Sliding Window Random Linear Code (RLC) Forward Erasure Correction (FEC) Schemes for FECFRAME

Vincent Roca, Belkacem Teibi

TSVWG
Internet-Draft
Intended status: Standards Track
Expires: September 5, 2018

V. Roca
B. Teibi
INRIA

March 4, 2018

Sliding Window Random Linear Code (RLC) Forward Erasure Correction (FEC) Schemes for FECFRAME

draft-ietf-tsvwg-rlc-fec-scheme-02

Abstract

This document describes two fully-specified FEC Schemes for Sliding Window Random Linear Codes (RLC), one for RLC over GF(2) (binary case), a second one for RLC over GF(2^^8), both of them with the possibility of controlling the code density. They are meant to protect arbitrary media streams along the lines defined by FECFRAME extended to sliding window FEC codes. These sliding window FEC codes rely on an encoding window that slides over the source symbols, generating new repair symbols whenever needed. Compared to block FEC codes, these sliding window FEC codes offer key advantages with real-time flows in terms of reduced FEC-related latency while often providing improved erasure recovery capabilities.

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 https://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 5, 2018.

Table of Contents

1. Introduction

Application-Level Forward Erasure Correction (AL-FEC) codes, or simply FEC codes, are a key element of communication systems. They are used to recover from packet losses (or erasures) during content delivery sessions to a large number of receivers (multicast/broadcast transmissions). This is the case with the FLUTE/ALC protocol [RFC6726] in case of reliable file transfers over lossy networks, and the FECFRAME protocol for reliable continuous media transfers over lossy networks.

The present document only focusses on the FECFRAME protocol, used in multicast/broadcast delivery mode, with contents that feature stringent real-time constraints: each source packet has a maximum validity period after which it will not be considered by the destination application.

1.1. Limits of Block Codes with Real-Time Flows

With FECFRAME, there is a single FEC encoding point (either a end-host/server (source) or a middlebox) and a single FEC decoding point (either a end-host (receiver) or middlebox). In this context, currently standardized AL-FEC codes for FECFRAME like Reed-Solomon [RFC6865], LDPC-Staircase [RFC6816], or Raptor/RaptorQ, are all linear block codes: they require the data flow to be segmented into blocks of a predefined maximum size.

Defining this block size requires to find an appropriate balance between robustness and decoding latency: the larger the block size, the higher the robustness (e.g., in front of long packet erasure bursts), but also the higher the maximum decoding latency (i.e., the maximum time required to recover an lost (erased) packet thanks to FEC protection). Therefore, with a multicast/broadcast session where different receivers experience different packet loss rates, the block

size should be chosen by considering the worst communication conditions one wants to support, but without exceeding the desired maximum decoding latency. This choice will impact all receivers.

1.2. Lower Latency and Better Protection of Real-Time Flows with the Sliding Window RLC Codes

This document introduces two fully-specified FEC Schemes that follow a totally different approach: the Sliding Window Random Linear Codes (RLC) over either Finite Field GF(2) or GF(8). These FEC Schemes are used to protect arbitrary media streams along the lines defined by FECFRAME extended to sliding window FEC codes [fecframe-ext]. These FEC Schemes are extremely efficient for instance with media that feature real-time constraints sent within a multicast/broadcast session.

The RLC codes belong to the broad class of sliding window AL-FEC codes (A.K.A. convolutional codes). The encoding process is based on an encoding window that slides over the set of source packets (in fact source symbols as we will see in Section 3.2), and which is either of fixed or variable size (elastic window). Repair packets (symbols) are generated and sent on-the-fly, after computing a random linear combination of the source symbols present in the current encoding window.

At the receiver, a linear system is managed from the set of received source and repair packets. New variables (representing source symbols) and equations (representing the linear combination of each repair symbol received) are added upon receiving new packets. Variables are removed when they are too old with respect to their validity period (real-time constraints), as well as the associated equations they are involved in (Appendix A introduces an optimization that extends the time a variable is considered in the system). Lost source symbols are then recovered thanks to this linear system whenever its rank permits it.

With RLC codes (more generally with sliding window codes), the protection of a multicast/broadcast session also needs to be dimensioned by considering the worst communication conditions one wants to support. However the receivers experiencing a good to medium communication quality will observe a FEC-related latency close to zero [Roca17] since an isolated lost source packet is quickly recovered with the following repair packet. On the opposite, with a block code, recovering an isolated lost source packet always requires waiting the end of the block for the first repair packet to arrive. Additionally, under certain situations (e.g., with a limited FEC-related latency budget and with constant bit rate transmissions after FECFRAME encoding), sliding window codes achieve more easily a target

transmission quality (e.g., measured by the residual loss after FEC decoding) by sending fewer repair packets (i.e., higher code rate) than block codes.

1.3. Small Transmission Overheads with the Sliding Window RLC FEC Scheme

The Sliding Window RLC FEC Scheme is designed so as to reduce the transmission overhead. The main requirement is that each repair packet header must enable a receiver to reconstruct the set of source symbols plus the associated coefficients used during the encoding process. In order to minimize packet overhead, the set of source symbols in the encoding window as well as the set of coefficients over GF(2^^m) (where m is 1 or 8, depending on the FEC Scheme) used in the linear combination are not individually listed in the repair packet header. Instead, each FEC Repair Packet header contains:

- the Encoding Symbol Identifier (ESI) of the first source symbol in the encoding window as well as the number of symbols (since this number may vary with a variable size, elastic window). These two pieces of information enable each receiver to easily reconstruct the set of source symbols considered during encoding, the only constraint being that there cannot be any gap;
- the seed used by a coding coefficients generation function (Section 3.5). This information enables each receiver to generate the same set of coding coefficients over GF(2^^m) as the sender;

Therefore, no matter the number of source symbols present in the encoding window, each FEC Repair Packet features a fixed 64-bit long header, called Repair FEC Payload ID (Figure 7). Similarly, each FEC Source Packet features a fixed 32-bit long trailer, called Explicit Source FEC Payload ID (Figure 5), that contains the ESI of the first source symbol (see the ADUI and source symbol mapping, Section 3.2).

1.4. Document Organization

This fully-specified FEC Scheme follows the structure required by [RFC6363], section 5.6. "FEC Scheme Requirements", namely:

3. Procedures: This section describes procedures specific to this FEC Scheme, namely: RLC parameters derivation, ADUI and source symbols mapping, pseudo-random number generator, and coding coefficients generation function;
4. Formats and Codes: This section defines the Source FEC Payload ID and Repair FEC Payload ID formats, carrying the signalling information associated to each source or repair symbol. It also defines the FEC Framework Configuration Information (FFCI) carrying signalling information for the session;

5. FEC Code Specification: Finally this section provides the code specification.

2. Definitions and Abbreviations

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].

This document uses the following definitions and abbreviations:

GF(q) denotes a finite field (also known as the Galois Field) with q elements. We assume that q = 2^^m in this document
m defines the length of the elements in the finite field, in bits. In this document, m is equal to 1 or 8
ADU: Application Data Unit
ADUI: Application Data Unit Information (includes the F, L and padding fields in addition to the ADU)
E: size of an encoding symbol (i.e., source or repair symbol), assumed fixed (in bytes)
br_in: transmission bitrate at the input of the FECFRAME sender, assumed fixed (in bits/s)
br_out: transmission bitrate at the output of the FECFRAME sender, assumed fixed (in bits/s)
max_lat: maximum FEC-related latency within FECFRAME (in seconds)
cr: RLC coding rate, ratio between the total number of source symbols and the total number of source plus repair symbols
plr: packet loss rate on the packet erasure channel
ew_size: encoding window current size at a sender (in symbols)
ew_max_size: encoding window maximum size at a sender (in symbols)
dw_max_size: decoding window maximum size at a receiver (in symbols)
ls_max_size: linear system maximum size (or width) at a receiver (in symbols)
PRNG: pseudo-random number generator
pmms_rand(maxv): PRNG defined in Section 3.4 and used in this specification, that returns a new random integer in [0; maxv-1]
DT: coding coefficients density threshold, an integer between 0 and 15 (inclusive) the controls the fraction of coefficients that are non zero

3. Procedures

This section introduces the procedures that are used by this FEC Scheme.

3.1. Parameters Derivation

The Sliding Window RLC FEC Scheme relies on several key parameters:

Maximum FEC-related latency budget, max_lat (in seconds) A source ADU flow can have real-time constraints, and therefore any FECFRAME related operation must take place within the validity period of each ADU. When there are multiple flows with different real-time constraints, we consider the most stringent constraints (see [RFC6363], Section 10.2, item 6, for recommendations when several flows are globally protected). The maximum FEC-related latency budget, max_lat, accounts for all sources of latency added by FEC encoding (at a sender) and FEC decoding (at a receiver). Other sources of latency (e.g., added by network communications) are out of scope and must be considered separately (said differently, they have already been deducted from max_lat). max_lat can be regarded as the latency budget permitted for all FEC-related operations. This is an input parameter that enables to derive other internal parameters as explained below;

Encoding window current (resp. maximum) size, ew_size (resp. ew_max_size) (in symbols):
these parameters are used by a sender during FEC encoding. More precisely, each repair symbol is a linear combination of the ew_size source symbols present in the encoding window when RLC encoding took place. In all situations, we MUST have:

```
ew_size <= ew_max_size
```

Decoding window maximum size, dw_max_size (in symbols): at a receiver, this parameter determines the maximum size of the decoding window. Said differently, this is the maximum number of received or lost source symbols in the linear system (i.e., the variables) that are still within their latency budget. In situations where packets are sent with a fixed period, the dw_max_size parameter directly determines the maximum decoding latency experienced by the receiver, which necessarily needs to be in line with the maximum FEC-related latency budget. Note also that the optimization detailed in Appendix A can extend the linear system with additional old source symbols (that timed-out) beyond dw_max_size;

Symbol size, E (in bytes) and RLC code rate (cr): the E parameter determines the (source or repair) symbol sizes. The cr parameter determines the code rate, i.e., the amount of redundancy added to the flow (it is the ratio between the total number of source symbols and the total number of source plus repair symbols). These two parameters are input parameters that enable to derive other internal parameters as explained below. In practice they will usually be fixed, especially with multicast/broadcast transmissions. In specific use-cases, in particular with unicast

transmissions in presence of a feedback mechanism that estimates the communication quality (out-of-scope of FECFRAME), the code rate may be adjusted dynamically.

Let us assume that the encoding symbol size (E, in bytes) and code rate (cr) are constant. Let us also assume a constant transmission bitrate (br_out, in bits/s) at the output of the FECFRAME sender (as in [Roca17]). It means that the source flow bitrate needs to be adjusted according to the added repair flow overhead in order to keep the total transmission bitrate fixed and equal to br_out. In order to comply with the maximum FEC-related latency budget we need:

```
dw_max_size = (max_lat * br_out * cr) / (8 * E)
```

Sometimes the opposite can happen: the source flow bitrate at the input of the FECFRAME sender is fixed (br_in, in bits/s). It means that the transmission bitrate at the output of the FECFRAME sender will be higher, depending on the added repair flow overhead. In order to comply with the maximum FEC-related latency budget we need:

```
dw_max_size = (max_lat * br_in) / (8 * E)
```

Finally, there are situations where no such assumption can be made (e.g., with a variable bit rate input flow). In that case the encoding and decoding window maximum sizes may be initialized, based on the input flow features (e.g., the peak bitrate if it is known) and great care must be taken on timing aspects at a sender (see Section 3.3) and receiver. The details of how to manage these situations are use-case dependent and out of scope of this document.

Then, once the dw_max_size has been determined, the ew_max_size can be defined. For decoding to be possible, it is required that the encoding window maximum size be at most equal to the decoding window maximum size. It is often good practice to choose [Roca17]:

```
ew_max_size = dw_max_size * 0.75
```

However any value ew_max_size < dw_max_size can be used without impact on the FEC-related latency budget. Finding the optimal value will depend on the use-case details and should be determined after simulations or field trials. This is of course out of scope of this document.

Note that the decoding beyond maximum latency optimization (Appendix A) enables an old source symbol to be kept in the linear system beyond the FEC-related latency budget, but not delivered to the receiving application. In any case, the linear system maximum

size is greater than (with the decoding optimization) or equal to (without) the decoding window maximum size:

ls_max_size >= dw_max_size

3.2. ADU, ADUI and Source Symbols Mappings

An ADU, coming from the application, cannot be mapped to source symbols directly. Indeed, a lost ADU recovered at a receiver must contain enough information to be assigned to the right application flow (UDP port numbers and IP addresses cannot be used to that purpose as they are not protected by FEC encoding). This requires adding the flow identifier to each ADU before doing FEC encoding.

Additionally, since ADUs are of variable size, padding is needed so that each ADU (with its flow identifier) contribute to an integral number of source symbols. This requires adding the original ADU length to each ADU before doing FEC encoding. Because of these requirements, an intermediate format, the ADUI, or ADU Information, is considered [RFC6363].

For each incoming ADU, an ADUI is created as follows. First of all, 3 bytes are prepended (Figure 1):

Flow ID (F) (8-bit field): this unsigned byte contains the integer identifier associated to the source ADU flow to which this ADU belongs. It is assumed that a single byte is sufficient, which implies that no more than 256 flows will be protected by a single FECFRAME instance.

Length (L) (16-bit field): this unsigned integer contains the length of this ADU, in network byte order (i.e., big endian). This length is for the ADU itself and does not include the F, L, or Pad fields.

Then, zero padding is added to the ADU if needed:

Padding (Pad) (variable size field): this field contains zero padding to align the F, L, ADU and padding up to a size that is multiple of E bytes (i.e., the source and repair symbol length).

The data unit resulting from the ADU and the F, L, and Pad fields is called ADUI. Since ADUs can have different sizes, this is also the case for ADUIs. However an ADUI always contributes to an integral number of source symbols.

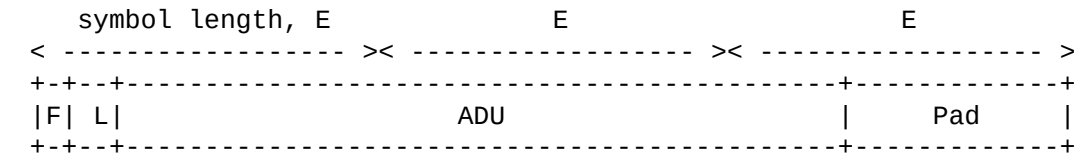

Figure 1: ADUI Creation example (here 3 source symbols are created for this ADUI).

Note that neither the initial 3 bytes nor the optional padding are sent over the network. However, they are considered during FEC encoding, and a receiver who lost a certain FEC Source Packet (e.g., the UDP datagram containing this FEC Source Packet) will be able to recover the ADUI if FEC decoding succeeds. Thanks to the initial 3 bytes, this receiver will get rid of the padding (if any) and identify the corresponding ADU flow.

3.3. Encoding Window Management

Source symbols and the corresponding ADUs are removed from the encoding window:

- when the sliding encoding window has reached its maximum size, ew_max_size. In that case the oldest symbol MUST be removed before adding a new symbol, so that the current encoding window size always remains inferior or equal to the maximum size: ew_size <= ew_max_size;
- when an ADU has reached its maximum validity duration in case of a real-time flow. When this happens, all source symbols corresponding to the ADUI that expired SHOULD be removed from the encoding window;

Source symbols are added to the sliding encoding window each time a new ADU arrives, once the ADU to source symbols mapping has been performed (Section 3.2). The current size of the encoding window, ew_size, is updated after adding new source symbols. This process may require to remove old source symbols so that: ew_size <= ew_max_size.

Note that a FEC codec may feature practical limits in the number of source symbols in the encoding window (e.g., for computational complexity reasons). This factor may further limit the ew_max_size value, in addition to the maximum FEC-related latency budget (Section 3.1).

3.4. Pseudo-Random Number Generator

The RLC codes rely on the following Pseudo-Random Number Generator (PRNG), identical to the PRNG used with LDPC-Staircase codes ([RFC5170], section 5.7).

The Park-Miler "minimal standard" PRNG [PM88] MUST be used. It defines a simple multiplicative congruential algorithm: Ij+1 = A * Ij (modulo M), with the following choices: A = 7^^5 = 16807 and M = 2^^31 - 1 = 2147483647. A validation criteria of such a PRNG is the following: if seed = 1, then the 10,000th value returned MUST be equal to 1043618065.

Several implementations of this PRNG are known and discussed in the literature. An optimized implementation of this algorithm, using only 32-bit mathematics, and which does not require any division, can be found in [rand31pmc]. It uses the Park and Miller algorithm [PM88] with the optimization suggested by D. Carta in [CA90]. The history behind this algorithm is detailed in [WI08]. Yet, any other implementation of the PRNG algorithm that matches the above validation criteria, like the ones detailed in [PM88], is appropriate.

This PRNG produces, natively, a 31-bit value between 1 and 0x7FFFFFFE (2^^31-2) inclusive. Since it is desired to scale the pseudo-random number between 0 and maxv-1 inclusive, one must keep the most significant bits of the value returned by the PRNG (the least significant bits are known to be less random, and modulo-based solutions should be avoided [PTVF92]). The following algorithm MUST be used:

Input:

raw_value: random integer generated by the inner PRNG algorithm, between 1 and 0x7FFFFFFE (2^^31-2) inclusive.
maxv: upper bound used during the scaling operation.

Output:

scaled_value: random integer between 0 and maxv-1 inclusive.

Algorithm:

```
scaled_value = (unsigned long) ((double)maxv * (double)raw_value /
(double)0x7FFFFFFF);
(NB: the above C type casting to unsigned long is equivalent to
using floor() with positive floating point values.)
```

In this document, pmms_rand(maxv) denotes the PRNG function that implements the Park-Miller "minimal standard" algorithm, defined above, and that scales the raw value between 0 and maxv-1 inclusive, using the above scaling algorithm.

Additionally, the pmms_srand(seed) function must be provided to enable the initialization of the PRNG with a seed before calling pmms_rand(maxv) the first time. The seed is a 31-bit integer between 1 and 0x7FFFFFFE inclusive. In this specification, the seed is restricted to a value between 1 and 0xFFFF inclusive, as this is the Repair_Key 16-bit field value of the Repair FEC Payload ID (Section 4.1.3).

3.5. Coding Coefficients Generation Function

The coding coefficients, used during the encoding process, are generated at the RLC encoder by the generate_coding_coefficients() function each time a new repair symbol needs to be produced. The fraction of coefficients that are non zero (i.e., the density) is controlled by the DT (Density Threshold) parameter. When DT equals 15, the maximum value, the function guaranties that all coefficients are non zero (i.e., maximum density). When DT is between 0 (minimum value) and strictly inferior to 15, the average probability of having a non zero coefficient equals (DT +1) / 16.

These considerations apply both the RLC over GF(2) and RLC over GF(2^^8), the only difference being the value of the m parameter. With the RLC over GF(2) FEC Scheme (Section 5), m MUST be equal to 1. With RLC over GF(2^^8) FEC Scheme (Section 4), m MUST be equal to 8.

```
<CODE BEGINS>
/*
 * Fills in the table of coding coefficients (of the right size)
 * provided with the appropriate number of coding coefficients to
 * use for the repair symbol key provided.
 *
 * (in) repair_key    key associated to this repair symbol. This
 *                    parameter is ignored (useless) if m=2 and dt=15
 * (in) cc_tab[]      pointer to a table of the right size to store
 *                    coding coefficients. All coefficients are
 *                    stored as bytes, regardless of the m parameter,
 *                    upon return of this function.
 * (in) cc_nb         number of entries in the table. This value is
 *                    equal to the current encoding window size.
 * (in) dt            integer between 0 and 15 (inclusive) that
 *                    controls the density. With value 15, all
 *                    coefficients are guaranteed to be non zero
 *                    (i.e. equal to 1 with GF(2) and equal to a
```

```
     *                    value in {1,... 255} with GF(2^^8)), otherwise
     *                    a fraction of them will be 0.
     * (in) m             Finite Field GF(2^^m) parameter. In this
     *                    document only values 1 and 8 are considered.
     * (out)              returns an error code
     */
   int generate_coding_coefficients (UINT16    repair_key,
                                     UINT8     cc_tab[],
                                     UINT16    cc_nb,
                                     UINT8     dt,
                                     UINT8     m)
   {
       UINT32    i;

       if (dt > 15) {
           return SOMETHING_WENT_WRONG; /* bad dt parameter */
       }
       if (repair_key == 0 && dt != 15 && m != 2) {
           return SOMETHING_WENT_WRONG; /* bad repair_key parameter */
       }
       switch (m) {
       case 1:
           if (dt == 15) {
               /* all coefficients are 1 */
               memset(cc_tab, 1, cc_nb);
           } else {
               /* here coefficients are either 0 or 1 */
               pmms_srand(repair_key);
               for (i = 0 ; i < cc_nb ; i++) {
                   if (pmms_rand(16) <= dt) {
                       cc_tab[i] = (UINT8) 1;
                   } else {
                       cc_tab[i] = (UINT8) 0;
                   }
               }
           }
           break;

       case 8:
           pmms_srand(repair_key);
           if (dt == 15) {
               /* coefficient 0 is avoided here in order to include
                * all the source symbols */
               for (i = 0 ; i < cc_nb ; i++) {
                   do {
                       cc_tab[i] = (UINT8) pmms_rand(256);
                   } while (cc_tab[i] == 0);
               }
```

```
            } else {
                /* here a certain fraction of coefficients should be 0 */
                for (i = 0 ; i < cc_nb ; i++) {
                    if (pmms_rand(16) <= dt) {
                        do {
                            cc_tab[i] = (UINT8) pmms_rand(256);
                        } while (cc_tab[i] == 0);
                    } else {
                        cc_tab[i] = 0;
                    }
                }
            }
            break;

        default:
            /* bad parameter m */
            return SOMETHING_WENT_WRONG;
        }
        return EVERYTHING_IS_OKAY;
    }
    <CODE ENDS>
```

Figure 2: Coding Coefficients Generation Function pseudo-code

4. Sliding Window RLC FEC Scheme over GF(2^^8) for Arbitrary ADU Flows

This fully-specified FEC Scheme defines the Sliding Window Random Linear Codes (RLC) over GF(2^^8).

4.1. Formats and Codes

4.1.1. FEC Framework Configuration Information

The FEC Framework Configuration Information (or FFCI) includes information that MUST be communicated between the sender and receiver(s). More specifically, it enables the synchronization of the FECFRAME sender and receiver instances. It includes both mandatory elements and scheme-specific elements, as detailed below.

4.1.1.1. Mandatory Information

- FEC Encoding ID: the value assigned to this fully specified FEC Scheme MUST be XXXX, as assigned by IANA (Section 10).

When SDP is used to communicate the FFCI, this FEC Encoding ID is carried in the 'encoding-id' parameter.

4.1.1.2. FEC Scheme-Specific Information

The FEC Scheme-Specific Information (FSSI) includes elements that are specific to the present FEC Scheme. More precisely:

Encoding symbol size (E): a non-negative integer that indicates the size of each encoding symbol in bytes;

This element is required both by the sender (RLC encoder) and the receiver(s) (RLC decoder).

When SDP is used to communicate the FFCI, this FEC Scheme-specific information is carried in the 'fssi' parameter in textual representation as specified in [RFC6364]. For instance:

fssi=E:1400

If another mechanism requires the FSSI to be carried as an opaque octet string (for instance, after a Base64 encoding), the encoding format consists of the following 2 octets:

Encoding symbol length (E): 16-bit field.

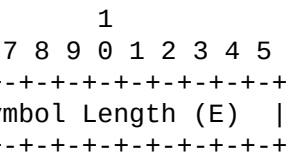

Figure 3: FSSI Encoding Format

4.1.2. Explicit Source FEC Payload ID

A FEC Source Packet MUST contain an Explicit Source FEC Payload ID that is appended to the end of the packet as illustrated in Figure 4.

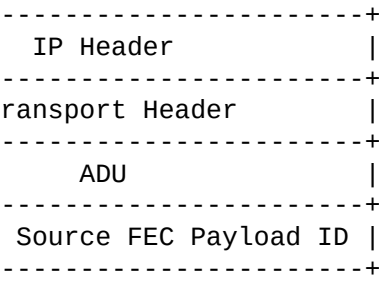

Figure 4: Structure of an FEC Source Packet with the Explicit Source FEC Payload ID

More precisely, the Explicit Source FEC Payload ID is composed of the following field (Figure 5):

Encoding Symbol ID (ESI) (32-bit field): this unsigned integer identifies the first source symbol of the ADUI corresponding to this FEC Source Packet. The ESI is incremented for each new source symbol, and after reaching the maximum value (2^32-1), wrapping to zero occurs.

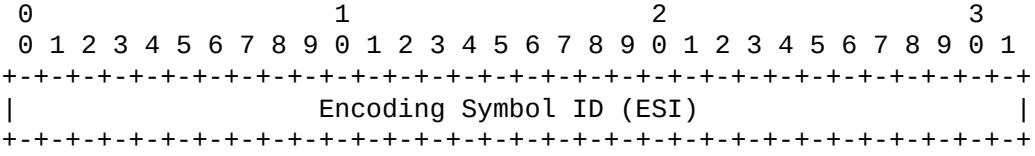

Figure 5: Source FEC Payload ID Encoding Format

4.1.3. Repair FEC Payload ID

A FEC Repair Packet can contain one or more repair symbols. When there are several repair symbols, all of them MUST have been generated from the same encoding window, using Repair_Key values that are managed as explained below. A receiver can easily deduce the number of repair symbols within a FEC Repair Packet by comparing the received FEC Repair Packet size (equal to the UDP payload size when UDP is the underlying transport protocol) and the symbol size, E, communicated in the FFCI.

A FEC Repair Packet MUST contain a Repair FEC Payload ID that is prepended to the repair symbol as illustrated in Figure 6.

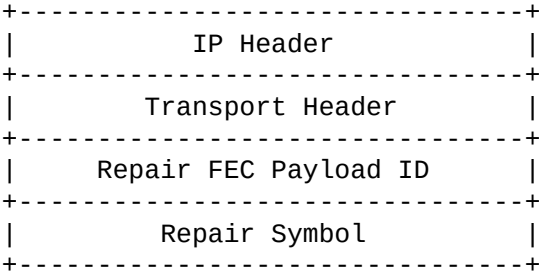

Figure 6: Structure of an FEC Repair Packet with the Repair FEC Payload ID

More precisely, the Repair FEC Payload ID is composed of the following fields (Figure 7):

Repair_Key (16-bit field): this unsigned integer is used as a seed by the coefficient generation function (Section 3.5) in order to generate the desired number of coding coefficients. Value 0 MUST NOT be used. When a FEC Repair Packet contains several repair symbols, this repair key value is that of the first repair symbol. The remaining repair keys can be deduced by incrementing by 1 this value, up to a maximum value of 65535 after which it loops back to 1 (note that 0 is not a valid value).

Density Threshold for the coding coefficients, DT (4-bit field): this unsigned integer carried the Density Threshold (DT) used by the coding coefficient generation function Section 3.5. More precisely, it controls the probability of having a non zero coding coefficient, which equals (DT+1) / 16. When a FEC Repair Packet contains several repair symbols, the DT value applies to all of them;

Number of Source Symbols in the encoding window, NSS (12-bit field):

this unsigned integer indicates the number of source symbols in the encoding window when this repair symbol was generated. When a FEC Repair Packet contains several repair symbols, this NSS value applies to all of them;

ESI of First Source Symbol in the encoding window, FSS_ESI (32-bit field):

this unsigned integer indicates the ESI of the first source symbol in the encoding window when this repair symbol was generated. When a FEC Repair Packet contains several repair symbols, this FSS_ESI value applies to all of them;

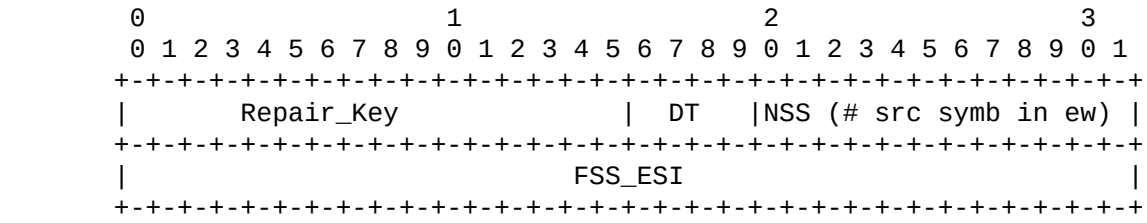

Figure 7: Repair FEC Payload ID Encoding Format

4.1.4. Additional Procedures

The following procedure applies:

- The ESI of source symbols MUST start with value 0 for the first source symbol and MUST be managed sequentially. Wrapping to zero happens after reaching the maximum 32-bit value.

5. Sliding Window RLC FEC Scheme over GF(2) for Arbitrary ADU Flows

This fully-specified FEC Scheme defines the Sliding Window Random Linear Codes (RLC) over GF(2) (binary case).

5.1. Formats and Codes

5.1.1. FEC Framework Configuration Information

5.1.1.1. Mandatory Information

- FEC Encoding ID: the value assigned to this fully specified FEC Scheme MUST be YYYY, as assigned by IANA (Section 10).

When SDP is used to communicate the FFCI, this FEC Encoding ID is carried in the 'encoding-id' parameter.

5.1.1.2. FEC Scheme-Specific Information

All the considerations of Section 4.1.1.2 apply here.

5.1.2. Explicit Source FEC Payload ID

All the considerations of Section 4.1.1.2 apply here.

5.1.3. Repair FEC Payload ID

All the considerations of Section 4.1.1.2 apply here, with the only exception that the Repair_Key field is useless if DT = 15 (indeed, in that case all the coefficients are necessarily equal to 1 and the coefficient generation function does not use any PRNG). When DT = 15 it is RECOMMENDED that the sender use value 0 for the Repair_Key field, but a receiver SHALL ignore this field.

5.1.4. Additional Procedures

All the considerations of Section 4.1.1.2 apply here.

6. FEC Code Specification

6.1. Encoding Side

This section provides a high level description of a Sliding Window RLC encoder.

Whenever a new FEC Repair Packet is needed, the RLC encoder instance first gathers the ew_size source symbols currently in the sliding encoding window. Then it chooses a repair key, which can be a non

zero monotonically increasing integer value, incremented for each repair symbol up to a maximum value of 65535 (as it is carried within a 16-bit field) after which it loops back to 1 (indeed, being used as a PRNG seed, value 0 is prohibited). This repair key is communicated to the coefficient generation function (Section Section 3.5) in order to generate ew_size coding coefficients. Finally, the FECFRAME sender computes the repair symbol as a linear combination of the ew_size source symbols using the ew_size coding coefficients. When E is small and when there is an incentive to pack several repair symbols within the same FEC Repair Packet, the appropriate number of repair symbols are computed. The only constraint is to increment by 1 the repair key for each of them, keeping the same ew_size source symbols, since only the first repair key will be carried in the Repair FEC Payload ID. The FEC Repair Packet can then be sent. The source versus repair FEC packet transmission order is out of scope of this document and several approaches exist that are implementation specific.

Other solutions are possible to select a repair key value when a new FEC Repair Packet is needed, for instance by choosing a random integer between 1 and 65535. However, selecting the same repair key as before (which may happen in case of a random process) is only meaningful if the encoding window has changed, otherwise the same FEC Repair Packet will be generated.

6.2. Decoding Side

This section provides a high level description of a Sliding Window RLC decoder.

A FECFRAME receiver needs to maintain a linear system whose variables are the received and lost source symbols. Upon receiving a FEC Repair Packet, a receiver first extracts all the repair symbols it contains (in case several repair symbols are packed together). For each repair symbol, when at least one of the corresponding source symbols it protects has been lost, the receiver adds an equation to the linear system (or no equation if this repair packet does not change the linear system rank). This equation of course re-uses the ew_size coding coefficients that are computed by the same coefficient generation function (Section Section 3.5), using the repair key and encoding window descriptions carried in the Repair FEC Payload ID. Whenever possible (i.e., when a sub-system covering one or more lost source symbols is of full rank), decoding is performed in order to recover lost source symbols. Each time an ADUI can be totally recovered, padding is removed (thanks to the Length field, L, of the ADUI) and the ADU is assigned to the corresponding application flow (thanks to the Flow ID field, F, of the ADUI). This ADU is finally passed to the corresponding upper application. Received FEC Source

Packets, containing an ADU, can be passed to the application either immediately or after some time to guaranty an ordered delivery to the application. This document does not mandate any approach as this is an operational and management decision.

With real-time flows, a lost ADU that is decoded after the maximum latency or an ADU received after this delay should not be passed to the application. Instead the associated source symbols should be removed from the linear system maintained by the receiver(s). Appendix A discusses a backward compatible optimization whereby those late source symbols may still be used in order to improve the global robustness.

7. Implementation Status

Editor's notes: RFC Editor, please remove this section motivated by RFC 6982 before publishing the RFC. Thanks.

An implementation of the Sliding Window RLC FEC Scheme for FECFRAME exists:

- Organisation: Inria
- Description: This is an implementation of the Sliding Window RLC FEC Scheme limited to GF(2^^8). It relies on a modified version of our OpenFEC (http://openfec.org) FEC code library. It is integrated in our FECFRAME software (see [fecframe-ext]).
- Maturity: prototype.
- Coverage: this software complies with the Sliding Window RLC FEC Scheme.
- Lincensing: proprietary.
- Contact: vincent.roca@inria.fr

8. Security Considerations

The FEC Framework document [RFC6363] provides a comprehensive analysis of security considerations applicable to FEC Schemes. Therefore, the present section follows the security considerations section of [RFC6363] and only discusses specific topics.

8.1. Attacks Against the Data Flow

8.1.1. Access to Confidential Content

The Sliding Window RLC FEC Scheme specified in this document does not change the recommendations of [RFC6363]. To summarize, if confidentiality is a concern, it is RECOMMENDED that one of the solutions mentioned in [RFC6363] is used with special considerations to the way this solution is applied (e.g., is encryption applied

before or after FEC protection, within the end-system or in a middlebox) to the operational constraints (e.g., performing FEC decoding in a protected environment may be complicated or even impossible) and to the threat model.

8.1.2. Content Corruption

The Sliding Window RLC FEC Scheme specified in this document does not change the recommendations of [RFC6363]. To summarize, it is RECOMMENDED that one of the solutions mentioned in [RFC6363] is used on both the FEC Source and Repair Packets.

8.2. Attacks Against the FEC Parameters

The FEC Scheme specified in this document defines parameters that can be the basis of attacks. More specifically, the following parameters of the FFCI may be modified by an attacker who targets receivers (Section 4.1.1.2):

- FEC Encoding ID: changing this parameter leads the receivers to consider a different FEC Scheme, which enables an attacker to create a Denial of Service (DoS);
- Encoding symbol length (E): setting this E parameter to a different value will confuse the receivers and create a DoS. More precisely, the FEC Repair Packets received will probably no longer be multiple of E, leading receivers to reject them;

It is therefore RECOMMENDED that security measures are taken to guarantee the FFCI integrity, as specified in [RFC6363]. How to achieve this depends on the way the FFCI is communicated from the sender to the receiver, which is not specified in this document.

Similarly, attacks are possible against the Explicit Source FEC Payload ID and Repair FEC Payload ID: by modifying the Encoding Symbol ID (ESI), or the repair key, NSS or FSS_ESI. It is therefore RECOMMENDED that security measures are taken to guarantee the FEC Source and Repair Packets as stated in [RFC6363].

8.3. When Several Source Flows are to be Protected Together

The Sliding Window RLC FEC Scheme specified in this document does not change the recommendations of [RFC6363].

8.4. Baseline Secure FEC Framework Operation

The Sliding Window RLC FEC Scheme specified in this document does not change the recommendations of [RFC6363] concerning the use of the IPsec/ESP security protocol as a mandatory to implement (but not

mandatory to use) security scheme. This is well suited to situations where the only insecure domain is the one over which the FEC Framework operates.

9. Operations and Management Considerations

The FEC Framework document [RFC6363] provides a comprehensive analysis of operations and management considerations applicable to FEC Schemes. Therefore, the present section only discusses specific topics.

9.1. Operational Recommendations: Finite Field GF(2) Versus GF(2^^8)

The present document specifies two FEC Schemes that differ on the Finite Field used for the coding coefficients. It is expected that the RLC over GF(2^^8) FEC Scheme will be mostly used since it warrants a higher packet loss protection. In case of small encoding windows, the associated processing overhead is not an issue (e.g., we measured decoding speeds between 745 Mbps and 2.8 Gbps on an ARM Cortex-A15 embedded board in [Roca17]). Of course the CPU overhead will increase with the encoding window size, because more operations in the GF(2^^8) finite field will be needed.

The RLC over GF(2) FEC Scheme offers an alternative. In that case operations symbols can be directly XOR-ed together which warrants high bitrate encoding and decoding operations, and can be an advantage with large encoding windows. However packet loss protection is significantly reduced by using this FEC Scheme.

9.2. Operational Recommendations: Coding Coefficients Density Threshold

In addition to the choice of the Finite Field, the two FEC Schemes define a coding coefficient density threshold (DT) parameter. This parameter enables a sender to control the code density, i.e., the proportion of coefficients that are non zero on average. With RLC over GF(2^^8), it is usually appropriate that small encoding windows be associated to a density threshold equal to 15, the maximum value, in order to warrant a high loss protection.

On the opposite, with larger encoding windows, it is usually appropriate that the density threshold be reduced. With large encoding windows, an alternative can be to use RLC over GF(2) and a density threshold equal to 7 (i.e., an average density equal to 1/2) or smaller.

Note that using a density threshold equal to 15 with RLC over GF(2) is equivalent to using an XOR code that compute the XOR sum of all the source symbols in the encoding window. In that case: (1) a

single repair symbol can be produced for any encoding window, and (2) the repair_key parameter becomes useless (the coding coefficients generation function does not rely on the PRNG).

10. IANA Considerations

This document registers two values in the "FEC Framework (FECFRAME) FEC Encoding IDs" registry [RFC6363] as follows:

- YYYY refers to the Sliding Window Random Linear Codes (RLC) over GF(2) FEC Scheme for Arbitrary Packet Flows, as defined in Section 5 of this document.
- XXXX refers to the Sliding Window Random Linear Codes (RLC) over GF(2^^8) FEC Scheme for Arbitrary Packet Flows, as defined in Section 4 of this document.

11. Acknowledgments

The authors would like to thank Marie-Jose Montpetit for her valuable feedbacks on this document.

12. References

12.1. Normative References

[fecframe-ext] Roca, V. and A. Begen, "Forward Error Correction (FEC) Framework Extension to Sliding Window Codes", Transport Area Working Group (TSVWG) draft-ietf-tsvwg-fecframe-ext (Work in Progress), March 2018, <https://tools.ietf.org/html/draft-ietf-tsvwg-fecframe-ext>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC6363] Watson, M., Begen, A., and V. Roca, "Forward Error Correction (FEC) Framework", RFC 6363, DOI 10.17487/RFC6363, October 2011, <https://www.rfc-editor.org/info/rfc6363>.

[RFC6364] Begen, A., "Session Description Protocol Elements for the Forward Error Correction (FEC) Framework", RFC 6364, DOI 10.17487/RFC6364, October 2011, <https://www.rfc-editor.org/info/rfc6364>.

12.2. Informative References

[CA90] Carta, D., "Two Fast Implementations of the Minimal Standard Random Number Generator", Communications of the ACM, Vol. 33, No. 1, pp.87-88, January 1990.

[PM88] Park, S. and K. Miller, "Random Number Generators: Good Ones are Hard to Find", Communications of the ACM, Vol. 31, No. 10, pp.1192-1201, 1988.

[PTVF92] Press, W., Teukolsky, S., Vetterling, W., and B. Flannery, "Numerical Recipies in C; Second Edition", Cambridge University Press, ISBN: 0-521-43108-5, 1992.

[rand31pmc] Whittle, R., "31 bit pseudo-random number generator", September 2005, <http://www.firstpr.com.au/dsp/rand31/rand31-park-miller-carta.cc.txt>.

[RFC5170] Roca, V., Neumann, C., and D. Furodet, "Low Density Parity Check (LDPC) Staircase and Triangle Forward Error Correction (FEC) Schemes", RFC 5170, DOI 10.17487/RFC5170, June 2008, <https://www.rfc-editor.org/info/rfc5170>.

[RFC6726] Paila, T., Walsh, R., Luby, M., Roca, V., and R. Lehtonen, "FLUTE - File Delivery over Unidirectional Transport", RFC 6726, DOI 10.17487/RFC6726, November 2012, <https://www.rfc-editor.org/info/rfc6726>.

[RFC6816] Roca, V., Cunche, M., and J. Lacan, "Simple Low-Density Parity Check (LDPC) Staircase Forward Error Correction (FEC) Scheme for FECFRAME", RFC 6816, DOI 10.17487/RFC6816, December 2012, <https://www.rfc-editor.org/info/rfc6816>.

[RFC6865] Roca, V., Cunche, M., Lacan, J., Bouabdallah, A., and K. Matsuzono, "Simple Reed-Solomon Forward Error Correction (FEC) Scheme for FECFRAME", RFC 6865, DOI 10.17487/RFC6865, February 2013, <https://www.rfc-editor.org/info/rfc6865>.

[Roca16] Roca, V., Teibi, B., Burdinat, C., Tran, T., and C. Thienot, "Block or Convolutional AL-FEC Codes? A Performance Comparison for Robust Low-Latency Communications", HAL open-archive document,hal-01395937 https://hal.inria.fr/hal-01395937/en/, November 2016, <https://hal.inria.fr/hal-01395937/en/>.

[Roca17] Roca, V., Teibi, B., Burdinat, C., Tran, T., and C. Thienot, "Less Latency and Better Protection with AL-FEC Sliding Window Codes: a Robust Multimedia CBR Broadcast Case Study", 13th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob17), October 2017 https://hal.inria.fr/hal-01571609v1/en/, October 2017, <https://hal.inria.fr/hal-01571609v1/en/>.

[WI08] Whittle, R., "Park-Miller-Carta Pseudo-Random Number Generator", http://www.firstpr.com.au/dsp/rand31/, January 2008, <http://www.firstpr.com.au/dsp/rand31/>.

Appendix A. Decoding Beyond Maximum Latency Optimization

This annex introduces non normative considerations. They are provided as suggestions, without any impact on interoperability. For more information see [Roca16].

It is possible to improve the decoding performance of sliding window codes without impacting maximum latency, at the cost of extra CPU overhead. The optimization consists, for a receiver, to extend the linear system beyond the decoding window, by keeping a certain number of old source symbols:

```
ls_max_size > dw_max_size
```

Usually the following choice is a good trade-off between decoding performance and extra CPU overhead:

```
ls_max_size = 2 * dw_max_size
```

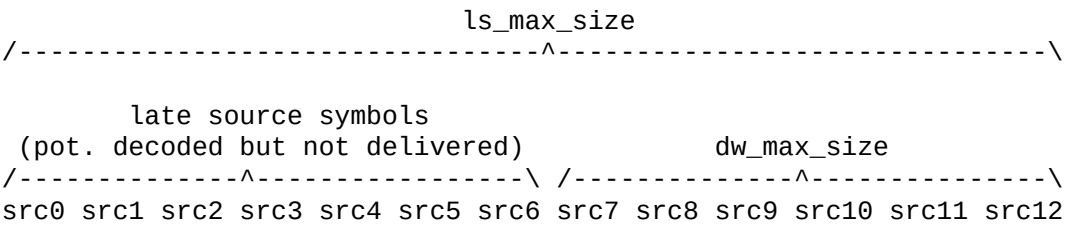

Figure 8: Relationship between parameters to decode beyond maximum latency.

It means that source symbols, and therefore ADUs, may be decoded even if the added latency exceeds the maximum value permitted by the application. It follows that the corresponding ADUs SHOULD NOT be delivered to the application and SHOULD be dropped once they are no longer needed. However, decoding these "late symbols" significantly improves the global robustness in bad reception conditions and is therefore recommended for receivers experiencing bad communication conditions [Roca16]. In any case whether or not to use this optimization and what exact value to use for the ls_max_size parameter are decisions made by each receiver independently, without any impact on the other receivers nor on the source.

Authors' Addresses

Vincent Roca
INRIA
Grenoble
France

EMail: vincent.roca@inria.fr

Belkacem Teibi
INRIA
Grenoble
France

EMail: belkacem.teibi@inria.fr