# FECFRAMEv2: Adding Sliding Encoding Window Capabilities to the FEC Framework: Problem Position

Vincent Roca

IRTF Network Coding Research Group (NWCRG)
Internet-Draft
Intended status: Informational
Expires: December 31, 2015

V. Roca
INRIA

June 29, 2015

# FECFRAMEv2: Adding Sliding Encoding Window Capabilities to the FEC Framework: Problem Position

draft-roca-nwcrg-fecframev2-problem-position-01

## 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

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 December 31, 2015.

## Copyright Notice

Table of Contents

## 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

Code Rate:

> the ratio between the number of Input Symbols and the number of Output Symbols at given coding node. The Code Rate belongs to a ]0; 1] interval. A Code Rate close to 1 indicates that a small number of Repair Symbols have been produced during the encoding process

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

Source Symbol:

an original unit of data, before any coding process is applied

Repair Symbol:

an Output Symbol that is not a Source Symbol

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.

(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.

ADU Information (ADUI):

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

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 one repair symbol along with a Repair FEC Payload ID and possibly an RTP header.

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.

To Be Completed...

### 2.3. Abbreviations

This document uses the following abbreviations.

NC:

Network Coding

PT:

RTP Payload Type field, [RFC3550]

## 3. Key aspects of FECFRAME

Let us discuss some key aspects 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 still not a full featured transport protocol.

### 3.2. 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;
- the way source symbols are created from the flow(s) of incoming packets (see Section 5.2);
- signaling for source packets (optional), called Source FEC Payload ID;
- signaling for repair packets (mandatory), called Repair FEC Payload ID;

### 3.3. FECFRAME is pretty flexible

FECFRAME is pretty 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 for backward compatibility purposes: non-FECFRAME aware receivers simply drop packets with unknown PT;
- 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 (e.g., RTP) and transport protocols (e.g., UDP, DCCP) if this latter preserves datagram boundaries;
- can be used with unicast or multicast/broadcast transmissions;

### 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. This information, called FEC Framework Configuration Information [RFC6363], describes:

- the incoming flows (content description and flow identification);
- the outgoing flows, for source and repair packets;
- what FEC Scheme is used, identified via the FEC Encoding ID;
- 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) 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:

- to update [RFC6363] adding the support of sliding (potentially elastic) encoding windows along with the source block approach;
- to specify dedicated FEC Schemes, working with convolutional codes. All the details of the codes, associated signaling, creation of the sliding encoding windows 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 source flow(s) and repair 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 as 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.

### 4.5. Application at the transport versus network versus MAC layer

FECFRAME, as specified in [RFC6363], has been designed only with the transport layer use-case in mind. It MUST be located between the application and transport layers.

The question of having FECFRAME used in lower protocol layers is not considered in [RFC6363]. Whether this is feasible and appropriate, given the typical FECFRAME use-cases, is an open question that remains to be discussed.

## 5. A few key architectural aspects for FECFRAMEv2

A few key architectural aspects follow, proving the feasibility of adding sliding encoding window support to FECFRAMEv2. We assume hereafter that FECFRAMEv2 follows the initial spirit of FECFRAME, i.e., is applied in end-to-end (see Section 4.2). Extending FECFRAMEv2 to other situations, for instance with in-network re-coding, is not considered here and remains to be discussed.

### 5.1. High level view of FECFRAMEv2 in sliding encoding window mode

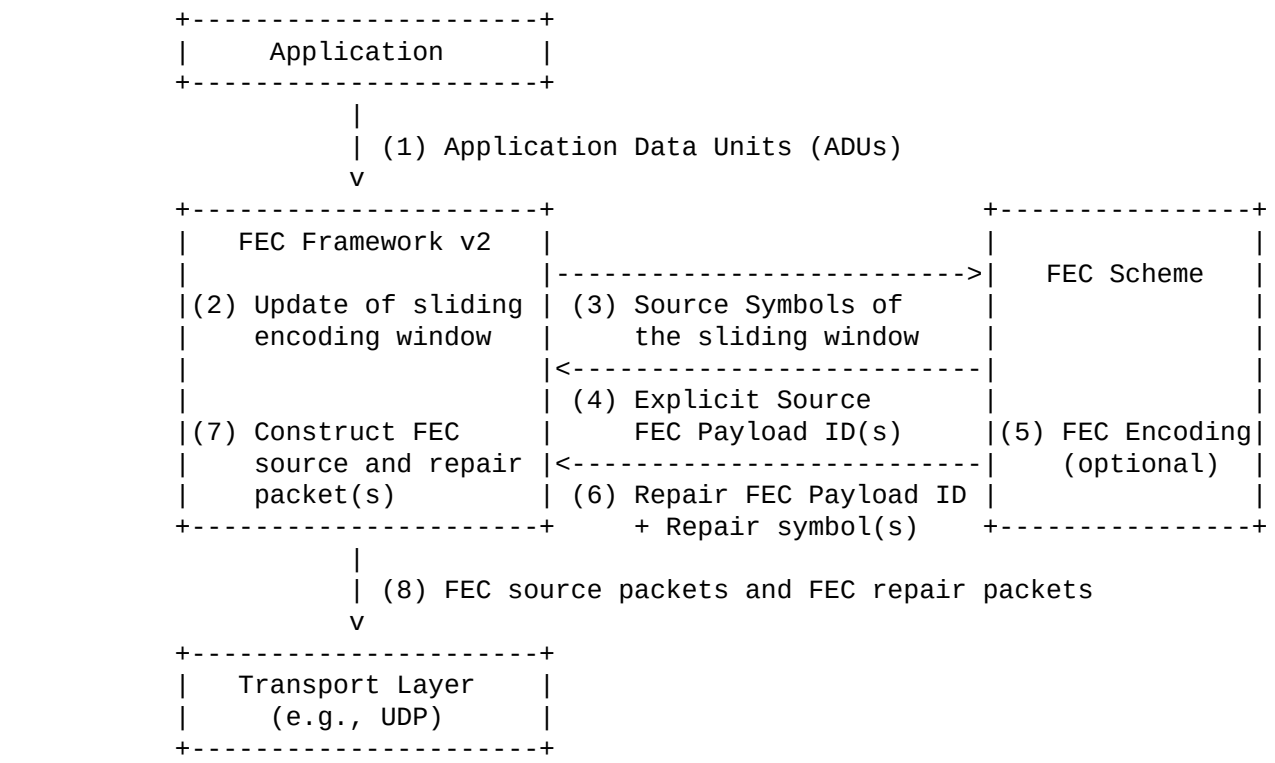


Figure 1: Architecture of FECFRAMEv2 in sliding encoding window mode, at a sender/coding node.

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 returns the Explicit Source FEC Payload ID(s) (step 4) when applicable so that the new symbol(s) can be sent immediately (steps 7 and 8 limited to FEC source packets). When FECFRAMEv2 needs to send one or several FEC repair packets, it asks the FEC Scheme to create one or several repair symbols (step 5) along with their Repair FEC Payload ID (step 6). Note that depending on the FEC scheme, the number repair symbols may be limited at a certain point of time. The packets are then sent (steps 7 and 8).

When FECFRAMEv2 work in the traditional block mode, Figure 2 and Figure 3 of [RFC6363] remain valid.

## 5.2. ADU / source symbol mapping

Let us now detail the ADU / 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 with the target symbol length (see below) (Pad field of Figure 2). This augmented ADU is called ADUI (see Figure 2).

Then source symbols are mapped to ADUIs. There is not necessarily a 1-1 bijection between ADUIs and symbols, because incoming packets can have largely varying sizes. In that case it MAY make sense to have small symbols of size significantly lower than the PMTU, as in [RFC6681]. On the opposite, [RFC6816] and [RFC6865] more conservatively associate one symbol per ADUI. (NB: this is different from ALC or NORM where source and repair packets are all of the same size, except maybe the packet containing the last source symbol of an object).

The source and repair symbol length can therefore be determined in different ways. It can be of fixed length, indicated in the FEC Framework Configuration Information (FFCI). Otherwise, for situations where there is a single source symbol per ADUI, the symbol length:

- with FECFRAMEv2 in block mode or FECFRAME, can be dynamic and chosen on a per-block basis (see [RFC6865], section 4.3, "E" parameter);
- with FECFRAMEv2 in window mode, can be dynamic and chosen on a per-window basis. In that case the length of repair symbols will dynamically evolve as well in order to be equal to the maximum ADUI size in the encoding window at the time of encoding;
- in all cases, can be statically defined, for instance of the maximum (P)MTU size minus the various header sizes;

Figure 2 illustrates the creation of the ADUI, from incoming ADUs, and the correspondence to source symbols in the case there is a single source symbol per ADUI. It assumes the symbol length is aligned with the maximum ADUI length currently in the sliding encoding window (here corresponding to the ADU of index i+2).

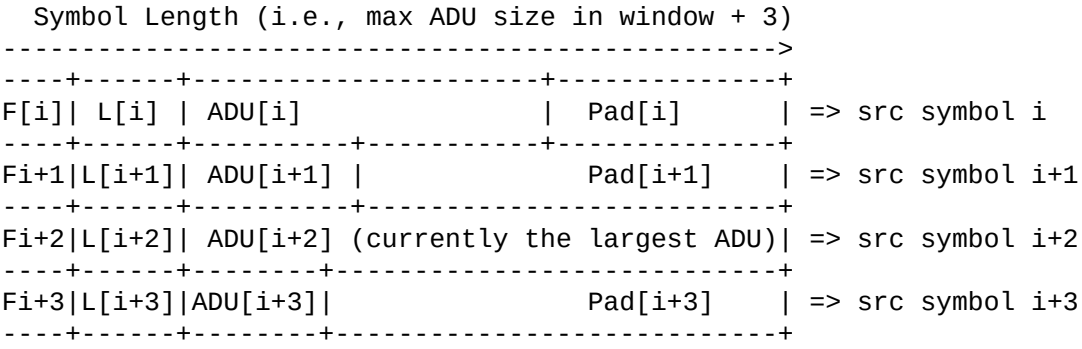


Figure 2: ADUI and source symbols, simple case with a single source symbol per ADUI, of length equal to the maximum ADUI size in the sliding encoding window.

When FECFRAMEv2 works in a sliding encoding window mode and the symbol length is chosen on a per-window basis, coding operations can require to adjust the L and Pad fields of each ADUI present in the current sliding encoding window. Indeed, as the maximum length of ADUs in the window varies, the various paddings and therefore L fields need to be updated accordingly. A FECFRAMEv2 implementation must accommodate with this constraint.

## 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:

- 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;

- 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:

- 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);

- once the sliding encoding window has reached its maximum size, for situations where there is an upper limit to the sliding encoding window;

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

- 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. Whether or not this is desirable is an open question;

## 5.4. About symbol identifiers

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 MAY 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 transmitted within the Repair FEC Payload ID. 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 so called encoding symbols are uniquely identified both by their Source Block Number (SBN) and Encoding Symbol ID (ESI), the first k ESI values denoting source symbols, the remaining n-k ESI values the repair symbols [RFC5052]. In sliding encoding window mode, there is no SBN concept (as there is no block) and the ESI space cannot be split into source and repair identifiers as before. Therefore FECFRAMEv2 adds the the Repair (R) bit to distinguish two identifier spaces, one for source symbols (when the "R" bit is 0) and one for repair symbols (when the "R" bit is 1).

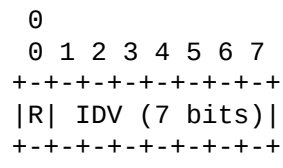


Figure 3: Symbol Identifier, 8-bit version.

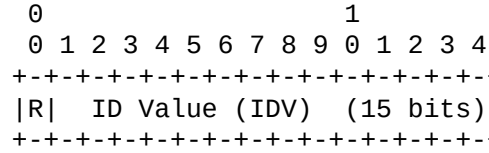


Figure 4: Symbol Identifier, 16-bit version.

Figure 3 and Figure 4 illustrate Symbol ID (SID) format for situations where 8 bits or 16 bits are sufficient to uniquely identify any symbol. Other formats (e.g., 24-bit or 32-bit SID formats) are possible when meaningful. The SID is composed of:

Repair (R) flag (1-bit field):

> when "R" is 0, it indicates this is a source symbol, and when "R" is 1, indicates this is a repair symbol;

ID Value (IDV) 7-bit (resp. 15-bit) field:

> sequence number of a symbol, that may be either a source or repair symbol, depending on the "R" flag. The IDV starts from 0, is incremented in a monotonic way, and wraps-around to zero after reaching the maximum value permitted by the IDV field size;

Therefore each source (and optionally repair) symbol of a flow is uniquely identified by its Symbol ID: SID = {R; IDV}. Different flows have different flow identifiers within the current FECFRAMEv2 instance, so the {flowID, SID} tuple enables to identify any symbol.

IDV being a b-bit long field (e.g., with b=7 or 15 in the above figures), wrap-arounds will occur after the maximum value is achieved. This has two consequences:

- the maximum sliding encoding window size MUST be at most equal to 2^^b;
- in situations where the network may delay packets significantly, there is a risk of confusion. A symbol significantly delayed during transmission, that has been removed from the current

sliding encoding window and after an IDV wrap-around, may be mis-interpreted as a fresh symbol. Note that a timestamp information carried in this packet may help identifying this situation, however this cannot be considered as the general case (e.g., flows do not necessarily use RTP framing). A security margin is therefore needed that consist of a value for b sufficiently high to avoid such confusions no matter what happens. What security margin to consider is a deployment decision that is out of the scope of this document.

### 5.5. New convolutional FEC schemes

A convolutional FEC Scheme for the erasure channel is required for FECFRAMEv2 working in sliding encoding window mode.

Several convolutional codes are possible, Random Linear Codes (RLC) being one option among others. Depending on the code, the Source and Repair FEC Payload IDs will be adapted. Some codes will also require the transmission of the coding vectors in addition to the current sliding window composition. These considerations, among others, will be specified as part of the FEC Scheme.

### 5.6. Updating the SDP session-level description

For FECFRAMEv2, the SDP session-level description needs to indicate whether:

- coding follows a block approach;
- coding follows a sliding encoding window approach;
- coding follows both of them: for instance a FEC repair packet flow may use one approach and another FEC repair packet flow, sent on a different UDP connection, may use the other one. This could be useful in order to preserve backward compatibility with legacy receivers, or in order to combine the advantages of both coding approaches;

Other details (list to be defined) need to be specified in the session-level description.

## 6. Relationships with the Tetrys and DNC proposals

### 6.1. Is FECFRAMEv2 fundamentally different from Tetrys?

The target use-cases for FECFRAME and Tetrys [tetrys-id] are rather similar: essentially end-to-end, with unicast or multicast flows, with a single or multiple sources and flows.

In terms of coding, FECFRAMEv2 and Tetrys follow the same sliding window approach, potentially elastic. The main difference comes from the manner the sliding window progresses.

Feedback flows are missing from FECFRAMEv2, but probably exist in the upper RTP/RTCP protocol (if used), in a simplified manner as they provide global statistics rather than per source packet reception/ decoding status. Adding a detailed symbol-level acknowledgement mechanism to FECFRAMEv2, although feasible, is not in line with the original FECFRAME philosophy. Tetrys also performs code rate adjustment, based on feedback information. The same can be done with FECFRAMEv2 in case an RTCP feedback flow is available, with global reception/loss statistics.

If Tetrys is a full featured protocol, with a specific header, FECFRAME is more a shim layer. Whether it is appropriate or not to turn FECFRAMEv2 into something more elaborate, thereby breaking backward compatibility and the initial philosophy of FECFRAME, remains an open question.

### 6.2. Is FECFRAMEv2 fundamentally different from Dynamic Network Coding?

Some of the key concepts of DNC [dnc-id] can be added to FECFRAMEv2 fairly easily: coding decisions and flow management decisions may depend on timing aspects thanks to a timestamp carried in the various packets. Very often, real-time flows will already contain a timestamp as part of their RTP header which facilitates these decisions.

## 7. Security Considerations

TBD

## 8. Privacy Considerations

TBD

## 9. IANA Considerations

N/A

## 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, July 2003.

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

[RFC6363] Watson, M., Begen, A., and V. Roca, "Forward Error Correction (FEC) Framework", RFC 6363, October 2011.

## 10.2. Informative References

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

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

[RFC6681] Watson, M., Stockhammer, T., and M. Luby, "Raptor Forward Error Correction (FEC) Schemes for FECFRAME", RFC 6681, August 2012.

[RFC6816] Roca, V., Cunche, M., and J. Lacan, "Simple Low-Density Parity Check (LDPC) Staircase Forward Error Correction (FEC) Scheme for FECFRAME", RFC 6816, December 2012.

[RFC6865] Roca, V., Cunche, M., Lacan, J., Bouabdallah, A., and K. Matsuzono, "Simple Reed-Solomon Forward Error Correction (FEC) Scheme for FECFRAME", RFC 6865, February 2013.

[dnc-id] Montpetit, M., Roca, V., and j. jonathan.detchart@isae.fr, "Dynamic Network Coding", draft-montpetit-dynamic-nc-00 (work in progress), March 2015.

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

[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.

[tetrys-id]
Detchart, J., Lochin, E., Lacan, J., and V. Roca, "Tetrys, an On-the-Fly Network Coding protocol", draft-detchart-nwcrg-tetrys-01 (work in progress), March 2015.

## 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/