Low Density Parity Check (LDPC) Forward Error Correction (draft-ietf-rmt-bb-fec-ldpc-01.txt)

Vincent Roca, Christoph Neumann, David Furodet

RMT
Internet-Draft
Expires: April 13, 2006

V. Roca
C. Neumann
INRIA
D. Furodet
STMicroelectronics

October 10, 2005

Low Density Parity Check (LDPC) Forward Error Correction
draft-ietf-rmt-bb-fec-ldpc-00.txt

Status of this Memo

Copyright Notice

Abstract

This document describes two Fully-Specified FEC Schemes, LDPC-Staircase and LDPC-Triangle, and their application to the reliable delivery of objects on packet erasure channels. These systematic FEC codes belong to the well known class of ``Low Density Parity Check'' (LDPC) codes, and are large block FEC codes in these sense of RFC3453.

Table of Contents

1. Introduction

RFC 3453 [RFC3453] introduces large block FEC codes as an alternative to small block FEC codes like Reed-Solomon. The main advantage of such large block codes is the possibility to operate efficiently on source blocks of size several tens of thousands (or more) source symbols. The present document introduces the Fully-Specified FEC Encoding ID XX that is intended to be used with the "Low Density Parity Check" (LDPC) Staircase FEC codes, and the Fully-Specified FEC Encoding ID YY that is intended to be used with the "Low Density Parity Check" (LDPC)-Triangle FEC codes [Roca04][Mac03]. Both schemes belong the broad class of large block codes.

-- editor's note: This document makes use of the FEC Encoding ID values XX and YY that will be specified after IANA assignment --

LDPC codes rely on a dedicated matrix, called a "Parity Check Matrix", at the encoding and decoding ends. The parity check matrix defines relationships (or constraints) between the various encoding symbols (i.e. source symbols and repair symbols), that are later used by the decoder to reconstruct the original k source symbols if some of them are missing. These codes are systematic, in the sense that the encoding symbols include the source symbols in addition to the redundant symbols.

Since the encoder and decoder must operate on the same parity check matrix, some information must be communicated between them, as part of the FEC Object Transmission information.

A publicly available reference implementation of these codes is available and distributed under a GNU/LGPL license [LDPCrefimpl]. To the best of our knowledge, there is no patent or patent application identified as being used in the LDPC-Staircase and LDPC-Triangle FEC schemes.

2. Requirements notation

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 [RFC2119].

3. Definitions, Notations and Abbreviations

3.1 Definitions

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

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 source block length in symbols, i.e. the number of source symbols of a source block

n denotes the encoding block length, i.e. the number of encoding symbols generated for a source block

E denotes the encoding symbol length in bytes

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

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

max_n Maximum Number of Encoding Symbols generated for any source block

srand(s) denotes the initialization function of the pseudo-random number generator, where s is the seed (s > 0)

rand(m) denotes a pseudo-random number generator, that returns a new random integer in [0; m-1] each time it is called

3.3 Abbreviations

This document uses the following abbreviations:

ESI Encoding Symbol ID

FEC OTI FEC Object Transmission Information

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 (12 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^^12 blocks per object.

The Encoding Symbol ID (20 bit field) identifies which specific encoding symbol generated from the source block is carried in the packet payload. There is a maximum of 2^^20 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 en 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 thanks to a dedicated function, as explained in Section 5.5

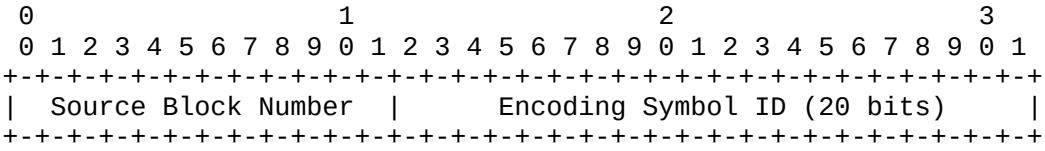


Figure 1: FEC Payload ID encoding format for FEC Encoding ID XX and YY

4.2 FEC Object Transmission Information

4.2.1 Mandatory Elements

- FEC Encoding ID: the Fully-Specified FEC Schemes described in this document use the FEC Encoding ID XX for LDPC-Staircase and FEC Encoding ID YY for LDPC-Triangle.

4.2.2 Common Elements

The following elements MUST be used 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:

  maximum transfer length = 2^^12 * B * E

  For instance, if B=2^^19 (because of a code rate of 1/2, Section 5.2), and if E=1024 bytes, then the maximum transfer length is 2^^41 bytes.

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

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

- 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

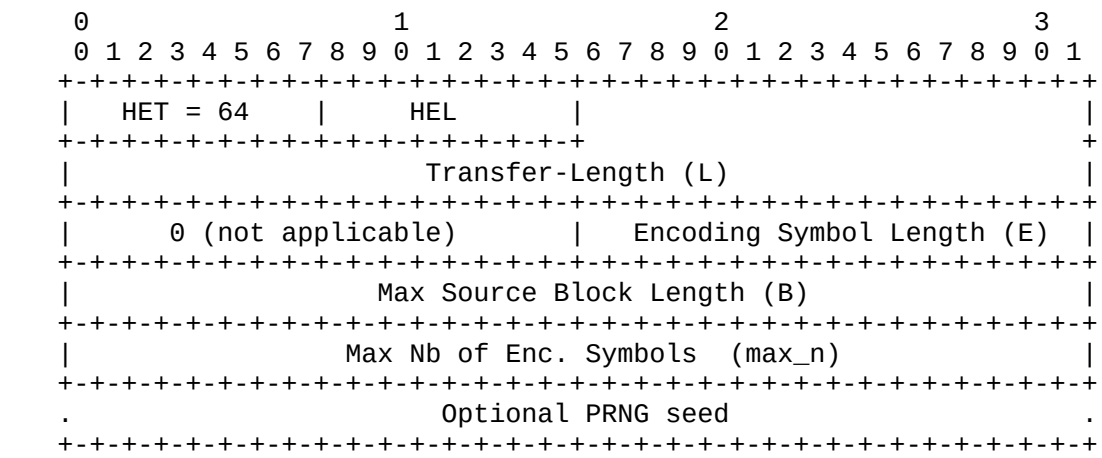
- PRNG seed: The seed is a 32 bit value used to initialize the Pseudo Random Number Generator (defined in Section 5.6). This element is optional. Whether or not it is present in the FEC OTI will be signaled in the associated encoding format through an appropriate mechanism (see Section 4.2.4). When the PRNG seed is not carried within the FEC OTI, it is assumed that encoder and decoders use another way to communicate the information, or use a fixed, predefined value.

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.



The HEL (Header Extension Length) indicates whether the optional PRNG seed is present or not.

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-PRNG-seed (optional)

When no PRNG seed is to be carried in the FEC OTI, the sender simply omits the FEC-OTI-PRNG-seed element.

5. Procedures

This section defines procedures that are common to FEC Encoding IDs XX and YY.

5.1 General

The B (maximum source block length in symbols) and E (encoding symbol length in bytes) parameters are first determined, as explained in the following sections.

The source object is then partitioned using the block partitioning algorithm specified in [fec-bb-revised]. To that purpose, the B, L (object transfer length in bytes), and E arguments are provided. As a result, the object is partitioned into N source blocks. These blocks are numbered consecutively from 0 to N-1. The first I source blocks consist of A_large source symbols, the remaining N-I source blocks consist of A_small source symbols. Each source symbol is E bytes in length, except perhaps the last symbol which may be shorter.

For each block the actual number of encoding symbols is determined, as explained in the following section.

Then, FEC encoding and decoding can be done block per block, independently. To that purpose, a parity check matrix is created, that forms a system of linear equations between the repair and source symbols of a given block, where the basic operator is XOR.

This parity check matrix is logically divided into two parts: the left side (from column 0 to k-1) which describes the occurrence of each source symbol in the equation system; and the right side (from column k to n-1) which describes the occurrence of each repair symbol in the equation system. An entry (a "1") in the matrix at position (i,j) (i.e. at row i and column j) means that the symbol with ESI i appears in equation j of the system. The only difference between the LDPC-Staircase and LDPC-Triangle schemes is the construction of the right sub-matrix.

The following sections detail how the B, E, and n parameters are determined (respectively Section 5.2, Section 5.3 and Section 5.4), how encoding symbol groups are created (Section 5.5), and finally specify the PRNG (Section 5.6).

5.2 Determining the Maximum Source Block Length (B)

The B parameter (maximum source block length in symbols) depends on several parameters: the code rate (rate), the Encoding Symbol ID field length of the FEC Payload ID (20 bits), as well as possible

internal codec limitations.

The B parameter cannot be larger than the following values, derived from the FEC Payload ID limitations, for a given code rate:

```
max1_B = 2 ^^ (20 - ceil(Log2(1/rate)))
```

Some common max1_B values are:

- rate == 1 (no repair symbols): max_B = 2 ^^ 20 = 1,048,576
- 1 > rate >= 1/2: max1_B = 2 ^^ 19 = 524,288 symbols
- 1/2 > rate >= 1/4: max1_B = 2 ^^ 18 = 262,144 symbols
- 1/4 > rate >= 1/8: max1_B = 2 ^^ 17 = 131,072 symbols

Additionally, a codec MAY impose other limitations on the maximum block size. This is the case for instance when the codec uses internally 16 bit integers to store the Encoding Symbol ID, since it does not enable to store all the possible values of a 20 bit field. Other limitations (e.g. available working memory) may also apply. 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.

5.3 Determining the Encoding Symbol Length (E)

-- editor's note: the E parameter is a function of the object transfer length. Since LDPC codes are known to offer better protection for large blocks, the smaller the object, the smaller E should be in order to increase the number of symbols the object is composed of. The optimal values that should be used for E as a function of the object transfer length are under study. --

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

5.4 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 5.2 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:

a. max_n = floor(B / R)

b. 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:

a. n = floor(k * max_n / B)

5.5 Identifying the Symbols of an Encoding Symbol Group

When multiple encoding symbols are sent in the same packet, the FEC Payload ID information of the packet MUST refer to the first encoding symbol. It MUST then be possible to identify each symbol from this single FEC Payload ID. To that purpose, the symbols of an Encoding Symbol Group (i.e. packet):

- MUST all be either source symbols, or repair symbols. Therefore only source packets and repair packets are permitted, not mixed ones.
- are identified by a function, ESIs_of_group(), that takes as argument:
  - for a sender, the index of the Encoding Symbol Group (i.e. packet) that the application wants to create,
  - for a receiver, the ESI information contained in the FEC Payload ID.

  and returns the list of G Encoding Symbol IDs that will be packed together. In case of a source packet, the G source symbols are taken consecutively. In case of a repair packet, the G repair symbols are chosen randomly, as explained below.

The system must first be initialized by creating a random permutation of the n-k indexes. This initialization function MUST be called immediately after creating the parity check matrix. More precisely, since the PRNG seed is not re-initialized, no call to the PRNG function must have happened between the time the parity check matrix has been initialized and the time the following initialization function is called. This is true both at a sender and at a receiver.

```
/*
 * Use only in case G > 1, i.e. when encoding symbol
 * groups are actually needed.
 */
initialize_tables ()
{
    int i;
    int randInd;

    /* initialize the two tables that map ID
     * (i.e. ESI-k) to/from TxSequence:
     *    - IDtoTxseq
     *    - txseqToID
     */
    for (i = 0; i < n - k; i++) {
        IDtoTxseq[i]=i;
        txseqToID[i]=i;
    }
    /* now randomize everything */
    for (i = 0; i < n - k; i++) {
        randInd = rand(n - k);
        backup = IDtoTxseq[i];
        IDtoTxseq[i] = IDtoTxseq[randInd];
        IDtoTxseq[randInd] = backup;
        txseqToID[IDtoTxseq[i]] =  i;
        txseqToID[IDtoTxseq[randInd]] = randInd;
    }
    return;
}
```

It is then possible, at the sender, to determine the sequence of G Encoding Symbol IDs that will be part of the group.

```
   /*
    * Use only in case G > 1, i.e. when encoding symbol
    * groups are actually needed.
    * PktIdx (IN):  index of the packet, in {0..ceil(T/G)} range
    * ESIs[] (OUT): list of ESI of the packet
    */
   ESIs_of_group (int      PktIdx,
                  ESI_t    ESIs[])
   {
       int i;

       if (is_source_packet(PktIdx) == true) {
           /* this is a source packet */
           ESIs[0] = (PktIdx * G) % k;
           for (i = 0; i < G; i++) {
                   ESIs[i] = ESIs[0] + i;
           }
       } else {
           /* this is a repair packet */
           for (i = 0; i < G; i++) {
               ESIs[i] =
                   k +
                   txseqToID[(i + (PktIdx - nbSourcePkts) * G)
                             % (n - k)];
           }
       }
       return;
   }
```

Similarly, upon receiving an Encoding Symbol Group (i.e. packet), a receiver can determine the sequence of G Encoding Symbol IDs from the first ESI, esi0, that is contained in the FEC Payload ID.

```
/*
 * Use only in case G > 1, i.e. when encoding symbol
 * groups are actually needed.
 * esi0 (IN):  : ESI contained in the FEC Payload ID
 * ESIs[] (OUT): list of ESI of the packet
 */
ESIs_of_group (ESI_t    esi0,
               ESI_t    ESIs[])
{
    int i;

    if (is_source_packet(esi0) == true) {
        /* this is a source packet */
        for (i = 0; i < G; i++) {
            ESIs[i] = (esi0 + i) % k;
        }
    } else {
        /* this is a repair packet */
        for (i = 0; i < G; i++) {
            ESIs[i] =
                k +
                txseqToID[(i + IDtoTxseq[esi0 - k])
                          % (n - k)];
        }
    }
}
```

5.6 Pseudo Random Number Generator

The present FEC Encoding ID relies on a pseudo-random number generator (PRNG) that must be fully specified, in particular in order to enable the receivers and the senders to build the same parity check matrix. The minimal standard generator [Park88] is used. It defines a simple multiplicative congruential algorithm: Ij+1 = A * Ij (modulo M), with the following choices: A = 7^^5 = 16807 and M = 2^^31 - 1 = 2147483647. The PRNG must be initialized with a seed that is strictly greater than 0.

```
double seed; /* assumed initialized with a seed > 0 */
#define A 16807.0
#define M 2147483647.0
#define Q 127773.0 /* M div A */
#define R 2836.0 /* M mod A */

/*
 * Initialize the PRNG with a seed > 0.
 */
void srand (int s)
{
   if (s > 0) seed = s;
   else exit(-1);
}

/*
 * Returns a random integer in [0; maxv-1]
 * derived from [Park88].
 */
int rand (int maxv)
{
   double lo, hi, test;
   double rand;

   hi = (int)(seed / Q);
   lo = seed - Q*hi;
   test = A*lo - R*hi;
   if (test > 0.0)
      seed = test;
   else
      seed = test + M;
   rand = seed / M;
   if (rand == 1.0)
      return 0;
   else
      return ((int)(rand * (double)maxv));
}
```

6. Full Specification of the LDPC-Staircase Scheme

6.1 General

LDPC-Staircase is identified by the Fully-Specified FEC Encoding ID XX.

LDPC-Staircase is based on the pseudo-random number generator specified in Section 5.6. Therefore the seed used to initiate the PRNG is an instance-specific FEC Object Transmission Information optional element. When the PRNG seed is not carried within the FEC OTI, it is assumed that encoder and decoders use another way to communicate the information, or use a fixed, predefined value.

6.2 Parity Check Matrix Creation

The matrix creation algorithm for LDPC-Staircase is described in the following. The algorithm can be divided into two parts: The left side of the matrix where the occurrence of the source symbols in the equations is described, and the right side of the matrix where repair symbols are described. The left side is generated with the following algorithm:

```
/* initialize a list of possible choices to
 * guarantee a homogeneous "1" distribution */
for (h = 3*k-1; h >= 0; h--) {
    u[h] = h % (n-k);
}
/* left limit within the list of possible choices, u[] */
t = 0;

for (j = 0; j < k; j++) { /* for each source symbol column */
    for (h = 0; h < 3; h++) { /* add 3 "1s" */
        /* check that valid available choices remain */
        for (i = t; i < 3*k && matrix_has_entry(u[i],j); i++);

        if (i < 3*k) {
            /* choose one index within the
             * list of possible choices */
            do {
                i = t + rand(3*k-t);
            } while (matrix_has_entry(u[i],j));
            matrix_insert_entry(u[i],j);

            /* replace with u[t] which has never been chosen */
            u[i] = u[t];
            t++;
        } else {
            /* no choice left, choose one randomly */
            do {
                i = rand(n-k);
            } while (matrix_has_entry(i,j));
            matrix_insert_entry(i,j);
        }
    }
}

/* Add extra bits to avoid rows with less than two checks. */
for (i = 0; i < n-k; i++) { /* for each row */
    if (degree_of_row(i) == 0) {
        j = rand(k);
        e = matrix_insert_entry(i,j);
    }
    if (degree_of_row(i) == 1) {
        do {
            j = rand(k);
        } while (matrix_has_entry(i,j));
        matrix_insert_entry(i,j);
    }
}
```

The right side (the staircase) is generated with the following algorithm:

```
for (i = 0; i < n-k; i++) { /* for each row */
    matrix_insert_entry(i,k+i);
    if (i > 0)
        matrix_insert_entry(i,k+i-1);
}
```

Note that just after creating this parity check matrix, when encoding symbol groups are used, the function initializing the two random permutation tables (Section 5.5) MUST be called. This is true both at a sender and at a receiver.

6.3 Encoding

Thanks to the staircase matrix, repair symbol creation is straightforward: each repair symbol is equal to the sum of all source symbols in the associated equation, plus the previous repair packet. Therefore encoding MUST follow the natural repair symbol order, i.e. generate repair symbol with ESI i before symbol ESI i+1 and MUST start with the first repair symbol.

6.4 Decoding

Decoding basically consists in solving a system of n-k linear equations whose variables are the source an repair symbols. Of course, the final goal is to recover the value of source symbols only.

To that purpose, many techniques are possible. One of them is the following trivial algorithm: Given a set of linear equations, if one of them has only one remaining unknown variable, then the value of this variable is that of the constant term. So, replace this variable by its value in all the remaining linear equations and reiterate. The value of several variables can therefore be found recursively. Applied to LDPC FEC codes working over an erasure packet, the parity check matrix defines a set of linear equations whose variables are the source symbols and repair symbols. Receiving or decoding a symbol is equivalent to having the value of a variable. Appendix A sketches a possible implementation of this algorithm.

The pivot of Gauss technique, as well as derived versions, is another possibility.

Because interoperability does not depend on the decoding algorithm used, the current document does not recommand any particular technique. This choice is left to the codec implementer.

7. Full Specification of the LDPC-Triangle Scheme

7.1 General

LDPC-Triangle is identified by the Fully-Specified FEC Encoding ID YY.

LDPC-Triangle is based on the pseudo-random number generator specified in Section 5.6. Therefore the seed used to initiate the PRNG is an instance-specific FEC Object Transmission Information optional element. When the PRNG seed is not carried within the FEC OTI, it is assumed that encoder and decoders use another way to communicate the information, or use a fixed, predefined value.

7.2 Parity Check Matrix Creation

The matrix creation algorithm for LDPC-Triangle is the following. The left side is the same as for LDPC-Staircase (see Section 6.2). The right side (the triangle) is generated with the following algorithm:

```
for (i = 0; i < n-k; i++) { /* for each row */
    /* create the identity */
    matrix_insert_entry(i,k+i);
    if (i > 0) {
        /* create the staircase */
        matrix_insert_entry(i,k+i-1);

        /* fill the triangle */
        int j = i;
        for (l = 0; l < j; l++) {
            if (j != 0) {
                temp = rand(j);
                matrix_insert_entry(pchkMatrix, i, k+j);
            }
        }
    }
}
```

Note that just after creating this parity check matrix, when encoding symbol groups are used, the function initializing the two random permutation tables (Section 5.5) MUST be called. This is true both at a sender and at a receiver.

7.3 Encoding

Just like LDPC-Triangle repair symbol creation is straightforward: each repair symbol is equal to the sum of all source symbols in the

associated equation, plus the (previously built) repair packets specified in the triangle. Therefore encoding MUST follow the natural repair symbol order, i.e. generate repair symbol with ESI i before symbol ESI i+1 and MUST start with the first repair symbol.

7.4 Decoding

Decoding basically consists in solving a system of n-k linear equations, whose variables are the source an repair symbols. Of course, the final goal is to recover the value of source symbols only. To that purpose, many techniques are possible, as explained in Section 6.4.

Because interoperability does not depend on the decoding algorithm used, the current document does not recommand any particular technique. This choice is left to the codec implementer.

8. Security Considerations

The security considerations for this document are the same as they are for RFC 3452 [RFC3452].

9. Intellectual Property

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

10. Acknowledgments

Section 5.4 is derived from a previous Internet-Draft, and we would like to thank S. Peltotalo and J. Peltotalo for their contribution.

We would also like to thank Pascal Moniot from STMicroelectronics for his comments.

11. References

11.1 Normative References

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

[RFC3452] Luby, M., Vicisano, L., Gemmell, J., Rizzo, L., Handley, M., and J. Crowcroft, "Forward Error Correction (FEC) Building Block", RFC 3452, December 2002.

[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.

[fec-bb-revised] Watson, M., Luby, M., and L. Vicisano, "Forward Error Correction (FEC) Building Block (revised)", draft-ietf-rmt-fec-bb-revised-01.txt draft-ietf-rmt-fec-bb-revised-01.txt, September 2005.

11.2 Informative References

[LDPCrefimpl] Roca, V., Neumann, C., and J. Laboure, "LDPC-Staircase/ LDPC-Triangle Codec Reference Implementation", MCLv3 project PLANETE Research Team, INRIA Rhone-Alpes, June 2005.

[Mac03] MacKay, D., "Information Theory, Inference and Learning Algorithms", Cambridge University Press, ISBN: 0521642981, 2003.

[Park88] Park, S. and K. Miller, "Random Number Generators: Good Ones are Hard to Find", Communications of the ACM Vol 31, No 10, pp.1192-1201, 1988.

[Roca04] Roca, V. and C. Neumann, "Design, Evaluation and Comparison of Four Large Block FEC Codecs: LDPC, LDGM, LDGM-Staircase and LDGM-Triangle, Plus a Reed-Solomon Small Block FEC Codec", INRIA Research Report RR-5225, June 2004.

Authors' Addresses

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

Phone:
Email: vincent.roca@inrialpes.fr
URI:

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

Phone:
Email: christoph.neumann@inrialpes.fr
URI:

David Furodet
STMicroelectronics
12, Rue Jules Horowitz
BP217
Grenoble Cedex 38019
France

Phone:
Email: david.furodet@st.com
URI:

Appendix A. Trivial Decoding Algoritm (Informative Only)

A trivial decoding algorithm is the following:

Initialization: allocate a partial sum buffer, partial_sum_i, for each line i, and reset it to 0.

For each newly received or decoded symbol s_i with ESI i:

1. If s_i is an already decoded or received symbol, return immediately and do nothing.

2. If s_i is a source symbol, it is permanently stored in memory.

3. For each equation j having a degree greater than one (i.e. more than one unknown variable), with an entry in column i (i.e. having s_i as a variable), do the following:

   + add s_i to partial_sum_i;

   + remove the entry (j, i) of the H matrix.

   + If the new degree of equation j is one, we have decoded a new packet and have to remember the index of the equation in a list of indexes for newly decoded packets for step 4.

4. For all newly generated packets s_l in step 3:

   + remove the last entry in equation j,

   + copy partial_sum_j to the buffer associate with symbol s_l,

   + goto step 1 with the newly created symbol s_l

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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