Network Working Group
Internet-Draft
Intended status: Experimental
Expires: February 24, 2013

J. Zinky
A. Caro
Raytheon BBN Technologies
G. Stein
Laboratory for
Telecommunications Sciences
August 23, 2012

Random Binary FEC Scheme for Bundle Protocol
draft-zinky-dtnrg-random-binary-fec-scheme-00

Abstract

This document describes the Random Binary Forward Error Correcting (FEC) Scheme for the Erasure Coding Extension [ErasureCoding] to the DTN Bundle Protocol [RFC5050]. The Random Binary FEC scheme is a Fully-Specified FEC scheme adhering to the specification guidelines of the FEC Building Block [RFC5052]. The DTN Bundle protocol is used as the Content Delivery Protocol. This FEC scheme is one of many possible FEC schemes that may be used by the Erasure Coding Extension. The Random Binary FEC scheme has several properties that makes it efficient in the case where Data Objects are divided into hundreds to thousands of source-symbols and where the resources available of decoding are substantially greater than the resources available for encoding.

Table of Contents

1. Introduction

The Coding Layer of the Erasure Coding Extension encodes an ordered array of Chunks into an Encoding, which consisting of an Encoding Data and Encoding Vector (coding formula coefficients). The DTN Bundle protocol is used as the Content Delivery Protocol (CDP) to transfer the Encoding to the destination. When a significate number of Encodings have arrived, they are decoded and the resulting ordered array of Chunks is delivered to the Data Object layer at the destination.

For Random Binary Coding, Encoding Vectors are generated randomly and are sent with the Encoding Data as a unit in the same Bundle. This allows the encoding process and transfer overhead to be relatively efficient at the cost of an expensive decode process. Each Encoding is effectively a repair symbol and carries the same potential information about the Chunks. Any Encoding can be treated as equivalent to any other Encoding. Thus, the DTN communication channel can be extremely poor, and can reorder, delay, or drop a large percentage of the Encodings. The only important factor is the number of non-duplicate Encodings that arrive. Random Binary coding can generate a large number (exponential in N) of non-duplicate Encodings to compensate for huge drop rates, even greater than 99% drops. Also, receipt feedback does not have to acknowledge specific Encodings, but only has to summarize the state of the received Encoding Set, such as the expected number of Encodings that need to be received before all Chunks can be decoded.

The Random Binary FEC Scheme may be configured to behave like a wide variety of traditional FEC schemes by restricting which Encodings are generated. Different Encoding restrictions may be used depending on the expected conditions of the DTN and the application transfer requirements. For example, in the case where bundles are not reordered and the drop rate is low, the system could be configured to behave like a block parity FEC. Configuration options are described in Section 7.

This document only addresses the Coding Layer of the Erasure Coding Extension and follows the organization recommended in [RFC5052]. The first section introduces the Random Binary FEC definitions, notation, and formats. Following sections describe procedures used to implement the scheme and the actual process of encoding, decoding and recoding. An additional section describes how to configure Random Binary FEC to handle different DTN conditions and end user requirements. The document ends with discussions on Security and IANA considerations.

2. Terminology

The terminology used in this document follows the terminology of Erasure Coding Extension to the Bundle Protocol [ErasureCoding] and the FEC Building Block [RFC5052]. These documents have more comprehensive descriptions for the concepts used in this document.

2.1. Definitions

Data Octets is the array of octets that stores the Data Object and meta data to be transferred. The Data Octets array is treated as a whole entity and has a UUID. The Data Object Layer divides the Data Octets into an ordered array of equal length Chunks, which are considered the input and output for the Coding Layer. The Data Object Layer is responsible for padding the last Chunk and storing the Data Object length in the meta data.

Coding Formula maps Chunks onto Encoding Data. The Random Binary FEC coding formula is a linear combination of Chunks over the mathematical field GF(2).

E = Vn-1 * Cn-1 + ... + Vi * Ci + ... + V0 * C0
Where the V's are the binary coefficients of the coding formula and the C's are the Chunks. The coding formula is identified by its array of binary coefficients (V) and is stored in the Encoding Vector. The coding formula can also be concisely represented as binary dot product (see Section 2.2) between the vector of coefficients and the vector of Chunks.
E = V dot C

Hamming Weight is the number of coefficients with a nonzero value in an Encoding Vector. Encodings with low (sparse) Hamming weights need only a few XOR operations to generate the Encoding Data. Low Hamming weights are desirable for the encoding process, because they consume fewer encoder resources to generate.

Rank is a measure of independence for a set of Encodings. An Encoding Set is a group of Encodings that share the same UUID. The Encoding Vectors from the Encoding Set can be combined as the rows of a KxN binary matrix (S), where K is the number of Encodings in the Encoding Set and N is the number of Chunks. The Rank of this matrix is the number of linearly independent Encodings in the Encoding Set. When an Encoding is added to an Encoding Set, it is called Innovative, if it is not a linear combination of the already received encodings, thereby raising the rank of the matrix S. Otherwise, it is called Redundant. If an Encoding Set has rank equal to the number of Chunks (N), then the Encoding Set has full rank and can be used to solve for all

Chunks. Calculating the rank of an Encoding Set is discussed in Section 5.3

Transmission Overhead is the expected number of extra Encodings received in order to have a full rank Encoding Set. All the Encodings generated by this FEC scheme can be considered repair symbols, so when Encodings are added to an Encoding Set, some of the Encodings may be redundant. Transmission overhead is the expected number of redundant Encodings before the Encoding Set can be solved.

2.2. Notation

number_of_chunks (N or n) is the number Chunks.

chunk_length (L) is the number of octets in a Chunk. All Chunks for Data Octets with the same UUID MUST have the same length.

Chunk Index (i) range from 0 to N - 1.

Chunk (C) is an octet array of the raw data from Data Octets.

Encoding Data (E) is an octet array of the coding data.

Encoding Vector (V) is a binary array of coefficients that represents the coding formula.

Encoding Set Matrix (S) is a binary matrix that is formed from N linearly independent Encoding Vectors.

Bitwise Exclusive Or (XOR) is an operator on two octet arrays. Bitwise XOR is an expensive operation and efficient implementations are discussed in Section 5.1.

Binary Dot Product (dot) is an operator on a vector of binary coefficients and a vector of octet arrays. The result is an octet array. The Binary Dot Product XORs together the octet arrays that corresponded to nonzero values in the binary vector and ignores the octet arrays that correspond to zeros.

The following table summarizes the corresponding terms used in [RFC5052] and in [ErasureCoding].

RFC5052	DTN Erasure Coding
Source Symbol	Chunk
Repair Symbol	Encoding Data
Encoding Symbol	Chunk or Encoding Data
Encoding Symbol Length	Chunk Length
FEC Payload ID	Encoding Vector
Generator Matrix	Encoding Set Matrix

Table 1: Comparison of Terms

2.3. Abbreviations

For convenience the following Abbreviation are used in this document.

BPA: Bundle Protocol Agent, see [RFC5050]

CDP: Content Delivery Protocol, see [RFC5052]

DTN: Delay/Disruption Tolerant Network, see [RFC5050]

FEC: Forward Error Correction, see [RFC5052]

SDNV: Self-Delimiting Numeric Values, see [RFC6256]

XOR: Exclusive or (math operation)

2.4. 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. Random Binary FEC Overview

The Random Binary FEC scheme is a specific implementation of the Coding Layer that defines the encoding, decoding, and recoding process.

The Random Binary FEC scheme encodes Chunks by creating the linear combination of Chunks over the mathematical field GF(2). In GF(2), the values of the coefficients are zero and one, multiply is the AND Boolean operator, and addition is the XOR Boolean operator. A linear combination sums together the Chunks, using the XOR operator, whose coefficients are one in the Encoding Vector. In the general case, the coefficients of an Encoding Vector are selected at random, so a large number of Encodings can be generated (2^N), where N is the number of Chunks. Every Encoding can be considered a repair symbol, except for the degenerate cases where only one coefficient is nonzero. In this case, only one Chunk is XORed into the Encoding, so the Encoding acts as a source symbol.

Decoding requires collecting enough Encodings to solve the set of binary linear equations. To accomplish this, at least N innovative Encodings must be collected in order to solve the linear equations and decode all N Chunks. For Random Binary coding, decoding is an expensive operation for which there is no guarantee that the first N Encodings received will be sufficient to decode all the Chunks, i.e. it is expected that some of the Encodings received will be redundant. Decoding is unlikely to decode any chunks until roughly N distinct Encodings have arrived and will, with probability greater than 1/2, be able to decode all chunks when N + 2 distinct encodings have arrived [Stud2006]. So the arrival process is fairly abrupt with no Chunks being delivered even though many Encodings have arrived and then quickly around the time the Nth distinct Encoding arrives all the Chunks can be Decoded.

In the case where the DTN network has very short contact times relative to the transmission time of the Data Object and nodes move randomly, Encodings for the same Data Object may take radically different paths to the destination. Encodings may be Recoded by Intermediate Recoders along the path, to reduce the chance that duplicate Encodings will be delivered to the destination from alternative paths. Transmissions to multiple destinations also benefit, because the destinations do not have to receive the same Encodings, just enough non-duplicate Encodings.

4. Formats and Codes

This section maps the concepts defined in the FEC Building Blocks [RFC5052] to the Content Delivery Protocol services offered by the Erasure Coding Extension of the DTN Bundle Protocol [ErasureCoding]. This FEC Scheme fully specifies the functions of a Coding Layer for the Erasure Coding Extension. Encoding Vector and Encoding Data are the two data structures used to represent an Encoding at the Coding Layer. This section shows how they are formatted into the Erasure Coding Extension Block and Bundle Payload for delivery by the DTN Bundle Protocol. The section's organization follows the FEC Building Blocks recommendation for easy comparison with other FEC schemes.

4.1. FEC Payload ID

As defined in the [RFC5052], the FEC Payload ID contains information that indicates to the FEC decoder the relationships between the encoding symbols (Encoding Data) carried by a particular packet (Bundle) and the FEC encoding transformation. For the Random Binary FEC Scheme, the FEC Payload ID is the Encoding Vector, which holds the coefficients of the coding formula from all Chunks to the particular Encoding Data. The raw Encoding Vector is a binary array of coefficients that is the length of the number_of_chunks (N), which can be 1000's of bits long. Section 4.2.3 defines how the Encoding Vector is efficiently formated into the FEC Scheme Parameters field of the Erasure Coding Extension Block.

4.2. FEC Object Transmission Information

The FEC Object Transmission Information contains information which is essential to the decoder in order to decode the encoded object.

4.2.1. Mandatory

FEC Encoding ID is an the ID for the type of FEC Scheme. For the Erasure Coding Extension, the FEC Encoding ID also defines the format of the FEC Scheme Parameters field of the Erasure Coding Extension Block. The Random Binary Scheme has four different formats, so it has four different FEC Encoding IDs. The FEC Encoding ID is stored in the FEC Scheme Type field of the Erasure Coding Extension Block as an SDNV value. Section 4.2.3 defines the values of FEC Encoding IDs used by the Random Binary FEC Scheme.

4.2.2. Common

The following parameters are common to all FEC Schemes.

Transfer-Length is the length of the array of unencoded Data Octets to be transfered. The transfer length is not stored in a field in the bundle, but is calculated by the formula:

transfer_length = number_of_chunks * chunk_length

Encoding-Symbol-Length is the length of the octet array that can either hold an Encoding Data (repair symbol) or a Chunk (source symbol), i.e. the chunk_length. The Encoding Data is transfered as the Bundle Payload. So the Encoding-Symbol-Length is the length of the Bundle Payload and is not stored as an explicit field in the Erasure Coding Extension Block.

Maximum-Source-Block-Length The Random Binary FEC scheme does not use Blocks, so does not use Maximum Source Block Length.

Max-Number-of-Encoding-Symbols The Random Binary FEC scheme does not limit the number of Encoding Bundles that can be generated and sent by the Coding Layer. But the Regulating Layer of Erasure Coding Extension does define a Handling Specification field that COULD be used to limit the rate and amount of Redundant Encoding Bundles that are forwarded by the Intermediate Regulating Layer. Thus, effectively setting a limit on the Max-Number-of-Encoding-Symbols forwarded by Intermediate Nodes.

4.2.3. Scheme-Specific

Encoding Data (repair symbol) is the result of applying the coding formula to the Chunks (source symbols). Encoding Data is a array of octets. Encoding Data is stored in the Bundle Payload in highest octet index first order, with no padding and no trailing zero. The Bundle Payload only contains the Encoding Data octet array, no additional payload header is used by the Random Binary FEC scheme.

Encoding Vector is an array of binary coefficients of the coding formula with length equal to the number_of_chunks. The Encoding Vector is stored into the Coding Parameters Field of the EC Extension Block. Several formats are specified in this document, each of which reduces the number of bits needed to transmit the vector in certain cases. The number of bits needed depends on the contents of the vector, i.e., on the number and distribution of the ones in the vector. An Encoding Vector MAY be represented in any of the formats, but the choice of format can dramatically

reduce the length of the Coding Parameter field. Encodings for the same Object UUID MAY use different vector formats. Encoders SHOULD dynamically choose the shortest format, when constructing an Encoding Bundle. Decoders and Recoders SHALL support all formats.

4.2.3.1. **Full Binary Array:** FEC Scheme Type = 1

The most general Encoding Vector format is to send all the binary coefficients as an array of octets. The Encoding Vector binary coefficients are packed 8 coefficients to an octet. The lowest octet index and bit index is 0. If the number of binary coefficients is not a multiple of 8, padding bits are added to the highest indicies using the value of zero. The resulting octet array is sent with the highest index first.

Field	Type	Description
Packed Binary Array	Octets	All binary coefficients of the Encoding Vector packed into an octet array.

Table 2: Full Vector

The bit array has the following implicit parameters:

octet_array_length = ceiling(number_of_chunks / 8)

octet_index = floor(coeff_index / 8)

bit_within_octet = coeff_index MOD 8

4.2.3.2. **List of Chunk Indicies: FEC Scheme Type = 2**

For Encoding Vectors with a low Hamming weight, i.e. with few coefficients that have the value of one, a list of the vector indicies for the ones reduce the parameter length. The list of indices starts with the list length, followed by the list of indicies. All numbers SHALL be in SDNV format. The index list has the following format:

Field	Type	Description
List Length	SDNV	The number of indicies in the Encoding Vector with coefficient of one.
Index List	SDNV	List of indicies with coefficient value equal to one. The indicies SHOULD be in order from least to largest. Duplicate indicies SHALL NOT be sent by the Encoder and SHALL be ignored by the Decoder.

Table 3: List of Indicies

4.2.3.3. Windowed Binary Array: FEC Scheme Type = 3

When an Encoding Vector has its ones grouped in a single small range of indicies, for example Windowed encoding [Stud2006], a partial bit vector should be sent. The starting index is sent along with a bit array of that contains all the coefficients that are ones. The Windowed Octet Array has the following format:

Field	Type	Description
Lowest Index	SDNV	The lowest index value with a coefficient of one. Bit index for zero of the Packed Octet Array will be offset by this index.
Length Octet Array	SDNV	octet_array_length = ceiling((highest_index - lowest_index) / 8)
Packed Binary Array	SDNV	Encoding Vector packed into octet array using bit array to octet array mapping from FEC Scheme Type 1.

Table 4: Partial Vector

4.2.3.4. Finite Field Array: FEC Scheme Type = 4

Random Binary FEC is a special case of a Random Finite Field FEC, where the finite field is specifically GF(2), instead of GF(2^m). In the general Random Finite Field FEC, Encoding Vector coefficients are represented as a binary number of length m. The finite field GF(2^m) is characterized by a irreducible polynomial from Section 8.1 of [RFC5510]. Higher values of m, such as m=8, are useful in situations

where the number of Chunks is small and it is desirable to reduce the number of redundant Encodings that are expected to be received in order to get a full rank. Random Binary FEC implementations MUST be able to interpret a Finite Field Array with m=1, as a Full Binary Array.

For transmission, Encoding Vector coefficients are packed into an array of bits. The lowest bit index and coefficient index is 0. The coefficient with index 0 is packed into with its least significant bit into bit index 0. Subsequent coefficients are concatenated to fill the bit array. The bit array is packed into a octet array. If the number of coefficients (n) times their length (m) is not a multiple of 8, padding bits are added to the highest bit positions using the value of zero. The resulting octet array is sent with the highest index first.

Field	Type	Description
Finite Field Degree (m)	SDNV	Specifies the finite field of the form GF(2^m), where m is the Finite Field Degree.
Packed Coefficients Array	Octets	Encoding Vector packed into an octet array, with each coefficient is represented as a binary number of length m.

Table 5: Finite Field

The octet array has the following implicit parameters:

octet_array_length = ceiling((number_of_chunks * m) / 8)

octet_index = floor((coeff_index * m) / 8)

starting_bit_within_octet = (coeff_index * m) MOD 8

5. Procedures

The Random Binary FEC scheme uses the following procedures which are common to the encode, decode and recode processes.

5.1. Bitwise XOR

Encoding involves the XOR operation on multiple Chunks to form an Encoding Data. Decoding involves the XOR operation on multiple Encoding Datas to recover a Chunk. XORing two octet arrays logically takes every bit in one array and performs the XOR operation on the corresponding bit in the other array. That is, the octet index and the bit position within the octet are the same. The results are put into the corresponding bit of a new array. Note that bits that do not share the same index do not interact with each other. So even though Chunks and Encoding Data are defined as octet arrays, the bit-wise XOR can be implemented using any convenient memory unit, such as byte, int or long.

The XOR operation is the most CPU intensive operation used by this FEC scheme, so the number of XOR operations SHOULD be minimized and the XOR operation implementation SHOULD be efficient. To minimize XORs in the encoding process, a low Hamming weight Encoding Vector SHOULD be used. To maximize the efficiency of the XOR operation, the largest memory unit available SHOULD be used, such as 64 bit long.

5.2. Solve

To solve the simultaneous equations to decode the Encodings back into Chunks, the most general solution is to use Gaussian Elimination to either invert the Encoding Set matrix or to algebraically solve the equations directly. The Encoding Vectors are used as rows to form a Encoding Set matrix (S). The Encoding Data can be used to form vector (e). The encoding process can be represented in matrix notation as a vector of Encoding Data (e) that was created by multiplying the Encoding Set matrix (S) by the vector of Chunks (c).
e = c S
Gaussian Elimination can be used to calculate the inverse of the encoding matrix (S^-1). The Chunks can be recovered by multiplying the vector of Encodings Datas by the inverted encoding matrix.
c = e S^-1
If the Hamming weight of the Encoding Vectors are low and hence the Encoding Set matrix is sparse. Solving the equations algebraically instead of using the matrix inversion, usually results in less octet array XOR operations.

Gaussian Elimination is an expensive operation that involves O(N^3) operations over the field GF(2) and O(N^2) XOR operations on Encoding

Data octet arrays. A large body of research has been conducted to create efficient algorithms to solve simultaneous equations and will not be presented in this document, but SHOULD be exploited by implementations of the Random Binary FEC scheme. Many of these algorithms involve restricting the form of the Encoding Vectors, with dramatic reductions in encoding or decoding cost, but with other tradeoffs in terms of reliability, bandwidth used, or other systemic factors. Section 7 discusses several options on how to configure the encoding process to best match the tradeoffs.

5.3. Rank

The rank operation determines the number of linearly independent Encodings in an Encoding Set, i.e the rank of the Encoding Set matrix S. The rank operator is less expensive than solving the whole Encoding Set. If the rank is calculated incrementally as each Encoding is inserted into its Encoding Set, then an insert has O(N^2) operations in the field GF(2), but needs no XOR operations on Encoding Data octet arrays. Given the reduced cost of the rank operator, it can be used to determine which Encodings to use in the solving process. It is also used to detect redundant Encodings. As with solving, a large body of research has been conducted to create efficient algorithms to calculate rank and will not be presented in this document, but SHOULD be exploited by implementations of the Random Binary FEC scheme.

6. Random Binary FEC code specification

The Coding layer consists of an Encoder and Decoder at the end points. Intermediate Recoders may also be used to generate new Encodings from previously received Encodings to reduce the chance that duplicate Encodings are propagated over different paths to the destination.

6.1. Encoder

The encoding process transforms a linear combination of Chunks into an Encoding. The encoding coefficients are stored in the Encoding Vector. For the general Random Binary Encoding the coefficients are generated randomly, for example by setting K indicies to one and the rest to zero. The Hamming weight of the of the Encoding Vector SHOULD be low to reduce the number of bitwise XOR operations done by the Encoder process. Empirical measurement show a Hamming weight of O(log N) generates Encodings that are as diverse as using Hamming weights of O(N) [Stud2006]. One caution on generating Encoding Vectors, if all the Hamming weights for Encodings in an Encoding Set are even, then the Encoding Set can not be solved [Stud2006]. A simple solution to avoid this situation is to generate Encoding Vectors with odd Hamming weights.

The Encoding Data (E) is generated by taking the binary dot product of the Encoding Vector (V) and the vector of Chunks (C). That is, the Encoding Data accumulates an octet array that is the XOR of Chunks whose coefficient is one in the Encoding Vector.
E = V dot C

6.2. Decoder

When the Encoding Set rank is equal to the number of Chunks (N), then N linearly independent Encodings can be extracted and used to solve for the Chunks, see Section 5.2. If the encoding process is restricted, then simplified decoding algorithms can be used that exploit the restriction. The choice of decoding algorithm is left to the implementation, but to support interoperability, implementations MUST support the unrestricted encoding process. For example, a decoder could detect the pattern of Encodings and use the appropriate decoder algorithm, but would default to Gaussian Elimination.

Decoding can be done incrementally by partially solving the decoding equations as the Encodings arrive. For some configurations of the encoding process, such as Block Parity, some Chunks can be solved before all N innovative Encodings have arrived. In these cases, the Decoder MAY deliver these Chunks to the Data Object layer before all Chunks can be decoded.

6.3. Intermediate Recoder

Intermediate Recoders generate new Encodings from the Encodings that it has already received. Recoding reduces the chance that duplicate Encodings are delivered over different paths to the destination. The recode operation selects several Encodings from the Encoding Set at Random. The selected Encodings are combined to form a new Encoding. The combination procedure is as follows

The new Encoding Vector is the XOR of the coefficients of all the selected Encoding Vectors

The new Encoding Data is the XOR of the octet arrays of all the selected Encoding Datas.

When two Encodings with Hamming weight less than N/2 are combined, the resulting Hamming weight tends be larger than the originals. Conversely, when two Encodings with Hamming weight more than N/2 are combined, the resulting Hamming weight tend be smaller than the originals. Thus, the recoding process drives the Hamming weight towards N/2. As Encoding Bundles are transfered across the DTN, recoding can change any special configuration restrictions put on the encoding process. Recoding SHOULD have the option to be disabled as part of the Regulating Layer Handling Specification.

Care should be given to the recoding process to insure that all Encodings in the Encoding Set are represented in the new stream of recoded Encodings. If each new Encoding draws always from the whole Encoding Set, then some Encodings will be chosen less often than others. Hence their information will not be propagated as much as Encodings that were selected more often. This problem is a form of the Coupon Collectors Problem, which results in an Encoding Stream that needs to receive up to N Log (N) Encodings instead of only N + 2 to receive the full rank. One solution is to generate new Encodings in cycles. Each Encoding is allowed to be used only once during a cycle and when all Encodings are used a new cycle begins.

7. Configure

The Random Binary Scheme may be configured to implement the following basic FEC schemes, all of which can be represented by the formats in Section 4.2.3 . The configurations restrict the coding formulas, which results in encoding streams with different properties, and potentially different decoding algorithms. Some of these FEC schemes are described in other RFCs and are fully specified here for the content delivery protocol using DTN bundles.

7.1. Full Random Binary

Full Random Binary is the generic configuration on which other configuration characteristics are compared. Full Random Binary generates encodings randomly over the full range of possible Encoding Vectors. A new Encoding is generated by randomly setting each coefficient in the Encoding Vector to one, with a probability of 1/2. The resulting stream of Encodings has an average Hamming weight of N / 2. So the encoding process has O(N) octet array XORs. The received Encoding Set has no special structure, so the decoder must use full Gaussian Elimination. The algebraic solver algorithm does not have an advantage over the matrix inversion in this case, so the decoder process has O(N^2) octet array XORs. The expected transmission overhead is only N + 1.6, when the number of Chunks is on the order of 1000's. Finally, the coefficients of the Encoding Vector has no restrictions, so the Encoding Vector is packed into the full vector format (Table 2).

7.2. Windowed Erasure Codes

With a simple restriction for how the random coefficients are generated, the encoding and decoding cost can be dramatically reduced while still maintaining the low transmission overhead of the Full Random configuration [Stud2006].

The windowed configuration first restricts the Hamming weight and then restricts the range that coefficients can be set to one to a "window" of consecutive indicies. The Hamming weight is restricted to 2 log (N), but should be odd. So the encoder process is O(log N) instead of O(N) with Full Random. The window has a length of 2 sqr(N) and the offset is chosen randomly for each new Encoding, with wrapping from highest to lowest index. With a slight modification to the Gaussian Elimination algorithm, the decoder can algebraically solve the Encoding Set with a windowed matrix in O(N^2.5), instead of the O(N^3) for Full Random. The transmission overhead remains close to the N + 1.6 overhead of the Full Random. Unfortunately, unconstrained Recoding will disrupt the specialized form of the windowed encoding matrix, which will result in higher decoding times

to again be O(N^3). Finally the coefficients are restricted to a window, so the Encoding Vector should be packed into the partial vector format (Table 4).

7.3. Compact No-code FEC Scheme

The degenerate case of sending only Encodings with Hamming weight of one, i.e. only source symbols, can behave like an additional fragmentation layer or as the test case named "Compact No-code FEC Scheme" in [RFC5445]. In this configuration, the encoding and decoding process perform no work, but the system is not protected from any dropped bundles. Since the Encoding Vector has only one coefficient with value of one, its index should be packed into the list of indicies format (Table 3).

7.4. Block Parity

To show the flexibility of the Random Binary FEC scheme, the classic block parity FEC scheme described in [RFC5445] can be fully specified for the DTN content delivery protocol using DTN bundles. As in the Compact No-code FEC Scheme, source symbols are sent unencoded with its Chunk index packed into the list of indicies format (Table 3). The block parity repair symbol has all the coefficients in the block to set to one, and uses partial vector format (Table 4). The normal incremental decoder will automatically detect source symbols as solved. The parity repair symbol will be applied, if any source symbols are dropped, or it is treated as redundant, if no bundles where dropped in the block. Chunks may be delivered as they arrive. The Block Parity FEC scheme is practical in the case where dropped bundles are rare and not bursty.

8. Security Considerations

No additional security considerations have been identified beyond those described in [ErasureCoding]

9. IANA Considerations

The Random Binary Scheme uses three FEC Encoding IDs. The assigned IDs should be less than 128 in order to fit into one byte using SDNV values. The reference implementation uses the following FEC Scheme Types:

Full Binary Array = 1

List of Chunk Indicies = 2

Windowed Binary Array = 3

Finite Field Array = 4

10. References

10.1. Normative References

[ErasureCoding]
Zinky, J., Caro, A., and G. Stein, "Bundle Protocol Erasure Coding Extension", draft-zinky-dtnrg-erasure-coding-extension-00 (work in progress), Aug 2012.

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

[RFC5050] Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, November 2007.

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

[RFC5510] Lacan, J., Roca, V., Peltotalo, J., and S. Peltotalo, "Reed-Solomon Forward Error Correction (FEC) Schemes", RFC 5510, April 2009.

[RFC6256] Eddy, W. and E. Davies, "Using Self-Delimiting Numeric Values in Protocols", RFC 6256, May 2011.

10.2. Informative References

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

[Stud2006]
Studholme, C. and I. Blake, "Windowed Erasure Codes", IEEE International Symposium on Information Theory, July 2006.

Authors' Addresses

John Zinky
Raytheon BBN Technologies
10 Moulton St.
Cambridge, MA 02138
US

Email: jzinky@bbn.com

Armando Caro
Raytheon BBN Technologies
10 Moulton St.
Cambridge, MA 02138
US

Email: acaro@bbn.com

Gregory Stein
Laboratory for Telecommunications Sciences
8080 Greenmead Drive
College Park, MD 20740
US

Email: gstein@ece.umd.edu