Generic Application Programming Interface (API) for Window-Based Codes

Vincent Roca, Jonathan Detchart, Cédric Adjih, Morten V. Pedersen, Ian Swett

NWCRG
Internet-Draft
Intended status: Informational
Expires: April 29, 2018

V. Roca (Ed.)
INRIA
J. Detchart
ISAE - Supaero
C. Adjih
INRIA
M. Pedersen
Aalborg University
I. Swett
Google

October 26, 2017

Generic Application Programming Interface (API) for Window-Based Codes
draft-roca-nwcrg-generic-fec-api-00

Abstract

This document introduces a generic Application Programming Interface (API) for window-based FEC codes. This API is meant to be usable by any sliding window FEC code, independently of the FEC Scheme or network coding protocol that may rely on it. This API defines the core procedures and functions meant to control the codec (i.e., implementation of the FEC code), but leaves out all upper layer aspects (e.g., signalling) that are the responsibility of the application making use of the codec. A goal of this document is to pave the way for a future open-source implementation of such 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 https://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on April 29, 2018.

Table of Contents

1. Introduction

This document introduces a generic Application Programming Interface (API) for window-based FEC codes. This API is meant to be usable by any window-based FEC code, independently of the FEC Scheme or network coding protocol that may rely on it. This API defines the core procedures and functions meant to control the codec (i.e., implementation of the FEC code), but leaves out all upper layer aspects (e.g., signalling) that are the responsibility of the application making use of the codec.

A goal of this document is to pave the way for a future open-source implementation of such codes.

This API must be compatible with:

- MDS and non-MDS codes;
- Fixed rate and rateless codes;
- Codes restricted to end-to-end use-cases as well as codes compatible with in-network re-encoding use-cases;

Since this is a usage consideration, this API is not impacted by the intra-flow versus inter-flow nature of the use-case, nor is it impacted by the single-path versus multi-paths nature of the use-case.

Last but not least, having a common generic API is future-proof. Whatever may appear in the future can be more easily integrated into existing software thanks to a common API.

A few words about block FEC codes and why we do not try to encompass them in this API...

2. Definitions and Abbreviations

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

This document uses the following definitions and abbreviations:

XXX

3. Existing APIs

Editor’s comment: for the moment, what follows is meant to list the various proposals in order to be able to compare and agree on what should be done.

3.1. Morten API proposal

3.2. Jonathan API proposal

```
<CODE BEGINS>
/** a status for function calls **/
typedef enum {
    STATUS_OK,
    STATUS_ERROR,
    /* ... */
```

```
} status_t;


/** defines the galois field used (at least the size, maybe we need to separate the implementations (Lookup Tables or Xor-based)) **/
typedef enum {
    GF_16,
    GF_64,
    GF_256
} galois_field_t;


/*
 * coding coefficient generators: specifies the algorithm used to generate the coefficients for the linear combinations
 *
 * NOTE: the choice of the finite field could done here (RLC_GF256, RLC_GF_16, ...) rather than using a different structure
 */
typedef enum {
    RLC,
    VDM
    /* ... */
} coding_coefficients_generator_identifier_t;


 /***
  **
  ** encoder side
  **
  ***/


/**
 * NOTE for the callbacks in the sw_encoder: this is a proposition, we could also use 2 structures (source_t and repair_t) to avoid sending too many parameters in the callbacks.
 **/


/**
 * context: a context as a generic pointer (defined by the application if needed and given in sw_encoder_set_callbacks)
 * src: the source data unit to consider
 * src_id: the id of the source unit set by the encoder
 * src_sz: the size in bytes of the data unit
 **/
typedef void (*sw_encoder_callback_source_ready)(void *context,void* src, uint32_t src_id, size_t src_sz);


/**
 * context: a context as a generic pointer (defined by the application if needed and given in sw_encoder_set_callbacks)
 * rep: the repair data unit generated by the encoder
 * rep_id: the id of the repair unit given by the encoder
```

```
 * rep_sz: the size in bytes of the repair data unit
 * src_ids: the array of the source unit ids used to generate the repair unit
 * src_coefs: the coefficients used for each source units to generate the repair unit
 * nb_src_in: number of src units in the linear combination (repair unit) (also the number of elements of src_ids and src_coefs)
 **/
typedef void (*sw_encoder_callback_repair_ready)(void *context,void* rep, uint32_t rep_id, size_t rep_sz, uint32_t* src_ids, uint8_t *src_coefs, size_t nb_src_in);


/** a structure containing all we need to encode **/
typedef struct sw_encoder sw_encoder_t;


/**
 * @brief Init a sliding window encoder by giving a galois field size, a coefficient generator function, and the maximum size of the window
 * @param galois_field                    The size of the galois field used to create the coefficients and to encode.
 * @param ccgi                                    The function used to generate the coefficients (depends on the gf_size)
 * @param max_wnd_size                    set the maximum of source units t consider inside the coding window (the oldest units will be destroyed)
 * @return a sw_encoder_t structure.
 **/
sw_encoder_t* sw_encoder_init(galois_field_t galois_field, coding_coefficients_generator_identifier_t ccgi, uint32_t max_wnd_size);


/**
 * @brief Set the callbacks to get the encoded (repair) data
 * @param encoder                           The encoder initialized
 * @param context                           A generic context defined by the application (will be given in the callback)
 * @param src_callback                    The function to be called when a source data unit has been processed by the encoder
 * @param rep_callback                   The function to be called when a repair data unit has been generated
 **/
status_t sw_encoder_set_callbacks(sw_encoder_t* encoder, void* context, sw_encoder_callback_source_ready src_callback, sw_encoder_callback_repair_ready rep_callback);


/**
 * @brief Gives a source data unit  and its id to the encoder
 * @param encoder     The encoder structure.
 * @param src                 The data to add
 * @param sz             The size in bytes of the data unit
 **/
status_t sw_encoder_add_source(sw_encoder_t* encoder, void* src, size_t sz);


/**
 * @brief Removes the corresponding source data unit from the encoding window by giving and id
 * @param encoder     The encoder structure
 * @param id             The id of the data unit
 * @return                    STATUS_OK if the data unit has been found and removed, STATUS_ERROR if the data unit doesn't exist
 **/
status_t sw_encoder_remove_source(sw_encoder_t* encoder, uint32_t id);
```

```
/**
 * @brief generates a repair data unit and calls the corresponding callback.
 * @param encoder             The encoder structure.
 **/
status_t sw_encoder_generate_repair(sw_encoder_t* encoder);


/* ... */
status_t sw_encoder_set_control_parameter(sw_encoder_t* encoder, uint32_t type, void* value, uint32_t length);


/* ... */
status_t sw_encoder_get_control_parameter(sw_encoder_t* encoder, uint32_t type, void* value, uint32_t length);


/**
 * @brief Release an encoder structure
 * @param encoder        The encoder structure
 **/
status_t  sw_encoder_release(sw_encoder_t* encoder);


 /***
  **
  ** decoder side
  **
  ***/


/**
 * NOTE for the callback in the sw_decoder: this is a proposition, we could also use an opaque structure (source_t) to avoid sending too many parameters in the callback.
 **/
/**
 * context: a context as a generic pointer (defined by the application if needed and given in sw_encoder_set_callbacks)
 * src: the source data unit to consider
 * src_id: the id of the source unit used in the decoder
 * src_sz: the size in bytes of the data unit
 **/
typedef void (*sw_decoder_callback_source_ready)(void *context,void* src, uint32_t src_id, size_t sz);


/** a structure containing all we need to decode **/
typedef struct sw_decoder sw_decoder_t;


/**
 * @brief Init a sliding window decoder by giving a galois field size, a coefficient generator function, and the maximum size of the window
 * @param galois_field                    The size of the galois field used to create the coefficients and to encode.
```

```
 * @param ccgi                                    The function used to generate the coefficients (depends on the gf_size)
 * @return a sw_decoder_t structure.
 **/
sw_decoder_t* sw_decoder_init(galois_field_t galois_field, coding_coefficients_generator_identifier_t ccgi);


/**
 * @brief Set the callback to get the encoded (repair) data
 * @param decoder                           The decoder initialized
 * @param context                          A generic context defined by the application (will be given in the callback)
 * @param callback                           The function to be called
 **/
status_t sw_decoder_set_callback_source_ready(sw_decoder_t* decoder, void* context, sw_decoder_callback_source_ready callback);


/**
 * NOTE for the next decode functions: we could also use 2 opaque structures to represent the source and repair units (source_t or repair_t)
 **/


/**
 * @brief decode some source data units by giving to the decoder a new source data unit
 * @param decoder                            the decoder structure
 * @param src                                    the new source data unit
 * @param src_id                           the id of the new source data unit (given by an encoder)
 * @param src_sz                           the size in bytes of the new source data unit
 **/
status_t sw_decoder_decode_with_source(sw_decoder_t* decoder, void* src, uint32_t src_id, size_t src_sz);


/**
 * @brief decode some source data units by giving to the decoder a new repair data unit
 * @param decoder                            the decoder structure
 * @param rep                                    the new repair data unit
 * @param rep_id                           the id of the new repair data unit (given by an encoder)
 * @param rep_sz                           the size in bytes of the new repair data unit
 * @param src_ids:                           the array of the source unit ids used to generate this repair unit
 * @param src_coefs:                    the coefficients used for each source units to generate this repair unit
 * @param nb_src_in:                    number of src units in the linear combination (repair unit) (also the number of elements of src_ids and src_coefs)
 **/
status_t sw_decoder_decode_with_repair(sw_decoder_t* decoder, void* rep, uint32_t rep_id, size_t rep_sz, uint32_t* src_ids, uint8_t * src_coefs, size_t nb_src_in);


/* ... */
status_t sw_decoder_set_control_parameter(sw_decoder_t* decoder, uint32_t type, void* value, uint32_t length);


/* ... */
```

```
status_t sw_decoder_get_control_parameter(sw_decoder_t* decoder, uint32_t type, void* value, uint32_t length);


/**
 * @brief Release a decoder structure
 * @param decoder        The decoder structure to release
 **/
status_t sw_decoder_release(sw_decoder_t* decoder);
<CODE ENDS>
```

Jonathan API proposal

3.3. Cedric API proposal

For DRAGONCAST/DragonNet/GardiNet:

- an API could be globally pretty similar;
- there is a maintained set of symbols of the "codec" where online Gaussian Elimination is performed. But this same set, is used to also re-code packets for generation. For this to work, one uses as pivot the highest index (instead of the lowest in standard RREF), in order to avoid adding symbols with higher indices in the decoding process.
- another set of differences would be that the protocol has more control over the coding process than our current codec proposal. The reason is that DRAGONCAST (re)codes for several neighbors, and in such scenario, there is no "obvious" decision that can be made, for instance:
  - which source symbols (indices) should be present in a generated packet: -> tradeoff: helping the maximum number of nodes (emphasis on "new" undecoded source symbol indices) -vs- helping the neighbor which is the most late in the decoding process (emphasis on "old" source symbols)
  - which symbols should be kept in the decoding process (or dropped): -> tradeoff: helping coding by keeping old symbols (be able to generate symbols for late neighbors) vs keeping up with decoding (and never throwing away a new symbol with high indices). (I. Amdouni discussed such issues in https://tools.ietf.org/html/draft-amdouni-nwcrg-cisew-00 for instance).
- In the current implementation, the packet generation process is done in the protocol which directly "peeks" in the set of symbols in the codec, and creates a linear combination with the ones that suits it.
- Technically the callbacks from the "codec" are:
  - notification of a source symbol is decoded;

* notification that the set of symbols is full (protocol can remove symbols it sees fits, especially decoded symbols);
* for the "over-the-air" reflashing application, the codec can ask the protocol if an already removed source symbol is available (on the assumption that it has been written somewhere else);

3.4. Ian API proposal

3.5. Vincent API proposal

3.5.1. General

```
<CODE BEGINS>
/**
 * The fec_codec_id_t enum identifies the FEC code/codec being used.
 * Since a given fec_codec_id can be used by one or several FEC schemes (that specify
 * both the codes and way of using these codes), it is distinct from the FEC Encoding
 * ID.
 */
typedef enum {
        CODEC_NIL = 0,
        CODEC_RLC
} codec_id_t;


/**
 * Function return value, indicating whether the function call succeeded or not.
 * In case of failure, the detailed error type is returned in a global variable,
 * of_errno (see of_errno.h).
 *
 * STATUS_OK = 0      Success
 * STATUS_FAILURE,    Failure. The function called did not succeed to perform
 *                    its task, however this is not an error. This can happen
 *                    for instance when decoding did not succeed (which is a
 *                    valid output).
 * STATUS_ERROR,      Generic error type. The caller is expected to be able
 *                    to call the library in the future after having corrected
 *                    the error cause.
 * STATUS_FATAL_ERROR Fatal error. The caller is expected to stop using this
 *                    codec instance immediately (it replaces an exit() system
 *                    call).
 */
typedef enum {
        STATUS_OK = 0,
        STATUS_FAILURE,
        STATUS_ERROR,
        STATUS_FATAL_ERROR
```

```
} status_t;


/**
 * Throughout the API, a pointer to this structure is used as an identifier of the current
 * codec instance, also known as "session".
 *
 * This generic structure is meant to be extended by each codec and new pieces of information
 * that are specific to each codec be specified there. However, all the codec specific structures
 * MUST begin the same entries as the ones provided in this generic structure, otherwise
 * hazardous behaviors may happen.
 */
typedef struct session {
        codec_id_t   codec_id;
        codec_type_t codec_type;
} session_t;


/**
* Generic FEC parameter structure used by set_fec_parameters().
*
* This generic structure is meant to be extended by each codec and new pieces of information
* that are specific to each codec be specified there. However, all the codec specific structures
* MUST begin the same entries as the ones provided in this generic structure, otherwise
* hazardous behaviors may happen.
*/
typedef struct {
       /** SENDER and RECEIVER: maximum number of source symbols used for any repair symbol. */
       UINT32      coding_window_max_size;


        /** RECEIVER only: maximum number of source symbols kept in current linear
         * system. If the linear system grows above this limit, older source symbols
         * in excess are removed and the application callback called if set. This
         * value MUST be larger than the coding_window_max_size. */
    UINT32      linear_system_max_size;
   UINT32      encoding_symbol_length;
} parameters_t;
<CODE ENDS>
```

Vincent API proposal

3.5.2. Session Management

```
<CODE BEGINS>
/**
 * This function allocates and partially initializes a new session structure.
 * Throughout the API, a pointer to this session is used as an identifier of the
```

```
 * current codec instance.
 *
 * @param ses           (IN/OUT) address of the pointer to a session. This pointer is updated
 *                      by this function.
 *                      In case of success, it points to a session structure allocated by the
 *                      library. In case of failure it points to NULL.
 * @param codec_id      identifies the FEC code/codec being used.
 * @param codec_type    indicates if this is a coder or a decoder.
 * @param verbosity     set the verbosity level
 * @return              Completion status. The ses pointer is updated according to the success return status.
 */
status_t     create_codec_instance (session_t**  ses,
                                    codec_id_t   codec_id,
                                    codec_type_t codec_type,
                                    uint32_t     verbosity);


/**
 * This function releases all the internal resources used by this FEC codec instance.
 * None of the source symbol buffers will be free'ed by this function, even those decoded by
 * the library if any, regardless of whether a callback has been registered or not. It's the
 * responsibility of the caller to free them.
 *
 * @param ses           (IN) Pointer to the session.
 * @return              Completion status
 */
status_t     release_codec_instance (session_t*        ses);


/**
 * Second step of the initialization, where the application specifies code(c) specific parameters.
 *
 * At a receiver, the parameters can be extracted from the FEC OTI that is usually communicated
 * to the receiver by either an in-band mechanism or an out-of-band mechanism, or set statically
 * for a specific use-case.
 *
 * @param ses           (IN) Pointer to the session.
 * @param params        (IN) pointer to a structure containing the FEC parameters associated to
 *                      a specific FEC codec.
 * @return              Completion status.
 */
status_t     set_fec_parameters (session_t*         ses,
                                 parameters_t*      params);


/**
 * This function sets a FEC scheme/FEC codec specific control parameter,
 * using a type/value method.
```

```
 *
 * @param ses           (IN) Pointer to the session.
 * @param type          (IN) Type of parameter. This type is FEC codec ID specific.
 * @param value         (IN) Pointer to the value of the parameter. The type of the object pointed
 *                      is FEC codec ID specific.
 * @param length        (IN) length of pointer value
 * @return              Completion status.
 */
status_t     set_control_parameter (session_t*    ses,
                                    UINT32        type,
                                    void*         value,
                                    UINT32        length);


/**
 * This function gets a FEC scheme/FEC codec specific control parameter,
 * using a type/value/length method.
 *
 * @param ses           (IN) Pointer to the session.
 * @param type          (IN) Type of parameter. This type is FEC codec ID specific.
 * @param value         (IN/OUT) Pointer to the value of the parameter. The type of the object
 *                      pointed is FEC codec ID specific. This function updates the value object
 *                      accordingly. The application, who knows the FEC codec ID, is responsible
 *                      to allocating the appropriate object pointed by the value pointer.
 * @param length        (IN) length of pointer value
 * @return              Completion status.
 */
status_t     get_control_parameter (session_t*    ses,
                                    UINT32        type,
                                    void*         value,
                                    UINT32        length);
<CODE ENDS>
```

Vincent API proposal

3.5.3. Callback Functions

```
<CODE BEGINS>
/**
 * Set the various callback functions for this session.
 * All the callback functions require an opaque context parameter, that must be
 * initialized accordingly by the application, since it is application specific.
 *
 * @param ses           (IN) Pointer to the session.
 *
 * @param decoded_source_symbol_callback
 *                      (IN) Pointer to the function, within the application, that
 *                      needs to be called each time a source symbol is decoded.
```

```
 *
 * @param available_source_symbol_callback
 *                      (IN) Pointer to the function, within the application, that
 *                      needs to be called each time a source symbol is decoded and
 *                      all computations performed (i.e., the buffer does contain the
 *                      symbol value).
 *
 * @param source_symbol_removed_from_coding_window_callback
 *                      (IN) Pointer to the function, within the application, that
 *                      needs to be called each time a source symbol is removed from
 *                      the left side of the coding window, at a SENDER because this
 *                      window has slided to the right, or at a RECEIVER because this
 *                      old source symbol is now forgotten.
 *
 * @param context_4_callback
 *                      (IN) Pointer to the application-specific context that will be
 *                      passed to the callback function (if any). This context is not
 *                      interpreted by this function.
 *
 * @return              Completion status.
 */


status_t     set_callback_functions (of_session_t*        ses,
    void* (*decoded_source_symbol_callback) (void *context,
                                             UINT32      size,           /* size of decoded source symbol */
                                             UINT32      esi),           /* encoding symbol ID */
    void (*available_source_symbol_callback) (void       *context,
                                              void       *new_symbol_buf,/* symbol buffer */
                                              UINT32     size,           /* size of decoded source symbol */
                                              UINT32     esi),           /* encoding symbol ID */
    void (*source_symbol_removed_from_coding_window_callback)
                                             (void   *context,
                                              UINT32 old_symbol_esi),
    void*                context_4_callback);
<CODE ENDS>
```

Vincent API proposal

3.5.4. Coding window functions

TBD

3.5.5. Coding coefficients functions

```
<CODE BEGINS>
/**
* SENDER:   this function specifies the coding coefficients chosen by the application if this is the way the codec
```

```
*      works. This function MUST be called before calling build_repair_symbol().
* RECEIVER: communicate the coding coefficients associated to a repair symbol and carried in the packet header.
*      This function MUST be called before calling decode_with_new_repair_symbol().
*
* @param ses
* @param coding_coefs_tab      (IN) table of coding coefficients to be associated to each of the source symbols
*                              currently in the coding window. The size (number of bits) of each coefficient
*                              depends on the FEC scheme. The allocation and release of this table is under the
*                              responsibility of the application.
* @param nb_coefs_in_tab       (IN) number of entries (i.e., coefficients) in the table.
* @return                      Completion status.
*/
status_t     set_coding_coefficients_tab (session_t*     ses,
                                          void*          coding_coefs_tab,
                                          UINT32         nb_coefs_in_tab);

/**
* SENDER:   this function enables the application to retrieve the set of coding coefficients generated and used by
*      build_repair_symbol().
* RECEIVER: never used.
*
* @param ses
* @param coding_coefs_tab      (IN/OUT) pointer of a table of coding coefficients to be associated to each of the
*                              source symbols currently in the coding window. The size (number of bits) of each
*                              coefficient depends on the FEC scheme. The allocation and release of this table is
*                              under the responsibility of the application. Upon return of this function, this
*                              table is allocated and filled with each coefficient value.
* @param nb_coefs_in_tab       (IN/OUT) pointer to the number of entries (i.e., coefficients) in the table.
*                              Upon calling this function, this number must be zero. Upon return of this function
*                              this number is initialized with the actual number of entries in the coeffs_tab[].
* @return                      Completion status (OF_STATUS_OK, FAILURE, ERROR or FATAL_ERROR).
*/
status_t     get_coding_coefficients_tab (session_t*     ses,
                                          void**         coding_coefs_tab,
                                          UINT32*        nb_coefs_in_tab);

/**
* The coding coefficients may be generated in a deterministic manner, (e.g., through the use of a PRNG and the
* repair symbol ESI used as a seed). This is the case with RLC codes.
*
* SENDER:   generate all coefficients. This function MUST be called before calling build_repair_symbol().
* RECEIVER: generate all coefficients. This function MUST be called before calling decode_with_new_repair_symbol().
*
* @param ses
* @param params                (IN) pointer to a codec specific structure containing the required parameters.
*                              These parameters can include a repair symbol ESI or key among other things.
* @return                      Completion status.
*/
```

```
status_t     generate_coding_coefficients (session_t*     ses,
                                           void*          params);
<CODE ENDS>
```

Vincent API proposal

3.5.6. Encoder specific functions

```
<CODE BEGINS>
/**
 * Create a single repair symbol, i.e. perform an encoding.
 * This function requires that the application has previously set the coding window and if needed the coding coefficients
 * appropriately. After that, the application can call this function.
 *
 * @param ses
 * @param new_repair_symbol_buf (IN) The pointer to the buffer for the repair symbol to build can either point to a buffer
 *                              allocated by the application, or let to NULL meaning that this function will allocate
 *                              memory.
 * @return                      Completion status.
 */
status_t     ccod_build_repair_symbol (session_t*      ses,
                                       void*           new_repair_symbol_buf);
<CODE ENDS>
```

Vincent API proposal

3.5.7. Decoder specific functions

```
<CODE BEGINS>
/**
 * Submit a received source symbol and try to progress in the decoding. For each decoded source
 * symbol, if any, the application is informed through the dedicated callback functions.
 *
 * This function usually returns OF_STATUS_OK, regardless of whether this new symbol enabled the
 * decoding of one or several source symbols, unless an error occured. This function cannot return
 * OF_STATUS_FAILURE.
 *
 * @param ses
 * @param new_src_symbol_buf    (IN) Pointer to the new source symbol now available (i.e. a new symbol received by
 *                              the application, or a decoded symbol in case of a recursive call if it makes sense).
 * @param new_symbol_esi_or_key (IN) encoding symbol ID of the new source symbol or key if there is no notion of ESI.
 * @return                      Completion status.
 */
status_t     decode_with_new_source_symbol (session_t*   ses,
                                            void* const     new_src_symbol_buf,
                                            UINT32          new_symbol_esi_or_key);


/**
 * Submit a received repair symbol and try to progress in the decoding. For each decoded source
 * symbol, if any, the application is informed through the dedicated callback functions.
 *
 * This function requires that the application has previously set the coding window and the coding coefficients appropriately.
 * After that, the application can call this function. The
 * application keeps a full control of the repair symbol buffer, i.e., the application is in charge
 * of freeing this buffer as soon as it believes appropriate to do so (a copy is kept by the codec).
 *
 * This function usually returns OF_STATUS_OK, regardless of whether this new symbol enabled the
 * decoding of one or several source symbols, unless an error occurred. This function cannot return
 * OF_STATUS_FAILURE.
 *
 * @param ses
 * @param new_repair_symbol_buf (IN) Pointer to the new repair symbol now available (i.e. a new symbol received by
 *                              the application or a decoded symbol in case of a recursive call if it makes sense).
 * @return                      Completion status.
 */
status_t     decode_with_new_repair_symbol (session_t*    ses,
                                            void* const   new_repair_symbol_buf);
<CODE ENDS>
```

Vincent API proposal

4. Security Considerations

TBD

5. IANA Considerations

N/A.

6. Acknowledgments

The authors would like to thank TBD.

7. 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, <https://www.rfc-editor.org/info/rfc2119>.

Authors' Addresses

Vincent Roca
INRIA
Grenoble
France

EMail: vincent.roca@inria.fr

Jonathan Detchart
ISAE - Supaero
France

EMail: jonathan.detchart@isae-supaero.fr

Cedric Adjih
INRIA
France

EMail: cedric.adjih@inria.fr

Morten V. Pedersen
Aalborg University
Denmark

EMail: mvp@es.aau.dk

Ian Swett
Google
USA

EMail: ianswett@google.com