Internet Engineering Task Force
INTERNET-DRAFT
draft-luby-rmt-bb-fec-raptor-object-00.txt

RMT WG
M. Luby
Digital Fountain
A. Shokrollahi
EPFL
19 October 2004
Expires: April 2005

Raptor Forward Error Correction (FEC) Schemes for Objects

Status of this Document

This document is a product of the IETF RMT WG. Comments should be addressed to the authors, or the WG's mailing list at rmt@lbl.gov.

Abstract

This document describes the Raptor code and its application to reliable delivery of objects. Two Fully-Specified Forward Error Correction (FEC) schemes are introduced, one for a non-systematic version of Raptor and one for a systematic version of Raptor, that supplements the FEC schemes described in RFC 3452.

Raptor is a fountain code, i.e., as many encoding symbols as needed can be generated by the encoder on-the-fly from the source symbols of a source block. The decoder is able to recover the source block from any set of encoding symbols only slightly more in number than the number of source symbols.

Table of Contents

1. Introduction

This document describes the Raptor code and its application to reliable delivery of objects. Two Fully-Specified Forward Error Correction (FEC) schemes are introduced, one for a non-systematic version of Raptor and one for a systematic version of Raptor, that supplements the FEC schemes described in RFC 3452.

We first provide a simple and easy to implement description of a non-systematic Raptor encoder and decoder and then describe how to apply this version to reliable delivery of objects. We then describe how to modify the non-systematic Raptor code to make it systematic, and then describe how to apply the systematic Raptor codes to reliable delivery of objects. Thus, we introduce two new Fully-Specified FEC Schemes for reliable object delivery, one for the non-systematic Raptor code and one for the systematic Raptor code.

Raptor is a fountain code (as for example defined in [10]), i.e., as many encoding symbols as needed can be generated by the encoder on-the-fly from the source symbols of a source block. The decoder is able to recover the source block from any set of encoding symbols only slightly more in number than the number of source symbols. This fountain property holds for both the non-systematic and the systematic versions of Raptor. There are many advantages to using fountain codes versus other types of FEC codes, some of which are described in [9] and [10].

This document inherits the context, language, declarations and restrictions of the FEC building block [4]. This document also uses some of the terminology of the companion document [14] which describes the use of FEC codes within the context of reliable IP multicast transport and provides an introduction to some commonly used FEC codes.

Building blocks are defined in RFC 3048 [7]. This document is a product of the IETF RMT WG and follows the general guidelines provided in RFC 3269 [3].

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 [2].

1.1. Notation

We use the following notation hereafter. For a positive value x let floor(x) be x rounded down to the nearest integer and let ceil(x) be x rounded up to the nearest integer.

For positive integers i and j let i MOD j denote i modulo j. For example, 11 MOD 3 = 2.

For positive integers i and j let i^j denote i raised to the power j. For example, 2^16 = 65,536.

Note that 65,521 is the largest prime smaller than 2^16.

For equal-length bit strings X and Y let X XOR Y denote the bit-by-bit exclusive-or of X and Y. For example, 1010 XOR 1100 = 0110.

1.2. Gray sequences

We use a Gray sequence in the description of the generation of the half symbols in the pre-coding step, as described below. For all positive integers i, let g[i] be defined as follows. Let b[i] be the highest order bit that is different in the binary representation of i-1 and i. Then, the binary representation of g[i] is obtained by flipping bit b[i] of g[i-1]. Table 1.2 provides some example values for g[i]. Note that the sequence defined by g[.] has the property that each pair of consecutive elements in the sequence differ in exactly one bit position.

i	g[i]
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

Table 1.2: Gray sequence g[.]

We also use the function cnt[i], where cnt[i] returns the number of bits that are set to one in the binary representation of i. For example the

binary representation of 25 is 11001, and thus cnt[25] = 3.

For any fixed positive integer j let g[.,j] be the subsequence of g[.] where for each element in the sequence exactly j bits are set to 1. Thus, g[.,j] can be defined based on g[.] as follows.

* c = 0, i = 0
* Do forever
    - While (cnt(g[c]) not= j) do c = c+1
    - g[i,j] = g[c]
    - c = c+1
    - i = i+1

For example, g[0,2] = g[2], g[1,2] = g[4], g[2,2] = g[6], g[3,2] = g[8], g[4,2] = g[12], etc. Note that g[.,j] has the property that each pair of consecutive elements in the sequence differ in exactly two bit positions. For all i = 1, let Pos1[i,j] be one of the bit positions in which g[i,j] and g[i-1,j] differ and let Pos2[i,j] be the other bit position in which they differ. For example, Pos1[2,2] = 0 and Pos2[2,2] = 1.

1.3. Work

The complexity of encoding and decoding is measured in terms of the number of bytes of symbols that are exclusive-ored together or copied, which is defined to be the work. Thus, for example, if a symbol is 64 bytes long, then computing the exclusive-or of two symbols counts as 64 bytes of work, and copying a symbol from one location to another also counts as 64 bytes of work. The total encoding and decoding times depend also on the amount of bookkeeping operations that are needed to

determine which symbols are exclusive-ored together or copied. But since the symbols are typically relatively long, and since when there are multiple source blocks the bookkeeping operations are done only once and can be amortized over all the source blocks, the exclusive-or and copy operations of symbols provide a rough estimate of the relative time it takes to encode and decode on different CPU/OS platforms. Furthermore, the efficiency of the bookkeeping operations are implementation dependent, and in an efficient implementation the bookkeeping operations take a fraction of the time that the exclusive-or

and copy operations take.

2. Raptor Encoding of a Source Block

2.1. Overview

Symbols are the fundamental data units of the encoding and decoding process, and for each source block all symbols are the same size, typically a few bytes long. The atomic operation performed on symbols for both encoding and decoding is the exclusive-or operation.

A pre-coding step is used to produce L-K redundant symbols from the K source symbols, where L > K, and the combination of the K source symbols and the L-K redundant symbols form the L pre-coding symbols. Section 2.2 **describes how the pre-coding symbols are generated from the source** symbols.

Each encoding packet contains a Encoding Symbol ID (ESI) and encoding symbols. The ESI is used to generate a (d,a,b)-triple for each encoding symbol carried in the encoding packet using the generators described in Section 2.3. Then, each (d,a,b)-triple is used to generate the corresponding encoding symbol from the pre-coding symbols as described in Section 2.3.3.

2.2. Pre-coding

The pre-coding step consists of generating redundant symbols from the K source symbols as follows. The redundant symbols consist of S LDPC symbols and H half symbols. Let X be the smallest positive integer such that X*(X-1) = 2*K. The value of S is the smallest positive prime integer that is at least ceil(0.01*K) + X. The value of H is the smallest integer such that choose(H,H') >= K + S, where H' = ceil(H/2) and where choose(i,j) denotes the number of ways j objects can be chosen from among i objects without repetition. Then L = K+S+H. Let the positions of the pre-coding symbols range from 0 to L-1, where the first K are the source symbols, the next S are the LDPC symbols, and the final H are the half symbols.

For i = 0,...,L-1 let C[i] denote the ith pre-coding symbol. Note that C[0],..., C[K-1] are the original source symbols. Initialize all the redundant symbols C[K],...,C[L-1] to all zeroes.

The S LDPC symbols are defined as follows.

* For i = 0,...,K-1 do
    - a = 1 + (floor(i/S) MOD (S-1))
    - b = i MOD S
    - C[K + b] = C[K + b] XOR C[i]
    - b = (b + a) MOD S
    - C[K + b] = C[K + b] XOR C[i]
    - b = (b + a) MOD S
    - C[K + b] = C[K + b] XOR C[i]

The H half symbols are defined as follows.

* For h = 0,...,H-1 do
    - For i = 0,...,K+S-1 do
        - If bit h of g[i,H'] is equal to 1 then C[h+K+S] = C[h+K+S] XOR C[i].

Equivalently, the H half symbols can be defined as follows, which suggests an efficient implementation:

* T = C[0].
* For i = 1,...,K+S-1 do

    o C[Pos1[i,H']+K+S] = C[Pos1[i,H']+K+S] XOR T.

    o C[Pos2[i,H']+K+S] = C[Pos2[i,H']+K+S] XOR T.

    o T = T XOR C[i].

* For all bit positions h of g[K+S-1,H'] that are equal to 1 do

    o C[h+K+S] = C[h+K+S] XOR T.

2.3. Generators

All of the generators described in the following subsections are used in the generation of encoding symbols.

2.3.1. Random Generator

The random number generator Rand[X, i, m] is defined as follows, where X is a 2-byte unsigned integer, i is an unsigned integer and m is a positive integer and the value produced is an integer between 0 and m-1. Let X0 be first byte and let X1 the second byte of X. Let V0 and V1 be arrays of 256 entries each, where each entry is a random 4-byte unsigned integer. Then,

```
Rand[X, i, m] = (V0[(X0 + i) MOD 256] XOR V1[(X1 + i) MOD 256]) MOD m
```

2.3.2. Degree Generator

The degree generator Deg[v] is defined as follows, where v is an integer that is at least 0 and less than 2^20 = 1,048,576.

* In Table 2.3.2, find the index j such that f[j-1] <= v < f[j]

* Deg[v] = d[j]

j	f[j]	d[j]
0	0	--
1	10,241	1
2	491,582	2
3	712,794	3
4	831,695	4
5	948,446	10

6	**1,032,189**	11
7	**1,048,576**	40

Table 2.3.2: Defines the degree distribution for encoding symbols

2.3.3. Encoding Symbol Generator

Let L be the number of pre-coding symbols of the source block, let L' be the smallest prime integer greater than or equal to L, and let C[0],..., C[L-1] be the pre-coding symbols of the source block. The encoding symbol generator Enc[d, a, b] is defined as follows, where d is a degree, a is between 1 and L'-1 and b is between 0 and L'-1.

* While (b = L) do b = (b + a) MOD L'

* Enc[d, a, b] = C[b].

* For j = 1,...,d-1 do

    o b = (b + a) MOD L'

    o While (b = L) do b = (b + a) MOD L'

    o Enc[d, a, b] = Enc[d, a, b] XOR C[b]

2.4. Encoding work

The work to produce the LDPC symbols is 3 times the total length in bytes of the source symbols. The work to produce the half symbols is essentially 3 times the total length in bytes of the source symbols. Thus, the total work for generating the pre-coding is 6 times the total length in bytes of the source symbols.

>From the degree distribution described in Section 2.3.2 via Table 2.3.2, it is not hard to see that the work on average to generate encoding symbols is 4.63 times the total length in bytes of the encoding symbols generated.

3. Raptor Decoding of a Source Block

3.1. Overview

This subsection provides an overview of Raptor decoding of a source block. It is assumed that the decoder knows the structure of the source block it is to decode, including the symbol length and the number K of symbols in the source block.

>From the algorithms described in 2.2, the Raptor decoder can calculate the total number L = K+S+H of pre-coding symbols and determine how they were generated from the source block to be decoded. It is assumed that the received encoding symbols for the source block to be decoded are passed to the decoder. Furthermore, for each such encoding symbol it is assumed that [d,a,b]-triple that was used to compute the encoding symbol from the pre-coding symbols is passed to the decoder, and this allows the decoder to use a portion of the encoding algorithm described in Section 2.3.3 to compute the number and set of pre-coding symbols used to generate the encoding symbol.

Let N >= K be the number of received encoding symbols for a source block and let M = S+H+N. The following M by L bit matrix A can be derived from the information passed to the decoder for the source block to be decoded. Let C be the column vector of the L pre-coding symbols, and let D be the column vector of M symbols with values known to the receiver, where the first S+H of the M symbols are zero-valued symbols that correspond to LDPC and half symbols (these are check symbols for the LDPC and half symbols, and not the LDPC and half symbols themselves), and the remaining N of the M symbols are the received encoding symbols for the source block. Then, A is the bit matrix that satisfies A*C = D, where here * denotes matrix multiplication over GF[2]. In particular, A[i,j] = 1 if the pre-coding symbol corresponding to index j is exclusive-or'd into the LDPC, half or encoding symbol corresponding to index i in the encoding, or if index i corresponds to a LDPC or half symbol and index j corresponds to the same LDPC or half symbol. For all other i and j, A[i,j] = 0.

Decoding a source block is equivalent to decoding C from known A and D. (This is equivalent to recovering the K source symbols since if they can be recovered then the other L-K pre-coding symbols can be recomputed.) It is clear that C can be decoded if and only if the rank of A over GF[2] is L.

The first step in decoding C is to form a decoding schedule. In this step A is converted, using Gaussian elimination (using row operations and row and column reorderings) and after discarding M - L rows, into the L by L identity matrix. The decoding schedule consists of the sequence of row operations and row and column re-orderings during the Gaussian elimination process, and only depends on A and not on D. The decoding of C from D can take place concurrently with the forming of the decoding schedule, or the decoding can take place afterwards based on the decoding schedule.

The correspondence between the decoding schedule and the decoding of C is as follows. Let c[0] = 0, c[1] = 1,...,c[L-1] = L-1 and d[0] = 0, d[1] = 1,...,d[M-1] = M-1 initially.

* Each time row i of A is exclusive-ored into row i' in the decoding schedule then in the decoding process symbol D[d[i]] is exclusive-ored into symbol D[d[i']].

* Each time row i is exchanged with row i' in the decoding schedule then in the decoding process the value of d[i] is exchanged with the value of d[i'].

* Each time column j is exchanged with column j' in the decoding schedule then in the decoding process the value of c[j] is exchanged with the value of c[j'].

>From this correspondence it is clear that the total number of exclusive-ors of symbols in the decoding of the source block is the number of row operations (not exchanges) in the Gaussian elimination. Since A is the L by L identity matrix after the Gaussian elimination and after discarding the last M - L rows, it is clear at the end of successful decoding that the L symbols D[d[0]], D[d[1]],..., D[d[L-1]] are the values of the L symbols C[c[0]], C[c[1]],..., C[c[L-1]].

The order in which Gaussian elimination is performed to form the decoding schedule has no bearing on whether or not the decoding is successful. However, the speed of the decoding depends heavily on the order in which Gaussian elimination is performed. (Furthermore, maintaining a sparse representation of A is crucial, although this document does not describe the details of how this is done). The remainder of this section focuses on the order in which Gaussian elimination should be performed.

3.2. First Phase

The first phase of the Gaussian elimination the matrix A is conceptually partitioned into submatrices. The submatrix sizes are parameterized by non-negative integers i and u which are initialized to 0. The submatrices of A are:

(1) The submatrix I defined by the intersection of the first i rows and first i columns. This is the identity matrix at the end of

each step in the phase.

(2) The submatrix defined by the intersection of the first i rows and all but the first i columns and last u columns. All entries of this submatrix are zero.

(3) The submatrix defined by the intersection of the first i columns and all but the first i rows. All entries of this submatrix are zero.

(4) The submatrix U defined by the intersection of all the rows and the last u columns.

(5) The submatrix X formed by the intersection of all but the first i columns and the last u columns and all but the first i rows.

Figure 3.2 illustrates the submatrices of A. At the beginning of the first phase X = A. In each step, a row of A is chosen. The following graph defined by the structure of X is used in determining which row of A is chosen. The columns that intersect X are the nodes in the graph, and the rows that have exactly 2 ones in X are the edges of the graph that connect the two columns (nodes) in the positions of the two ones. A component in this graph is a maximal set of nodes (columns) and edges (rows) such that there is a path between each pair of nodes/edges in the graph. The size of a component is the number of nodes (columns) in the component.

Figure 3.2: Submatrices of A in the first phase

There are at most L steps in the first phase. The phase ends successfully when i + u = L, i.e. when X and the all zeroes submatrix above X have disappeared and A consists of I, the all zeroes submatrix below I, and U. The phase ends unsuccessfully in decoding failure if at some step before X disappears there is no non-zero row in X to choose in that step. In each step, a row of A is chosen as follows:

Row Choice

* If all entries of X are zero then no row is chosen and decoding fails.

* Let r be the minimum integer such that at least one row of A has exactly r ones in X.

* If r not= 2 then choose a row with exactly r ones in X with minimum original degree among all such rows.

* If r = 2 then choose any row with exactly 2 ones in X that is part of a maximum size component in the graph defined by X.

After the row is chosen in the step the first row of A that intersects X is exchanged with the chosen row so that the chosen row is the first row that intersects X. The columns of A among those that intersect X are reordered so that one of the r ones in the chosen row appears in the first column of X and so that the remaining r-1 ones appear in the last columns of X. Then, the chosen row is exclusive-ored into all the other rows of A below the chosen row that have a one in the first column of X. Finally, i is incremented by 1 and u is incremented by r-1, which completes the step.

3.3. Second Phase

The submatrix U is further partitioned into the first i rows, UU, and the remaining M - i rows, UL. Gaussian elimination is performed in the second phase on UL to either determine that its rank is less than u (decoding failure) or to convert it into a matrix where the first u rows is the identity matrix (success of the second phase). Call this u by u identity matrix UI. The M - L rows of A that intersect UL - UI are discarded. After this phase A has L rows and L columns.

3.4. Third Phase

After the second phase the only portion of A which needs to be zeroed out to finish converting A into the L by L identity matrix is UU. The number of rows i of the submatrix UU is generally much larger than the number of columns u of UU. To zero out UU efficiently, the following precomputation matrix UE is computed based on UI in the third phase and then UE is used in the fourth phase to zero out UU. The u rows of UI are partitioned into ceil(u/7) groups of 7 rows each. Then, for each group of 7 rows all non-zero combinations of the 7 rows are computed, resulting in 2^7- 1 = 127 rows (this can be done with 2^7-7-1 = 120 exclusive-ors of rows per group, since the combinations of Hamming weight one that appear in UI do not need to be recomputed). Thus, the resulting precomputation matrix UE has ceil(u/7)*127 rows and u columns. Note that UE is not formally a part of matrix A, but will be used in the fourth phase to zero out UU.

3.5. Fourth Phase

For each of the first i rows of A, for each group of 7 columns in the UU submatrix of this row, if the set of 7 column entries in UU are not all zero then the row of the precomputation matrix UE that matches the

pattern in the 7 columns is exclusive-ored into the row, thus zeroing out those 7 columns in the row at the cost of exclusive-oring one row of UE into the row.

After this phase A is the L by L identity matrix and a complete decoding schedule has been successfully formed. Then, as explained at the beginning of Section 3.1, the corresponding decoding consisting of exclusive-oring known encoding symbols can be executed to recover the source block based on the decoding schedule.

Only rows corresponding to recovering a source symbol need be considered in this phase if only the source symbols and not all the pre-coding symbols are to be decoded. However, for the systematic Raptor codes described in Section 5 all of the pre-coding symbols need be recovered.

4. Raptor Object Delivery

This section specifies how to use the Raptor code specified in Sections **2 and 3 for reliable delivery of an object to many receivers in an IP** multicast network. This could also be used for delivery using other types of networks, e.g., unicast networks, but this is outside the scope of this document. This version of Raptor is a non-systematic code.

This section also defines a new Fully-Specified FEC scheme (as defined in RFC3452 [4]). with a corresponding new FEC Encoding ID (with an as yet undefined number) and the corresponding FEC Object Transmission Information and FEC Payload ID format.

4.1. Motivation

The basic properties of Raptor for reliable object delivery are that, for any packet loss conditions, for delivery of objects of any relevant size: (a) reception overhead of each individual receiver is minimal; (b) the total transmission time and bandwidth needed to deliver objects to any number of receivers is minimal.

In the solution described in this document the amount of working memory

needed for decoding can be much smaller than the object size and still provide the above properties, and the amount of computation needed to encode and decode is minimal.

Raptor is a fountain code (as for example defined in [10], called expandable codes in [14]), i.e., as many encoding packets as needed can be generated on-the-fly, each containing unique encoding symbols that

are equally useful for recovering a object. There are many advantages to using fountain codes versus other types of FEC codes, some of which are described in [9], [10] and [11]. One advantage is that, regardless of packet loss conditions and receiver availability, fountain codes minimize the number of encoding packets each receiver needs to receive to reconstruct a object. This is true even under harsh packet loss conditions and when for example mobile receivers are only intermittently turned-on or available over a long object delivery session.

One advantage of the fountain property of Raptor is that it makes it possible to decide during the session how many encoding packets to generate and send. This can be useful if for example there is feedback from receivers indicating whether or not they received enough encoding packets to recover a object. When packet loss conditions are less severe than expected the transmission can be terminated early. When packet loss conditions are more severe than expected or receivers are unavailable more often than expected the transmission can be seamlessly extended.

Alternatively, if a fixed duration object delivery session is used and after the conclusion of the initial session feedback is received which indicates that many receivers have not yet received enough packets to recover the object then it would be advantageous to schedule a repair session. For example, the scheduled duration of the initial session can be short, assuming optimistically small losses, and then the duration can be dynamically extended only if required. This flexibility and ability to optimize transmission bandwidth usage is simple with a fountain code.

4.2. Source blocks and sub-blocks

The maximum source block size B in bytes is recommended to be 4 MB in this document. Thus, objects that are larger than B bytes in size are partitioned into more than one source block. Limiting the source block size to at most B bytes in size ensures that the encoding length of a source block can potentially be many times larger than the source block, and thus object delivery using this specification can handle very high packet loss conditions.

The maximum block size W in bytes that can be decoded in working memory is recommended to be 256 KB in this document. Thus, source blocks that are larger than W bytes in size are partitioned into N > 1 sub-blocks,

and the Raptor decoder decodes one sub-block at a time. Each sub-block consists of the same number K of sub-symbols, where each sub-symbol is T bytes long. Then, each source symbol of the source block is T*N bytes long, and consists of the concatenation of exactly one sub-symbol from each of the N sub-blocks. Figure 4.2 shows a source block placed into a

two dimensional array, where each entry is a T-byte sub-symbol, each row is a sub-block and each column is a source symbol. The number shown in each sub-symbol entry indicates their original order within the source block. For example, the sub-symbol numbered K contains bytes T*K through T*(K+1)-1 of the source block. Thus, the first K sub-symbols of the source block numbered 0 to K-1 form the first sub-block, the next K sub-symbols of the source block numbered K to 2*K-1 form the second sub-block, etc. Then, source symbol i is the concatenation of the ith sub-symbol from each of the sub-blocks, which corresponds to the sub-symbols of the source block numbered i, K+i, 2*K+i,..., (N-1)*K+i.

```
0          1    2       ...   ...   ... ...     K-1
K          K+1  K+2     ...   ...   ... ...     2*K-1
2*K      2*K+1 2*K+2 ...   ...   ...    ...     3*K-1
(N-1)*K ...     ...     ...   ...   ... ...     N*K-1
```

Figure 4.2: Source symbols from sub-symbols structure

Encoding symbols are generated from the source symbols composed of sub-symbols,and thus the size of an encoding symbol is T*N bytes consisting of N encoding sub-symbols each of size T bytes.

4.3. Session and packet information

The overall structure of the FEC Object Transmission Information and the FEC Payload ID are both defined in [4]. The receiver needs to receive the specific FEC Object Transmission Information in a session description (for example, carried in a FLUTE FDT as described in [8]) generally before starting to receive packets for a object to determine some of the critical parameters needed to decode the object. The FEC Payload ID is carried in each packet to identify the encoding symbols carried in that packet.

4.3.1. FEC Object Transmission Information

Besides the FEC Encoding ID, the FEC Object Transmission Information includes:

* The object size F in bytes

* The payload size P of each packet in bytes

* The maximum source block size B in bytes

* The maximum working memory size W in bytes that indicates the largest sub-block that can be decoded in receiver working memory.

A suggested value of P could be for example 512 bytes for object delivery over multicast cellular networks, or 1024 bytes for other multicast networks. In general P must be equal to G*N*T, where G is defined below.

A suggested value of B is 4 MB. This means that objects that are at most 4 MB will consist of one source block, and that objects larger than **4 MB will be partitioned into more than one source block.** The method used to partition a object larger than 4 MB into source blocks is described in [8]. In general B must be at least W.

A suggested value of the maximum size W of a sub-block that can be decoded in working memory is 256 KB for example for delivery of objects to cellular devices. Other values of W could also be suitable, e.g. W = **512 KB or W = 128 KB.** How a source block is partitioned into sub-blocks depends on whether the source block size is smaller or larger than working memory W, and is described below for the suggested values of B and W.

For the suggested values the algorithms described below compute for each source block:

* The number G of encoding symbols for the source block carried in each encoding packet

* The number K of source symbols in the source block

* The number N of sub-blocks into which the source block is partitioned

* The sub-symbol size T in bytes for the sub-blocks.

The symbol size is thus N*T bytes for the source block.

4.3.2. FEC Payload ID

The FEC Payload ID is shown in Figure 4.3.2 and consists of a two-byte Source Block Number (SBN) and a two-byte Encoding Symbol ID (ESI), and

thus the overall FEC Payload ID is 4 bytes long. This allows sending of objects of size up to P*2^32 bytes. The FEC Payload ID is included in the header of each packet carrying encoding symbols in its payload to identify how the encoding symbols are generated from the source block.

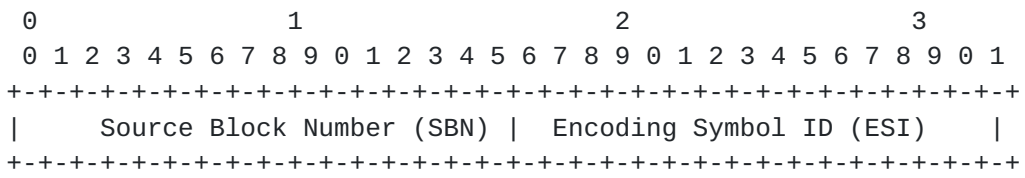

Figure 4.3.2: FEC Payload ID

A longer FEC Payload ID could be used for other services. For example with an FEC Payload ID that consists of a four-byte SBN and a four-byte ESI, much larger source blocks can be used and much larger objects can be sent. would require some minor modifications to how the ESI is used to generate encoding symbols.

4.4. Encoding an object

All the examples in the subsequent subsections used the suggested parameter settings of P = 512 bytes, W = 256 KB and B = 4 MB.

4.4.1. Smaller objects

When the object size F <= W the object consists of a single source block that in turn consists of one sub-block, i.e., Z = N = 1. The number K of source symbols in the object is computed as ceil(F/T). For P = 512 bytes and W = 256 KB, the number G of encoding symbols placed into each encoding packet and the symbol size T is determined by Table 4.4.1 based on the object size F. In general, for other values of P and W, it is recommended that G and T are set in such a way such that K is at least 1,000 and T is as large as possible.

The SBN is set to 0 in the FEC Payload ID of each packet and the ESI is

set to a two-byte value X, where the value of X should be different in each encoding packet. The ESI is used to uniquely identify the encoding symbols generated from the L pre-coding symbols of a source block that are placed into a single packet, where the value of L and the pre-coding symbols are generated as described in Section 2. The sequence of ESI values for the packets sent for the object is generated as follows.

Generating ESI values

* Let Q be the largest prime integer such that Q = floor(2^16/G)

* Let A be an integer such that 0 < A < Q

* Let B be an integer such that 0 <= B < Q

* Let N be the number of packets to be sent in the session

* X = B

* For i = 0,...,N-1 do

    o Use X as the ESI for the next packet

    o X = (X + A) MOD Q

If additional packets for the object are to be send at a later time then the ESI's for the packets sent in that session can be generated using the above algorithm using the same value for A and using the value of X at the end of the first session for the value of B. For example, suppose G = 16 (and thus Q = 4,093), A = 1 and B = 0. If 150 packets are sent in the first session then their ESI values are 0, 1, ..., 149, and at the end X = 150. Suppose 50 additional packets are to be sent in the MBMS repair session. Then A = 1 and B = 150, and the ESI values are 150, 151, ..., 199 and at the end X = 200.

The values of the encoding symbols placed into the packet with ESI X are computed as follows:

Generating encoding symbols E0[X],..., EG-1[X]

* v = Rand[X, 0, 2^20]

* Y = (X*G) MOD 2^16

* Repeat the following for i = 0,...,G-1

    o d = Deg[v]

    o a = 1 + Rand[Y, 1, L'-1]

    o b = Rand[Y, 2, L']

- Ei[X] = Enc[d, a, b]
- v = (v + 2^20/G) MOD 2^20
- Y = (Y + 1) MOD 2^16

Because of the way ESIs are generated for packets in different sessions, it is easy to see that all of the ESIs in the different sessions are all distinct and all the Y values used to generate the (d, a, b)-triple for each encoding symbol are all distinct as long as the total number of packets generated is at most Q.

F range	N	G	T	K range
512 B < F = 64 KB	1	16	32 bytes	16 = K = 2,048
64 KB < F = 128 KB	1	8	64 bytes	1,024 < K = 2,048
128 KB < F = 256 KB	1	4	128 bytes	1,024 < K = 2,048

Table 4.4.1: Source block parameters for small objects when P = 512 bytes and W = 256 KB

EXAMPLE 1
Object size F = 50 KB
Number of object packets = 100
Number of source blocks Z = 1
Number of sub-blocks N = 1
Number of encoding symbols per encoding packet G = 16
Symbol size T = 32 bytes
Number of source symbols K = 1,600

EXAMPLE 2
Object size F = 100 KB bytes
Number of object packets = 200
Number of source blocks Z = 1
Number of sub-blocks N = 1
Number of encoding symbols per encoding packet G = 8
Symbol size T = 64 bytes
Number of source symbols K = 1,600

EXAMPLE 3
Object size F = 200 KB bytes
Number of object packets = 400
Number of source blocks Z = 1
Number of sub-blocks N = 1
Number of encoding symbols per encoding packet G = 4
Symbol size T = 128 bytes
Number of source symbols K = 1,600

4.4.2. Larger objects

When the object size F is more than W but at most B then the object consists of a single source block that is partitioned into N sub-blocks, where the length of each sub-block is greater than W/2 but at most W. The value of N is the smallest power of two such that N*W >= F. The size I of each sub-block is computed as ceil(F/N), and the number K of sub-symbols per sub-block is computed as ceil(I/T). Table 4.4.2 is used to determine the sub-block parameters based on the object size F when P = 512 bytes, W = 256 KB and B = 4 MB. In general, for other values of W and B, it is recommended that G and T are powers of two and set in such a way such that K is at least 2,000 and T is as large as possible.

The SBN is set to 0 in the FEC Payload ID of each packet and the ESI is set to a two-byte value X, where the value of X should be different in each encoding packet. The source block is partitioned into N sub-blocks, and each encoding packet contains G encoding symbols generated from the source block consisting of the N sub-blocks, where the values of N and G are obtained from Table 4.4.2. The ESI values for the packets and the values of the G encoding symbols E0[X],..., EG-1[X] placed into the packet with ESI X are computed as described in Section 4.4.1.

F range	N	G	T	K range
256 KB < F = 512 KB	2	4	64 bytes	2,048 < K = 4,096
512 KB < F = 1 MB	4	2	64 bytes	2,048 < K = 4,096
1 MB < F = 2 MB	8	1	64 bytes	2,048 < K = 4,096
2 MB < F = 4 MB	16	1	32 bytes	4,096 < K = 8,192

Table 4.4.2: Source block parameters for large objects when P = 512 bytes, W = 256 KB and B = 4 MB

EXAMPLE 4
Object size F = 400 KB
Number of object packets = 800
Number of source blocks Z = 1
Number of sub-blocks N = 2
Size of a sub-block in bytes I = 200 KB
Number of encoding symbols per encoding packet G = 4
Sub-symbol size T = 64 bytes
Symbol size N*T = 128 bytes
Number of source symbols per source block K = 3,200

EXAMPLE 5
Object size F = 800 KB
Number of object packets = 1,600
Number of source blocks Z = 1
Number of sub-blocks N = 4
Size of a sub-block in bytes I = 200 KB
Number of encoding symbols per encoding packet G = 2
Sub-symbol size T = 64 bytes
Symbol size N*T = 256 bytes
Number of source symbols per source block K = 3,200

EXAMPLE 6
Object size F = 1.6 MB
Number of object packets = 3,200
Number of source blocks Z = 1
Number of sub-blocks N = 8
Size of a sub-block in bytes I = 200 KB
Number of encoding symbols per encoding packet G = 1
Sub-symbol size T = 64 bytes
Symbol size N*T = 512 bytes
Number of source symbols per source block K = 3,200

EXAMPLE 7
Object size F = 3 MB
Number of object packets = 6,144
Number of source blocks Z = 1
Number of sub-blocks N = 16
Size of a sub-block in bytes I = 192 KB
Number of encoding symbols per encoding packet G = 1
Sub-symbol size T = 32 bytes
Symbol size N*T = 512 bytes

Number of source symbols per source block K = 6,144

4.4.3. Large objects

When the object size F is more than B then the object is partitioned into more than one source block using the Algorithm for Computing Source Block Structure described in Section 5.1.2.3 of FLUTE [8]. The inputs to that algorithm are:

* The maximum number of source symbols per source block. This is set to the ratio of B/P, since there is one symbol per packet. If B = 4 MB and P = 512 bytes, the maximum number of source symbols per source block is 8,192.

* The transfer length in bytes. This is set to the object size F.

* The symbol length in bytes. This is set to P since there is one symbol per packet.

The output of the algorithm is the number Z of source blocks, and the number and lengths of source symbols in each of the Z source blocks (with possibly the last symbol of the last source block logically filled out with zeroes to a full length symbol).

Each encoding packet contains one encoding symbol generated from one of the Z source blocks. The SBN of each packet is set to the number of the source block from which the encoding symbol carried in the payload of the packet is generated, and thus the SBN is between 0 and Z-1. The ESI values for each source block are generated using the algorithm described in Section 4.4.1 independently for each source block, and thus the same ESI values are used for different source blocks of the same object. The method for generating an encoding symbol from a source block is as described in Section 4.4.2, where the source block is substituted for the object in that description.

EXAMPLE 8
Object size F = 9 MB
Number of object packets = 18,432
Number of source blocks Z = 3
Size of source blocks = 3 MB (all equal size in this example)

For each of the 3 source blocks, the rest of the construction is the same as for Example 6 where the source block is substituted for the object in that description.

4.4.4. Encoding work per object

The encoding work per object can be derived from Section 2, i.e., the work to generate the pre-coding for a object is 6*F and the work on average to generate packet payloads containing encoding symbols for a object is 4.63 times the total length in bytes of the generated packet payloads.

4.5. Decoding an object

Just like for encoding, there are three cases of how to decide how to recover the object, depending on the object size F. When F <= W then the object is decoded as a single source block. Each received encoding packet contains an ESI and one or more encoding symbols. The ESI is used to reconstruct how each encoding symbol contained in the encoding packet was generated, and then when enough encoding packets have arrived the object is decoded.

When B >= F > W then the object consists of a single source block that is partitioned into multiple sub-blocks that are decoded in a coordinated way. As each encoding packet arrives, the ESI is saved, and each encoding sub-symbol corresponding to a sub-block is saved in a temporary object dedicated to that sub-block. Then, based on the received ESIs the decoding schedule can be formed based on the algorithms described in Section 3, and then that decoding schedule is applied to the encoding sub-symbols for each sub-block in sequence, decoding each sub-block within the limits of the working memory resources based on the algorithms described in Section 3, where the sub-block is substituted for the source block and the encoding sub-symbols are substituted for the encoding symbols in that description.

When F >= B then the object consists of multiple source blocks that are in turn partitioned into multiple sub-blocks. Each source block is handled independently of the other source blocks, i.e., all the ESIs for a source block are saved in a temporary object dedicated to that source block, and each encoding sub-symbol corresponding to a sub-block within the source block are saved in a temporary object dedicated to that sub-block within the source block. Then, for each source block in turn, based on the received ESIs for that source block the decoding schedule can be formed, and then that decoding schedule is applied to the

encoding sub-symbols for each sub-block in sequence, decoding each sub-block within the limits of the working memory resources.

The decoding memory requirements for Raptor are slightly more than the sub-block length. For example, a 100 KB object that is encoded and

decoded as a single source block takes slightly more than 100 KB of working memory for the decoding, whereas a 3 MB object is partitioned into 16 sub-blocks of 192 KB each, and thus the working memory needed to decode is slightly more than 192 KB. Decoding can be performed in place, i.e., a sub-block can be recovered in place using the same working memory that contains the encoding sub-symbols for the sub-block just before it is decoded.

4.5.1. Decoding work per object

The decoding work per object depends slightly on the reception overhead, i.e., the more packets received for an object the less the total work it takes to decode the object. For all object sizes the average decoding workload per object is around 10 when the reception overhead is 0.02, and drops to around 8 when the reception overhead is 0.04, where the decoding workload is the decoding work divided by the number of bytes in the object, and thus the decoding workload is the number of bytes that are exclusive-ored on average to decode each byte of the object. The decoding work per object is independent of the amount of packet loss and packet loss patterns.

Since the sub-symbols that are exclusive-ored in the decoding generally are many bytes long (sizes ranging from 32 bytes to 128 bytes depending on the object size), and since a CPU can generally exclusive-or together several bytes in a single operation, the exclusive-oring can be pipelined in such a way that decoding is very efficient.

4.5.2. Decoding failure probability

The decoding failure probability d for a given reception overhead e is the probability the decoding process fails to recover the source block when the number of received encoding symbols is (1+e)*K, where K is the number of source symbols in the source block. For all values of K that are at least approximately 1,000 the decoding failure probability is at most around d = 10^-3 when the reception overhead is e = 0.01 and the decoding failure probability is at most around d = 10^-6 when the reception overhead is e = 0.02. The decoding failure probability continues to drop off as the reception overhead increases, but at a much slower rate.

These results were obtained by running the decoder many times on different packet sequences until several decoding failures were observed and then reporting the average number of times decoding failed. In some cases this required running the decoder tens of millions of times.

Because each encoding packet is generated at random independently of all other encoding packets, the received encoding packets are random for any loss pattern of encoding packets that does not depend on their values. Since packet loss is independent of packet content, the value of d is independent of packet loss.

There are other versions of Raptor codes with much smaller failure probabilities, but this is beyond the scope of this document.

5. Systematic Raptor

5.1. Overview

The Raptor systematic encoder is used to generate repair symbols from a source block that consists of K source symbols. The overall strategy for doing this is as follows. A first step is to generate systematic information associated with K. This systematic information is used to generate K triples (d[0], a[0], b[0]),..., (d[K-1], a[K-1], b[K-1]) for the source symbols.

The K source symbol triples are then used to decode L intermediate pre-coding symbols from the source symbols using the Raptor decoder described in Section 3. By the way the systematic information is generated, the K source symbol triples are guaranteed to uniquely decode the L intermediate pre-coding symbols, i.e., if the K source symbol triples are used to produce K encoding symbols from the intermediate pre-coding symbols using the Raptor encoder described in Section 2 then these encoding symbols are exactly the source symbols. Furthermore, the K source symbol triples are generated from the same distribution as other triples generated by the Raptor encoder, and thus the source symbols can be thought of as random encoding symbols generated from the intermediate pre-coding symbols. The idea is to then use the Raptor encoder to produce additional random triples and then use the Raptor encoder to produce other encoding symbols, called repair symbols, from the intermediate pre-coding symbols using the additional triples. The source symbols and the repair symbols are sent to receivers.

If a receiver receives all K of the source symbols then there is no need for any decoding. If there is at least one missing source symbol then

from the systematic information associated with K the receiver can generate the K source symbol triples and the additional triples corresponding to received repair symbols. The Raptor decoder can be used to recover the intermediate pre-coding symbols from any received combination of source and repair symbols that is slightly more than K symbols in total using the corresponding triples for the received

symbols, and then the Raptor encoder can be used to recover the missing source symbols from the intermediate pre-coding symbols using the triples for the missing source symbols.

5.2. Systematic information

The K triples for the source symbols are generated from the systematic information as described in Section 5.3. How the systematic information is generated is described in Section 5.6. The systematic information for each relevant value of K can either be stored or generated as needed at each receiver. The systematic information associated with a source block of K source symbols is:

* An integer T.

* Integers A and B such that 0 < A < 65,521 and 0 <= B < 65,521.

* A list of T integers 0 < M[0] < M[1] < ... < M[T-1] < K+T.

Typically, T can be represented with one byte, while all the other values can be represented with two bytes. For example, for K = 1,000, the value of T is at most 10, so that the description of the systematic information is at most 25 bytes. This representation can be further shortened, e.g., the sequence M[0], ... , M[T-1] could be stored as successive differences, and the differences typically take one byte.

5.3. Generating source symbol triples from systematic information

The K source symbol triples (d[0], a[0], b[0]), ... , (d[K-1], a[K-1], b[K-1]) are generated as follows from the systematic information (T, A, B, M[0],..., M[T-1]) associated with K. Note that L' is smallest prime that is at least the number L of intermediate pre-coding symbols.

Generate K triples for source symbols

* Set i = 0, j = 0, t = 0, X = B.

* While i < K do

    o If t < T and j = M[t] then t = t+1

- o Else
  - d[i] = Deg[Rand[X,0, 2^20]]
  - a[i] = 1 + Rand[X, 1, L'-1]
  - b[i] = Rand[X, 2, L']
  - i = i+1
- o X = (X + A) MOD 65,521
- o j = j + 1

Note that it is not necessary to store T explicitly, since it can be obtained from the other data.

5.4. Calculating the intermediate pre-coding symbols

The L intermediate pre-coding symbols are obtained by applying the Raptor decoder described in Section 3 to the source symbols C[0], C[1], ..., C[K-1] using the decoding schedule calculated from the K source symbol triples (d[0], a[0], b[0]),..., (d[K-1], a[K-1], b[K-1]). The procedure for obtaining the source symbol triples guarantees that the decoding is successful. The result of this operation is the L intermediate pre-coding symbols I[0], I[1],..., I[L-1].

5.5. Work and decoding failure probability

At the sender, the intermediate pre-coding symbols can be calculated with essentially the same work as a decoding step of the Raptor decoder described in Section 3. From the degree distribution of the Raptor encoder described in Section 2, it can be seen that the work to generate

repair symbols from intermediate pre-coding symbols is on average 4.63 times the total length in bytes of generated repair symbols.

At the receiver, there is no work to be done if all the source symbols are received. If there is at least one missing source symbol, then the average work for decoding the intermediate pre-coding symbols from received source and repair symbols is essentially the same as the average work for the Raptor decoder described in Section 3. The additional work to recreate missing source symbols from the intermediate pre-coding symbols is on average 4.63 times the length in bytes of

missing source symbols.

The decoding failure probability of the Raptor systematic decoder for a given reception overhead is the same as for Raptor decoder described in Section 3.

5.6. Calculating the systematic information

In the description below o will denote the chosen reception overhead. We recall the matrix interpretation of the Raptor decoding process as described in Section 3, and the four phases of the decoding process. In that decoding algorithm an elimination process is applied to a matrix A that is derived from the received symbols, and the pre-coding symbols. The matrix has N rows corresponding to the N received symbols, and S+H rows corresponding to the S LDPC and H half symbols. The systematic information associated with K is computed as follows:

(1) Set A = 53,591, B = 10,267

(2) Compute N = ceil((1 + o)*K)

(3) Repeat the following until the systematic information is generated:

    (a) X = B

    (b) Repeat the following for i = 0,...,N-1

        (i) Y[i] = Rand[X, 0, 2^16].

        (ii) X = (X+A) MOD 65,521

    (c) Apply the first phase of the decoding algorithm described in Section 3 until the number of rows in the matrix I corresponding to received symbols is K-10, or until the first phase is finished, whichever is first. Let Y[i1],

Y[i2], ..., Y[im] denote the Y-values corresponding to these rows.

(d) If the first phase is not yet finished, i.e., if the matrix X has columns, then replace the matrix U by the matrix obtained from X and U.

(e) In the second phase of the algorithm (elimination of the matrix U) use the rows of the matrix corresponding to the LDPC and the half-symbols as pivot rows.

(f) If this is not possible, then

(i) A = (A*10,267) MOD 65,521

(ii) B = X

(iii) Go to Step (3a).

(g) If the pivoting on the LDPC and half symbols is possible in Step (3e) then

(i) Let Y[im+1],...,Y[iK] denote the Y-values of the pivot rows of U that do not correspond to the LDPC or half symbols.

(ii) Set E be the maximum value among i1, i2,..., iK . Set T = E+1-K and compute the T indices among 0,..., E missing from i1, i2, ..., iK, and let these indices in increasing order be M[0], M[1],..., M[T-1].

(h) Return the systematic information (T, A, B, M[0], M[1],..., M[T-1]).

The systematic information associated for K can be computed for all relevant values of K once and for all and packaged with the Raptor decoder software to receivers. Alternatively, receivers may calculate the systematic information using the algorithm described above for relevant values of K on an as needed basis. As another alternative, a mixed strategy can be used, where the systematic information for some values of K is packaged with the Raptor decoder software and as systematic information for other values of K is needed the receiver computes this on the fly, and once computed potentially saved for later use. Furthermore, the K source symbol triples for a given value of K can be saved for reuse with other source blocks once they are computed.

6. Raptor Systematic Object Delivery

This section specifies how to use the Raptor systematic code specified in Section 5 for reliable delivery of an object to many receivers in an IP multicast network. This could also be used for delivery using other types of networks, e.g., unicast networks, but this is outside the scope of this document.

This section also defined a new Fully-Specified FEC scheme (as defined in RFC3452 [4] and a corresponding new FEC Encoding ID (with an as yet

undefined number) and the corresponding FEC Object Transmission Information and FEC Payload ID format.

The FEC scheme described in this section is almost identical to the FEC scheme described in Section 4. As described below, the only difference in encoding is that the sent encoding symbols for a source block consist of the source symbols of the source block and the repair symbols generated from the intermediate pre-coding symbols, whch are in turn generated from teh source symbols using the systematic information. There are analogous differences in the decoding algorithm. The overall properties of this FEC scheme are exactly the same as those for the FEC scheme defined in Section 4, except that: (a) the encoding and decoding are both slightly more complex; (b) the original object is sent as part of the encoding.

6.1. Session and packet information

The overall structure of the FEC Object Transmission Information and the FEC Payload ID are both defined in [4]. The receiver needs to receive the specific FEC Object Transmission Information in a session description (for example, carried in a FLUTE FDT as described in [8]) generally before starting to receive packets for a object to determine some of the critical parameters needed to decode the object. The FEC Payload ID is carried in each packet to identify the encoding symbols carried in that packet.

6.1.1. FEC Object Transmission Information

Besides the FEC Encoding ID, the FEC Object Transmission Information is the same as for the FEC scheme described in Section 4: the object size F in bytes; the payload size P of each packet in bytes: the maximum source block size B in bytes; the maximum working memory size W in bytes that indicates the largest sub-block that can be decoded in receiver working memory.

Some suggested values for these parameters, the relationships between them, and the way parameters G, K, N and T are computed based on them is

all the same as described in Section 4.

6.1.2. FEC Payload ID

The FEC Payload ID is the same as for the FEC scheme described in Section 4.

6.2. Encoding an object

All the examples in the subsequent subsections used the suggested parameter settings of P = 512 bytes, W = 256 KB and B = 4 MB.

6.2.1. Smaller objects

When the object size F <= W the object consists of a single source block that in turn consists of one sub-block, i.e., Z = N = 1.

Everything is computed exactly the same way as described in Section 4.4.1, except for how the encoding symbols, i.e., source and repair symbols, and ESIs are generated.

Each packet sent for the object either contains all source symbols (source packet) or all repair symbols (repair packet).

Each source packet contains G source symbols, except for perhaps the last source packet which may be either of shorter length or padded out to the same length as all other source packets if K is not a multiple of G. **The source symbols are placed into source packets sequentially,** i.e., the first source packet contains the first G source symbols, the second source packet contains the next G source symbols, etc. The ESI carried in the ith source packet is (i-1)*G, which is the index of the first source symbol carried in that packet where the indices of the source symbols are 0,...,K-1.

The K source symbol triples (d[0], a[0], b[0]),..., (d[K-1], a[K-1], b[K-1]) are generated as described in Section 5.3 from the systematic information (T, A, B, M[0],..., M[T-1]) associated with K. Then, the Raptor decoder described in Section 3 is used to recover the L intermediate pre-coding symbols from the K source symbols using the K

source symbol triples.

The ESI values placed into the R repair RTP packets are K, K+G, K+2*G, ... , K+(R-1)*G, respectively. The G repair symbol triples (d[0], a[0], b[0]),..., (d[G-1], a[G-1], b[G-1]) for the repair symbols placed into a repair RTP packet with ESI i are computed as follows, and this depends on T, A, and B, which is a portion of the systematic information associated with K.

Generate G repair symbol triples for repair packet with ESI i

* X = (B + (i+T)*A)) MOD 65,521

* v = Rand[X, 0, 2^20]

* Repeat the following for j = 0,...,G-1

    o d[j] = Deg[v]

    o a[j] = 1 + Rand[X, 1, L'-1]

    o b[j] = Rand[X, 2, L']

    o v = (v + ceil(2^20/G)) MOD 2^20

    o X = (X + A) MOD 65,521

The Raptor encoder described in Section 2 is used to generate the G repair symbols from the L intermediate pre-coding symbols using the G triples associated with the repair packet carrying the repair symbols.

6.2.2. Larger objects

When the object size F is more than W but at most B then the object consists of a single source block that is partitioned into N sub-blocks, where the length of each sub-block is greater than W/2 but at most W.

Everything is computed exactly the same way as described in Section 4.4.2, except that Section 6.2.1 is used instead of Section 4.4.1.

6.2.3. Large objects

When the object size F is more than B then the object is partitioned into more than one source block using the Algorithm for Computing Source Block Structure described in Section 5.1.2.3 of FLUTE [8]. Everything is computed exactly the same way as described in Section 4.4.3, except that Section 6.2.1 is used instead of Section 4.4.1 and Section 6.2.2 is used instead of Section 4.4.2.

6.3. Decoding an object

Just like for encoding, there are three cases of how to decide how to recover the object, depending on the object size F. When F <= W then the object is decoded as a single source block. Each received encoding packet contains an ESI and one or more encoding symbols. The ESI is used to reconstruct how each encoding symbol contained in the encoding packet was generated, and then when enough encoding packets have arrived the object is decoded.

Decoding is performed on a sub-block by sub-block basis. A decoding schedule is created based on received ESIs for a source block, and then that same decoding schedule can be used to decode all sub-blocks of the source block. Each sub-block is decoded using the algorithms described in Section 3, where the sub-block is substituted for the source block and the encoding sub-symbols are substituted for the encoding symbols in that description.

Decoding of each source block works as follows. If no source symbols are missing then no decoding is necessary. If some source symbols are missing then it is checked to see if enough repair symbols have been received to recover the missing source symbols. If so, it decodes the missing source symbols as follows:

* From the systematic information (T, A, B, M[0],..., M[T-1]) associated with K, generate the K source symbol triples using the algorithm in Section 5.3.

* Determine which source symbols have been received.

* From the ESIs in received repair packets, generate the triples for the received repair symbols using the algorithm in Section 6.2.1.

* Use the Raptor decoder described in Section 3 to recover the L intermediate pre-coding symbols from the received source symbols and repair symbols using their corresponding triples.

* Use the Raptor encoder described in Section 2 and the triples for the missing source symbols to recover the missing source symbols

from the L intermediate pre-coding symbols.

7. Security Considerations

The security considerations for this document are the same as they are for RFC 3452 [4].

8. 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 RFC 3452 [4]. This document assigns the Fully-Specified FEC Encoding ID X under the ietf:rmt:fec:encoding name-space to "Raptor object". The FEC Payload ID format and corresponding FEC Object Transmission Information associated with FEC Encoding ID X is described in Section 4.3, and the corresponding FEC scheme is described in Section 4 of this document.

This document assigns the Fully-Specified FEC Encoding ID Y under the ietf:rmt:fec:encoding name-space to "Raptor systematic object". The FEC Payload ID format and corresponding FEC Object Transmission Information associated with FEC Encoding ID Y is described in Section 6.1, and the corresponding FEC scheme is described in Section 6 of this document.

9. Intellectual Property

Digital Fountain does have intellectual property rights associated with the technology described in this document, and intends to provide a full IPR statement specific to this document to the IETF that will satisfy the requirements of the IETF.

10. Normative References

[1] Bradner, S., "The Internet Standards Process -- Revision 3", RFC 2026, October 1996.

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

[3] Kermode, R., Vicisano, L., ``Author Guidelines for Reliable Multicast Transport (RMT) Building Blocks and Protocol Instantiation documents'', RFC 3269, April 2002.

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

[5] Luby, M. and Vicisano, L., "Compact Forward Error Correction (FEC) Schemes", RFC 3695, February 2004.

[6] Mankin, A., Romanow, A., Bradner, S., Paxson V., "IETF Criteria for Evaluating Reliable Multicast Transport and Application Protocols", RFC 2357, June 1998.

[7] Whetten, B., Vicisano, L., Kermode, R., Handley, M., Floyd, S., Luby, M., "Reliable Multicast Transport Building Blocks for One-to-Many Bulk-Data Transfer", RFC 3048, January 2001.

[8] T. Paila, M. Luby, R. Lehtonen, V. Roca, R. Walsh, ''FLUTE - File Delivery over Unidirectional Transport'', IETF RMT working group, RFC 3926, October 2004

11. Informative References

[9] M. Luby. ''LT Codes'', Proceedings of The 43rd Annual IEEE Symposium on Foundations of Computer Science, November 16-19 2002, pp. 271-282.

[10] A. Shokrollahi, ''Raptor Codes'', Digital Fountain Technical Report, DF2003-06-001

[11] J. Byers, M. Luby, M. Mitzenmacher, ''A Digital Fountain Approach to Asynchronous Reliable Multicast'', IEEE J. on Selected Areas in Communications, Special Issue on Network Support for Multicast Communication, Vol. 20, No. 8, October 2002, pp. 1528 - 1540

[12] Luby, M., Gemmell, J., Vicisano, L., Rizzo, L. and J. Crowcroft, "Asynchronous Layered Coding (ALC) Protocol Instantiation", RFC 3450 December 2002.

[13] Luby, M., Gemmell, J., Vicisano, L., Rizzo, L., Handley, M. and J. Crowcroft, "Layered Coding Transport (LCT) Building Block", RFC 3451 December 2002.

[14] 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.

12. Authors' Addresses

Michael Luby
luby@digitalfountain.com
Digital Fountain, Inc.
39141 Civic Center Drive

Suite 300
Fremont, CA 94538
U.S.A.

M. Amin Shokrollahi
amin.sholrollahi@epfl.ch
Laboratory of Algorithms
Laboratory of Algorithmic Mathematics
EPFL
IC-IIF-ALGO
PSE-A
1015 Lausanne
Switzerland

13. Full Copyright Statement