



Structured RLC codes: an update

Vincent Roca, Kazuhisa Matsuzono

► **To cite this version:**

Vincent Roca, Kazuhisa Matsuzono. Structured RLC codes: an update. IETF89 - NWCRG meeting, Mar 2014, London, United Kingdom. <hal-00955772>

HAL Id: hal-00955772

<https://hal.inria.fr/hal-00955772>

Submitted on 5 Mar 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Structured RLC codes: an update

Vincent Roca (Inria, France)

Kazuhisa Matsuzono (NICT, Japan) ✕

IETF89, NWCRG meeting

March 6th, 2014, London

(✕ part of the work done while visiting Inria as post-doctorate)

Note well

- **we, authors, didn't try to patent** any of the material included in this presentation
- **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

<http://irtf.org/ipr>

Our proposal and some results in **block** mode... A reminder

For details, see:

<http://www.ietf.org/proceedings/88/slides/slides-88-nwcrq-2.pdf>

Goals (from IETF88)

- design codes that
 - can be used indifferently as **sliding/elastic/block** codes
 - can be used with encoding window/block sizes in **1-10,000s symbols** range
 - keep high enc./decoding speeds and erasure recovery performance in all cases
 - can be used as **small-rate** codes
 - it's not necessarily required, but it simplifies many things
 - focus only on use-cases that need **end-to-end coding**
 - e.g. for FLUTE/ALC, FECFRAME, or Tetrys
 - enable **compact and robust signaling** (essential!)
 - vectors can help for tiny k values but it's unfeasible above
 - use a known function + key (e.g. PRNG + seed)

Two key ideas

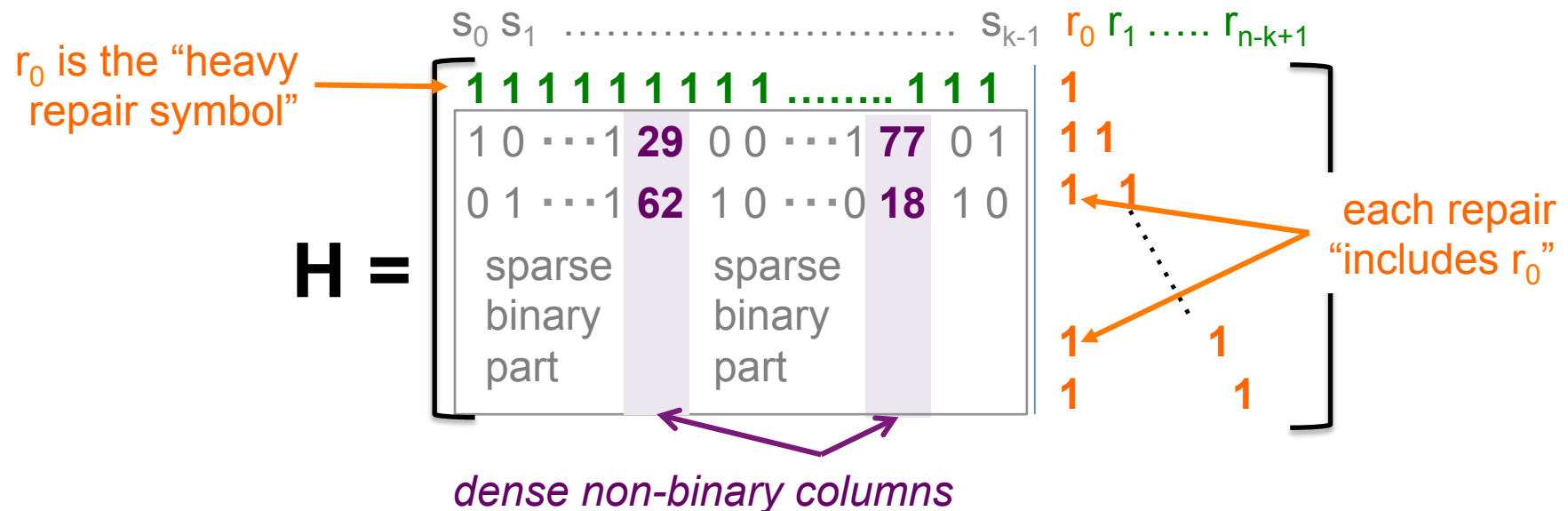
- idea 1: mix binary and non binary coefficients
 - most equations are **sparse** and coefficients **binary**
 - a limited number of columns are **dense** and use **non-binary** coefficients on $GF(2^8)$
- idea 2: add a structure
 - add a **single dense row** (e.g. XOR sum of all source symbols) and make **all repair symbols depend on it**

Let's put ideas 1 and 2 together

- 3 key parameters

- k **block or encoding window size**
- D_{bin} **controls the density of the sparse sub-matrices**
- D_{nonbin} **controls number of dense non-binary columns**
 - $\{D_{nonbin}, D_{bin}\}$ depend on k and a target maximum average overhead

- example: in **block mode**



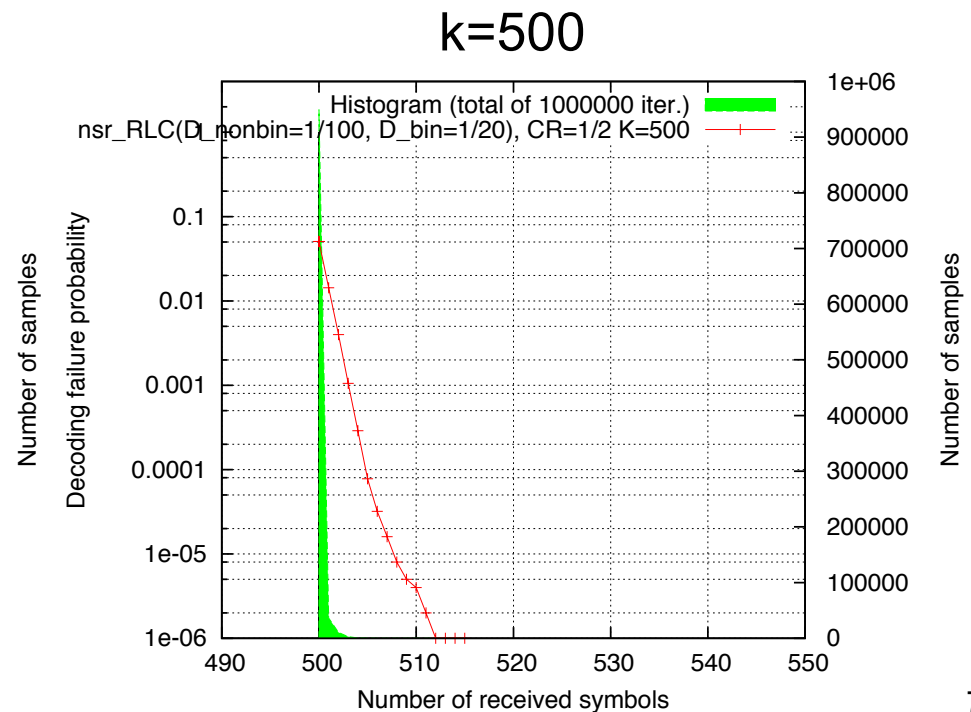
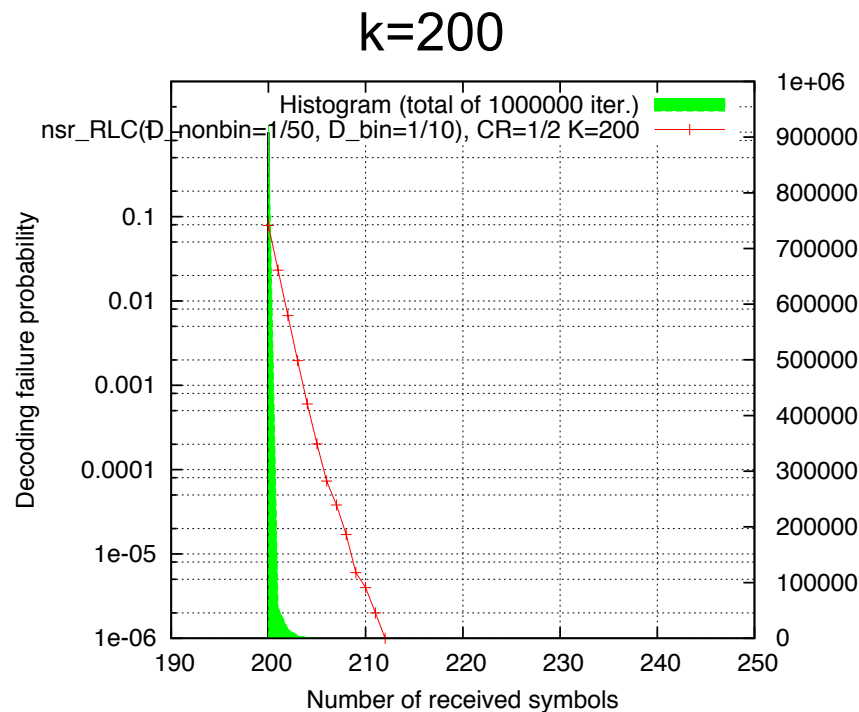
It works well as a **block AL-FEC code**

- it works well on average...

- parameters are chosen so that the average overhead is always below, say 10^{-3} (meaning $k \cdot 10^{-3}$ add. symbols needed)

- and when looking at decoding failure proba. curves

- no visible error floor at 10^{-5} failure probability ☺

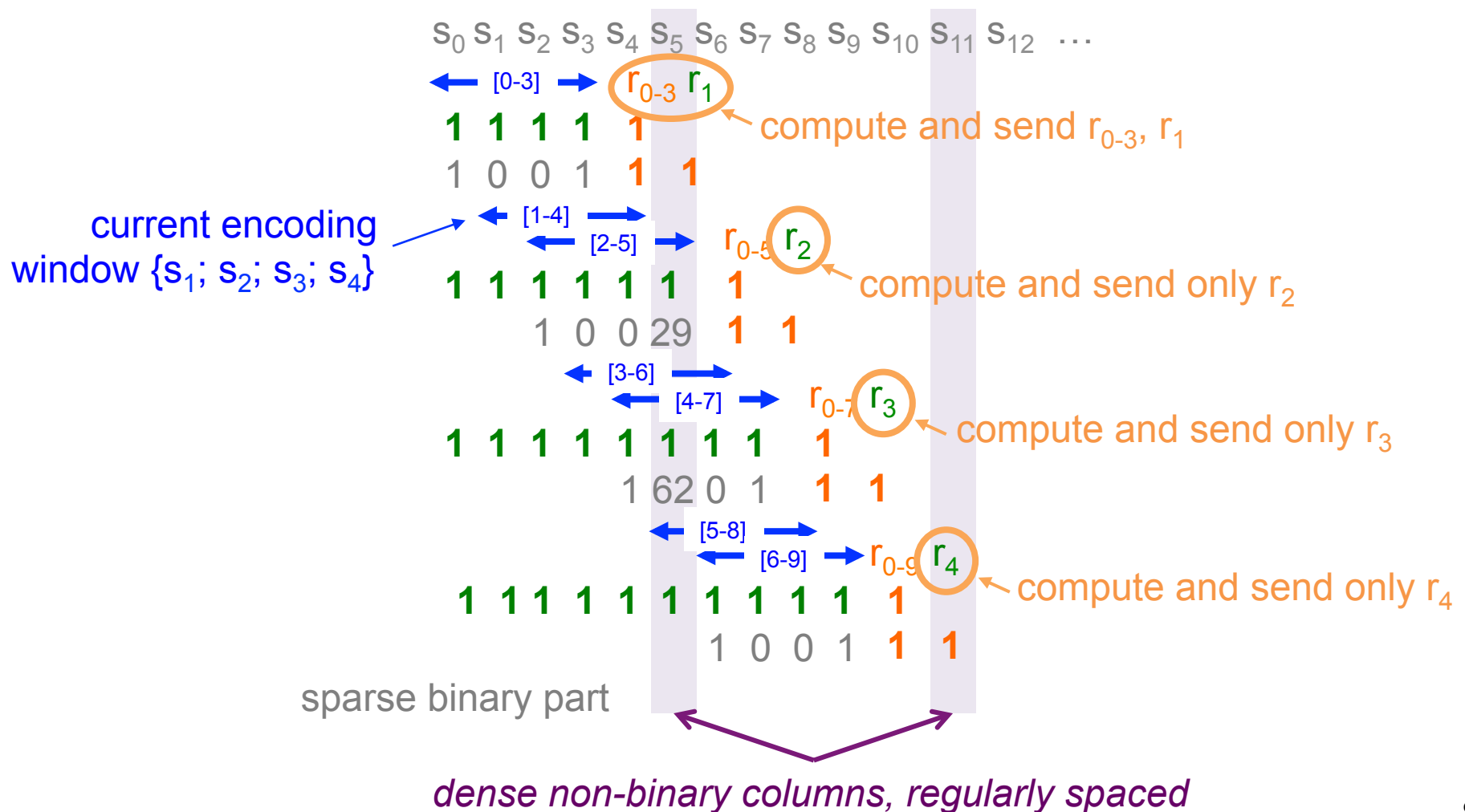


What about **sliding window** mode?

Structured RLC in sliding window mode

- with a fixed length (k) sliding window

○ example: $k=4$, $CR=2/3 \Rightarrow$ send one repair after 2 src symbols



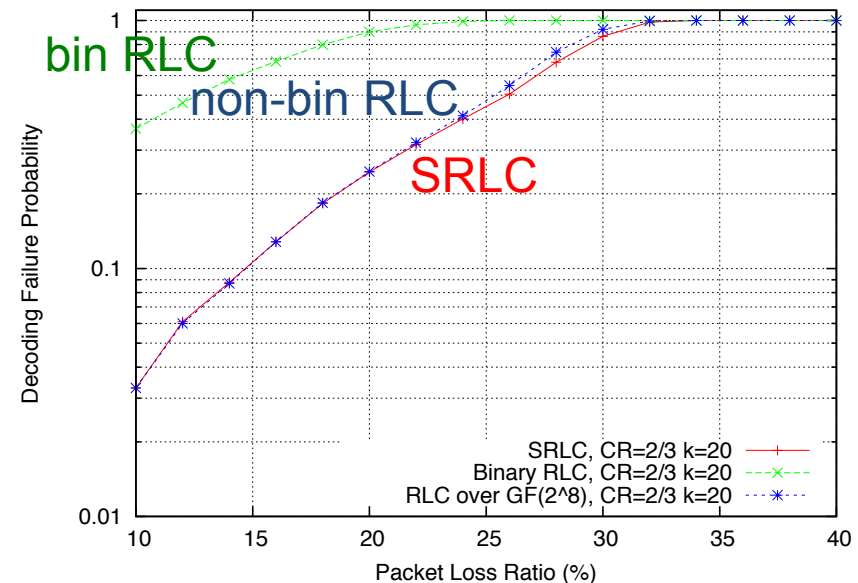
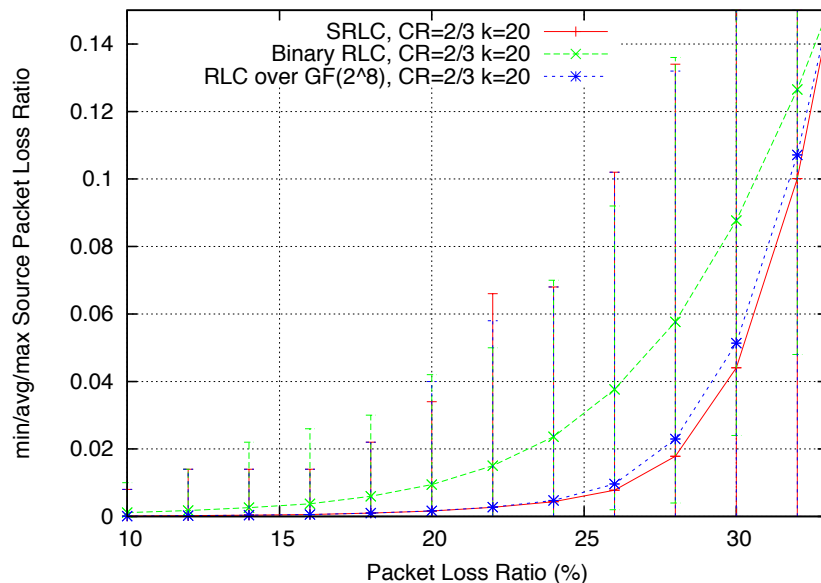
Struct. RLC in sliding window mode (cont')

- about the previous example
 - at session start, we wait k symbols to be available, and then compute and send a few repair symbols to match the target code rate
 - afterwards we mix source and repair symbols in a periodic way
 - each repair that is not a heavy symbol “accumulates” the current heavy repair symbol
 - i.e. the **XOR sum from s_0 to the highest known symbol**
 - **the current sum repair symbol is sent from time to time**
 - the $D_{\text{nonbin}}/D_{\text{bin}}$ are set according to the fixed k value and desired average overhead, using pre-calculated tables

A few experiments

- test conditions (small $k=20$)

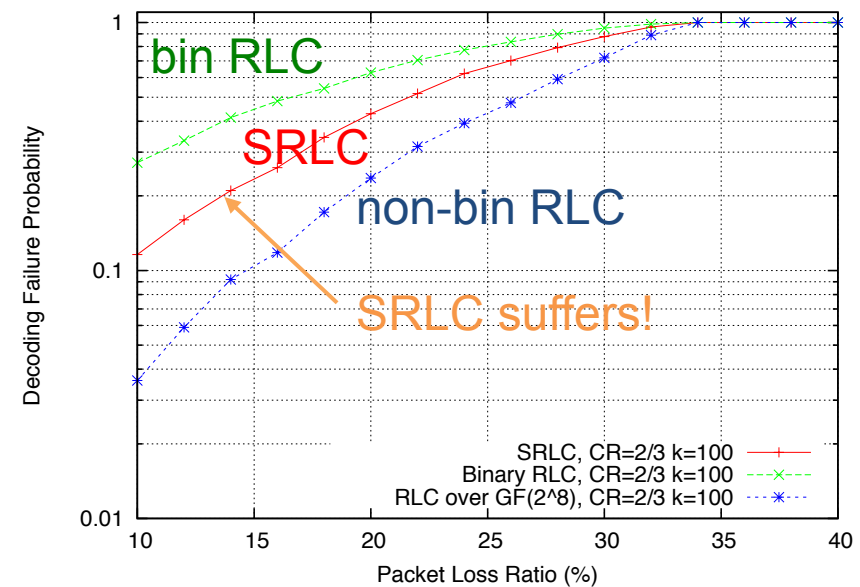
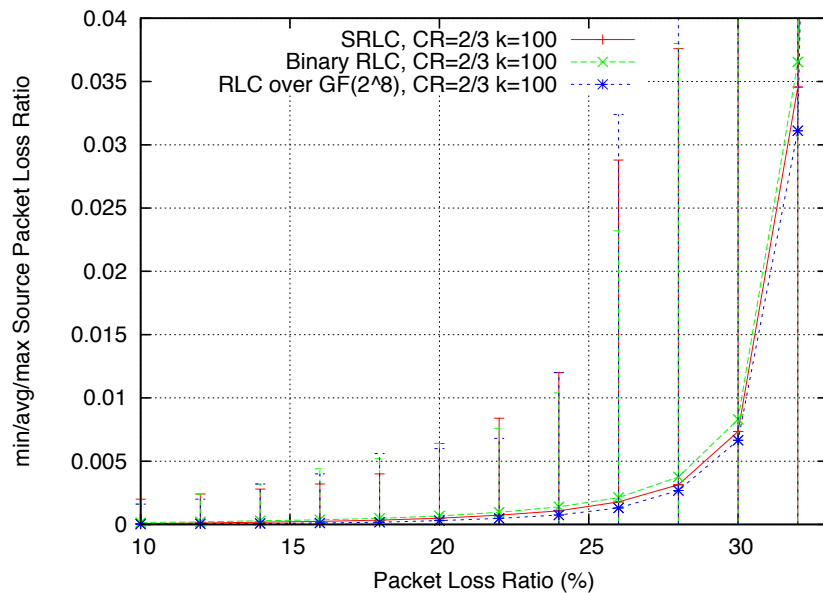
- the encoding window (size $k = 20$) slides over a flow of $25 \cdot k = 500$ source symbols
- $CR = 2/3$, send 1 repair after 2 source symbols
- plot $Pr_{fail}(plr)$ post-repair curves for the whole transmission
 - does not catch the number of non recovered source symbols



- non-bin coefficients are essential
- the heavy repair symbol improves performance WRT. RLC over $GF(2^8)$

A few experiments... (cont')

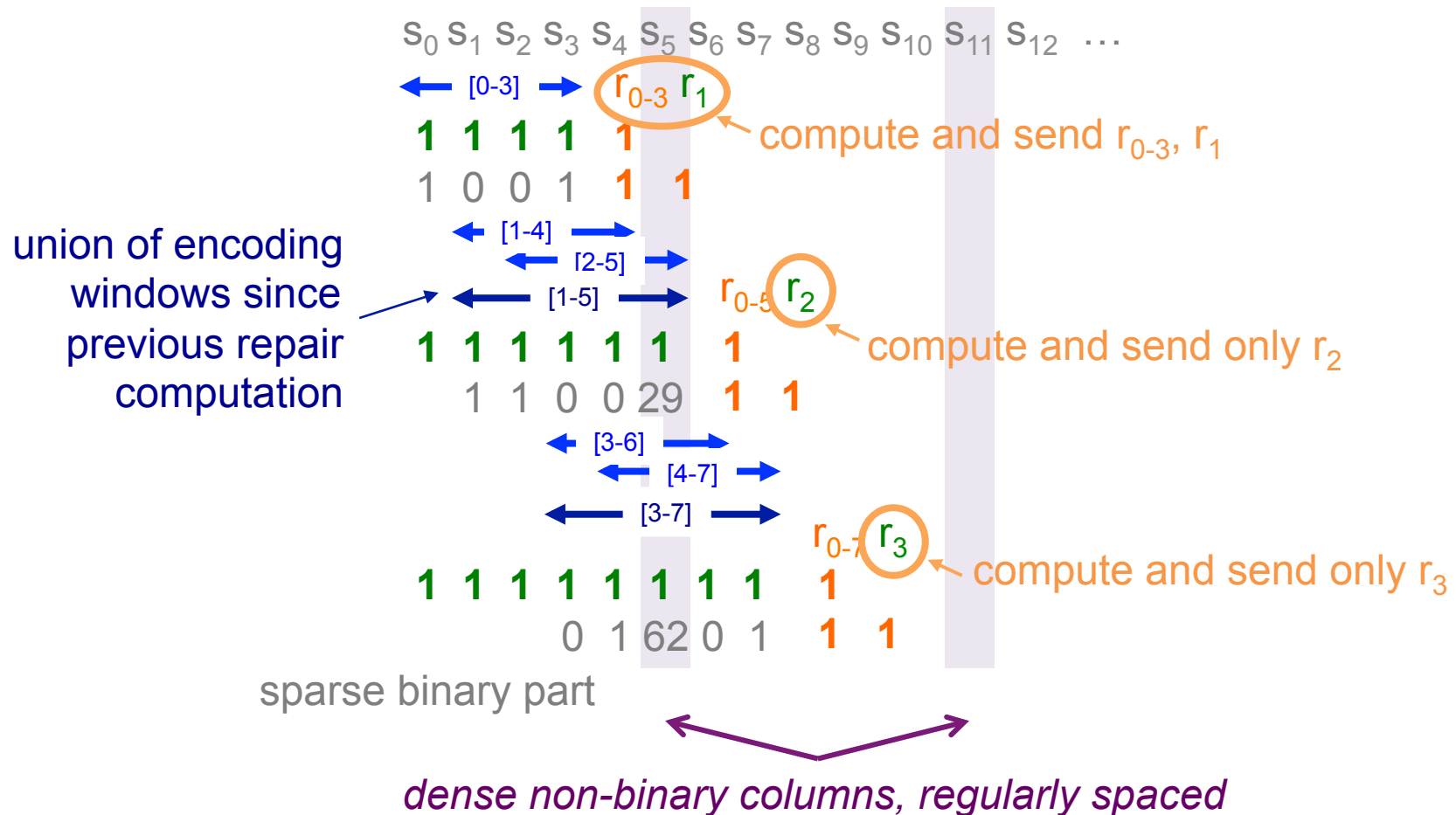
- test conditions (medium $k=100$)
 - the encoding window (size $k = 100$) slides over a flow of $25 \cdot k = 2500$ source symbols
 - $CR = 2/3$, send 1 repair after 2 source symbols
 - plot $Pr_{fail}(plr)$ post-repair curves for the whole transmission
 - does not catch the number of non recovered source symbols



- we reused D_{bin}/D_{nonbin} values computed for the block mode, which is perhaps not appropriate here...

An improvement (under progress)

- consider the union of encoding windows when computing new repair symbols...
 - will make a difference with small k and high CR values



Conclusions

Conclusions

- our proposal tries to take the best of RLC
 - fill in the gap between sliding/elastic window codes and block codes
 - use the right technique (bin vs. non-bin coefficients) at the right time, in the right way
 - find balance between erasure recovery perf. and complexity
- a lot remains to be done yet...
 - how **fast** is it?
 - e.g., compared to our optimized LDPC-Staircase/RS codecs
 - how does it **scale** with k ?
 - e.g., compared to our optimized LDPC-Staircase codec
 - define **signaling** aspects
 - it's a critical practical topic

Thank you!



National Institute of
Information and Communications Technology