NWCRG
Internet-Draft
Intended status: Experimental
Expires: January 7, 2016

J. Detchart
E. Lochin
J. Lacan
ISAE
V. Roca
INRIA

July 6, 2015

# Tetrys, an On-the-Fly Network Coding protocol
# draft-detchart-nwcrg-tetrys-02

## Abstract

This document describes Tetrys, an On-The-Fly Network Coding (NC) protocol that can be used to transport delay and loss sensitive data over a lossy network. Tetrys can recover from erasures within a RTT-independent delay, thanks to the transmission of coded packets. It can be used for both unicast, multicast and anycast communications.

## 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 January 7, 2016.

Table of Contents

1. Introduction . . . . . 2
  1.1. Requirements Notation . . . . . 3
2. Definitions, Notations and Abbreviations . . . . . 3
3. Architecture . . . . . 4
  3.1. Use Cases . . . . . 4
  3.2. Overview . . . . . 5
4. Packet Format . . . . . 6
  4.1. Common Header Format . . . . . 6
    4.1.1. Header Extensions . . . . . 8
  4.2. Source Packet Format . . . . . 9
  4.3. Coded Packet Format . . . . . 10
  4.4. Acknowledgement Packet Format . . . . . 11
5. The Coding Coefficient Generator Identifiers . . . . . 12
  5.1. Definition . . . . . 12
  5.2. Table of Identifiers . . . . . 12
6. Tetrys Basic Functions . . . . . 12
  6.1. Encoding . . . . . 12
    6.1.1. Encoding Vector Formats . . . . . 13
  6.2. The Elastic Encoding Window . . . . . 16
  6.3. Recoding . . . . . 17
    6.3.1. Principle . . . . . 17
    6.3.2. Generating a coded symbol at an intermediate node . . 17
  6.4. Decoding . . . . . 17
7. Security Considerations . . . . . 17
8. Privacy Considerations . . . . . 17
9. IANA Considerations . . . . . 17
10. Acknowledgments . . . . . 18
11. References . . . . . 18
  11.1. Normative References . . . . . 18
  11.2. Informative References . . . . . 18
Authors' Addresses . . . . . 18

## 1. Introduction

This document describes Tetrys, a novel network coding protocol. Network codes were introduced in the early 2000s [AHL-00] to address the limitations of transmission over the Internet (delay, capacity and packet loss). While the use of network codes is fairly recent in the Internet community, the use of application layer erasure codes in the IETF has already been standardized in the RMT [RMT] and the FECFRAME [FECFRAME] working groups. The protocol presented here can be seen as a network coding extension to standards solutions. The

current proposal can be considered as a combination of network erasure coding and feedback mechanisms [Tetrys].

The main innovation of the Tetrys protocol is in the generation of coded packets from an elastic encoding window periodically updated with the receiver’s feedbacks. This update is done in such a way that any source packets coming from an input flow is included in the encoding window as long as it is not acknowledged or the encoding window did not reach a size limit. This mechanism allows for losses on both the forward and return paths and in particular is resilient to acknowledgement losses.

With Tetrys, a coded packet is a linear combination over a finite field of the data source packets belonging to the coding window. The choice of the finite field of the coefficients is a trade-off between the best performance (with non-binary coefficients) and the system constraints (binary codes in an energy constrained environment) and is driven by the application.

Thanks to the elastic encoding window, the coded packets are built on-the-fly, by using an algorithm or a function to choose the coefficients. The redundancy ratio can be dynamically adjusted, and the coefficients can be generated in different ways along a transmission. Compared to FEC block codes, this allows to reduce the bandwidth use and the decoding delay.

### 1.1. 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].

## 2. Definitions, Notations and Abbreviations

The terminology used in this document is presented below. It is aligned with the FECFRAME terminology as well as with recent activities in the Network Coding Research Group.

Source symbol: a symbol that has to be transmitted between the ingress and egress of the network.

Coded symbol: a linear combination over a finite field of a set of source symbols.

Source symbol ID: a sequence number to identify the source symbols.

Coded symbol ID: a sequence number to identify the coded symbols.

Encoding coefficients: elements of the finite field characterizing the linear combination used to generate a coded symbol.

Encoding vector: set of the encoding coefficients and input source symbol IDs.

Source packet: a source packet contains a source symbol with its associated IDs.

Coded packet: a coded packet contains a coded symbol, the coded symbol’s ID and encoding vector.

Input symbol: a symbol at the input of the Tetrys Encoding Building Block.

Output symbol: a symbol generated by the Tetrys Encoding Building Block. For a non systematic mode, all output symbols are coded symbols. For a systematic mode, output symbols can be the input symbols and a number of coded symbols that are linear combinations of the input symbols.

Feedback packet: a feedback packet is a packet containing information about the decoded or received source symbols. It can also bring additional information about the Packet Error Rate or the number of various packets in the receiver decoding window.

Elastic Encoding Window: an encoder-side buffer that stores all the non-acknowledged source packets of the input flow that are involved in the coding process.

Coding Coefficient Generator Identifier: a unique identifier that define a function or an algorithm allowing to generate the encoding vector.

Code rate: Define the rate between the number of input symbols and the number of output symbols.

## 3. Architecture

-- Editor’s note: The architecture used in this document should be aligned with the future NC Architecture document [NWCRG-ARCH]. --

### 3.1. Use Cases

Tetrys is well suited, but not limited to the use case where there is a single flow originated by a single source, with intra stream coding that takes place at a single encoding node. Note that the input stream can be a multiplex of several upper layer streams.

Transmission can be over a single path or multiple paths. In addition, the flow can be sent in unicast, multicast, or anycast mode.

## 3.2. Overview

```
+----------+                                                +----------+
|          |                                                |          |
|    App   |                                                |    App   |
|          |                                                |          |
+----------+                                                +----------+
     |                                                            ^
     |  source                                             source  |
     |  symbols                                            symbols |
     |                                                             |
     v                                                             |
+----------+                  +----------+                  +----------+
|          |  output packets  |          |  output packets  |          |
|  Tetrys  |----------------->|  Tetrys  |----------------->|  Tetrys  |
|  Encoder | feedback packets |  Recoder | feedback packets |  Decoder |
|          |<-----------------|          |<-----------------|          |
+----------+                  +----------+                  +----------+
```

Figure 1: Tetrys Architecture

The Tetrys protocol features several key functionalities:

- o On-the-fly encoding;
- o Recoding;
- o Decoding;
- o Signaling, to carry in particular the symbol identifiers in the encoding window and the associated coding coefficients when meaningful, in a manner that was previously used in FEC;
- o Feedback management;
- o Elastic window management;
- o Channel estimation;
- o Dynamic adjustment of the code rate and flow control;
- o Congestion control management (if appropriate);

  -- Editor’s note: must be discussed --

- o Tetrys packet header creation and processing;
- o -- Editor’s note: something else? --

These functionalities are provided by several building blocks:

- o The Tetrys Building Block: this BB is used during encoding, recoding and decoding processes. It must be noted that Tetrys does not mandate a specific building block. Instead any building block compatible with the elastic encoding window feature of Tetrys can be used.
- o The Window Management Building Block: this building block is in charge of managing the encoding encoding window at a Tetrys sender.

  -- Editor’s note: Is it worth moving it in a dedicated BB? To be discussed --

- o Other ?

In order to enable future components and services to be added dynamically, Tetrys adds a header extension mechanism, compatible with that of LCT, NORM, FECFRAME [REFS].

# 4. Packet Format

## 4.1. Common Header Format

All types of Tetrys packets share the same common header format (see Figure 2).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   V   | C |S|     Reserved    |   HDR_LEN     |  Packet Type  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Congestion Control Information (CCI, length = 32*C bits)    |
|                           ...                                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Transport Session Identifier (TSI, length = 32*S bits)     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Header Extensions (if applicable)              |
|                           ...                                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: Common Header Format

-- Editor’s note: this format inherits from the LCT header format (RFC 5651) with slight modifications. --

- o Tetrys version number (V): 4 bits. Indicates the Tetrys version number. The Tetrys version number for this specification is 1.

- o Congestion control flag (C): 2 bits. C=0 indicates the Congestion Control Information (CCI) field is 0 bits in length. C=1 indicates the CCI field is 32 bits in length. C=2 indicates the CCI field is 64 bits in length. C=3 indicates the CCI field is 96 bits in length.

  -- Editor’s note: version number and congestion control to be discussed --

- o Transport Session Identifier flag (S): 1 bit. This is the number of full 32-bit words in the TSI field. The TSI field is 32*S bits in length, i.e., the length is either 0 bits or 32 bits.

- o Reserved (Resv): 9 bits. These bits are reserved. In this version of the specification, they MUST be set to zero by senders and MUST be ignored by receivers.

- o Header length (HDR_LEN): 8 bits. Total length of the Tetrys header in units of 32-bit words. The length of the Tetrys header MUST be a multiple of 32 bits. This field can be used to directly access the portion of the packet beyond the Tetrys header, i.e., to the first other header if it exists, or to the packet payload if it exists and there is no other header, or to the end of the packet if there are no other headers or packet payload.

- o Packet Type: 8 bits. Type of packet.

- o Congestion Control Information (CCI): 0, 32, 64, or 96 bits Used to carry congestion control information. For example, the congestion control information could include layer numbers, logical channel numbers, and sequence numbers. This field is opaque for the purpose of this specification. This field MUST be 0 bits (absent) if C=0. This field MUST be 32 bits if C=1. This field MUST be 64 bits if C=2. This field MUST be 96 bits if C=3.

- o Transport Session Identifier (TSI): 0 or 32 bits. The TSI uniquely identifies a session among all sessions from a particular sender. The TSI is scoped by the IP address of the sender, and thus the IP address of the sender and the TSI together uniquely identify the session. Although a TSI in conjunction with the IP address of the sender always uniquely identifies a session, whether or not the TSI is included in the Tetrys header depends on

what is used as the TSI value. If the underlying transport is UDP, then the 16-bit UDP source port number MAY serve as the TSI for the session. If the TSI value appears multiple times in a packet, then all occurrences MUST be the same value. If there is no underlying TSI provided by the network, transport or any other layer, then the TSI MUST be included in the Tetrys header.

#### 4.1.1. Header Extensions

Header Extensions are used in Tetrys to accommodate optional header fields that are not always used or have variable size. The presence of Header Extensions can be inferred by the Tetrys header length (HDR_LEN). If HDR_LEN is larger than the length of the standard header, then the remaining header space is taken by Header Extension fields.

If present, Header Extensions MUST be processed to ensure that they are recognized before performing any congestion control procedure or otherwise accepting a packet. The default action for unrecognized Header Extensions is to ignore them. This allows the future introduction of backward-compatible enhancements to Tetrys without changing the Tetrys version number. Non-backward-compatible Header Extensions CANNOT be introduced without changing the Tetrys version number.

There are two formats for Header Extension fields, as depicted in Figure 3. The first format is used for variable-length extensions, with Header Extension Type (HET) values between 0 and 127. The second format is used for fixed-length (one 32-bit word) extensions, using HET values from 128 to 255.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  HET (<=127)  |       HEL     |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
.                                                               .
.              Header Extension Content (HEC)                   .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  HET (>=128)  |       Header Extension Content (HEC)          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3: Header Extension Format

- o Header Extension Type (HET): 8 bits The type of the Header Extension. This document defines a number of possible types. Additional types may be defined in future versions of this specification. HET values from 0 to 127 are used for variable-length Header Extensions. HET values from 128 to 255 are used for fixed-length 32-bit Header Extensions.

- o Header Extension Length (HEL): 8 bits The length of the whole Header Extension field, expressed in multiples of 32-bit words. This field MUST be present for variable-length extensions (HETs between 0 and 127) and MUST NOT be present for fixed-length extensions (HETs between 128 and 255).

- o Header Extension Content (HEC): variable length The content of the Header Extension. The format of this sub-field depends on the Header Extension Type. For fixed-length Header Extensions, the HEC is 24 bits. For variable-length Header Extensions, the HEC field has variable size, as specified by the HEL field. Note that the length of each Header Extension field MUST be a multiple of 32 bits. Also note that the total size of the Tetrys header, including all Header Extensions and all optional header fields, cannot exceed 255 32-bit words.

## 4.2. Source Packet Format

A source packet is the encapsulation of a source symbol, a source symbol ID and a Common Packet Header. The source symbols can have variable sizes.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
/                       Common Packet Header                    /
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Source Symbol ID                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
/                             Payload                           /
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: Source Packet Format

Common Packet Header: a common packet header where Packet Type=0.

Source Symbol ID: the sequence number to identify a source symbol.

Payload: the payload (source symbol)

### 4.3. Coded Packet Format

A coded packet is the encapsulation of a coded symbol, a coded symbol ID, the associated encoding vector and the Common Packet Header. As the source symbols CAN have variable sizes, each source symbol size need to be encoded, and the result must be stored in the coded packet as the Encoded Payload Size (16 bits): as it is an optional field, the encoding vector MUST signal the use of variable source symbol sizes with the field V (see Section 6.1.1.2).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
/                       Common Packet Header                    /
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Coded Symbol ID                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
/                         Encoding Vector                       /
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Encoded Payload Size      |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                                                               |
/                            Payload                            /
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5: Coded Packet Format

Common Packet Header: a common packet header where Packet Type=1.

Coded Symbol ID: the sequence number to identify a coded symbol.

Encoding Vector: an encoding vector to define the linear combination used (coefficients, and source symbols).

Encoded Payload Size: the coded payload size used if the source symbols have variable size (optional, Section 6.1.1.2)).

Payload: the coded symbol.

### 4.4. Acknowledgement Packet Format

A Tetrys Decoding Building Block or Tetrys Recoding Building Block MAY send back to another building block some Acknowledgement packets. They contain information about what it is received and/or decoded, and other information such as a packet loss rate or the size of the decoding buffers. The acknowledgement packets are OPTIONAL hence they could be omitted or lost in transmission without impacting the basic protocol performance.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
/                       Common Packet Header                    /
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Nb of missing source symbols                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Nb of not already used coded symbols             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     First Source Symbol ID                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   SACK size   |                                               |
+-+-+-+-+-+-+-+-+                                               +
|                                                               |
/                          SACK Vector                          /
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6: Acknowledgement Packet Format

Common Packet Header: a common packet header where Packet Type=2.

Nb missing source symbols: the number of missing source symbols in the receiver.

Nb of not already used coded symbols: the number of not already used coded symbols in the receiver that have not already been used for decoding. Meaning the number of linear combinations containing at least 2 unknown source symbols.

First Source Symbol ID: ID of the first source symbol to acknowledge.

SACK size: the size of the SACK vector in 32-bit words. For instance, with value 2, the SACK vector is 64 bits long.

SACK vector: bit vector indicating the acknowledged symbols following the first source symbol ID. The "First Source Symbol" is not included in this bit vector. A bit equal to 1 at position i means that the source symbol of ID equal to "First Source Symbol ID" + i + 1 is acknowledged by this acknowledgment packet.

# 5. The Coding Coefficient Generator Identifiers

## 5.1. Definition

The Coding Coefficient Generator Identifier defines a function or an algorithm to build the coding coefficients used to generate the coded symbols. They MUST be known by all the Building Blocks.

## 5.2. Table of Identifiers

0000: GF256 Vandermonde based coefficients. Each coefficient is build as alpha^( (source_symbol_id*coded_symbol_id) % 255).

0001: GF16 Vandermonde based coefficients. Each coefficient is build as alpha^( (source_symbol_id*coded-symbol_id) % 15).

0010: SRLC.

Others: To be discussed.

# 6. Tetrys Basic Functions

## 6.1. Encoding

At the beginning of a transmission, a Tetrys Encoding Building Block MUST choose an initial code rate (added redundancy) as it doesn’t know the packet loss rate of the channel. In steady state, the Tetrys Encoding Building Block generates coded symbols when it receives some information from the decoding or recoding blocks.

When a Tetrys Encoding Building Block needs to generate a coded symbol, it considers the set of source symbols stored in the Elastic Encoding Window. These source symbols are the set of source symbols which are not yet acknowledged by the receiver.

A Tetrys Encoding Building Block SHOULD set a limit of the Elastic Encoding Window size. This allows to reduce the complexity by considering less source symbols. It also provides a coping mechanism if all the acknowledgment packets are lost.

At the generation of a coded symbol, the Tetrys Encoding Building Block generates an encoding vector containing the IDs of the source

symbols stored in the Elastic Encoding Window. For each source symbol, a finite field coefficient is determined using a Coding Coefficient Generator. This generator CAN take as input the source symbol ID and the coded symbol ID and CAN determine a coefficient in a deterministic way. A classical example of such deterministic function is a generator matrix where the rows are indexed by the source symbol IDs and the columns by the coded symbol IDs. For example, the entries of this matrix can be built from a Vandermonde structure, like Reed-Solomon codes, or from a sparse binary matrix, like Low-Density Generator Matrix codes. Finally, the coded symbol is the sum of the source symbols multiplied by their corresponding coefficients.

### 6.1.1. Encoding Vector Formats

The encoding vectors are sent in each coded symbols. They CAN contain the source symbol IDs and/or the coefficients.

To avoid the overhead of transmitting all the source symbol IDs, the following algorithm is used to compress them.

#### 6.1.1.1. Transmitting the source symbol IDs

The source symbol IDs are organized as a sorted list of 32-bit integers. Instead of sending the full list, a differential transform to reduce the number of bits needed to represent an ID is used.

##### 6.1.1.1.1. Compressing the Source symbol IDs

Assume the symbol IDs used in the combination are: [1..3],[5..6],[8..10].

1. Keep the first element in the packet as the first_source_id: 1.

2. Apply a differential transform to the others elements ([3,5,6,8,10]) which removes the element i-1 to the element i, starting with the first_source_id as i0, and get the list L => [2,2,1,2,2]

3. Compute b, the number of bits needed to store all the elements, which is ceil(log2(max(L))): here, 2 bits.

4. Write b in the corresponding field, and write all the b * [(2 * NB blocks) - 1] elements in a bit vector, here: 10 10 01 10 10.

#### 6.1.1.1.2. Decompressing the Source symbol IDs

When a Tetrys Decoding Building Block wants to reverse the operations, this algorithm is used:

1. Rebuild the list of the transmitted elements by reading the bit vector and b: [10 10 01 10 10] => [2,2,1,2,2]

2. Apply the reverse transform by adding successively the elements, starting with first_source_id: [1,1+2,(1+2)+2,(1+2+2)+1,...] => [1,3,5,6,8,10]

3. Rebuild the blocks using the list and first_source_id: [1..3],[5..6],[8..10].

#### 6.1.1.2. Encoding Vector Format

The encoding vector CAN be used to store the source symbol IDs included in the associated coded symbol, the coefficients used in the combination, or both. It CAN be used to send only the number of source symbols included in the coded symbol.

If the source IDs are stored, the nb of blocks MUST be different from 0.

The encoding vector format uses a 4-bit Coding Coefficient Generator Identifier to identity the algorithm to generate the coefficients, and contains a set of blocks for the source symbol IDs used in the combination. In this format, the number of blocks is stored as a 8-bit unsigned integer. To reduce the overhead, a compressed way to store the symbol IDs is used: the IDs are not stored as themselves, but stored as the difference between the previous.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     EV_LEN    |  CCGI |I|C|V| |   NB_BLOCKS   |   NB_COEFS    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       FIRST_SOURCE_ID                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     b_id    |                                                 |
+-+-+-+-+-+-+-+              id_bit_vector        +-+-+-+-+-+-+-+
|                                                 |   Padding   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    b_coef   |                                                 |
+-+-+-+-+-+-+-+            coef_bit_vector        +-+-+-+-+-+-+-+
|                                                 |   Padding   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 7: Encoding Vector Format

- o Encoding Vector Length (EV_LEN): size in units of 32-bit words.

- o Coding Coefficient Generator Identifier (CCGI): 4-bit ID to identify the algorithm or the function used to generate the coefficients (see Section 5). As a CCGI is included in each encoded vector, it can dynamically change between the generation of 2 coded symbols.

- o Store the IDs flag (I): 1 bit to know if an encoding vector contains the list of the IDs used. MUST be 1 if the Encoding Vector stores the source symbol IDs.

- o Store the coefficients flag (C): 1 bit to know if an encoding vector contains information about the coefficients used.

- o Having source symbols with variable size flag (V): set V to 1 if the combination which refers the encoding vector is a combination of source symbols with variable sizes. In this case, the coded packets MUST have the ’Encoded Payload Size’ field.

- o Number of blocks used to store the source symbol IDs (NB_BLOCKS): the number of blocks used to store all the source symbol IDs.

- o Number of coefficients (NB_COEFS): The number of the coefficients used to generate the associated coded symbol.

- o The first source Identifier (FIRST_SOURCE_ID): the first source symbol ID used in the combination.

- o Number of bits for each edge block (b_id): the number of bits needed to store the edge (see Section 6.1.1.1).

- o The compressed edge blocks (id_bit_vector): equal to b_id * (NB_BLOCKS * 2 - 1).

- o Number of bits needed to store each coefficient (b_coef): the number of bits used to store the coefficients.

- o The coefficients (coef_bit_vector): The coefficients stored (as a vector of b_coef * NB_COEFS).

- o Padding: padding to have an Encoding Vector size multiple of 32-bit (for the id and coefficient part).

## 6.2. The Elastic Encoding Window

When an input source symbol is passed to a Tetrys Encoding Building Block, it is added to the Elastic Encoding Window. This window MUST have a limit set by the encoding building Block (depending of the use case: unicast, multicast, file transfer, real-time transfer, ...). If the Elastic Encoding Window reached its limit, the window slides over the symbols: the first (oldest) symbols are removed. Then, a packet containing this symbol can be sent onto the network. As an element of the coding window, this symbol is included in the next linear combinations created to generate the coded symbols.

As explained below, the receiver or the recoder sends periodic feedback indicating the received or decoded source symbols. In the case of a unicast transmission, when the sender receives the information that a source symbol was received and/or decoded by the receiver, it removes this symbol from the coding window.

In a multicast transmission:

- o If the acknowledgement packets are not enabled, the coding window grows up to a limit. When the limit is reached, the oldest symbols are removed from the coding window.

- o If the acknowledgement packets are enabled, a source symbol is removed from the coding window when all the receivers have received or decoded it or when the coding window reaches its limit.

### 6.3. Recoding

#### 6.3.1. Principle

A Tetrys Recoding Block maintains a list of the ID of the source symbols included in the Elastic Coding Window of the sender. It also stores a set of received source and coded symbols able to regenerate the set or a subset of the symbols of the Elastic Coding Window. In other words, if R1, ..., Rt represent t received symbols and S1, ..., Sk represent the set or a subset of the source symbols of the Elastic Coding Window, there exists a t*k-matrix M such that (R1, ..., Rt).M = (S1, ..., Sk).

#### 6.3.2. Generating a coded symbol at an intermediate node

At the generation of a coded symbol, the Tetrys Recoding Building Block generates an encoding vector containing the IDs of the source symbols stored in the Elastic Encoding Window or in the subset of the Elastic Encoding Window that it is able to regenerate. The Tetrys Recoding Building Block then generates a new coded symbol ID different from the received coded symbol IDs. From this coded symbol ID and the source symbol IDs of (S1, ..., Sk), a vector of coefficients is determined using a Coding Coefficient Generator. Let (a1, ...,ak) denote the obtained coefficients. To compute the linear combination (s1, ..., Sk).transpose(a1, ..., ak) the Tetrys Recoding Building block computes the vector v = (a1, ...,ak).transpose(M) and then computes the coded symbol R = (R1, ..., Rt).transpose(v). It can be verified that the new coded symbol is obtained without any decoding operation.

### 6.4. Decoding

A classical matrix inversion is sufficient to recover the source symbols.

## 7. Security Considerations

N/A

## 8. Privacy Considerations

N/A

## 9. IANA Considerations

N/A

## 10. Acknowledgments

N/A

## 11. References

### 11.1. Normative References

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

### 11.2. Informative References

[AHL-00] Ahlswede, R., Ning Cai, , Li, S., and R. Yeung, "Network information flow", IEEE Transactions on Information Theory vol.46, no.4, pp.1204,1216, July 2000.

[FECFRAME] Watson, M., Begen, A., and V. Roca, "Forward Error Correction (FEC) Framework", Request for Comments 6363, October 2011.

[NWCRG-ARCH] NWCRG, , "Network Coding Architecture", TBD TBD.

[RMT] Vicisano, L., Gemmel, J., Rizzo, L., Handley, M., and J. Crowcroft, "Forward Error Correction (FEC) Building Block", Request for Comments 3452, December 2002.

[Tetrys] Lacan, J. and E. Lochin, "Rethinking reliability for long-delay networks", International Workshop on Satellite and Space Communications 2008 (IWSSC08), October 2008.

## Authors' Addresses

Jonathan Detchart
ISAE
10, avenue Edouard-Belin
BP 54032
Toulouse CEDEX 4  31055
France

Email: jonathan.detchart@isae.fr

Emmanuel Lochin
ISAE
10, avenue Edouard-Belin
BP 54032
Toulouse CEDEX 4  31055
France

Email: emmanuel.lochin@isae.fr

Jerome Lacan
ISAE
10, avenue Edouard-Belin
BP 54032
Toulouse CEDEX 4  31055
France

Email: jerome.lacan@isae.fr

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

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

IRTF Network Coding Research Group (NWCRG)
Internet-Draft
Intended status: Informational
Expires: May 18, 2017

V. Roca
INRIA

November 14, 2016

# FECFRAMEv2: Adding Sliding Encoding Window Capabilities to the FEC Framework: Problem Position
# draft-roca-nwcrg-fecframev2-problem-position-03

## Abstract

The Forward Error Correction (FEC) Framework (or FECFRAME) (RFC 6363) has been defined by the FECFRAME IETF WG to enable the use of FEC Encoding with real-time flows in a flexible manner. The original FECFRAME specification only considers the use of block FEC codes, wherein the input flow(s) is(are) segmented into a sequence of blocks and FEC encoding performed independently on a per-block basis. This document discusses an extension of FECFRAME in order to enable a sliding (potentially elastic) window encoding of the input flow(s), using convolutional FEC codes for the erasure channel, as an alternative to block FEC codes.

## Status of This Memo



## Copyright Notice

Table of Contents

1. Introduction . . . . . 2
2. Notations, Definitions and Abbreviations . . . . . 4
2.1. Requirements Notation . . . . . 4
2.2. Definitions . . . . . 4
3. Key features of FECFRAME . . . . . 5
3.1. FECFRAME is more a shim layer than a protocol instantiation . . . . . 6
3.2. FECFRAME is highly flexible . . . . . 6
3.3. Details are in each FEC Scheme . . . . . 6
3.4. FECFRAME needs session-level description . . . . . 7
4. Application of FECFRAME (RFC 6363) to network coding use-cases: a discussion . . . . . 7
4.1. Block versus convolutional codes . . . . . 7
4.2. End-to-end versus in-network re-coding . . . . . 8
4.3. Single versus multi-sources, intra versus inter-flows coding . . . . . 8
4.4. Single versus multi-paths . . . . . 8
5. Architectural considerations for FECFRAMEv2 . . . . . 9
5.1. FECFRAMEv2 in sliding encoding window mode . . . . . 9
5.2. ADU(I) to source symbol mapping . . . . . 11
5.3. Sliding encoding window management . . . . . 12
5.4. Encoding Symbol Identifiers (ESI) . . . . . 13
5.5. Block and convolutional co-existence in a given FECFRAMEv2 session . . . . . 14
6. Security Considerations . . . . . 14
7. Privacy Considerations . . . . . 15
8. IANA Considerations . . . . . 15
9. Acknowledgments . . . . . 15
10. References . . . . . 15
10.1. Normative References . . . . . 15
10.2. Informative References . . . . . 16
Author’s Address . . . . . 17

## 1. Introduction

The Forward Error Correction (FEC) Framework (or FECFRAME) [RFC6363], produced by the FECFRAME IETF WG [fecframe-charter], describes a framework for using Forward Error Correction (FEC) codes with applications in public and private IP networks to provide protection

against packet loss. The framework supports applying FEC to arbitrary packet flows over unreliable transport and is primarily intended for real-time, or streaming, media. This framework can be used to define Content Delivery Protocols that provide FEC for streaming media delivery or other packet flows. Content Delivery Protocols defined using this framework can support any FEC scheme (and associated FEC codes) that is compliant with various requirements defined in [RFC6363]. Thus, Content Delivery Protocols can be defined that are not specific to a particular FEC scheme, and FEC schemes can be defined that are not specific to a particular Content Delivery Protocol.

However, it is REQUIRED in [RFC6363] that the FEC scheme operate in a block manner, i.e., the input flow(s) MUST be segmented into a sequence of blocks, and FEC encoding (at a sender/coding node) and decoding (at a receiver/decoding node) MUST be performed independently on a per-block basis. This approach has a major impact on coding and decoding delays when used with block FEC codes (e.g., [RFC6681], [RFC6816] or [RFC6865]) since encoding requires that all the source symbols be known at the encoder. In case of continuous input flow(s), even if source symbols can be sent immediately, repair symbols are necessarily delayed by the block creation time, that directly depends on the block size (i.e., the number of source symbols in this block, k). This block creation time is also the minimum decoding latency any receiver will experience in case of erasures, since no repair symbol for the current block can be received before. A good value for the block size is necessarily a good balance between the minimum decoding latency at the receivers (which must be in line with the most stringent real-time requirement of the flow(s)) and the desired robustness in front of long erasure bursts (which depends on the block size).

On the opposite, a convolutional code associated to a sliding encoding window (of fixed size) or a sliding elastic encoding window (of variable size) removes this minimum decoding delay, since repair symbols can be generated and sent on-the-fly, at any time, from the source symbols present in the current encoding window. Using a sliding encoding window mode is therefore highly beneficial to real-time flows, one of the primary targets of FECFRAME.

The present document introduces the FECFRAME framework specificities, its limits, and options to extend it to sliding (optionally elastic) encoding windows and convolutional codes.

## 2. Notations, Definitions and Abbreviations

### 2.1. 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].

### 2.2. Definitions

This document uses the following definitions, that are mostly inspired from [RFC5052], [RFC6363] and [nc-taxonomy-id].

Packet Erasure Channel:

a communication path where packets are either dropped (e.g., by a congested router or because the number of transmission errors exceeds the correction capabilities of the physical layer codes) or received. When a packet is received, it is assumed that this packet is not corrupted

Systematic Code:

code in which the Source Symbols are part of the Output Symbols

Input Symbol:

a unit of data that is provided as an input to the coding process, in a given coding node. It may be a source symbol or an already encoded repair symbol if in-network re-coding is considered

Output Symbol:

a unit of data that is produced as an output of the coding process, in a given coding node

Application Data Unit (ADU):

The unit of source data provided as payload to the transport layer. Depending on the use-case, an ADU may use an RTP encapsulation.

ADU Information (ADUI):

a unit of data constituted by the ADU and the associated Flow ID, Length and Padding fields (Section 5.2).

Source Symbol:

an original unit of data, before any coding process is applied. Source symbols are the result of the fragmentation of ADUIs.

Repair Symbol:

an Output Symbol that is not a Source Symbol.

FEC Source Packet:

At a sender (respectively, at a receiver) a payload submitted to (respectively, received from) the transport protocol containing an ADU along with an Explicit Source FEC Payload ID (if present).

FEC Repair Packet:

At a sender (respectively, at a receiver) a payload submitted to (respectively, received from) the transport protocol containing a repair symbol (or several repair symbols with certain FEC Schemes) along with a Repair FEC Payload ID (and possibly an RTP header in some cases).

(Source) ADU Flow:

A sequence of ADUs associated with a transport-layer flow identifier (such as the standard 5-tuple {Source IP address, source port, destination IP address, destination port, transport protocol}). Depending on the use-case, several ADU flows may be protected together by the FECFRAME framework.

FEC Source Packet Flow:

A sequence of FEC Source Packets.

FEC Repair Packet Flow:

A sequence of FEC Repair Packets.

FEC Framework Configuration Information (FFCI):

Information which controls the operation of the FEC Framework. The FFCI enables the synchronization of the FECFRAME sender and receiver instances.

## 3. Key features of FECFRAME

### 3.1. FECFRAME is more a shim layer than a protocol instantiation

FECFRAME is not a full featured Protocol Instantiation (unlike ALC [RFC5775] and NORM [RFC5740] for instance). It is more a shim layer, or more precisely a framework for using FEC inside existing transport protocols. For instance when FECFRAME is used end-to-end inside a single RTP/UDP stream (the simplest use-case), RTP [RFC3550] and UDP are the transport protocols and FECFRAME is a functional component that performs FEC encoding/decoding and generates RTP compliant repair packets. Even if specific headers are defined for the associated FEC Scheme, FECFRAME is not a full featured transport protocol.

### 3.2. FECFRAME is highly flexible

FECFRAME is highly flexible in the way it can be used. In particular FECFRAME:

- can protect a single RTP flow [RFC3550], repair packets being themselves RTP packets, possibly multiplexed in the same UDP connection but using a different Payload Type (PT) to distinguish them from source packets. This is particularly useful if backward compatibility is mandatory: non-FECFRAME aware receivers simply drop packets with unknown PT. However this should be regarded as a particular case;

- can protect a single source flow that does NOT use RTP, where repair packets are NOT RTP packets either;

- can protect several source flows, from the same source or from several sources, some of them being RTP flows but not necessarily the other ones;

- can generate a single repair flow or multiple repair flows;

- can be used with any upper protocols (RTP or any other protocol) and transport protocols (e.g., UDP, DCCP) if this latter preserves datagram boundaries;

- can be used with unicast or multicast/broadcast transmissions;

### 3.3. Details are in each FEC Scheme

In the FECFRAME architecture, most technical details are in the FEC Scheme. In particular a FEC Scheme defines:

- FEC code specifications and associated FEC Encoding ID;

- o the way source symbols are created from the data units coming from the application(s), called Application Data Units (ADU);

- o signaling for FEC Source Packets (optional), called Source FEC Payload ID;

- o signaling for FEC Repair Packets (mandatory), called Repair FEC Payload ID;

### 3.4. FECFRAME needs session-level description

FECFRAME works in conjunction with SDP (or a similar protocol) to specify high level per FECFRAME Instance (i.e., per-session) signaling [RFC6364]. This information, called FEC Framework Configuration Information [RFC6363], describes:

- o the incoming flows (content description and flow identification);

- o the outgoing flows, for source and repair packets;

- o what FEC Scheme is used, identified via the FEC Encoding ID;

- o and the FEC Scheme specific parameters.

In practice, the FEC Scheme is valid for the whole FECFRAME Instance duration, since no update mechanism has been defined to carry a new SDP session description reliably and in real-time to all the potential receivers. This is different from ALC or NORM where the FEC Scheme selection is made on a per-object manner (rather than per-session).

## 4. Application of FECFRAME (RFC 6363) to network coding use-cases: a discussion

The FECFRAME framework has a certain number of features and restrictions. We discuss each of them below, at the light of the use-cases identified for Network Coding.

### 4.1. Block versus convolutional codes

FECFRAME, as described in [RFC6363], MUST be associated to block FEC codes. For instance ([RFC6363], section 5.1) says:

> "1. Construction of a source block from ADUs. The FEC code will be applied to this source block to produce the repair payloads."

Therefore the input flow(s) is (resp. are) segmented into a sequence of blocks, FEC encoding being performed independently on a per-block basis.

However this is not a fundamental limitation, in the sense that the same FECFRAME architecture can be used with sliding (potentially elastic) encoding windows, associated with convolutional codes. To that purpose it is sufficient:

- o to update [RFC6363] adding the support of sliding (potentially elastic) encoding windows along with the source block approach;

- o to specify dedicated FEC Schemes, working with convolutional FEC codes for the erasure channel. All the details of the codes, the required signaling, the management of the sliding encoding window and creation of source symbols will be defined in these FEC Schemes.

### 4.2. End-to-end versus in-network re-coding

The FECFRAME architecture, as specified in [RFC6363], MUST feature a single encoding node and a single decoding node. These nodes may be the source and destination nodes, or may be middle-boxes, or any combination.

The question of having multiple in-network re-coding operations is not considered in [RFC6363]. The question whether this is feasible and appropriate, given the typical FECFRAME use-cases, is an open question that remains to be discussed.

### 4.3. Single versus multi-sources, intra versus inter-flows coding

FECFRAME, as specified in [RFC6363], can globally protect several flows that can originate either from a single source or from multiple sources. This also means that FECFRAME can perform both intra-flow coding or inter-flows coding. The only requirement is that those flows be identified and signaled to the FECFRAME encoder and decoder via the FEC Framework Configuration Information (e.g., it can be detailed in the SDP description).

From this point of view, FECFRAME is already in line with advanced network-coding use-cases.

### 4.4. Single versus multi-paths

FECFRAME, as specified in [RFC6363], does not specify nor restrict how the FEC Source Packet Flow(s) and FEC Repair Packet Flow(s) are to be transmitted: whether they go through the same path (e.g., when

they are sent to the same UDP connection) or through multiple paths is irrelevant to FECFRAME since it is an operational and management aspect. However, it is anticipated that when several repair flows are generated, offering different protections levels (e.g., through different code-rates), these repair flows will often use different paths, to better accommodate receiver heterogeneity.

From this point of view, FECFRAME is already in line with advanced network-coding use-cases.

## 5. Architectural considerations for FECFRAMEv2

Several architectural considerations are now discussed for version 2 of FECFRAME. We assume hereafter that FECFRAMEv2 follows the initial spirit of FECFRAME, i.e., is only applied in end-to-end (see Section 4.2). From what follows we show that adding sliding encoding window support to FECFRAMEv2 is simple and can coexist with legacy FECFRAME flows. Extending FECFRAMEv2 to other situations, e.g., with in-network re-coding, is not considered in this document.

### 5.1. FECFRAMEv2 in sliding encoding window mode

```
   +----------------------+
   |     Application      |
   +----------------------+
              |
              | (1) Application Data Units (ADUs)
              v
   +----------------------+                          +----------------+
   |   FEC Framework v2   |                          |                |
   |                      |-------------------------->|   FEC Scheme   |
   |(2) Update of sliding | (3) Source Symbols of    |                |
   |    encoding window   |     the sliding window   |                |
   |                      |<--------------------------|                |
   |                      | (4) Explicit Source      |                |
   |(7) Construct FEC     |     FEC Payload ID(s)    |(5) FEC Encoding|
   |    source and repair |<--------------------------|    (optional)  |
   |    packet(s)         | (6) Repair FEC Payload ID |                |
   +----------------------+     + Repair symbol(s)   +----------------+
              |
              | (8) FEC source packets and FEC repair packets
              v
   +----------------------+
   |   Transport Layer    |
   |     (e.g., UDP)      |
   +----------------------+
```

Figure 1: Architecture of FECFRAMEv2 in sliding encoding window mode, at a sender/coding node.

Figure 1 (adapted from [RFC6865]) illustrates the general architecture of FECFRAMEv2 when working in sliding encoding window mode. The difference with respect to the [RFC6363] architecture lies in steps 2 to 6. Instead of creating a source block, composed of a certain number of ADUs plus their associated flow/length/padding information (see for instance [RFC6865]), FECFRAMEv2 in sliding encoding window mode continuously updates this window (step 2) and communicates the set of symbols to the FEC Scheme (step 3). This latter then returns the Explicit Source FEC Payload ID(s) (step 4) so that the new symbol(s) can be sent immediately. When FECFRAMEv2 needs to send one or several FEC repair packets (this is determined by the desired target code rate), it asks the FEC Scheme to create one or several repair symbols (step 5) along with their Repair FEC Payload ID (step 6). The associated FEC Repair Packets are then sent (steps 7 and 8).

When FECFRAMEv2 works with a block FEC Scheme, Figure 2 and Figure 3 of [RFC6363] remain valid, without any change.

## 5.2. ADU(I) to source symbol mapping

Let us now detail the ADU to source symbol mapping. As in FECFRAME, each ADU is first prepended with its {flow ID, length} information (respectively the F and L fields of Figure 2) and potentially zero padded to align to a multiple of the target symbol length ("0 padding" field of Figure 2). This augmented ADU is called ADUI.

ADUIs are then mapped to source symbols. Since incoming ADUs can have largely varying sizes, it makes sense to use a symbol size significantly lower than the PMTU (as in [MBMS], section 8.2.2.7) which means that large ADUIs will be segmented into several source symbols while small ADUIs may fit into a single or low number of symbols. This has the advantage of limiting transmission overhead if at the same time the FEC Scheme enables the transmission of several repair symbols in the same FEC Repair Packet. However one may also choose to associate a symbol size equal to the maximum ADUI size of the current block, in case of a block FEC Scheme, as in [RFC6816] or [RFC6865], in order to always have a one-to-one mapping between ADUIs and source symbols.

The block versus sliding window mode does have an impact on the strategy chosen. More precisely:

- o FECFRAMEv2 in sliding window mode MUST use a fixed symbol size, indicated in the FEC Framework Configuration Information (FFCI).

- o FECFRAMEv2 in block mode and FECFRAME MAY use a dynamic symbol size, chosen on a per-block basis, or MAY use a fixed symbol size, indicated in the FEC Framework Configuration Information (FFCI).

In any case it is recommended that the symbol size be small enough with respect to the PMTU.

FEC code related considerations can impact the choice of a symbol size (assuming they are of fixed size). This is out of the scope of this document.

Figure 2 illustrates the creation of the ADUIs from incoming ADUs and the mapping to source symbols in case of small, fixed size symbols.

```
      Symbol Length             Symbol Length
  <----------------------><----------------------->
  +----+----+-------------------------+------------+
  |F[i]|L[i]|ADU[i]                   | 0 padding  | => symbols j, j+1
  +----+----+-------+-----+-----------+------------+
  |Fi+1|Li+1|ADUi+1 |  0  |                        => symbol j+2
  +----+----+-------+---+-+
  |Fi+2|Li+2|ADUi+2     |0|                        => symbols j+3
  +----+----+-----------+-+------+-----------------+
  |Fi+3|Li+3|ADUi+3              | 0 padding       | => symbols j+4, j+5
  +----+----+--------------------+-----------------+
```

Figure 2: ADUI and source symbols, case of small symbol sizes, for either FECFRAME or FECFRAMEv2.

### 5.3. Sliding encoding window management

Let us now detail the sliding window update process at a sender. Two kinds of limitations exist that impact the sliding window management:

- o at the FEC Scheme level: this latter can have internal or practical limitations (e.g., for complexity reasons) that limit the number of source symbols in the encoding window;

- o at the FECFRAMEv2 instance level: the source flows can have real-time constraints that limit the number of source symbols in the encoding window;

The most stringent limitation defines the maximum window size in terms of either number of source symbols or number of ADUs (depending on the relationship between them, see Section 5.2, they can be equal or not).

Source symbols are added to the sliding encoding window as ADUs arrive.

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

- o after a certain delay, for situations where the sliding encoding window is managed on a time basis. The motivation is that an old ADU of a real-time flow becomes useless after a certain delay. The ADU retention delay in the sliding encoding window is therefore initialized according to the real-time features of incoming flow(s);

- o once the sliding encoding window has reached its maximum size, when there is an upper limit to the sliding encoding window size;

- o when the sliding encoding window is of fixed size, the oldest symbol is removed each time a new symbol is added;

- o if the sender knows that a particular ADU has been correctly received by the receiver(s), the corresponding source symbol(s) is(are) removed. Of course this mechanism requires that an acknowledgement mechanism be setup to inform the FECFRAMEv2 sender of good ADU reception, which is out of the scope of FECFRAMEv2.

### 5.4. Encoding Symbol Identifiers (ESI)

Any **source** symbol of a flow MUST be uniquely identified during the full duration where this symbol is useful.

Depending on the FEC Scheme being used, a **repair** symbol of a flow may or not need to be uniquely identified during the full duration where this symbol is useful. For instance, being able to identify a repair symbol is OPTIONAL with Random Linear Codes (RLC) since the coding window content and associated coding vector are communicated in the Repair FEC Payload ID and nothing else is needed to process this repair symbol. But being able to identify a repair symbol is REQUIRED with FEC Schemes that use this symbol identifier during the encoding and decoding processes (this is the case for instance with any block FEC code and some of the convolutional FEC codes).

In block mode, the encoding symbols are uniquely identified both by their Source Block Number (SBN) and Encoding Symbol ID (ESI), the first k ESI values identifying source symbols and the remaining n-k ESI values the repair symbols [RFC5052]. In sliding encoding window mode, the situation is totally different:

- o since there is no block, there is no SBN;

- o since there is no block, the ESI space that identifies source symbols is linear, each source symbol having an ESI that is 1 greater than the previous source symbol, except when a wrap-around to zero occurs after reaching the maximum ESI value permitted by the ESI field size (see below);

- o an ESI space dedicated to repair symbols is used when the FEC Scheme requires repair symbols to be identified. This ESI space is logically different from the ESI space used for source symbols. Therefore the same ESI value identifies different symbols depending on whether we are considering a FEC source packet or FEC repair packet. This is the context (e.g., the transport identifiers like the destination UDP port number) that enables a FECFRAME receiver to distinguish between source and repair symbols, not the ESI value;

Since the ESI space is limited by the header/trailer ESI field size to b bits (as specified by the FEC Scheme), wrap-around to zero is unavoidable with long FECFRAMEv2 sessions. This has two consequences:

- o the maximum sliding encoding window size MUST be smaller than 2^^b, and in practice be significantly smaller;

- o if the network may significantly delay packets, there is a risk of confusion if an ESI wrap-around takes place in the meantime, since the delayed symbol may be misinterpreted as a fresh symbol. A security margin is therefore needed that consists in having a "b" value sufficiently large to avoid such confusions. What security margin to consider is a deployment decision that also depends on the various flow transmission bitrates. Note that a timestamp information carried in FEC Source Packets may help identifying delayed packets. However this is not a generic mechanism since ADU flows are not required to use RTP framing.

### 5.5. Block and convolutional co-existence in a given FECFRAMEv2 session

When two (or more) FEC Repair Packet Flows are used in a given FECFRAME session, it is possible to have both a block FEC Scheme on one flow and a convolutional FEC Scheme on the other flow, both of them protecting the same ADU flow(s). This can be useful in order to preserve backward compatibility, legacy receivers joining the FEC Repair Packet Flow corresponding to the block FEC Scheme and ignoring the other flow.

The SDP description associated to this FECFRAMEv2 session indicates if a FEC Repair Packet flow works in block mode or sliding encoding window mode. This is done through the FEC Encoding ID communicated via the "a=fec-repair-flow: encoding-id=0; ..." attribute [RFC6364] (or "a=FEC-declaration:VALUE encoding-id=VALUE" attribute in case of [MBMS]). Then, from this FEC Encoding ID, the FECFRAME receiver can easily deduce if the FEC Scheme corresponds to a block or a convolutional FEC code.

## 6. Security Considerations

Adding the new sliding window mode to FECFRAMEv2 (what this document is about) in addition to the block mode of FECFRAME, while keeping the end-to-end approach of FECFRAME, does not fundamentally change the situation from a security point of view. Therefore all the security considerations detailed in [RFC6363] also apply to FECFRAMEv2. More precisely:

- o the problem statement, section 9.1 of [RFC6363];

- o the attacks against the data flows, section 9.2 of [RFC6363];
- o the attacks against the FEC parameters, section 9.3 of [RFC6363];
- o the discussion related to the FEC protection of several source flows, section 9.4 of [RFC6363];
- o and the baseline secure FEC Framework operation, section 9.5 of [RFC6363];

all apply to FECFRAMEv2, regardless of whether it follows a block or sliding window mode. Security considerations specific to a FEC Scheme, if any, will have to be discussed in the associated FEC Scheme document.

## 7. Privacy Considerations

Adding the new sliding window mode to FECFRAMEv2 (what this document is about), in addition to the block mode of FECFRAME, does not change the situation from a privacy point of view. Those considerations will be discussed in an update of [RFC6363].

## 8. IANA Considerations

N/A

## 9. Acknowledgments

The author wants to thank Morten V. Pedersen for his comments to this document.

## 10. References

### 10.1. Normative References

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

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <http://www.rfc-editor.org/info/rfc3550>.

[RFC5052] Watson, M., Luby, M., and L. Vicisano, "Forward Error Correction (FEC) Building Block", RFC 5052, DOI 10.17487/RFC5052, August 2007, <http://www.rfc-editor.org/info/rfc5052>.

[RFC6363] Watson, M., Begen, A., and V. Roca, "Forward Error Correction (FEC) Framework", RFC 6363, DOI 10.17487/RFC6363, October 2011, <http://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, <http://www.rfc-editor.org/info/rfc6364>.

## 10.2. Informative References

[fecframe-charter] FECFRAME WG, IETF., "FEC Framework (fecframe) charter", URL: http://www.ietf.org/wg/concluded/fecframe.html, March 2013.

[MBMS] 3rd Generation Partnership Project (3GPP) SA4 Working Group, , "Multimedia Broadcast/Multicast Service (MBMS): Protocols and codecs", 3GPP TS 26.346 http://www.3gpp.org/DynaReport/26346.htm, March 2016.

[nc-taxonomy-id] Firoiu, V., Adamson, B., Roca, V., Adjih, C., Bilbao, J., Fitzek, F., Masucci, A., and M. Montpetit, "Network Coding Taxonomy", draft-irtf-nwcrg-network-coding-taxonomy-00 (work in progress), November 2014.

[RFC5740] Adamson, B., Bormann, C., Handley, M., and J. Macker, "NACK-Oriented Reliable Multicast (NORM) Transport Protocol", RFC 5740, DOI 10.17487/RFC5740, November 2009, <http://www.rfc-editor.org/info/rfc5740>.

[RFC5775] Luby, M., Watson, M., and L. Vicisano, "Asynchronous Layered Coding (ALC) Protocol Instantiation", RFC 5775, DOI 10.17487/RFC5775, April 2010, <http://www.rfc-editor.org/info/rfc5775>.

[RFC6681] Watson, M., Stockhammer, T., and M. Luby, "Raptor Forward Error Correction (FEC) Schemes for FECFRAME", RFC 6681, DOI 10.17487/RFC6681, August 2012, <http://www.rfc-editor.org/info/rfc6681>.

[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, <http://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, <http://www.rfc-editor.org/info/rfc6865>.

## Author’s Address

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

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