) or against the FEC parameters that
   are sent either in-band (e.g., in an EXT_FTI or FDT Instance) or out-
   of-band (e.g., in a session description).

9.2. Attacks against the Data Flow

   First of all, let us consider the attacks against the data flow.

9.2.1. Access to Confidential Objects

   Access control to the object being transmitted is typically provided
   by means of encryption. This encryption can be done over the whole
   object (e.g., by the content provider, before the FEC encoding
   process), or be done on a packet per-packet basis (e.g., when IPsec
   Encapsulating Security Payload (ESP) is used [RFC4303]). If access
   control is a concern, it is RECOMMENDED that one of these solutions
   be used. Even if we mention these attacks here, they are not related
   nor facilitated by the use of FEC.

9.2.2. Content Corruption

   Protection against corruptions (e.g., after sending forged packets)
   is achieved by means of a content integrity verification/sender
   authentication scheme. This service can be provided at the object
   level, but in that case a receiver has no way to identify which
   symbol(s) are corrupted if the object is detected as corrupted. This
   service can also be provided at the packet level. In this case,
   after removing all forged packets, the object may be recovered
   sometimes. Several techniques can provide this source
   authentication/content integrity service:

   o At the object level, the object MAY be digitally signed (with
      public key cryptography), for instance by using RSASSA-PKCS1-v1_5
      [RFC3447]. This signature enables a receiver to check the object
      integrity, once the object has been fully decoded. Even if
      digital signatures are computationally expensive, this calculation
      occurs only once per object, which is usually acceptable.

   o At the packet level, each packet can be digitally signed. A major
      limitation is the high computational and transmission overheads
      that this solution requires (unless Elliptic Curve Cryptography
      (ECC) is used). To avoid this problem, the signature may span a
      set of symbols (instead of a single one) in order to amortize the
      signature calculation. But if a single symbol is missing, the
      integrity of the whole set cannot be checked.

   o At the packet level, a Group Message Authentication Code (MAC)
      [RFC2104] scheme can be used; for instance, by using HMAC-SHA-256
      with a secret key shared by all the group members (i.e., the
      sender(s) and receivers). Thanks to the secret key, this
      technique creates a cryptographically secured digest of a packet
      that is sent along with the packet. The Group MAC scheme does not

create prohibitive processing load nor transmission overhead, but
it has a major limitation: it only provides a group
authentication/integrity service since all group members share the
same secret group key, which means that each member can send a
forged packet. It is therefore restricted to situations where
group members are fully trusted (or in association with another
technique as a pre-check).

o At the packet level, TESLA [RFC4082] is a very attractive and
   efficient solution that is robust to losses, provides a true
   authentication/integrity service, and does not create any
   prohibitive processing load or transmission overhead. Yet
   checking a packet requires a small delay (a second or more) after
   its reception.

Techniques relying on public key cryptography (digital signatures and
TESLA during the bootstrap process, when used) require that public
keys be securely associated to the entities. This can be achieved by
a Public Key Infrastructure (PKI), or by a PGP Web of Trust, or by
pre-distributing the public keys of each group member.

Techniques relying on symmetric key cryptography (group MAC) require
that a secret key be shared by all group members. This can be
achieved by means of a group key management protocol, or simply by
pre-distributing the secret key (but this manual solution has many
limitations).

It is up to the developer and deployer, who know the security
requirements and features of the target application area, to define
which solution is the most appropriate. Nonetheless, in case there
is any concern of the threat of object corruption, it is RECOMMENDED
that at least one of these techniques be used.

9.3. Attacks against the FEC Parameters

Let us now consider attacks against the FEC parameters (or FEC OTI).
The FEC OTI can either be sent in-band (i.e., in an EXT_FTI or in an
FDT Instance containing FEC OTI for the object) or out-of-band (e.g.,
in a session description). Attacks on these FEC parameters can
prevent the decoding of the associated object: for instance,
modifying the B parameter will lead to a different block partitioning
at a receiver thereby compromising decoding; or setting the m
parameter to 16 instead of 8 with FEC Encoding ID 2 will increase the
processing load while compromising decoding.

It is therefore RECOMMENDED that security measures be taken to
guarantee the FEC OTI integrity. To that purpose, the packets
carrying the FEC parameters sent in-band in an EXT_FTI header

extension SHOULD be protected by one of the per-packet techniques
described above: digital signature, group MAC, or TESLA. When FEC
OTI is contained in an FDT Instance, this FDT Instance object SHOULD
be protected, for instance, by digitally signing it with XML digital
signatures [RFC3275]. Finally, when FEC OTI is sent out-of-band
(e.g., in a session description), this FEC OTI SHOULD be protected,
for instance, by digitally signing the object that includes this FEC
OTI.

The same considerations concerning the key management aspects apply
here also.

10. 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 [RFC5052].

This document assigns the Fully-Specified FEC Encoding ID 2 under the
"ietf:rmt:fec:encoding" name-space to "Reed-Solomon Codes over
GF(2^^m)".

This document assigns the Fully-Specified FEC Encoding ID 5 under the
"ietf:rmt:fec:encoding" name-space to "Reed-Solomon Codes over
GF(2^^8)".

This document assigns the FEC Instance ID 0 scoped by the Under-
Specified FEC Encoding ID 129 to "Reed-Solomon Codes over GF(2^^8)".
More specifically, under the "ietf:rmt:fec:encoding:instance" sub-
name-space that is scoped by the "ietf:rmt:fec:encoding" called
"Small Block Systematic FEC Codes", this document assigns FEC
Instance ID 0 to "Reed-Solomon Codes over GF(2^^8)".

11. Acknowledgments

The authors want to thank Brian Adamson, Igor Slepchin, Stephen Kent,
Francis Dupont, Elwyn Davies, Magnus Westerlund, and Alfred Hoenes
for their valuable comments. The authors also want to thank Luigi
Rizzo for his comments and for the design of the reference Reed-
Solomon codec.

12. References

12.1. Normative References

   [RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
               Requirement Levels", BCP 14, RFC 2119, March 1997.

   [RFC5052]   Watson, M., Luby, M., and L. Vicisano, "Forward Error
               Correction (FEC) Building Block", RFC 5052, August 2007.

   [RFC5445]   Watson, M., "Basic Forward Error Correction (FEC)
               Schemes", RFC 5445, March 2009.

12.2. Informative References

   [RFC3453]   Luby, M., Vicisano, L., Gemmell, J., Rizzo, L., Handley,
               M., and J. Crowcroft, "The Use of Forward Error
               Correction (FEC) in Reliable Multicast", RFC 3453,
               December 2002.

   [RS-codec] Rizzo, L., "Reed-Solomon FEC codec", available at
               http://info.iet.unipi.it/~luigi/vdm98/vdm980702.tgz and
               mirrored at http://planete-bcast.inrialpes.fr/, revised
               version of July 1998.

   [Rizzo97]   Rizzo, L., "Effective Erasure Codes for Reliable Computer
               Communication Protocols", ACM SIGCOMM Computer
               Communication Review Vol.27, No.2, pp.24-36, April 1997.

   [MWS77]   Mac Williams, F. and N. Sloane, "The Theory of Error
               Correcting Codes", North Holland, 1977.

   [GO94]   Gohberg, I. and V. Olshevsky, "Fast algorithms with
               preprocessing for matrix-vector multiplication problems",
               Journal of Complexity, pp. 411-427, vol. 10, 1994.

   [RFC5170]   Roca, V., Neumann, C., and D. Furodet, "Low Density
               Parity Check (LDPC) Forward Error Correction", RFC 5170,
               June 2008.

   [RFC5053]   Luby, M., Shokrollahi, A., Watson, M., and T.
               Stockhammer, "Raptor Forward Error Correction Scheme",
               RFC 5053, October 2007.

   [ALC]   Luby, M., Watson, M., and L. Vicisano, "Asynchronous
               Layered Coding (ALC) Protocol Instantiation", Work
               in Progress, November 2008.

   [NORM]   Adamson, B., Bormann, C., Handley, M., and J. Macker,
                 "NACK-Oriented Reliable Multicast Protocol", Work
                 in Progress, March 2009.

   [FLUTE]   Paila, T., Walsh, R., Luby, M., Lehtonen, R., and V.
                 Roca, "FLUTE - File Delivery over Unidirectional
                 Transport", Work in Progress, September 2008.

   [RFC3447]   Jonsson, J. and B. Kaliski, "Public-Key Cryptography
                 Standards (PKCS) #1: RSA Cryptography Specifications
                 Version 2.1", RFC 3447, February 2003.

   [RFC4303]   Kent, S., "IP Encapsulating Security Payload (ESP)",
                 RFC 4303, December 2005.

   [RFC2104]   "HMAC: Keyed-Hashing for Message Authentication",
                 RFC 2104, February 1997.

   [RFC4082]   "Timed Efficient Stream Loss-Tolerant Authentication
                 (TESLA): Multicast Source Authentication Transform
                 Introduction", RFC 4082, June 2005.

   [RFC3275]   Eastlake 3rd, D., Reagle, J., and D. Solo, "(Extensible
                 Markup Language) XML-Signature Syntax and Processing",
                 RFC 3275, March 2002.

Authors' Addresses

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

   EMail: jerome.lacan@isae.fr
   URI: http://pagespro.isae.fr/jerome-lacan/

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

   EMail: vincent.roca@inria.fr
   URI: http://planete.inrialpes.fr/people/roca/

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

   EMail: jani.peltotalo@tut.fi
   URI: http://mad.cs.tut.fi/

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

   EMail: sami.peltotalo@tut.fi
   URI: http://mad.cs.tut.fi/