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

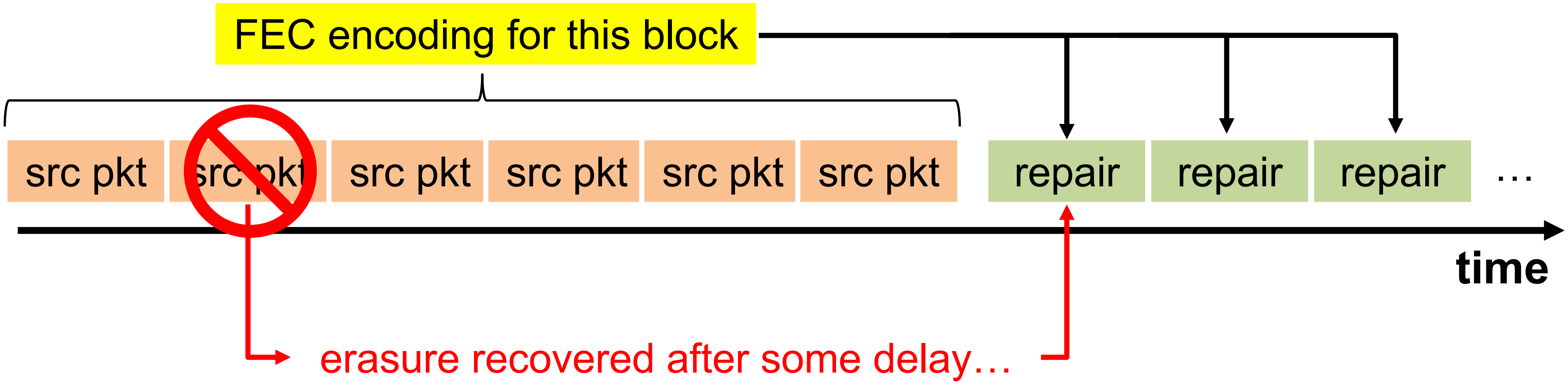
Note well for FECFRAME-ext + RLC I-Ds

- **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

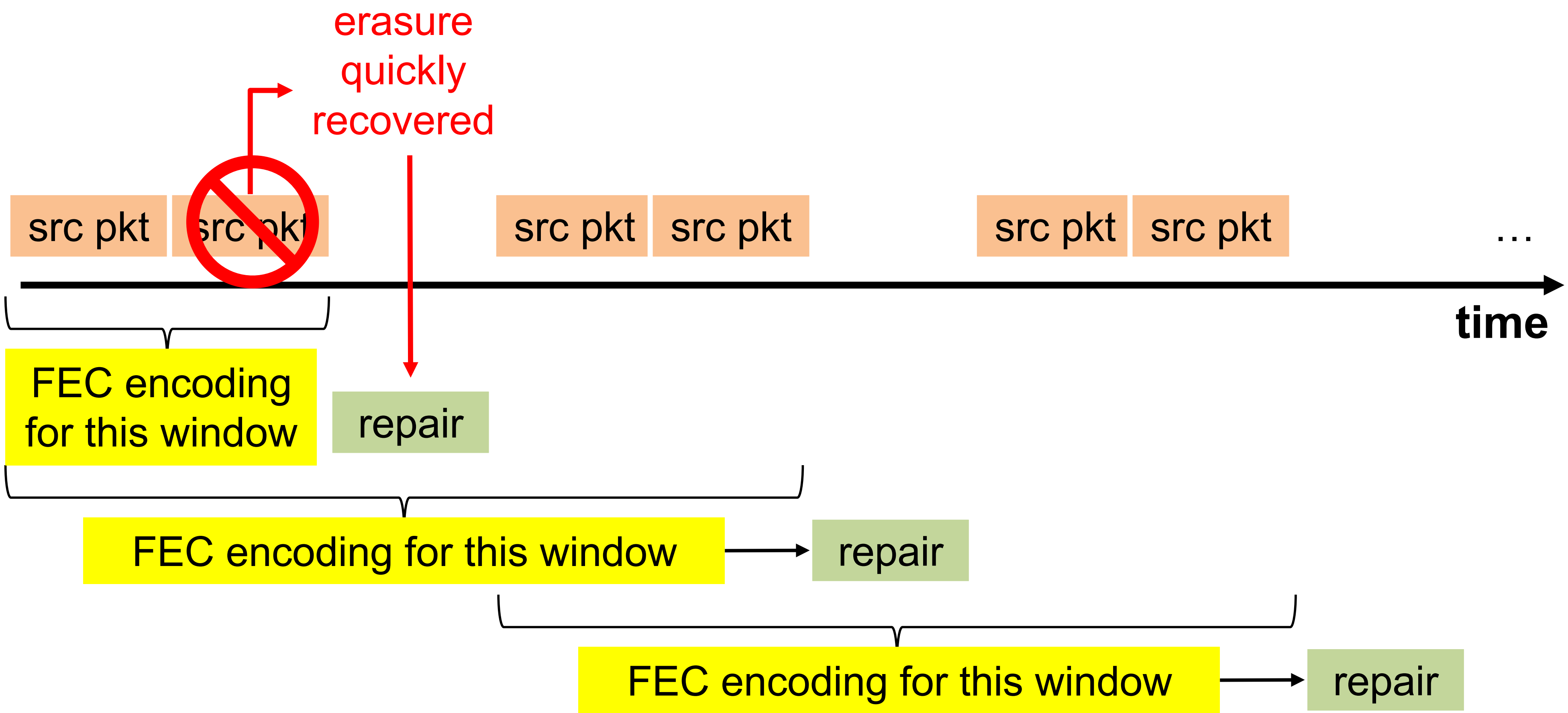
Reminder: this I-D is about…

- 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



Reminder: goal is to extend it to codes based on sliding encoding window



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

https://datatracker.ietf.org/doc/draft-roca-tsvwg-rlc-fec-scheme/

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

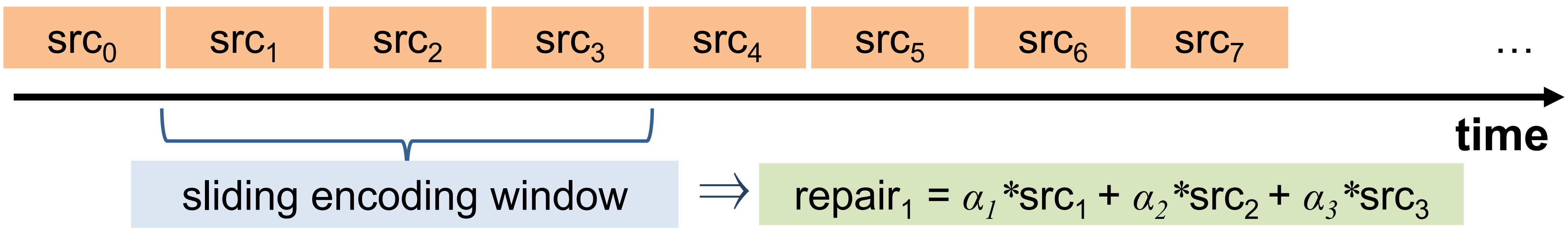
- for **lossy networks** (e.g., Internet or wireless nets)
  - **we call it an *"erasure channel"***

- 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**



Understanding RLC encoding in 1 minute…

- "R" in RLC stands for Random…

  ⇒ **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)

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Repair_Key              |  NSS (# source symbols in ew) |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            FSS_ESI                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+                                                               +
+      repair packet payload follows…                           +
```

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)

lost pkt

lost pkt

received source packets are added to the constant terms

$$\begin{cases} \alpha_2 * \mathrm{src}_2 + \alpha_3 * \mathrm{src}_3 = \alpha_1 * \mathrm{src}_1 + \mathrm{repair}_1 \\ \alpha'_1 * \mathrm{src}_2 + \alpha'_2 * \mathrm{src}_3 = \alpha'_3 * \mathrm{src}_4 + \mathrm{repair}_2 \\ \alpha''_1 * \mathrm{src}_2 + \alpha''_2 * \mathrm{src}_3 = \alpha''_3 * \mathrm{src}_4 + \mathrm{repair}_3 \end{cases}$$

2 unknowns, 3 equations $\Rightarrow$ high probability to solve the system ☺

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

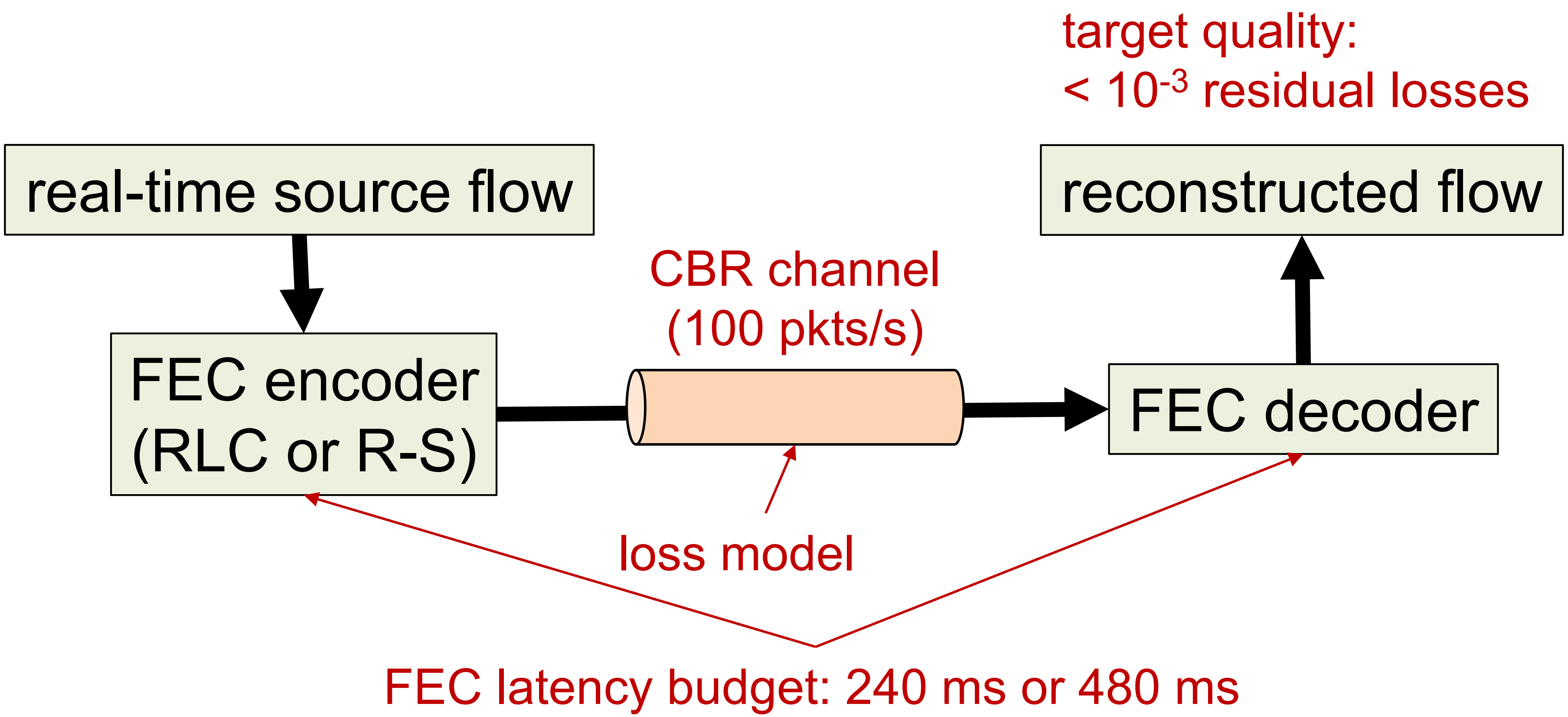
sliding window code

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(*)**

(*) ETSI, "Evaluation of MBMS FEC enhancements (final report)," Dec. 2015, 3GPP TR 26.947 version 13.0.0 Rel. 13

Experimental setup…



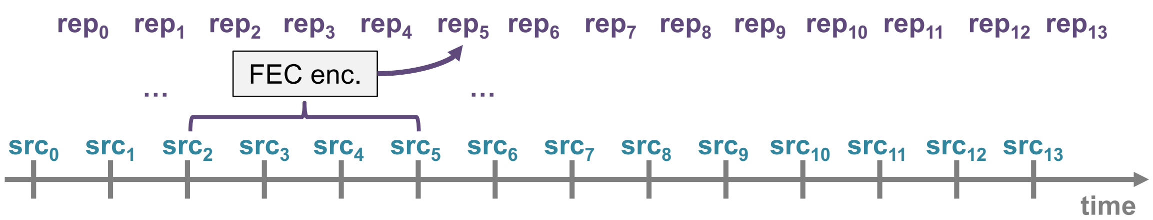
How much repair traffic to achieve the target quality?

Determines:

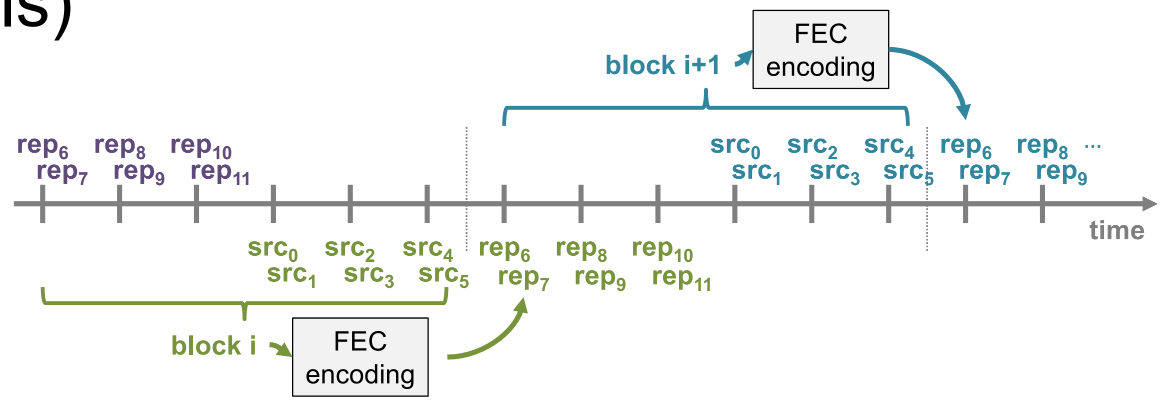
- block or en/decoding window sizes
- maximum source flow bitrate

Experimental setup…

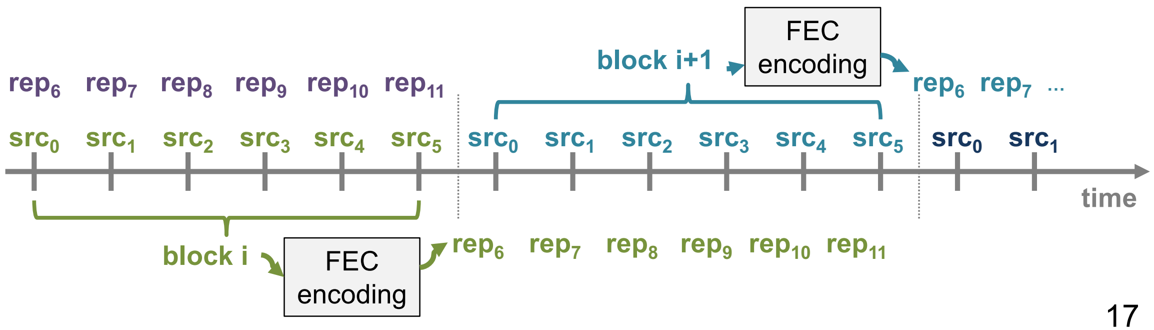
- take CBR packet scheduling into account
  - **RLC**



  - 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**

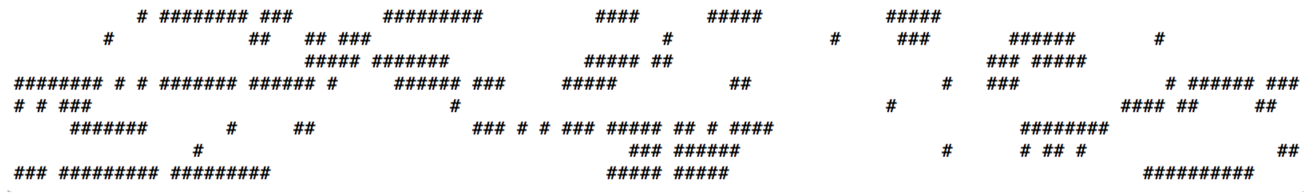
    4 different average loss rates (1%, 5%, 10%, 20%)

each "#" indicates a loss

120 km/h vehicle passenger, 20% average loss rate

  - **pedestrian ⇒ loss bursts**

    4 different average loss rates (1%, 5%, 10%, 20%)

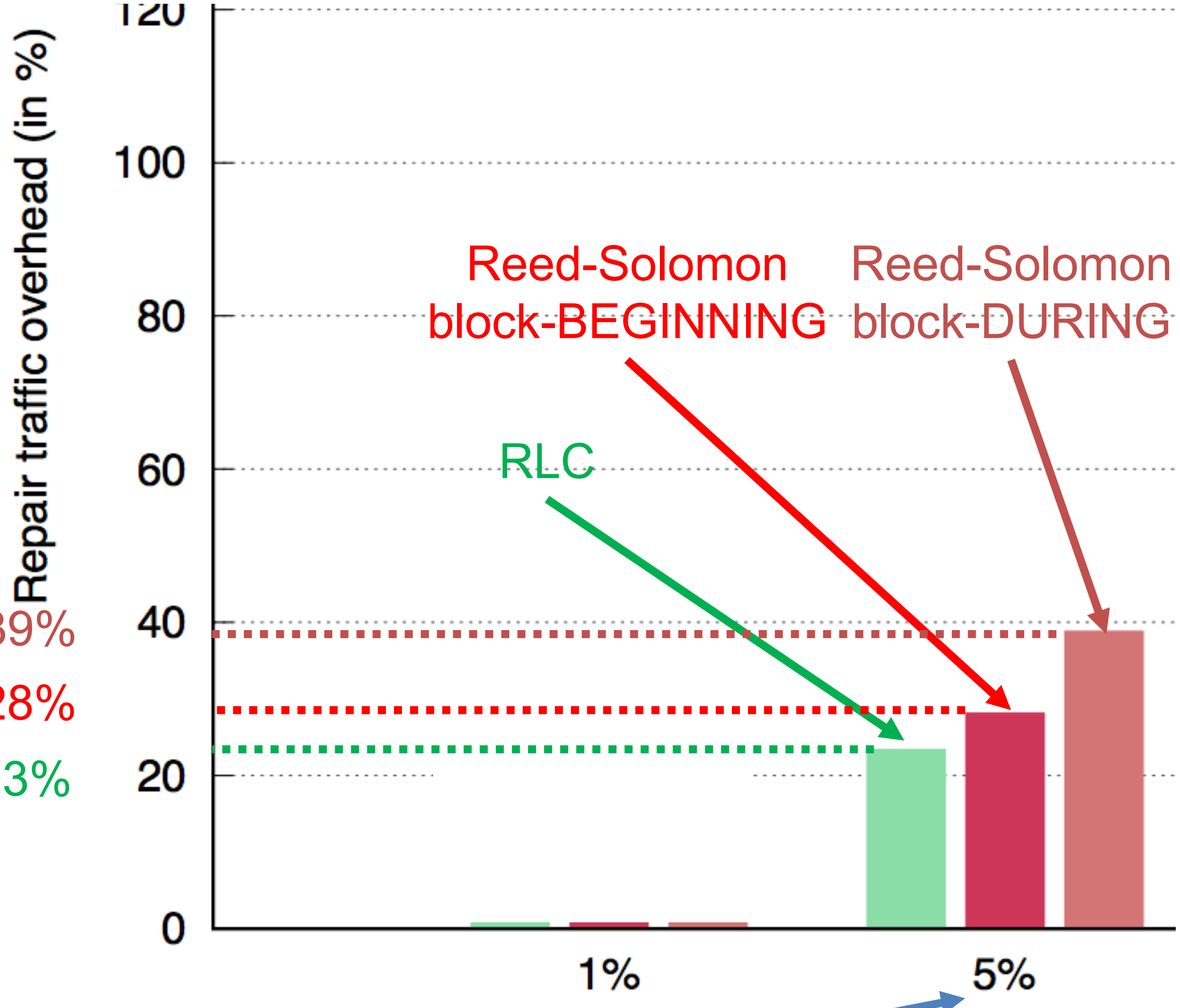


3 km/h vehicle passenger, 20% average loss rate

Understanding the following figures

for given **loss model** and **latency budget**, in order to achieve 10^{-3} quality

required repair traffic overhead (100% means that repair traffic has same bitrate as source traffic)



RS-DURING: 39%

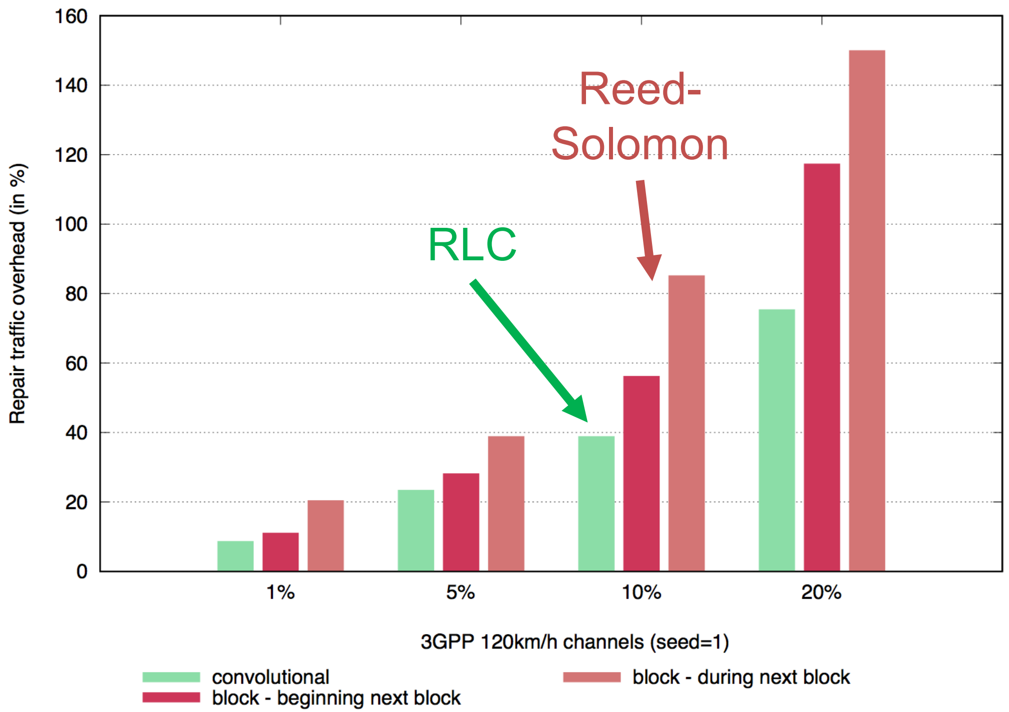
RS-BEGINNING: 28%

RLC: 23%

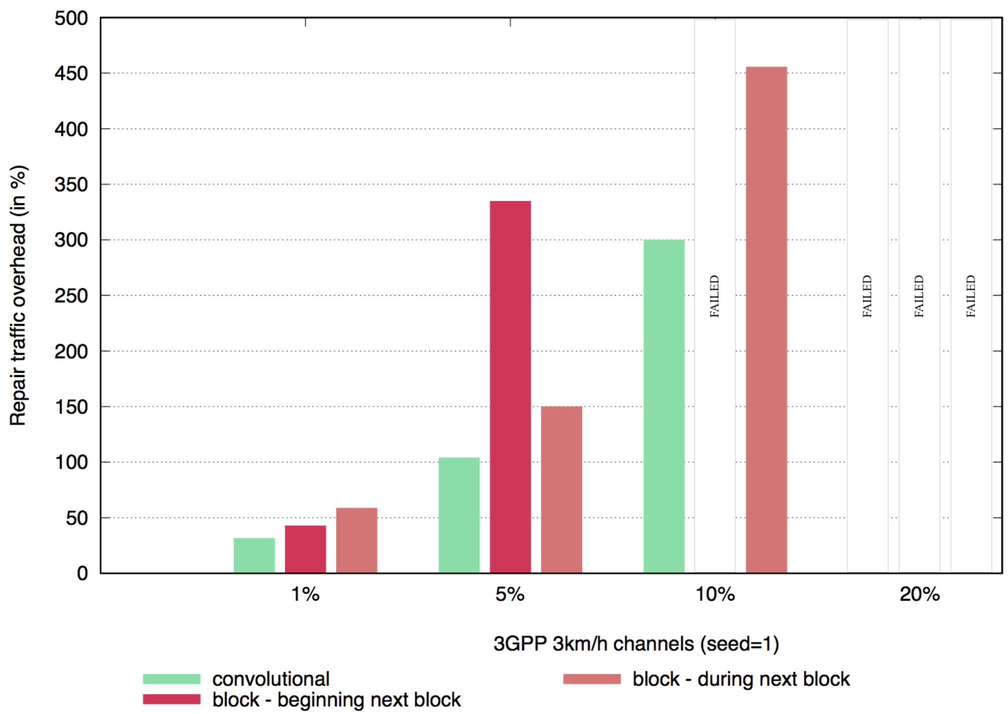
average loss rate for the channel

Results: min. FEC protection required…

240 ms latency budget for FEC



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

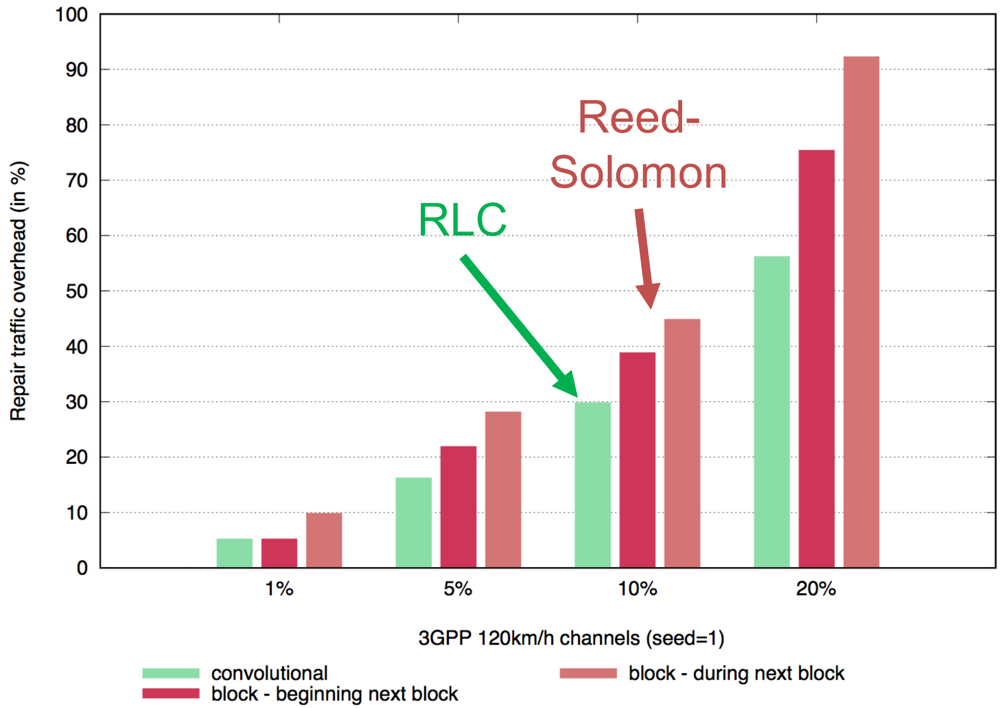


(b) 240 ms budget, 3 km/h channel

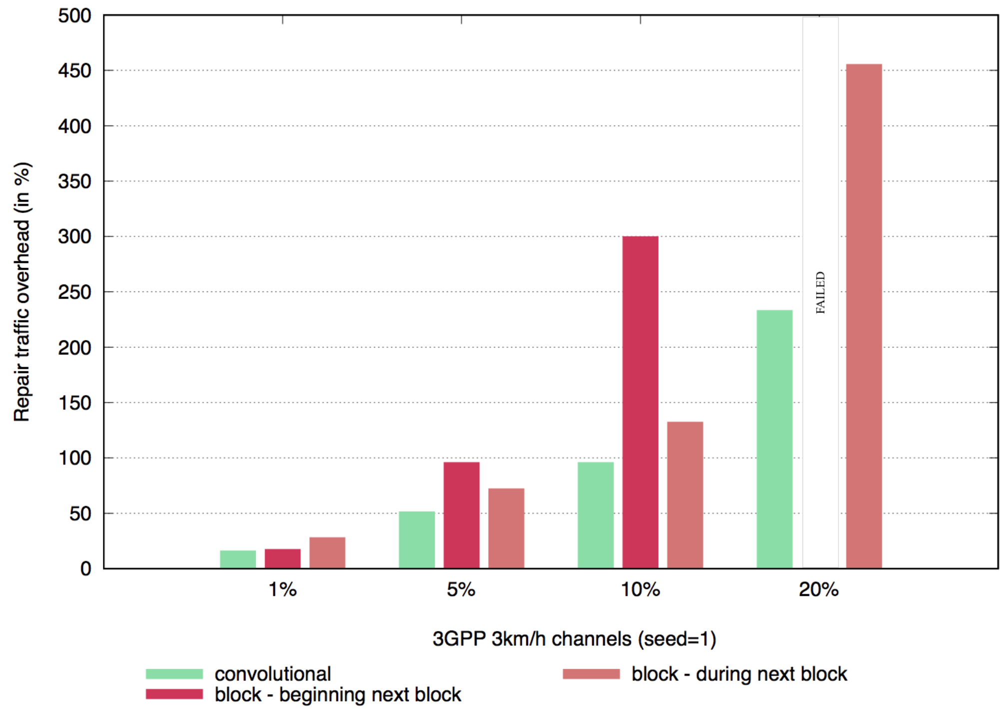
RLC is **always significantly better**, achieving the desired target quality with significantly less repair traffic!

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?

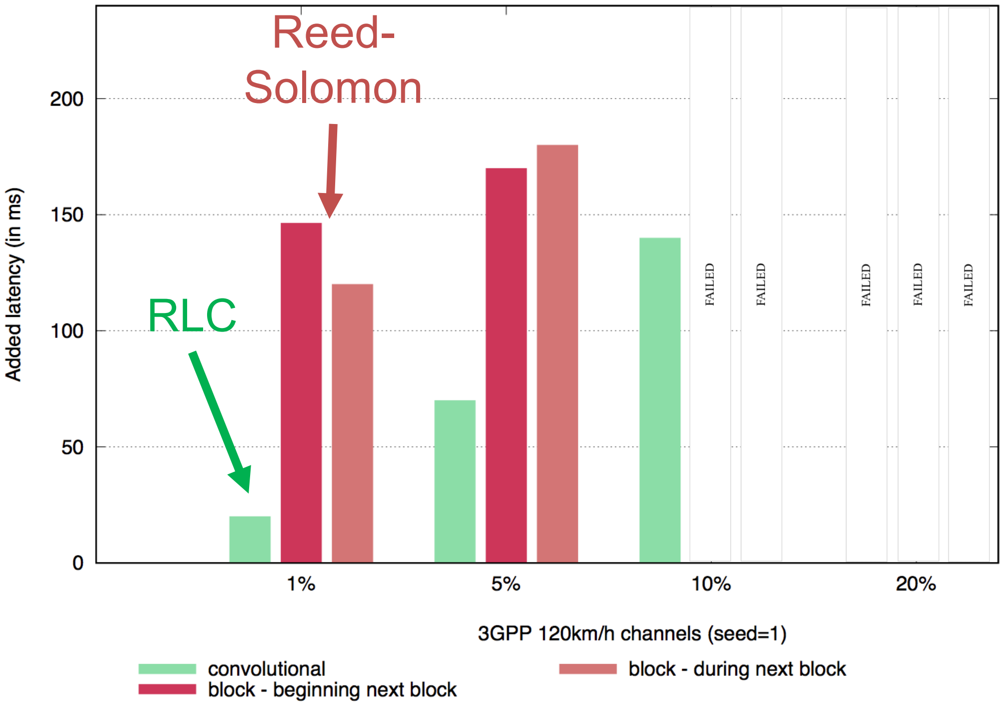
- we're dealing with multicast/broadcast, so…
  - ❍many receivers with different channels

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

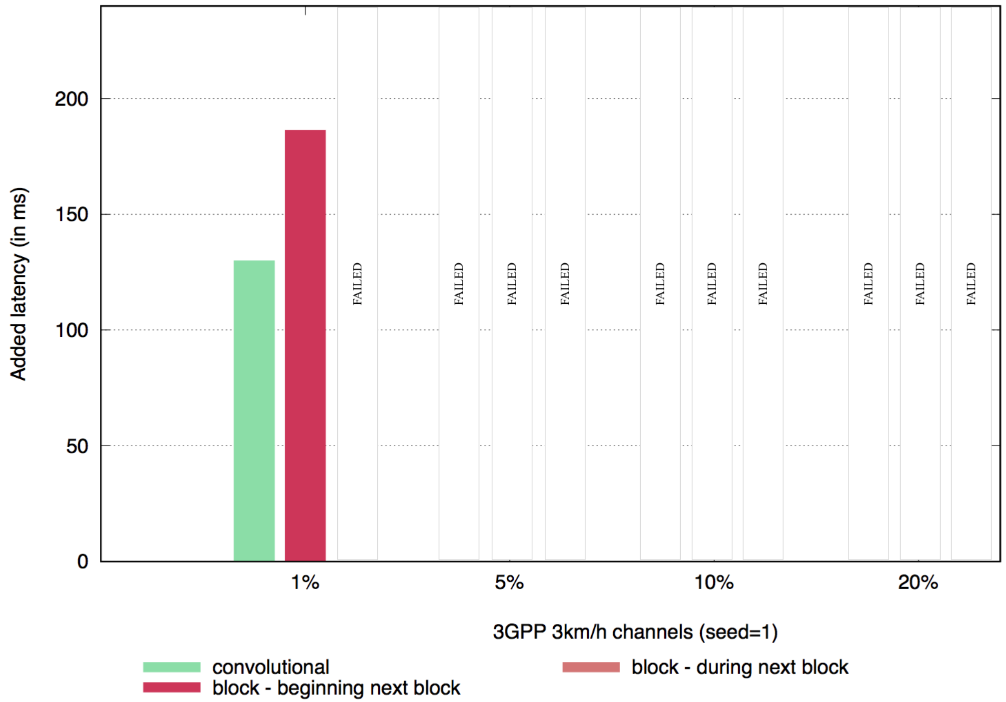
  - ❍**use this repair traffic overhead for the (single) multicast data flow**
  - ❍**measure the experienced latency sufficient for a 10^{-3} residual loss rate for each supported channel**
  - ❍**compare…**

And in terms of latency…

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



(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

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?