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

draft-roca-nwcrg-generic-fec-api-01

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

NWCRG, IETF101, London
Status of the work

- I-D updated (yesterday)
  - includes 3 APIs for sliding window codes
    - from Vincent/Jonathan/Morten
    - independently developed
    - there’s running code behind each of them
  - plus link to an open-source, freely usable, C-language, sliding window codec + protocol
    - Cédric (GardiNet): https://gitlab.inria.fr/GardiNet/liblc/
    - implemented differently (not as a standalone codec)

- A few comments after analyzing these APIs…
**Which API? Reminder…**

- the codec is a component of a much larger software

memory management  code rate adaptation management
tunnel management  signaling header creation / parsing
congestion control  transmission / reception
selective ACK creation / parsing

out of scope for this I-D

<codec API> ➔ <low level codec>

- session management()
- encoding/decoding window()
- set/get coding coefficient()
- build coded symbol()
- decode with rcvd src/rep symbol()
Question 1: what type of FEC codes?

- API compatible with different codes?
  - Our position: YES

- API compatible with block and sliding window codes?
  - Our position: ONLY sliding window codes
  - Detail: 2 APIs out of 3 restrict themselves to sliding window. The 3rd one addresses both but result is not fully satisfying. Comes from largely different approaches that could make API way more complex...

- API compatible with end-to-end and in-network recoding use-cases?
  - Our position: YES
Question 2: should the ADU to source symbols mapping be done by the codec?

- **background:**
  - it is FEC Scheme dependent
  - useful to address variable size ADUs
  - it has major impacts (parameters, implementation complexity especially at a receiver)

- **question:** should it be hidden in the codec?
  - **our position:** leave it to the caller
  - **consequence:** API only handles source and repair symbols
Question 3: should the codec initialize and process the source/repair headers?

● background:
  ○ e.g., an additional buffer filled by the codec upon encoding
  ○ hides more details inside the “codec”…
  ○ but it makes the “codec” do more than just the coding part… It’s more a FEC Scheme (code + signalling)

● question: should it be hidden in the codec?
  ○ our position: leave it to the caller
  ○ the codec focusses on what matters: coding/decoding only
Question 4: should the codec bother with timing aspects?

- **background:**
  - the source flow can have timing requirements (e.g., limited validity period). Should the codec know about it?
  - e.g., decoding window vs. linear system size distinction

- **question:** should the codec consider timing req.?
  - **our position:** leave it to the caller
  - let the codec be agnostic of any timing aspect... Timing is an application concept
Question 5: about **hardware requirements**

- is there any specificity to hardware codecs (e.g., FPGA) that should be considered?
  - it was a good IETF 100 comment...
  - …but none of us has any experience

- any opinion?
To sum up

- choosing where to place API is not trivial
  - we design an API to a low level codec, not to a FEC Scheme

next step…
- start with actual design