HAL
open science

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

Vincent Roca

To cite this version:

Vincent Roca. FECFRAMEv2: Adding Sliding Encoding Window Capabilities to the FEC Framework: Problem Position. 2016, pp.18. hal-01141470v3

HAL Id: hal-01141470

https://inria.hal.science/hal-01141470v3

Submitted on 24 May 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

IRTF Network Coding Research Group (NWCRG)
Internet-Draft
Intended status: Informational
Expires: November 21, 2016

V. Roca
INRIA

May 20, 2016

FECFRAMEv2: Adding Sliding Encoding Window Capabilities to the FEC Framework: Problem Position
draft-roca-nwcrg-fecframev2-problem-position-02

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 November 21, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

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:

- o 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;

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

- o 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;

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

- o 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;

- o 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:

- o 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, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[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/