FECFRAME – extension
Adding convolutional FEC codes support to the FEC Framework

Vincent Roca, Inria, France
Ali Begen, Networked Media, Turkey
https://datatracker.ietf.org/doc/draft-roca-tsvwg-fecframev2/

March 2017, IETF98, Chicago
Note well for FECFRAME-ext + RLC I-Ds

- we, authors, didn’t try to patent any of the material included in this presentation/I-D
- we, authors, are not reasonably aware of patents on the subject that may be applied for by our employer
- if you believe some aspects may infringe IPR you are aware of, then fill in an IPR disclosure and please, let us know
Reminder: this ID is about…

- an EXTENSION of the FEC Framework (or FECFRAME) / RFC 6363
  - goal of FECFRAME is to add AL-FEC protection to real-time unicast or multicast flows

- FECFRAME already part of 3GPP Multimedia Broadcast/Multicast Service (MBMS) standards
  - everybody's interested by the same content at the same time at the same place
    - FLUTE/ALC ⇒ files
    - FECFRAME ⇒ streaming
  - end-to-end latency DOES matter
Reminder: RFC 6363 is limited to Block codes

FEC encoding for this block

src pkt src pkt src pkt src pkt src pkt src pkt

repair repair repair

...erasure recovered after some delay...
Reminder: goal is to extend it to codes based on sliding encoding window

FEC encoding for this window

repair

FEC encoding for this window

repair

FEC encoding for this window

repair

Time
Changes since IETF 97

- as discussed during IETF'97, this is an extension
  - does NOT compromise backward compatibility of FECFRAME
  - does NOT remove any capability to FECFRAME
  - does NOT obsolete RFC 6363

- current I-D
  - keeps the structure of RFC 6363
  - includes additional text specific to convolutional codes

  - I-D is streamlined (18 pages long)...
  - … and easier to read 😊

No technical substantive change, only form changed!
Random Linear Codes (RLC)
FEC Scheme
Convolutional FEC codes for FECFRAME

Vincent Roca, Inria, France

March 2017, IETF98, Chicago
It's a FEC Scheme

• it details:
  ◦ the code specifications: "how do we encode and decode?"
    ⇒ pretty simple
  ◦ the signaling: "how do we identify packets?", "how do we synchronize RLC encoder and decoder?"
    ⇒ a bit more complex

• for lossy networks (e.g., Internet or wireless nets)
  ◦ we call it an "erasure channel"

• based on a sliding encoding window
  ◦ we call it a "convolutional code"
Understanding RLC encoding in 1 minute

- There's a sliding encoding window
  - It slides over the continuous data flow

- You need a repair packet?
  - Compute a linear combination of packets currently in the encoding window

\[
\text{repair}_1 = \alpha_1 \times \text{src}_1 + \alpha_2 \times \text{src}_2 + \alpha_3 \times \text{src}_3
\]
**Understanding RLC encoding in 1 minute...**

- "R" in RLC stands for Random...
  - \( \Rightarrow \) coefficients are chosen *randomly* over a certain Finite Field, using a seed and a PRNG

- send this repair packet plus a signaling header
  - header is called "FEC Repair Payload ID"

---

the seed

- description of the encoding window (ID of 1st symbol + # symbols)
Understanding RLC decoding in 1 minute

- it's all a matter of solving a linear system...
  - each received repair packets adds an "equation"
  - source packets are the "variables"
    - lost packets are "unknowns", others are summed to the constant terms
  - use Gaussian elimination (or something else)

\[
\begin{align*}
\alpha_2 \cdot \text{src}_2 + \alpha_3 \cdot \text{src}_3 &= \alpha_1 \cdot \text{src}_1 + \text{repair}_1 \\
\alpha'_1 \cdot \text{src}_2 + \alpha'_2 \cdot \text{src}_3 &= \alpha'_3 \cdot \text{src}_4 + \text{repair}_2 \\
\alpha''_1 \cdot \text{src}_2 + \alpha''_2 \cdot \text{src}_3 &= \alpha''_3 \cdot \text{src}_4 + \text{repair}_3
\end{align*}
\]

2 unknowns, 3 equations \(\Rightarrow\) high probability to solve the system \(\smile\)
A new FEC Scheme with a big inheritance

- same manner to specify a FEC Scheme as with block codes for FECFRAME
  - same I-D structure
  - except we’re not talking about "blocks" anymore

- similar source packet to source symbol mapping
  - NB: I sometimes erroneously used "packet" instead of "symbol" in previous slides for the sake of simplicity

- similar signaling
  - main difference: two Encoding Symbol ID spaces, one for source, one for repair, instead of a single one
The key question: 
Does it work?
Two types of benefits for conv. codes

● **Reduced FEC added latency**
  
  Intuition:
  ○ Repair packets are quickly produced and they quickly recover an isolated loss

● **Improved robustness for real-time flows**
  
  Intuition:
  ○ Encoding windows overlap with one another which better protects against long loss bursts
  ○ Because of reduced latency, encoding/decoding windows are larger than blocks for block codes
Experimental setup

- compare RLC vs. Reed-Solomon codes

- sliding window code
- ideal block code (max. loss recovery performance!)

- evaluation based on true C-language codecs, using an update of http://openfec.org
  - only transmissions are simulated

- assume CBR transmissions
  - because 3GPP defines CBR channels
  - because it's more realistic (more FEC protection means less source traffic, no congestion control impact)

- use 3GPP loss scenarios representative of mobile use-cases(*)

Experimental setup...

How much repair traffic to achieve the target quality?
Determines:
• block or en/decoding window sizes
• maximum source flow bitrate

target quality: < $10^{-3}$ residual losses

real-time source flow → FEC encoder (RLC or R-S) → CBR channel (100 pkts/s) → loss model → FEC decoder → reconstructed flow

FEC latency budget: 240 ms or 480 ms
Experimental setup...

- take CBR packet scheduling into account

  - RLC

  - two possibilities with Reed-Solomon (depends on implementation details)
    1. block-BEGINNING
    2. block-DURING
Experimental setup...

- take 3GPP mobility scenarios into account
  - vehicle passenger ⇒ losses are "evenly" spread
    4 different average loss rates (1%, 5%, 10%, 20%)
    
    ![Diagram: 120 km/h vehicle passenger, 20% average loss rate]
    
    each "#" indicates a loss

  - pedestrian ⇒ loss bursts
    4 different average loss rates (1%, 5%, 10%, 20%)
    
    ![Diagram: 3 km/h vehicle passenger, 20% average loss rate]
required repair traffic overhead (100% means that repair traffic has same bitrate as source traffic)

- RS-DURING: 39%
- RS-BEGINNING: 28%
- RLC: 23%

average loss rate for the channel

Understanding the following figures for given loss model and latency budget, in order to achieve $10^{-3}$ quality

- Reed-Solomon block-BEGINNING
- Reed-Solomon block-DURING
- RLC

Table II. Parameters for simulations with fixed code rate

<table>
<thead>
<tr>
<th>Channel</th>
<th>3 km/h, 20% loss</th>
<th>3 km/h, 10% loss</th>
<th>3 km/h, 5% loss</th>
<th>3 km/h, 1% loss</th>
<th>120 km/h, 20% loss</th>
<th>120 km/h, 10% loss</th>
<th>120 km/h, 5% loss</th>
<th>120 km/h, 1% loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>14</td>
<td>15</td>
<td>12</td>
<td>16</td>
<td>15</td>
<td>16</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>$dw$</td>
<td>24</td>
<td>31</td>
<td>24</td>
<td>27</td>
<td>30</td>
<td>31</td>
<td>24</td>
<td>30</td>
</tr>
<tr>
<td>$ew$</td>
<td>10</td>
<td>23</td>
<td>18</td>
<td>30</td>
<td>22</td>
<td>30</td>
<td>18</td>
<td>30</td>
</tr>
<tr>
<td>Improvement ratio</td>
<td>$1.64$</td>
<td>$2.43$</td>
<td>$4.14$</td>
<td>$7.38$</td>
<td>$2.00$</td>
<td>$2.64$</td>
<td>$3.50$</td>
<td>$4.14$</td>
</tr>
</tbody>
</table>
Results: min. FEC protection required...

240 ms latency budget for FEC

(a) 240 ms budget, 120 km/h channel
(b) 240 ms budget, 3 km/h channel

RLC is always significantly better, achieving the desired target quality with significantly less repair traffic!
Results: min. FEC protection required...

480 ms latency budget for FEC ⇒ longer block/sliding window sizes

(c) 480 ms budget, 120 km/h channel

(d) 480 ms budget, 3 km/h channel

With a double "latency budget", RLC remains significantly better
And in terms of latency?

- we're dealing with multicast/broadcast, so...
  - many receivers with different channels

  ⇒ decide the worst channel you want to support and maximum repair traffic overhead you can "tolerate"

- use this repair traffic overhead for the (single) multicast data flow
- measure the experienced latency sufficient for a $10^{-3}$ residual loss rate for each supported channel
- compare...
And in terms of latency...

240 ms latency budget for FEC, and **fixed 50% repair traffic** (code rate=2/3)

![Graph showing latency for different conditions](image)

(a) 240 ms budget, 120 km/h channel
(b) 240 ms budget, 3 km/h channel

more channels are supported by RLC, and **the added latency to good receivers is far below the maximum 240 ms latency budget**
Running code

- (non-public) FECFRAME implementation available
  - I did it
  - compliant to 3GPP MBMS
  - successful interoperability tests

- (non-public) FECFRAME-extended implementation almost here
  - I'm still working on it

- (non-public) RLC implementation
  - leverages on our [https://openfec.org](https://openfec.org)
To finish

- our I-Ds are not yet finalized…
  - … but reasonably mature

- we already have a use-case
  - 3GPP standardization activity on Mission Critical Push-To-Talk (audio + video + file)

- Q: WG-Item document?