Less Latency and Better Protection with AL-FEC Sliding Window Codes: a Robust Multimedia CBR Broadcast Case Study

Vincent Roca, Belkacem Teibi, Christophe Burdinat, Tuan Tran, Cédric Thienot

Less Latency and Better Protection with AL-FEC Sliding Window Codes: a Robust Multimedia CBR Broadcast Case Study

Vincent Roca* Belkacem Teibi* Christophe Burdinat † Tuan Tran † Cédric Thienot †
*Inria, France †Expway, France
{vincent.roca,belkacem.teibi}@inria.fr, {christophe.burdinat,tuan.tran,cedric.thienot}@expway.com

***Abstract*—Application-Level Forward Erasure Correction (AL-FEC) codes have become a key component of communication systems in order to recover from packet losses. This work analyzes the benefits of the AL-FEC codes based on a sliding encoding window (A.K.A. convolutional codes) for the reliable broadcast of real-time flows to a potentially large number of receivers over a constant bit rate channel. It first details the initialization of both sliding window codes and traditional block codes in order to keep the maximum AL-FEC decoding latency below a target latency budget. Then it presents detailed performance analyzes using official 3GPP mobility traces, representative of our use case which involves mobile receivers. This work highlights the major benefits of RLC codes, representative of sliding window codes, that outperform any block code, from Raptor codes (that are part of 3GPP MBMS standard) up to ideal MDS codes, both in terms of reduced added latency and improved robustness. It also demonstrates that our RLC codec features decoding speeds that are an order of magnitude higher than that of Raptor codes.**

***Keywords*—Packet loss channel; AL-FEC codes; sliding window codes; RLC codes; low latency communications; Robust multimedia communications; 3GPP MBMS;**

I. Introduction

Internet and wireless networks are erasure channels where router congestion, severe packet corruptions (e.g., caused by poor reception conditions), or intermittent connectivity are source of packet losses (A.K.A. "erasures"). If retransmissions can improve robustness, situations exist that prevent them: for instance there is no return channel with a unidirectional broadcast network, or scalability issues may prevent using feedbacks (e.g., broadcast sessions to a huge number of receivers), or the extra RTT required by feedbacks and retransmissions may be too large for real-time flows.

This is why AL-FEC codes have become a key component of large scale content distribution systems, relying on broadcast/multicast technologies to efficiently send the same content to a potentially huge number of receivers. This is the case for large scale file content delivery where scalability does matter but not latency. This is also the case for large scale real-time media transfer applications, where both scalability and latency matter. If the first use-case is typically managed by FLUTE/ALC (RFC 6726 [1]), the second use-case is managed by FECFRAME (RFC 6363 [2]). Both protocols are also part of the 3GPP Multimedia Broadcast and Multicast Services (MBMS) standard [3]. A typical example is a popular sport event where multiple video streams from various cameras in the stadium are broadcast in real time to spectators' smartphones, using LTE/4G and WiFi multicast capabilities.

However currently standardized AL-FEC codes for FLUTE/ALC and FECFRAME (simple XOR, Raptor(Q) [4], Reed-Solomon [5] and LDPC-Staircase [6]) are all block codes: the data flow must be segmented into blocks of predefined size. A major limit of these codes is the added latency, caused by the AL-FEC encoding and decoding process, very penalizing with real-time flows. Surprisingly AL-FEC codes based on a sliding encoding window (A.K.A. "convolutional" codes) have not gathered much interest in IETF and 3GPP standardisation activities so far.

In this work we compare sliding window (more precisely Random Linear Codes, or "RLC") and block codes (more precisely Reed-Solomon and Raptor codes) for real-time content broadcast/multicast distribution, using 3GPP MBMS use-cases. We assume, and this is a clear difference with our previous work [7] (an unpublished working document), that the outgoing traffic, source plus repair, is sent over a Constant Bit Rate (CBR) channel. We also assume that AL-FEC encoding and decoding operations are performed end-to-end (e.g., within the broadcast server and the spectators' terminals with the stadium example).

Our contributions are threefold:

- we explain how latency budget constraints of the real-time flow are translated into the AL-FEC codes internal parameters, taking into consideration key implementation aspects;
- we provide a comprehensive comparison of sliding window and block codes, using the official mobility traces used by 3GPP for AL-FEC performance evaluations. These tests highlight the huge benefits in terms of reduced added latency and improved robustness made possible by the RLC sliding window codes over any block code;
- we provide encoding/decoding speed evaluations in an embedded ARM-based board, using state of the art RLC, Reed-Solomon, and Raptor codecs. Results prove that RLC codes offer an order of magnitude decoding speed improvement with respect to Raptor, the current AL-FEC solution of 3GPP MBMS.

These contributions leverages on the authors experience in implementing (e.g., http://openfec.org), standardising (IETF

and 3GPP [8], [9]), commercialising and also deploying 3GPP MBMS solutions (see http://www.expway.com). This contribution is also supported by parallel IETF standardisation activities that explain how to extend FECFRAME to support sliding window codes [10] and specify an RLC FEC Scheme [8]. If we do not detail signaling considerations here (typically information needed to synchronize the RLC encoder and decoder and information carried in the RLC packet headers), the interested reader is invited to refer to [8].

This work does not consider side aspects such as congestion control or cross-traffic (our use-case is restricted to a single CBR flow), nor the dynamic code rate adaptation based on feedbacks (our use-case is limited to a single multicast/broadcast data flow, common to all receivers). This is a deliberate choice, in line with [11], meant to answer the main question: which AL-FEC code features the best performance, all things considered?

This paper is organized as follows: first of all, Section II introduces different manners to use block and sliding window codes and details their impacts on the latency budget and internal parameters; then Section III discusses the target use-case; Section IV introduces in-depth performance analyses; finally we discuss related works and conclude in Section VI.

II. Block or Sliding window: What Changes?

Let us first analyze the impacts of block and sliding window codes in terms of maximum FEC-related latency at a receiver, a key aspect with real-time flows.

A. About Block and Sliding Window Codes

The two families operate in a totally different manner. With block codes, the encoding process is the result of a linear combination of all the source packets (i.e., the packets containing the original content) of the current block. Hereafter we call k this block size (in number of packets). In this work we consider two such codes: Reed-Solomon codes [5], an ideal block code, and Raptor codes [12]. Note that this work does not consider RaptorQ codes because (1) they are not part of the 3GPP MBMS standard and (2) they will not perform better than the ideal block code we considered in terms of loss recovery performance. With the Reed-Solomon codes we consider, encoding requires computing a dedicated generator matrix based on a Vandermonde construction, and then computing combinations of source packet using the matrix entries. With Raptor, encoding is a two-step process: intermediate packets are first computed, then linear combinations of these intermediate packets (LT transform) generate repair packets.

Sliding window codes proceed differently: at any time, one or more repair packets can be produced by computing a linear combination of the source packets currently present in the encoding window that slides over the source packet flow. Hereafter we call ew_size this encoding window size (in number of packets). Sliding window codes differ depending on the linear combination computing. In this work we focus on a very simple solution, Random Linear Codes (RLC), as specified in [8], where coefficients used in linear combinations are chosen randomly over a certain finite field (we restrict ourselves to $GF(2^8)$ in this work). In order to reduce packet overheads and because we do not perform in-transit recoding, a repair packet header only consists of a Pseudo-Random Number Generator (PRNG) seed, as well as a concise description of the sliding window (first source packet identifier and number of packets)[1]. The repair packet header size (64 bits) [8] is therefore similar to that of Reed-Solomon and Raptor codes (48 bits).

B. Source and Repair Packets Ordering with Block Codes

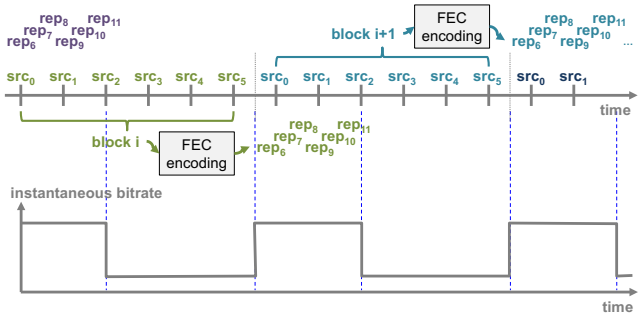
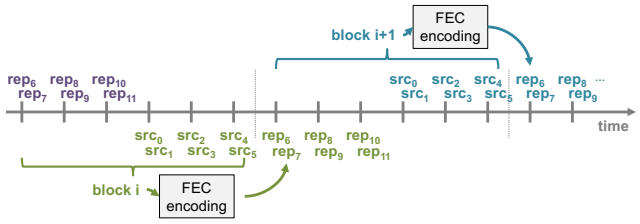
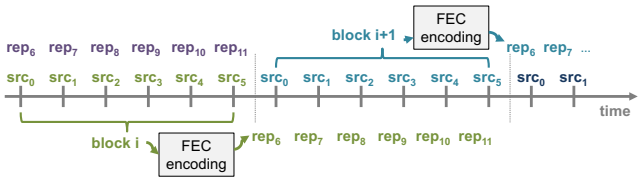


Fig. 1. Case of block codes and VBR transmissions ($cr = 0.5$, $k = 6$).



(a) CBR transmissions, repair packets sent at BEGINNING of next block



(b) CBR transmissions, repair packets sent DURING the next block

Fig. 2. Case of block codes and CBR transmissions ($cr = 0.5$, $k = 6$).

With block codes, repair packets are produced once all the source packets of the current block are known. If source packets can be sent as soon as available (see "block - DURING" scheme, below), the transmission of repair packets necessarily happens afterwards. Several options exist then, depending on the target communication channel:

- repair packets are sent immediately after the source packets, as fast as possible (Fig. 1). It results in Variable Bit Rate (VBR) transmissions as in our preliminary work [7] where repair packets were sent "instantaneously". Being incompatible with our target use-case, this approach will **never** be considered;
- repair packets are sent at the beginning of the following block (Fig. 2-(a)). This approach requires to delay the transmission of source packets to guarantee CBR

[1]This is a difference with RLNC codes often used in network coding where in-transit recoding requires carrying all the coefficients (1 byte each with $GF(2^8)$), in addition to the sliding window description.

transmissions in the outgoing channel. An advantage is that repair packets are available sooner at a receiver. This mode will be called **"block - BEGINNING"**. From an implementation viewpoint, it means that a single FIFO is managed by the FECFRAME sender: after queuing all source packets of a block, the associated repair packets are generated and queued, and so on. Packets are then sent progressively, under the control of a CBR traffic shaper;

- repair packet transmissions are evenly spread during the whole block that follows for CBR transmissions in the outgoing channel (Fig. 2-(b)). This natural approach does not impact source packet transmission and will be called **"block - DURING"**. From an implementation viewpoint, it means that two FIFOs are managed by the FECFRAME sender: one for source packets, one for repair packets. Packets are then sent progressively, under the control of a CBR traffic shaper that serves alternatively both FIFOs;

C. Packets Ordering with Sliding Window Codes

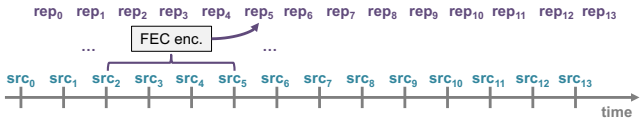


Fig. 3. Case of sliding window codes and CBR transmissions ($cr = 0.5$, $ew\_size = 4$).

With sliding window codes, repair packets creation and transmission are evenly spread and intermixed with source packets. Transmissions are naturally of CBR type (Fig. 3).

D. FEC-Related Latency Budget and Jitter

Because we only focus on FEC-related latency, we can assume, without any loss of generality, that transmissions happen in a constant time. This delay can be safely ignored and is not considered hereafter.

We also consider a real-time source flow (e.g., from a live event), where each source packet has a fixed "latency budget" for FEC-related operations. A lost source packet that cannot be decoded by this delay is useless for the application.

Of course the decoding time is not constant: it is a source of jitter that needs to be compensated by buffering (Fig. 4). If the maximum jitter equals the latency budget, we will see that sliding window codes enable this latency to be close to zero in good reception conditions. If the upper application is informed of good, stable conditions, then the anti-jitter buffering can easily be adjusted accordingly. *Therefore with good, stable reception conditions, we will see that content playout can take place with a very small added FEC-related latency, well below the latency budget.*

E. Impacts on Latency and Internal Parameter Initialization

Let $lat\_budget\_in\_sec$ be the latency budget in seconds for all FEC-related operations. This latency budget does not consider communication latency or other protocol processing latency: these aspects are out of scope of this work. Let

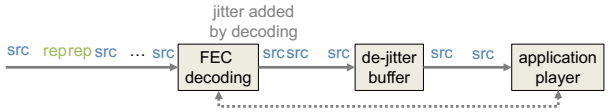


Fig. 4. De-jittering of source packets at a receiver after FEC decoding.

us assume that packets are all of fixed size, $symb\_size$[2]. 3GPP communication channels are always of CBR type and each channel is assigned a predefined bitrate, br. Therefore the latency budget immediately translates into a number of packets:

$$lat\_budget\_in\_pkts = \frac{lat\_budget\_in\_sec * br}{8 * symb\_size} \quad (1)$$

The AL-FEC nature and packet scheduling impact the way this latency budget must be considered:

With the block - BEGINNING mode: In case of losses, decoding block i happens as soon as enough source and/or repair packets are received, which happens during the beginning of block $i+1$. It follows that the latency budget must cover block i (completely even, if source packets are delayed by the sender) and the beginning of block $i+1$. A total of $n + (n-k) = k * (2/cr - 1)$ packets are sent during this latency budget. It follows that:

$$k = \frac{lat\_budget\_in\_pkts * cr}{2 - cr} \quad (2)$$

With the block - DURING mode: In case of losses, decoding block i happens as soon as enough source and/or repair packets are received, which happens during the transmission of block $i+1$. With the worst supported channel, it happens at most at the end of block $i+1$, or earlier with more favorable channels. It follows that the latency budget must cover exactly two blocks. A total of $2 * k/cr$ packets are sent during this latency budget: k/cr for the source and repair packets of block i, plus $k * (1/cr - 1)$ repair packets for block $i-1$, and k source packets for block $i+1$. It follows that:

$$k = \frac{lat\_budget\_in\_pkts * cr}{2} \quad (3)$$

With sliding window codes: FEC decodings at a receiver happen regularly since repair packets are intermixed with source packets. Transmissions are by design of CBR type.

[2]We assume each packet contains one symbol, the AL-FEC processing unit, and we ignore the notion of symbols altogether for simplicity purposes.

Common	
$symb\_size$	packet size, assumed fixed (in bytes)
br	CBR channel bitrate (in bps)
$lat\_budget\_in\_sec$	latency budget expressed in seconds
$lat\_budget\_in\_pkts$	latency budget expressed in packets
cr	AL-FEC coding rate
plr	packet loss rate on the erasure channel
Block codes	
k	block size (in packets)
sliding window codes	
ew_size	max. encoding window size at a sender (in packets)
dw_size	max. decoding window size at a receiver (in packets)
	or # of source packets in L.S. that didn't time-out yet
ls_size	max. linear system size (width) at a receiver (in packets)

TABLE I. TERMINOLOGY AND NOTATIONS USED IN THIS WORK.

The decoding window size, dw_size, is what determines the maximum decoding latency experienced by the receiver with the worst channel [7]. During the latency budget, a total of dw_size/cr source and repair packets are sent. It follows that:

$$dw\_size = lat\_budget\_in\_pkts * cr \quad (4)$$

With a memoryless channel, [7] suggests using $ew\_size = dw\_size/2$. If there are loss bursts, as is the case in the present work, increasing this value is favourable. In any case, this choice has no impact on the maximum decoding latency experienced by the worst receiver which is controlled by dw_size. In the following we choose:

$$ew\_size = dw\_size * 0.75 \quad (5)$$

III. The Robust Multimedia CBR Broadcast Use Case

We consider a real-time source flow (e.g., from a live event) where each source packet has a fixed "latency budget" for FEC-related operations: 240 or 480 ms. A lost source packet that cannot be decoded by this delay is useless for the application. If these values seem arbitrary, they are in line:

- with the need to have a sufficiently long protection period to address the desired robustness (we will see that a 480 ms latency budget is absolutely needed for harsh channels);
- at the same time, with the need to limit the global end-to-end latency.

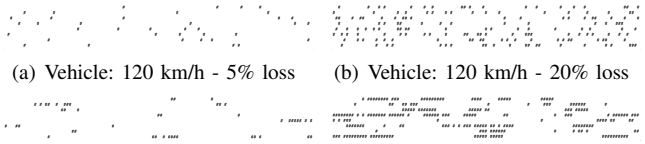
Then the various AL-FEC codes are evaluated over a CBR channel, re-using the configuration specified by 3GPP for the "Selection and characterisation of application layer Forward Error Correction (FEC)" ([11], table 8). More precisely we focus on:

- LTE MBMS Bearer bitrates: 1.0656 Mbps
- Radio Link Control - SDU size: 1332 bytes
- Radio Link Control - SDU frequency: 10ms

meaning the CBR flow (e.g., from the FECFRAME sender) sent on the LTE channel is composed of 100 pps, each of size 1332 bytes, for a bitrate slightly above 1 Mbps.

Since mobile terminals (sender or receiver) are considered, we rely on the official 3GPP mobility scenarios ([11] tables 6 and 7). Eight different channel types are provided, either for a vehicle passenger (120 km/h) or a pedestrian (3 km/h), with four different loss rates, from 1% to 20%. For each scenario, four instances generated with different seeds are provided. Fig. 5 highlights how different the channels are: a pedestrian will observe long bursts of losses (long periods behind obstacles) while a vehicle passenger will observe evenly spread losses. Experiments in Section IV will confirm that pedestrian channels are harsh environments that require strong protection.

Finally, this work does not consider side aspects such as congestion control or cross-traffic (our 3GPP target use-case is restricted to a single CBR flow), nor dynamic code rate adaptation (our 3GPP target use-case is limited to a single multicast/broadcast data flow). This is a deliberate choice in order to answer the key question: which AL-FEC code features the best performance, all things considered?



(a) Vehicle: 120 km/h - 5% loss (b) Vehicle: 120 km/h - 20% loss

(c) Pedestrian: 3 km/h - 5% loss (d) Pedestrian: 3 km/h - 20% loss

Fig. 5. Excerpts of 3GPP mobility channels for 5% and 20% loss rates and either a vehicle passenger (120 km/h) or pedestrian (3 km/h). Each trace indicates which packet is correctly received (space sign) or lost ("#" sign) over the time (approx. 1,0000 packets are considered for each trace). The different types of channels (losses regularly spread vs. in bursts) are clearly visible.

IV. Performance Evaluation with 3GPP Channels

A. Performance Evaluation Methodology

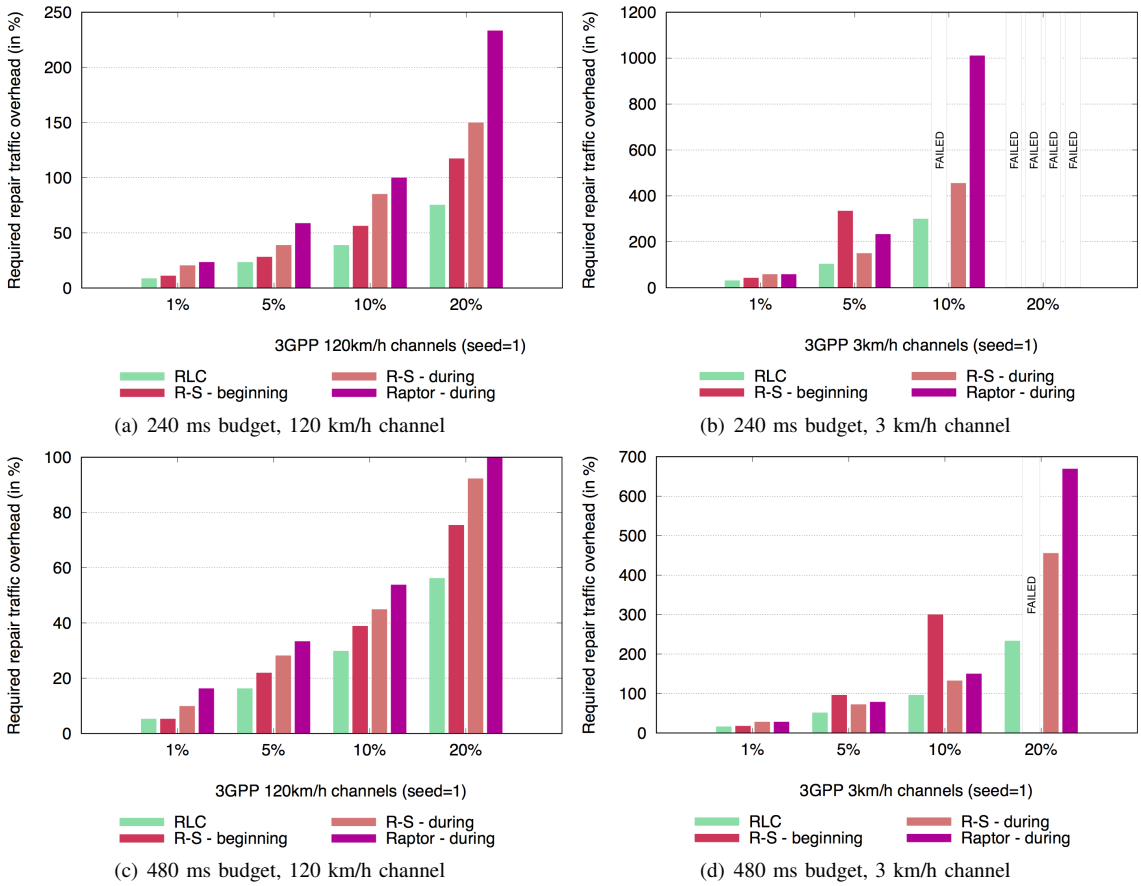
The block and sliding window codes performance have been fairly compared, using our OpenFEC (http://openfec.org) performance evaluation environment and C-language full-featured codecs. We chose Reed-Solomon as a representative of ideal block codes, and Random Linear Codes (RLC) for sliding window codes. We only considered Finite Field $GF(2^8)$ in both cases and did not try to experiment with any RLC variant. We also chose a linear system large enough (i.e., $ls\_size = 60$) for the "decoding beyond maximum latency" optimisation [7] to behave optimally.

The loss recovery performance tests leverage on the `eperftool` simulator of OpenFEC. This tool consists of three steps. The first step consists in generating all the packets of the sender, performing actual AL-FEC encoding using the above codecs. The second step consists in defining the packet transmission order (e.g., to simulate the block-BEGINNING or DURING modes) and applying packet losses on this packet flow. Being simulated, these transmissions are fully reproductible. Finally the receiver loops over the received packet flow and attempts decoding when appropriate. All latency measurements are simulated, each packet transmission corresponding to $10ms$ (Section III). For reliable results, performance metrics are always calculated over long sessions: in each test $1,000,000$ source packets are sent, which represents several $10,000$ of blocks for instance.

Because Raptor codes (also known as "Raptor10") are already part of 3GPP MBMS specifications, they are considered by the following tests. Being non-ideal block codes (they are only asymptotically good as the block size increases), their loss recovery performance is behind that of Reed-Solomon codes. In order to limit this performance penalty, we consider the most favourable recommended Raptor configuration, $G = 10$, meaning that each packet is sub-divided into 10 sub-packets in order to artificially increase the block size, in line with Raptor specifications [4], [3]. If a higher G value would improve the loss recovery performance, it would however remain behind that of Reed-Solomon (we will see RLC is anyway an order of magnitude more efficient) and Raptor encoding/decoding speeds would also be negatively impacted (whereas RLC is already significantly faster).

B. Required AL-FEC Protection to Achieve a Target Quality

Let us first evaluate, for each 3GPP mobility channel and a 240 ms or 480 ms latency budget, the required protection



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

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

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

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

Fig. 6. **Required AL-FEC protection** to achieve 10^{-3} residual loss quality with a 240ms or 480ms latency budget, depending on the mobility scenario. A missing bar indicates a failure to achieve the target quality.

to achieve the target quality: a residual loss rate after FEC decoding at most 10^{-3} (at most 1 missing source packet out of 1000). The protection is measured by the repair traffic overhead, which is linked to the code rate:

$$repair\_ovherhead = (\frac{1}{cr} - 1) * 100$$

For instance, $cr = 2/3$ corresponds to a 50% repair overhead (50% traffic in addition to source traffic), and $cr = 0.5$ to a 100% repair overhead (traffic is doubled). From Fig. 6, sliding window codes always yield the best results, no matter the block code variant considered nor the channel. The target quality is always achieved (except for the most difficult channel), and with less repair overhead.

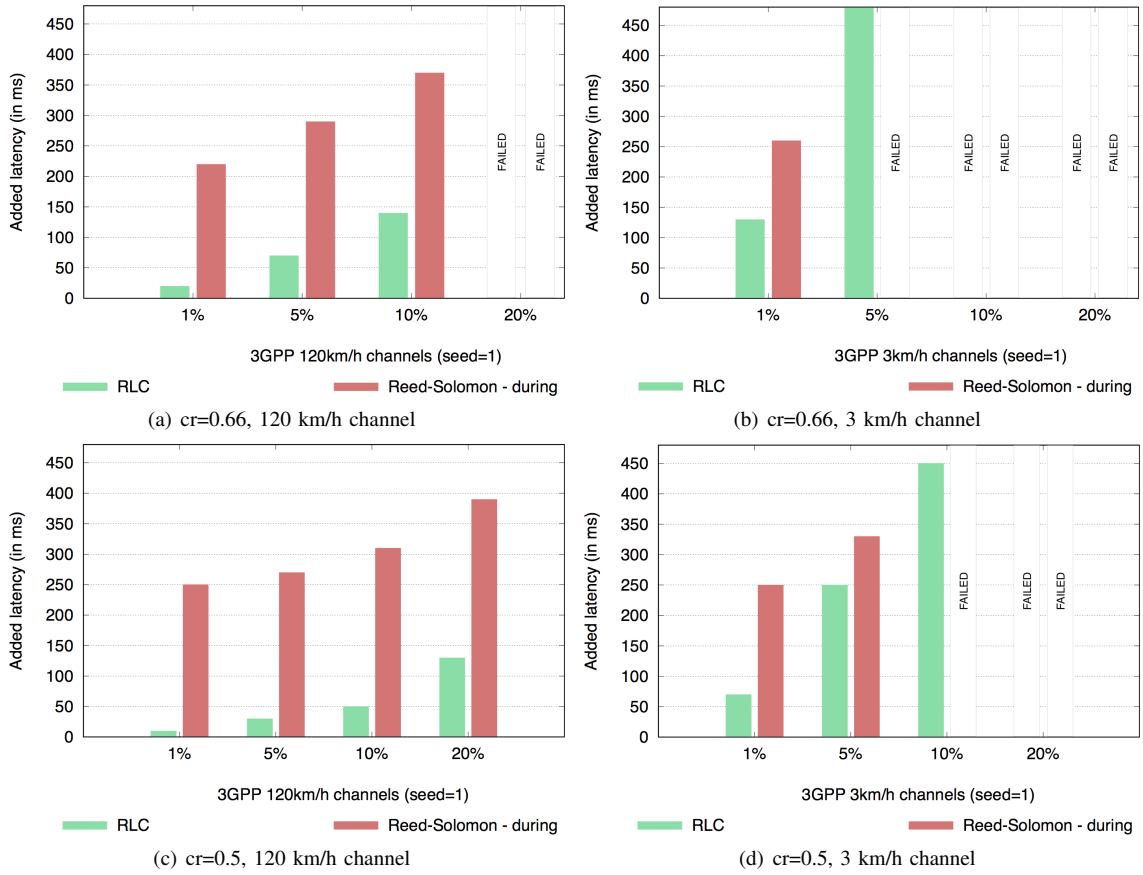
Channel	RLC	R-S - BEGINNING	R-S - DURING
120 km/h, 1% loss	dw = 45, ew = 33	k = 43	k = 21
120 km/h, 5% loss	dw = 41, ew = 30	k = 33	k = 18
120 km/h, 10% loss	dw = 36, ew = 27	k = 27	k = 16
120 km/h, 20% loss	dw = 30, ew = 22	k = 19	k = 12
3 km/h, 1% loss	dw = 41, ew = 30	k = 35	k = 18
3 km/h, 5% loss	dw = 31, ew = 23	k = 16	k = 13
3 km/h, 10% loss	dw = 24, ew = 18	k = 6	k = 10
3 km/h, 20% loss	dw = 14, ew = 10	FAILURE	k = 4

TABLE II. EVOLUTION OF THE K, DW, AND EW PARAMETERS ACROSS THE MOBILITY SCENARIOS, WITH A 480MS LATENCY BUDGET.

To further analyze the situation, we now study how k, dw, and ew evolve as the mobility scenario becomes more difficult. Table II lists them in case of a 480 ms latency budget. With block codes, as more repair traffic is needed, the fixed budget latency requires the block size to be progressively reduced, until it reaches ridiculously small values (e.g., $k = 4$). Interestingly, the "block - BEGINNING" mode behaves well with good channels (i.e., 120 km/h channels, Fig. 6(a) and 6(c)), but becomes counterproductive with serious loss bursts and finally fails to achieve the target quality (Fig. 6(b) and 6(d)). This is in line with intuition since the "block - BEGINNING" mode concentrates source and repair packets of a given block to a smaller time period compared to the other mode, and therefore is more rapidly impacted by long loss bursts. With sliding window codes, the encoding window is significantly larger which favors robustness, as Fig. 6 highlighted.

Note that all the above results have been achieved with 3GPP channels for $seed = 1$. Three more channels are available for each scenario (seeds 2, 3 and 4). Since they yield very similar results, we do not show them.

From the above tests it appears that the "block - BEGINNING" mode should not be used (i.e., two two FIFOs should always be used by the sender, see Section II-B), and we therefore ignore this configuration from the following tests. It also appears that Raptor codes, that behave badly with small blocks (even if using $G = 10$ mitigates this impact), are not appropriate. However we keep them into consideration, being part of the current 3GPP MBMS standard, as a reference.



(a) cr=0.66, 120 km/h channel

(b) cr=0.66, 3 km/h channel

(c) cr=0.5, 120 km/h channel

(d) cr=0.5, 3 km/h channel

Fig. 7. **Experienced latency** of RLC and Reed-Solomon - DURING codes, for code rates 0.66 (top) and 0.5 (bottom), and 480 ms latency budget.

C. Experienced Robustness and Latency Once the Code Rate is Chosen

In Section IV-B we determined the required AL-FEC protection to achieve a certain quality. Since we focus on multicast/broadcast transmissions, a single data stream will be used that should satisfy most of the receivers, if not all. A strategic choice is needed: can we consider all or only a subset of the mobility scenarios, and at what cost? Answering this question means choosing a certain code rate. We believe that a code rate below 0.5 is unreasonable (this is in line with [2]), even if the worst channels cannot be supported (a larger latency budget would be needed). Therefore we focus on:

- code rate 0.66: the CBR flow consists of $2/3$ of source traffic and $1/3$ of repair traffic;
- code rate 0.5: the CBR flow is equally divided into source and repair traffic.

Other parameters are derived by Equations (2)-(5) (Table III).

	code rate 0.66		code rate 0.5	
	240 ms	480 ms	240 ms	480 ms
RLC	ew = 11 dw = 15	ew = 23 dw = 31	ew = 9 dw = 12	ew = 18 dw = 24
Reed-Solomon - DURING	k = 7	k = 14	k = 6	k = 12
Raptor - DURING (G=10)	k = 70	k = 140	k = 60	k = 120

TABLE III. PARAMETERS FOR SIMULATIONS WITH FIXED CODE RATE.

Fig. 7-(a) to (d) show the resulting experienced latency to achieve the 10^{-3} target quality and a 480 ms latency budget for RLC and Reed-Solomon only, for the two code rates. Unsurprisingly, the latency with sliding window codes is always significantly lower. With good channels, the needed latency to achieve the target quality is an order of magnitude (2.0 or 11.0 times) lower with sliding window codes with code rate 0.66, and this gain is even higher with code rate 0.5. This result supports our claim that good receivers will experience a significantly reduced latency, with sliding window codes, or said differently, that FEC protection does not negatively impact their experienced latency.

D. Encoding/Decoding Speed on an Embedded Board

We carried out speed evaluations on a Compulab Eval-AM57x board, featuring a dual-core ARM Cortex-A15@1.5GHz CPU (TI Sitara AM5728) (all codecs leverage on its NEON SIMD facility), 2GB RAM, and running 32-bit Linux Debian (4.4.41 kernel), in order to be relatively close to smartphones' hardware, our target devices. All tests are carried out on a single core, using 100% of it, and re-use the same `eperftool` tool as previously. Transmissions are only simulated (everything takes place in the same process), meaning it has no significant impact. Speed is evaluated by measuring the total encoding (resp. decoding) time. Since these processing times fluctuate a little bit across experiments (a test can last a few seconds during which operating system activities can be triggered), we repeat them 10 times and keep the highest encoding (resp. decoding) speed in order to better estimate the "true" speed. Because the three codecs have been natively

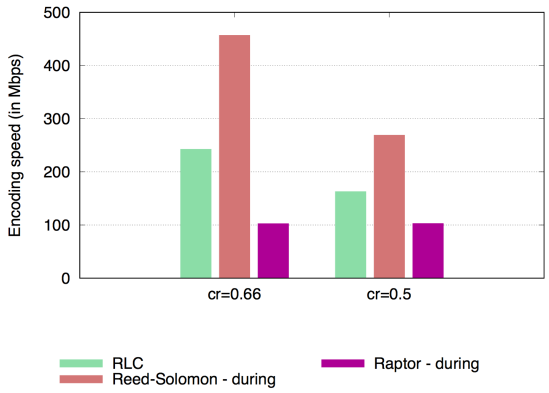


Fig. 8. **Encoding speed** of the various codecs for 480 ms latency budget.

developed for our OpenFEC library and rely on the same low level libraries, our comparaison is fair. It should also be noted that the Raptor codec considered is Expway's highly optimized commercial codec (e.g., decoding relies on Structured Gaussian Elimination) and not Qualcomm's Raptor10 codec. To the best of our knowledge, nobody has developed such a large set of AL-FEC codecs nor published such a comparison, making this comparison unique.

Let us focus on the 480 ms latency budget case, the most realistic configuration in terms of number of supported channels. The encoding speeds are shown in Fig. 8. We see that Raptor features the lowest speed, which is easy to understand: any encoding first requires solving a linear system in order to produce intermediate packets that are in a second step used by LT encoding to produce repair packets. On the opposite, Reed-Solomon and RLC codes feature a much higher encoding speed, with a clear advantage to Reed-Solomon. Indeed, Reed-Solomon is favoured by its reuse of the generator matrix across tests (since the k parameter remains the same, so does the matrix). This matrix creation time, extremely significant, is here negligible. Secondly RLC uses $ew = 23$ or 18 unlike Reed-Solomon where $k = 14$ or 12. If this size increase is highly beneficial in terms of robustness, it also increases the computational complexity.

Decoding speeds are shown in Fig. 9. Across all supported channels, the Raptor decoding speeds are both constant and extremely low (between 114 and 155 Mbps for $cr = 0.66$, and 86 and 134 Mbps for $cr = 0.5$). This speed is close to the encoding speeds, because encoding and decoding are similar with Raptor: first of all a linear system is solved in order to re-build all the intermediate packets, and in a second step an LT encoding recovers the erased source packets. On the opposite, RLC achieves decoding speeds between 807 Mbps and 2.833 Gbps for $cr = 0.66$, and 745 Mbps and 2.044 Gbps for $cr = 0.5$. If Reed-Solomon is significantly faster than RLC for very good channels (1% average loss rate), this is often the opposite at 5% and higher loss rates.

These tests prove that *RLC (and Reed-Solomon) offer an order of magnitude encoding and decoding speed improvement over the existing Raptor codes*. Of course, these results are only meaningful for the use-case considered (e.g., the situation is totally different for file transfers that involve large block sizes, the primary target for Raptor codes).

V. Related Works

Such sliding window AL-FEC codes as Random Linear Network Codes (RLNC) [13] have received a lot of attention for network coding use-cases. Their benefits against block codes have been studied for instance in [14]. Our work differs by the end-to-end nature of our use-case, without in-network re-coding capability, and also by the fact we consider broadcast/multicast communications without any feedback channel.

The recent work from Wunderlich et al. [15] is very close to ours as it compares variants of RLNC codes for real-time flows: a variant based on a block approach and a variant based on either infinite or finite sliding window. If authors claim to have the "first practical generation-less sliding window RLNC scheme", the RLC codes we consider in the current work and the previous ones [7][8] already fall into this category: they are designed from scratch as finite size sliding window codes. The main differences with respect to this very interesting work come from the methodology followed: we consider the mobility scenarios specified by the 3GPP SA4 working group to compare AL-FEC performance, we consider a unprecedented set of codes, including Raptor codes, and perform performance evaluation in a non traditional but highly meaningful manner: what is the required AL-FEC protection required to achieve the target residual-loss quality? This is in our opinion the key method to compare the various codes. Finally [15] considers sub-optimal configurations, ignoring the highly effective "Decoding Beyond Maximum Latency" optimization altogether [16][7] unlike our work.

VI. Conclusions and Discussions

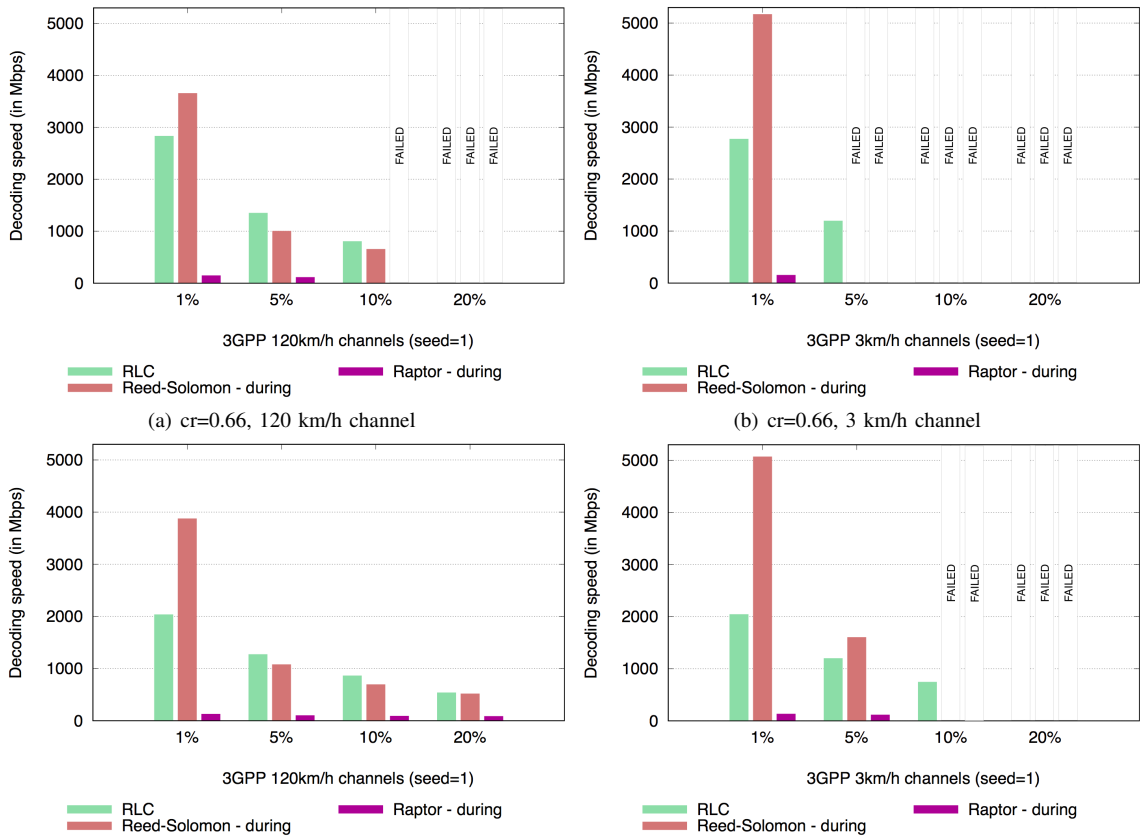
This work demonstrates that AL-FEC sliding window codes, in particular RLC codes, outperform all block codes when dealing with real-time flows and CBR communications, both in terms of reduced latency and perhaps more importantly improved robustness against various types of mobility scenarios. The tests also show that our RLC codec outperforms Raptor codes in terms of encoding and decoding speeds, in addition to the above benefits.

This work does not consider the diversity of sliding window codes. It focusses on "basic" RLC on GF(2^8) without trying to experiment with sparse variants, with other finite fields, nor with structured codes [17][18]. However, if these alternatives are most probably valuable for high bitrate flows and large encoding/decoding windows, this is less significant for the present work where ew_size is at most equal to 33.

If these results motivate our proposal of extending FECFRAME to sliding window codes [10][8], we also believe these codes could be benefit to other transport protocols where latency does matter. For instance, they could favourably replace trivial unidimensional or bidimensional (A.K.A. interleaved) XOR codes considered in such protocols as QUIC [19] or RTP [20]. Such trivial XOR codes exhibit bad loss recovery performance compared to their transmission overheads: simplicity is important but it should not compromise efficiency. Using RLC instead will be the subject of future works.

References

[1] T. Paila, R. Walsh, M. Luby, V. Roca, and R. Lehtonen, "FLUTE - File Delivery over Unidirectional Transport," Nov. 2012, IETF Request for Comments, RFC 6726.



(a) cr=0.66, 120 km/h channel

(b) cr=0.66, 3 km/h channel

(c) cr=0.5, 120 km/h channel

(d) cr=0.5, 3 km/h channel

Fig. 9. **Decoding speed** of the various codecs for code rates 0.66 (top) and 0.5 (bottom), and 480 ms latency budget.

[2] M. Watson, A. Begen, and V. Roca, "Forward error correction (fec) framework," Jun. 2011, IETF Request for Comments, RFC 6363.

[3] "3gpp; technical specification group services and system aspects; multimedia broadcast/multicast service (mbms); protocols and codecs (release 13)," Mar. 2016, 3GPP TR 26.346 version 13.4.0 Release 13.

[4] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder, "RaptorQ Forward Error Correction Scheme for Object Delivery," IETF Request for Comments, RFC 6330 (Standards Track), Aug. 2011.

[5] J. Lacan, V. Roca, J. Peltotalo, and S. Peltotalo, "Reed-solomon forward error correction (fec) schemes," Apr. 2009, IETF Request for Comments, RFC 5510.

[6] V. Roca, C. Neumann, and D. Furodet, "Low density parity check (ldpc) staircase and triangle forward error correction (fec) schemes," Jun. 2008, IETF Request for Comments, RFC 5170.

[7] V. Roca, B. Teibi, C. Burdinat, T. Tran, and C. Thienot, "Block or Convolutional AL-FEC Codes? A Performance Comparison for Robust Low-Latency Communications," Nov. 2016, unpublished working document, HAL open-archive, hal-01395937.

[8] V. Roca, "Sliding window random linear code (rlc) forward erasure correction (fec) scheme for fecframe," Jul. 2017, IETF TSVWG work in progress, draft-ietf-tsvwg-rlc-fec-scheme-00.

[9] V. Roca, M. Cunche, C. Thienot, J. Detchart, and J. Lacan, "RS + LDPC-Staircase codes for the erasure channel: Standards, usage and performance," in *9th IEEE Conf. on Wireless and Mobile Computing, Networking and Communications (WiMob 2013)*, Aug. 2013.

[10] V. Roca and A. Begen, "Forward error correction (fec) framework extension to sliding window codes," Jul. 2017, IETF TSVWG work in progress, draft-ietf-tsvwg-fecframe-ext-00.

[11] ETSI, "Evaluation of mbms fec enhancements (final report)," Dec. 2015, 3GPP TR 26.947 version 13.0.0 Release 13.

[12] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer, "Raptor Forward Error Correction Scheme for Object Delivery," Oct. 2007, IETF Request for Comments, RFC 5053 (Standards Track).

[13] F. Fitzek, M. Pedersen, J. Heide, and M. Medard, "Network coding: Applications and implementations on mobile devices," in *5th ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks*, Oct. 2010.

[14] M. Toemoeskoezi, F. Fitzek, D. Lucani, M. Pedersen, and P. Seeling, "On the delay characteristics for point-to-point links using random linear network coding with on-the-fly coding capabilities," in *20th European Wireless Conference*, May 2014.

[15] S. Wunderlich, F. Gabriel, S. Pandi, and F. Fitzek, "We dont need no generation - a practical approach to sliding window rlnc," in *IEEE 2017 Wireless Days*, Mar. 2017.

[16] P.-U. Tournoux, T. Tran-Thai, E. Lochin, and J. Lacan, "When on-the-fly erasure code makes late video decoding happen," in *25th ACM Workshop on Network and OS Support for Digital Audio and Video (NOSSDAV'15)*, Mar. 2015.

[17] K. Matsuzono, V. Roca, and H. Asaeda, "Structured Random Linear Codes (SRLC): Bridging the Gap between Block and Convolutional Codes," in *IEEE Global Communications Conference (GLOBECOM'14)*, Dec. 2014.

[18] D. E. Lucani, M. V. Pedersen, D. Ruano, C. W. Sorensen, F. Fitzek, J. Heide, and O. Geil, "Fulcrum network codes: A code for fluid allocation of complexity," Nov. 2015, arxiv 1404.6620v2.

[19] I. Swett, "QUIC FEC v1," Feb. 2016, unpublished working paper.

[20] M. Zanaty, V. Singh, A. C. Begen, and G. Mandyam, "RTP Payload Format for Flexible Forward Error Correction (FEC)," Oct. 2016, IETF PAYLOAD work in progress, draft-ietf-payload-flexible-fec-scheme-03.