Generic API for Sliding Window FEC Codes

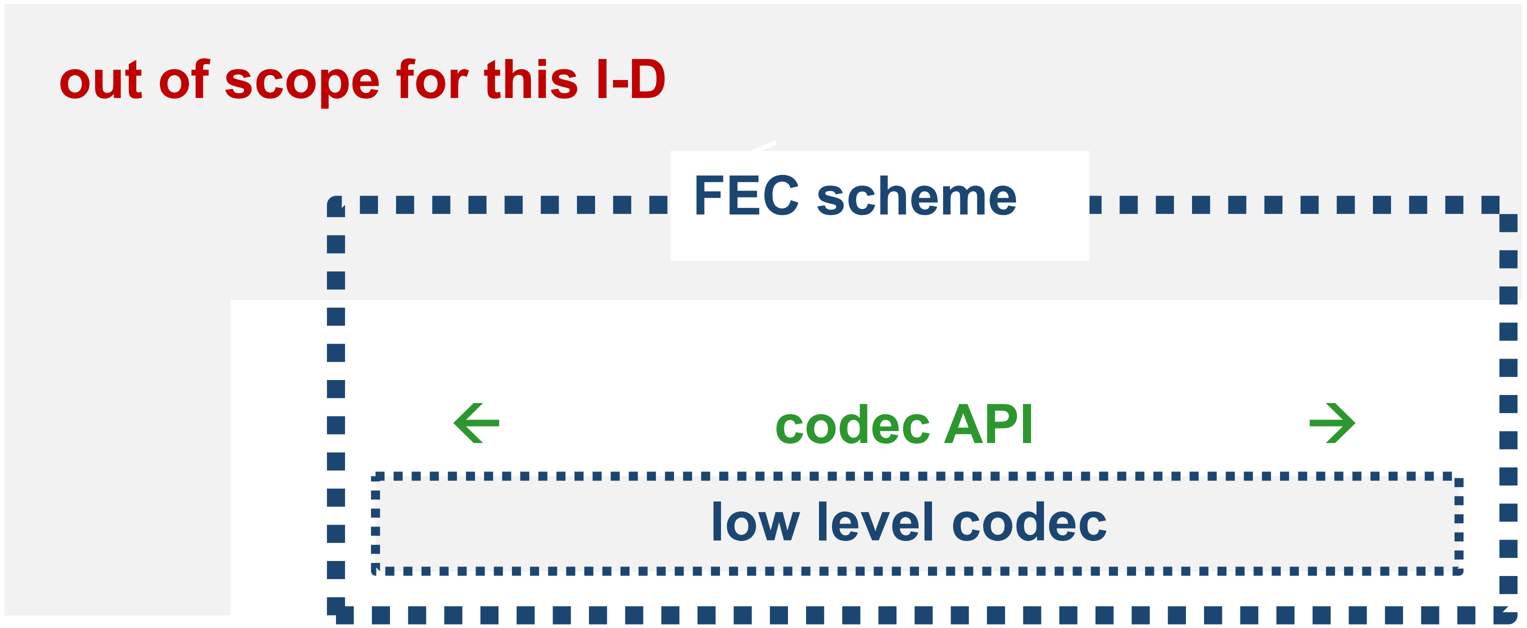
draft-roca-nwcrg-generic-fec-api-02

Vincent Roca (Inria) (ed), Jonathan Detchart (ISAE-Supaéro)
Cédric Adjih (Inria), M. Pedersen (Steinwurf ApS)

IETF102, Montreal, July 19th, 2018

Reminder: a component of a larger software

- **location of the API**
  - we design an API to a low level codec, not to a FEC Scheme
  - the same codec may be used in several FEC Schemes



Reminder: a component of a larger software (2)

- **in scope:**

▪ session management	(sender and receiver)
▪ encoding window management	(sender and receiver)
▪ set/get/generate coding coefficients	(sender and receiver)
▪ build coded symbol	(**sender** only)
▪ decode with newly recvd src/repair symbol	(**receiver** only)

Reminder: a component of a larger software (3)

- **out of scope (non exhaustive)**
  - ~~packet **transmission and reception**;~~
  - ~~**signaling** header creation / parsing;~~
  - ~~**ADU to source symbol mapping**;~~
  - ~~memory management;~~
  - ~~code rate adjustment, for instance thanks to the knowledge of losses at a receiver via feedbacks;~~
  - ~~selective ACK creation and parsing;~~
  - ~~congestion control, etc.~~

Reminder: design goals

- **API compatible with sliding window codes only**
  - ✓ block codes out of scope for the sake of simplicity
- **API compatible with different codes**
  - ✓ differ WRT sliding window management, coding coefficient generation, Finite Field considered, etc.
- **API compatible with end-to-end and in-network recoding use-cases**
  - ✓ RLNC is in scope, RLC too

API structure

- **4.1. General Definitions Common to the Encoder and Decoder**
  - ✓ general definitions, including FEC codepoints (see later)
- **4.2. Coding Window Functions at an Encoder and Decoder**
  - ✓ reset/add symbol to/remove from the coding window
- **4.3. Coding Coefficients Functions at an Encoder and Decoder**
  - ✓ set/generate/get coding coefficients
- **4.4. Encoder**
  - ✓ create/release session, callbacks, parameters, build repair
- **4.5. Decoder**
  - ✓ create/release session, callbacks, parameters, decode with received source/repair symbol

FEC codepoints

GF(2^8), something else?

it's never sent (only the FEC Scheme ID is sent)

- **identifier that fully identifies a codec locally, including parameters like its Galois Field, or the coding coefficient generator (if several exist), or specific features**

e.g. variable density equations

is there an internal coef. generator or does the application list them?

- **several codepoints may exist for the same FEC code, one per codec**
  - ✓ **codepoint 1: general purpose codec for code A**
  - ✓ **codepoint 2: optimized codec for code A**

FEC codepoints (2)

- **Example (will be extended beyond RLC codes, of course)**

```
typedef enum {
    GA_NULL_CODEPOINT = 0,
    /* codepoint for RLC sliding window code, GF(2^8) and variable
     * density (as in FECFRAME FEC Enc. ID XXX). */
    GA_RLC_GF_256_VAR_DENSITY_CODEC,
    /* codepoint for RLC sliding window code, GF(2) and variable
     * density (as in FECFRAME FEC Enc. ID YYY). */
    GA_RLC_GF_2_VAR_DENSITY_CODEC,
    /* list here other identifiers for any FEC codec of interest */
} ga_codepoint_t;
```

Coding window management

- **reset the window**
- **add source symbols**
  - one by one: `add_source_symbol_to_coding_window()`
  - or all at a time: `add_source_symbol_tab_to_coding_window()`
- **remove a source symbol**
  - one at a time: `remove_source_symbol_from_coding_window()`
  - e.g., because a sender knows this source symbol has been received

- **at a sender/encoder, add source symbols progressively, they are automatically removed and application informed of it with a callback**

Coding coefficient management

- **the application can submit it’s coding coefficient list (ex. RLNC)**
  - at an encoder or decoder
  - use the `set_coding_coefs_tab()` function
  - useful when coefficients depend on external conditions (e.g., during recoding at an intermediate node) or are transmitted in headers

- **or the codec may feature a generation function (ex. RLC)**
  - at an encoder or decoder
  - use the `generate_coding_coefs(key, …)` function
  - … and the `get_coding_coefs_tab()` function to retrieve the coefficients generated to add them to the packet header if needed

Encoding

- **principles**
  - make sure coding window is ready
    - ✓ **add new source symbols if any, otherwise leave the coding window (assumed already intialized)**
  - generate or submit coding coefficients
  - call `build_repair_symbol()` each time it’s needed, i.e., depending on the code rate

Decoding

- **principles for a new repair symbol**
  - make sure coding window is ready
    - ✓ **reset and specify source symbols mentioned in the packet header**
  - generate or submit coding coefficients
    - ✓ **as mentioned in the packet header**
  - call `decode_with_new_repair_symbol()`

- **principles for a new source symbol**
  - call `decode_with_new_source_symbol()`

Encoder callbacks

- **called during important events at an encoder**

```
ga_status_t ga_encoder_set_callback_functions (
        ga_encoder_t*  enc,
        void (*source_symbol_removed_from_coding_window_callback) (
                    void*       context,
                    uint32_t    old_symbol_esi),
        void* context_4_callback);
```

- ✓ **each time an (old) source symbol needs to be removed from the coding window, the application's callback function is called**
  - e.g., because the coding windows cannot exceed a certain size
- ✓ **… if the application doesn't care, do not register any function!**

Decoder callbacks

- **called during important events at a decoder**

```
ga_status_t ga_decoder_set_callback_functions (
        ga_decoder_t*  dec,
        void (*source_symbol_removed_from_coding_window_callback) (
                    void*       context,
                    uint32_t    old_symbol_esi),
        void* (*decoded_source_symbol_callback) (
                    void        *context,
                    uint32_t    esi),
        void (*available_source_symbol_callback) (
                    void        *context,
                    void        *new_symbol_buf,
                    uint32_t    esi),
        void*   context_4_callback);
```

What’s next?

- **start open-source codec**
  - asolutly required to challenge this API proposal

- **perhaps:**
  - change `uint32_t    esi`
    - ✓ **to something more flexible (certain FEC Schemes may use an ESI that does not fit into 32-bit words)**

- **not sure it’s appropriate to hardware codecs (e.g., FPGA)**
  - because data transfers are at the symbol level (a symbol may be significantly smaller than a packet)
  - don’t know how to change it!