Reliable Multicast Transport
Internet-Draft
Intended status: Standards Track
Expires: September 9, 2011

M. Luby
Qualcomm Incorporated
A. Shokrollahi
EPFL
M. Watson
Netflix Inc.
T. Stockhammer
Nomor Research
L. Minder
Qualcomm Incorporated

March 8, 2011

RaptorQ Forward Error Correction Scheme for Object Delivery
draft-ietf-rmt-bb-fec-raptorq-05

Abstract

This document describes a Fully-Specified FEC scheme, corresponding to FEC Encoding ID 6 (to be confirmed (tbc)), for the RaptorQ forward error correction code and its application to reliable delivery of data objects.

RaptorQ codes are a new family of codes that provide superior flexibility, support for larger source block sizes and better coding efficiency than Raptor codes in RFC5053. RaptorQ is also 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 of data. The decoder is able to recover the source block from any set of encoding symbols for most cases equal to the number of source symbols and in rare cases with slightly more than the number of source symbols.

The RaptorQ code described here is a systematic code, meaning that all the source symbols are among the encoding symbols that can be generated.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months

and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 9, 2011.

Copyright Notice

Table of Contents

1. Introduction

This document specifies an FEC Scheme for the RaptorQ forward error correction code for object delivery applications. The concept of an FEC Scheme is defined in RFC5052 [RFC5052] and this document follows the format prescribed there and uses the terminology of that document. The RaptorQ code described herein is a next generation of the Raptor code described in RFC5053 [RFC5053]. The RaptorQ code provides superior reliability, better coding efficiency, and support for larger source block sizes than the Raptor code of RFC5053 [RFC5053]. These improvements simplify the usage of the RaptorQ code in an object delivery Content Delivery Protocol compared to RFC5053 [RFC5053].

The RaptorQ FEC Scheme is a Fully-Specified FEC Scheme corresponding to FEC Encoding ID 6 (tbc).

> Editor's Note: The finalized FEC encoding ID is still to be defined, but '6 (tbc)' is used as temporary value in this Internet Draft expecting sequential use of FEC encoding IDs in the IANA registration process.

RaptorQ 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 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.

The code described in this document is a systematic code, that is, the original source symbols can be sent unmodified from sender to receiver, as well as a number of repair symbols. For more backgound on the use of Forward Error Correction codes in reliable multicast, see [RFC3453].

2. Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Formats and Codes

3.1. FEC Payload IDs

The FEC Payload ID MUST be a 4-octet field defined as follows:

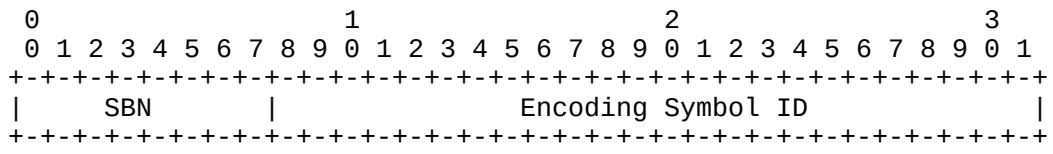

Figure 1: FEC Payload ID format

- Source Block Number (SBN), (8 bits, unsigned integer): A non-negative integer identifier for the source block that the encoding symbols within the packet relate to.

- Encoding Symbol ID (ESI), (24 bits, unsigned integer): A non-negative integer identifier for the encoding symbols within the packet.

The interpretation of the Source Block Number and Encoding Symbol Identifier is defined in Section 4.

3.2. FEC Object Transmission Information

3.2.1. Mandatory

The value of the FEC Encoding ID MUST be 6, as assigned by IANA (see Section 7).

3.2.2. Common

The Common FEC Object Transmission Information elements used by this FEC Scheme are:

- Transfer Length (F), (40 bits, unsigned integer): A non-negative integer that is at most 946270874880. This is the transfer length of the object in units of octets.

- Symbol Size (T), (16 bits, unsigned integer): A positive integer that is less than 2^^16. This is the size of a symbol in units of octets.

The encoded Common FEC Object Transmission Information format is shown in Figure 2.

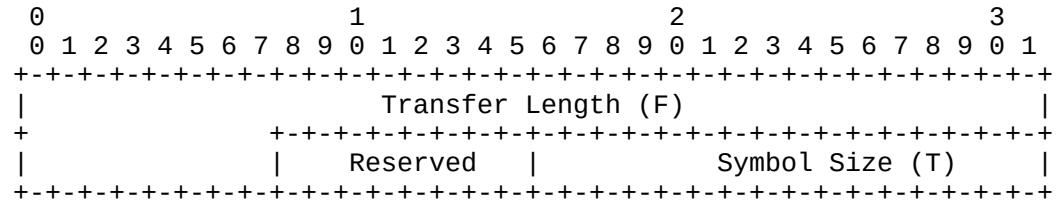

Figure 2: Encoded Common FEC OTI for RaptorQ FEC Scheme

NOTE 1: The limit of 946270874880 on the transfer length is a consequence of the limitation on the symbol size to 2^^16-1, the limitation on the number of symbols in a source block to 56403 and the limitation on the number of source blocks to 2^^8.

3.2.3. Scheme-Specific

The following parameters are carried in the Scheme-Specific FEC Object Transmission Information element for this FEC Scheme:

- The number of source blocks (Z) (12 bits, unsigned integer)
- The number of sub-blocks (N) (12 bits, unsigned integer)
- A symbol alignment parameter (Al) (8 bits, unsigned integer)

These parameters are all positive integers. The encoded Scheme-specific Object Transmission Information is a 4-octet field consisting of the parameters Z, N and Al as shown in Figure 3.

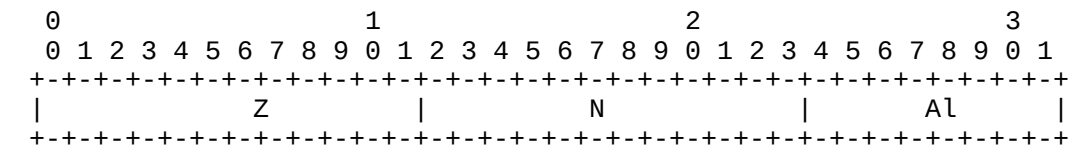

Figure 3: Encoded Scheme-specific FEC Object Transmission Information

The encoded FEC Object Transmission Information is a 12-octet field consisting of the concatenation of the encoded Common FEC Object Transmission Information and the encoded Scheme-specific FEC Object Transmission Information.

These three parameters define the source block partitioning as described in Section 4.4.1.2

4. Procedures

4.1. Introduction

For any undefined symbols or functions used in this section, in particular the functions "ceil" and "floor", refer to Section 5.1.

4.2. Content Delivery Protocol Requirements

This section describes the information exchange between the RaptorQ FEC Scheme and any Content Delivery Protocol (CDP) that makes use of the RaptorQ FEC Scheme for object delivery.

The RaptorQ encoder scheme and RaptorQ decoder scheme for object delivery require the following information from the CDP:

- The transfer length of the object, F, in octets
- A symbol alignment parameter, Al
- The symbol size, T, in octets, which MUST be a multiple of Al
- The number of source blocks, Z
- The number of sub-blocks in each source block, N

The RaptorQ encoder scheme for object delivery additionally requires:

- the object to be encoded, F octets

The RaptorQ encoder scheme supplies the CDP with the following information for each packet to be sent:

- Source Block Number (SBN)
- Encoding Symbol ID (ESI)
- Encoding symbol(s)

The CDP MUST communicate this information to the receiver.

4.3. Example Parameter Derivation Algorithm

This section provides recommendations for the derivation of the three transport parameters, T, Z and N. This recommendation is based on the following input parameters:

- F the transfer length of the object, in octets
- WS the maximum size block that is decodable in working memory, in octets
- P' the maximum payload size in octets, which is assumed to be a multiple of Al
- Al the symbol alignment parameter, in octets
- SS a parameter where the desired lower bound on the sub-symbol size is SS*Al
- K'_max the maximum number of source symbols per source block.

  Note: Section 5.1.2 defines K'_max to be 56403.

Based on the above inputs, the transport parameters T, Z and N are calculated as follows:

Let,

- T = P'
- Kt = ceil(F/T)
- N_max = floor(T/(SS*Al))
- for all n=1, ..., N_max
  - KL(n) is the maximum K' value in Table 2 in Section 5.6 such that

    K' <= WS/(Al*(ceil(T/(Al*n))))

- Z = ceil(Kt/KL(N_max))
- N is the minimum n=1, ..., N_max such that ceil(Kt/Z) <= KL(n)

It is RECOMMENDED that each packet contains exactly one symbol. However, receivers SHALL support the reception of packets that contain multiple symbols.

The value Kt is the total number of symbols required to represent the source data of the object.

The algorithm above and that defined in Section 4.4.1.2 ensure that the sub-symbol sizes are a multiple of the symbol alignment

parameter, Al. This is useful because the sum operations used for encoding and decoding are generally performed several octets at a time, for example at least 4 octets at a time on a 32 bit processor. Thus the encoding and decoding can be performed faster if the sub-symbol sizes are a multiple of this number of octets.

The recommended setting for the input parameter Al is 4.

The parameter WS can be used to generate encoded data which can be decoded efficiently with limited working memory at the decoder. Note that the actual maximum decoder memory requirement for a given value of WS depends on the implementation, but it is possible to implement decoding using working memory only slightly larger than WS.

4.4. Object Delivery

4.4.1. Source block construction

4.4.1.1. General

In order to apply the RaptorQ encoder to a source object, the object may be broken into Z >= 1 blocks, known as source blocks. The RaptorQ encoder is applied independently to each source block. Each source block is identified by a unique Source Block Number (SBN), where the first source block has SBN zero, the second has SBN one, etc. Each source block is divided into a number, K, of source symbols of size T octets each. Each source symbol is identified by a unique Encoding Symbol Identifier (ESI), where the first source symbol of a source block has ESI zero, the second has ESI one, etc.

Each source block with K source symbols is divided into N >= 1 sub-blocks, which are small enough to be decoded in the working memory. Each sub-block is divided into K sub-symbols of size T'.

Note that the value of K is not necessarily the same for each source block of an object and the value of T' may not necessarily be the same for each sub-block of a source block. However, the symbol size T is the same for all source blocks of an object and the number of symbols, K is the same for every sub-block of a source block. Exact partitioning of the object into source blocks and sub-blocks is described in Section 4.4.1.2 below.

4.4.1.2. Source block and sub-block partitioning

The construction of source blocks and sub-blocks is determined based on five input parameters, F, Al, T, Z and N and a function Partition[]. The five input parameters are defined as follows:

- F the transfer length of the object, in octets
- Al a symbol alignment parameter, in octets
- T the symbol size, in octets, which MUST be a multiple of Al
- Z the number of source blocks
- N the number of sub-blocks in each source block

These parameters MUST be set so that ceil(ceil(F/T)/Z) <= K'_max. Recommendations for derivation of these parameters are provided in Section 4.3.

The function Partition[I,J] derives parameters for partitioning a block of size I into J approximately equal sized blocks, and more specifically partitions I into JL blocks of length IL and JS blocks of length IS. Specifically, the output of Partition[I, J] is the sequence (IL, IS, JL, JS), where IL = ceil(I/J), IS = floor(I/J), JL = I - IS * J and JS = J - JL.

The source object MUST be partitioned into source blocks and sub-blocks as follows:

Let

- Kt = ceil(F/T),
- (KL, KS, ZL, ZS) = Partition[Kt, Z],
- (TL, TS, NL, NS) = Partition[T/Al, N].

Then, the object MUST be partitioned into Z = ZL + ZS contiguous source blocks, the first ZL source blocks each having KL*T octets, i.e. KL source symbols of T octets each, and the remaining ZS source blocks each having KS*T octets, i.e. KS source symbols of T octets each.

If Kt*T > F then for encoding purposes, the last symbol of the last source block MUST be padded at the end with Kt*T-F zero octets.

Next, each source block with K source symbols MUST be divided into N = NL + NS contiguous sub-blocks, the first NL sub-blocks each consisting of K contiguous sub-symbols of size of TL*Al octets and the remaining NS sub-blocks each consisting of K contiguous sub-symbols of size of TS*Al octets. The symbol alignment parameter Al ensures that sub-symbols are always a multiple of Al octets.

Finally, the m-th symbol of a source block consists of the concatenation of the m-th sub-symbol from each of the N sub-blocks. Note that this implies that when N > 1 then a symbol is NOT a contiguous portion of the object.

4.4.2. Encoding packet construction

Each encoding packet contains the following information:

- Source Block Number (SBN)
- Encoding Symbol ID (ESI)
- encoding symbol(s)

Each source block is encoded independently of the others. Each encoding packet contains encoding symbols generated from the one source block identified by the SBN carried in the encoding packet. Source blocks are numbered consecutively from zero.

Encoding Symbol ID values from 0 to K-1 identify the source symbols of a source block in sequential order, where K is the number of source symbols in the source block. Encoding Symbol IDs K onwards identify repair symbols generated from the source symbols using the RaptorQ encoder.

Each encoding packet either consists entirely of source symbols (source packet) or entirely of repair symbols (repair packet). A packet may contain any number of symbols from the same source block. In the case that the last source symbol in a source packet includes padding octets added for FEC encoding purposes then these octet need not be included in the packet. Otherwise, each packet MUST contain only whole symbols.

The Encoding Symbol ID, X, carried in each source packet is the Encoding Symbol ID of the first source symbol carried in that packet. The subsequent source symbols in the packet have Encoding Symbol IDs, X+1 to X+G-1, in sequential order, where G is the number of symbols in the packet.

Similarly, the Encoding Symbol ID, X, placed into a repair packet is the Encoding Symbol ID of the first repair symbol in the repair packet and the subsequent repair symbols in the packet have Encoding Symbol IDs X+1 to X+G-1 in sequential order, where G is the number of symbols in the packet.

Note that it is not necessary for the receiver to know the total number of repair packets.

4.4.3. Example receiver recovery strategies

A receiver can use the received encoding symbols for each source block of an object to recover the source symbols for that source block independently of all other source blocks.

If there is one sub-block per source block, i.e., N = 1, then the portion of the data in the original object in its original order associated with a source block consists of the concatenation of the source symbols of a source block in consecutive ESI order.

If there are multiple sub-blocks per source block, i.e., if N > 1, then the portion of the data in the original object in its original order associated with a source block consists of the concatenation of the sub-blocks associated with the source block, where sub-symbols within each sub-block are in consecutive ESI order. In this case, there are different receiver source block recovery strategies worth considering depending on the Random Access Memory (RAM), as outlined below.

One strategy is to recover the source symbols of a source block using the decoding procedures applied to the received symbols for the source block to recover the source symbols as described in Section 5, and then to reorder the sub-symbols of the source symbols so that all consecutive sub-symbols of the first sub-block are first, followed by all consecutive sub-symbols of the second sub-block, etc., followed by all consecutive sub-symbols of the Nth sub-block. This strategy is especially applicable if you have if you have enough RAM to decode an entire source block.

Another strategy is to separately recover the sub-blocks of a source block. For example, a receiver may de-multiplex and store sub-symbols associated with each sub-block separately as packets containing encoding symbols arrive, and then use the stored sub-symbols received for a sub-block to recover that sub-block using the decoding procedures described in Section 5. This strategy is especially applicable if you have enough RAM to decode only one subblock at a time.

5. RaptorQ FEC Code Specification

5.1. Background

For the purpose of the RaptorQ FEC code specification in this section, the following definitions, symbols and abbreviations apply. A basic understanding of linear algebra, matrix operations, and finite fields is assumed in this section. In particular, matrix

multiplication and matrix inversion operations over a mixture of the finite fields GF[2] and GF[256] are used. A basic familiarity with sparse linear equations, and efficient implementations of algorithms that take advantage of sparse linear equations, is also quite beneficial to an implementer of this specification.

5.1.1. Definitions

- Source block: a block of K source symbols which are considered together for RaptorQ encoding and decoding purposes.
- Extended Source Block: a block of K' source symbols, where K' >= K constructed from a source block and zero or more padding symbols.
- Symbol: a unit of data. The size, in octets, of a symbol is known as the symbol size. The symbol size is always a positive integer.
- Source symbol: the smallest unit of data used during the encoding process. All source symbols within a source block have the same size.
- Padding symbol: a symbol with all zero bits that is added to the source block to form the extended source block.
- Encoding symbol: a symbol that can be sent as part of the encoding of a source block. The encoding symbols of a source block consist of the source symbols of the source block and the repair symbols generated from the source block. Repair symbols generated from a source block have the same size as the source symbols of that source block.
- Repair symbol: the encoding symbols of a source block that are not source symbols. The repair symbols are generated based on the source symbols of a source block.
- Intermediate symbols: symbols generated from the source symbols using an inverse encoding process based on pre-coding relationships. The repair symbols are then generated directly from the intermediate symbols. The encoding symbols do not include the intermediate symbols, i.e., intermediate symbols are not sent as part of the encoding of a source block. The intermediate symbols are partitioned into LT symbols and PI symbols for the purposes of the encoding process.
- LT symbols: A process similar to that described in [LTCodes] is used to generate part of the contribution to each generated encoding symbol from the portion of the intermediate symbols designated as LT symbols.

- PI symbols: A process even simpler than that described in [LTCodes] is used to generate the other part of the contribution to each generated encoding symbol from the portion of the intermediate symbols designated as PI symbols. In the decoding algorithm suggested in Section 5.4, the PI symbols are inactivated at the start, i.e., are placed into the matrix U at the beginning of the first phase of the decoding algorithm. Because the symbols corresponding to the columns of U are sometimes called the "inactivated" symbols, and since the PI symbols are inactivated at the beginning, they are considered "permanently inactivated".

- HDPC symbols: There is a small subset of the intermediate symbols that are HDPC symbols. Each HDPC symbol has a pre-coding relationship with a large fraction of the other intermediate symbols. HDPC means "High Density Parity Check".

- LDPC symbols: There is a moderate-sized subset of the intermediate symbols that are LDPC symbols. Each LDPC symbol has a pre-coding relationship with a small fraction of the other intermediate symbols. LDPC means "Low Density Parity Check".

- Systematic code: a code in which all source symbols are included as part of the encoding symbols of a source block. The RaptorQ code as described herein is a systematic code.

- Encoding Symbol ID (ESI): information that uniquely identifies each encoding symbol associated with a source block for sending and receiving purposes.

- Internal Symbol ID (ISI): information that uniquely identifies each symbol associated with an extended source block for encoding and decoding purposes.

- Arithmetic operations on octets and symbols and matrices: The operations that are used to produce encoding symbols from source symbols and vice-versa. See Section 5.7.

5.1.2. Symbols

i, j, u, v, h, d, a, b, d1, a1, b1, v, m, x, y represent values or variables of one type or another, depending on the context.

X denotes a non-negative integer value that is either an ISI value or an ESI value, depending on the context.

ceil(x) denotes the smallest integer which is greater than or equal to x, where x is a real value.

floor(x) denotes the largest integer which is less than or equal to x, where x is a real value.

min(x,y) denotes the minimum value of the values x and y, and in general the minimum value of all the argument values.

max(x,y) denotes the maximum value of the values x and y, and in general the maximum value of all the argument values.

i % j denotes i modulo j.

i + j denotes the sum of i and j. If i and j are octets, respectively symbols, this designates the arithmetic on octets, respectively symbols, as defined in Section 5.7. If i and j are integers, then it denotes the usual integer addition.

i * j denotes the product of i and j. If i and j are octets, this designates the arithmetic on octets, as defined in Section 5.7. If i is an octet and j is a symbol, this denotes the multiplication of a symbol by an octet, as also defined in Section 5.7. Finally, if i and j are integers, i * j denotes the usual product of integers.

a ^^ b denotes the operation a raised to the power b. If a is an octet and b is a non-negative integer, this is understood to mean a*a*...*a (b terms), with '*' being the octet product as defined in Section 5.7.

u ^ v denotes, for equal-length bit strings u and v, the bitwise exclusive-or of u and v.

Transpose[A] denotes the transposed matrix of matrix A. In this specification, all matrices have entries that are octets.

A^^-1 denotes the inverse matrix of matrix A. In this specification, all the matrices have octets as entries, so it is understood that the operations of the matrix entries are to be done as stated in Section 5.7 and A^^-1 is the matrix inverse of A with respect to octet arithmetic.

K denotes the number of symbols in a single source block.

K’ denotes the number of source plus padding symbols in an extended source block. For the majority of this specification, the padding symbols are considered to be additional source symbols.

K’_max denotes the maximum number of source symbols that can be in a single source block. Set to 56403.

L denotes the number of intermediate symbols for a single extended source block.

S denotes the number of LDPC symbols for a single extended source block. These are LT symbols. For each value of K’ shown in Table 2 in Section 5.6, the corresponding value of S is a prime number.

H denotes the number of HDPC symbols for a single extended source block. These are PI symbols.

B denotes the number of intermediate symbols that are LT symbols excluding the LDPC symbols.

W denotes the number of intermediate symbols that are LT symbols. For each value of K’ in Table 2 shown in Section 5.6, the corresponding value of W is a prime number.

P denotes the number of intermediate symbols that are PI symbols. These contain all HDPC symbols.

P1 denotes the smallest prime number greater than or equal to P.

U denotes the number of non-HDPC intermediate symbols that are PI symbols.

C denotes an array of intermediate symbols, C[0], C[1], C[2],..., C[L-1].

C’ denotes an array of the symbols of the extended source block, where C’[0], C’[1], C’[2],..., C’[K-1] are the source symbols of the source block and C’[K], C’[K+1],..., C’[K’-1] are padding symbols.

V0, V1, V2, V3 denote four arrays of 32-bit unsigned integers, V0[0], V0[1],..., V0[255] ; V1[0], V1[1],..., V1[255]; V2[0], V2[1],..., V2[255]; and V3[0], V3[1],..., V3[255] as shown in Section 5.5.

Rand[y, i, m] denotes a pseudo-random number generator

Deg[v] denotes a degree generator

Enc[K', C ,(d, a, b, d1, a1, b1)] denotes an encoding symbol generator

Tuple[K', X] denotes a tuple generator function

T denotes the symbol size in octets.

J(K') denotes the systematic index associated with K'.

G denotes any generator matrix.

I_S denotes the SxS identity matrix.

5.2. Overview

This section defines the systematic RaptorQ FEC code.

Symbols are the fundamental data units of the encoding and decoding process. For each source block all symbols are the same size, referred to as the symbol size T. The atomic operations performed on symbols for both encoding and decoding are the arithmetic operations defined in Section 5.7.

The basic encoder is described in Section 5.3. The encoder first derives a block of intermediate symbols from the source symbols of a source block. This intermediate block has the property that both source and repair symbols can be generated from it using the same process. The encoder produces repair symbols from the intermediate block using an efficient process, where each such repair symbol is the exclusive OR of a small number of intermediate symbols from the block. Source symbols can also be reproduced from the intermediate block using the same process. The encoding symbols are the combination of the source and repair symbols.

An example of a decoder is described in Section 5.4. The process for producing source and repair symbols from the intermediate block is designed so that the intermediate block can be recovered from any sufficiently large set of encoding symbols, independent of the mix of source and repair symbols in the set. Once the intermediate block is recovered, missing source symbols of the source block can be recovered using the encoding process.

Requirements for a RaptorQ compliant decoder are provided in Section 5.8. A number of decoding algorithms are possible to achieve

these requirements. An efficient decoding algorithm to achieve these requirements is provided in Section 5.4.

The construction of the intermediate and repair symbols is based in part on a pseudo-random number generator described in Section 5.3. This generator is based on a fixed set of 1024 random numbers which must be available to both sender and receiver. These numbers are provided in Section 5.5. Encoding and decoding operations for RaptorQ use operations on octets. Section 5.7 describes how to perform these operations.

Finally, the construction of the intermediate symbols from the source symbols is governed by "systematic indices", values of which are provided in Section 5.6 for specific extended source block sizes between 6 and K'_max = 56403 source symbols. Thus, the RaptorQ code supports source blocks with between 1 and 56403 source symbols.

5.3. Systematic RaptorQ encoder

5.3.1. Introduction

For a given source block of K source symbols, for encoding and decoding purposes the source block is augmented with K'-K additional padding symbols, where K' is the smallest value that is at least K in the systematic index Table 2 of Section 5.6. The reason for padding out a source block to a multiple of K' is to enable faster encoding and decoding, and to minimize the amount of table information that needs to be stored in the encoder and decoder.

For purposes of transmitting and receiving data, the value of K is used to determine the number of source symbols in a source block, and thus K needs to be known at the sender and the receiver. In this case the sender and receiver can compute K' from K and the K'-K padding symbols can be automatically added to the source block without any additional communication. The encoding symbol ID (ESI) is used by a sender and receiver to identify the encoding symbols of a source block, where the encoding symbols of a source block consist of the source symbols and the repair symbols associated with the source block. For a source block with K source symbols, the ESIs for the source symbols are 0,1,2,...,K-1 and the ESIs for the repair symbols are K, K+1, K+2,... . Using the ESI for identifying encoding symbols in transport ensures that the ESI values continue consecutively between the source and repair symbols.

For purposes of encoding and decoding data, the value of K' derived from K is used as the number of source symbols of the extended source block upon which encoding and decoding operations are performed, where the K' source symbols consist of the original K source symbols

and an additional K'-K padding symbols. The internal symbol ID (ISI) is used by the encoder and decoder to identify the symbols associated with the extended source block, i.e., for generating encoding symbols and for decoding. For a source block with K original source symbols, the ISIs for the original source symbols are 0,1,2,...,K-1, the ISIs for the K'-K padding symbols are K, K+1, K+2,..., K'-1, and the ISIs for the repair symbols are K', K'+1, K'+2,... . Using the ISI for encoding and decoding allows the padding symbols of the extended source block to be treated the same way as other source symbols of the extended source block, and that a given prefix of repair symbols are generated in a consistent way for a given number K' of source symbols in the extended source block independent of K.

The relationship between the ESIs and the ISIs is simple: the ESIs and the ISIs for the original K source symbols are the same, the K'-K padding symbols have an ISI but do not have a corresponding ESI (since they are symbols that are neither sent nor received), and a repair symbol ISI is simply the repair symbol ESI plus K'-K. The translation between ESIs used to identify encoding symbols sent and received and the corresponding ISIs used for encoding and decoding, and the proper padding of the extended source block with padding symbols used for encoding and decoding, is the responsibility of the padding function in the RaptorQ encoder/decoder.

5.3.2. Encoding overview

The systematic RaptorQ encoder is used to generate any number of repair symbols from a source block that consists of K source symbols placed into an extended source block C'. Figure 4 shows the encoding overview.

The first step of encoding is to construct an extended source block by adding zero or more padding symbols such that the total number of symbols, K', is one of the values listed in Section 5.6. Each padding symbol consists of T octets where the value of each octet is zero. K' MUST be selected as the smallest value of K' from the table of Section 5.6 which is greater than or equal to K.

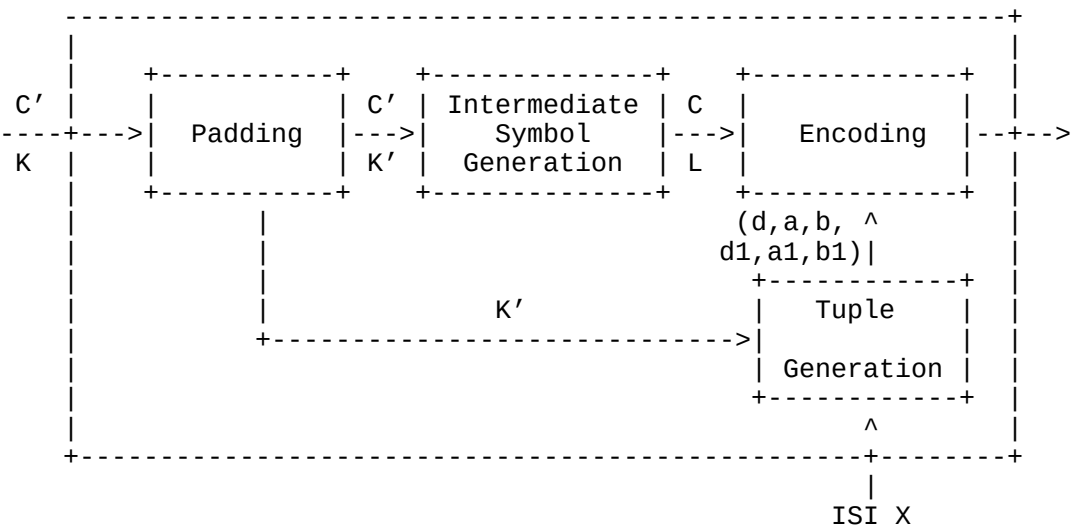

Figure 4: Encoding Overview

Let C'[0], ..., C'[K-1] denote the K source symbols.

Let C'[K], ..., C'[K'-1] denote the K'-K padding symbols, which are all set to zero bits. Then, C'[0],..., C'[K'-1] are the symbols of the extended source block upon which encoding and decoding are performed.

In the remainder of this description these padding symbols will be considered as additional source symbols and referred to as such. However, these padding symbols are not part of the encoding symbols, i.e., they are not sent as part of the encoding. At a receiver, the value of K' can be computed based on K, then the receiver can insert K'-K padding symbols at the end of a source block of K' source symbols and recover the remaining K source symbols of the source block from received encoding symbols.

The second step of encoding is to generate a number, L > K', of intermediate symbols from the K' source symbols. In this step, K' source tuples (d[0], a[0], b[0], d1[0], a1[0], b1[0]), ..., (d[K'-1], a[K'-1], b[K'-1], d1[K'-1], a1[K'-1], b1[K'-1]) are generated using the Tuple[] generator as described in Section 5.3.5.4. The K' source tuples and the ISIs associated with the K' source symbols are used to determine L intermediate symbols C[0],..., C[L-1] from the source symbols using an inverse encoding process. This process can be realized by a RaptorQ decoding process.

Certain "pre-coding relationships" must hold within the L intermediate symbols. Section 5.3.3.3 describes these relationships. Section 5.3.3.4 describes how the intermediate symbols are generated from the source symbols.

Once the intermediate symbols have been generated, repair symbols can be produced. For a repair symbol with ISI X>K', the tuple of non-negative integers, (d, a, b, d1, a1, b1) can be generated, using the Tuple[] generator as described in Section 5.3.5.4. Then, the (d, a, b, d1, a1, b1)-tuple and the ISI X is used to generate the corresponding repair symbol from the intermediate symbols using the Enc[] generator described in Section 5.3.5.3. The corresponding ESI for this repair symbol is then X-(K'-K). Note that source symbols of the extended source block can also be generated using the same process, i.e., for any X < K', the symbol generated using this process has the same value as C'[X].

5.3.3. First encoding step: Intermediate Symbol Generation

5.3.3.1. General

This encoding step is a pre-coding step to generate the L intermediate symbols C[0], ..., C[L-1] from the source symbols C'[0], ..., C'[K'-1], , where L > K' is defined in Section 5.3.3.3. The intermediate symbols are uniquely defined by two sets of constraints:

1. The intermediate symbols are related to the source symbols by a set of source symbol tuples and by the ISIs of the source symbols. The generation of the source symbol tuples is defined in Section 5.3.3.2 using the the Tuple[] generator as described in Section 5.3.5.4.

2. A number of pre-coding relationships hold within the intermediate symbols themselves. These are defined in Section 5.3.3.3

The generation of the L intermediate symbols is then defined in Section 5.3.3.4.

5.3.3.2. Source symbol tuples

Each of the K' source symbols is associated with a source symbol tuple (d[X], a[X], b[X], d1[X], a1[X], b1[X]) for 0 <= X < K'. The source symbol tuples are determined using the Tuple generator defined in Section 5.3.5.4 as:

For each X, 0 <= X < K'

(d[X], a[X], b[X], d1[X], a1[X], b1[X]) = Tuple[K, X]

5.3.3.3. Pre-coding relationships

The pre-coding relationships amongst the L intermediate symbols are defined by requiring that a set of S+H linear combinations of the intermediate symbols evaluate to zero. There are S LDPC and H HDPC symbols, and thus L = K'+S+H. Another partition of the L intermediate symbols is into two sets, one set of W LT symbols and another set of P PI symbols, and thus it is also the case that L = W+P. The P PI symbols are treated differently than the W LT symbols in the encoding process. The P PI symbols consist of the H HDPC symbols together with a set of U = P-H of the other K' intermediate symbols. The W LT symbols consist of the S LDPC symbols together with W-S of the other K' intermediate symbols. The values of these parameters are determined from K' as described below where H(K'), S(K'), and W(K') are derived from Table 2 in Section 5.6.

Let

- S = S(K')
- H = H(K')
- W = W(K')
- L = K' + S + H
- P = L - W
- P1 denote the smallest prime number greater than or equal to P
- U = P - H
- B = W - S
- C[0], ..., C[B-1] denote the intermediate symbols that are LT symbols but not LDPC symbols.
- C[B], ..., C[B+S-1] denote the S LDPC symbols that are also LT symbols.
- C[W], ..., C[W+U-1] denote the intermediate symbols that are PI symbols but not HDPC symbols.
- C[L-H], ..., C[L-1] denote the H HDPC symbols that are also PI symbols.

The first set of pre-coding relations, called LDPC relations, is described below and requires that at the end of this process the set of symbols D[0] , ..., D[S-1] are all zero:

- Initialize the symbols D[0] = C[B], ..., D[S-1] = C[B+S-1].
- For i = 0, ..., B-1 do
  - a = 1 + floor(i/S)
  - b = i % S
  - D[b] = D[b] + C[i]
  - b = (b + a) % S
  - D[b] = D[b] + C[i]
  - b = (b + a) % S
  - D[b] = D[b] + C[i]
- For i = 0, ..., S-1 do
  - a = i % P
  - b = (i+1) % P
  - D[i] = D[i] + C[W+a] + C[W+b]

Recall that the addition of symbols is to be carried out as specified in Section 5.7.

Note that the LDPC relations as defined in the algorithm above are linear, so there exists an S x B matrix G_LDPC,1 and an S x P matrix G_LDPC,2 such that

$$G_{LDPC,1} * \text{Transpose}[(C[0],...., C[B-1])] + G_{LDPC,2} * \text{Transpose}(C[W], ..., C[W+P-1]) + \text{Transpose}[(C[B], ..., C[B+S-1])] = 0$$

(The matrix G_LDPC,1 is defined by the first loop in the above algorithm, and G_LDPC,2 can be deduced from the second loop.)

The second set of relations among the intermediate symbols C[0], ..., C[L-1] are the HDPC relations and they are defined as follows:

Let

- alpha denote the octet represented by integer 2 as defined in Section 5.7.

- MT denote an H x (K' + S) matrix of octets, where for j=0,...,K'+S-2 the entry MT[i,j] is the octet represented by the integer 1 if i= Rand[j+1,6,H] or i = (Rand[j+1,6,H] + Rand[j+1,7,H-1] + 1) % H and MT[i,j] is the zero element for all other values of i, and for j=K'+S-1, MT[i,j] = alpha^^i for i=0,...,H-1.

- GAMMA denote a (K'+S) x (K'+S) matrix of octets, where

  GAMMA[i,j] =

  alpha ^^ (i-j) for i >= j,

  0 otherwise.

Then the relationship between the first K'+S intermediate symbols C[0], ..., C[K'+S-1] and the H HDPC symbols C[K'+S], ..., C[K'+S+H-1] is given by:

Transpose[C[K'+S], ..., C[K'+S+H-1]] + MT * GAMMA * Transpose[C[0], ..., C[K'+S-1]] = 0,

where '*' represents standard matrix multiplication utilizing the octet multiplication to define the multiplication between a matrix of octets and a matrix of symbols (in particular the column vector of symbols) and '+' denotes addition over octet vectors.

5.3.3.4. Intermediate symbols

5.3.3.4.1. Definition

Given the K' source symbols C'[0], C'[1],..., C'[K'-1] the L intermediate symbols C[0], C[1],..., C[L-1] are the uniquely defined symbol values that satisfy the following conditions:

1. The K' source symbols C'[0], C'[1],..., C'[K'-1] satisfy the K' constraints

   C'[X] = Enc[K', (C[0],..., C[L-1]), (d[X], a[X], b[X], d1[X], a1[X], b1[X])], for all X, 0 <= X < K',

   where (d[X], a[X], b[X], d1[X], a1[X], b1[X])) = Tuple[K',X], Tuple[] is defined in Section 5.3.5.4 and Enc[] is described in Section 5.3.5.3.

2. The L intermediate symbols C[0], C[1],..., C[L-1] satisfy the pre-coding relationships defined in Section 5.3.3.3

5.3.3.4.2. Example method for calculation of intermediate symbols

This section describes a possible method for calculation of the L intermediate symbols C[0], C[1],..., C[L-1] satisfying the constraints in Section 5.3.3.4.1

The L intermediate symbols can be calculated as follows:

Let

- C denote the column vector of the L intermediate symbols, C[0], C[1],..., C[L-1].

- D denote the column vector consisting of S+H zero symbols followed by the K' source symbols C'[0], C'[1], ..., C'[K'-1].

Then the above constraints define an L x L matrix A of octets such that:

A*C = D

The matrix A can be constructed as follows:

Let:

- G_LDPC,1 and G_LDPC,2 be S x B and S x P matrices as defined in Section 5.3.3.3.

- G_HDPC be the H x (K'+S) matrix such that

  G_HDPC * Transpose(C[0], ..., C[K'+S-1]) = Transpose(C[K'+S], ..., C[L-1]),

  i.e. G_HDPC = MT*GAMMA

- I_S be the S x S identity matrix

- I_H be the H x H identity matrix

- G_ENC be the K' x L matrix such that

  G_ENC * Transpose[(C[0], ..., C[L-1])] = Transpose[(C'[0],C'[1],...,C'[K'-1])],

i.e. G_ENC[i,j] = 1 if and only if C[j] is included in the symbols which are summed to produce Enc[K', (C[0], ..., C[L-1]), (d[i], a[i], b[i], d1[i], a1[i], b1[i])] and G_ENC[i,j] = 0 otherwise.

Then:

- The first S rows of A are equal to G_LDPC,1 | I_S | G_LDPC,2.
- The next H rows of A are equal to G_HDPC | I_H.
- The remaining K' rows of A are equal to G_ENC.

The matrix A is depicted in Figure (Figure 5) below:

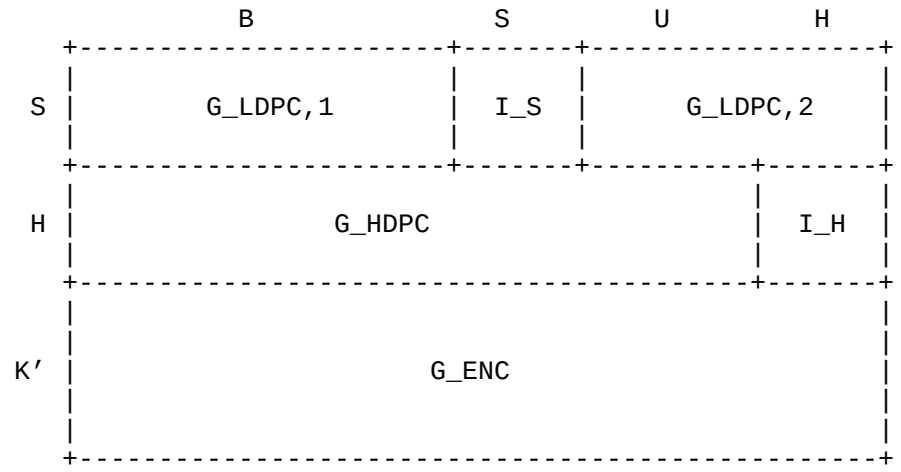

Figure 5: The matrix A

The intermediate symbols can then be calculated as:

C = (A^^-1)*D

The source tuples are generated such that for any K' matrix A has full rank and is therefore invertible. This calculation can be realized by applying a RaptorQ decoding process to the K' source symbols C'[0], C'[1],..., C'[K'-1] to produce the L intermediate symbols C[0], C[1],..., C[L-1].

To efficiently generate the intermediate symbols from the source symbols, it is recommended that an efficient decoder implementation such as that described in Section 5.4 be used.

5.3.4. Second encoding step: Encoding

In the second encoding step, the repair symbol with ISI X (X >= K') is generated by applying the generator Enc[K', (C[0], C[1],..., C[L-1]), (d, a, b, d1, a1, b1)] defined in Section 5.3.5.3 to the L intermediate symbols C[0], C[1],..., C[L-1] using the tuple (d, a, b, d1, a1, b1)=Tuple[K',X].

5.3.5. Generators

5.3.5.1. Random Number Generator

The random number generator Rand[y, i, m] is defined as follows, where y is a non-negative integer, i is a non-negative integer less than 256, and m is a positive integer and the value produced is an integer between 0 and m-1. Let V0, V1, V2 and V3 be the arrays provided in Section 5.5.

Let

- x0 = (y + i) mod 2^^8
- x1 = (floor(y / 2^^8) + i) mod 2^^8
- x2 = (floor(y / 2^^16) + i) mod 2^^8
- x3 = (floor(y / 2^^24) + i) mod 2^^8

Then

Rand[y, i, m] = (V0[x0] ^ V1[x1] ^ V2[x2] ^ V3[x3]) % m

5.3.5.2. Degree Generator

The degree generator Deg[v] is defined as follows, where v is a non-negative integer that is less than 2^^20 = 1048576. Given v, find index d in Table 1 such that f[d-1] <= v < f[d], and set Deg[v] = min(d, W-2).). Recall that W is derived from K' as described in Section 5.3.3.3.

Index d	f[d]	Index d	f[d]
0	0	1	5243
2	529531	3	704294
4	791675	5	844104
6	879057	7	904023
8	922747	9	937311
10	948962	11	958494
12	966438	13	973160
14	978921	15	983914
16	988283	17	992138
18	995565	19	998631
20	1001391	21	1003887
22	1006157	23	1008229
24	1010129	25	1011876
26	1013490	27	1014983
28	1016370	29	1017662
30	1048576		

Table 1: Defines the degree distribution for encoding symbols

5.3.5.3. Encoding Symbol Generator

The encoding symbol generator Enc[K', (C[0], C[1],..., C[L-1]), (d, a, b, d1, a1, b1)] takes the following inputs:

- K' is the number of source symbols for the extended source block. Let L, W, B, S, P and P1 be derived from K' as described in Section 5.3.3.3.

- (C[0], C[1],..., C[L-1]) is the array of L intermediate symbols (sub-symbols) generated as described in Section 5.3.3.4
- (d, a, b, d1, a1, b1) is a source tuple determined from ISI X using the Tuple generator defined in Section 5.3.5.4, whereby
  - d is a positive integer denoting an encoding symbol LT degree
  - a is a positive integer between 1 and W-1 inclusive
  - b is a non-negative integer between 0 and W-1 inclusive
  - d1 is a positive integer that has value either 2 or 3 inclusive denoting an encoding symbol PI degree
  - a1 is a positive integer between 1 and P1-1 inclusive
  - b1 is a non-negative integer between 0 and P1-1 inclusive

The encoding symbol generator produces a single encoding symbol as output (referred to as result), according to the following algorithm:

- result = C[b]
- For j = 1, ..., d-1 do
  - b = (b + a) % W
  - result = result + C[b]
- While (b1 >= P) do b1 = (b1+a1) % P1
- result = result + C[W+b1]
- For j = 1, ..., d1-1 do
  - b1 = (b1 + a1) % P1
  - While (b1 >= P) do b1 = (b1+a1) % P1
  - result = result + C[W+b1]
- Return result

5.3.5.4. Tuple generator

The tuple generator Tuple[K',X] takes the following inputs:

- K' - The number of source symbols in the extended source block
- X - An ISI

Let

- L be determined from K' as described in Section 5.3.3.3
- J=J(K') be the systematic index associated with K', as defined in Table 2 inSection 5.6

The output of tuple generator is a tuple, (d, a, b, d1, a1, b1), determined as follows:

- A = 53591 + J*997
- if (A % 2 == 0) { A = A + 1 }
- B = 10267*(J+1)
- y = (B + X*A) % 2^^32
- v = Rand[y, 0, 2^^20]
- d = Deg[v]
- a = 1 + Rand[y, 1, W-1]
- b = Rand[y, 2, W]
- If (d<4) { d1 = 2 + Rand[X, 3, 2] } else { d1 = 2 }
- a1 = 1 + Rand[X, 4, P1-1]
- b1 = Rand[X, 5, P1]

5.4. Example FEC decoder

5.4.1. General

This section describes an efficient decoding algorithm for the RaptorQ code introduced in this specification. Note that each received encoding symbol is a known linear combination of the intermediate symbols. So each received encoding symbol provides a

linear equation among the intermediate symbols, which, together with the known linear pre-coding relationships amongst the intermediate symbols gives a system of linear equations. Thus, any algorithm for solving systems of linear equations can successfully decode the intermediate symbols and hence the source symbols. However, the algorithm chosen has a major effect on the computational efficiency of the decoding.

5.4.2. Decoding an extended source block

5.4.2.1. General

It is assumed that the decoder knows the structure of the source block it is to decode, including the symbol size, T, and the number K of symbols in the source block and the number K' of source symbols in the extended source block.

From the algorithms described in Section 5.3, the RaptorQ decoder can calculate the total number L = K'+S+H of intermediate symbols and determine how they were generated from the extended source block to be decoded. In this description it is assumed that the received encoding symbols for the extended source block to be decoded are passed to the decoder. Furthermore, for each such encoding symbol it is assumed that the number and set of intermediate symbols whose sum is equal to the encoding symbol are passed to the decoder. In the case of source symbols, including padding symbols, the source symbol tuples described in Section 5.3.3.2 indicate the number and set of intermediate symbols which sum to give each source symbol.

Let N >= K' be the number of received encoding symbols to be used for decoding, including padding symbols for an extended source block and let M = S+H+N. Then with the notation of Section 5.3.3.4.2 we have A*C=D.

Decoding an extended source block is equivalent to decoding C from known A and D. It is clear that C can be decoded if and only if the rank of A is L. Once C has been decoded, missing source symbols can be obtained by using the source symbol tuples to determine the number and set of intermediate symbols which must be summed to obtain each missing source symbol.

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 a multiple, beta, of row i of A is added to row i' in the decoding schedule then in the decoding process the symbol beta*D[d[i]] is added to symbol D[d[i']] .
- Each time a row i of A is multiplied by an octet beta, then in the decoding process the symbol D[d[i]] is also multiplied by beta.
- 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 operations on symbols in the decoding of the extended 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 is not described here). The remainder of this section describes an order in which Gaussian elimination could be performed that is relatively efficient.

5.4.2.2. First Phase

In the first phase of the Gaussian elimination the matrix A is conceptually partitioned into submatrices and additionally, a matrix X is created. This matrix has as many rows and columns as A, and it will be a lower triangular matrix throughout the first phase. At the beginning of this phase, the matrix A is copied into the matrix X. The submatrix sizes are parameterized by non-negative integers i and

u which are initialized to 0 and P, the number of PI symbols, respectively. 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 V formed by the intersection of all but the first i columns and the last u columns and all but the first i rows.

Figure 6 illustrates the submatrices of A. At the beginning of the first phase V consists of the first L-P columns of A and U consists of the last P columns corresponding to the PI symbols. In each step, a row of A is chosen.

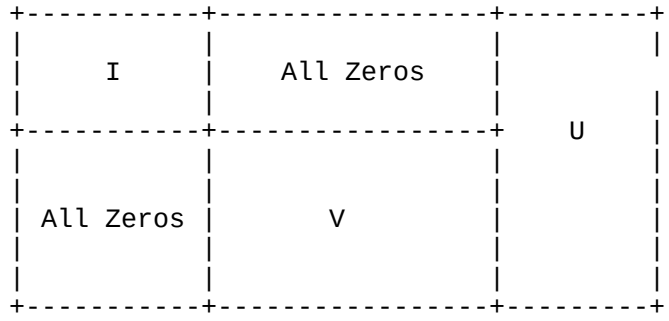

Figure 6: Submatrices of A in the first phase

The following graph defined by the structure of V is used in determining which row of A is chosen. The columns that intersect V are the nodes in the graph, and the rows that have exactly 2 non-zero entries in V and are not HDPC rows 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.

There are at most L steps in the first phase. The phase ends successfully when i + u = L, i.e., when V and the all zeroes submatrix above V 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 V disappears there is no non-zero row in V to choose in that step. In each step, a row of A is chosen as follows:

- If all entries of V 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 non-zeroes in V.
  - If r != 2 then choose a row with exactly r non-zeroes in V with minimum original degree among all such rows, except that HDPC rows should not be chosen until all non-HDPC rows have been processed.
  - If r = 2 and there is a row with exactly 2 ones in V then choose any row with exactly 2 ones in V that is part of a maximum size component in the graph described above that is defined by V.
  - If r = 2 and there is no row with exactly 2 ones in V then choose any row with exactly 2 non-zeroes in V.

After the row is chosen in this step the first row of A that intersects V is exchanged with the chosen row so that the chosen row is the first row that intersects V. The columns of A among those that intersect V are reordered so that one of the r non-zeroes in the chosen row appears in the first column of V and so that the remaining r-1 non-zeroes appear in the last columns of V. The same row and column operations are also performed on the matrix X. Then, an appropriate multiple of the chosen row is added to all the other rows of A below the chosen row that have a non-zero entry in the first column of V. Specifically, if a row below the chosen row has entry beta in the first column of V, and the chosen row has entry alpha in the first column of V, then beta/alpha multiplied by the chosen row is added to this row to leave a zero value in the first column of V. Finally, i is incremented by 1 and u is incremented by r-1, which completes the step.

Note that efficiency can be improved if the row operations identified above are not actually performed until the affected row is itself chosen during the decoding process. This avoids processing of row

operations for rows which are not eventually used in the decoding process and in particular avoid those rows for which beta!=1 until they are actually required. Furthermore, the row operations required for the HDPC rows may be performed for all such rows in one process, by using the algorithm described in Section 5.3.3.3.

5.4.2.3. Second Phase

At this point, all the entries of X outside the first i rows and i columns are discarded, so that X has lower triangular form. The last i rows and columns of X are discarded, so that X now has i rows i columns. The submatrix U is further partitioned into the first i rows, U_upper, and the remaining M - i rows, U_lower. Gaussian elimination is performed in the second phase on U_lower 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 I_u. The M - L rows of A that intersect U_lower - I_u are discarded. After this phase A has L rows and L columns.

5.4.2.4. 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 U_upper. The number of rows i of the submatrix U_upper is generally much larger than the number of columns u of U_upper. Moreover, at this time, the matrix U_upper is typically dense, i.e., the number of nonzero entries of this matrix is large. To reduce this matrix to a sparse form, the sequence of operations performed to obtain the matrix U_lower needs to be inverted. To this end, the matrix X is multiplied with the submatrix of A consisting of the first i rows of A. After this operation the submatrix of A consisting of the intersection of the first i rows and columns equals to X, whereas the matrix U_upper is transformed to a sparse form.

5.4.2.5. Fourth Phase

For each of the first i rows of U_upper do the following: if the row has a nonzero entry at position j, and if the value of that nonzero entry is b, then add to this row b times row j of I_u. After this step, the submatrix of A consisting of the intersection of the first i rows and columns is equal to X, the submatrix U_upper consists of zeros, the submatrix consisting of the intersection of the last u rows and the first i columns consists of zeros, and the submatrix consisting of the last u rows and columns is is the matrix I_u.

5.4.2.6. Fifth Phase

For j from 1 to i perform the following operations:

1. if A[j,j] is not one, then divide row j of A by A[j,j].

2. For l from 1 to j-1, if A[j,l] is nonzero, then add A[j,l] multiplied with row l of A to row j of A.

After this phase A is the L by L identity matrix and a complete decoding schedule has been successfully formed. Then, the corresponding decoding consisting of summing known encoding symbols can be executed to recover the intermediate symbols based on the decoding schedule. The tuples associated with all source symbols are computed according to Section 5.3.3.2. The tuples for received source symbols are used in the decoding. The tuples for missing source symbols are used to determine which intermediate symbols need to be summed to recover the missing source symbols.

5.5. Random Numbers

The four arrays V0, V1, V2 and V3 used in Section 5.3.5.1 are provided below. There are 256 entries in each of the four arrays. The indexing into each array starts at 0, and the entries are 32-bit unsigned integers.

5.5.1. The table V0

251291136, 3952231631, 3370958628, 4070167936, 123631495, 3351110283,
3218676425, 2011642291, 774603218, 2402805061, 1004366930,
1843948209, 428891132, 3746331984, 1591258008, 3067016507,
1433388735, 504005498, 2032657933, 3419319784, 2805686246,
3102436986, 3808671154, 2501582075, 3978944421, 246043949,
4016898363, 649743608, 1974987508, 2651273766, 2357956801, 689605112,
715807172, 2722736134, 191939188, 3535520147, 3277019569, 1470435941,
3763101702, 3232409631, 122701163, 3920852693, 782246947, 372121310,
2995604341, 2045698575, 2332962102, 4005368743, 218596347,
3415381967, 4207612806, 861117671, 3676575285, 2581671944,
3312220480, 681232419, 307306866, 4112503940, 1158111502, 709227802,
2724140433, 4201101115, 4215970289, 4048876515, 3031661061,
1909085522, 510985033, 1361682810, 129243379, 3142379587, 2569842483,
3033268270, 1658118006, 932109358, 1982290045, 2983082771,
3007670818, 3448104768, 683749698, 778296777, 1399125101, 1939403708,
1692176003, 3868299200, 1422476658, 593093658, 1878973865,
2526292949, 1591602827, 3986158854, 3964389521, 2695031039,
1942050155, 424618399, 1347204291, 2669179716, 2434425874,
2540801947, 1384069776, 4123580443, 1523670218, 2708475297,
1046771089, 2229796016, 1255426612, 4213663089, 1521339547,

3041843489, 420130494, 10677091, 515623176, 3457502702, 2115821274,
2720124766, 3242576090, 854310108, 425973987, 325832382, 1796851292,
2462744411, 1976681690, 1408671665, 1228817808, 3917210003,
263976645, 2593736473, 2471651269, 4291353919, 650792940, 1191583883,
3046561335, 2466530435, 2545983082, 969168436, 2019348792,
2268075521, 1169345068, 3250240009, 3963499681, 2560755113,
911182396, 760842409, 3569308693, 2687243553, 381854665, 2613828404,
2761078866, 1456668111, 883760091, 3294951678, 1604598575,
1985308198, 1014570543, 2724959607, 3062518035, 3115293053,
138853680, 4160398285, 3322241130, 2068983570, 2247491078,
3669524410, 1575146607, 828029864, 3732001371, 3422026452,
3370954177, 4006626915, 543812220, 1243116171, 3928372514,
2791443445, 4081325272, 2280435605, 885616073, 616452097, 3188863436,
2780382310, 2340014831, 1208439576, 258356309, 3837963200,
2075009450, 3214181212, 3303882142, 880813252, 1355575717, 207231484,
2420803184, 358923368, 1617557768, 3272161958, 1771154147,
2842106362, 1751209208, 1421030790, 658316681, 194065839, 3241510581,
38625260, 301875395, 4176141739, 297312930, 2137802113, 1502984205,
3669376622, 3728477036, 234652930, 2213589897, 2734638932,
1129721478, 3187422815, 2859178611, 3284308411, 3819792700,
3557526733, 451874476, 1740576081, 3592838701, 1709429513,
3702918379, 3533351328, 1641660745, 179350258, 2380520112,
3936163904, 3685256204, 3156252216, 1854258901, 2861641019,
3176611298, 834787554, 331353807, 517858103, 3010168884, 4012642001,
2217188075, 3756943137, 3077882590, 2054995199, 3081443129,
3895398812, 1141097543, 2376261053, 2626898255, 2554703076,
401233789, 1460049922, 678083952, 1064990737, 940909784, 1673396780,
528881783, 1712547446, 3629685652, 1358307511

5.5.2. The table V1

807385413, 2043073223, 3336749796, 1302105833, 2278607931, 541015020,
1684564270, 372709334, 3508252125, 1768346005, 1270451292,
2603029534, 2049387273, 3891424859, 2152948345, 4114760273,
915180310, 3754787998, 700503826, 2131559305, 1308908630, 224437350,
4065424007, 3638665944, 1679385496, 3431345226, 1779595665,
3068494238, 1424062773, 1033448464, 4050396853, 3302235057,
420600373, 2868446243, 311689386, 259047959, 4057180909, 1575367248,
4151214153, 110249784, 3006865921, 4293710613, 3501256572, 998007483,
499288295, 1205710710, 2997199489, 640417429, 3044194711, 486690751,
2686640734, 2394526209, 2521660077, 49993987, 3843885867, 4201106668,
415906198, 19296841, 2402488407, 2137119134, 1744097284, 579965637,
2037662632, 852173610, 2681403713, 1047144830, 2982173936, 910285038,
4187576520, 2589870048, 989448887, 3292758024, 506322719, 176010738,
1865471968, 2619324712, 564829442, 1996870325, 339697593, 4071072948,
3618966336, 2111320126, 1093955153, 957978696, 892010560, 1854601078,
1873407527, 2498544695, 2694156259, 1927339682, 1650555729,
183933047, 3061444337, 2067387204, 228962564, 3904109414, 1595995433,

1780701372, 2463145963, 307281463, 3237929991, 3852995239,
2398693510, 3754138664, 522074127, 146352474, 4104915256, 3029415884,
3545667983, 332038910, 976628269, 3123492423, 3041418372, 2258059298,
2139377204, 3243642973, 3226247917, 3674004636, 2698992189,
3453843574, 1963216666, 3509855005, 2358481858, 747331248,
1957348676, 1097574450, 2435697214, 3870972145, 1888833893,
2914085525, 4161315584, 1273113343, 3269644828, 3681293816,
412536684, 1156034077, 3823026442, 1066971017, 3598330293,
1979273937, 2079029895, 1195045909, 1071986421, 2712821515,
3377754595, 2184151095, 750918864, 2585729879, 4249895712,
1832579367, 1192240192, 946734366, 31230688, 3174399083, 3549375728,
1642430184, 1904857554, 861877404, 3277825584, 4267074718,
3122860549, 666423581, 644189126, 226475395, 307789415, 1196105631,
3191691839, 782852669, 1608507813, 1847685900, 4069766876,
3931548641, 2526471011, 766865139, 2115084288, 4259411376,
3323683436, 568512177, 3736601419, 1800276898, 4012458395, 1823982,
27980198, 2023839966, 869505096, 431161506, 1024804023, 1853869307,
3393537983, 1500703614, 3019471560, 1351086955, 3096933631,
3034634988, 2544598006, 1230942551, 3362230798, 159984793, 491590373,
3993872886, 3681855622, 903593547, 3535062472, 1799803217, 772984149,
895863112, 1899036275, 4187322100, 101856048, 234650315, 3183125617,
3190039692, 525584357, 1286834489, 455810374, 1869181575, 922673938,
3877430102, 3422391938, 1414347295, 1971054608, 3061798054,
830555096, 2822905141, 167033190, 1079139428, 4210126723, 3593797804,
429192890, 372093950, 1779187770, 3312189287, 204349348, 452421568,
2800540462, 3733109044, 1235082423, 1765319556, 3174729780,
3762994475, 3171962488, 442160826, 198349622, 45942637, 1324086311,
2901868599, 678860040, 3812229107, 19936821, 1119590141, 3640121682,
3545931032, 2102949142, 2828208598, 3603378023, 4135048896

5.5.3. The table V2

1629829892, 282540176, 2794583710, 496504798, 2990494426, 3070701851,
2575963183, 4094823972, 2775723650, 4079480416, 176028725,
2246241423, 3732217647, 2196843075, 1306949278, 4170992780,
4039345809, 3209664269, 3387499533, 293063229, 3660290503,
2648440860, 2531406539, 3537879412, 773374739, 4184691853,
1804207821, 3347126643, 3479377103, 3970515774, 1891731298,
2368003842, 3537588307, 2969158410, 4230745262, 831906319,
2935838131, 264029468, 120852739, 3200326460, 355445271, 2296305141,
1566296040, 1760127056, 20073893, 3427103620, 2866979760, 2359075957,
2025314291, 1725696734, 3346087406, 2690756527, 99815156, 4248519977,
2253762642, 3274144518, 598024568, 3299672435, 556579346, 4121041856,
2896948975, 3620123492, 918453629, 3249461198, 2231414958,
3803272287, 3657597946, 2588911389, 242262274, 1725007475,
2026427718, 46776484, 2873281403, 2919275846, 3177933051, 1918859160,
2517854537, 1857818511, 3234262050, 479353687, 200201308, 2801945841,
1621715769, 483977159, 423502325, 3689396064, 1850168397, 3359959416,

3459831930, 841488699, 3570506095, 930267420, 1564520841, 2505122797,
593824107, 1116572080, 819179184, 3139123629, 1414339336, 1076360795,
512403845, 177759256, 1701060666, 2239736419, 515179302, 2935012727,
3821357612, 1376520851, 2700745271, 966853647, 1041862223, 715860553,
171592961, 1607044257, 1227236688, 3647136358, 1417559141,
4087067551, 2241705880, 4194136288, 1439041934, 20464430, 119668151,
2021257232, 2551262694, 1381539058, 4082839035, 498179069, 311508499,
3580908637, 2889149671, 142719814, 1232184754, 3356662582,
2973775623, 1469897084, 1728205304, 1415793613, 50111003, 3133413359,
4074115275, 2710540611, 2700083070, 2457757663, 2612845330,
3775943755, 2469309260, 2560142753, 3020996369, 1691667711,
4219602776, 1687672168, 1017921622, 2307642321, 368711460,
3282925988, 213208029, 4150757489, 3443211944, 2846101972,
4106826684, 4272438675, 2199416468, 3710621281, 497564971, 285138276,
765042313, 916220877, 3402623607, 2768784621, 1722849097, 3386397442,
487920061, 3569027007, 3424544196, 217781973, 2356938519, 3252429414,
145109750, 2692588106, 2454747135, 1299493354, 4120241887,
2088917094, 932304329, 1442609203, 952586974, 3509186750, 753369054,
854421006, 1954046388, 2708927882, 4047539230, 3048925996,
1667505809, 805166441, 1182069088, 4265546268, 4215029527,
3374748959, 373532666, 2454243090, 2371530493, 3651087521,
2619878153, 1651809518, 1553646893, 1227452842, 703887512,
3696674163, 2552507603, 2635912901, 895130484, 3287782244,
3098973502, 990078774, 3780326506, 2290845203, 41729428, 1949580860,
2283959805, 1036946170, 1694887523, 4880696, 466000198, 2765355283,
3318686998, 1266458025, 3919578154, 3545413527, 2627009988,
3744680394, 1696890173, 3250684705, 4142417708, 915739411,
3308488877, 1289361460, 2942552331, 1169105979, 3342228712,
698560958, 1356041230, 2401944293, 107705232, 3701895363, 903928723,
3646581385, 844950914, 1944371367, 3863894844, 2946773319,
1972431613, 1706989237, 29917467, 3497665928

5.5.4. The table V3

1191369816, 744902811, 2539772235, 3213192037, 3286061266,
1200571165, 2463281260, 754888894, 714651270, 1968220972, 3628497775,
1277626456, 1493398934, 364289757, 2055487592, 3913468088,
2930259465, 902504567, 3967050355, 2056499403, 692132390, 186386657,
832834706, 859795816, 1283120926, 2253183716, 3003475205, 1755803552,
2239315142, 4271056352, 2184848469, 769228092, 1249230754,
1193269205, 2660094102, 642979613, 1687087994, 2726106182, 446402913,
4122186606, 3771347282, 37667136, 192775425, 3578702187, 1952659096,
3989584400, 3069013882, 2900516158, 4045316336, 3057163251,
1702104819, 4116613420, 3575472384, 2674023117, 1409126723,
3215095429, 1430726429, 2544497368, 1029565676, 1855801827,
4262184627, 1854326881, 2906728593, 3277836557, 2787697002,
2787333385, 3105430738, 2477073192, 748038573, 1088396515,
1611204853, 201964005, 3745818380, 3654683549, 3816120877,

3915783622, 2563198722, 1181149055, 33158084, 3723047845, 3790270906,
3832415204, 2959617497, 372900708, 1286738499, 1932439099,
3677748309, 2454711182, 2757856469, 2134027055, 2780052465,
3190347618, 3758510138, 3626329451, 1120743107, 1623585693,
1389834102, 2719230375, 3038609003, 462617590, 260254189, 3706349764,
2556762744, 2874272296, 2502399286, 4216263978, 2683431180,
2168560535, 3561507175, 668095726, 680412330, 3726693946, 4180630637,
3335170953, 942140968, 2711851085, 2059233412, 4265696278,
3204373534, 232855056, 881788313, 2258252172, 2043595984, 3758795150,
3615341325, 2138837681, 1351208537, 2923692473, 3402482785,
2105383425, 2346772751, 499245323, 3417846006, 2366116814,
2543090583, 1828551634, 3148696244, 3853884867, 1364737681,
2200687771, 2689775688, 232720625, 4071657318, 2671968983,
3531415031, 1212852141, 867923311, 3740109711, 1923146533,
3237071777, 3100729255, 3247856816, 906742566, 4047640575,
4007211572, 3495700105, 1171285262, 2835682655, 1634301229,
3115169925, 2289874706, 2252450179, 944880097, 371933491, 1649074501,
2208617414, 2524305981, 2496569844, 2667037160, 1257550794,
3399219045, 3194894295, 1643249887, 342911473, 891025733, 3146861835,
3789181526, 938847812, 1854580183, 2112653794, 2960702988,
1238603378, 2205280635, 1666784014, 2520274614, 3355493726,
2310872278, 3153920489, 2745882591, 1200203158, 3033612415,
2311650167, 1048129133, 4206710184, 4209176741, 2640950279,
2096382177, 4116899089, 3631017851, 4104488173, 1857650503,
3801102932, 445806934, 3055654640, 897898279, 3234007399, 1325494930,
2982247189, 1619020475, 2720040856, 885096170, 3485255499,
2983202469, 3891011124, 546522756, 1524439205, 2644317889,
2170076800, 2969618716, 961183518, 1081831074, 1037015347,
3289016286, 2331748669, 620887395, 303042654, 3990027945, 1562756376,
3413341792, 2059647769, 2823844432, 674595301, 2457639984,
4076754716, 2447737904, 1583323324, 625627134, 3076006391, 345777990,
1684954145, 879227329, 3436182180, 1522273219, 3802543817,
1456017040, 1897819847, 2970081129, 1382576028, 3820044861,
1044428167, 612252599, 3340478395, 2150613904, 3397625662,
3573635640, 3432275192

5.6. Systematic indices and other parameters

Table 2 below specifies the supported values of K'. The table also specifies for each supported value of K' the systematic index J(K'), the number H(K') of HDPC symbols, the number S(K') of LDPC symbols, and the number W(K') of LT symbols. For each value of K', the corresponding values of S(K') and W(K') are prime numbers.

The systematic index J(K') is designed to have the property that the set of source symbol tuples (d[0], a[0], b[0], d1[0], a1[0], b1[0]), ..., (d[K'-1], a[K'-1], b[K'-1], d1[K'-1], a1[K'-1], b1[K'-1]) are such that the L intermediate symbols are uniquely defined, i.e., the

matrix A in Figure 6 has full rank and is therefore invertible.

K'	J(K')	S(K')	H(K')	W(K')
10	254	7	10	17
12	630	7	10	19
18	682	11	10	29
20	293	11	10	31
26	80	11	10	37
30	566	11	10	41
32	860	11	10	43
36	267	11	10	47
42	822	11	10	53
46	506	13	10	59
48	589	13	10	61
49	87	13	10	61
55	520	13	10	67
60	159	13	10	71
62	235	13	10	73
69	157	13	10	79
75	502	17	10	89
84	334	17	10	97
88	583	17	10	101
91	66	17	10	103
95	352	17	10	107
97	365	17	10	109

101	562	17	10	113
114	5	19	10	127
119	603	19	10	131
125	721	19	10	137
127	28	19	10	139
138	660	19	10	149
140	829	19	10	151
149	900	23	10	163
153	930	23	10	167
160	814	23	10	173
166	661	23	10	179
168	693	23	10	181
179	780	23	10	191
181	605	23	10	193
185	551	23	10	197
187	777	23	10	199
200	491	23	10	211
213	396	23	10	223
217	764	29	10	233
225	843	29	10	241
236	646	29	10	251
242	557	29	10	257
248	608	29	10	263
257	265	29	10	271

263	505	29	10	277
269	722	29	10	283
280	263	29	10	293
295	999	29	10	307
301	874	29	10	313
305	160	29	10	317
324	575	31	10	337
337	210	31	10	349
341	513	31	10	353
347	503	31	10	359
355	558	31	10	367
362	932	31	10	373
368	404	31	10	379
372	520	37	10	389
380	846	37	10	397
385	485	37	10	401
393	728	37	10	409
405	554	37	10	421
418	471	37	10	433
428	641	37	10	443
434	732	37	10	449
447	193	37	10	461
453	934	37	10	467
466	864	37	10	479

478	790	37	10	491
486	912	37	10	499
491	617	37	10	503
497	587	37	10	509
511	800	37	10	523
526	923	41	10	541
532	998	41	10	547
542	92	41	10	557
549	497	41	10	563
557	559	41	10	571
563	667	41	10	577
573	912	41	10	587
580	262	41	10	593
588	152	41	10	601
594	526	41	10	607
600	268	41	10	613
606	212	41	10	619
619	45	41	10	631
633	898	43	10	647
640	527	43	10	653
648	558	43	10	661
666	460	47	10	683
675	5	47	10	691
685	895	47	10	701

693	996	47	10	709
703	282	47	10	719
718	513	47	10	733
728	865	47	10	743
736	870	47	10	751
747	239	47	10	761
759	452	47	10	773
778	862	53	10	797
792	852	53	10	811
802	643	53	10	821
811	543	53	10	829
821	447	53	10	839
835	321	53	10	853
845	287	53	10	863
860	12	53	10	877
870	251	53	10	887
891	30	53	10	907
903	621	53	10	919
913	555	53	10	929
926	127	53	10	941
938	400	53	10	953
950	91	59	10	971
963	916	59	10	983
977	935	59	10	997

989	691	59	10	1009
1002	299	59	10	1021
1020	282	59	10	1039
1032	824	59	10	1051
1050	536	59	11	1069
1074	596	59	11	1093
1085	28	59	11	1103
1099	947	59	11	1117
1111	162	59	11	1129
1136	536	59	11	1153
1152	1000	61	11	1171
1169	251	61	11	1187
1183	673	61	11	1201
1205	559	61	11	1223
1220	923	61	11	1237
1236	81	67	11	1259
1255	478	67	11	1277
1269	198	67	11	1291
1285	137	67	11	1307
1306	75	67	11	1327
1347	29	67	11	1367
1361	231	67	11	1381
1389	532	67	11	1409
1404	58	67	11	1423

1420	60	67	11	1439
1436	964	71	11	1459
1461	624	71	11	1483
1477	502	71	11	1499
1502	636	71	11	1523
1522	986	71	11	1543
1539	950	71	11	1559
1561	735	73	11	1583
1579	866	73	11	1601
1600	203	73	11	1621
1616	83	73	11	1637
1649	14	73	11	1669
1673	522	79	11	1699
1698	226	79	11	1723
1716	282	79	11	1741
1734	88	79	11	1759
1759	636	79	11	1783
1777	860	79	11	1801
1800	324	79	11	1823
1824	424	79	11	1847
1844	999	79	11	1867
1863	682	83	11	1889
1887	814	83	11	1913
1906	979	83	11	1931

1926	538	83	11	1951
1954	278	83	11	1979
1979	580	83	11	2003
2005	773	83	11	2029
2040	911	89	11	2069
2070	506	89	11	2099
2103	628	89	11	2131
2125	282	89	11	2153
2152	309	89	11	2179
2195	858	89	11	2221
2217	442	89	11	2243
2247	654	89	11	2273
2278	82	97	11	2311
2315	428	97	11	2347
2339	442	97	11	2371
2367	283	97	11	2399
2392	538	97	11	2423
2416	189	97	11	2447
2447	438	97	11	2477
2473	912	97	11	2503
2502	1	97	11	2531
2528	167	97	11	2557
2565	272	97	11	2593
2601	209	101	11	2633

2640	927	101	11	2671
2668	386	101	11	2699
2701	653	101	11	2731
2737	669	101	11	2767
2772	431	101	11	2801
2802	793	103	11	2833
2831	588	103	11	2861
2875	777	107	11	2909
2906	939	107	11	2939
2938	864	107	11	2971
2979	627	107	11	3011
3015	265	109	11	3049
3056	976	109	11	3089
3101	988	113	11	3137
3151	507	113	11	3187
3186	640	113	11	3221
3224	15	113	11	3259
3265	667	113	11	3299
3299	24	127	11	3347
3344	877	127	11	3391
3387	240	127	11	3433
3423	720	127	11	3469
3466	93	127	11	3511
3502	919	127	11	3547

3539	635	127	11	3583
3579	174	127	11	3623
3616	647	127	11	3659
3658	820	127	11	3701
3697	56	127	11	3739
3751	485	127	11	3793
3792	210	127	11	3833
3840	124	127	11	3881
3883	546	127	11	3923
3924	954	131	11	3967
3970	262	131	11	4013
4015	927	131	11	4057
4069	957	131	11	4111
4112	726	137	11	4159
4165	583	137	11	4211
4207	782	137	11	4253
4252	37	137	11	4297
4318	758	137	11	4363
4365	777	137	11	4409
4418	104	139	11	4463
4468	476	139	11	4513
4513	113	149	11	4567
4567	313	149	11	4621
4626	102	149	11	4679

4681	501	149	11	4733
4731	332	149	11	4783
4780	786	149	11	4831
4838	99	149	11	4889
4901	658	149	11	4951
4954	794	149	11	5003
5008	37	151	11	5059
5063	471	151	11	5113
5116	94	157	11	5171
5172	873	157	11	5227
5225	918	157	11	5279
5279	945	157	11	5333
5334	211	157	11	5387
5391	341	157	11	5443
5449	11	163	11	5507
5506	578	163	11	5563
5566	494	163	11	5623
5637	694	163	11	5693
5694	252	163	11	5749
5763	451	167	11	5821
5823	83	167	11	5881
5896	689	167	11	5953
5975	488	173	11	6037
6039	214	173	11	6101

6102	17	173	11	6163
6169	469	173	11	6229
6233	263	179	11	6299
6296	309	179	11	6361
6363	984	179	11	6427
6427	123	179	11	6491
6518	360	179	11	6581
6589	863	181	11	6653
6655	122	181	11	6719
6730	522	191	11	6803
6799	539	191	11	6871
6878	181	191	11	6949
6956	64	191	11	7027
7033	387	191	11	7103
7108	967	191	11	7177
7185	843	191	11	7253
7281	999	193	11	7351
7360	76	197	11	7433
7445	142	197	11	7517
7520	599	197	11	7591
7596	576	199	11	7669
7675	176	211	11	7759
7770	392	211	11	7853
7855	332	211	11	7937

7935	291	211	11	8017
8030	913	211	11	8111
8111	608	211	11	8191
8194	212	211	11	8273
8290	696	211	11	8369
8377	931	223	11	8467
8474	326	223	11	8563
8559	228	223	11	8647
8654	706	223	11	8741
8744	144	223	11	8831
8837	83	223	11	8923
8928	743	223	11	9013
9019	187	223	11	9103
9111	654	227	11	9199
9206	359	227	11	9293
9303	493	229	11	9391
9400	369	233	11	9491
9497	981	233	11	9587
9601	276	239	11	9697
9708	647	239	11	9803
9813	389	239	11	9907
9916	80	239	11	10009
10017	396	241	11	10111
10120	580	251	11	10223

10241	873	251	11	10343
10351	15	251	11	10453
10458	976	251	11	10559
10567	584	251	11	10667
10676	267	257	11	10781
10787	876	257	11	10891
10899	642	257	12	11003
11015	794	257	12	11119
11130	78	263	12	11239
11245	736	263	12	11353
11358	882	269	12	11471
11475	251	269	12	11587
11590	434	269	12	11701
11711	204	269	12	11821
11829	256	271	12	11941
11956	106	277	12	12073
12087	375	277	12	12203
12208	148	277	12	12323
12333	496	281	12	12451
12460	88	281	12	12577
12593	826	293	12	12721
12726	71	293	12	12853
12857	925	293	12	12983
13002	760	293	12	13127

13143	130	293	12	13267
13284	641	307	12	13421
13417	400	307	12	13553
13558	480	307	12	13693
13695	76	307	12	13829
13833	665	307	12	13967
13974	910	307	12	14107
14115	467	311	12	14251
14272	964	311	12	14407
14415	625	313	12	14551
14560	362	317	12	14699
14713	759	317	12	14851
14862	728	331	12	15013
15011	343	331	12	15161
15170	113	331	12	15319
15325	137	331	12	15473
15496	308	331	12	15643
15651	800	337	12	15803
15808	177	337	12	15959
15977	961	337	12	16127
16161	958	347	12	16319
16336	72	347	12	16493
16505	732	347	12	16661
16674	145	349	12	16831

16851	577	353	12	17011
17024	305	353	12	17183
17195	50	359	12	17359
17376	351	359	12	17539
17559	175	367	12	17729
17742	727	367	12	17911
17929	902	367	12	18097
18116	409	373	12	18289
18309	776	373	12	18481
18503	586	379	12	18679
18694	451	379	12	18869
18909	287	383	12	19087
19126	246	389	12	19309
19325	222	389	12	19507
19539	563	397	12	19727
19740	839	397	12	19927
19939	897	401	12	20129
20152	409	401	12	20341
20355	618	409	12	20551
20564	439	409	12	20759
20778	95	419	13	20983
20988	448	419	13	21191
21199	133	419	13	21401
21412	938	419	13	21613

21629	423	431	13	21841
21852	90	431	13	22063
22073	640	431	13	22283
22301	922	433	13	22511
22536	250	439	13	22751
22779	367	439	13	22993
23010	447	443	13	23227
23252	559	449	13	23473
23491	121	457	13	23719
23730	623	457	13	23957
23971	450	457	13	24197
24215	253	461	13	24443
24476	106	467	13	24709
24721	863	467	13	24953
24976	148	479	13	25219
25230	427	479	13	25471
25493	138	479	13	25733
25756	794	487	13	26003
26022	247	487	13	26267
26291	562	491	13	26539
26566	53	499	13	26821
26838	135	499	13	27091
27111	21	503	13	27367
27392	201	509	13	27653

27682	169	521	13	27953
27959	70	521	13	28229
28248	386	521	13	28517
28548	226	523	13	28817
28845	3	541	13	29131
29138	769	541	13	29423
29434	590	541	13	29717
29731	672	541	13	30013
30037	713	547	13	30323
30346	967	547	13	30631
30654	368	557	14	30949
30974	348	557	14	31267
31285	119	563	14	31583
31605	503	569	14	31907
31948	181	571	14	32251
32272	394	577	14	32579
32601	189	587	14	32917
32932	210	587	14	33247
33282	62	593	14	33601
33623	273	593	14	33941
33961	554	599	14	34283
34302	936	607	14	34631
34654	483	607	14	34981
35031	397	613	14	35363

35395	241	619	14	35731
35750	500	631	14	36097
36112	12	631	14	36457
36479	958	641	14	36833
36849	524	641	14	37201
37227	8	643	14	37579
37606	100	653	14	37967
37992	339	653	14	38351
38385	804	659	14	38749
38787	510	673	14	39163
39176	18	673	14	39551
39576	412	677	14	39953
39980	394	683	14	40361
40398	830	691	15	40787
40816	535	701	15	41213
41226	199	701	15	41621
41641	27	709	15	42043
42067	298	709	15	42467
42490	368	719	15	42899
42916	755	727	15	43331
43388	379	727	15	43801
43840	73	733	15	44257
44279	387	739	15	44701
44729	457	751	15	45161

45183	761	751	15	45613
45638	855	757	15	46073
46104	370	769	15	46549
46574	261	769	15	47017
47047	299	787	15	47507
47523	920	787	15	47981
48007	269	787	15	48463
48489	862	797	15	48953
48976	349	809	15	49451
49470	103	809	15	49943
49978	115	821	15	50461
50511	93	821	16	50993
51017	982	827	16	51503
51530	432	839	16	52027
52062	340	853	16	52571
52586	173	853	16	53093
53114	421	857	16	53623
53650	330	863	16	54163
54188	624	877	16	54713
54735	233	877	16	55259
55289	362	883	16	55817
55843	963	907	16	56393
56403	471	907	16	56951

Table 2: Systematic indices and other parameters

5.7. Operating with Octets, Symbols and Matrices

5.7.1. General

This remainder of this section describes the arithmetic operations that are used to generate encoding symbols from source symbols and to generate source symbols from encoding symbols. Mathematically, octets can be thought of as elements of a finite field, i.e., the finite field GF(256) with 256 elements, and thus the addition and multiplication operations and identity elements and inverses over both operations are defined. Matrix operations and symbol operations are defined based on the arithmetic operations on octets. This allows a full implementation of these arithmetic operations without having to understand the underlying mathematics of finite fields.

5.7.2. Arithmetic Operations on Octets

Octets are mapped to non-negative integers in the range 0 through 255 in the usual way: A single octet of data from a symbol, B[7],B[6],B[5],B[4],B[3],B[2],B[1],B[0], where B[7] is the highest order bit and B[0] is the lowest order bit, is mapped to the integer i=B[7]*128+B[6]*64+B[5]*32+B[4]*16+B[3]*8+B[2]*4+B[1]*2+B[0].

The addition of two octets u and v defined as the XOR operation, i.e.,

u + v = u ^ v.

Subtraction is defined in the same way, so we also have

u - v = u ^ v.

The zero element (additive identity) is the octet represented by the integer 0. The additive inverse of u is simply u, i.e.,

u + u = 0.

The multiplication of two octets is defined with the help of two tables OCT_EXP and OCT_LOG, which are given in Section 5.7.3 and Section 5.7.4, respectively. The table OCT_LOG maps octets (other than the zero element) to non-negative integers, and OCT_EXP maps non-negative integers to octets. For two octets u and v, we define

u * v =

0, if either u or v are 0,

OCT_EXP[OCT_LOG[u] + OCT_LOG[v]] otherwise.

Note that the '+' on the right hand side of the above is the usual integer addition, since its arguments are ordinary integers.

The division u / v of two octets u and v, and where v != 0, is defined as follows:

u / v =

0, if u == 0,

OCT_EXP[OCT_LOG[u] - OCT_LOG[v] + 255] otherwise.

The one element (multiplicative identity) is the octet represented by the integer 1. For an octet u that is not the zero element, i.e., the multiplicative inverse of u is

OCT_EXP[255 - OCT_LOG[u]].

The octet denoted by alpha is the octet with the integer representation 2. If i is a non-negative integer 0 <= i < 256, we have

alpha^^i = OCT_EXP[i].

5.7.3. The table OCT_EXP

The table OCT_EXP contains 510 octets. The indexing starts at 0 and ranges up to 509, and the entries are the octets with the following positive integer representation:

1, 2, 4, 8, 16, 32, 64, 128, 29, 58, 116, 232, 205, 135, 19, 38, 76, 152, 45, 90, 180, 117, 234, 201, 143, 3, 6, 12, 24, 48, 96, 192, 157, 39, 78, 156, 37, 74, 148, 53, 106, 212, 181, 119, 238, 193, 159, 35, 70, 140, 5, 10, 20, 40, 80, 160, 93, 186, 105, 210, 185, 111, 222, 161, 95, 190, 97, 194, 153, 47, 94, 188, 101, 202, 137, 15, 30, 60, 120, 240, 253, 231, 211, 187, 107, 214, 177, 127, 254, 225, 223, 163, 91, 182, 113, 226, 217, 175, 67, 134, 17, 34, 68, 136, 13, 26, 52, 104, 208, 189, 103, 206, 129, 31, 62, 124, 248, 237, 199, 147, 59, 118, 236, 197, 151, 51, 102, 204, 133, 23, 46, 92, 184, 109, 218, 169, 79, 158, 33, 66, 132, 21, 42, 84, 168, 77, 154, 41, 82, 164, 85, 170, 73, 146, 57, 114, 228, 213, 183, 115, 230, 209, 191, 99, 198, 145, 63, 126, 252, 229, 215, 179, 123, 246, 241, 255, 227, 219, 171, 75, 150, 49, 98, 196, 149, 55, 110, 220, 165, 87, 174, 65, 130, 25, 50, 100, 200, 141, 7, 14, 28, 56, 112, 224, 221, 167, 83, 166, 81, 162, 89, 178, 121, 242, 249, 239, 195, 155, 43, 86, 172, 69, 138, 9, 18, 36, 72, 144, 61, 122, 244, 245, 247, 243, 251, 235, 203, 139, 11,

22, 44, 88, 176, 125, 250, 233, 207, 131, 27, 54, 108, 216, 173, 71, 142, 1, 2, 4, 8, 16, 32, 64, 128, 29, 58, 116, 232, 205, 135, 19, 38, 76, 152, 45, 90, 180, 117, 234, 201, 143, 3, 6, 12, 24, 48, 96, 192, 157, 39, 78, 156, 37, 74, 148, 53, 106, 212, 181, 119, 238, 193, 159, 35, 70, 140, 5, 10, 20, 40, 80, 160, 93, 186, 105, 210, 185, 111, 222, 161, 95, 190, 97, 194, 153, 47, 94, 188, 101, 202, 137, 15, 30, 60, 120, 240, 253, 231, 211, 187, 107, 214, 177, 127, 254, 225, 223, 163, 91, 182, 113, 226, 217, 175, 67, 134, 17, 34, 68, 136, 13, 26, 52, 104, 208, 189, 103, 206, 129, 31, 62, 124, 248, 237, 199, 147, 59, 118, 236, 197, 151, 51, 102, 204, 133, 23, 46, 92, 184, 109, 218, 169, 79, 158, 33, 66, 132, 21, 42, 84, 168, 77, 154, 41, 82, 164, 85, 170, 73, 146, 57, 114, 228, 213, 183, 115, 230, 209, 191, 99, 198, 145, 63, 126, 252, 229, 215, 179, 123, 246, 241, 255, 227, 219, 171, 75, 150, 49, 98, 196, 149, 55, 110, 220, 165, 87, 174, 65, 130, 25, 50, 100, 200, 141, 7, 14, 28, 56, 112, 224, 221, 167, 83, 166, 81, 162, 89, 178, 121, 242, 249, 239, 195, 155, 43, 86, 172, 69, 138, 9, 18, 36, 72, 144, 61, 122, 244, 245, 247, 243, 251, 235, 203, 139, 11, 22, 44, 88, 176, 125, 250, 233, 207, 131, 27, 54, 108, 216, 173, 71, 142

5.7.4. The table OCT_LOG

The table OCT_LOG contains 255 non-negative integers. The table is indexed by octets interpreted as integers. The octet corresponding to the zero element, which is represented by the integer 0, is excluded as an index, and thus indexing starts at 1 and ranges up to 255, and the entries are the following:

0, 1, 25, 2, 50, 26, 198, 3, 223, 51, 238, 27, 104, 199, 75, 4, 100, 224, 14, 52, 141, 239, 129, 28, 193, 105, 248, 200, 8, 76, 113, 5, 138, 101, 47, 225, 36, 15, 33, 53, 147, 142, 218, 240, 18, 130, 69, 29, 181, 194, 125, 106, 39, 249, 185, 201, 154, 9, 120, 77, 228, 114, 166, 6, 191, 139, 98, 102, 221, 48, 253, 226, 152, 37, 179, 16, 145, 34, 136, 54, 208, 148, 206, 143, 150, 219, 189, 241, 210, 19, 92, 131, 56, 70, 64, 30, 66, 182, 163, 195, 72, 126, 110, 107, 58, 40, 84, 250, 133, 186, 61, 202, 94, 155, 159, 10, 21, 121, 43, 78, 212, 229, 172, 115, 243, 167, 87, 7, 112, 192, 247, 140, 128, 99, 13, 103, 74, 222, 237, 49, 197, 254, 24, 227, 165, 153, 119, 38, 184, 180, 124, 17, 68, 146, 217, 35, 32, 137, 46, 55, 63, 209, 91, 149, 188, 207, 205, 144, 135, 151, 178, 220, 252, 190, 97, 242, 86, 211, 171, 20, 42, 93, 158, 132, 60, 57, 83, 71, 109, 65, 162, 31, 45, 67, 216, 183, 123, 164, 118, 196, 23, 73, 236, 127, 12, 111, 246, 108, 161, 59, 82, 41, 157, 85, 170, 251, 96, 134, 177, 187, 204, 62, 90, 203, 89, 95, 176, 156, 169, 160, 81, 11, 245, 22, 235, 122, 117, 44, 215, 79, 174, 213, 233, 230, 231, 173, 232, 116, 214, 244, 234, 168, 80, 88, 175

5.7.5. Operations on Symbols

Operations on symbols have the same semantics as operations on vectors of octets of length T in this specification. Thus, if U and V are two symbols formed by the octets u[0], ..., u[T-1] and v[0], ..., v[T-1], respectively, the sum of symbols U + V is defined to be the component-wise sum of octets, i.e., equal to the symbol D formed by the octets d[0], ..., d[T-1], such that

$$d[i] = u[i] + v[i], \quad 0 \leq i < T.$$

Furthermore, if beta is an octet, the product beta*U is defined to be the symbol D obtained by multiplying each octet of U by beta, i.e.,

$$d[i] = beta*u[i], \quad 0 \leq i < T.$$

5.7.6. Operations on Matrices

All matrices in this specification have entries that are octets, and thus matrix operations and definitions are defined in terms of the underlying octet arithmetic, e.g., operations on a matrix, matrix rank and matrix inversion.

5.8. Requirements for a Compliant Decoder

If a RaptorQ compliant decoder receives a mathematically sufficient set of encoding symbols generated according to the encoder specification in Section 5.3 for reconstruction of a source block then such a decoder SHOULD recover the entire source block.

A RaptorQ compliant decoder SHALL have the following recovery properties for source blocks with K' source symbols for all values of K' in Table 2 of Section 5.6.

1. If the decoder receives K' encoding symbols generated according to the encoder specification in Section 5.3 with corresponding ESIs chosen independently and uniformly at random from the range of possible ESIs then on average the decoder will fail to recover the entire source block at most 1 out of 100 times.

2. If the decoder receives K'+1 encoding symbols generated according to the encoder specification in Section 5.3 with corresponding ESIs chosen independently and uniformly at random from the range of possible ESIs then on average the decoder will fail to recover the entire source block at most 1 out of 10,000 times.

3. If the decoder receives K'+2 encoding symbols generated according to the encoder specification in Section 5.3 with corresponding

ESIs chosen independently and uniformly at random from the range of possible ESIs then on average the decoder will fail to recover the entire source block at most 1 out of 1,000,000 times.

Note that the Example FEC Decoder specified in Section 5.4 fulfills both requirements, i.e.

1. it can reconstruct a source block as long as it receives a mathematically sufficient set of encoding symbols generated according to the encoder specification in Section 5.3;

2. it fulfills the mandatory recovery properties from above.

6. Security Considerations

Data delivery can be subject to denial-of-service attacks by attackers which send corrupted packets that are accepted as legitimate by receivers. This is particularly a concern for multicast delivery because a corrupted packet may be injected into the session close to the root of the multicast tree, in which case the corrupted packet will arrive at many receivers. This is particularly a concern when the code described in this document is used because the use of even one corrupted packet containing encoding data may result in the decoding of an object that is completely corrupted and unusable. It is thus RECOMMENDED that source authentication and integrity checking are applied to decoded objects before delivering objects to an application. For example, a SHA-1 hash [SHA1] of an object may be appended before transmission, and the SHA-1 hash is computed and checked after the object is decoded but before it is delivered to an application. Source authentication SHOULD be provided, for example by including a digital signature verifiable by the receiver computed on top of the hash value. It is also RECOMMENDED that a packet authentication protocol such as TESLA [RFC4082] be used to detect and discard corrupted packets upon arrival. This method may also be used to provide source authentication. Furthermore, it is RECOMMENDED that Reverse Path Forwarding checks be enabled in all network routers and switches along the path from the sender to receivers to limit the possibility of a bad agent successfully injecting a corrupted packet into the multicast tree data path.

Another security concern is that some FEC information may be obtained by receivers out-of-band in a session description, and if the session description is forged or corrupted then the receivers will not use the correct protocol for decoding content from received packets. To avoid these problems, it is RECOMMENDED that measures be taken to prevent receivers from accepting incorrect session descriptions,

e.g., by using source authentication to ensure that receivers only accept legitimate session descriptions from authorized senders.

7. IANA Considerations

Values of FEC Encoding IDs and FEC Instance IDs are subject to IANA registration. For general guidelines on IANA considerations as they apply to this document, see [RFC5052]. IANA is requested to assign a value under the ietf:rmt:fec:encoding name-space to "RaptorQ Code" as the Fully-Specified FEC Encoding ID value associated with this specification, preferably the value 6.

8. Acknowledgements

Thanks are due to Ranganathan (Ranga) Krishnan. Ranga Krishnan has been very supportive in finding and resolving implementation details and in finding the systematic indices. In addition, Habeeb Mohiuddin Mohammed and Antonios Pitarokoilis, both from the Munich University of Technology (TUM) and Alan Shinsato have done two independent implementations of the RaptorQ encoder/decoder that have helped to clarify and to resolve issues with this specification.

9. References

9.1. Normative references

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

[RFC4082] Perrig, A., Song, D., Canetti, R., Tygar, J., and B. Briscoe, "Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction", RFC 4082, June 2005.

[SHA1] "Secure Hash Standard", Federal Information Processing Standards Publication (FIPS PUB) 180-1, April 2005.

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

9.2. Informative references

[RFC3453] Luby, M., Vicisano, L., Gemmell, J., Rizzo, L., Handley, M., and J. Crowcroft, "The Use of Forward Error Correction

(FEC) in Reliable Multicast", RFC 3453, December 2002.

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

[LTCodes] Luby, "LT codes," Annual IEEE Symposium on Foundations of Computer Science, pp. 271 -- 280, November 2002.

Authors' Addresses

Michael Luby
Qualcomm Incorporated
3165 Kifer Road
Santa Clara, CA 95051
U.S.A.

Email: luby@qualcomm.com

Amin Shokrollahi
EPFL
Laboratoire d'algorithmique
EPFL
Station 14
Batiment BC
Lausanne 1015
Switzerland

Email: amin.shokrollahi@epfl.ch

Mark Watson
Netflix Inc.
100 Winchester Circle
Los Gatos, CA 95032
U.S.A.

Email: watsonm@netflix.com

Thomas Stockhammer
Nomor Research
Brecherspitzstrasse 8
Munich 81541
Germany

Email: stockhammer@nomor.de

Lorenz Minder
Qualcomm Incorporated
3165 Kifer Road
Santa Clara, CA 95051
U.S.A.

Email: lminder@qualcomm.com

RMT
Internet-Draft
Intended status: Experimental
Expires: August 14, 2011

V. Roca
INRIA
B. Adamson
Naval Research Laboratory

February 10, 2011

FCAST: Scalable Object Delivery for the ALC and NORM Protocols
draft-ietf-rmt-fcast-03

Abstract

This document introduces the FCAST object (e.g., file) delivery application on top of the ALC and NORM reliable multicast protocols. FCAST is a highly scalable application that provides a reliable object delivery service.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 14, 2011.

Copyright Notice

Table of Contents

1. Introduction

This document introduces the FCAST reliable and scalable object (e.g., file) delivery application. Two variants of FCAST exist:

- FCAST/ALC that relies on the Asynchronous Layer Coding (ALC) [RFC5775] and the Layered Coding Transport (LCT) [RFC5651] reliable multicast transport protocol, and
- FCAST/NORM that relies on the NACK-Oriented Reliable Multicast (NORM) [RFC5740] reliable multicast transport protocol.

Hereafter, the term FCAST denotes either FCAST/ALC or FCAST/NORM. FCAST is not a new protocol specification per se. Instead it is a set of data format specifications and instructions on how to use ALC and NORM to implement a file-casting service.

Depending on the target use case, the delivery service provided by FCAST is more or less reliable. For instance, with FCAST/ALC used in ON-DEMAND mode over a time period that largely exceeds the typical download time, the service can be considered as fully reliable. Similarly, when FCAST is used along with a session control application that collects reception information and takes appropriate corrective measures (e.g., a direct point-to-point retransmission of missing packets, or a new multicast recovery session), then the service can be considered as fully reliable. On the opposite, if FCAST operates in PUSH mode, then the service is usually only partially reliable, and a receiver that is disconnected during a sufficient time will perhaps not have the possibility to download the object.

Depending on the target use case, the FCAST scalability is more or less important. For instance, if FCAST/ALC is used on top of purely unidirectional transport channels, with no feedback information at all, which is the default mode of operation, then the scalability is maximum since neither FCAST, nor ALC, UDP or IP generates any feedback message. On the opposite, the FCAST/NORM scalability is typically limited by NORM scalability itself. Similarly, if FCAST is used along with a session control application that collects reception information from the receivers, then this session control application may limit the scalability of the global object delivery system. This situation can of course be mitigated by using a hierarchy of feedback message aggregators or servers. The details of this are out of the scope of the present document.

A design goal behind FCAST is to define a streamlined solution, in order to enable lightweight implementations of the protocol stack, and limit the operational processing and storage requirements. A

consequence of this choice is that FCAST cannot be considered as a versatile application, capable of addressing all the possible use-cases. On the opposite, FCAST has some intrinsic limitations. From this point of view it differs from FLUTE [RMT-FLUTE] which favors flexibility at the expense of some additional complexity.

A good example of the design choices meant to favor the simplicity is the way FCAST manages the object meta-data: by default, the meta-data and the object content are sent together, in a compound object. This solution has many advantages in terms of simplicity as will be described later on. However, as such, it also has an intrinsic limitation since it does not enable a receiver to decide in advance, before beginning the reception of the compound object, whether the object is of interest or not, based on the information that may be provided in the meta-data. Therefore this document defines additional techniques that may be used to mitigate this limitation. It is also possible that some use-cases require that each receiver download the whole set of objects sent in the session (e.g., with mirroring tools). When this is the case, the above limitation is no longer be a problem.

1.1. Applicability

FCAST is compatible with any congestion control protocol designed for ALC/LCT or NORM. However, depending on the use-case, the data flow generated by the FCAST application might not be constant, but instead be bursty in nature. Similarly, depending on the use-case, an FCAST session might be very short. Whether and how this will impact the congestion control protocol is out of the scope of the present document.

FCAST is compatible with any security mechanism designed for ALC/LCT or NORM. The use of a security scheme is strongly RECOMMENDED (see Section 6).

FCAST is compatible with any FEC scheme designed for ALC/LCT or NORM. Whether FEC is used or not, and the kind of FEC scheme used, is to some extent transparent to FCAST.

FCAST is compatible with both IPv4 and IPv6. Nothing in the FCAST specification has any implication on the source or destination IP address.

2. Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this

document are to be interpreted as described in [RFC2119].

3. Definitions, Notations and Abbreviations

3.1. Definitions

This document uses the following definitions:

FCAST/ALC: denotes the FCAST application running on top of the ALC/LCT reliable transport protocol;

FCAST/NORM: denotes the FCAST application running on top of the NORM reliable transport protocol;

FCAST: denotes either FCAST/ALC or FCAST/NORM;

Compound Object: denotes an ALC or NORM transport object composed of the Compound Object Header (Section 5.1), including any meta-data, and the content of the original application object (e.g., a file);

Carousel: denotes the process of sending Compound Objects implemented by a FCAST sender;

Carousel Instance: denotes a fixed set of registered Compound Objects that are sent by the carousel during a certain number of cycles. Whenever Compound Objects need to be added or removed, a new Carousel Instance is defined;

Carousel Instance Descriptor (CID): denotes a special object that lists the Compound Objects that comprise a given Carousel Instance;

Carousel Cycle: denotes a transmission round within which all the Compound Objects registered in the Carousel Instance are transmitted a certain number of times. By default, Compound Objects are transmitted once per cycle, but higher values are possible, that might differ on a per-object basis;

Transmission Object Identifier (TOI): denotes the numeric identifier associated to a specific object by the underlying transport layer. In the case of ALC, this corresponds to the TOI described in that specification while for the NORM specification this corresponds to the NormTransportId described there.

3.2. Abbreviations

This document uses the following abbreviations:

CID: Carousel Instance Descriptor

CIID: Carousel Instance IDentifier

FEC OTI: FEC Object Transmission Information

TOI: Transmission Object Identifier

4. FCAST Principles

4.1. FCAST Content Delivery Service

The basic goal of FCAST is to transmit objects to a group of receivers in a reliable way. The receiver set MAY be restricted to a single receiver or MAY include possibly a very large number of receivers. FCAST is specified to support two forms of operation:

1. FCAST/ALC: where the FCAST application is meant to run on top of the ALC/LCT reliable multicast transport protocol, and

2. FCAST/NORM: where the FCAST application is meant to run on top of the NORM reliable multicast transport protocol.

This specification is designed such that both forms of operation share as much commonality as possible.

While the choice of the underlying transport protocol (i.e., ALC or NORM) and its parameters may limit the practical receiver group size, nothing in FCAST itself limits it. The transmission might be fully reliable, or only partially reliable depending upon the way ALC or NORM is used (e.g., whether FEC encoding and/or NACK-based repair requests are used or not), the way the FCAST carousel is used (e.g., whether the objects are made available for a long time span or not), and the way in which FCAST itself is employed (e.g., whether there is a session control application that might automatically extend an existing FCAST session until all receivers have received the transmitted content).

FCAST is designed to be as self-sufficient as possible, in particular in the way object meta-data is attached to object data content. However, for some uses, meta-data MAY also be communicated by an out-of-band mechanism that is out of the scope of the present document.

4.2. Meta-Data Transmission

FCAST usually carries meta-data elements by prepending them to the object it refers to. As a result, a Compound Object is created that is composed of a header followed by the original object data. This header is itself composed of the meta-data as well as several fields, for instance to indicate the boundaries between the various parts of this Compound Object (Figure 1).

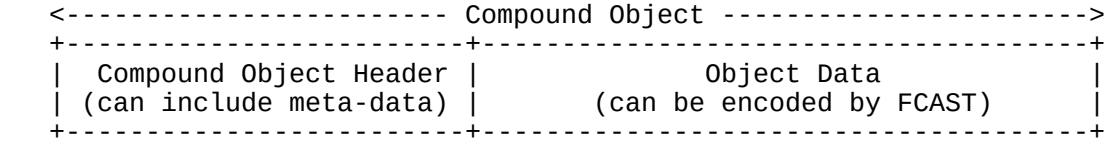

Figure 1: Compound Object composition.

Attaching the meta-data to the object is an efficient solution, since it guaranties that meta-data be received along with the associated object, and it allows the transport of the meta-data to benefit from any transport-layer erasure protection of the Compound Object (e.g., using FEC encoding and/or NACK-based repair). However a limit of this scheme, as such, is that a client does not know the meta-data of an object before beginning its reception, and in case of erasures affecting the meta-data, not until the object decoding is completed. The details of course depend upon the transport protocol and the FEC code used.

In certain use-cases, FCAST can also be associated to another in-band (e.g., via NORM INFO messages, Section 4.8) or out-of-band signaling mechanism. In that case, this mechanism can be used in order to carry the whole meta-data (or a subset of it), possibly ahead of time.

4.3. Meta-Data Content

The meta-data associated to an object can be composed of, but are not limited to:

- Content-Location: the URI of the object, which gives the name and location of the object;
- Content-Type: the MIME type of the object;
- Content-Length: the size of the initial object, before any content encoding (if any). Note that this content length does not include the meta-data nor the fixed size Compound Object header;

- Content-Encoding: the optional encoding of the object performed by FCAST. If there is no Content-Encoding entry, the receiver MUST assume that the object is not encoded (default). The support of gzip encoding, or any other solution, remains optional.

- Content-MD5: the MD5 message digest of the object in order to check its integrity. Note that this digest is meant to protect from transmission and processing errors, not from deliberate attacks by an intelligent attacker. Note also that this digest only protects the object, not the header, and therefore not the meta-data. A separate checksum is provided to that purpose (Section 5.1);

- a digital signature for this object;

This list is not limited and new meta-data information can be added. For instance, when dealing with very large objects (e.g., that largely exceed the working memory of a receiver), it can be interesting to split this object into several sub-objects (or slices). When this happens, the meta-data associated to each sub-object MUST include the following entries:

- Fcast-Obj-Slice-Nb: the total number of slices. A value strictly greater than 1 indicates that this object is the result of a split of the original object;

- Fcast-Obj-Slice-Idx: the slice index (in the {0 .. SliceNb - 1} interval);

- Fcast-Obj-Slice-Offset: the offset at which this slice starts within the original object;

When meta-data elements are communicated out-of-band, in advance of data transmission, the following pieces of information MAY also be useful:

- TOI: the Transmission Object Identifier (TOI) of the object (Section 4.6), in order to enable a receiver to easily associate the meta-data to the object;

- FEC Object Transmission Information (FEC OTI). In this case the FCAST sender does not need to use the optional EXT_FTI mechanism of the ALC or NORM protocols.

4.4. Carousel Transmission

A set of FCAST Compound Objects scheduled for transmission are considered a logical "Carousel". A given "Carousel Instance" is comprised of a fixed set of Compound Objects. Whenever the FCAST application needs to add new Compound Objects to, or remove old Compound Objects from the transmission set, a new Carousel Instance is defined since the set of Compound Objects changes. Because of the native object multiplexing capability of both ALC and NORM, sender and receiver(s) are both capable to multiplex and demultiplex FCAST Compound Objects.

For a given Carousel Instance, one or more transmission cycles are possible. During each cycle, all of the Compound Objects comprising the Carousel are sent. By default, each object is transmitted once per cycle. However, in order to allow different levels of priority, some objects MAY be transmitted more often that others during a cycle, and/or benefit from higher FEC protection than others. This can be the case for instance for the CID objects (Section 4.5). For some FCAST usage (e.g., a unidirectional "push" mode), a Carousel Instance may be associated to a single transmission cycle. In other cases it may be associated to a large number of transmission cycles (e.g., in "on-demand" mode, where objects are made available for download during a long period of time).

4.5. Carousel Instance Descriptor Special Object

The FCAST sender CAN transmit an OPTIONAL Carousel Instance Descriptor (CID). The CID carries the list of the Compound Objects that are part of a given Carousel Instance, by specifying their respective Transmission Object Identifiers (TOI). However the CID does not describe the objects themselves (i.e., there is no meta-data). Additionally, the CID MAY include a "Complete" flag that is used to indicate that no further modification to the enclosed list will be done in the future. Finally, the CID MAY include a Carousel Instance ID that identifies the Carousel Instance it pertains to. These aspects are discussed in Section 5.2.

There is no reserved TOI value for the CID Compound Object itself, since this special object is regarded by ALC/LCT or NORM as a standard object. On the opposite, the nature of this object (CID) is indicated by means of a specific Compound Object header field (the "I" flag) so that it can be recognized and processed by the FCAST application as needed. A direct consequence is the following: since a receiver does not know in advance which TOI will be used for the following CID (in case of a dynamic session), he MUST NOT filter out packets that are not in the current CID’s TOI list. Said differently, the goal of CID is not to setup ALC or NORM packet

filters (this mechanism would not be secure in any case).

The use of a CID remains optional. If it is not used, then the clients progressively learn what files are part of the carousel instance by receiving ALC or NORM packets with new TOIs. However using a CID has several benefits:

- When the "Complete" flag is set (if ever), the receivers know when they can leave the session, i.e., when they have received all the objects that are part of the last carousel instance of this delivery session;

- In case of a session with a dynamic set of objects, the sender can reliably inform the receivers that some objects have been removed from the carousel with the CID. This solution is more robust than the "Close Object flag (B)" of ALC/LCT since a client with an intermittent connectivity might loose all the packets containing this B flag. And while NORM provides a robust object cancellation mechanism in the form of its NORM_CMD(SQUELCH) message in response to receiver NACK repair requests, the use of the CID provides an additional means for receivers to learn of objects for which it is futile to request repair;

- The TOI equivalence (Section 4.6) can be signaled with the CID. This is often preferable to the alternative solution where the equivalence is identified by examining the object meta-data, especially in case of erasures.

During idle periods, when the carousel instance does not contain any object, a CID with an empty TOI list MAY be transmitted. In that case, a new carousel instance ID MUST be used to differentiate this (empty) carousel instance from the other ones. This mechanism can be useful to inform the receivers that:

- all the previously sent objects have been removed from the carousel. It therefore improves the FCAST robustness even during "idle" period;

- the session is still active even if there is currently no content being sent. Said differently, it can be used as a heartbeat mechanism. If the "Complete" flag has not been set, it implicitly informs the receivers that new objects MAY be sent in the future;

The decisions of whether a CID should be used or not, how often and when a CID should be sent, are left to the sender and depend on many parameters, including the target use case and the session dynamics. For instance it may be appropriate to send a CID at the beginning of each new carousel instance, and then periodically. These operational

aspects are out of the scope of the present document.

4.6. Compound Object Identification

The FCAST Compound Objects are directly associated with the object-based transport service that the ALC and NORM protocols provide. In each of these protocols, the messages containing transport object content are labeled with a numeric transport object identifier (i.e., the ALC TOI and the NORM NormTransportId). For the purposes of this document, this identifier in either case (ALC or NORM) is referred to as the TOI.

There are several differences between ALC and NORM:

- o the ALC use of TOI is rather flexible, since several TOI field sizes are possible (from 16 to 112 bits), since this size can be changed at any time, on a per-packet basis, and since the TOI management is totally free as long as each object is associated to a unique TOI (if no wraparound happened);

- o the NORM use of TOI is more directive, since the TOI field is 16 bit long and since TOIs MUST be managed sequentially;

In both NORM and ALC, it is possible that the transport identification space may eventually wrap for long-lived sessions (especially with NORM where this phenomenon is expected to happen more frequently). This can possibly introduce some ambiguity in FCAST object identification if a sender retains some older objects in newer Carousel Instances with updated object sets. To avoid ambiguity the active TOIs (i.e., the TOIs corresponding to objects being transmitted) can only occupy half of the TOI sequence space. If an old object, whose TOI has fallen outside the current window, needs to be transmitted again, a new TOI must be used for it. In case of NORM, this constraint limits to 32768 the maximum number of objects that can be part of any carousel instance. In order to allow receivers to properly combine the transport packets with a newly-assigned TOI to those of associated to the previously-assigned TOI, a mechanism is required to equate the objects with the new and the old TOIs.

The preferred mechanism consists in signaling, within the CID, that the newly assigned TOI, for the current Carousel Instance, is equivalent to the TOI used within a previous Carousel Instance. By convention, the reference tuple for any object is the {TOI; CI ID} tuple used for its first transmission within a Carousel Instance. This tuple MUST be used whenever a TOI equivalence is provided.

An alternative solution, when meta-data can be processed rapidly

(e.g., by using NORM-INFO messages), consists for the receiver in identifying that both objects are the same, after examining the meta-data. The receiver can then take appropriate measures.

4.7. FCAST/ALC Additional Specificities

There are no additional detail or option for FCAST/ALC operation.

4.8. FCAST/NORM Additional Specificities

The NORM Protocol provides a few additional capabilities that can be used to specifically support FCAST operation:

1. The NORM_INFO message can convey "out-of-band" content with respect to a given transport object. With FCAST, it MAY be used to provide to the receivers a new, associated, Compound Object which contains the main Compound Object meta-data, or a subset of it. In that case the NORM_INFO Compound Object MUST NOT contain any Object Data field (i.e., it is only composed of the header), it MUST feature a non global checksum, and it MUST NOT include any padding field. The main Compound Object MUST in any case contain the whole meta-data (e.g., because a receiver MAY not support the NORM_INFO facility). Additionally, the meta-data entries contained in the NORM_INFO MUST be identical to the same entries in the main Compound Object. Finally, note that the availability of NORM_INFO for a given object is signaled through the use of a dedicated flag in the NORM_DATA message header. Along with NORM's NACK-based repair request signaling, it allows a receiver to quickly (and independently) request an object's NORM_INFO content. However, a limitation here is that the NORM_INFO Compound Object header MUST fit within the byte size limit defined by the NORM sender's configured "segment size" (typically a little less than the network MTU);

2. The NORM_CMD(SQUELCH) messages are used by the NORM protocol sender to inform receivers of objects that have been canceled when receivers make repair requests for such invalid objects. Along with the CID mechanism, a receiver has two efficient and reliable ways to discover old objects that have been removed from the carousel instance;

3. NORM also supports an optional positive acknowledgment mechanism that can be used for small-scale multicast receiver group sizes. Also, it may be possible in some cases for the sender to infer, after some period without receiving NACKs at the end of its transmission that the receiver set has fully received the transmitted content. In particular, if the sender completes its end-of-transmission series of NORM_CMD(FLUSH) messages without

receiving repair requests from the group, it may have some assurance that the receiver set has received the content prior to that point. These mechanisms are likely to help FCAST in achieving fully reliable transmissions;

It should be noted that the NORM_INFO message header may carry the EXT_FTI extension. The reliable delivery of the NORM_INFO content allows the individual objects' FEC Transmission Information to be provided to the receivers without burdening every packet (i.e. NORM_DATA messages) with this additional, but important, content. Examples are provided in Appendix A.

4.9. FCAST Sender Behavior

The following operations MAY take place at a sender:

1. The user (or another application) selects a set of objects (e.g., files) to deliver and submits them, along with their meta-data, to the FCAST application;

2. For each object, FCAST creates the Compound Object and registers this latter in the carousel instance;

3. The user then informs FCAST that all the objects of the set have been submitted. If the user knows that no new object will be submitted in the future (i.e., if the session's content is now complete), the user informs FCAST. Finally, the user specifies how many transmission cycles are desired (this number may be infinite);

4. At this point, the FCAST application knows the full list of Compound Objects that are part of the Carousel Instance and can create a CID if desired, possibly with the complete flag set;

5. The FCAST application can now define a transmission schedule of these Compound Objects, including the optional CID. This schedule defines in which order the packets of the various Compound Objects should be sent. This document does not specify any scheme. This is left to the developer within the provisions of the underlying ALC or NORM protocol used and the knowledge of the target use-case.

6. The FCAST application then starts the carousel transmission, for the number of cycles specified. Transmissions take place until:

   * the desired number of transmission cycles has been reached, or

* the user wants to prematurely stop the transmissions, or

* the user wants to add one or several new objects to the carousel, or on the opposite wants to remove old objects from the carousel. In that case a new carousel instance must be created.

7. If the session is not finished, then continue at Step 1 above;

4.10. FCAST Receiver Behavior

The following operations MAY take place at a receiver:

1. The receiver joins the session and collects incoming packets;

2. If the header portion of a Compound Object is entirely received (which may happen before receiving the entire object with some ALC/NORM configurations), or if the meta-data is sent by means of another mechanism prior to the object, the receiver processes the meta-data and chooses to continue to receive the object content or not;

3. When a Compound Object has been entirely received, the receiver processes the header, retrieves the object meta-data, perhaps decodes the meta-data, and processes the object accordingly;

4. When a CID is received, which is indicated by the 'C' flag set in the Compound Object header, the receiver decodes the CID, and retrieves the list of objects that are part of the current carousel instance. This list CAN be used to remove objects sent in a previous carousel instance that might not have been totally decoded and that are no longer part of the current carousel instance;

5. When a CID is received, the receiver also retrieves the list of TOI equivalences, if any, and takes appropriate measures, for instance by informing the transport layer;

6. When a receiver has received a CID with the "Complete" flag set, and has successfully received all the objects of the current carousel instance, it can safely exit from the current FCAST session;

7. Otherwise continue at Step 2 above.

5. FCAST Data Formats

This section details the various data formats used by FCAST.

5.1. Compound Object Header Format

In an FCAST session, Compound Objects are constructed by prepending the Compound Object Header (which may include meta-data) before the original object data content (Figure 2).

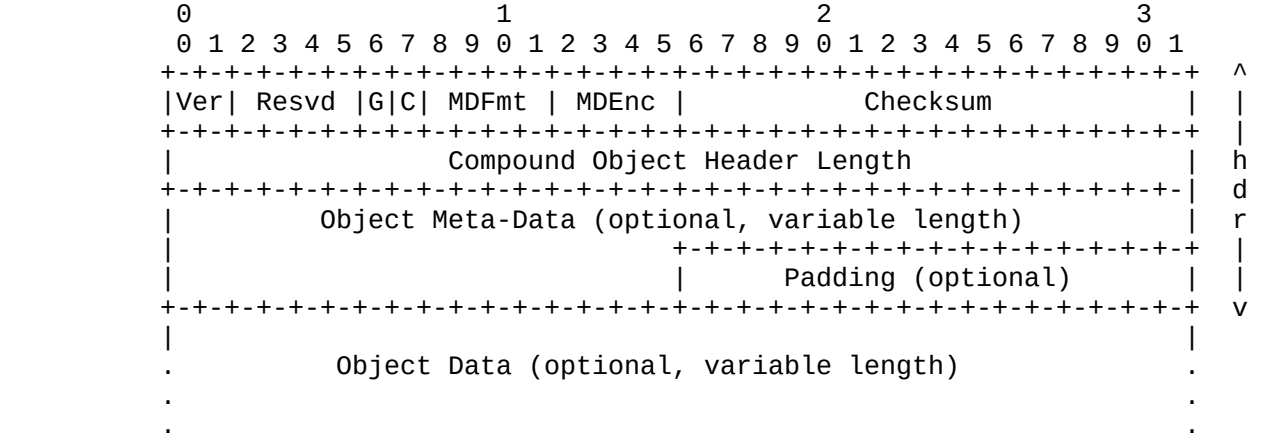

Figure 2: Compound Object Header with Meta-Data.

The Compound Object Header fields are:

Field	Description
Version	2-bit field that MUST be set to 0 in this specification and indicates the protocol version number.
Reserved	4-bit field that MUST be set to 0 in this specification and is reserved for future use. Receivers MUST ignore this field.
G	1-bit field that, when set to 1, indicates that the checksum encompasses the whole Compound Object (Global checksum). When set to 0, this field indicates that the checksum encompasses only the Compound Object header.
C	1-bit field that, when set to 1, indicates the object is a Carousel Instance Descriptor (CID). When set to 0, this field indicates that the transported object is a standard object.

Meta-Data Format (MDFmt)	4-bit field that defines the format of the object meta-data (see Section 7). An HTTP/1.1 metainformation format [RFC2616] MUST be supported and is associated to value 0. Other formats (e.g., XML) MAY be defined in the future.
Meta-Data Encoding (MDEnc)	4-bit field that defines the optional encoding of the Object Meta-Data field (see Section 7). By default, a plain text encoding is used and is associated to value 0. Gzip encoding MUST also be supported and is associated to value 1. Other encodings MAY be defined in the future.
Checksum	16-bit field that contains the checksum computed over either the whole Compound Object (when G is set to 1), or over the Compound Object header (when G is set to 0), using the Internet checksum algorithm specified in [RFC1071]. More precisely, the checksum field is the 16-bit one's complement of the one's complement sum of all 16-bit words to be considered. If a segment contains an odd number of octets to be checksummed, the last octet is padded on the right with zeros to form a 16-bit word for checksum purposes (this pad is not transmitted). While computing the checksum, the checksum field itself is set to zero.
Compound Object Header Length	32-bit field indicating total length (in bytes) of all fields of the Compound Object Header, except the optional padding. A header length field set to value 8 means that there is no meta-data included. When this size is not multiple to 32-bits words and when the Compound Object Header is followed by a non null Compound Object Data, padding MUST be added. It should be noted that the meta-data field maximum size is equal to (2^32 - 8) bytes.
Object Meta-Data	Optional, variable length field that contains the meta-data associated to the object. The format and encoding of this field is defined by the MDFmt MDEnc fields. With the default HTTP/1.1 format and plain text encoding, the Meta-Data is NULL-terminated plain text that follows the "TYPE" ":" "VALUE" "<CR-LF>" format used in HTTP/1.1 for metainformation [RFC2616]. The various meta-data items can appear in any order. The associated string, when non empty, MUST be NULL-terminated. When no meta-data is communicated, this field MUST be empty and the Compound Object Header Length MUST be equal to 8.

Padding	Optional, variable length field of zero-value bytes to align the start of the Object Data to 32-bit boundary. Padding is only used when the Compound Object Header Length value, in bytes, is not multiple of 4 and when the Compound Object Header is followed by non null Compound Object Data.	

The Compound Object Header is then followed by the Object Data, i.e., the original object possibly encoded by FCAST. Note that the length of this content is the transported object length (e.g., as specified by the FEC OTI) minus the Compound Object Header Length and optional padding if any.

5.2. Carousel Instance Descriptor Format

The format of the CID, which is a special Compound Object, is given in Figure 3.

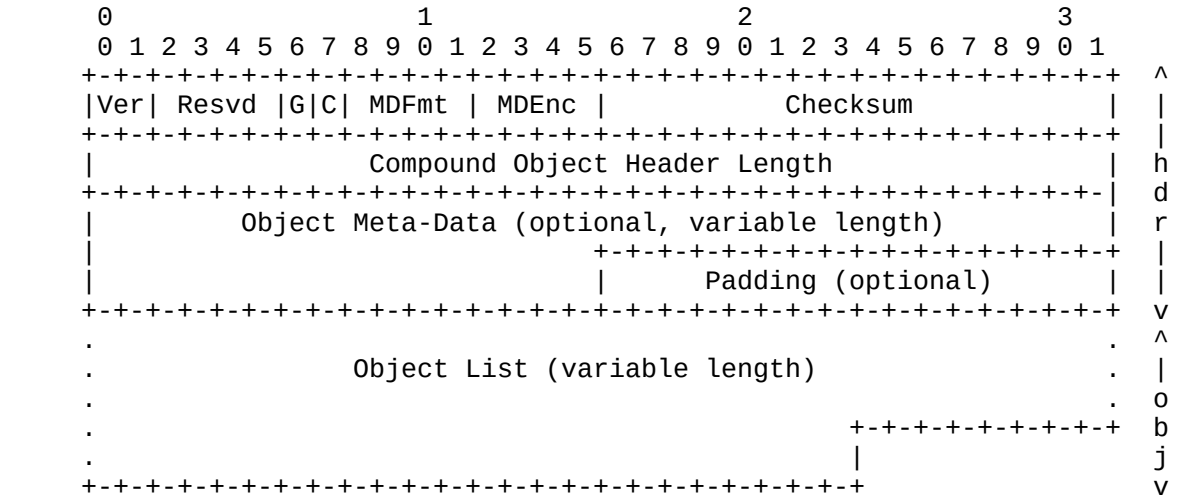

Figure 3: Carousel Instance Object Format.

Because the CID is transmitted as a special Compound Object, the following CID-specific meta-data entries are defined:

- Fcast-CID-Complete: when set to 1, it indicates that no new objects in addition to the ones whose TOI are specified in this CID, or the ones that have been specified in the previous CID(s), will be sent in the future. Otherwise it MUST be set to 0. This entry is optional. If absent, a receiver MUST conclude that the session is complete.

- Fcast-CID-ID: this entry contains the Carousel Instance IDentifier, or CIID. It starts from 0 and is incremented by 1 for each new carousel instance. This entry is optional if the FCAST session consists of a single, complete, carousel instance. In all other cases, this entry MUST be defined. In particular, the CIID is used by the TOI equivalence mechanism thanks to which any object is uniquely identified, even if the TOI is updated (e.g., after re-enqueuing the object with NORM). The Fcast-CID-ID value can also be useful to detect possible gaps in the Carousel Instances, for instance caused by long disconnection periods. Finally, it can also be useful to avoid problems when TOI wrapping to 0 takes place to differentiate the various incarnations of the TOIs if need be.

The motivation for making the Fcast-CID-Complete and Fcast-CID-ID entries optional is to simplify the simple case of a session consisting of a single, complete, carousel instance, with an Object List given in plain text, without any content encoding. In that case, the CID does not need to contain any meta-data entry.

Additionally, the following standard meta-data entries are often used (Section 4.3):

- Content-Length: it specifies the size of the object list, before any content encoding (if any).

- Content-Encoding: it specifies the optional encoding of the object list, performed by FCAST. For instance:

  Content-Encoding: gzip

  indicates that the Object List field has been encoded with gzip [RFC1952]. If there is no Content-Encoding entry, the receiver MUST assume that the Object List field is plain text (default). The support of gzip encoding, or any other solution, remains optional.

An empty Object List is valid and indicates that the current carousel instance does not include any object (Section 4.5). This can be specified by using the following meta-data entry:

Content-Length: 0

or simply by leaving the Object List empty. In both cases, padding MUST NOT be used and consequently the transported object length (e.g., as specified by the FEC OTI) minus the Compound Object Header Length equals zero.

The non-encoded (i.e., plain text) Object List, when non empty, is a NULL-terminated ASCII string. It can contain two things:

o a list of TOI values, and

o a list of TOI equivalences;

First of all, this string can contain the list of TOIs included in the current carousel instance, specified either as the individual TOIs of each object, or as TOI intervals, or any combination. The format of the ASCII string is a comma-separated list of individual "TOI" values or "TOI_a-TOI_b" elements. This latter case means that all values between TOI_a and TOI_b, inclusive, are part of the list. In that case TOI_a MUST be strictly inferior to TOI_b. If a TOI wrapping to 0 occurs in an interval, then two TOI intervals MUST be specified, TOI_a-MAX_TOI and 0-TOI_b.

This string can also contain the TOI equivalences, if any. The format is a comma-separated list of "(" newTOI "=" 1stTOI "/" 1stCIID ")" elements. Each element says that the new TOI, for the current Carousel Instance, is equivalent to (i.e., refers to the same object as) the provided identifier, 1stTOI, for the Carousel Instance of ID 1stCIID.

The ABNF specification is the following:

```
cid-list   =  *(list-elem *( "," list-elem))
list-elem    =  toi-elem / toieq-elem
toi-elem     =  toi-value / toi-interval
toi-value    =  1*DIGIT
toi-interval =  toi-value "-" toi-value
                ; additionally, the first toi-value MUST be
                ; strictly inferior to the second toi-value
toieq-elem   =  "(" toi-value "=" toi-value "/" ciid-value ")"
ciid-value   = 1*DIGIT
DIGIT        =  %x30-39
                ; a digit between 0 and 9, inclusive
```

For readability purposes, it is RECOMMENDED that all the TOI values in the list be given in increasing order. However a receiver MUST be able to handle non-monotonically increasing values. It is also RECOMMENDED to group the TOI equivalence elements together, at the end of the list, in increasing newTOI order. However a receiver MUST be able to handle lists of mixed TOI and TOI equivalence elements. Specifying a TOI equivalence for a given newTOI relieves the sender from specifying newTOI explicitly in the TOI list. However a receiver MUST be able to handle situations where the same TOI appears both in the TOI value and TOI equivalence lists. Finally, a given TOI value or TOI equivalence item MUST NOT be included multiple times in either list.

For instance, the following object list specifies that the current

Carousel Instance is composed of 8 objects, and that TOIs 100 to 104 are equivalent to the TOIs 10 to 14 of Carousel Instance ID 2 and refer to the same objects:

97,98,99,(100=10/2),(101=11/2),(102=12/2),(103=13/2),(104=14/2)

or equivalently:

97-104,(100=10/2),(101=11/2),(102=12/2),(103=13/2),(104=14/2)

6. Security Considerations

6.1. Problem Statement

A content delivery system is potentially subject to attacks. Attacks may target:

- the network (to compromise the routing infrastructure, e.g., by creating congestion),
- the Content Delivery Protocol (CDP) (e.g., to compromise the normal behavior of FCAST), or
- the content itself (e.g., to corrupt the objects being transmitted).

These attacks can be launched either:

- against the data flow itself (e.g., by sending forged packets),
- against the session control parameters (e.g., by corrupting the session description, the CID, the object meta-data, or the ALC/LCT control parameters), that are sent either in-band or out-of-band, or
- against some associated building blocks (e.g., the congestion control component).

In the following sections we provide more details on these possible attacks and sketch some possible counter-measures. We finally provide recommendations in Section 6.5.

6.2. Attacks Against the Data Flow

Let us consider attacks against the data flow first. At least, the following types of attacks exist:

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

- o attacks that try to corrupt the object being transmitted (e.g., to inject malicious code within an object, or to prevent a receiver from using an object, which is a kind of Denial of Service (DoS)).

6.2.1. Access to Confidential Objects

Access control to the object being transmitted is typically provided by means of encryption. This encryption can be done over the whole object (e.g., by the content provider, before submitting the object to FCAST), or be done on a packet per packet basis (e.g., when IPsec/ESP is used [RFC4303], see Section 6.5). If confidentiality is a concern, it is RECOMMENDED that one of these solutions be used.

6.2.2. Object Corruption

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

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

- o at the packet level, each packet can be digitally signed [RMT-SIMPLE-AUTH]. A major limitation is the high computational and transmission overheads that this solution requires. To avoid this problem, the signature may span a set of packets (instead of a single one) in order to amortize the signature calculation. But if a single packets is missing, the integrity of the whole set cannot be checked;

- o at the packet level, a Group Message Authentication Code (MAC) [RFC2104][RMT-SIMPLE-AUTH] scheme can be used, for instance by using HMAC-SHA-256 with a secret key shared by all the group members, senders and receivers. This technique creates a cryptographically secured digest of a packet that is sent along

with the packet. The Group MAC scheme does not create prohibitive processing load nor transmission overhead, but it has a major limitation: it only provides a group authentication/integrity service since all group members share the same secret group key, which means that each member can send a forged packet. It is therefore restricted to situations where group members are fully trusted (or in association with another technique as a pre-check);

- o at the packet level, Timed Efficient Stream Loss-Tolerant Authentication (TESLA) [RFC4082][RFC5776] is an attractive solution that is robust to losses, provides a true authentication/integrity service, and does not create any prohibitive processing load or transmission overhead. Yet checking a packet requires a small delay (a second or more) after its reception;

- o at the packet level, IPsec/ESP [RFC4303] can be used to check the integrity and authenticate the sender of all the packets being exchanged in a session (see Section 6.5).

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

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

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

6.3. Attacks Against the Session Control Parameters and Associated Building Blocks

Let us now consider attacks against the session control parameters and the associated building blocks. The attacker has at least the following opportunities to launch an attack:

- o the attack can target the session description,

- o the attack can target the FCAST CID,

- o the attack can target the meta-data of an object,
- o the attack can target the ALC/LCT parameters, carried within the LCT header or
- o the attack can target the FCAST associated building blocks, for instance the multiple rate congestion control protocol.

The consequences of these attacks are potentially serious, since they can compromise the behavior of content delivery system or even compromise the network itself.

6.3.1. Attacks Against the Session Description

An FCAST receiver may potentially obtain an incorrect Session Description for the session. The consequence of this is that legitimate receivers with the wrong Session Description are unable to correctly receive the session content, or that receivers inadvertently try to receive at a much higher rate than they are capable of, thereby possibly disrupting other traffic in the network.

To avoid these problems, it is RECOMMENDED that measures be taken to prevent receivers from accepting incorrect Session Descriptions. One such measure is the sender authentication to ensure that receivers only accept legitimate Session Descriptions from authorized senders. How these measures are achieved is outside the scope of this document since this session description is usually carried out-of-band.

6.3.2. Attacks Against the FCAST CID

Corrupting the FCAST CID is one way to create a Denial of Service attack. For example, the attacker can set the "Complete" flag to make the receivers believe that no further modification will be done.

It is therefore RECOMMENDED that measures be taken to guarantee the integrity and to check the sender’s identity of the CID. To that purpose, one of the counter-measures mentioned above (Section 6.2.2) SHOULD be used. These measures will either be applied on a packet level, or globally over the whole CID object. When there is no packet level integrity verification scheme, it is RECOMMENDED to digitally sign the CID.

6.3.3. Attacks Against the Object Meta-Data

Corrupting the object meta-data is another way to create a Denial of Service attack. For example, the attacker changes the MD5 sum associated to a file. This possibly leads a receiver to reject the files received, no matter whether the files have been correctly

received or not. When the meta-data are appended to the object, corrupting the meta-data means that the Compound Object will be corrupted.

It is therefore RECOMMENDED that measures be taken to guarantee the integrity and to check the sender's identity of the Compound Object. To that purpose, one of the counter-measures mentioned above (Section 6.2.2) SHOULD be used. These measures will either be applied on a packet level, or globally over the whole Compound Object. When there is no packet level integrity verification scheme, it is RECOMMENDED to digitally sign the Compound Object.

6.3.4. Attacks Against the ALC/LCT and NORM Parameters

By corrupting the ALC/LCT header (or header extensions) one can execute attacks on the underlying ALC/LCT implementation. For example, sending forged ALC packets with the Close Session flag (A) set to one can lead the receiver to prematurely close the session. Similarly, sending forged ALC packets with the Close Object flag (B) set to one can lead the receiver to prematurely give up the reception of an object. The same comments can be made for NORM.

It is therefore RECOMMENDED that measures be taken to guarantee the integrity and to check the sender's identity of each ALC or NORM packet received. To that purpose, one of the counter-measures mentioned above (Section 6.2.2) SHOULD be used.

6.3.5. Attacks Against the Associated Building Blocks

Let us first focus on the congestion control building block that may be used in an ALC or NORM session. A receiver with an incorrect or corrupted implementation of the multiple rate congestion control building block may affect the health of the network in the path between the sender and the receiver. That may also affect the reception rates of other receivers who joined the session.

When congestion control is applied with FCAST, it is therefore RECOMMENDED that receivers be required to identify themselves as legitimate before they receive the Session Description needed to join the session. If authenticating a receiver does not prevent this latter to launch an attack, it will enable the network operator to identify him and to take counter-measures. This authentication can be made either toward the network operator or the session sender (or a representative of the sender) in case of NORM. The details of how it is done are outside the scope of this document.

When congestion control is applied with FCAST, it is also RECOMMENDED that a packet level authentication scheme be used, as explained in

Section 6.2.2. Some of them, like TESLA, only provide a delayed authentication service, whereas congestion control requires a rapid reaction. It is therefore RECOMMENDED [RFC5775] that a receiver using TESLA quickly reduces its subscription level when the receiver believes that a congestion did occur, even if the packet has not yet been authenticated. Therefore TESLA will not prevent DoS attacks where an attacker makes the receiver believe that a congestion occurred. This is an issue for the receiver, but this will not compromise the network. Other authentication methods that do not feature this delayed authentication could be preferred, or a group MAC scheme could be used in parallel to TESLA to prevent attacks launched from outside of the group.

6.4. Other Security Considerations

Lastly, we note that the security considerations that apply to, and are described in, ALC [RFC5775], LCT [RFC5651], NORM [RFC5740] and FEC [RFC5052] also apply to FCAST as FCAST builds on those specifications. In addition, any security considerations that apply to any congestion control building block used in conjunction with FCAST also applies to FCAST. Finally, the security discussion of [RMT-SEC] also applies here.

6.5. Minimum Security Recommendations

We now introduce a mandatory to implement but not necessarily to use security configuration, in the sense of [RFC3365]. Since FCAST/ALC relies on ALC/LCT, it inherits the "baseline secure ALC operation" of [RFC5775]. Similarly, since FCAST/NORM relies on NORM, it inherits the "baseline secure NORM operation" of [RFC5740]. More precisely, in both cases security is achieved by means of IPsec/ESP in transport mode. [RFC4303] explains that ESP can be used to potentially provide confidentiality, data origin authentication, content integrity, anti-replay and (limited) traffic flow confidentiality. [RFC5775] specifies that the data origin authentication, content integrity and anti-replay services SHALL be used, and that the confidentiality service is RECOMMENDED. If a short lived session MAY rely on manual keying, it is also RECOMMENDED that an automated key management scheme be used, especially in case of long lived sessions.

Therefore, the RECOMMENDED solution for FCAST provides per-packet security, with data origin authentication, integrity verification and anti-replay. This is sufficient to prevent most of the in-band attacks listed above. If confidentiality is required, a per-packet encryption SHOULD also be used.

7. IANA Considerations

7.1. Namespace declaration for Object Meta-Data Format

This document requires a IANA registration for the following name-space: "Object Meta-Data Format" (MDFmt). Values in this namespace are 4-bit positive integers between 0 and 15 inclusive and they define the format of the object meta-data ((see Section 5.1).

Initial values for the LCT Header Extension Type registry are defined in Section 7.1.1. Future assignments are to be made through Expert Review [RFC5226].

7.1.1. Object Meta-Data Format registration

This document registers one value in the "Object Meta-Data Format" namespace as follows:

format name	Value
as per HTTP/1.1 metainformation format	0 (default)

All implementations MUST support format 0 (default).

7.2. Namespace declaration for Object Meta-Data Encoding

This document requires a IANA registration for the following name-space: "Object Meta-Data Encoding" (MDEnc). Values in this namespace are 4-bit positive integers between 0 and 15 inclusive and they define the optional encoding of the Object Meta-Data field (see Section 5.1).

Initial values for the LCT Header Extension Type registry are defined in Section 7.2.1. Future assignments are to be made through Expert Review [RFC5226].

7.2.1. Object Meta-Data Encoding registration

This document registers two values in the "Object Meta-Data Encoding" namespace as follows:

Name	Value
plain text	0 (default)
gzip	1

All implementations MUST support both value 0 (plain-text, default) and value 1 (gzip).

8. Acknowledgments

The authors are grateful to the authors of [ALC-00] for specifying the first version of FCAST/ALC. The authors are also grateful to Gorry Fairhurst and Lorenzo Vicisano for their valuable comments.

9. References

9.1. Normative References

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

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC1071] Braden, R., Borman, D., Partridge, C., and W. Plummer, "Computing the Internet checksum", RFC 1071, September 1988.

[RFC5651] Luby, M., Watson, M., and L. Vicisano, "Layered Coding Transport (LCT) Building Block", RFC 5651, October 2009.

[RFC5740] Adamson, B., Bormann, C., Handley, M., and J. Macker, "NACK-Oriented Reliable Multicast (NORM) Transport Protocol", RFC 5740, November 2009.

[RFC5775] Luby, M., Watson, M., and L. Vicisano, "Asynchronous Layered Coding (ALC) Protocol Instantiation", RFC 5775, April 2010.

[RFC1952] Deutsch, P., Gailly, J-L., Adler, M., Deutsch, L., and G. Randers-Pehrson, "GZIP file format specification version 4.3", RFC 1952, May 1996.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

9.2. Informative References

[ALC-00] Luby, M., Gemmell, G., Vicisano, L., Crowcroft, J., and B. Lueckenhoff, "Asynchronous Layered Coding: a Scalable Reliable Multicast Protocol", March 2000.

[RMT-FLUTE] Paila, T., Walsh, R., Luby, M., Roca, V., and R. Lehtonen, "FLUTE - File Delivery over Unidirectional Transport", Work in Progress, February 2011.

[RFC3365] Schiller, J., "Strong Security Requirements for Internet Engineering Task Force Standard Protocols", BCP 61, RFC 3365, August 2002.

[RMT-SEC] Roca, V., Adamson, B., and H. Asaeda, "Security and Reliable Multicast Transport Protocols: Discussions and Guidelines", Work in progress, draft-ietf-rmt-sec-discussion-05.txt, May 2010.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

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

[RFC4082] Perrig, A., Song, D., Canetti, R., Tygar, J., and B. Briscoe, "Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction", RFC 4082, June 2005.

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

[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.

[RFC5776] Roca, V., Francillon, A., and S. Faurite, "Use of Timed

Efficient Stream Loss-Tolerant Authentication (TESLA) in the Asynchronous Layered Coding (ALC) and NACK-Oriented Reliable Multicast (NORM) Protocols", RFC 5776, April 2010.

[RMT-SIMPLE-AUTH]
Roca, V., "Simple Authentication Schemes for the ALC and NORM Protocols", Work in progress draft-ietf-rmt-simple-auth-for-alc-norm-03.txt, July 2010.

Appendix A. FCAST Examples

A.1. Basic Examples

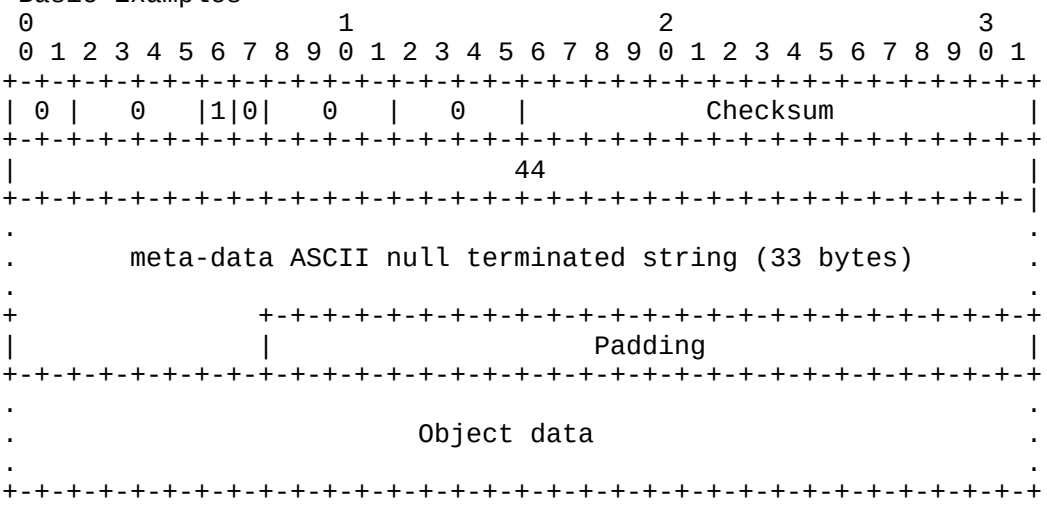

Figure 4: Compound Object Example.

Figure 4 shows a regular Compound Object where the meta-data ASCII string, in HTTP/1.1 meta-information format (MDFmt=0) contains:

Content-Location: example.txt <CR-LF>

This string is 33 bytes long, including the NULL-termination character. There is no gzip encoding of the meta-data (MDEnc=0) and there is no Content-Length information either since this length can easily be calculated by the receiver as the FEC OTI transfer length minus the header length. Finally, the checksum encompasses the whole Compound Object (G=1).

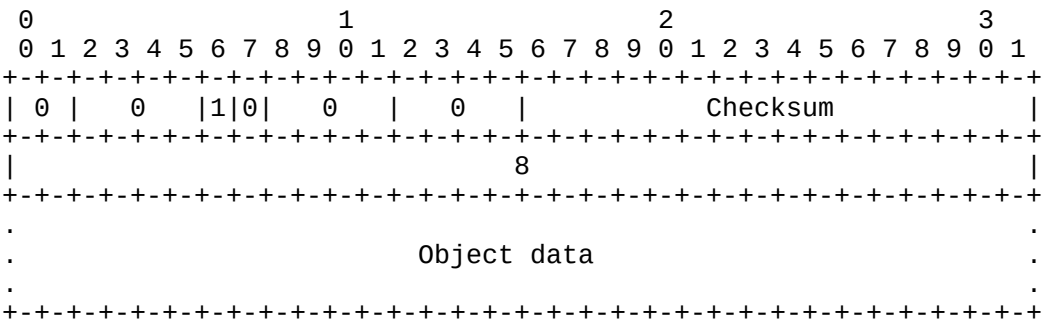

Figure 5: Compound Object Example with no Meta-Data.

Figure 5 shows a Compound Object without any meta-data. The fact there is no meta-data is indicated by the value 8 of the Compound Object Header Length field. No padding is required.

Figure 6 shows an example CID object, in the case of a static FCAST session, i.e., a session where the set of objects is set once and for all. There is no meta-data in this example since Fcast-CID-Complete and Fcast-CID-ID are both implicit.

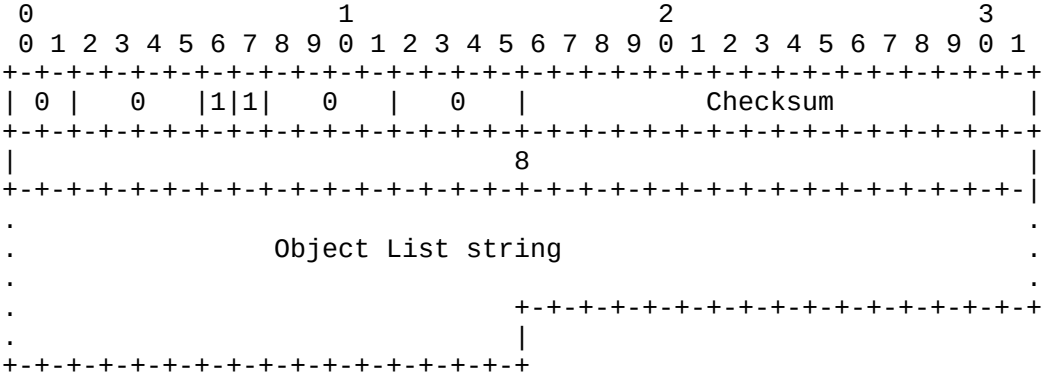

Figure 6: Example of CID, in case of a static session.

The object list contains the following 26 byte long string, including the NULL-termination character:

1,2,3,100-104,200-203,299

There are therefore a total of 3+5+4+1 = 13 objects in the carousel instance, and therefore in the FCAST session. There is no meta-data associated to this CID. The session being static and composed of a single Carousel Instance, the sender did not feel the necessity to carry a Carousel Instance ID meta-data.

A.2. FCAST/NORM with NORM_INFO Examples

In case of FCAST/NORM, the FCAST Compound Object meta-data (or a subset of it) can be carried as part of a NORM_INFO message, as a new Compound Object that does not contain any Compound Object Data. In the following example we assume that the whole meta-data is carried in such a message for a certain Compound Object. Figure 7 shows an example NORM_INFO message that contains the FCAST Compound Object Header and meta-data as its payload. In this example, the first 16 bytes are the NORM_INFO base header, the next 12 bytes are a NORM EXT_FTI header extension containing the FEC Object Transport Information for the associated object, and the remaining bytes are the FCAST Compound Object Header and meta-data. Note that "padding" MUST NOT be used and that the FCAST checksum only encompasses the Compound Object Header (G=0).

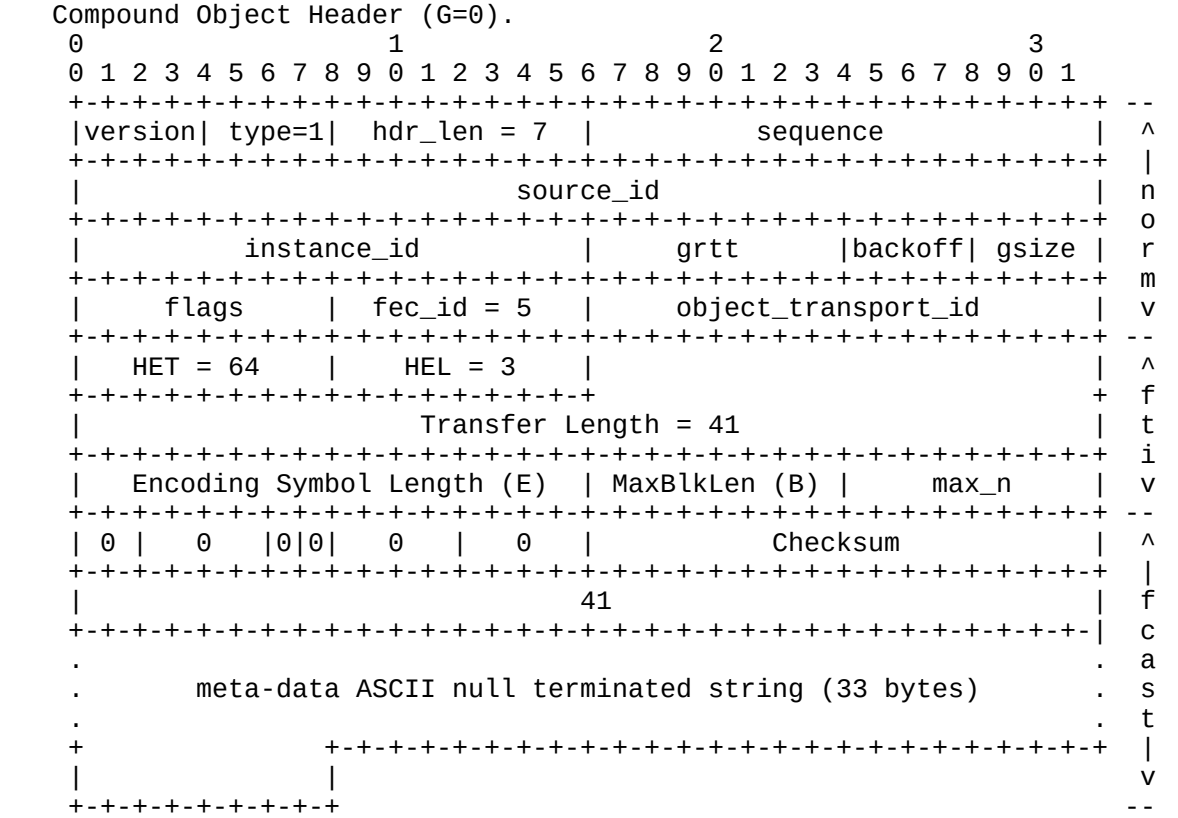

Figure 7: NORM_INFO containing an EXT_FTI header extension and an FCAST Compound Object Header

The NORM_INFO message shown in Figure 7 contains the EXT_FTI header extension to carry the FEC OTI. In this example, the FEC OTI format

is that of the Reed-Solomon FEC coding scheme for fec_id = 5 as described in [RFC5510]. Other alternatives for providing the FEC OTI would have been to either include it directly in the meta-data of the FCAST Compound Header, or to include an EXT_FTI header extension to all NORM_DATA packets (or a subset of them). Note that the NORM "Transfer_Length" is the total length of the associated FCAST Compound Object, i.e., 41 bytes.

The FCAST Compound Object in this example does contain the same meta-data and is formatted as in the example of Figure 4. With the combination of the FEC_OTI and the FCAST meta-data, the NORM protocol and FCAST application have all of the information needed to reliably receive and process the associated object. Indeed, the NORM protocol provides rapid (NORM_INFO has precedence over the associated object content), reliable delivery of the NORM_INFO message and its payload, the FCAST Compound Object Header.

Authors' Addresses

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

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

Brian Adamson
Naval Research Laboratory
Washington, DC 20375
USA

Email: adamson@itd.nrl.navy.mil
URI: http://cs.itd.nrl.navy.mil

Reliable Multicast Transport (RMT)
Internet-Draft
Obsoletes: 3926 (if approved)
Intended status: Standards Track
Expires: August 7, 2011

T. Paila
R. Walsh
Nokia
M. Luby
Qualcomm, Inc.
V. Roca
INRIA
R. Lehtonen
TeliaSonera

February 3, 2011

FLUTE - File Delivery over Unidirectional Transport
draft-ietf-rmt-flute-revised-12

Abstract

This document defines FLUTE, a protocol for the unidirectional delivery of files over the Internet, which is particularly suited to multicast networks. The specification builds on Asynchronous Layered Coding, the base protocol designed for massively scalable multicast distribution. This document obsoletes RFC3926.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 7, 2011.

Copyright Notice

Table of Contents

1. Introduction

This document defines FLUTE version 2, a protocol for unidirectional delivery of files over the Internet. This specification may not be backwards compatible with the previous experimental version defined in [RFC3926]. The specification builds on Asynchronous Layered Coding (ALC), version 1 [ID.ALC-revised], the base protocol designed for massively scalable multicast distribution. ALC defines transport of arbitrary binary objects. For file delivery applications mere transport of objects is not enough, however. The end systems need to know what the objects actually represent. This document specifies a technique called FLUTE - a mechanism for signaling and mapping the properties of files to concepts of ALC in a way that allows receivers to assign those parameters for received objects. Consequently, throughout this document the term 'file' relates to an 'object' as discussed in ALC. Although this specification frequently makes use of multicast addressing as an example, the techniques are similarly applicable for use with unicast addressing.

This document defines a specific transport application of ALC, adding the following specifications:

- Definition of a file delivery session built on top of ALC, including transport details and timing constraints.

- In-band signaling of the transport parameters of the ALC session.

- In-band signaling of the properties of delivered files.

- Details associated with the multiplexing of multiple files within a session.

This specification is structured as follows. Section 3 begins by defining the concept of the file delivery session. Following that it introduces the File Delivery Table that forms the core part of this specification. Further, it discusses multiplexing issues of transmission objects within a file delivery session. Section 4 describes the use of congestion control and channels with FLUTE. Section 5 defines how the Forward Error Correction (FEC) Object Transmission Information is to be delivered within a file delivery session. Section 6 defines the required parameters for describing file delivery sessions in a general case. Section 7 outlines security considerations regarding file delivery with FLUTE. Last, there are two informative appendices. Appendix A describes an envisioned receiver operation for the receiver of the file delivery session. Readers who want to see a simple example of FLUTE in operation should refer to Appendix A right away. Appendix B gives an example of a File Delivery Table.

This specification contains part of the definitions necessary to fully specify a Reliable Multicast Transport protocol in accordance with RFC2357.

This document obsoletes RFC3926 which contained a previous version of this specification and was published in the "Experimental" category. This Proposed Standard specification is thus based on RFC3926 updated according to accumulated experience and growing protocol maturity since the publication of RFC3926. Said experience applies both to this specification itself and to congestion control strategies related to the use of this specification.

The differences between RFC3926 and this document are listed in Section 11.

1.1. Applicability Statement

1.1.1. The Target Application Space

FLUTE is applicable to the delivery of large and small files to many hosts, using delivery sessions of several seconds or more. For instance, FLUTE could be used for the delivery of large software updates to many hosts simultaneously. It could also be used for continuous, but segmented, data such as time-lined text for subtitling - potentially leveraging its layering inheritance from ALC and LCT to scale the richness of the session to the congestion status of the network. It is also suitable for the basic transport of metadata, for example SDP [RFC.SDP] files which enable user applications to access multimedia sessions.

1.1.2. The Target Scale

Massive scalability is a primary design goal for FLUTE. IP multicast is inherently massively scalable, but the best effort service that it provides does not provide session management functionality, congestion control or reliability. FLUTE provides all of this using ALC and IP multicast without sacrificing any of the inherent scalability of IP multicast.

1.1.3. Intended Environments

All of the environmental requirements and considerations that apply to the RMT Building Blocks used by FLUTE shall also apply to FLUTE. These are the ALC protocol instantiation [ID.ALC-revised], the Layered Coding Transport (LCT) Building Block [RFC5651] and the FEC Building Block [RFC5052].

FLUTE can be used with both multicast and unicast delivery, but it’s

primary application is for unidirectional multicast file delivery. FLUTE requires connectivity between a sender and receivers but does not require connectivity from receivers to a sender. FLUTE inherently works with all types of networks, including LANs, WANs, Intranets, the Internet, asymmetric networks, wireless networks, and satellite networks.

FLUTE is compatible with both IPv4 or IPv6 as no part of the packet is IP version specific. FLUTE works with both multicast models: Any-Source Multicast (ASM) [RFC.ASM] and the Source-Specific Multicast (SSM) [PAPER.SSM].

FLUTE is applicable for both Internet use, with a suitable congestion control building block, and provisioned/controlled systems, such as delivery over wireless broadcast radio systems.

1.1.4. Weaknesses

FLUTE congestion control protocols depend on the ability of a receiver to change multicast subscriptions between multicast groups supporting different rates and/or layered codings. If the network does not support this, then the FLUTE congestion control protocols may not be amenable to these networks

FLUTE can also be used for point-to-point (unicast) communications. At a minimum, implementations of ALC MUST support the Wave and Equation Based Rate Control (WEBRC) [RFC.3738] multiple rate congestion control scheme [ID.ALC-revised]. However, since WEBRC has been designed for massively scalable multicast flows, it is not clear how appropriate it is to the particular case of unicast flows. Using a separate point-to-point congestion control scheme is another alternative. How to do that is outside the scope of the present document.

FLUTE provides reliability using the FEC building block. This will reduce the error rate as seen by applications. However, FLUTE does not provide a method for senders to verify the reception success of receivers, and the specification of such a method is outside the scope of this document.

2. Conventions used in this Document

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 [RFC.2119].

The terms "object" and "transmission object" are consistent with the

definitions in ALC [ID.ALC-revised] and LCT [RFC.LCT]. The terms "file" and "source object" are pseudonyms for "object".

3. File delivery

Asynchronous Layered Coding [ID.ALC-revised] is a protocol designed for delivery of arbitrary binary objects. It is especially suitable for massively scalable, unidirectional, multicast distribution. ALC provides the basic transport for FLUTE, and thus FLUTE inherits the requirements of ALC.

This specification is designed for the delivery of files. The core of this specification is to define how the properties of the files are carried in-band together with the delivered files.

As an example, let us consider a 5200 byte file referred to by "http://www.example.com/docs/file.txt". Using the example, the following properties describe the properties that need to be conveyed by the file delivery protocol.

* Identifier of the file, expressed as a URI. The identifier MAY provide a location for the file. In the above example: "http://www.example.com/docs/file.txt".

* File name (usually, this can be concluded from the URI). In the above example: "file.txt".

* File type, expressed as MIME media type. In the above example: "text/plain".

* File size, expressed in octets. In the above example: "5200". If the file is content encoded then this is the file size before content encoding.

* Content encoding of the file, within transport. In the above example, the file could be encoded using ZLIB [RFC.ZLIB]. In this case the size of the transmission object carrying the file would probably differ from the file size. The transmission object size is delivered to receivers as part of the FLUTE protocol.

* Security properties of the file such as digital signatures, message digests, etc. For example, one could use S/MIME [RFC.SMIME] as the content encoding type for files with this authentication wrapper, and one could use XML-DSIG [RFC.XML-DSIG] to digitally sign the file. XML-DSIG can also be used to provide tamper prevention e.g. on the Content-Location field. Content encoding is applied to file data before FEC protection.

For each unique file, FLUTE encodes the attributes listed above and other attributes as children of an XML file element. A table of XML file elements is transmitted as a special file called a 'File Delivery Table' (FDT) which is further described in the next subsection and in section 3.2

3.1. File delivery session

ALC is a protocol instantiation of Layered Coding Transport building block (LCT) [RFC.LCT]. Thus ALC inherits the session concept of LCT. In this document we will use the concept ALC/LCT session to collectively denote the interchangeable terms ALC session and LCT session.

An ALC/LCT session consists of a set of logically grouped ALC/LCT channels associated with a single sender sending ALC/LCT packets for one or more objects. An ALC/LCT channel is defined by the combination of a sender and an address associated with the channel by the sender. A receiver joins a channel to start receiving the data packets sent to the channel by the sender, and a receiver leaves a channel to stop receiving data packets from the channel.

One of the fields carried in the ALC/LCT header is the Transport Session Identifier (TSI). The (source IP address, TSI) pair uniquely identifies a session. Note that the TSI is scoped by the IP address, so the same TSI may be used by several source IP addresses at once. Thus, the receiver uses the (source IP address, TSI) pair from each packet to uniquely identify the session sending each packet. When a session carries multiple objects, the Transmission Object Identifier (TOI) field within the ALC/LCT header names the object used to generate each packet. Note that each object is associated with a unique TOI within the scope of a session.

A FLUTE session consistent with this specification MUST use FLUTE version 2 as specified in this document. Thus, all sessions consistent with this specification MUST set the FLUTE version to 2. The FLUTE version is carried within the EXT_FDT extension header (defined in section 3.4.1) in the ALC/LCT layer. A FLUTE session consistent with this specification MUST use ALC version 1 as specified in RFC 5775, and LCT version 1 as specified in RFC 5651.

If multiple FLUTE sessions are sent to a channel then receivers MUST determine the FLUTE protocol version, based on version fields and the (source IP address, TSI) carried in the ALC/LCT header of the packet. Note that when a receiver first begins receiving packets, it MAY NOT know the FLUTE protocol version, as not every LCT packet carries the EXT_FDT header (containing the FLUTE protocol version.) A new receiver MAY keep an open binding in the LCT protocol layer between

the TSI and the FLUTE protocol version, until the EXT_FDT header arrives. Alternately, a new receiver MAY discover a binding between TSI and FLUTE protocol version via a session discovery protocol that is out of scope in this document.

If the sender is not assigned a permanent IP address accessible to receivers, then packets that can be received by receivers contain a temporary IP address. In this case the TSI is scoped by this temporary IP address of the sender for the duration of the session. As an example, the sender may be behind a Network Address Translation (NAT) device that temporarily assigns an IP address for the sender. In this case the TSI is scoped by the temporary IP address assigned by the NAT. As another example, the sender may send its original packets using IPv6, but some portions of the network may not be IPv6 capable. Thus, there may be an IPv6 to IPv4 translator that changes the IP address of the packets to a different IPv4 address. In this case, receivers in the IPv4 portion of the network will receive packets containing the IPv4 address, and thus the TSI for them is scoped by the IPv4 address. How the IP address of the sender to be used to scope the session by receivers is delivered to receivers, whether it is a permanent IP address or a temporary IP address, is outside the scope of this document.

When FLUTE is used for file delivery over ALC the following rules apply:

* The ALC/LCT session is called a file delivery session.

* The ALC/LCT concept of 'object' denotes either a 'file' or a 'File Delivery Table Instance' (section 3.2)

* The TOI field MUST be included in ALC packets sent within a FLUTE session, with the exception that ALC packets sent in a FLUTE session with the Close Session (A) flag set to 1 (signaling the end of the session) and that contain no payload (carrying no information for any file or FDT) SHALL NOT carry the TOI. See section 5.1 of RFC 5651 [RFC.LCT] for the LCT definition of the Close Session flag, and see section 4.2 of RFC 5775 [ID.ALC-revised] for an example of the use of a TOI within an ALC packet.

* The TOI value '0' is reserved for delivery of File Delivery Table Instances. Each non expired File Delivery Table Instance is uniquely identified by an FDT Instance ID within the EXT_FDT header defined in section 3.4.1.

* Each file in a file delivery session MUST be associated with a TOI (>0) in the scope of that session.

* Information carried in the headers and the payload of a packet is scoped by the source IP address and the TSI. Information particular to the object carried in the headers and the payload of a packet is further scoped by the TOI for file objects, and is further scoped by both the TOI and the FDT Instance ID for FDT Instance objects.

3.2. File Delivery Table

The File Delivery Table (FDT) provides a means to describe various attributes associated with files that are to be delivered within the file delivery session. The following lists are examples of such attributes, and are not intended to be mutually exclusive nor exhaustive.

Attributes related to the delivery of file:

- TOI value that represents the file

- FEC Object Transmission Information (including the FEC Encoding ID and, if relevant, the FEC Instance ID)

- Size of the transmission object carrying the file

- Aggregate rate of sending packets to all channels

Attributes related to the file itself:

- Name, Identification and Location of file (specified by the URI)

- MIME media type of file

- Size of file

- Encoding of file

- Message digest of file

Some of these attributes MUST be included in the file description entry for a file, others are optional, as defined in section 3.4.2.

Logically, the FDT is a set of file description entries for files to be delivered in the session. Each file description entry MUST include the TOI for the file that it describes and the URI identifying the file. The TOI carried in each file description entry

is how FLUTE names the ALC/LCT data packets used for delivery of the file. Each file description entry may also contain one or more descriptors that map the above-mentioned attributes to the file.

Each file delivery session MUST have an FDT that is local to the given session. The FDT MUST provide a file description entry mapped to a TOI for each file appearing within the session. An object that is delivered within the ALC session, but not described in the FDT, other than the FDT itself, is not considered a 'file' belonging to the file delivery session. Handling of these unmapped TOIs (Non-zero TOIs that are not resolved by the FDT) is out of scope of this specification.

Within the file delivery session the FDT is delivered as FDT Instances. An FDT Instance contains one or more file description entries of the FDT. Any FDT Instance can be equal to, a subset of, a superset of, overlap with or complement any other FDT Instance. A certain FDT Instance may be repeated multiple times during a session, even after subsequent FDT Instances (with higher FDT Instance ID numbers) have been transmitted. Each FDT Instance contains at least a single file description entry and at most the exhaustive set of file description entries of the files being delivered in the file delivery session.

A receiver of the file delivery session keeps an FDT database for received file description entries. The receiver maintains the database, for example, upon reception of FDT Instances. Thus, at any given time the contents of the FDT database represent the receiver's current view of the FDT of the file delivery session. Since each receiver behaves independently of other receivers, it SHOULD NOT be assumed that the contents of the FDT database are the same for all the receivers of a given file delivery session.

Since the FDT database is an abstract concept, the structure and the maintenance of the FDT database are left to individual implementations and are thus out of scope of this specification.

3.3. Dynamics of FDT Instances within file delivery session

The following rules define the dynamics of the FDT Instances within a file delivery session:

* For every file delivered within a file delivery session there MUST be a file description entry included in at least one FDT Instance sent within the session. A file description entry contains at a minimum the mapping between the TOI and the URI.

* An FDT Instance MAY appear in any part of the file delivery session and packets for an FDT Instance MAY be interleaved with packets for other files or other FDT Instances within a session.

* The TOI value of '0' MUST be reserved for delivery of FDT Instances. The use of other TOI values for FDT Instances is outside the scope of this specification.

* The FDT Instance is identified by the use of a new fixed length LCT Header Extension EXT_FDT (defined later in this section.) Each non expired FDT Instance is uniquely identified within the file delivery session by its FDT Instance ID, carried by the EXT_FDT Header Extension. Any ALC/LCT packet carrying an FDT Instance MUST include EXT_FDT.

* It is RECOMMENDED that an FDT Instance that contains the file description entry for a file is sent at least once before sending the described file within a file delivery session. This recommendation is intended to minimize the amount of file data which may be received by receivers in advance of the FDT Instance containing the entry for a file (such data must either be speculatively buffered or discarded). Note that this possibility cannot be completely eliminated since the first transmission of FDT data may be lost.

* Within a file delivery session, any TOI > 0 MAY be described more than once. An example: previous FDT Instance 0 describes TOI of value '3'. Now, subsequent FDT Instances can either keep TOI '3' unmodified on the table, not include it, or augment the description. However, subsequent FDT Instances MUST NOT change the parameters already described for a specific TOI.

* An FDT Instance is valid until its expiration time. The expiration time is expressed within the FDT Instance payload as an UTF-8 decimal representation of a 32 bit unsigned integer. The value of this integer represents the 32 most significant bits of a 64 bit Network Time Protocol (NTP) [RFC.NTP] time value. These 32 bits provide an unsigned integer representing the time in seconds relative to 0 hours 1 January 1900 in case of the prime epoch (era 0) [NTPv4]. The handling of time wraparound (to happen in 2036) requires to consider the associated epoch. In any case, both a sender and a receiver can determine to which (136 year) epoch the FDT Instance expiration time value pertains to by choosing the epoch for which the expiration time is closest in time to the current time.

* The space of FDT Instance IDs is limited and so senders should take care to always have a large enough supply of FDT Instance IDs corresponding to unexpired FDTs when specifying FDT expiration times.

* The receiver SHOULD NOT use a received FDT Instance to interpret packets received beyond the expiration time of the FDT Instance.

* A sender MUST use an expiration time in the future upon creation of an FDT Instance relative to its Sender Current Time (SCT).

* Any FEC Encoding ID MAY be used for the sending of FDT Instances. The default is to use the Compact No-code FEC Encoding ID 0 [RFC.FECSchemes] for the sending of FDT Instances. (Note that since FEC Encoding ID 0 is the default for FLUTE, this implies that Source Block Number and Encoding Symbol ID lengths both default to 16 bits each.)

* If the receiver does not understand the FEC Encoding ID in a FDT Instance, the receiver MUST NOT decode the associated FDT.

* It is RECOMMENDED that the mechanisms used for file attribute delivery should achieve a delivery probability that is higher than the file recovery probability and the file attributes should be delivered at this higher priority before the delivery of the associated files begins.

Generally, a receiver needs to receive an FDT Instance describing a file before it is able to recover the file itself. In this sense FDT Instances are of higher priority than files. Additionally, a FLUTE sender SHOULD assume receivers will not receive all packets pertaining to FDT Instances. The way FDT Instances are transmitted has a large impact on satisfying the recommendation above. When there is a single file transmitted in the session, one way to satisfy the recommendation above is to repeatedly transmit on a regular enough basis FDT Instances describing the file while the file is being transmitted. If an FDT Instance is longer than one packet payload in length, it is RECOMMENDED that an FEC code that provides protection against loss be used for delivering this FDT Instance. When there are multiple files in a session concurrently being transmitted to receivers, the way the FDT Instances are structured and transmitted also has a large impact. As an example, a way to satisfy the recommendation above is to transmit an FDT Instance that describes all files currently being transmitted, and to transmit this FDT Instance reliably, using the same techniques as explained for the case when there is a single file transmitted in a session. If instead the concurrently transmitted files are described in separate FDT Instances, another way to satisfy this recommendation is to

transmit all the relevant FDT Instances reliably, using the same techniques as explained for the case when there is a single file transmitted in a session.

In any case, how often the description of a file is sent in an FDT Instance, how often an FDT Instance is sent, and how much FEC protection is provided for an FDT Instance (if longer than one packet payload) are dependent on the particular application and are outside the scope of this document.

Sometimes the various attributes associated with files that are to be delivered within the file delivery session are sent out-of-band (rather than in-band, within one or several FDT Instances). The details of how this is done are out of the scope of this document. However, it is still RECOMMENDED that any out-of-band transmission be managed in such a way that a receiver will be able to recover the attributes associated with a file with as much or greater reliability as the receiver is able to receive enough packets containing encoding symbols to recover the file. For example, the probability of a randomly chosen receiver being able to recover a given file can often be estimated based on a statistical model of reception conditions, the amount of data transmitted and the properties of any Forward Error Correction in use. The recommendation above suggests that mechanisms used for file attribute delivery should achieve higher a delivery probability than the file recovery probability.

3.4. Structure of FDT Instance packets

FDT Instances are carried in ALC packets with TOI = 0 and with an additional REQUIRED LCT Header extension called the FDT Instance Header. The FDT Instance Header (EXT_FDT) contains the FDT Instance ID that uniquely identifies FDT Instances within a file delivery session. The FDT Instance Header is placed in the same way as any other LCT extension header. There MAY be other LCT extension headers in use.

The FDT Instance is encoded for transmission, like any other object, using an FEC Scheme (which MAY be the Compact No-Code FEC Scheme) The LCT extension headers are followed by the FEC Payload ID, and finally the Encoding Symbols for the FDT Instance which contains one or more file description entries. A FDT Instance MAY span several ALC packets - the number of ALC packets is a function of the file attributes associated with the FDT Instance. The FDT Instance Header is carried in each ALC packet carrying the FDT Instance. The FDT Instance Header is identical for all ALC/LCT packets for a particular FDT Instance.

The overall format of ALC/LCT packets carrying an FDT Instance is

depicted in the Figure 1 below. All integer fields are carried in "big-endian" or "network order" format, that is, most significant byte (octet) first. As defined in [ID.ALC-revised], all ALC/LCT packets are sent using UDP.

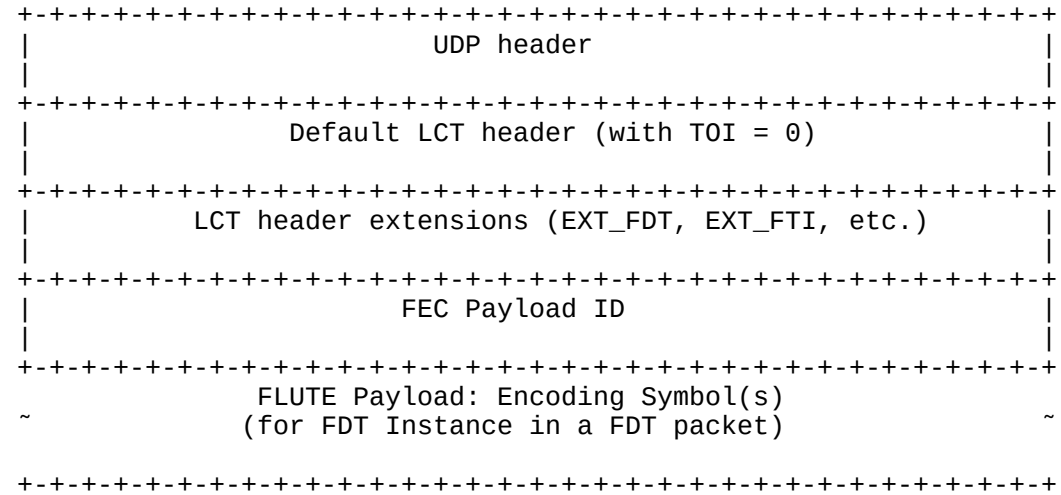

Figure 1: Overall FDT Packet

3.4.1. Format of FDT Instance Header

The FDT Instance Header (EXT_FDT) is a new fixed length, ALC PI specific LCT header extension [RFC.LCT]. The Header Extension Type (HET) for the extension is 192. Its format is defined below:

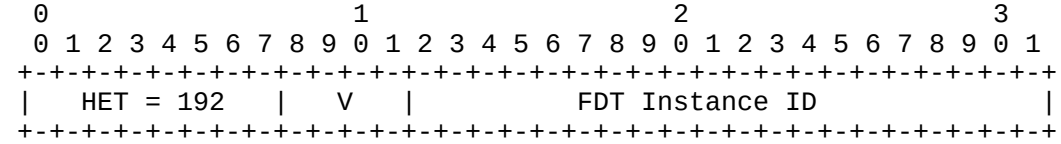

Figure 2

Version of FLUTE (V), 4 bits:

This document specifies FLUTE version 2. Hence in any ALC packet that carries FDT Instance and that belongs to the file delivery session as specified in this specification MUST set this field to '2'.

FDT Instance ID, 20 bits:

For each file delivery session the numbering of FDT Instances starts from '0' and is incremented by one for each subsequent FDT Instance. After reaching the maximum value (2^20-1), the numbering starts from the smallest FDT Instance value assigned to an expired FDT Instance. When wraparound from a greater FDT Instance ID value to a smaller FDT Instance ID value occurs, the smaller FDT Instance ID value is considered logically higher than the greater FDT Instance ID value. Senders SHOULD NOT re-use an FDT Instance ID value that is already in use for a non-expired FDT Instance. Sender behavior when all the FDT Instance IDs are used by non expired FEC Instances is outside the scope of this specification and left to individual implementations of FLUTE. Receipt of an FDT Instance that reuses an FDT Instance ID value that is currently used by a non expired FDT Instance SHOULD be considered as an error case. Receiver behavior in this case is outside the scope of this specification and left to individual implementations of FLUTE. Receivers MUST be ready to handle FDT Instance ID wraparound and situations where missing FDT Instance IDs result in increments larger than one.

3.4.2. Syntax of FDT Instance

The FDT Instance contains file description entries that provide the mapping functionality described in 3.2 above.

The FDT Instance is an XML structure that has a single root element "FDT-Instance". The "FDT-Instance" element MUST contain "Expires" attribute, which tells the expiration time of the FDT Instance. In addition, the "FDT-Instance" element MAY contain the "Complete" attribute (boolean), which, when TRUE, signals that this "FDT Instance" includes the set of "File" entries that exhausts both the set of files delivered so far and also the set of files to be delivered in the session. This implies that no new data will be provided in future FDT Instances within this session (i.e., that either FDT Instances with higher ID numbers will not be used or if they are used, will only provide identical file parameters to those already given in this and previous FDT Instances). The "Complete" attribute is therefore used to provide a complete list of files in an entire FLUTE session (a "complete FDT").

The "FDT-Instance" element MAY contain attributes that give common parameters for all files of an FDT Instance. These attributes MAY also be provided for individual files in the "File" element. Where the same attribute appears in both the "FDT-Instance" and the "File" elements, the value of the attribute provided in the "File" element takes precedence.

For each file to be declared in the given FDT Instance there is a single file description entry in the FDT Instance. Each entry is

represented by element "File" which is a child element of the FDT Instance structure.

The attributes of "File" element in the XML structure represent the attributes given to the file that is delivered in the file delivery session. The value of the XML attribute name corresponds to MIME field name and the XML attribute value corresponds to the value of the MIME field body. Each "File" element MUST contain at least two attributes "TOI" and "Content-Location". "TOI" MUST be assigned a valid TOI value as described in section 3.3 above. "Content-Location" MUST be assigned a valid URI as defined in [RFC.HTTP11] which identifies the object to be delivered, for example a URI with the "http" or "file" URI scheme. The semantics for any two "File" elements declaring the same "Content-Location" but differing "TOI" is that the element appearing in the FDT Instance with the greater FDT Instance ID is considered to declare newer instance (e.g. version) of the same "File".

In addition to mandatory attributes, the "FDT-Instance" element and the "File" element MAY contain other attributes of which the following are specifically pointed out.

* The attribute "Content-Type" SHOULD be included and, when present, MUST be used for the purpose defined in [RFC.HTTP11].

* Where the length is described, the attribute "Content-Length" MUST be used for the purpose as defined in [RFC.HTTP11]. The transfer length is defined to be the length of the object transported in octets. It is often important to convey the transfer length to receivers, because the source block structure needs to be known for the FEC decoder to be applied to recover source blocks of the file, and the transfer length is often needed to properly determine the source block structure of the file. There generally will be a difference between the length of the original file and the transfer length if content encoding is applied to the file before transport, and thus the "Content-Encoding" attribute is used. If the file is not content encoded before transport (and thus the "Content-Encoding" attribute is not used) then the transfer length is the length of the original file, and in this case the "Content-Length" is also the transfer length. However, if the file is content encoded before transport (and thus the "Content-Encoding" attribute is used), e.g., if compression is applied before transport to reduce the number of octets that need to be transferred, then the transfer length is generally different than the length of the original file, and in this case the attribute "Transfer-Length" MAY be used to carry the transfer length.

* Whenever content encoding is applied the attribute "Content-Encoding" MUST be included. Whenever the attribute "Content-Encoding" is included it MUST be used as described in [RFC.HTTP11].

* Where the MD5 message digest is described, the attribute "Content-MD5" MUST be used for the purpose as defined in [RFC.HTTP11].

* The FEC Object Transmission Information attributes as described in section 5.2.

The following specifies the XML Schema [XML-Schema-Part-1][XML-Schema-Part-2] for FDT Instance:

```
BEGIN
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="urn:ietf:params:xml:ns:fdt"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="urn:ietf:params:xml:ns:fdt"
           elementFormDefault="qualified">
  <xs:element name="FDT-Instance" type="FDT-InstanceType"/>
  <xs:complexType name="FDT-InstanceType">
    <xs:sequence>
      <xs:element name="File" type="FileType" maxOccurs="unbounded"/>
      <xs:any namespace="##other" processContents="skip"
              minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="Expires"
                  type="xs:string"
                  use="required"/>
    <xs:attribute name="Complete"
                  type="xs:boolean"
                  use="optional"/>
    <xs:attribute name="Content-Type"
                  type="xs:string"
                  use="optional"/>
    <xs:attribute name="Content-Encoding"
                  type="xs:string"
                  use="optional"/>
    <xs:attribute name="FEC-OTI-FEC-Encoding-ID"
                  type="xs:unsignedByte"
                  use="optional"/>
    <xs:attribute name="FEC-OTI-FEC-Instance-ID"
                  type="xs:unsignedLong"
                  use="optional"/>
    <xs:attribute name="FEC-OTI-Maximum-Source-Block-Length"
                  type="xs:unsignedLong"
                  use="optional"/>
```

```
      <xs:attribute name="FEC-OTI-Encoding-Symbol-Length"
                    type="xs:unsignedLong"
                    use="optional"/>
      <xs:attribute name="FEC-OTI-Max-Number-of-Encoding-Symbols"
                    type="xs:unsignedLong"
                    use="optional"/>
      <xs:attribute name="FEC-OTI-Scheme-Specific-Info"
                    type="xs:base64Binary"
                    use="optional"/>
      <xs:anyAttribute processContents="skip"/>
    </xs:complexType>
    <xs:complexType name="FileType">
      <xs:sequence>
        <xs:any namespace="##other" processContents="skip"
                minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="Content-Location"
                    type="xs:anyURI"
                    use="required"/>
      <xs:attribute name="TOI"
                    type="xs:positiveInteger"
                    use="required"/>
      <xs:attribute name="Content-Length"
                    type="xs:unsignedLong"
                    use="optional"/>
      <xs:attribute name="Transfer-Length"
                    type="xs:unsignedLong"
                    use="optional"/>
      <xs:attribute name="Content-Type"
                    type="xs:string"
                    use="optional"/>
      <xs:attribute name="Content-Encoding"
                    type="xs:string"
                    use="optional"/>
      <xs:attribute name="Content-MD5"
                    type="xs:base64Binary"
                    use="optional"/>
      <xs:attribute name="FEC-OTI-FEC-Encoding-ID"
                    type="xs:unsignedByte"
                    use="optional"/>
      <xs:attribute name="FEC-OTI-FEC-Instance-ID"
                    type="xs:unsignedLong"
                    use="optional"/>
      <xs:attribute name="FEC-OTI-Maximum-Source-Block-Length"
                    type="xs:unsignedLong"
                    use="optional"/>
      <xs:attribute name="FEC-OTI-Encoding-Symbol-Length"
                    type="xs:unsignedLong"
```

```
                use="optional"/>
      <xs:attribute name="FEC-OTI-Max-Number-of-Encoding-Symbols"
                type="xs:unsignedLong"
                use="optional"/>
      <xs:attribute name="FEC-OTI-Scheme-Specific-Info"
                type="xs:base64Binary"
                use="optional"/>
      <xs:anyAttribute processContents="skip"/>
    </xs:complexType>
  </xs:schema>
  END
```

Figure 3

Any valid FDT Instance MUST use the above XML Schema. This way FDT provides extensibility to support private attributes within the file description entries. Those could be, for example, the attributes related to the delivery of the file (timing, packet transmission rate, etc.).

In case the basic FDT XML Schema is extended in terms of new descriptors (attributes or elements), for descriptors applying to a single file, those MUST be placed within the element "File". For descriptors applying to all files described by the current FDT Instance, those MUST be placed within the element "FDT-Instance". It is RECOMMENDED that the new attributes applied in the FDT are in the format of MIME fields and are either defined in the HTTP/1.1 specification [RFC.HTTP11] or another well-known specification.

3.4.3. Content Encoding of FDT Instance

The FDT Instance itself MAY be content encoded, for example compressed. This specification defines FDT Instance Content Encoding Header (EXT_CENC). EXT_CENC is a new fixed length LCT header extension [RFC.LCT]. The Header Extension Type (HET) for the extension is 193. If the FDT Instance is content encoded, the EXT_CENC MUST be used to signal the content encoding type. In that case, EXT_CENC header extension MUST be used in all ALC packets carrying the same FDT Instance ID. Consequently, when EXT_CENC header is used, it MUST be used together with a proper FDT Instance Header (EXT_FDT). Within a file delivery session, FDT Instances that are not content encoded and FDT Instances that are content encoded MAY both appear. If content encoding is not used for a given FDT Instance, the EXT_CENC MUST NOT be used in any packet carrying the FDT Instance. The format of EXT_CENC is defined below:

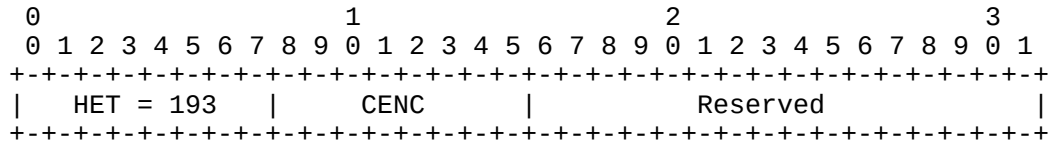

Figure 4

Content Encoding Algorithm (CENC), 8 bits:

This field signals the content encoding algorithm used in the FDT Instance payload. This subsection reserves the Content Encoding Algorithm values 0, 1, 2 and 3 for null, ZLIB [RFC.ZLIB], DEFLATE [RFC.DEFLATE] and GZIP [RFC.GZIP] respectively.

Reserved, 16 bits:

This field MUST be set to all '0'. This field SHOULD be ignored on reception.

3.5. Multiplexing of files within a file delivery session

The delivered files are carried as transmission objects (identified with TOIs) in the file delivery session. All these objects, including the FDT Instances, MAY be multiplexed in any order and in parallel with each other within a session, i.e., packets for one file may be interleaved with packets for other files or other FDT Instances within a session.

Multiple FDT Instances MAY be delivered in a single session using TOI = 0. In this case, it is RECOMMENDED that the sending of a previous FDT Instance SHOULD end before the sending of the next FDT Instance starts. However, due to unexpected network conditions, packets for the FDT Instances MAY be interleaved. A receiver can determine which FDT Instance a packet contains information about since the FDT Instances are uniquely identified by their FDT Instance ID carried in the EXT_FDT headers.

4. Channels, congestion control and timing

ALC/LCT has a concept of channels and congestion control. There are four scenarios in which FLUTE is envisioned to be applied.

(a) Use of a single channel and a single-rate congestion control protocol.

(b) Use of multiple channels and a multiple-rate congestion control protocol. In this case the FDT Instances MAY be delivered on more than one channel.

(c) Use of a single channel without congestion control supplied by ALC, but only when in a controlled network environment where flow/ congestion control is being provided by other means.

(d) Use of multiple channels without congestion control supplied by ALC, but only when in a controlled network environment where flow/ congestion control is being provided by other means. In this case the FDT Instances MAY be delivered on more than one channel.

When using just one channel for a file delivery session, as in (a) and (c), the notion of 'prior' and 'after' are intuitively defined for the delivery of objects with respect to their delivery times.

However, if multiple channels are used, as in (b) and (d), it is not straightforward to state that an object was delivered 'prior' to the other. An object may begin to be delivered on one or more of those channels before the delivery of a second object begins. However, the use of multiple channels/layers may complete the delivery of the second object before the first. This is not a problem when objects are delivered sequentially using a single channel. Thus, if the application of FLUTE has a mandatory or critical requirement that the first transmission object must complete 'prior' to the second one, it is RECOMMENDED that only a single channel is used for the file delivery session.

Furthermore, if multiple channels are used then a receiver joined to the session at a low reception rate will only be joined to the lower layers of the session. Thus, since the reception of FDT Instances is of higher priority than the reception of files (because the reception of files depends on the reception of an FDT Instance describing it), the following is RECOMMENDED:

1. The layers to which packets for FDT Instances are sent SHOULD NOT be biased towards those layers to which lower rate receivers are not joined. For example, it is okay to put all the packets for an FDT Instance into the lowest layer (if this layer carries enough packets to deliver the FDT to higher rate receivers in a reasonable amount of time), but it is not okay to put all the packets for an FDT Instance into the higher layers that only high rate receivers will receive.

2. If FDT Instances are generally longer than one Encoding Symbol in length and some packets for FDT Instances are sent to layers that lower rate receivers do not receive, an FEC Encoding other than Compact No-code FEC Encoding ID 0 [RFC.FECSchemes] SHOULD be used to deliver FDT Instances. This is because in this case, even when there is no packet loss in the network, a lower rate receiver will not receive all packets sent for an FDT Instance.

5. Delivering FEC Object Transmission Information

FLUTE inherits the use of FEC building block [RFC5052] from ALC. When using FLUTE for file delivery over ALC the FEC Object Transmission Information MUST be delivered in-band within the file delivery session. There are two methods to achieve this: the use of ALC specific LCT extension header EXT_FTI [ID.ALC-revised] and the use of FDT. The latter method is specified in this section. The use of EXT_FTI requires repetition of the FEC Object Transmission Information to ensure reception (though not necessarily in every packet) and thus may entail higher overhead than the use of the FDT, but may also provide more timely delivery of the FEC Object Transmission Information.

The receiver of file delivery session MUST support delivery of FEC Object Transmission Information using the EXT_FTI for the FDT Instances carried using TOI value 0. For the TOI values other than 0 the receiver MUST support both methods: the use of EXT_FTI and the use of FDT.

The FEC Object Transmission Information that needs to be delivered to receivers MUST be exactly the same whether it is delivered using EXT_FTI or using FDT (or both). The FEC Object Transmission Information that MUST be delivered to receivers is defined by the FEC Scheme. This section describes the delivery using FDT.

The FEC Object Transmission Information regarding a given TOI may be available from several sources. In this case, it is RECOMMENDED that the receiver of the file delivery session prioritize the sources in the following way (in the order of decreasing priority).

1. FEC Object Transmission Information that is available in EXT_FTI.

2. FEC Object Transmission Information that is available in the FDT.

The FDT delivers FEC Object Transmission Information for each file using an appropriate attribute within the "FDT-Instance" or the "File" element of the FDT structure.

* "Transfer-Length" carries the Transfer-Length Object Transmission Information element defined in [RFC5052].

* "FEC-OTI-FEC-Encoding-ID" carries the "FEC Encoding ID" Object Transmission Information element defined in [RFC5052], as carried in the Codepoint field of the ALC/LCT header.

* "FEC-OTI-FEC-Instance-ID" carries the "FEC Instance ID" Object Transmission Information element defined in [RFC5052] for Under-specified FEC Schemes.

* "FEC-OTI-Maximum-Source-Block-Length" carries the "Maximum Source Block Length" Object Transmission Information element defined in [RFC5052], if required by the FEC Scheme.

* "FEC-OTI-Encoding-Symbol-Length" carries the "Encoding Symbol Length" Object Transmission Information element defined in [RFC5052], if required by the FEC Scheme.

* "FEC-OTI-Max-Number-of-Encoding-Symbols" carries the "Maximum Number of Encoding Symbols" Object Transmission Information element defined in [RFC5052], if required by the FEC Scheme.

* "FEC-OTI-Scheme-specific-information" carries the "encoded scheme-specific FEC Object Transmission Information" as defined in [RFC5052], if required by the FEC Scheme.

In FLUTE, the FEC Encoding ID (8 bits) for a given TOI MUST be carried in the Codepoint field of the ALC/LCT header. When the FEC Object Transmission Information for this TOI is delivered through the FDT, then the associated "FEC-OTI-FEC-Encoding-ID" attribute and the Codepoint field of all packets for this TOI MUST be the same.

6. Describing file delivery sessions

To start receiving a file delivery session, the receiver needs to know transport parameters associated with the session. Interpreting these parameters and starting the reception therefore represents the entry point from which thereafter the receiver operation falls into the scope of this specification. According to [ID.ALC-revised], the transport parameters of an ALC/LCT session that the receiver needs to know are:

* The source IP address;

* The number of channels in the session;

* The destination IP address and port number for each channel in the session;

* The Transport Session Identifier (TSI) of the session;

* An indication that the session is a FLUTE session. The need to demultiplex objects upon reception is implicit in any use of FLUTE, and this fulfills the ALC requirement of an indication of whether or not a session carries packets for more than one object (all FLUTE sessions carry packets for more than one object).

Optionally, the following parameters MAY be associated with the session (Note, the list is not exhaustive):

* The start time and end time of the session;

* FEC Encoding ID and FEC Instance ID when the default FEC Encoding ID 0 is not used for the delivery of FDT;

* Content Encoding format if optional content encoding of FDT Instance is used, e.g., compression;

* Some information that tells receiver, in the first place, that the session contains files that are of interest;

* Definition and configuration of congestion control mechanism for the session ;

* Security parameters relevant for the session.

* FLUTE version number.

It is envisioned that these parameters would be described according to some session description syntax (such as SDP [RFC.SDP] or XML based) and held in a file which would be acquired by the receiver before the FLUTE session begins by means of some transport protocol (such as Session Announcement Protocol [RFC.SAP], email, HTTP [RFC.HTTP11], SIP [RFC.SIP], manual pre-configuration, etc.) However, the way in which the receiver discovers the above-mentioned parameters is out of scope of this document, as it is for LCT and ALC. In particular, this specification does not mandate or exclude any mechanism.

7. Security Considerations

7.1. Problem Statement

A content delivery system is potentially subject to attacks. Attacks may target:

* the network (to compromise the routing infrastructure, e.g., by creating congestion),

* the Content Delivery Protocol (CDP) (e.g., to compromise the normal behavior of FLUTE), or

* the content itself (e.g., to corrupt the files being transmitted).

These attacks can be launched either:

* against the data flow itself (e.g., by sending forged packets),

* against the session control parameters (e.g., by corrupting the session description, the FDT Instances, or the ALC/LCT control parameters) that are sent either in-band or out-of-band, or

* against some associated building blocks (e.g., the congestion control component).

In the following sections we provide more details on these possible attacks and sketch some possible counter-measures. We provide recommendations in Section 7.5.

7.2. Attacks against the data flow

Let us consider attacks against the data flow first. At least, the following types of attacks exist:

* attacks that are meant to give access to a confidential file (e.g., in case of a non-free content) and

* attacks that try to corrupt the file being transmitted (e.g., to inject malicious code within a file, or to prevent a receiver from using a file, which is a kind of Denial of Service, DoS).

7.2.1. Access to confidential files

Access control to the file being transmitted is typically provided by means of encryption. This encryption can be done over the whole file i.e. before applying FEC protection (e.g., by the content provider, before submitting the file to FLUTE), or be done on a packet per

packet basis (e.g., when IPsec/ESP is used [RFC.4303], see Section 7.5). If confidentiality is a concern, it is RECOMMENDED that one of these solutions be used.

7.2.2. File corruption

Protection against corruptions (e.g., if an attacker sends forged packets) is achieved by means of a content integrity verification/ sender authentication scheme. This service can be provided at the file level i.e. before applying content encoding and forward error correction encoding. In that case a receiver has no way to identify which symbol(s) is(are) corrupted if the file is detected as corrupted. This service can also be provided at the packet level i.e. after applying content encoding and forward error correction encoding, on a packet by packet basis. In this case, after removing all corrupted packets, the file may be in some cases recovered from the remaining correct packets.

Integrity protection applied at the file level has the advantage of lower overhead since only relatively few bits are added to provide the integrity protection compared to the file size. However it has the disadvantage that it cannot distinguish between correct packets and corrupt packets and therefore correct packets, which may form the majority of packets received, may be unusable. Integrity protection applied at the packet level has the advantage that it can distinguish between correct and corrupt packets at the cost of additional per packet overhead.

Several techniques can provide this source authentication/content integrity service:

* at the file level, the file MAY be digitally signed, for instance by using RSASSA-PKCS1-v1_5 [RFC.3447]. This signature enables a receiver to check the file integrity, once this latter has been fully decoded. Even if digital signatures are computationally expensive, this calculation occurs only once per file, which is usually acceptable;

* at the packet level, each packet can be digitally signed [RMT-SIMPLE-AUTH]. A major limitation is the high computational and transmission overheads that this solution requires. To avoid this problem, the signature may span a set of symbols (instead of a single one) in order to amortize the signature calculation, but if a single symbol is missing, the integrity of the whole set cannot be checked;

* at the packet level, a Group Message Authentication Code (MAC) [RFC.2104][RMT-SIMPLE-AUTH] scheme can be used, for instance by using HMAC-SHA-256 with a secret key shared by all the group members, senders and receivers. This technique creates a cryptographically secured digest of a packet that is sent along with the packet. The Group MAC scheme does not create prohibitive processing load nor transmission overhead, but it has a major limitation: it only provides a group authentication/integrity service since all group members share the same secret group key, which means that each member can send a forged packet. It is therefore restricted to situations where group members are fully trusted (or in association with another technique as a pre-check);

* at the packet level, TESLA [RFC.4082][MSEC-TESLA] is an attractive solution that is robust to losses, provides a true authentication/ integrity service, and does not create any prohibitive processing load or transmission overhead. Yet checking a packet requires a small delay (a second or more) after its reception;

* at the packet level, IPsec/ESP [RFC.4303] can be used to check the integrity and authenticate the sender of all the packets being exchanged in a session (see Section 7.5).

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

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

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

7.3. Attacks against the session control parameters and associated Building Blocks

Let us now consider attacks against the session control parameters and the associated building blocks. The attacker has at least the following opportunities to launch an attack:

* the attack can target the session description,

* the attack can target the FDT Instances,

* the attack can target the ALC/LCT parameters, carried within the LCT header or

* the attack can target the FLUTE associated building blocks, for instance the multiple rate congestion control protocol.

The consequences of these attacks are potentially serious, since they might compromise the behavior of content delivery system itself.

7.3.1. Attacks against the Session Description

A FLUTE receiver may potentially obtain an incorrect Session Description for the session. The consequence of this is that legitimate receivers with the wrong Session Description are unable to correctly receive the session content, or that receivers inadvertently try to receive at a much higher rate than they are capable of, thereby possibly disrupting other traffic in the network.

To avoid these problems, it is RECOMMENDED that measures be taken to prevent receivers from accepting incorrect Session Descriptions. One such measure is source authentication to ensure that receivers only accept legitimate Session Descriptions from authorized senders. How these measures are achieved is outside the scope of this document since this session description is usually carried out-of-band.

7.3.2. Attacks against the FDT Instances

Corrupting the FDT Instances is one way to create a Denial of Service attack. For example, the attacker changes the MD5 sum associated to a file. This possibly leads a receiver to reject the files received, no matter whether the files have been correctly received or not.

Corrupting the FDT Instances is also a way to make the reception process more costly than it should be. This can be achieved by changing the FEC Object Transmission Information when the FEC Object Transmission Information is included in the FDT Instance. For example, an attacker may corrupt the FDT Instance in such a way that Reed-Solomon over GF(2^^16) be used instead of GF(2^^8) with FEC Encoding ID 2. This may significantly increase the processing load while compromising FEC decoding.

It is therefore RECOMMENDED that measures be taken to guarantee the integrity and to check the sender’s identity of the FDT Instances. To that purpose, one of the counter-measures mentioned above

(Section 7.2.2) SHOULD be used. These measures will either be applied on a packet level, or globally over the whole FDT Instance object. Additionally, XML digital signatures [RFC.XML-DSIG] are a way to protect the FDT Instance by digitally signing it. When there is no packet level integrity verification scheme, it is RECOMMENDED to rely on XML digital signatures of the FDT Instances.

7.3.3. Attacks against the ALC/LCT parameters

By corrupting the ALC/LCT header (or header extensions) one can execute attacks on underlying ALC/LCT implementation. For example, sending forged ALC packets with the Close Session flag (A) set to one can lead the receiver to prematurely close the session. Similarly, sending forged ALC packets with the Close Object flag (B) set to one can lead the receiver to prematurely give up the reception of an object.

It is therefore RECOMMENDED that measures be taken to guarantee the integrity and to check the sender’s identity of the ALC packets received. To that purpose, one of the counter-measures mentioned above (Section 7.2.2) SHOULD be used.

7.3.4. Attacks against the associated Building Blocks

Let us first focus on the congestion control building block, that may be used in the ALC session. A receiver with an incorrect or corrupted implementation of the multiple rate congestion control building block may affect the health of the network in the path between the sender and the receiver. That may also affect the reception rates of other receivers who joined the session.

When congestion control building block is applied with FLUTE, it is therefore RECOMMENDED that receivers be required to identify themselves as legitimate before they receive the Session Description needed to join the session. How receivers identify themselves as legitimate is outside the scope of this document. If authenticating a receiver does not prevent this latter to launch an attack, it will enable the network operator to identify him and to take counter-measures.

When congestion control building block is applied with FLUTE, it is also RECOMMENDED that a packet level authentication scheme be used, as explained in Section 7.2.2. Some of them, like TESLA, only provide a delayed authentication service, whereas congestion control requires a rapid reaction. It is therefore RECOMMENDED [ID.ALC-revised] that a receiver using TESLA quickly reduces its subscription level when the receiver believes that a congestion did occur, even if the packet has not yet been authenticated. Therefore

TESLA will not prevent DoS attacks where an attacker makes the receiver believe that a congestion occurred. This is an issue for the receiver, but this will not compromise the network. Other authentication methods that do not feature this delayed authentication could be preferred, or a group MAC scheme could be used in parallel to TESLA to prevent attacks launched from outside of the group.

7.4. Other Security Considerations

Lastly, we note that the security considerations that apply to, and are described in, ALC [ID.ALC-revised], LCT [RFC.LCT] and FEC [RFC5052] also apply to FLUTE as FLUTE builds on those specifications. In addition, any security considerations that apply to any congestion control building block used in conjunction with FLUTE also apply to FLUTE.

7.5. Minimum Security Recommendations

We now introduce a mandatory to implement but not necessarily to use security configuration, in the sense of [RFC.3365]. Since FLUTE relies on ALC/LCT, it inherits the "baseline secure ALC operation" of [ID.ALC-revised]. More precisely, security is achieved by means of IPsec/ESP in transport mode. [RFC.4303] explains that ESP can be used to potentially provide confidentiality, data origin authentication, content integrity, anti-replay and (limited) traffic flow confidentiality. [ID.ALC-revised] specifies that the data origin authentication, content integrity and anti-replay services SHALL be supported, and that the confidentiality service is RECOMMENDED. If a short lived session MAY rely on manual keying, it is also RECOMMENDED that an automated key management scheme be used, especially in case of long lived sessions.

Therefore, the RECOMMENDED solution for FLUTE provides per-packet security, with data origin authentication, integrity verification and anti-replay. This is sufficient to prevent most of the in-band attacks listed above. If confidentiality is required, a per-packet encryption SHOULD also be used.

8. IANA Considerations

This specification contains five separate items for IANA Considerations:

1. Registration Request for XML Schema of FDT Instance.

2. Media-Type Registration Request for application/fdt+xml.

3. Content Encoding Algorithm Registration Request.

4. Registration of the EXT_FDT LCT Header Extension Type

5. Registration of the EXT_CENC LCT Header Extension Type

8.1. Registration Request for XML Schema of FDT Instance

Document [RFC.3688] defines an IANA maintained registry of XML documents used within IETF protocols. The following is the registration request for the FDT XML schema.

Registrant Contact: Toni Paila (toni.paila (at) nokia.com)

XML: The XML Schema specified in Section 3.4.2

8.2. Media-Type Registration Request for application/fdt+xml

This section provides the registration request, as per [RFC.MIME4a], [RFC.MIME4b] and [RFC.XML-Media-Types], to be submitted to IANA following IESG approval.

Type name: application

Subtype name: fdt+xml

Required parameters: none

Optional parameters: none

Encoding considerations: The fdt+xml type consists of UTF-8 ASCII characters [RFC.UTF8] and must be well-formed XML.

Additional content and transfer encodings may be used with fdt+xml files, with the appropriate encoding for any specific file being entirely dependent upon the deployed application.

Restrictions on usage: Only for usage with FDT Instances which are valid according to the XML schema of section 3.4.2.

Security considerations: fdt+xml data is passive, and does not generally represent a unique or new security threat. However, there is some risk in sharing any kind of data, in that unintentional information may be exposed, and that risk applies to fdt+xml data as

well.

Interoperability considerations: None

Published specification: The present document including section 3.4.2. The specified FDT Instance functions as an actual media format of use to the general Internet community and thus media type registration under the Standards Tree is appropriate to maximize interoperability.

Applications which use this media type: Not restricted to any particular application

Additional information:

Magic number(s): none
File extension(s): An FDT Instance may use the extension ".fdt" but this is not required.
Macintosh File Type Code(s): none

Person and email address to contact for further information: Toni Paila (toni.paila (at) nokia.com)

Intended usage: Common

Author/Change controller: IETF

8.3. Content Encoding Algorithm Registration Request

Values of Content Encoding Algorithms are subject to IANA registration. The value of Content Encoding Algorithm is a numeric non-negative index. In this document, the range of values for Content Encoding Algorithms is 0 to 255. This specification already assigns the values 0, 1, 2 and 3 as described in section 3.4.3.

8.3.1. Explicit IANA Assignment Guidelines

This document defines a name-space called "Content Encoding Algorithms".

IANA has established and manages the new registry for the "FLUTE Content Encoding Algorithm" name-space. The values that can be assigned within this name-space are numeric indexes in the range [0, 255], boundaries included. Assignment requests are granted on a "Specification Required" basis as defined in RFC 2434 [RFC.Guidelines-Iana-Section]. Note that the values 0, 1, 2 and 3 of this registry are already assigned by this document as described in section 3.4.3.

8.4. Registration of EXT_FDT LCT Header Extension Type

This document registers value 192 for the EXT_FDT LCT Header Extension defined in Section 3.4.1.

8.5. Registration of EXT_CENC LCT Header Extension Type

This document registers value 193 for the EXT_CENC LCT Header Extension defined in Section 3.4.3.

9. Acknowledgements

The following persons have contributed to this specification: Brian Adamson, Mark Handley, Esa Jalonen, Roger Kermode, Juha-Pekka Luoma, Topi Pohjolainen, Lorenzo Vicisano, and Mark Watson. The authors would like to thank all the contributors for their valuable work in reviewing and providing feedback regarding this specification.

10. Contributors

Jani Peltotalo
Tampere University of Technology
P.O. Box 553 (Korkeakoulunkatu 1)
Tampere FIN-33101
Finland
Email: jani.peltotalo (at) tut.fi

Sami Peltotalo
Tampere University of Technology
P.O. Box 553 (Korkeakoulunkatu 1)
Tampere FIN-33101
Finland
Email: sami.peltotalo (at) tut.fi

Magnus Westerlund
Ericsson Research
Ericsson AB
SE-164 80 Stockholm
Sweden
EMail: magnus.westerlund (at) ericsson.com

Thorsten Lohmar
Ericsson Research (EDD)
Ericsson Allee 1
52134 Herzogenrath, Germany
EMail: thorsten.lohmar (at) ericsson.com

11. Change Log

11.1. RFC3926 to draft-ietf-rmt-flute-revised-12

Incremented FLUTE protocol version from 1 to 2, due to IESG concerns about backwards compatibility.

Updated dependencies to other RFCs to revised versions, e.g., changed ALC reference from RFC 3450 to RFC 5775, changed LCT reference from RFC 3451 to RFC 5651, etc.

Two additional items are added in the IANA considerations section, specifically the registration of two values in the LCT Header Extension Types registry (192 for EXT_FDT and 193 for EXT_CENC).

Added clarification for the use of FLUTE for unicast communications in Section 1.1.4.

Clarified how to reliably deliver the FDT in Section 3.3 and the possibility of using an out-of-band delivery of FDT information.

Clarified how to address FDT Instance expiration time wraparound with the notion of "epoch" of NTPv4 in Section 3.3.

Clarified what should be considered as erroneous situations in Section 3.4.1 (definition of FDT Instance ID). In particular a receiver MUST be ready to handle FDT Instance ID wraparounds and missing FDT Instances.

Updated the security section to define IPsec/ESP as a mandatory to implement security solution in Section 7.5.

Removed the 'Statement of Intent' from the Section 1. The statement of intent was meant to clarify the "Experimental" status of RFC3926. It does not apply to this draft that is intended for "Standard Track" submission.

Added clarification on XML-DSIG in the end of Section 3.

Revised the use of word "complete" in the Section 3.2.

Clarified Figure 1 WRT "Encoding Symbol(s) for FDT Instance".

Clarified the FDT Instance ID wrap-around in the end of Section 3.4.1.

Clarification for "Complete FDT" in the Section 3.4.2.

Added semantics for the case two TOIs refer to same Content-Location. Now it is in line how 3GPP and DVB interpret the case.

In the Section 3.4.2 XML Schema of FDT instance is modified to various advices. For example, extension by element was missing but is now supported. Also namespace definition is changed to URN format.

Clarified FDT-schema extensibility in the end of Section 3.4.2.

The CENC value allocation is added in the end of Section 3.4.3.

Section 5 is modified so that EXT_FTI and the FEC issues are replaced by a reference to LCT specification. We count on revised LCT specification to specify the EXT_FTI.

Added a clarifying paragraph on the use of Codepoint in the very end of Section 5.

Reworked Section 8 - IANA Considerations. Now it contains three IANA registration requests:

* Registration Request for XML Schema of FDT Instance (urn:ietf:params:xml:schema:fdt)

* Media-Type Registration Request for application/fdt+xml

* Content Encoding Algorithm Registration Request (ietf:rmt:cenc)

Added Section 10 - Contributors.

Revised list of both Normative as well as Informative references.

Added a clarification that receiver should ignore reserved bits of Header Extension type 193 upon reception.

Minor changes to remove forward references (use before definition) or refer to forward reference sections.

Elaborate on just what kind of networks cannot support FLUTE congestion control (1.1.4)

In Section 3.2 revise "several" (meaning 3-n vs. "couple" = 2) to "multiple" (meaning 2-n)

Move Section 3.3 requirement to send FDT more reliably than files, to a bulleted RECOMMENDED requirement, making check-off easier for testers.

Sharpen Section 3.3 definition that future FDT file instances can "augment" (meaning enhance) rather than "complement" (sometimes meaning negate, which is not allowed) the file parameters.

Elaborate in Section 3.3 and Section 4 that FEC Encoding ID = 0 is Compact No-code FEC, so that the reader doesn't have to search other RFCs to understand these protocol constants used by FLUTE.

Require in Section 3.3 that FLUTE receivers SHALL NOT attempt to decode FDTs if they do not understand the FEC Encoding ID

Remove restriction of Section 3.3 in bullet #4 that TOI=0 for the FDT, to be consistent with Appendix, bullet 6, and elsewhere. An FDT is signaled by an FDT Instance ID, NOT only by TOI = 0.

Standardize on the term "expiration time" and avoid using the redundant but possibly confusing term "expiry time".

To interwork with experimental flute, stipulate in Section 3.1 that only 1 instantiation of all 3 protocols FLUTE, ALC, and LCT, can be associated with a session (source IP-Address, TSI) and mention in Section 6 that you may (optionally) derive the FLUTE version from the file delivery session description.

Use a software writing tool to lower reading grade level and simplify Section 3.1.

12. References

12.1. Normative references

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

[ID.ALC-revised] Luby, M., Watson, M., and L. Vicisano, "Asynchronous Layered Coding (ALC) Protocol Instantiation", RFC 5775, April 2010.

[RFC.LCT] Luby, M., Watson, M., and L. Vicisano, "Layered Coding Transport (LCT) Building Block", RFC 5651, October 2009.

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

[RFC.FECSchemes]

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

[RFC.NTP] Mills, D., "Network Time Protocol (Version 3), Specification, Implementation and Analysis", RFC 1305, March 1992.

[RFC.HTTP11] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[XML-Schema-Part-1] Thompson, H., Beech, D., Maloney, M., and N. Mendelsohn, "XML Schema Part 1: Structures", W3C Recommendation, May 2001.

[XML-Schema-Part-2] Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes", W3C Recommendation, May 2001.

[RFC.XML-Media-Types] Murata, M., St.Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[RFC.UTF8] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 3629, November 2003.

[RFC.Guidelines-Iana-Section] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, May 2008.

[RFC.ZLIB] Deutsch, P. and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, May 1996.

[RFC.DEFLATE] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, May 1996.

[RFC.GZIP] Deutsch, P., "GZIP file format specification version 4.3", RFC 1952, May 1996.

[RFC.3738] Luby, M. and V. Goyal, "Wave and Equation Based Rate Control (WEBRC) Building Block", RFC 3738, April 2004.

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

[RFC5651] Luby, M., Watson, M., and L. Vicisano, "Layered Coding Transport (LCT) Building Block", RFC 5651, October 2009.

12.2. Informative references

[RFC3926] Paila, T., Luby, M., Lehtonen, R., Roca, V., and R. Walsh, "FLUTE - File Delivery over Unidirectional Transport", RFC 3926, October 2004.

[RFC.SAP] Handley, M., Perkins, C., and E. Whelan, "Session Announcement Protocol", RFC 2974, October 2000.

[RFC.SDP] Handley, M., Jacobson, V., and C. Perkins, "Session Description Protocol", RFC 4566, July 2006.

[RFC.ASM] Deering, S., "Host Extensions for IP Multicasting", RFC 1112, STD 5, August 1989.

[PAPER.SSM]
Holbrook, H., "A Channel Model for Multicast, Ph.D. Dissertation, Stanford University, Department of Computer Science, Stanford, California", August 2001.

[NTPv4] Kasch, W., Mills, D., and J. Burbank, "Network Time Protocol Version 4 Protocol And Algorithms Specification", draft-ietf-ntp-ntpv4-proto-13 (work in progress) (work in progress), October 2009.

[RFC.3365]
Schiller, J., "Strong Security Requirements for Internet Engineering Task Force Standard Protocols", BCP 61, RFC 3365, August 2002.

[RFC.SMIME]
Ramsdell, B., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification", RFC 3851, July 2004.

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

[RFC.MIME4a]

Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", RFC 4288, December 2005.

[RFC.MIME4b]
Freed, N. and J. Klensin, "Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures", RFC 4289, December 2005.

[RFC.SIP] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: session initiation protocol", RFC 3261, June 2002.

[RFC.3688]
Mealling, M., "The IETF XML Registry", RFC 3688, January 2004.

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

[RFC.2104]
Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[RFC.4082]
Perrig, A., Canetti, R., Tygar, J D., and B. Briscoe, "Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction", RFC 4082, June 2005.

[MSEC-TESLA]
Roca, V., Francillon, A., and S. Faurite, "Use of TESLA in the ALC and NORM Protocols", draft-ietf-msec-tesla-for-alc-norm-10.txt (work in progress), October 2009.

[RMT-SIMPLE-AUTH]
Roca, V., "Simple Authentication Schemes for the ALC and NORM Protocols", draft-ietf-rmt-simple-auth-for-alc-norm-02.txt (work in progress), October 2009.

Appendix A. Receiver operation (informative)

This section gives an example how the receiver of the file delivery session may operate. Instead of a detailed state-by-state specification the following should be interpreted as a rough sequence of an envisioned file delivery receiver.

1. The receiver obtains the description of the file delivery session identified by the pair: (source IP address, Transport Session Identifier). The receiver also obtains the destination IP addresses and respective ports associated with the file delivery session.

2. The receiver joins the channels in order to receive packets associated with the file delivery session. The receiver may schedule this join operation utilizing the timing information contained in a possible description of the file delivery session.

3. The receiver receives ALC/LCT packets associated with the file delivery session. The receiver checks that the packets match the declared Transport Session Identifier. If not, packets are silently discarded.

4. While receiving, the receiver demultiplexes packets based on their TOI and stores the relevant packet information in an appropriate area for recovery of the corresponding file. Multiple files can be reconstructed concurrently.

5. Receiver recovers an object. An object can be recovered when an appropriate set of packets containing Encoding Symbols for the transmission object have been received. An appropriate set of packets is dependent on the properties of the FEC Encoding ID and FEC Instance ID, and on other information contained in the FEC Object Transmission Information.

6. Objects with TOI = 0 are reserved for FDT Instances. All FDT Instances are signaled by including an EXT_FDT header extension in the LCT header. The EXT_FDT header contains an FDT Instance ID (i.e. an FDT version number.) If the object has an FDT Instance ID 'N', the receiver parses the payload of the instance 'N' of FDT and updates its FDT database accordingly.

7. If the object recovered is not an FDT Instance but a file, the receiver looks up its FDT database to get the properties described in the database, and assigns the file the given properties. The receiver also checks that the received content length matches with the description in the database. Optionally, if MD5 checksum has been used, the receiver checks that the

calculated MD5 matches the description in the FDT database.

8. The actions the receiver takes with imperfectly received files (missing data, mismatching digestive, etc.) is outside the scope of this specification. When a file is recovered before the associated file description entry is available, a possible behavior is to wait until an FDT Instance is received that includes the missing properties.

9. If the file delivery session end time has not been reached go back to 3. Otherwise end.

Appendix B. Example of FDT Instance (informative)

```
<?xml version="1.0" encoding="UTF-8"?>
<FDT-Instance xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ietf:params:xml:ns:fdt
                      ietf-flute-fdt.xsd"
  Expires="2890842807">
  <File
    Content-Location="http://www.example.com/menu/tracklist.html"
    TOI="1"
    Content-Type="text/html"/>
  <File
    Content-Location="http://www.example.com/tracks/track1.mp3"
    TOI="2"
    Content-Length="6100"
    Content-Type="audio/mp3"
    Content-Encoding="gzip"
    Content-MD5="+VP5IrWploFkZWc11iLDdA=="
    Some-Private-Extension-Tag="abc123"/>
</FDT-Instance>
```

Authors' Addresses

Toni Paila
Nokia
Itamerenkatu 11-13
Helsinki 00180
Finland

Email: toni.paila@nokia.com

Rod Walsh
Nokia
Visiokatu 1
Tampere FIN-33720
Finland

Email: rod.walsh@nokia.com

Michael Luby
Qualcomm, Inc.
3165 Kifer Rd.
Santa Clara, CA 95051
US

Email: luby@qualcomm.com

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

Email: vincent.roca@inria.fr

Rami Lehtonen
TeliaSonera
Hatanpaan valtatie 18
Tampere FIN-33100
Finland

Email: rami.lehtonen@teliasonera.com

RMT
Internet-Draft
Expires: August 20, 2011

R. Walsh
I. Curcio
Nokia Research Center
J. Peltotalo
S. Peltotalo
Tampere University of Technology
H. Mehta

February 16, 2011

SDP Descriptors for FLUTE
draft-ietf-rmt-flute-sdp-00

Abstract

This document specifies the use of SDP to describe the parameters required to begin, join, receive data from, and/or end FLUTE sessions. It also provides a Composite Session SDP media grouping semantic for grouping media streams into protocol-specific sessions, such as multiple-channel FLUTE sessions.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 20, 2011.

Table of Contents

1. Introduction

The Session Description Protocol (SDP) [RFC4566] provides a general-purpose format for describing multimedia sessions in announcements or invitations. SDP uses an entirely textual data format (the US-ASCII subset of UTF-8 [RFC3629]) to maximize portability among transports. SDP does not define a protocol, but only the syntax to describe a multimedia session with sufficient information to participate in that session. Session descriptions may be sent using arbitrary existing application protocols for transport (e.g. FLUTE [I-D.ietf-rmt-flute-revised], SAP [RFC2974], SIP [RFC3261], RTSP [RFC2326], HTTP [RFC2616], email etc.).

SDP defines two protocol identifiers that represent unreliable connectionless protocols. These are RTP/AVP and UDP. These are appropriate choices for multimedia streams. [RFC4145] defines protocol identifiers for connection-oriented reliable transports: TCP and TCP/TLS.

This document defines a new protocol identifier for File Delivery over Unidirectional Transport (FLUTE) protocol [I-D.ietf-rmt-flute-revised] and other required SDP attributes for initiating a FLUTE session. The formal ABNF syntax [RFC5234] is used for the attributes. This SDP syntax is independent of Any Source Multicast (ASM) or Source Specific Multicast (SSM) is used to route the media.

Note, this document may also be used to describe sessions of the experimental FLUTE specification [RFC3926].

2. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119 [RFC2119].

3. FLUTE Descriptors

The FLUTE specification [I-D.ietf-rmt-flute-revised] describes the optional and required parameters for a FLUTE session. This document specifies the SDP parameters for FLUTE sessions that can be used for the discovery of FLUTE download and/or service announcement sessions. Listed below are the required and optional SDP parameters for FLUTE sessions (the parameters introduced, or made mandatory, by this specification but not inherited from the FLUTE specification are marked with an asterisk "*").

The required parameters are:

- The source IP address;
- The number of channels in the session;
- The destination IP address and port number for each channel in the session;
- The Transport Session Identifier (TSI) of the session;
- An indication that the session is a FLUTE session;
* The start time and end time of the session.

The optional parameters are:

- FEC Object Transmission Information;
- Some information that tells receiver in the first place, that the session contains files that are of interest;
- Definition and configuration of congestion control mechanism for the session [Editorial note: under consideration];
- Security parameters relevant for the session [Editorial note: under consideration];
* Bandwidth specification.

(Note, best practise to provide parameters for FLUTE’s optional content encoding of FDT Instances is in-band within FLUTE sessions and is therefore not specified using SDP.)

(Note, the out-of-band FEC Object Transmission Information useful for FLUTE sessions is limited to capabilities describing FEC Encoding ID(s) and FEC Instance ID(s) as FLUTE provides header fields for

machine configuration for object reception. This specification also provides a "fec-oti-extension", as an informative appendix, so that the same SDP syntax can be used to describe sessions using protocols other than FLUTE that do not have an in-band mechanism for FEC machine configuration.)

The semantics of a FLUTE session within an SDP description differ slightly from that of the well-establish RTP session descriptions. A FLUTE session includes one or more FLUTE channels which are each a distinct media stream. (Note, the SDP specification [RFC4566] use of the term "media stream" is semantically equivalent to the FLUTE specification use of the term "channel".) Generally, each RTP media is recognised as a distinct RTP media session. Hence, to preserve harmony with RTP media sessions within SDP descriptions, the optional Composite Session mechanism is specified in this document, using the SDP Grouping Framework [RFC5888].

The description of these parameters in SDP is presented in the following sections.

3.1. FLUTE Protocol Identifier

The following is the ABNF syntax for an "m=" line, as specified by RFC4566 [RFC4566]:

```
media-field = "m=" media SP port ["/" integer] SP
  proto 1*(SP fmt) CRLF
```

We define a new value for the "proto" sub-field: FLUTE/UDP. The FLUTE/UDP protocol identifier specifies that the session being described will use the FLUTE [I-D.ietf-rmt-flute-revised] protocol on top of a UDP connection.

As described below, more than one FLUTE session may be described by a single SDP using the Composite Session mechanism.

The fmt (format) list may be ignored for FLUTE. The fmt list of FLUTE "m=" lines MAY contain a single "*" character to indicate that miscellaneous and unspecified MIME types (file formats) are contained in the FLUTE session. Use of any other values (MIME types) in a FLUTE fmt list is out of scope of this specification. "0" is known to be used in the fmt list to represent the same as "*", in a non-standard way, and so implementers may take this into account. An example of FLUTE/UDP protocol identifier is shown in Section 4.

FLUTE is a general file delivery protocol and so it is not considered necessary to identify a list of media types per FLUTE session or channel in the session description.

3.2. Composite Session Semantics

The Composite Session mechanism enables the grouping of media lines in to distinct sessions. The complete Composite Session semantics are protocol-specific - as determined by the protocol id of the grouped media lines. This section defines the Composite Session semantic generically and protocol id independently. Subsection 3.2.1. defines the FLUTE/UDP protocol identifier specific semantic.

This mechanism is useful where multiple FLUTE sessions are described as part of a larger service or application, and so where maintaining and delivering session descriptions together (with a shared delivery fate) is good practice. It may also improve bandwidth efficiency by eliminating repetition of redundant descriptors that would be necessary with multiple discrete SDP instances.

The Composite Session mechanism inherits the "group" and "mid" attributes from the SDP grouping framework [RFC5888] and introduces the "CS" (Composite Session) token as a "semantics-extension".

When the Composite Session mechanism is used: the SDP grouping framework [RFC5888] MUST be used (and requirements from that are inherited); and the "CS" token MUST be used with the "group" attribute to indicate a Composite Session grouping. The SDP grouping framework declares groups at session-level and labels media (with the "mid" attribute) at media-level. Hence, all media identified by their "mid" values by an "a=group:CS" line belong to the same Composite Session group and inherit the grouping specified for that value at session-level.

The first (leftmost) mid value declared for a Composite Session group is the Primary Media. Just as session-level attributes are inherited to media-level declarations (unless specifically overwritten by an additional media-level attribute), Primary Media attributes SHALL be inherited to all media of a particular Composite Session group and these MAY be overwritten where an attribute syntax allows.

3.2.1. Composite Session Semantics for FLUTE Sessions

When a complete SDP description specifies only one FLUTE session, using the Composite Session mechanism is OPTIONAL. When a complete SDP description specifies more than one FLUTE session, using the Composite Session mechanism is REQUIRED.

The Composite Session provides an unambiguous way to define multiple FLUTE sessions as distinct from multiple the media-sessions semantics of RTP. It is useful for describing more than one FLUTE session in an SDP instance and so its use and support are OPTIONAL. For SDP

instances which describe multiple FLUTE sessions, the Composite Session semantics MUST be used. Whenever an SDP describes just one FLUTE session with more than a single media stream of FLUTE protocol identifier (i.e. a FLUTE channel), use of Composite Session semantics is RECOMMENDED.

To support simple applications, as well as ensure harmony with FLUTE SDP standards outside of the IETF [3GPP.26.346], when the Composite Session mechanism is not used for media of the UDP/FLUTE protocol, exactly one FLUTE session is specified within the SDP description and all UDP/FLUTE media that SDP description belong to the same FLUTE session (this is known as the Restricted Behaviour).

The Composite Session mechanism SHOULD NOT be used where the target clients are expected to include simpler FLUTE SDP parsers, such as in 3GPP MBMS [3GPP.26.346]. In this Restricted Behaviour only UDP/FLUTE media SHALL be described.

A partial example of using the Composite Session mechanism for FLUTE is shown below.

```
<other session-level attributes>
a=group:CS 1 2
a=group:CS 3
m=application 12345 FLUTE/UDP *
a=mid:1
<other media-level attributes>
m=application 12346 FLUTE/UDP *
a=mid:2
<other media-level attributes>
m=application 56789 FLUTE/UDP *
a=mid:3
<other media-level attributes>
```

The example shows two groups with the 1st and 3rd media ("m=") lines (mid values 1 and 3) being the Primary Media for each group respectively. In the example, the media with mid value "2" inherits attributes of the media with mid value "1".) Each of these groups identifies a separate FLUTE Session. Several of the attributes subsequently specified in this document use this feature of Primary Media inheritance to all media of a Composite Session.

3.2.2. Composite Session Semantics for Protocols other than FLUTE

The Composite Session mechanism solves the problem of describing multiple FLUTE sessions in a single SDP instance. However, this does not place any restrictions on the use of the Composite Session mechanism with transport protocols other than FLUTE/UDP, nor on

whether a complete SDP would include media of other transport protocols too. Specification of semantics beyond the use of FLUTE sessions is outside the scope of this document.

3.3. Source IP Address

The Asynchronous Layered Coding (ALC) [RFC5775] and the Layered Coding Transport (LCT) [RFC5651] specifications require that all the channels of a single ALC/LCT session are from the same source IP address. Hence, there MUST be exactly one source IP address per FLUTE session, and therefore one source IP address per each description of a FLUTE session description. Restricted behaviour is one source IP address per each complete SDP. Where multiple FLUTE sessions are described within one SDP instance this means one source IP address per each Composite Session.

The source IP address MUST be defined according to the source-filter attribute ("a=source-filter") [RFC4570], with the following exceptions:

- The source-filter attribute MUST be included in any SDP describing FLUTE per FLUTE session described.
- The number of source-filter attributes in any SDP describing FLUTE must be exactly equal to the number of FLUTE sessions described in that SDP.
- In the restricted behaviour of only one FLUTE session description in an SDP and no use of the Composite Session mechanism: The source-filter attribute MUST be in the session part of the session description and MUST NOT be given per media. Note, the requirement that there must not be more than a single source-filter attribute in the session part is inherited from the SDP Source Filter specification [RFC4570].
- Where the Composite Session mechanism is used: The source-filter attribute MUST be in the media part of Primary Media of each distinct FLUTE session, and MUST NOT be given in other media declarations but these, nor in the session-level part of the SDP.
- Exactly one source address is specified by any instance of this attribute. Exactly one source address MUST be given in an inclusive-mode "src-list". Exclusive-mode MUST NOT be used.
- The "*" value MUST be used for the "dest-address" sub-field, even when the FLUTE session employs only a single channel (e.g. a multicast group).

An example of the use of this attribute is:

a=source-filter: incl IN IP6 * 2001:0DB8:1:2:240:96FF:FE25:8EC9

This example uses the source-filter attribute to describe an IPv6 source address.

3.4. Transport Session Identifier

The combination of the TSI and the source IP address identifies a FLUTE session. Each TSI MUST uniquely identify a FLUTE session for a given source IP address during the time that the session is active and also for a large time before and after the active session time. This requirement is inherited from LCT [RFC5651]. Note, the SDP specification [RFC4566] advises that sessions expire 30 minutes after the session-end time given in the t-field. This should be considered an absolute minimum interpretation of a "large time". TSI reuse is NOT RECOMMENDED whenever possible (thus, making "large time" unbounded regarding TSI reuse).

The TSI MUST be described by the "flute-tsi" attribute.

There MUST be exactly one occurrence of the "flute-tsi" attribute per FLUTE session description of a SDP description.

- The number of "flute-tsi" attributes in any SDP describing FLUTE must be exactly equal to the number of FLUTE sessions described in that SDP.

- In the restricted behaviour of only one FLUTE session description in an SDP and no use of the Composite Session mechanism: The "flute-tsi" attribute MUST be in the session part of the session description and MUST NOT be given per media. A "flute-tsi" attribute in the session-part SHALL be used to identify restricted behaviour.

- Where the Composite Session mechanism is used: The "flute-tsi" attribute MUST be in the media part of Primary Media of each distinct FLUTE session, and MUST NOT be given in other media declarations but these, and MUST NOT be given in the session-level part of the SDP.

The syntax for the attribute in ABNF is given below:

```
flute-tsi-line = "a=flute-tsi:" tsi CRLF
tsi = 1*DIGIT
```

Note, the range of values a TSI can adopt depends on the bitlength of

the TSI for a session as defined by RFC5651 [RFC5651].

3.5. Session Timing Parameters

The SDP timing field "t=" [RFC4566] MUST be used to indicate the FLUTE session start and end times. This value applies to all FLUTE and transport sessions defined in a single SDP instance and, thus, FLUTE sessions of different timing values need to be declared in different SDP instances.

Note, implementors may assume reasonable clock synchronisation between SDP description, receiver wall clock and sender wall clock (within 60 seconds) unless specified otherwise for a specific deployment. The method to achieve this is beyond the scope of the current specifications, but may use well known and mature approaches such as SNTP [RFC5905].

3.6. Channelisation Descriptors

This section specifies the description of the channel(s) used within a FLUTE session. The required parameters for channelisation description are:

- Number of channels
- Destination IP address and port number for channels

3.6.1. Number of Channels

The FLUTE specification allows the use of multiple channels (e.g. multicast groups) to transport the files of a single FLUTE session. This is referred to as FLUTE session channelisation in this document. A FLUTE channel is equivalent to an ALC/LCT channel. An ALC/LCT channel is defined by the combination of a sender and an address associated with the channel by the sender. Details of each channel are defined by SDP media-level information also described in this document. The number of channels is calculated by summing the number of unique destination IP address and port number pairs for a certain FLUTE session (assignment of media to FLUTE sessions is done with presence of absence of the Composite Session grouping).

The OPTIONAL "flute-ch" attribute describes the number of channels used by the source to transmit the FLUTE session. When present, it is used to validate the channel number calculation based on the number of destination address/port pairs, and it is expected to be used where SDP proxies and other automatic and manual editing that introduces errors would cause bad failure conditions at the client.

When the "flute-ch" attribute is used:

- The number of "flute-ch" attributes in any SDP describing FLUTE MUST be exactly equal to the number of FLUTE sessions described in that SDP. A client SHOULD discard all of an SDP instance if this condition is not met. Alternative behaviour, such as retries at delivery, error reporting and partial use of SDP instances known to include errors, are beyond the scope of this document.

- In the restricted behaviour of only one FLUTE session description in an SDP and no use of the Composite Session mechanism: The "flute-ch" attribute MUST be in the session part of the session description and MUST NOT be given per media.

- Where the Composite Session mechanism is used: The "flute-ch" attribute MUST be in the media part of Primary Media of each distinct FLUTE session, and MUST NOT be given in other media declarations but these, nor in the session-level part of the SDP.

The syntax for the attribute in ABNF is given below:

```
flute-channel-line = "a=flute-ch:" ch CRLF
ch = integer
;integer is as defined in [RFC4566], and its value is the number of
;channels used by the source to transmit data in a FLUTE session.
```

3.6.2. Destination IP Address and Port Number for Channels

SDP media-level information describes one or more channels. The channel parameters MUST be given per channel and are:

- Destination IP address

- Destination port number

The destination IP address MUST be defined according to the connection data field ("c=") of SDP [RFC4566]. The destination port number MUST be defined according to the "port" sub-field of the media description field ("m=") of SDP [RFC4566].

A "c=" line can describe multiple addresses by using "number of addresses" sub-field, and also an "m=" line can describe multiple ports by using "number of ports" sub-field. So multiple channels can be described by using one "c=" line and one "m=" line (called "slash notation").

When more than one channel is used in a multicast FLUTE session, it is RECOMMENDED that the channels are differentiated based on

destination IP address, and channels are not differentiated based on destination port (although those ports could be same or different for each of the channels). Whenever destination port number is used to differentiate between FLUTE channels, the same destination IP address MUST be used for all channels in that FLUTE session. Note, when more than one channel is used in a unicast FLUTE session, the channels have to be differentiated based on destination ports, as only one destination IP address could be used.

In the case (always with a unicast session) where the same destination IP address is used for all the channels of the session and only the destination port number differentiates channels, the destination IP address MAY be given by the connection data field at session-level for all channels (if so, the connection data field MUST NOT be used at media-level).

In the case where each channel has a different destination IP address, the destination IP addresses MUST be given at media-level, i.e. following an "m=" line.

The sequence of multiple channels MUST be determined by the order in which their media descriptions are defined in the session description (i.e. the first media description gives the first channel in the sequence). This applies individually to each FLUTE session of an SDP whether one or more FLUTE sessions are described. In the case of the slash notation usage for specifying multiple destination addresses or ports, the order of the channel sequence MUST be lowest value first and highest last. Note, slash notation for both destination IP address and port would be incompatible with requirement to not use both destination IP address and port to differentiate channels in a FLUTE session and thus slash notation for both destination IP address and port is not allowed for a single FLUTE session - i.e. for a single composite session (when the SDP describes multiple FLUTE sessions) or for a single SDP (when only one FLUTE session is described).

Also we need to indicate the presence of a FLUTE session on a certain channel. This is done by using the "m=" line in the SDP description as shown in the following example:

```
m=application 12345 FLUTE/UDP *
c=IN IP6 FF33::8000:1
```

In the above SDP attributes, the "m=" line indicates the media used and the "c=" line together with "m=" line's "port" sub-field indicates the corresponding channel's address and port respectively. Thus, in the above example, the media is transported on a channel that uses FLUTE over UDP. Further, the "c=" line indicates the

channel's address, which, in this case, is an IPv6 address, and "m=" line indicates the channel's port (12345).

Note, the value of the destination IP address can indicate whether a multicast media belongs to an ASM or a SSM group as described by [RFC4607].

3.7. FEC Object Transmission Information

An SDP description for a FLUTE session MAY include FEC Object Transmission Information (FEC-OTI) [RFC5052]. FEC parameters can be placed either at session-level or at media-level, although it is RECOMMENDED to place them at session-level. Furthermore, if FEC parameters are placed at media level (contrary to the recommendation) and the Composite Session mechanism is used, they SHOULD only be placed in the Primary Media for any FLUTE session description. If FEC declarations on both session and media level use the same reference number (fec-ref) then the media level declaration takes precedence for that media component. FEC parameters include:

- FEC Encoding ID
- FEC Instance ID (for FEC Encoding IDs 128-255)

Where FEC-OTI is given, FEC parameters MUST be described in a "FEC-declaration" attribute. Multiple instances of this attribute MAY exist both at session-level and media-level. If an instance exists at session-level (or in a Primary Media), a reference to it MAY be used at media-level, and an attribute "FEC" MUST be defined for this purpose.

The syntax for the attributes in ABNF is given below:

```
fec-declaration-line = "a=FEC-declaration:" fec-ref SP
  fec-enc-id [";" SP fec-inst-id] CRLF
fec-ref = 1*3DIGIT
;value is the SDP-internal identifier for FEC-declaration

fec-enc-id = "encoding-id=" enc-id
enc-id = 1*DIGIT
;value is the FEC Encoding ID used

fec-inst-id = "instance-id=" inst-id
inst-id = 1*DIGIT
;value is the FEC Instance ID used

fec-line = "a=FEC:" fec-ref CRLF
```

Examples of FEC-OTI are shown in Section 4.

The FEC parameters are for capabilities description for the session. These parameters do not mandate a certain machine configuration but instead indicate which capabilities might be needed for successful reception of objects from specific channels. (Note, any "FDT-like" fuller description of files in the session could give the FEC parameters per file). FLUTE's FDT syntax (and codepoint header field usage) allows complete specification of these FEC parameters in-band with FLUTE (per file). Thus machine configuration can be performed using FLUTE alone.

More complete list of notes on the design logic for the FEC-OTI descriptors is provided as an appendix to this document.

The identification and description of any congestion control (CC) instance related to layered media (multiple FLUTE channels) is orthogonal to the FEC declarations and other aspects of this document. Hence, CC descriptions are not in scope of this document.

3.8. Content Description Pointer

The syntax of the information that tells receiver, in the first place, that the session contains files that are of interest is out of scope of this document. However, the SDP MAY include a content description pointer at the session-level and/or media-level (including Primary Media of Composite Sessions) to enable efficient linkage to such information.

The content description pointer attribute describes to the receiver(s) the URI where the content description is stored. The content description pointer MUST be defined according to the "content-desc" attribute.

The syntax for the attribute in ABNF is given below:

```
content-desc-line = "a=content-desc:" URI-reference CRLF
;URI-reference is as defined in [RFC3986].
```

An example of content description pointer is shown in Section 4.

3.9. Bandwidth Specification

The specification of bandwidth (data rate) is OPTIONAL and where included in the SDP it SHALL adhere to the following rules.

The maximum bit-rate required by a particular FLUTE media line (one or more FLUTE channels, depending on the usage or IP address and port

ranges) MAY be specified. In this case it is RECOMMENDED to use the TIAS bandwidth modifier [RFC3890] on media-level, although the AS bandwidth modifier [RFC4566] MAY be used on media-level.

The session bit-rate MAY also be specified. In this case it is RECOMMENDED to use the TIAS bandwidth modifier and the "a=maxprate" attribute for the session, and again AS is optional but not recommended.

TIAS is generally preferred as it allows the calculation of the bit-rate in environments with translation of IP version or transport protocol, where as AS does not and thus adds significant complexity in such environments.

Any Transport Independent (TIAS) bandwidth SHALL be the largest sum of the sizes of all FLUTE/UDP packets transmitted during any one second long period of the FLUTE session, depending on which level it is being used, expressed as kilobits. The size of the packet SHALL include all FLUTE, ALC, LCT and any extensions headers and payload. IP and UDP headers are excluded from the TIAS bit-rate calculation. Any Application Specific (AS) bandwidth SHALL be the largest sum of the sizes of all FLUTE/UDP packets transmitted during any one second long period for the related media line(s), expressed as kilobits. The size of the packet SHALL be the complete packet, i.e. IP, UDP and FLUTE headers, and the data payload.

3.9.1. Bandwidth Specification for Composite Sessions

Where the multimedia session bit-rate is specified (at SDP session level) this applies to all media, irrespective of whether the Composite Session mechanism is used to describe multiple sessions (e.g. multiple FLUTE sessions). So if multiple Composite Sessions are described in a single SDP and SDP session-level bit-rate is described, this session-level bit-rate would not relate to any single Composite Session.

A normal TIAS or AS bit-rate declaration at the Primary Media level is to be interpreted as media-specific and not imply any inheritance to other media of the same Composite Session. It is RECOMMENDED that aggregate Composite Session bandwidth is calculated as the sum of all constituent media bit-rate declarations. Specification of a descriptor specifically for aggregate Composite Session bandwidth is beyond the scope of this document.

3.10. SDP Specific Parameters

SDP [RFC4566] also mandates three parameters ("v=", "o=" and "s=") that would be present in every FLUTE SDP description regardless of

their usefulness to the FLUTE session description.

4. SDP Syntax Examples

This section gives examples of the use of SDP attributes to describe a FLUTE session.

```
v=0
o=user123 2890844526 2890842807 IN IP6 2001:0DB8::112E:144A:1E24
s=File delivery session example
i=More information
t=2873397496 2873404696
a=source-filter: incl IN IP6 * 2001:0DB8:1:2:240:96FF:FE25:8EC9
a=flute-tsi:3
a=flute-ch:2
a=FEC-declaration:0 encoding-id=0
a=FEC-declaration:1 encoding-id=129; instance-id=0
a=content-desc:http://www.example.com/flute-sessions/session001
m=application 12345 FLUTE/UDP *
c=IN IP6 FF33::8000:1
a=FEC:0
m=application 12346 FLUTE/UDP *
c=IN IP6 FF33::8000:2
a=FEC:1
```

Figure 1: An SDP for FLUTE Session with Two Channels

Figure 1 shows an example SDP description for FLUTE session with two channels.

The attribute defined in the line "a=source-filter: incl IN IP6 * 2001:0DB8:1:2:240:96FF:FE25:8EC9" describes a source filter. In this example the source indicates that the receiver(s) should include the given IP address (2001:0DB8:1:2:240:96FF:FE25:8EC9) into the session. It should be noted that although other possibilities may be used, in this case only the incl and * attributes may be used in the above attribute.

The attribute defined in the line "a=flute-tsi:3" describes the Transport Session Identifier for the session. The pair made of the source IP address and the TSI together uniquely identifies a FLUTE session.

The source indicates in the above example that it will transmit data in the FLUTE session on two channels (a=flute-ch:2). The source then specifies the channels.

The "a=FEC-declaration" lines describes two different FEC schemes used in the FLUTE session.

The "a=content-desc" line describes the URI where the content description is stored.

The line "m=application 12345 FLUTE/UDP *" indicates the media used for the channel. In this example, there are two "m=" lines for the two channels described.

The destination addresses for the channels are given in the "c=" lines. These also show to the receiver(s) that the channels are two (maybe more in other cases) consecutive channels.

The "a=FEC" lines at media-level reference FEC declarations at session-level ("a=FEC-declaration").

```
v=0
o=user123 2890844526 2890842807 IN IP6 2001:0DB8::112E:144A:1E24
s=File delivery session example
i=More information
t=2873397496 2873404696
a=source-filter: incl IN IP6 * 2001:0DB8:1:2:240:96FF:FE25:8EC9
a=flute-tsi:2
a=flute-ch:1
m=application 12345 FLUTE/UDP *
c=IN IP6 FF33::8000:1
a=FEC-declaration:0 encoding-id=129; instance-id=0
```

Figure 2: An SDP for FLUTE Session with One Channel

Figure 2 shows an example SDP description for FLUTE session with one channel.

```
v=0
o=user123 2890844526 2890842807 IN IP6 2001:0DB8::112E:144A:1E24
s=File delivery session example
i=More information
t=2873397496 2873404696
a=source-filter: incl IN IP6 * 2001:0DB8:1:2:240:96FF:FE25:8EC9
a=FEC-declaration:0 encoding-id=0
a=FEC-declaration:1 encoding-id=129; instance-id=0’
a=group:CS 1 2
a=group:CS 3 4
m=application 12345 FLUTE/UDP *
c=IN IP6 FF33::8000:1
a=flute-tsi:1
a=FEC:0
a=mid:1
m=application 12346 FLUTE/UDP *
c=IN IP6 FF33::8000:2
a=mid:2
m=application 12347 FLUTE/UDP *
c=IN IP6 FF33::8000:3
a=flute-tsi:2
a=FEC:1
a=mid:3
m=application 12348 FLUTE/UDP *
c=IN IP6 FF33::8000:4
a=mid:4
```

Figure 3: An SDP for composite FLUTE session

Figure 3 shows an example SDP description for composite FLUTE session.

5. Security Considerations

See [RFC4566] for security considerations specific to the Session Description Protocol in general. See also [RFC4570] for security consideration related to source address filters.

[I-D.ietf-rmt-flute-revised] provides security consideration regarding FLUTE sessions. The current document does not introduce additional security considerations beyond these prior specifications.

6. IANA Considerations

6.1. Transport Protocol

The "proto" sub-field of the media description field ("m=") describes the transport protocol used. This document registers one value: "FLUTE/UDP" is a reference to FLUTE [I-D.ietf-rmt-flute-revised] running over UDP/IP.

6.1.1. Media formats ("fmt")

FLUTE media using the "FLUTE/UDP" proto value may use the character "*" as their "fmt" value. The "*" character represents a wild card which indicates that miscellaneous and unspecified MIME types are contained in the FLUTE session. Alternatively a list of MIME types (file formats) may be given in the "fmt" list. These formats SHOULD be registered. Use of an existing MIME subtype for the format is encouraged. If no MIME subtype exists, it is RECOMMENDED that a suitable one is registered through the IETF process as decribed in RFC4289 [RFC4289].

6.2. Attribute Names

As recommended by [RFC4566], the new attribute names "flute-tsi", "flute-ch", "FEC-declaration", "FEC", "FEC-OTI-extension" and "content-desc" should be registered with IANA, as follows:

The following contact information shall be used for all registrations included here:

Contact: Rod Walsh
EMail: rod.walsh (at) nokia.com

SDP Attribute ("att-field"):
Attribute name: flute-tsi
Long form: FLUTE Transport Session Identifier
Type of name: att-field
Type of attribute: Session level or media level
Subject to charset: No
Purpose: See this document
Reference: This document
Values: See this document

SDP Attribute ("att-field"):
Attribute name: flute-ch
Long form: Number of Channels in a FLUTE Session
Type of name: att-field
Type of attribute: Session level or media level

```
   Subject to charset: No
   Purpose:            See this document
   Reference:          This document
   Values:             See this document

   SDP Attribute ("att-field"):
   Attribute name:     FEC-declaration
   Long form:          Forward Error Correction Declaration
   Type of name:       att-field
   Type of attribute:  Session level or media level
   Subject to charset: No
   Purpose:            See this document
   Reference:          This document
   Values:             See this document

   SDP Attribute ("att-field"):
   Attribute name:     FEC
   Long form:          A Reference to FEC Declaration
   Type of name:       att-field
   Type of attribute:  Media level
   Subject to charset: No
   Purpose:            See this document
   Reference:          This document
   Values:             See this document

   SDP Attribute ("att-field"):
   Attribute name:     FEC-OTI-extension
   Long form:          FEC Object Transmission Information extension
   Type of name:       att-field
   Type of attribute:  Session level or media level
   Subject to charset: No
   Purpose:            See this document
   Reference:          This document
   Values:             See this document

   SDP Attribute ("att-field"):
   Attribute name:     content-desc
   Long form:          Content Description Pointer
   Type of name:       att-field
   Type of attribute:  Session level or media level
   Subject to charset: No
   Purpose:            See this document
   Reference:          This document
   Values:             See this document
```

6.3. Composite Session Token to Differentiate FLUTE Sessions

IANA needs to register the following new 'semantics' attribute for the SDP grouping framework [RFC5888]:

Semantics	Token	Reference
Composite Session	CS	This document

It should be registered in the SDP parameters registry (http://www.iana.org/assignments/sdp-parameters) under Semantics for the "group" SDP Attribute.

7. Acknowledgements

The authors would like to thank all the people who gave feedback on this document.

8. Contributors

Magnus Westerlund
Ericsson Research
Ericsson AB
SE-164 80 Stockholm
Sweden
EMail: Magnus.Westerlund (at) ericsson.com

Joerg Ott
Aalto University
Otakaari 5A
FI-02150 Espoo
Finland
EMail: jo (at) netlab.tkk.fi

9. Change Log

9.1. From draft-mehta-rmt-flute-sdp-06 to draft-ietf-rmt-flute-sdp-00

Document name changed to reflect the status as an official working group draft.

All editorial notes removed, except those related to open CC and SEC discussion.

Clarified Composite Session Semantics regarding primary media. "The first media line declared..." changed to "The first (leftmost) mid value declared...".

Clarifying note added to disambiguate the apprent difference between LCT and SDP "guard interval" for session expiry and session id reuse. This include the non manditory recommendation to avoid TSI reuse whenever possible (e.g. for a 48bit TSI and non-extremely-high-frequancy session changes from a specific sender, this would often be the case).

Documented the previously implicit assumption regarding wall clock syncronization.

RFC5445 reference corrected (now in the informative references section).

10. References

10.1. Normative References

[I-D.ietf-rmt-flute-revised]
Paila, T., Walsh, R., Luby, M., Roca, V., and R. Lehtonen, "FLUTE - File Delivery over Unidirectional Transport", draft-ietf-rmt-flute-revised-12 (work in progress), February 2011.

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

[RFC5775] Luby, M., Watson, M., and L. Vicisano, "Asynchronous Layered Coding (ALC) Protocol Instantiation", RFC 5775, April 2010.

[RFC5651] Luby, M., Watson, M., and L. Vicisano, "Layered Coding Transport (LCT) Building Block", RFC 5651, October 2009.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

[RFC4570] Quinn, B. and R. Finlayson, "Session Description Protocol (SDP) Source Filters", RFC 4570, July 2006.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

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

[RFC5888] Camarillo, G. and H. Schulzrinne, "The Session Description Protocol (SDP) Grouping Framework", RFC 5888, June 2010.

[RFC3890] Westerlund, M., "A Transport Independent Bandwidth Modifier for the Session Description Protocol (SDP)", RFC 3890, September 2004.

10.2. Informative References

[RFC3926] Paila, T., Luby, M., Lehtonen, R., Roca, V., and R. Walsh, "FLUTE - File Delivery over Unidirectional Transport", RFC 3926, October 2004.

[RFC2974] Handley, M., Perkins, C., and E. Whelan, "Session Announcement Protocol", RFC 2974, October 2000.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC4145] Yon, D. and G. Camarillo, "TCP-Based Media Transport in the Session Description Protocol (SDP)", RFC 4145, September 2005.

[RFC4607] Holbrook, H. and B. Cain, "Source-Specific Multicast for IP", RFC 4607, August 2006.

[RFC2326] Schulzrinne, H., Rao, A., and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998.

[RFC4289] Freed, N. and J. Klensin, "Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures", BCP 13, RFC 4289, December 2005.

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

[RFC5905] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

[3GPP.26.346] 3GPP, "Multimedia Broadcast/Multicast Service (MBMS); Protocols and codecs", 3GPP TS 26.346 6.13.0, March 2009.

Appendix A. Use of FEC attributes with RTP sessions (informative)

The "FEC-declaration" and "FEC" attributes provide general FEC-OTI information in FEC Encoding ID and FEC Instance ID values. These may also be used for RTP sessions employing same FEC Building Block (e.g. as is done for 3GPP MBMS [3GPP.26.346]). However, semantics of RTP are different from FLUTE (FEC is per session not per object) and RTP does not have in-band mechanism to signal FEC OTI extensions. Thus, RTP FEC declarations are expected to be used for machine configuration as well as capability requirements specification (for FLUTE it is generally only the latter).

Hence, the FLUTE SDP, defined in this document, may be extended using a "FEC-OTI-extension" attribute, depending on the configuration needs of the FEC decoder used and the lack of an alternative means to signal the extended FEC-OTI information. The purpose of extended FEC-OTI information is to define FEC code/instance-specific OTI required for receiver FEC payload configuration. The contents of such an extension would be FEC code-specific and exact specification, beyond adherence to the ABNF below, needs to be specified by any FEC code using this attribute, and hence is outside the scope of this Appendix.

A "FEC-OTI-extension" attribute must be immediately preceded by its associated "FEC-declaration" attribute and so the full FEC-OTI, including extension, will be found in two neighbouring attribute lines. The fec-ref value binds a "FEC-OTI-extension" and "FEC-declaration attribute" pair.

The syntax for the attribute in ABNF is given below:

```
fec-oti-extension-line = "a=FEC-OTI-extension:" fec-ref SP
    oti-extension CRLF
oti-extension = base64
base64 = *base64-unit [base64-pad]
base64-unit = 4base64-char
base64-pad = 2base64-char "==" / 3base64-char "="
base64-char = ALPHA / DIGIT / "+" / "/"
```

Appendix B. Further Design Logic for FEC-OTI Descriptors

There are several reasons that the FEC Encoding and Instance IDs are optional capabilities descriptions:

1. It is not always necessary to explicitly describe the FEC capabilities in advance of the session - e.g. for simple (and short) sessions it can be more elegant to discover this from the session (FDT) itself (even when some mechanism for machine-readable session parameters, such as IP addresses and ports, is wanted in advance).

2. There may be some other out-of-band discovery of FEC capability requirements (e.g. well known-FEC/standardised capabilities for a certain application, verbal agreement between a group, etc.) that provides the FEC capability information. This document does not want to prevent this, and in this case repeating the information in SDP would be unnecessary and wasteful (and probably result in implementations not following the flute-sdp specification).

3. FLUTE defaults to Compact No-Code FEC [RFC5445] and support for this is mandatory for FLUTE anyway so it is a given (capability requirement) which does not need to be described by the SDP. In cases where only Compact No-Code FEC is required, there is no use in specifying any FEC Encoding (and Instance) IDs in the SDP (though it is allowed).

4. In cases where a FLUTE session description (SDP file) is not defined once for all time, it is possible that the FEC usage is not known in advance and the FEC capabilities would only be added to the SDP in a later version of that SDP file when the FEC codes have been selected (e.g. a larger audience may suggest stronger FEC to make FLUTE delivery more reliable, whereas additional bi-directional messages may be scalable for smaller groups).

5. Also, in cases where a FLUTE session description (SDP file) is very static (e.g. once for all time for that session), it is possible that the FEC usage is not known in advance and it needs to be left to some other mechanism (e.g. FDT) to discover any FEC capability requirements set closer to the session transmission - with the same examples as mentioned above.

Also, in a complex case of very many FEC codes being used in the session giving a full list in SDP is not seen as being reasonable (but this is likely to be a rare case anyway).

Authors' Addresses

Rod Walsh
Nokia Research Center
P.O. Box 100 (Visiokatu 1)
Tampere FI-33721
Finland

Email: rod.walsh (at) nokia.com

Igor D.D. Curcio
Nokia Research Center
P.O. Box 88 (Tieteenkatu 1)
Tampere FI-33721
Finland

Email: igor.curcio (at) nokia.com

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

Email: jani.peltotalo (at) tut.fi

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

Email: sami.peltotalo (at) tut.fi

Harsh Mehta

Email: harsh.mehta (at) gmail.com

RMT
Internet-Draft
Intended status: Experimental
Expires: January 3, 2011

V. Roca
INRIA

July 2, 2010

Simple Authentication Schemes for the ALC and NORM Protocols
draft-ietf-rmt-simple-auth-for-alc-norm-03

Abstract

This document introduces four schemes that provide a per-packet authentication and integrity service in the context of the ALC and NORM protocols. The first scheme is based on digital signatures. Because it relies on asymmetric cryptography, this scheme generates a high processing load at the sender and to a lesser extent at a receiver, as well as a significant transmission overhead. It is therefore well suited to low data rate sessions. The second scheme relies on the Elliptic Curve Digital Signature Algorithm (ECDSA). If this approach also relies an asymmetric cryptography, the processing load and the transmission overhead are significantly reduced compared to traditional digital signature schemes. It is therefore well suited to medium data rate sessions. The third scheme relies on a group Message Authentication Code (MAC). Because this scheme relies on symmetric cryptography, MAC calculation and verification are fast operations, which makes it suited to high data rate sessions. However it only provides a group authentication and integrity service, which means that it only protects against attackers that are not group members. Finally, the fourth scheme merges the digital signature and group group schemes, and is useful to mitigate DoS attacks coming from attackers that are not group members.

Table of Contents

1. Introduction

Many applications using multicast and broadcast communications require that each receiver be able to authenticate the source of any packet it receives to check check its integrity. For instance, ALC [RFC5775] and NORM [RFC5740] are two Content Delivery Protocols (CDP) designed to transfer reliably objects (e.g. files) between a session's sender and several receivers.

The NORM protocol is based on bidirectional transmissions. Each receiver acknowledges data received or, in case of packet erasures, asks for retransmissions. On the opposite, the ALC protocol defines unidirectional transmissions. Reliability can be achieved by means of cyclic transmissions of the content within a carousel, or by the use of proactive Forward Error Correction codes (FEC), or by the joint use of these mechanisms. Being purely unidirectional, ALC is massively scalable, while NORM is intrinsically limited in terms of the number of receivers that can be handled in a session. Both protocols have in common the fact that they operate at application level, on top of an erasure channel (e.g. the Internet) where packets can be lost (erased) during the transmission.

With these CDP, an attacker might impersonate the ALC or NORM session sender and inject forged packets to the receivers, thereby corrupting the objects reconstructed by the receivers. An attacker might also impersonate a NORM session receiver and inject forged feedback packets to the NORM sender.

In case of group communications, several solutions exist to provide the receiver some guaranties on the integrity of the packets it receives and on the identity of the sender of these packets. These solutions have different features that make them more or less suited to a given use case:

- digital signatures [RFC4359]: this scheme is well suited to low data rate flows, when a true packet sender authentication and packet integrity service is needed. However, digital signatures based on RSA asymmetric cryptography is limited by high computational costs and high transmission overheads. The use of ECC ("Elliptic Curve Cryptography") significantly relaxes these constraints, especially when seeking for higher security levels. For instance, the following key sizes provide equivalent security: 1024 bit RSA key versus 160 bit ECC key, or 2048 bit RSA key versus 224 bit ECC key.

- group Message Authentication Codes (MAC): this scheme is well suited to high data rate flows, when transmission overheads must be minimized. However this scheme cannot protect against attacks

coming from inside the group, where a group member impersonates the sender and sends forged messages to other receivers.

- TESLA (Timed Efficient Stream Loss-tolerant Authentication) [RFC4082][RFC5776]: this scheme is well suited to high data rate flows, when transmission overheads must be minimized, and when a true packet sender authentication and packet integrity service is needed. The price to pay is an increased complexity, in particular the need to loosely synchronize the receivers and the sender, as well as the need to wait for the key to be disclosed before being able to authenticate a packet.

The following table summarizes the pros/cons of each authentication/ integrity scheme used at application/transport level (where "-" means bad, "0" means neutral, and "+" means good):

	RSA Digital Signature	ECC Digital Signature	Group MAC	TESLA
True auth and integrity	Yes	Yes	No (group security)	Yes
Immediate auth	Yes	Yes	Yes	No
Processing load	-	0	+	+
Transmission overhead	-	0	+	+
Complexity	+	+	+	-

Several authentication schemes MAY be used in the same ALC or NORM session, even on the same communication path. Since all the above schemes make use of the same authentication header extension mechanism (Section 2.3, Section 4.3, Section 5.3) and [RFC5776], section 5.1), the same 4-bit "ASID" (Authentication Scheme IDentifier) has been reserved in all the specifications. The association between the "ASID" value and the actual authentication scheme is defined at session startup and communicated to all the group members by an out-of-band mechanism.

1.1. Scope of this Document

[RFC5776] explains how to use TESLA in the context of ALC and NORM protocols.

The current document specifies the use of the Digital Signature based on RSA asymmetric cryptography, the Elliptic Curve Digital Signature Algorithm (ECDSA) and Group MAC schemes. The current document also specifies the joint use of Digital Signature and Group MAC schemes which is useful to mitigate DoS attacks coming from attackers that are not group members.

Unlike the TESLA scheme, this specification considers the authentication/integrity of the packets generated by the session's sender as well as those generated by the receivers (NORM).

All the applications build on top of ALC and NORM directly benefit from the source authentication and packet integrity services defined in this document. For instance this is the case of the FLUTE application [RMT-FLUTE] built on top of ALC.

The current specification assumes that several parameters (like keying material) are communicated out-of-band, sometimes securely, between the sender and the receivers. This is detailed in Section 2.2 and Section 4.2.

1.2. Conventions Used in this Document

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].

1.3. Terminology and Notations

The following notations and definitions are used throughout this document:

- MAC is the Message Authentication Code;
- HMAC is the Keyed-Hash Message Authentication Code;
- sender denotes the sender of a packet that needs the authentication/integrity check service. It can be an ALC or NORM session sender, or a NORM session receiver in case of feedback traffic;
- receiver denotes the receiver of a packet that needs the authentication/integrity check service. It can be an ALC or NORM session receiver, or a NORM session sender in case of feedback traffic;

Digital signature related notations and definitions:

- K_pub is the public key used by a receiver to check a packet's signature. This key MUST be communicated to all receivers, before starting the session;

- K_priv is the private key used by a sender to generate a packet's signature;

- n_k is the private key and public key length, in bits. n_k is also the signature length, since both values are equal with digital signatures;

Group MAC related notations and definitions:

- K_g is a shared group key used by the senders and the receivers. This key MUST be communicated to all group members, confidentially, before starting the session;

- n_k is the group key length, in bits;

- n_m is the length of the truncated output of the MAC [RFC2104]. Only the n_m left-most bits (most significant bits) of the MAC output are kept;

2. RSA Digital Signature Scheme

2.1. Principles

The computation of the digital signature, using K_priv, MUST include the ALC or NORM header (with the various header extensions) and the payload when applicable. The UDP/IP/MAC headers MUST NOT be included. During this computation, the "Signature" field MUST be set to 0.

Upon receiving this packet, the receiver recomputes the Group MAC, using K_pub, and compares it to the value carried in the packet. During this computation, the Weak Group MAC field MUST also be set to 0. If the check fails, the packet MUST be immediately dropped.

Several "Signature Encoding Algorithms" can be used, including RSASSA-PKCS1-v1_5 and RSASSA-PSS. With these encodings, several "Signature Cryptographic Function" can be used, like SHA-256.

First, let us consider a packet sender. More specifically, from [RFC4359]: digital signature generation is performed as described in [RFC3447], Section 8.2.1 for RSASSA-PKCS1-v1_5 and Section 8.1.1 for RSASSA-PSS. The authenticated portion of the packet is used as the message M, which is passed to the signature generation function. The signer's RSA private key is passed as K. In summary (when SHA-256 is used), the signature generation process computes a SHA-256 hash of the authenticated packet bytes, signs the SHA-256 hash using the private key, and encodes the result with the specified RSA encoding type. This process results in a value S, which is the digital signature to be included in the packet.

With RSASSA-PKCS1-v1_5 and RSASSA-PSS signatures, the size of the signature is equal to the "RSA modulus", unless the "RSA modulus" is not a multiple of 8 bits. In that case, the signature MUST be prepended with between 1 and 7 bits set to zero such that the signature is a multiple of 8 bits [RFC4359]. The key size, which in practice is also equal to the "RSA modulus", has major security implications. [RFC4359] explains how to choose this value depending on the maximum expected lifetime of the session. This choice is out of the scope of this document.

Now let us consider a receiver. From [RFC4359]: Digital signature verification is performed as described in [RFC3447], Section 8.2.2 (RSASSA-PKCS1-v1_5) and [RFC3447], Section 8.1.2 (RSASSA-PSS). Upon receipt, the digital signature is passed to the verification function as S. The authenticated portion of the packet is used as the message M, and the RSA public key is passed as (n, e). In summary (when SHA-256 is used), the verification function computes a SHA-256 hash of

the authenticated packet bytes, decrypts the SHA-256 hash in the packet using the sender's public key, and validates that the appropriate encoding was applied. The two SHA-256 hashes are compared and if they are identical the validation is successful.

2.2. Parameters

Several parameters MUST be initialized by an out-of-band mechanism. The sender or group controller:

- MUST communicate his public key, for each receiver to be able to verify the signature of the packets received. As a side effect, the receivers also know the key length, n_k, and the signature length, the two parameters being equal;
- MAY communicate a certificate (which also means that a PKI has been setup), for each receiver to be able to check the sender's public key;
- MUST communicate the Signature Encoding Algorithm. For instance, [RFC3447] defines the RSASSA-PKCS1-v1_5 and RSASSA-PSS algorithms that are usually used to that purpose;
- MUST communicate the Signature Cryptographic Function, for instance SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512. Because of security threats on SHA-1, the use of SHA-256 is RECOMMENDED;
- MUST associate a value to the "ASID" field (Authentication Scheme Identifier) of the EXT_AUTH header extension (Section 2.3);

These parameters MUST be communicated to all receivers before they can authenticate the incoming packets. For instance it can be communicated in the session description, or initialized in a static way on the receivers, or communicated by means of an appropriate protocol. The details of this out-of-band mechanism are out of the scope of this document.

2.3. Authentication Header Extension Format

The integration of Digital Signatures is similar in ALC and NORM and relies on the header extension mechanism defined in both protocols. More precisely this document details the EXT_AUTH==1 header extension defined in [RFC5651].

Several fields are added in addition to the HET (Header Extension Type) and HEL (Header Extension Length) fields (Figure 1).

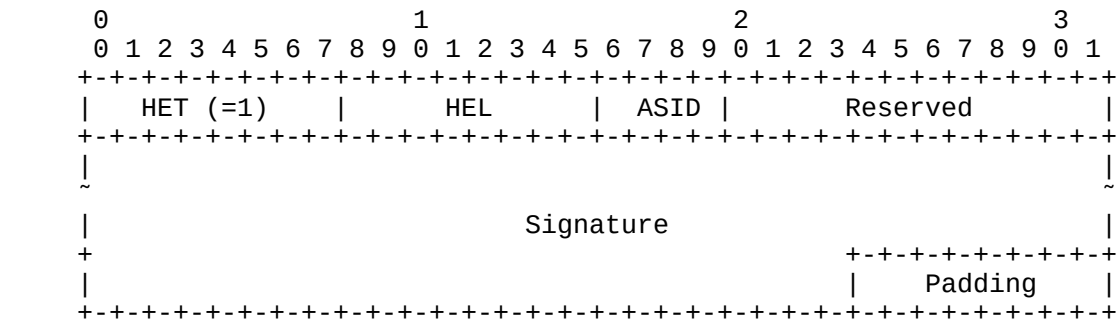

Figure 1: Format of the Digital Signature EXT_AUTH header extension.

The fields of the Digital Signature EXT_AUTH header extension are:

"ASID" (Authentication Scheme Identifier) field (4 bits):

The "ASID" identifies the source authentication scheme or protocol in use. The association between the "ASID" value and the actual authentication scheme is defined out-of-band, at session startup.

"Reserved" field (12 bits):

This is a reserved field that MUST be set to zero in this specification.

"Signature" field (variable size, multiple of 32 bits):

The "Signature" field contains a digital signature of the message. If need be, this field is padded (with 0) up to a multiple of 32 bits.

2.4. In Practice

Each packet sent MUST contain exactly one Digital Signature EXT_AUTH header extension. A receiver MUST drop all the packets that do not contain a Digital Signature EXT_AUTH header extension.

All receivers MUST recognize EXT_AUTH but MAY not be able to parse its content, for instance because they do not support digital signatures. In that case the Digital Signature EXT_AUTH header extension is ignored.

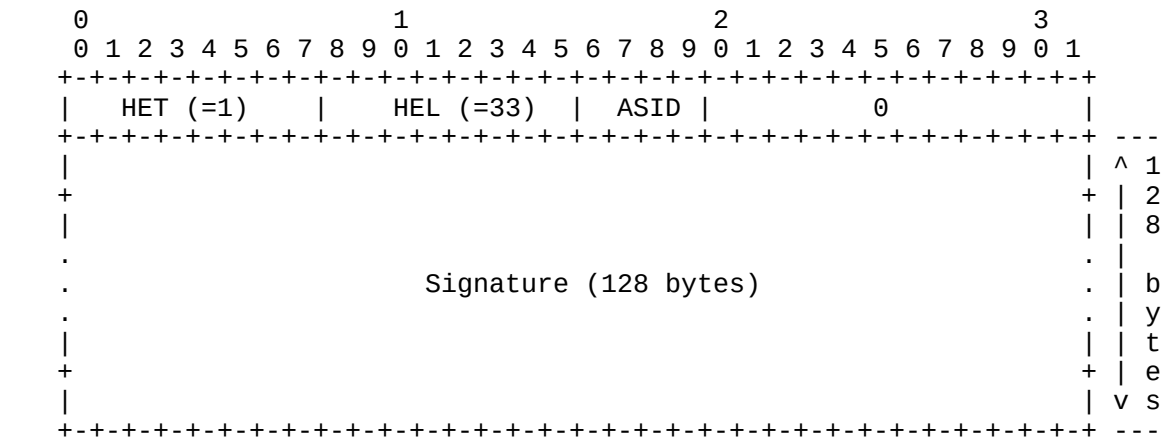

Figure 2: Example: Format of the Digital Signature EXT_AUTH header extension using 1024 bit signatures.

For instance Figure 2 shows the digital signature EXT_AUTH header extension when using 128 byte (1024 bit) key digital signatures (which also means that the signature field is 128 byte long). The Digital Signature EXT_AUTH header extension is then 132 byte long.

3. Elliptic Curve Digital Signature Scheme

3.1. Principles

The computation of the ECC digital signature, using K_priv, MUST include the ALC or NORM header (with the various header extensions) and the payload when applicable. The UDP/IP/MAC headers MUST NOT be included. During this computation, the "Signature" field MUST be set to 0.

Upon receiving this packet, the receiver recomputes the Group MAC, using K_pub, and compares it to the value carried in the packet. During this computation, the Weak Group MAC field MUST also be set to 0. If the check fails, the packet MUST be immediately dropped.

Several "Elliptic Curves" groups can be used, as well as several "Hash Algorithms". In practice both choices are related and there is a minimum hash algorithm size for any key size. Using a larger hash algorithm and then truncated the output is also feasible, however it consumes more processing power than is necessary. The following table lists the RECOMMENDED choices [RFC4754] [RFC5480].

Digital Signature Algorithm name [RFC4754]	Key Size (n_k)	Message Digest Algorithm	Elliptic Curve
ECDSA-256	256	SHA-256	secp256r1
ECDSA-384	384	SHA-384	secp384r1
ECDSA-521	512	SHA-512	secp521r1

The ECDSA-256, ECDSA-384 and ECDSA-521 are designed to offer security comparable with AES-128, AES-192 and AES-256 respectively [RFC4754].

3.2. Parameters

Several parameters MUST be initialized by an out-of-band mechanism. The sender or group controller:

- MUST communicate his public key, for each receiver to be able to verify the signature of the packets received. As a side effect, the receivers also know the key length, n_k, and the signature length, the two parameters being equal;

- MAY communicate a certificate (which also means that a PKI has been setup), for each receiver to be able to check the sender's public key;
- MUST communicate the Message Digest Algorithm;
- MUST communicate the Elliptic Curve;
- MUST associate a value to the "ASID" field (Authentication Scheme Identifier) of the EXT_AUTH header extension (Section 2.3);

These parameters MUST be communicated to all receivers before they can authenticate the incoming packets. For instance it can be communicated in the session description, or initialized in a static way on the receivers, or communicated by means of an appropriate protocol. The details of this out-of-band mechanism are out of the scope of this document.

3.3. Authentication Header Extension Format

The integration of ECC Digital Signatures is similar in ALC and NORM and relies on the header extension mechanism defined in both protocols. More precisely this document details the EXT_AUTH==1 header extension defined in [RFC5651].

Several fields are added in addition to the HET (Header Extension Type) and HEL (Header Extension Length) fields (Figure 1).

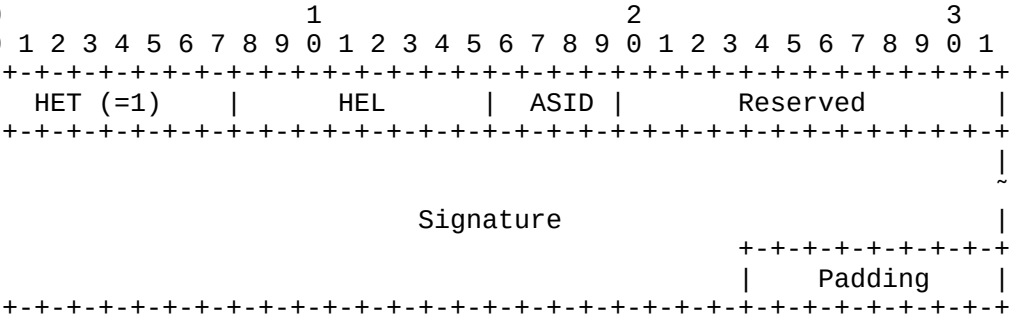

Figure 3: Format of the Digital Signature EXT_AUTH header extension.

The fields of the Digital Signature EXT_AUTH header extension are:

"ASID" (Authentication Scheme Identifier) field (4 bits):

The "ASID" identifies the source authentication scheme or protocol in use. The association between the "ASID" value and the actual authentication scheme is defined out-of-band, at session startup.

"Reserved" field (12 bits):

This is a reserved field that MUST be set to zero in this specification.

"Signature" field (variable size, multiple of 32 bits):

The "Signature" field contains a digital signature of the message. If need be, this field is padded (with 0) up to a multiple of 32 bits.

3.4. In Practice

Each packet sent MUST contain exactly one ECC Digital Signature EXT_AUTH header extension. A receiver MUST drop all the packets that do not contain an ECC Digital Signature EXT_AUTH header extension.

All receivers MUST recognize EXT_AUTH but MAY not be able to parse its content, for instance because they do not support ECC digital signatures. In that case the Digital Signature EXT_AUTH header extension is ignored.

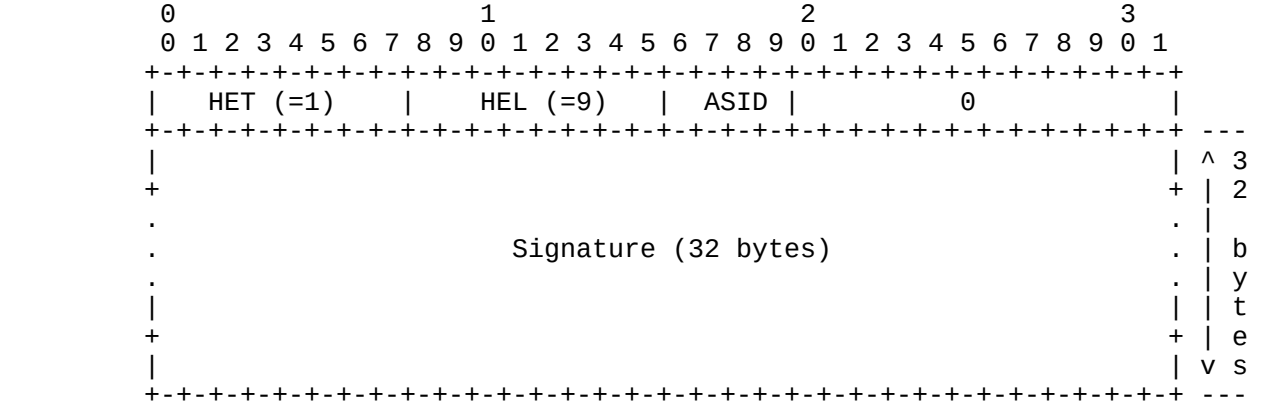

Figure 4: Example: Format of the ECC Digital Signature EXT_AUTH header extension using ECDSA-256 signatures.

For instance Figure 4 shows the digital signature EXT_AUTH header extension when using ECDSA-256 (256 bit) ECC digital signatures. The ECC Digital Signature EXT_AUTH header extension is then 36 byte long.

4. Group Message Authentication Code (MAC) Scheme

4.1. Principles

The computation of the Group MAC, using K_g, includes the ALC or NORM header (with the various header extensions) and the payload when applicable. The UDP/IP/MAC headers are not included. During this computation, the Weak Group MAC field MUST be set to 0. Then the sender truncates the MAC output to keep the n_m most significant bits and stores the result in the Group MAC Authentication header.

Upon receiving this packet, the receiver recomputes the Group MAC, using K_g, and compares it to the value carried in the packet. During this computation, the Group MAC field MUST also be set to 0. If the check fails, the packet MUST be immediately dropped.

[RFC2104] explains that it is current practice to truncate the MAC output, on condition that the truncated output length, n_m be not less than half the length of the hash and not less than 80 bits. However, this choice is out of the scope of this document.

4.2. Parameters

Several parameters MUST be initialized by an out-of-band mechanism. The sender or group controller:

- MUST communicate the Cryptographic MAC Function, for instance, HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, or HMAC-SHA-512. Because of security threats on SHA-1, the use of HMAC-SHA-256 is RECOMMENDED. As a side effect, the receivers also know the key length, n_k, and the non truncated MAC output length;

- MUST communicate the length of the truncated output of the MAC, n_m, which depends on the Cryptographic MAC Function chosen. Only the n_m left-most bits (most significant bits) of the MAC output are kept. Of course, n_m MUST be lower or equal to n_k;

- MUST communicate the K_g group key to the receivers, confidentially, before starting the session. This key might have to be periodically refreshed for improved robustness;

- MUST associate a value to the "ASID" field (Authentication Scheme Identifier) of the EXT_AUTH header extension (Section 4.3);

These parameters MUST be communicated to all receivers before they can authenticate the incoming packets. For instance it can be communicated in the session description, or initialized in a static way on the receivers, or communicated by means of an appropriate

protocol (this will be often the case when periodic re-keying is required). The details of this out-of-band mechanism are out of the scope of this document.

4.3. Authentication Header Extension Format

The integration of Group MAC is similar in ALC and NORM and relies on the header extension mechanism defined in both protocols. More precisely this document details the EXT_AUTH==1 header extension defined in [RFC5651].

Several fields are added in addition to the HET (Header Extension Type) and HEL (Header Extension Length) fields (Figure 5).

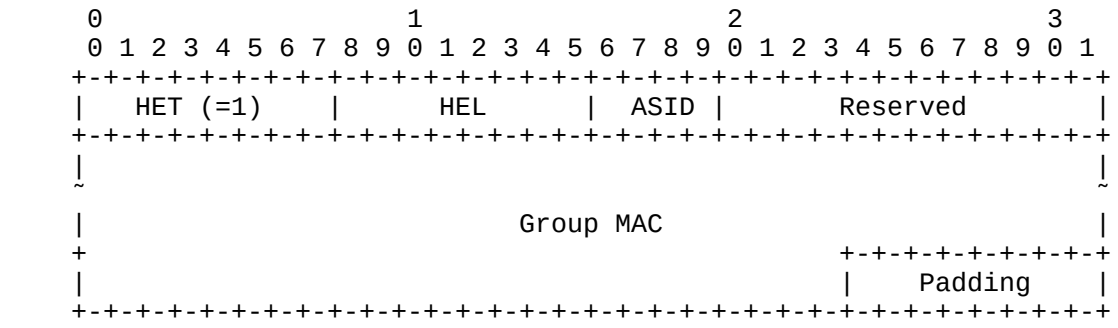

Figure 5: Format of the Group MAC EXT_AUTH header extension.

The fields of the Group MAC EXT_AUTH header extension are:

"ASID" (Authentication Scheme Identifier) field (4 bits):

The "ASID" identifies the source authentication scheme or protocol in use. The association between the "ASID" value and the actual authentication scheme is defined out-of-band, at session startup.

"Reserved" field (12 bits):

This is a reserved field that MUST be set to zero in this specification.

"Group MAC" field (variable size, multiple of 32 bits):

The "Group MAC" field contains a truncated Group MAC of the message. If need be, this field is padded (with 0) up to a multiple of 32 bits.

4.4. In Practice

Each packet sent MUST contain exactly one Group MAC EXT_AUTH header extension. A receiver MUST drop packets that do not contain a Group MAC EXT_AUTH header extension.

All receivers MUST recognize EXT_AUTH but MAY not be able to parse its content, for instance because they do not support Group MAC. In that case the Group MAC EXT_AUTH extension is ignored.

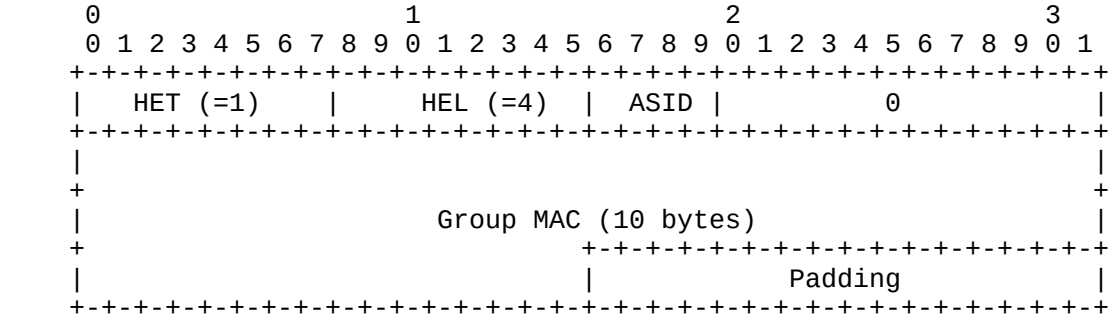

Figure 6: Example: Format of the Group MAC EXT_AUTH header extension using HMAC-SHA-1.

For instance Figure 6 shows the Group MAC EXT_AUTH header extension when using HMAC-SHA-1. The Group MAC EXT_AUTH header extension is then 16 byte long.

5. Combined Use of the RSA/ECC Digital Signatures and Group MAC Schemes

5.1. Principles

In some situations, it can be interesting to use both authentication schemes. The goal of the Group MAC is to mitigate DoS attacks coming from attackers that are not group members [RFC4082] by adding a light authentication scheme as a front-end.

More specifically, before sending a message, the sender sets the Signature field and Group MAC field to zero. Then the sender computes the Signature as detailed in Section 2.1 or in Section 3.1 and stores the value in the Signature field. Then the sender computes the Group MAC as detailed in Section 4.1 and stores the value in the Group MAC field. The (RSA or ECC) digital signature value is therefore protected by the Group MAC, which avoids DoS attacks where the attacker corrupts the digital signature itself.

Upon receiving the packet, the receiver first checks the Group MAC, as detailed in Section 4.1. If the check fails, the packet MUST be immediately dropped. Otherwise the receiver checks the Digital Signature, as detailed in Section 2.1. If the check fails, the packet MUST be immediately dropped.

This scheme features a few limits:

- the Group MAC is of no help if a group member (who knows K_g) impersonates the sender and sends forged messages to other receivers. DoS attacks are still feasible;
- it requires an additional MAC computing for each packet, both at the sender and receiver sides;
- it increases the size of the authentication headers. In order to limit this problem, the length of the truncated output of the MAC, n_m, SHOULD be kept small (see [RFC3711] section 9.5). In the current specification, n_m MUST be a multiple of 32 bits, and default value is 32 bits. As a side effect, with $n_m = 32$ bits, the authentication service is significantly weakened since the probability that any packet be successfully forged is one in 2^32. Since the Group MAC check is only a pre-check that is followed by the standard signature authentication check, this is not considered to be an issue.

For a given use-case, the benefits brought by the Group MAC must be balanced against these limitations.

5.2. Parameters

Several parameters MUST be initialized by an out-of-band mechanism, as defined in Section 2.2, Section 3.2 and Section 4.2.

5.3. Authentication Header Extension Format

The integration of combined RSA/ECC Digital Signature and Group MAC is similar in ALC and NORM and relies on the header extension mechanism defined in both protocols. More precisely this document details the EXT_AUTH==1 header extension defined in [RFC5651].

Several fields are added in addition to the HET (Header Extension Type) and HEL (Header Extension Length) fields (Figure 7).

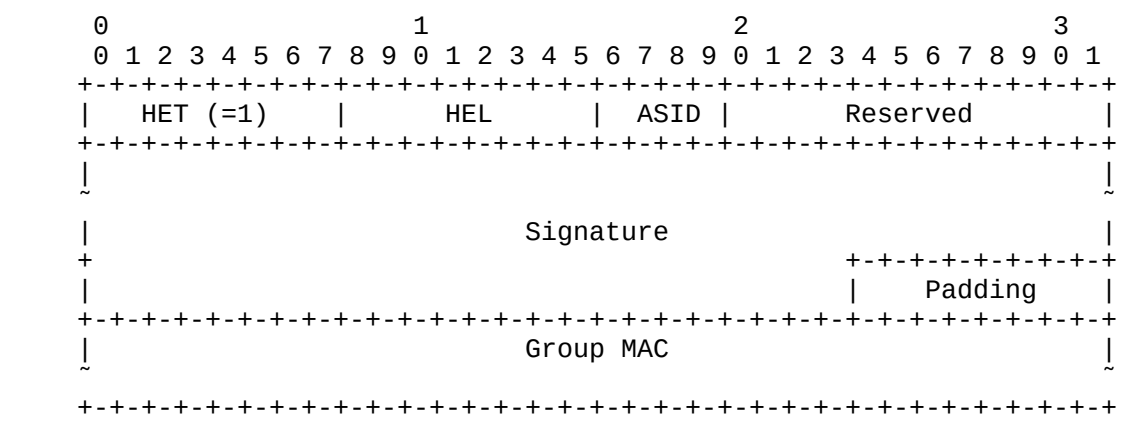

Figure 7: Format of the Group MAC EXT_AUTH header extension.

The fields of the Group MAC EXT_AUTH header extension are:

"ASID" (Authentication Scheme Identifier) field (4 bits):

The "ASID" identifies the source authentication scheme or protocol in use. The association between the "ASID" value and the actual authentication scheme is defined out-of-band, at session startup.

"Reserved" field (12 bits):

This is a reserved field that MUST be set to zero in this specification.

"Signature" field (variable size, multiple of 32 bits):

The "Signature" field contains a digital signature of the message. If need be, this field is padded (with 0) up to a multiple of 32 bits.

"Group MAC" field (variable size, multiple of 32 bits, by default 32 bits):

The "Group MAC" field contains a truncated Group MAC of the message.

5.4. In Practice

Each packet sent MUST contain exactly one combined Digital Signature/Group MAC EXT_AUTH header extension. A receiver MUST drop packets that do not contain a combined Digital Signature/Group MAC EXT_AUTH header extension.

All receivers MUST recognize EXT_AUTH but MAY not be able to parse its content, for instance because they do not support combined Digital Signature/Group MAC. In that case the combined Digital Signature/Group MAC EXT_AUTH extension is ignored.

It is RECOMMENDED that the n_m parameter of the group authentication scheme be small, and by default equal to 32 bits (Section 5.1).

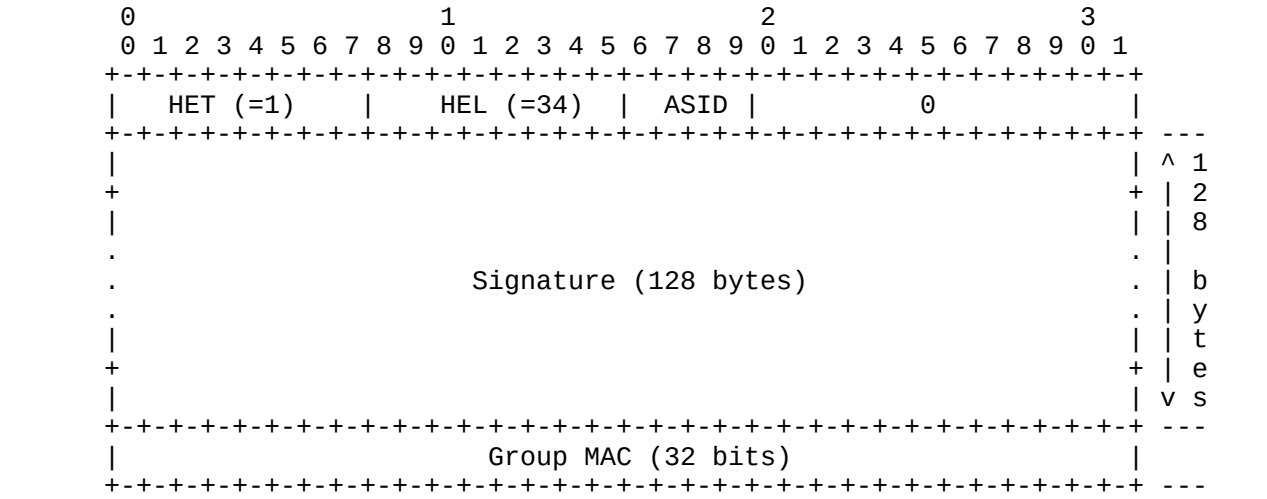

Figure 8: Example: Format of the combined RSA Digital Signature/Group MAC EXT_AUTH header extension using 1024 bit signatures.

For instance Figure 8 shows the combined Digital Signature/Group MAC

EXT_AUTH header extension when using 128 byte (1024 bit) key RSA digital signatures (which also means that the signature field is 128 byte long). The EXT_AUTH header extension is then 136 byte long.

6. IANA Considerations

This document does not require any IANA registration.

7. Security Considerations

7.1. Dealing With DoS Attacks

Digital signatures introduces new opportunities for an attacker to mount DoS attacks. For instance an attacker can try to saturate the processing capabilities of the receiver (faked packets are easy to create but checking them requires to compute a costly digital signature).

In order to mitigate these attacks, it is RECOMMENDED to use the combined Digital Signature/Group MAC scheme (Section 5.1). However, no mitigation is possible if a group member acts as an attacker.

7.2. Dealing With Replay Attacks

Replay attacks consist for an attacker to store a valid message and to replay it later on.

7.2.1. Impacts of Replay Attacks on the Simple Authentication Schemes

Since all the above authentication schemes are memoryless, replay attacks have no impact on these schemes.

7.2.2. Impacts of Replay Attacks on NORM

We review here the potential impacts of a replay attack on the NORM component. Note that we do not consider here the protocols that could be used along with NORM, for instance the congestion control protocols.

First, let us consider replay attacks within a given NORM session. NORM defines a "sequence" field that can be used to protect against replay attacks [RFC5740] within a given NORM session. This "sequence" field is a 16-bit value that is set by the message originator (sender or receiver) as a monotonically increasing number incremented with each NORM message transmitted. It is RECOMMENDED that a receiver check this sequence field and drop messages considered as replayed. Similarly, it is RECOMMENDED that a sender check this sequence, for each known receiver, and drop messages considered as replayed. In both cases, checking this sequence field SHOULD be done before authenticating the packet: if the sequence field has not been corrupted, the replay attack will immediately be identified, and otherwise the packet will fail the authentication test. This analysis shows that NORM itself is robust in front of replay attacks within the same session.

Now let us consider replay attacks across several NORM sessions. A

host participation in a NORM session is uniquely identified by the {"source_id"; "instance_id"} tuple. Therefore, when a given host participates in several NORM sessions, it is RECOMMENDED that the "instance_id" be changed for each NORM instance. It is also RECOMMENDED, when the Group MAC authentication/integrity check scheme is used, that the shared group key, K_g, be changed across sessions. Therefore, NORM can be made robust in front of replay attacks across different sessions.

7.2.3. Impacts of Replay Attacks on ALC

We review here the potential impacts of a replay attack on the ALC component. Note that we do not consider here the protocols that could be used along with ALC, for instance the layered or wave based congestion control protocols.

First, let us consider replay attacks within a given ALC session:

- Regular packets containing an authentication tag: a replayed message containing an encoding symbol will be detected once authenticated, thanks to the object/block/symbol identifiers, and will be silently discarded. This kind of replay attack is only penalizing in terms of memory and processing load, but does not compromise the ALC behavior.

- Control packets containing an authentication tag: ALC control packets, by definition, do not include any encoding symbol and therefore do not include any object/block/symbol identifier that would enable a receiver to identify duplicates. However, a sender has a very limited number of reasons to send control packets. More precisely:

  * At the end of the session, a "close session" (A flag) packet is sent. Replaying this packet has no impact since the receivers already left.

  * Similarly, replaying a packet containing a "close object" (B flag) has no impact since this object is probably already marked as closed by the receiver.

This analysis shows that ALC itself is robust in front of replay attacks within the same session.

Now let us consider replay attacks across several ALC sessions. An ALC session is uniquely identified by the {sender's IP address; Transport Session Identifier (TSI)} [RFC5651]. Therefore, when a given sender creates several sessions, it is RECOMMENDED that the TSI be changed for each ALC instance. It is also RECOMMENDED, when the

Group MAC authentication/integrity check scheme is used, that the shared group key, K_g, be changed across sessions. Therefore, ALC can be made robust in front of replay attacks across different sessions.

8. Acknowledgments

The author is grateful to the authors of [RFC4359], [RFC4754] and [RFC5480] that inspired several sections of the present document.

9. References

9.1. Normative References

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

[RFC5651] Luby, M., Watson, M., and L. Vicisano, "Layered Coding Transport (LCT) Building Block", RFC 5651, October 2009.

[RFC5740] Adamson, B., Bormann, C., Handley, M., and J. Macker, "NACK-Oriented Reliable Multicast (NORM) Transport Protocol", RFC 5740, November 2009.

[RFC5775] Luby, M., Watson, M., and L. Vicisano, "Asynchronous Layered Coding (ALC) Protocol Instantiation", RFC 5775, April 2010.

9.2. Informative References

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

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

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.

[RFC4082] Perrig, A., Song, D., Canetti, R., Tygar, J., and B. Briscoe, "Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction", RFC 4082, June 2005.

[RFC4359] Weis, B., "The Use of RSA/SHA-1 Signatures within Encapsulating Security Payload (ESP) and Authentication Header (AH)", RFC 4359, January 2006.

[RFC4754] Fu, D. and J. Solinas, "IKE and IKEv2 Authentication Using the Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 4754, January 2007.

[RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, March 2009.

[RFC5776] Roca, V., Francillon, A., and S. Faurite, "Use of Timed Efficient Stream Loss-Tolerant Authentication (TESLA) in the Asynchronous Layered Coding (ALC) and NACK-Oriented Reliable Multicast (NORM) Protocols", RFC 5776, April 2010.

[RMT-FLUTE]
Paila, T., Walsh, R., Luby, M., Lehtonen, R., and V. Roca, "FLUTE - File Delivery over Unidirectional Transport", Work in Progress, March 2010.

Author's Address

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

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

RMT
Internet-Draft
Intended status: Standards Track
Expires: September 8, 2011

M. Luby
Qualcomm Incorporated
T. Stockhammer
Nomor Research

March 7, 2011

Universal Object Delivery using RaptorQ
draft-luby-uod-raptorq-00

Abstract

This document describes a Fully-Specified FEC scheme, identified by the FEC Encoding ID 7 (to be determined (tbd)), for Universal Object Delivery using the RaptorQ Forward Error Correction (FEC) Scheme for Object Delivery. This document introduces a new FEC Payload ID, called the Universal Object Symbol Identifier (UOSI), and describes how to use the UOSI together with RaptorQ FEC Scheme to provide simplified and enhanced object delivery capabilities. In particular, flexible and simple support is provided for basic object delivery, and support is also provided for unequal error protection (UEP) object delivery, for bundled object delivery, and for any combination of UEP and bundled object delivery.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 8, 2011.

Copyright Notice

Table of Contents

1. Introduction

This document describes a Fully-Specified FEC scheme, identified by the FEC Encoding ID 7 (to be confirmed (tbc)), for Universal Object Delivery using the RaptorQ FEC Scheme specified in [I-D.ietf-rmt-bb-fec-raptorq], hereafter referred to as the UOD-RaptorQ FEC Scheme. This document introduces a new FEC Payload ID, called the universal object symbol identifier (UOSI), and describes how to use the UOSI together with RaptorQ FEC Scheme [I-D.ietf-rmt-bb-fec-raptorq] to provide simplified and enhanced object delivery capabilities. In particular, more flexible and simpler support is provided for basic object delivery when compared to the generic scheme defined in[I-D.ietf-rmt-bb-fec-raptorq], and support is also provided for unequal error protection (UEP) object delivery (for example as described in [PET]), for bundled object delivery, and for any combination of UEP and bundled object delivery.

2. Requirements Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Formats and Codes

3.1. FEC Payload IDs

The FEC Payload ID MUST be a 4-octet field defined as follows:

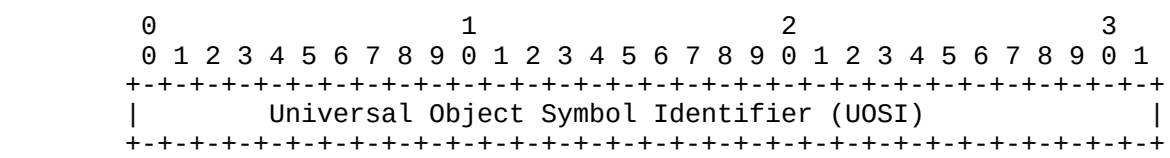

Figure 1: FEC Payload ID format

- Universal Object Symbol Identifier (UOSI), (32 bits, unsigned integer): A non-negative integer that, in conjunction with the FEC Object Transmission Information (OTI), is used to identify the encoding symbols contained within the packet.

The interpretation of the Universal Object Symbol Identifier is defined in Section 4.

3.2. FEC Object Transmission Information

3.2.1. General

For the delivery of a single object, or multiple objects, or a single object partitioned into parts with different priorities, or any combination of these, the FEC OTI is as described in this section. It should be noted that for each object delivered, the FEC OTI is exactly the same as specified in RaptorQ FEC Scheme [I-D.ietf-rmt-bb-fec-raptorq]. Each object described herein may be different for parts of the same file, or for different files, or combinations thereof. The relationship between the size of object I, F(I), and the size of the encoding symbol to be used for object I, T(I), determines the priority of object I in the transmission.

3.2.2. Mandatory

The value of the FEC Encoding ID MUST be 7 (tbd), as assigned by IANA (see Section 6).

3.2.3. Common

The Common FEC Object Transmission Information elements used by this FEC Scheme are:

- Number of objects (D), (8 bits, unsigned integer): A positive integer that is at most 255. The number of objects delivered is D, i.e., if D = 1 then there is one object (default value). D=0 MUST NOT be used.

- For I = 1,...,D, the Common FEC OTI elements specific to object I are:

  * Symbol Size (T(I)), (16 bits, unsigned integer): A positive integer that is less than 2^^16. This is the size of a symbol for object I in units of octets.

  * Transfer Length (F(I)), (40 bits, unsigned integer): A non-negative integer that is at most 2^^40. This is the transfer length of object I in units of octets.

The encoded Common FEC OTI format is shown in Figure 2 when D=2.

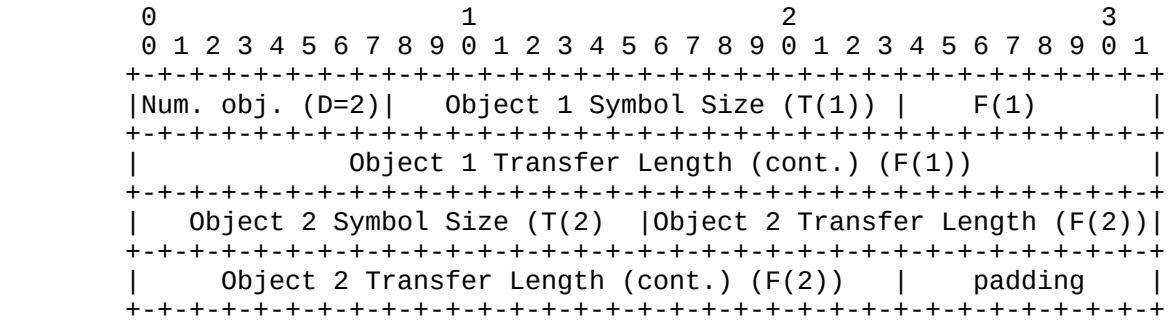

Figure 2: Encoded Common FEC OTI when D=2

3.2.4. Scheme-Specific

The following parameters are carried in the Scheme-Specific FEC OTI element for this FEC Scheme:

- A symbol alignment parameter (Al) (8 bits, unsigned integer)
- For I = 1,...,D, the Scheme-Specific FEC Object Transmission information elements specific to object I are:
  - The number of source blocks for object I (Z(I)) (12 bits, unsigned integer)
  - The number of sub-blocks for object I (N(I)) (12 bits, unsigned integer)

These parameters are all positive integers. The encoded Scheme-specific OTI is a (1+3*D)-octet field. An example for D=2 is shown in Figure 3.

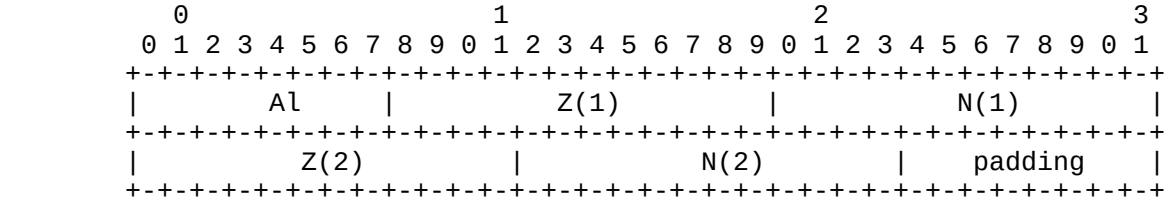

Figure 3: Encoded Scheme-specific FEC OTI when D=2

The encoded FEC OTI is a (2+10*D)-octet field consisting of the concatenation of the encoded Common FEC OTI and the encoded Scheme-specific FEC OTI.

4. Procedures

4.1. Introduction

All notation used in the document has exactly the same interpretation and meaning as in RaptorQ FEC Scheme [I-D.ietf-rmt-bb-fec-raptorq].

4.2. Content Delivery Protocol Requirements

This section describes the information exchange between the UOD-RaptorQ FEC Scheme and any Content Delivery Protocol (CDP) that makes use of the UOD-RaptorQ FEC Scheme for object delivery.

The UOD-RaptorQ encoder scheme and UOD-RaptorQ decoder scheme for object delivery require the following information from the CDP:

- The number of objects, D, to be transferred
- A symbol alignment parameter, Al
- For I = 1,...,D
  - The transfer length of object I, F(I), in octets
  - The symbol size to use for object I, T(I), in octets, which MUST be a multiple of Al
  - The number of source blocks for object I, Z(I)
  - The number of sub-blocks in each source block for object I, N(I)

The UOD-RaptorQ encoder scheme for object delivery additionally requires:

- For I = 1,...,D, the object I to be encoded consisting of F(I) octets

The UOD-RaptorQ encoder scheme supplies the CDP with the following information for each packet to be sent:

- Universal Object Symbol Identifier (UOSI)
- For I = 1,...,D, the encoding symbol(s) for object I

The CDP MUST communicate this information to the receiver.

4.3. Example Parameter Derivation Algorithm

For each object I, it is RECOMMENDED that the example parameter derivation algorithm described in Section 4.3 of the RaptorQ FEC Scheme [I-D.ietf-rmt-bb-fec-raptorq] be used, applied independently to each of the D objects.

The following are RECOMMENDATIONS:

- Al = 4 octets
- SS = 8 (which implies that the each sub-symbol will be at least SS*Al = 32 octets)
- For each I = 1,...,D, the size of the encoding symbol T(I) for object I is RECOMMENDED to be least SS*Al octets. (As stated earlier, T(I) MUST be a multiple of Al.)
- The payload size of each encoding packet is RECOMMENDED to be of size at least T, where T is the sum over I = 1,...,D of T(I).

4.4. Object Delivery

4.4.1. Source block construction

Exactly the same procedures as specified in Section 4.4.1 of the RaptorQ FEC Scheme [I-D.ietf-rmt-bb-fec-raptorq] are to be applied independently to each of the D objects.

4.4.2. Encoding packet construction

Each encoding packet contains the following information:

- Universal Object Symbol Identifier (UOSI)
- For I = 1,...,D, the encoding symbol(s) for object I

The following mappings provide the translations between the UOSI format used by the UOD-RaptorQ FEC Scheme and the (SBN, ESI) format of the FEC Payload ID used by the RaptorQ FEC Scheme [I-D.ietf-rmt-bb-fec-raptorq].

For each object I = 1,...,D, the mapping from a UOSI value C to the corresponding (SBN, ESI) values (A,B) for object I is:

- B = floor(C/Z(I))

o A = C - B*Z(I)

Similarly, for each object I = 1,...,D, the mapping from (SBN, ESI) values (A,B) for object I to a corresponding UOSI value C is:

o C = A + B*Z(I)

For each object I = 1,...,D, UOSI values from 0 to Kt(I)-1 identify the source symbols of object I in source block interleaved order, wherein Kt(I) = ceil(F(I)/T(I)). UOSI values from Kt(I) onwards identify repair symbols generated from the source symbols of object I using the RaptorQ encoder.

Each encoding packet may contain source symbols and repair symbols for objects. For each object I = 1,...,D, a packet may contain multiple encoding symbols generated from the same source block of object I, which may be a mixture of source and repair symbols. If the last source symbol of a source block includes padding octets added for FEC encoding purposes, then these octets MUST be included in any packet carrying this source symbol. Thus, each packet MUST contain only full-sized encoding symbols generated from the objects.

The UOSI, C, carried in each encoding packet is the UOSI of the first encoding symbol for each object carried in that packet. The subsequent encoding symbols in the packet for each object have UOSIs, C+1 to C+G-1, in sequential order, where G is the number of encoding symbols for each object in the packet.

It is RECOMMENDED that each encoding packet contain exactly one encoded symbol for each of the D objects.

It is RECOMMENDED that encoding packets be generated and sent according to the following procedure:

o For each UOSI value C = 0, 1, 2, 3, ..., generate and send an encoding packet as follows:

  * Set the value of the FEC Payload ID of the encoding packet to the UOSI value C.

  * For I = 1,...,D,

    + Determine the (SBN, ESI) values (A(I), B(I)) that correspond to UOSI value C.

    + Generate the encoding symbol E(I) of size T(I) that corresponds to (SBN, ESI) values (A(I), B(I)) from object I according to the procedures of the RaptorQ FEC Scheme

[I-D.ietf-rmt-bb-fec-raptorq].

+ Add encoding symbol E(I) to the payload of the encoding packet.

* Send the encoding packet.

Note that it is not necessary for the receiver to know the total number of encoding packets.

5. Security Considerations

Exactly the same as for the RaptorQ FEC Scheme [I-D.ietf-rmt-bb-fec-raptorq] apply.

6. IANA Considerations

Values of FEC Encoding IDs and FEC Instance IDs are subject to IANA registration. For general guidelines on IANA considerations as they apply to this document, see [RFC5052]. This document assigns the Fully-Specified FEC Encoding ID 7 (tbd) under the ietf:rmt:fec: encoding name-space to "UOD-RaptorQ Code".

7. References

7.1. Normative References

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

[I-D.ietf-rmt-bb-fec-raptorq] Luby, M., Shokrollahi, A., Watson, M., Stockhammer, T., and L. Minder, "RaptorQ Forward Error Correction Scheme for Object Delivery", draft-ietf-rmt-bb-fec-raptorq-04 (work in progress), August 2010.

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

7.2. Informative References

[PET] Albanese, Blomer, Edmonds, Luby, and Sudan, "Priority Encoding Transmission", IEEE Transactions on Information Theory, November 1996.

Authors' Addresses

Michael Luby
Qualcomm Incorporated
3165 Kifer Road
Santa Clara, CA 95051
U.S.A.

Email: luby@qualcomm.com

Thomas Stockhammer
Nomor Research
Brecherspitzstrasse 8
Munich 81541
Germany

Email: stockhammer@nomor.de