Design of a generic FEC API

Vincent Roca (Inria)
NWCRG Interim Meeting
Boston, September 2017
What does it mean?

- API compatible with **MDS and non-MDS codes**
- API compatible with **fixed-rate and rateless codes**
- API compatible with **block and sliding window codes**
- API compatible with codes for **end-to-end and in-network re-encoding use-cases**
The **KEY question**: why should we do it?

- **ease** FEC-enabled software development
  - an API provides guidelines
  - a common API reduces dependencies, making it easier to remove a codec and plug another one

- **ease** benchmarking
  - of codes, of codecs, of full solutions

- **ease** development of a future reference FEC codec
  - *(see discussion, later)*
The **KEY question**: why should we do it? (2)

- ease its adoption by SDO (standards developing org.)
  - a key asset for FEC scheme adoption by an SDO!
  - in the mid-term, an open API & open-source free codec is beneficial to everybody…
  - … even to those who already have a commercial offer

- because it’s feasible
  - within NWCRG, several of us developed FEC codecs / APIs
Yes, it’s feasible

- we (Inria) did it
  - public OpenFEC ([http://www.openfec.org/](http://www.openfec.org/)) provides API for Reed-Solomon and LDPC-Staircase
  - commercial, non-public OpenFEC adds support for Raptor and RLC
    - adding sliding window code support (e.g., RLC) required major evolutions of the API

- but
  - we’re not sure it’s the best API
  - we’d like to have an open, standardized solution
Close-up on requirements

What does it mean that the API should be compatible with:

1. MDS and non-MDS codes?

2. fixed-rate and rateless codes?

3. block and sliding window codes?

4. codes for end-to-end and codes for in-network re-encoding use-cases?
**Close-up 1: MDS vs. non-MDS codes**

● “Maximum Distance Separable” or “ideal” code
  ○ with a (k, n) block code, any subset of k encoding symbols out of the n possible enables to recover lost source symbols
  ○ said differently, with a linear code, any sub-system is non-singular

● impact:
  ○ ideal code:
    • decoding with >= k encoding symbols always succeeds
    • one knows in advance what will happen
  ○ non-ideal code:
    • decoding with >= k encoding symbols may or not succeed
    • API should enable a new decoding attempt, with additional symbols, if more are still expected

not too complex to address
Close-up 2: fixed rate vs. rateless codes

- is the number of repair symbols pre-defined (fixed rate) or potentially infinite (rateless)?
  - Reed-Solomon, LDPC, etc. → fixed-rate
  - Raptor, RLC, RLNC, etc. → rateless

- consequences on API:
  - use a function like: `build_repair_symbol()` to produce a new repair symbol each time it’s called
  - avoid using tables of predefined size for encoding symbols
    - main consequences are internal to the codec!

not too complex to address
Close-up 3: block vs. sliding window codes

- does the codec encode on a per-block basis?

- or with a sliding encoding window?
Close-up 3: block vs. sliding window (2)

Impact 1

- **Block**: manage a known set of source symbols
  - a different codec instance for each block:
    create/release_codec_instance()

- **Sliding window**: continuously changing set of source symbols
  - requires a single codec instance for the whole session
  - add_symbol_to/remove_symbol_from_coding_window(), reset_coding_window()
  - a callback symbol_removed_from_coding_window() is needed if the coding window is totally managed by the codec
impact 2

**block decoding**
- can defer decoding until a sufficient number of encoding symbols have been received (e.g., exactly $k$ with MDS codes), then call `finish_decoding()`
- test if a block is decoded: `is_decoding_complete()`

**continuous decoding**
- on-the-fly decoding required with `decode_with_new_source/repair_symbol()`

*in both cases, need a callback to be informed of newly decoded symbols: `decoded_source_symbol_callback()`*
Close-up 4: end-to-end vs. in-network re-encoding

- **end-to-end** means:
  - single encoding and decoding operation
  - a single input flow of source symbols

- **network coding** means:
  - potentially multiple in-transit re-encoding operations, usually a single decoding operation
  - various forms of intra-flow / inter-flow coding
  - several open questions in terms of symbol identification!

major consequences!
**Close-up 4: end-to-end vs. re-encoding (2)**

- **impact: coefficient management differs**
  - **RLNC (in-network re-encoding), sender:**
    - if coefficients are computed in the codec, `get_coding_coefficients()` helps the application to retrieve them and copy them into the repair packet
    - otherwise `set_coding_coefficients()` informs the codec of the coefficients to use before doing encoding
  - **RLNC (in-network re-encoding), receiver:**
    - `set_coding_coefficients()` informs the codec of the coefficients carried in the packet

- **RLC (end2end) [draft-ietf-itsvws-rlc-fec-scheme-00](https://datatracker.ietf.org/doc/draft-ietf-itsvws-rlc-fec-scheme-00/):**
  - coefficient generation internal to the FEC codec from a “key” carried in each repair packet
  - no need for `get/set_coding_coefficients()`; communicating the key to the codec is sufficient
Various additional aspects

● address different decoding algorithms, even for the same code
  ○ the decoding algorithm can impact the approach
    • on-the-fly decoding (e.g., with iterative decoding for Raptor and LDPC, or with sliding window codes) uses a decode_with_new_repair_symbol() function
    • otherwise a finish_decoding() function launches one-time decoding

● rely on callback functions for important events
  ○ decoded_source_symbol() callback (potentially another callback when a source symbol is about to be decoded but calculations not yet performed)
  ○ removed_from_coding_window() callback

● FEC scheme specific control parameters
  ○ set/get_control_parameter()
Next steps

- launch an API design team?
  - who wants to join?
  - focusses on FEC codes only (not protocols)

- work on an I-D
  - will leverage on existing codec development works (various implementations)
    - having different point of views required to improve API quality
  - is it feasible for next IETF?